

Technical Library

School of Computing and Information Systems

---

2013

## Xamarin Mobile Development

Jared Dickson  
*Grand Valley State University*

Follow this and additional works at: <https://scholarworks.gvsu.edu/cistechlib>

---

### ScholarWorks Citation

Dickson, Jared, "Xamarin Mobile Development" (2013). *Technical Library*. 167.  
<https://scholarworks.gvsu.edu/cistechlib/167>

---

This Project is brought to you for free and open access by the School of Computing and Information Systems at ScholarWorks@GVSU. It has been accepted for inclusion in Technical Library by an authorized administrator of ScholarWorks@GVSU. For more information, please contact [scholarworks@gvsu.edu](mailto:scholarworks@gvsu.edu).

# Xamarin Mobile Development

By

Jared Dickson

December, 2013

# Xamarin Mobile Development

By

Jared Dickson

A project submitted in partial fulfillment of the requirements for the degree of  
Master of Science in  
Computer Information Systems

at

Grand Valley State University  
December, 2013

---

**Dr. Robert Adams**

**December, 2013**

## Table of Contents

<b>Abstract.....</b>	<b>4</b>
<b>1. Introduction.....</b>	<b>4</b>
<b>2. Cross-Platform Mobile Development Approaches.....</b>	<b>5</b>
<b>2.1 Native.....</b>	<b>5</b>
<b>2.2 Mobile Web.....</b>	<b>6</b>
<b>2.3 Hybrid.....</b>	<b>6</b>
<b>2.4 Cross-Compiled.....</b>	<b>7</b>
<b>3. Xamarin.....</b>	<b>8</b>
<b>3.1 Hardware Features.....</b>	<b>8</b>
<b>3.2 Code Sharing.....</b>	<b>8</b>
<b>3.3 Native Bindings.....</b>	<b>8</b>
<b>3.4 Performance.....</b>	<b>9</b>
<b>3.4.1 iOS.....</b>	<b>9</b>
<b>3.4.2 Android.....</b>	<b>9</b>
<b>4. Experiments.....</b>	<b>10</b>
<b>4.1 Geolocation.....</b>	<b>10</b>
<b>4.2 Bluetooth.....</b>	<b>11</b>
<b>4.3 Shared Libraries.....</b>	<b>12</b>
<b>4.4 Native Bindings.....</b>	<b>13</b>
<b>5. Analysis and Conclusions.....</b>	<b>14</b>
<b>5.1 Technical Issues.....</b>	<b>14</b>
<b>5.2 Business Value.....</b>	<b>15</b>
<b>5.3 Summary.....</b>	<b>15</b>
<b>Bibliography.....</b>	<b>16</b>

# Abstract

Mobile application development is a diverse environment and requires specific languages and tools. This makes development across multiple platforms time and cost intensive. Many solutions have been presented to address the issues of cost and diversity, though none come without their own set of compromises. This project studied Xamarin as a mobile platform development tool allowing for cross-platform development that utilizes a single programming language, native UI elements, code sharing for reuse, and near-native performance. Specifically, we examined Xamarin's support for geolocation, bluetooth, shared libraries and native bindings. Analysis showed that Xamarin is not an answer for all problems, but a solid solution for the appropriate use cases.

## 1. Introduction

Mobile interactions happen on a variety of different platforms and devices utilizing their own operating systems, user-interface designs, app stores, hardware feature sets, device form factors (size), permission implementations, and software development toolkits. This diversity is only increased by the constantly expanding pool of hardware components and overall device design improvements.

One of the problems in mobile development is that of end-user isolation. The choice of platform, required features and operating system backwards compatibility segments users into groups which often limit the growth of the application user-base. To combat this there are a limited number of options available. The first solution was to rewrite the entire mobile application independently for each platform. This of course leads to high development and maintenance costs for these applications.

Several ideas, products and frameworks have presented themselves in an attempt to simplify and centralize development in the mobile environment. Mobile websites allow for web content to be created in a way that is specifically designed for smaller devices. The native approach is often still used by those desiring a native look and feel, or when high performance is required. Hybrid frameworks have also been designed to allow developers to package websites into an application while offering interaction with a specific hardware device feature-set, such as GPS or camera functionality.

The solution being presented in this paper is called Xamarin. Unlike other options, Xamarin offers developers a way to create cross-platform applications without sacrificing native UI elements or performance, while allowing code reuse between platforms. This is done through unifying development to a single programming language, C#. It utilizes the Mono runtime which has been developed to bring the C# language to Linux and Macintosh. This creates a single environment for developers to write mobile applications for multiple platforms.

## 2. Cross-Platform Mobile Development Approaches

Mobile development approaches include items such as native, mobile web, hybrid and cross-compiled.

### 2.1 Native

The process of porting mobile applications across a variety of platforms is a time-intensive task. Each of the three main platforms uses a different programming language. Apple devices running iOS are written using Objective-C as the primary language. Android devices use Java as the default programming language. Windows Phone uses C# as its main language. Having developers learn a new language for a specific platform can be very time consuming. The alternative, of bringing on new developers already fluent in the other platforms, can be costly to a business. When a decision is made to add a new feature to the existing application there will be a time delay between launching the new feature on the different platforms.

Mobile applications that are CPU intensive are often directed towards a native implementation as it will give you the best performance over the present options. One of the more obvious candidates for a native implementation are games. Intense graphics processing required by games is not suitable for solutions like JavaScript. Complex data object models which change frequently should also be implemented in a native solution. Graphs and data representation visuals can sometimes be accomplished using JavaScript, but as the number of data points grows the performance will begin to choke. If a solution is needed that will render a thousand or more data points, one should use a native build.

Each of the major platforms provides a guide for best practice design principles. Android recommends placing your menu on the bottom of the screen. Whereas iOS positions user menus on the top of the screen. These implementation patterns are beneficial in creating apps that look and feel comfortable to the end user. It assists them in feeling as if the application is an extension of their device instead of a foreign object. This can improve user retention rates over competing applications.

Staying native in mobile applications also provides for a better touch interaction experience. Gestures, pinches and swipes work exactly as they were intended to. Users don't get as frustrated interacting with the content because the object responses are intuitive, or at least consistent across the device. Not having an intermediate layer keeps performance up by preventing touch lag.

Studies have been done that give evidence to the idea that going with a native UI helps to increase the retention rate of users in mobile applications. In part, this is because the user already feels familiar with the components of a native

application. The user isn't left confused in how to interact with UI objects. Non-native platforms can sometimes leave the user guessing how different objects will react to their touches and gestures. Of course, this can be mitigated with a well designed UI.

## 2.2 Mobile Web

The first push of cross-platform mobile development has been through the use of mobile web. All modern smartphones come with a capable web browser. With the development of the responsive web through HTML5, CSS3 and JavaScript mobile phones have gained powerful tools of interaction. It provides a great platform in terms of flexibility, but fails to meet expectations when it comes to the always-on mentality of mobile.

Developers are able to push immediate updates to users through mobile web. Once the code is up on the web server it becomes accessible to users. Web developers are also able to develop quickly using skills practiced daily on normal websites. It takes advantage of an existing infrastructure and comes with an existing level of familiarity.

There are practical performance limitations that prevent mobile websites from being commonly used for interactions with dynamic information. However, these same limitations are more resilient with static data. Service interruption and poor signal can prevent mobile websites from being accessible to users when they need information. If a network connection isn't present, then the mobile websites will not be available. Unless data is cached, a user will be required to reload all data each time they visit the website. Load times are high because of cell phone network latency. Responsiveness of a mobile website can also be limited by the slow performance of JavaScript on mobile devices. It is normal to expect to see speeds to be lowered by a factor of 10 when JavaScript is moved from desktop to mobile. In the event that the web server goes offline for maintenance or malfunction users will be unable to access the information.

Developers are also restricted to the level of hardware interaction that is supported by HTML5 and the browser. Depending on your application, this could be a deciding factor. Features such as camera usage and the accelerometer aren't browser capable. Geolocation is supported by some mobile browsers and can be used by mobile web.

## 2.3 Hybrid Web

The idea of hybrid applications was born through crossing mobile web applications with native application packaging. It takes advantage of the existing skills of web developers allowing them to write applications using HTML5, CSS3 and JavaScript. In addition, developers are able to access native hardware and features through API

calls define by the wrapper. One of the most prominent of these frameworks used today is PhoneGap. It allows interaction with GPS, camera, compass, accelerometer and more.

This creates a great opportunity for the mobile development community as a whole. It brings new possibilities to light that were not available previously. A web developer can become a mobile application developer without needing to learn all of the intricate details of the various mobile platforms.

Since hybrid applications are rendered through the browsers, it isn't possible to use truly native UI components. A developer can use CSS3 techniques to style different UI components to effectively look identical to native UI components, but true duplication isn't possible. This is because you are not able to recreate the responsiveness of native UI elements. It is common that due to this lack of native UI support many developers have decided to create their own styles for their apps. This unique styling causes the mobile application to stand out, and can enable a creative branding which can assist in promoting the applications popularity in some circles.

Another compromise of any hybrid application is the delay that exists between when a new feature comes out to a mobile device, and when it is re-implemented for hybrid. Depending on the popularity of the feature, it could take a long time before it becomes accessible. This delay is not something you will find in Xamarin. However, many such platforms allow you to create your own plugins, enabling you to implement the bridge to the new feature on your own time for each of your target platforms.

## 2.4 Cross-Compiled

The method of developing cross-platform mobile applications in a single compiled programming language is called cross-compiling. It allows a developer to write an application in one language, and compile it independently for each targeted platform. This unification is generally helpful in reducing the amount of development time required to build mobile applications, as much of the code will be similar. Depending on the platform, native UI elements may or may not be accessible. Performance is better than hybrid applications, but not always as good as native implementations. A performance if often taken because there are many required libraries which need to be loaded into memory by the framework that is being used. Many such solutions also require a monthly developer fee as they are still being developed and improved.

Xamarin falls into this category of mobile development. All the code is written in C# and deployed to multiple platforms. It is based on the Mono runtime which was initially release in 2004, and is still in active development.

### 3. Xamarin

Xamarin is a cross-platform mobile application development solution that creates a unified environment for developers. Instead of writing code in Objective-C for iOS and Java for Android, it enables developers to write code in the C# language. Each application implementation is developed independently through a platform specific project in order to preserve native user interfaces. Within each application is direct access to the native APIs, allowing developers access to all the native hardware features. Between these projects you can re-use core code common to each application by using portable class libraries (PCL's) or other methods. This would include items such as data models, business logics, cloud integrations and database access. If a project requires use of a library written natively, a developer can create a native binding which allows access to it through C# calls. The performance you receive in Xamarin applications is quite close to native speed, and in certain situations can perform better. On average, Xamarin mobile applications can reuse 75% of their code between projects. Once development is complete, each project can be compiled and deployed to the appropriate app store.



#### 3.1 Hardware Features

The nature of the development of Xamarin lends itself to not impose any limits on the availability of the device hardware to mobile application developers. Xamarin generates a native binding directly to the iOS and Android SDK's. This provides a direct and identically matching link between the Xamarin libraries and the native APIs. All native objects are accessible using the same names, with the same properties and attributes, while having the same method calls. There is no re-implementation of code in Xamarin. All native calls are run through native APIs. GPS, accelerometer, camera, compass and all of the rest are accessible. The process allows Xamarin to release same-day update releases. One of the most published updates was that of same-day support for iOS 7.

#### 3.2 Code Sharing

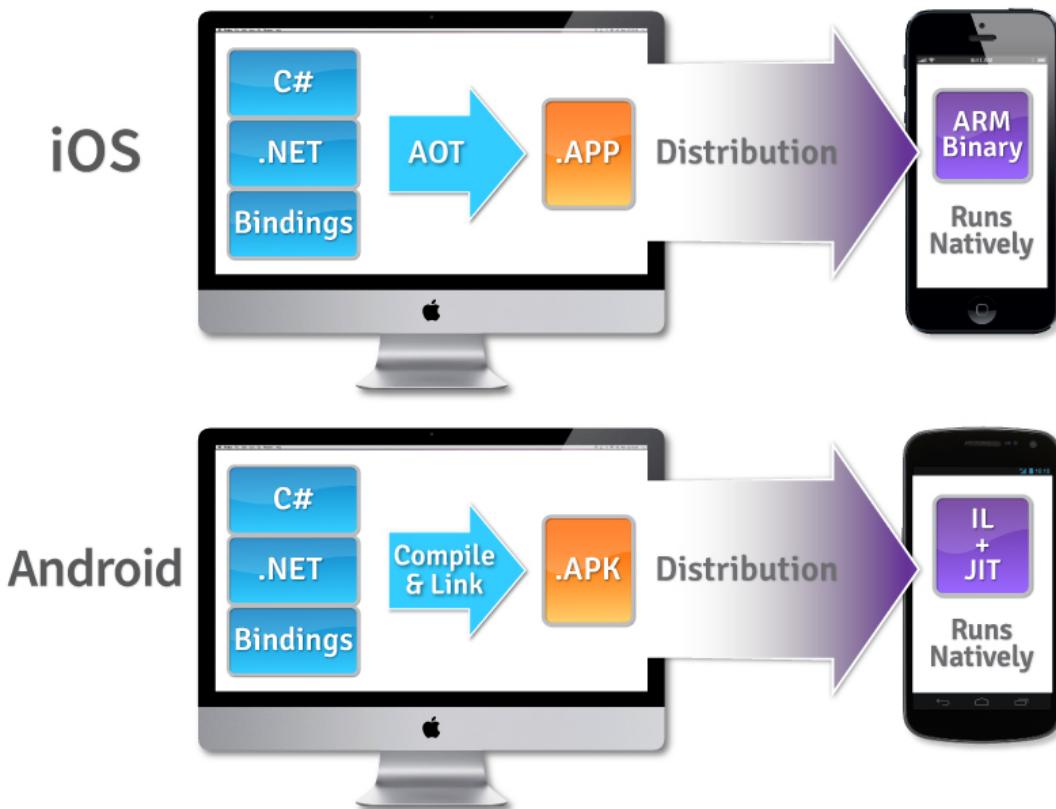
Portable Class Libraries, or PCLs, allow developers to write compiled libraries for sharing with multiple projects. Official support for PCL is one of the newest releases from Xamarin. Previous options available included: file linking, cloned project files, and Microsoft Project Linker. All of these methods provided a means of sharing class structures between projects. Depending on the code sharing route, extra steps may have to be taken to be setup correctly because much of it was a manual process. Writing a single library for a particular core project is the simplest avenue available for code sharing, and requires the least amount of maintenance..

### 3.3 Native Bindings

Native bindings are a mechanism available to iOS and Android projects that allow you access native libraries from within your Xamarin mobile application. This is done by creating a binding, or a mapping, of the calls from the C# library to the native library. It is a semi-automated process. Xamarin is able to create many of the connections, but does leave some loose ends for the developer to tie up.

One of the benefits of using native bindings is that it allows you to incorporate externally written libraries. This can save developer time in preventing the re-writing of important code. For example, optical character recognition (OCR) could be a requirement of a mobile application. Instead of writing all of the OCR code a developer could include, through native bindings, open source library packages to add extra functionality.

### 3.4 Performance



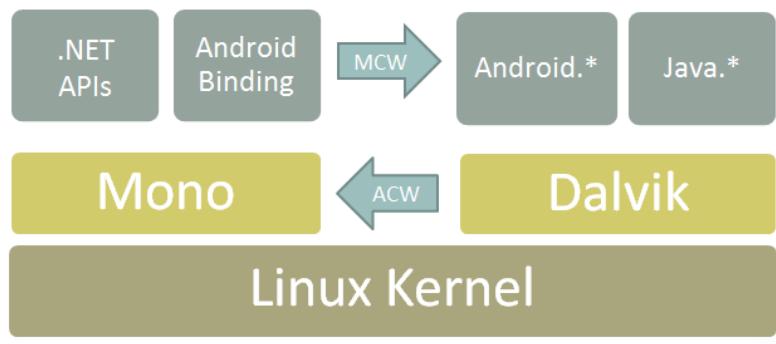
#### 3.4.1 iOS

C# code for iOS in Xamarin compiled to a native ARM binary. This allows Xamarin iOS mobile applications performance to be on par with their Objective-C based counterparts. The process involves compiling the application using the same LLVM Engine used by XCode.

### 3.4.2 Android

Android compilation with Xamarin produces interpreted language (IL) code. The IL code runs in the Mono runtime which is bundled installed with the mobile application. The Mono runtime runs directly on top of the Linux kernel, and not through the Dalvik virtual machine. Any code that calls to the native SDK are run through the Dalvik virtual machine through the Managed Callable Wrapper (MCW) and Android Callable Wrapper (ACW) communication channels. MCW is used to initiate a call to the native SDK passing on the necessary parameters. The ACW is used by the Dalvik VM to return results back to the Xamarin application. The overhead involved in this communication is negligible.

The result of this design achieves performance that, in certain cases, surpasses that of the Dalvik VM with faster computations. Calculating these performance metrics can be difficult because you cannot run a Mono test that interacts with the Dalvik VM and gain true results. To get meaningful data you need to plan your tests carefully. Xamarin did this through the XobotOS project (available on GitHub). The premise of the project was to gain a true insight into the performance and memory footprint benefits of C#. XobotOS is a semi-automated port of Android 4.0 from Java to C#. Once the port was complete several test cases that were created to test Xobot. They focused on common data-structures such as Binary Search Trees, Hash Tables and threading. The results concluded with C# leading in all of the reported cases. Up to an 800% performance increase was seen.



## 4. Experiments

During my time working with Xamarin I focused much of my effort into studying hardware feature integrations. I want to see how much work would be involved in creating a cross-platform application that was hardware feature intensive. There were many struggles than I ran into, but many of them were more towards the development environment setup than anything else. Other troubles were based on my misconception of this idea “write once, deploy everywhere”. Xamarin does assist in making this more feasible, but the native UI components and SDK elements really hinge on having a separate project per platform. As I worked I began to appreciate how this sort of architecture was necessary in order to allow Xamarin to keep up with fast paced mobile operating system updates.

## 4.1 Geolocation

Initially I was under the impression that I could write one implementation of geolocation and share it between all platforms. This idea was completely incorrect. The difficulty is that through Xamarin you only get what is already available in the platform SDK. There is no supplied hardware feature abstraction ready to go. This meant that I needed to write geolocation for each platform, and in a different way for each platform.

Once I made it over that hurdle GPS integration became quite simple. I first implemented it for Android. The code was quite simple. However, one of the things I learned was that many of these hardware level features are intricately tied into the view lifecycle. It caused me to understand the difficulty of writing a GPS abstraction that could integrate with multiple platforms well. It came down to a combination of complementing the UI actions of when to introduce new values, and the battery life associated with frequency and level of details requested of the GPS location.

After the completion of testing GPS locations I learned of the Xamarin Component Store. The purpose of this store was to host libraries built by other Xamarin users. Some were free, while others required a separate licensing fee. Within the list of components was one called Xamarin Mobile. Its purpose was to provide an abstraction layer for GPS, contacts and media. To continue my exploration, I downloaded the component and loaded it into each of my projects. It worked seamlessly for my simple requirements. The code was written to allow copy and paste between any of the supported platforms. It was optimized for a passive polling of geolocation coordinates to minimize battery drain. For any future projects I would consider it further, but presently it is still a beta software in preview mode.

## 4.2 Bluetooth

As I began to look into developing features which would utilize bluetooth I found it easy to access on Android, but had considerable difficulty when looking into development for iOS. On Android I developed a simple chat messaging application which was able to communicate between devices easily. What I determined is that prior to iOS 7 there was no easy developer support for bluetooth. Documentation for bluetooth on iOS simply didn't exist on the Xamarin support website. This caused me to take a broader approach to looking for a solution. I never completed bluetooth chat on iOS, and I expect it wasn't possible. This was before the release of iOS 7, and now I believe it could be done with the release of Core Bluetooth.

The most important thing I learned from my difficulty with bluetooth is that I don't need to only look to Xamarin documentation. It was a realization of something I knew before, but hadn't considered. Since I'm developing native applications against the native SDKs, I can use native code examples. Instead of only look to Xamarin documentation for help with iOS bluetooth connections I could look on StackOverflow for native iOS examples. All of the object and method calls were the same. I just needed to adjust for simple syntax changes. Xamarin applications aren't starting

from scratch, as much as they are building on the applications before them.

### 4.3 Shared Libraries

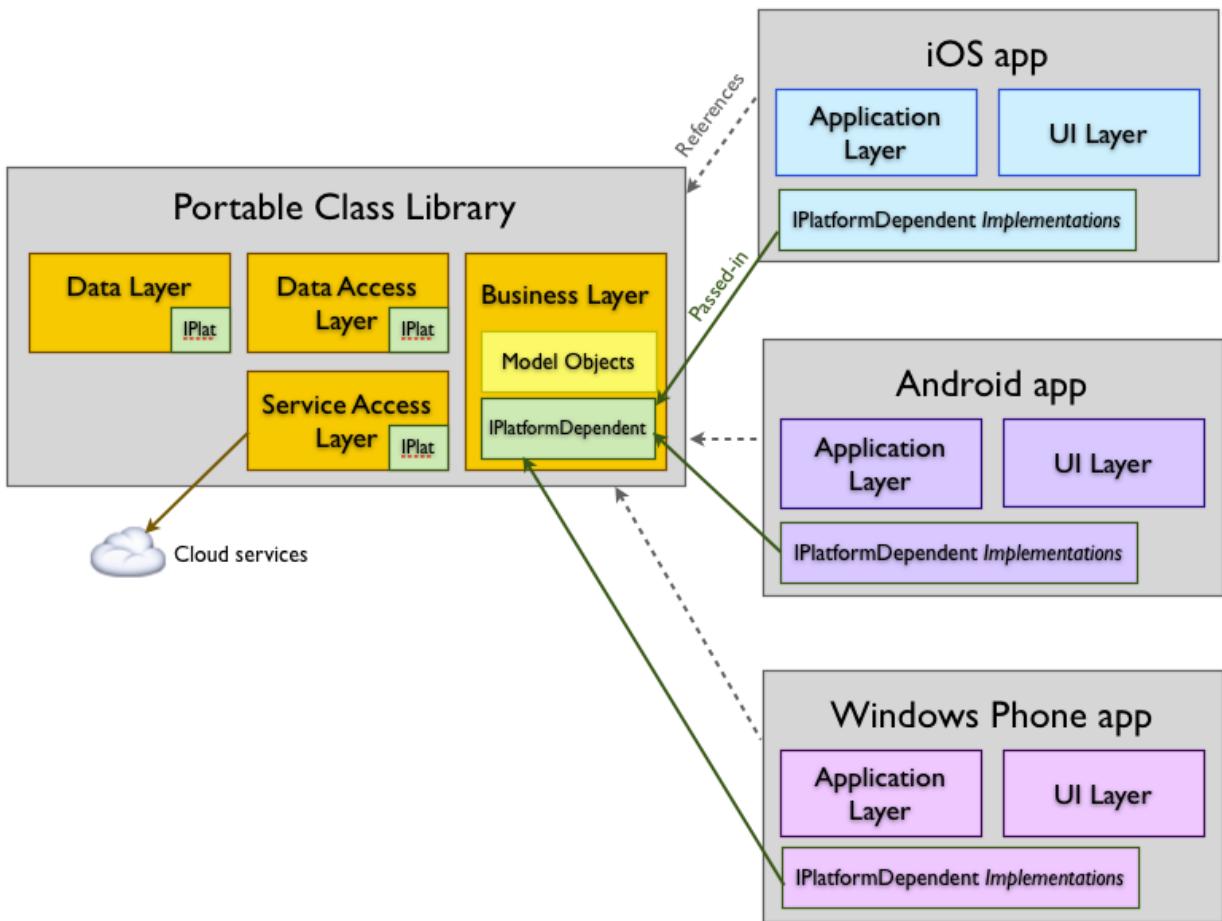
When I first began my study, Xamarin hadn't yet release official support for PCLs. File linking, cloned project files, and Microsoft Project Linker were the only supported options available. I didn't see the value in these options as long term solutions. They required overhead in a more hands on approach with keeping your core files up to date. Instead I spent my time looking into unofficial support for shared libraries.



I was not alone in desiring a better solution. I found my blog posts on other developers attempting the same thing I was. Other users were trying to create shared class libraries like those supported within Visual Studio. I found a blog on which a developer had posted his solution. It required adding two new XML files into my Visual Studio install directory. After I placed them there I was able to create a new PCL library targeted to Windows Phone 8, Android

and iOS. The PCL is a subset of .NET and does not include everything. It does provide enough core libraries to be useable for most projects. At the very least, it will still work to serve data classes.

Since my initial work Xamarin has release official support for portable class libraries. This has removed an enormous wall for developers. It helps to create a more seamless development environment. I was able to create a portable class library for use in Android, iOS, Windows Phone 8 and an ASP.NET MVC5 web application. The code worked almost perfectly. For the web application I needed to add a missing reference before the library could be used, but after adding the reference it worked very well. PCLs will work very well for applications also targeting desktop and web.



## 4.4 Native Bindings

Being able to use native bindings was important to me because I wanted to be able to reuse existing libraries without having to reimplement a bunch of code. The automated portion of Xamarin creating bindings was very helpful on the areas it could easily translate. Manually creating code bindings wasn't as easy. The compiler would throw errors telling you where to look for the error and tell you what it really expected, which was helpful. It becomes quite a bit of back and forth if you need to fix many connections. The process is simple enough, just as debugging normal code, but very tiresome when you expect the product to integrate without issue. In the end, you are able to map everything correctly and most of it is still done for you. Because of this I still find it to be very worthwhile to consider and use native bindings when compatible C# libraries aren't available. If you were to begin porting a native application to Xamarin I believe looking into using native bindings for the initial release would be helpful. In the future I expect to see this process greatly improved by Xamarin's development team. This is supported by the efficiency of Xamarin automated native SDK binding which provides same-day platform update releases.

## 5. Analysis and Conclusions

### 5.1 Technical Issues

The technical issues surrounding Xamarin mostly revolve around having a proper development setup for the platforms you will be targeting. Both in terms of the software you will be using to write code, and the hardware for building the applications. The other struggle developers might face is learning the development and design patterns of each mobile platform. This includes file structure as well as view lifecycles.

Each platform has specific needs before you can begin development. To develop for iOS you must have a Mac with OS X available. It isn't required for writing code, but is required for compiling, debugging and deploying. It is possible to share a Mac Mini between developers by deploying and building over the network. To develop Windows Phone applications you need to build using a Windows operating system. Android applications can be developed on both Mac and Windows computers. Xamarin Studio can be used to develop PCLs, iOS applications and Android applications on a Mac, but only Android and PCLs on Windows computers. Visual Studio is required to build Windows Phone applications. Visual Studio can be used to build iOS and Android applications, but iOS builds must be run on a Mac over the network. Once you have your development environment settled the rest is easy.

	Mac OS X	Windows	
IDE			+ <small>Xamarin Business Edition</small>
iOS			+ <small>Mac</small>
Android			
Mac			
Windows/Phone			

## **5.2 Business Value**

Xamarin has positioned itself to be a great asset to software companies already using Microsoft based technologies. Employees are already familiar with .NET technology and development software. There isn't a need to hire or contract out work when launching to a new platform. You can also save costly development time through the various code reducing avenues. It also integrates quickly and easily into your existing Microsoft infrastructure.

One of a company's best resources are it's employees. By tapping into in-house skills you can expand to mobile while protecting company culture. Many employees are already familiar with C# as it is often the preferred language for ASP.NET MVC websites. It is also possible to save valuable time and money by not having to hire new employees or contract out mobile work. These savings can assist in helping a business to grow at a managed rate while allowing developers to improve their skills in new and challenging ways..

Xamarin provides multiple ways to benefit a business financially through means of reducing development time. The main advantage is that up to 90% of code can be re-used across platforms using PCLs. Another means of reducing development time is fewer lines of code due to cleanliness of C# as a language, over the verbosity of Objective-C. Microsoft .NET developers won't need to take time to learn a new development environment since Visual Studio is a supported IDE. The reduced development time can be used to lower the final costs of an application, or to allow the teams to focus more on additional features.

Microsoft has published an Azure Mobile Services component available freely in the Xamarin component store. It enables developers to program using code-first methodologies to quickly save data to your Azure SQL database. Xamarin mobile applications integrate with existing Microsoft SQL Server deployments with Microsoft libraries. Communication with ASP.NET websites is also available using WebAPI and REST technologies.

## **5.3 Summary**

Through this study I believe that Xamarin has proved itself to be a valuable candidate for cross-platform mobile applications. It certainly isn't perfect for every use case as it may be overkill. In other ways it falls short if you or your company don't already write programs for the Microsoft platform. Because it is a young solution and has the support of Microsoft, I believe it will only continue to improve and grow as a development tool and platform.

Xamarin presents itself to be a good option for those who meet a few basic criteria. First, you should be targeting multiple platforms. If not, you may be better off going with a native solution. The next is that native user experience should be a priority. If native UI doesn't matter then it may be a case for a hybrid solution. When the previous criteria are met and you are also looking for great performance, Xamarin is probably the solution for you.

## Bibliography

Tomi Heimonen. 2009. Information needs and practices of active mobile Internet users. In Proceedings of the 6th International Conference on Mobile Technology, Application & Systems (Mobility '09). ACM, New York, NY, USA, , Article 50 , 8 pages. DOI=10.1145/1710035.1710085 <http://doi.acm.org/10.1145/1710035.1710085>

Chad Tossell, Philip Kortum, Ahmad Rahmati, Clayton Shepard, and Lin Zhong. 2012. Characterizing web use on smartphones. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12). ACM, New York, NY, USA, 2769-2778. DOI=10.1145/2207676.2208676 <http://doi.acm.org/10.1145/2207676.2208676>

Caspar Ryan and Atish Gonsalves. 2005. The effect of context and application type on mobile usability: an empirical study. In Proceedings of the Twenty-eighth Australasian conference on Computer Science - Volume 38 (ACSC '05), Vladimir Estivill-Castro (Ed.), Vol. 38. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 115-124.

Ribeiro, A.; da Silva, A.R., "Survey on Cross-Platforms and Languages for Mobile Apps," *Quality of Information and Communications Technology (QUATIC), 2012 Eighth International Conference on the* , vol., no., pp.255,260, 3-6 Sept. 2012. doi: 10.1109/QUATIC.2012.56

Ohr, J.; Turau, V., "Cross-Platform Development Tools for Smartphone Applications," Computer , vol.45, no.9, pp.72,79, Sept. 2012. doi: 10.1109/MC.2012.121

Raj, R.; Tolety, S.B., "A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach," India Conference (INDICON), 2012 Annual IEEE , vol., no., pp.625,629, 7-9 Dec. 2012. doi: 10.1109/INDCON.2012.6420693

Palmieri, M.; Singh, I.; Cicchetti, A., "Comparison of cross-platform mobile development tools," Intelligence in Next Generation Networks (ICIN), 2012 16th International Conference on , vol., no., pp.179,186, 8-11 Oct. 2012. doi: 10.1109/ICIN.2012.6376023

Spyros Xanthopoulos and Stelios Xinogalos. 2013. A comparative analysis of cross-platform development approaches for mobile applications. In Proceedings of the 6th Balkan Conference in Informatics (BCI '13). ACM, New York, NY, USA, 213-220. DOI=10.1145/2490257.2490292 <http://doi.acm.org/10.1145/2490257.2490292>

Dalmasso, I.; Datta, S.K.; Bonnet, C.; Nikaein, N., "Survey, comparison and evaluation of cross platform mobile application development tools," Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International , vol., no., pp.323,328, 1-5 July 2013. doi: 10.1109/IWCMC.2013.6583580

Drew Crawford. 2013. Why Mobile Web Apps Are Slow. (July 2013). Retrieved November 12, 2013 from  
<http://sealedabstract.com/rants/why-mobile-web-apps-are-slow/>

Xamarin. 2013. Android Advanced Topics. (November 2013). Retrieved November 12, 2013 from  
[http://docs.xamarin.com/guides/android/advanced\\_topics/](http://docs.xamarin.com/guides/android/advanced_topics/)

Xamarin. 2013. iOS Advanced Topics. (November 2013). Retrieved November 12, 2013 from  
[http://docs.xamarin.com/guides/ios/advanced\\_topics/](http://docs.xamarin.com/guides/ios/advanced_topics/)

Miguel de Icaza. 2012. Android Ported to C#. (May 2012). Retrieved November 12, 2013 from  
<http://blog.xamarin.com/android-in-c-sharp/>