Spring 3-2-2018

Xamarin Forms vs Native Platform Development

Austin Borop
Olivet Nazarene University, amborop@olivet.edu

Xamarin Forms vs Native Platform Development

Austin Borop

Olivet Nazarene University

Abstract

This research compared the development process of different technologies. These technologies were native iOS, native Android development, and Xamarin. Each of these technologies were testing using their recommended development tools. Each technology was used to create calculator application to meet a minimum specification. These specifications were that the calculator would have basic arithmetic functionality. The application needed to do all of these things through button presses and be able to display the output as well. The data analysis was time based and the duration necessary to complete each application. The research concluded with some interesting results. The expected results of native application development being superior to Xamarin forms wasn't as straightforward as originally thought. The development process yielded positive results for Xamarin in regards to time to develop who applications for multiple platforms. The result was a functional application for each platform. The Android and iOS development also yielded promising applications but not in the same timeframe as Xamarin. Xamarin ultimately reigned superior over the other methods.

Keywords: Xamarin, iOS, Android, development, platform, application

Introduction

Xamarin Forms as defined by their website is "…a cross-platform UI toolkit that allows developers to easily create native user interface layouts that can be shared across Android, iOS, and Windows Phone" (Xamarin 2018). Xamarin Forms achieves this cross-platform utility using their derivation of XML, called XAML. This research is intended to analyze the development process of a Xamarin Forms application compared to the traditional development process. The platforms that will be developed for are iOS version 11.2.6 and Android version 8.1. The exclusion of the Windows Platform of Windows 10 was intentionally disregarded for this research due to its low percentage of market share in the mobile platform market. Instead, this research will focus on those popular platforms as they are the most commonly targeted by developers for mobile applications. These platforms also have the property of their native development languages being completely different from the Xamarin Forms languages of XAML and C#. The Windows Phone does not share this commonality of the Android and iOS platforms. Some other technologies being demonstrated in this research are the Windows 10 Education platform and MacOS High Sierra. These platforms are key to this research because they run the development environments used for both the native and Xamarin Forms development suites. Windows 10 Education runs Visual Studio Community for Xamarin Forms development as well as Android Studio for native Android Development. However, due to Apple's Terms of Service, all compilation for the native and Xamarin Forms development for iOS devices was performed on MacOS High Sierra to comply with Apple. Any other technology used for this research will be introduced in the appropriate section to which it applies.

Since the technology has been introduced, it is time to supply cause for this research. This research into native vs Xamarin Forms platform development came out of the need for the best tool to quickly deploy a single app to multiple platforms. There are multiple technologies used to achieve this, but Xamarin was chosen for a few reasons. The use of the C# language showed that the technology was using a common, object-oriented language to develop the applications. The choice of XAML was not welcomed initially but since it proved to be a true derivation of the commonly used XML technology it wasn't difficult to adopt. This idea of programming the application once and having it run on multiple platforms natively encourages many programmers to investigate technologies that make these promises. Microsoft showed clear interest in Xamarin when they decided to purchase the company. Since Xamarin chose a Microsoft technology to be the base of their product, Microsoft did not have to do much to welcome the new company into the Microsoft family. All this support for Xamarin certainly makes the technology the most preferred and commonly used to achieve native cross-platform development for Android and iOS applications.

Finally, the type of analysis to be done in this research is intended to be taken from the perspective of a developer new to the mobile development space. Veteran mobile application developers certainly may have adopted any of the technologies used in this research in a quicker, more powerful use case. However, as a novice in the mobile development space, the process of adopting any of these technologies will be met with the same issues that any new technology will

present upon first use. Also, the languages being used in this research vary in their complexity and their utility. For this reason, the analysis that is done will not be based on the usefulness or the simplicity of the languages being used. Analysis will be based on the usefulness of the tools, IDE's, and the time necessary to produce a viable application. The application used for this analysis is a simple calculator that will be written in Xamarin Forms, Swift, and Java. Each implementation will be required to perform these basic calculations: Multiplication, Addition, Subtraction, and Division. In addition, the calculator will be able to display input given by the user and display the correct answer to simple calculations.

Xamarin Forms Development

Android

For Xamarin Forms development to begin, the proper environments had to be created on both a Windows 10 machine and a MacOS High Sierra machine. This process involved installing the Visual Studio IDE for the Windows 10 machine. During the installation process, all necessary Android SDK's must also be installed. For the purposed of this research the following Android SDK tools had to be installed: Android SDK Build-tools 19.1-26 rc2, Android 8.1.0 (API 27), SDK Platform (API 27), Android 8.0.0 (API 26), SDK Platform (API 26), Android 7.1.1 (API 26), and SDK Platform (API 25). These tools allowed the development of the Android variant of the Xamarin Forms application. When running the Android variant of the application, the use of an emulator was decided to be too unstable for this research and all tests were run on a Nexus 5x. This provided a native platform that was not contingent on the virtual machine capabilities of the Windows 10 PC and allowed the application to be restrained to the phone's processing power alone. It is important to note that the reason behind using the native platform of the phone over the use of the Windows 10 emulator was not strictly because of this reason. Unfortunately, there are issues with the tools provided by Visual Studio of Android VM deployments that cause extreme latency in the bootup process. There also is a lack of support for specific phone instances to be spun up through Visual Studio. The common solution to this problem is to invest in Xamarin's cloud platform that allows for testing on their library of devices. This is a paid service and is the preferred method that is suggested by Xamarin for application testing. However, for this research, this option was forgone in favor of recreating the position of a single developer setup. Since the Android Virtual Machines were forgone in this research, the Android debugger needed to be setup to deploy the application to the external Nexus 5X device. This process was straightforward and Visual Studio detected the device quickly. Once the developer mode was enabled on the phone, the application could be deployed to the Nexus 5X via a selection of the phone for the form of deployment. The build process for deploying an application from Visual Studio to test a platform specific application is not directly specified in the project. The platform that you are deploying to must be configured in your project to be "Set as Startup Project." The only indication that the user will be given that this has been completed is that the specified project will be replaced in a bold white look (Figure 1).

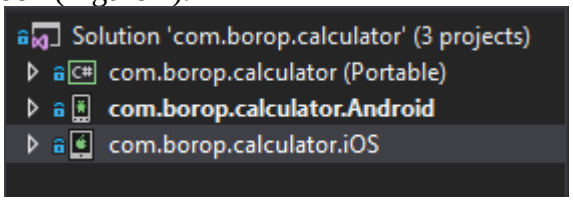

Figure 1. The selected Startup Project

Once a Startup Project has been configured, the options for deploying the application will change to meet specific type of project being deployed. For example, when the Android project is selected as the Startup Project, you are then able to deploy to Android Virtual Machines or a connected device via the Android Debugger. Fortunately, all of the code written for the General project of the application shown in Figure 1 as "com.borop.calculator (Portable)" is used for the Android version of the application. This is where the power of Xamarin Forms is introduced. All the code necessary for the application can be written in the General project but it will still apply to each individual solution. The only way to override this functionality is to specify explicitly in the Android specific application project. This project is where the programmer can introduce platform specific code that may change the way that the application operates on the platform being altered. Figure 2 shows the final Android version of the Xamarin Forms application deployed to a Nexus 5X.

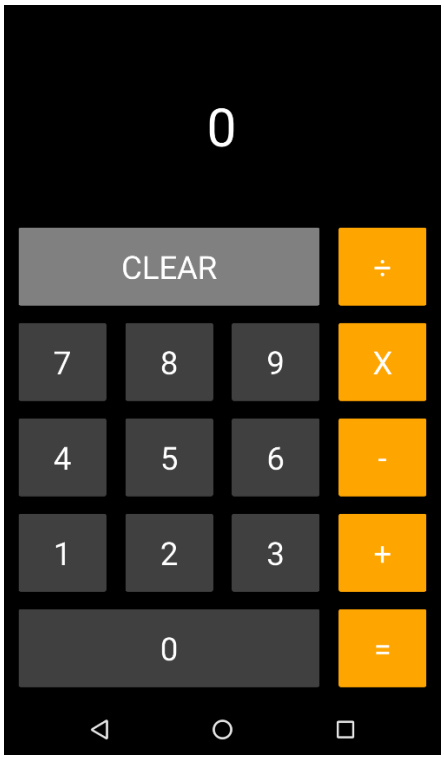

Figure 2. Xamarin Forms Android deployment

iOS

The iOS implementation of the Xamarin Forms application provided its own challenges for the project. As stated before, Apple's Terms of Service dictate that you must compile an iOS or Macintosh application on a Macintosh Computer. In order to abide within these terms, the project needed the ability to interact with a Macintosh. Visual Studio provided a method to achieve this via the local area network. The setup for this process took some time and overall hindered the development process, but it did eventually work. The process involved installing Apple's development tools called XCode and Microsoft's Visual Studio. XCode would install every iOS operating system utility necessary for developing an iPhone application and Visual Studio is able to take advantage of those utilities directly. Since this research intended on working from the perspective of a solo developer, no special development licenses were purchased. This meant that

there needed to be some special “trickery” suggested by the Xamarin website involved in order to test the iOS application. The process involved creating a blank project in XCode, copying unique identifiers relating to signing certificates and provisioning profiles, and using those copied unique identifiers in the Xamarin Forms application on Visual Studio. This process also required copying the entire “solution” as Visual Studio calls it and placing it on the Mac. Unfortunately, deploying an application over the network proves difficult when you must spoof credentials off an official iOS application. Deploying the application operates in a similar fashion as the Android implementation once those credentials have been given to Visual Studio. Figure 3 shows the final iOS version of the Xamarin Forms Application deployed to an iPhone X.

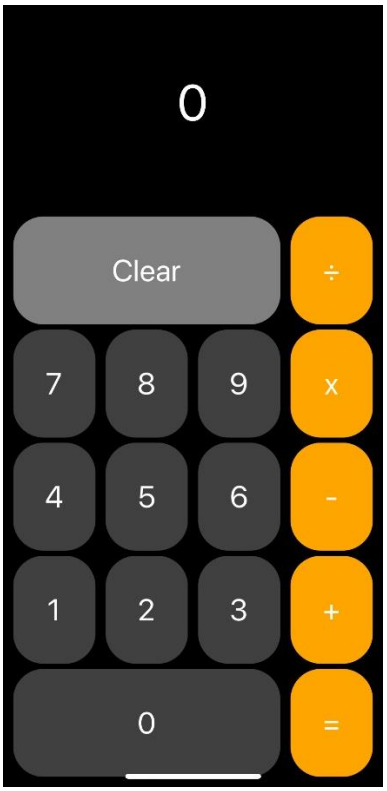

Figure 3. Xamarin Forms iOS deployment

Evaluation

From Figure 2 and Figure 3, it can be easily determined that there are some clear discrepancies on how these applications appear to the user. The first and most glaring is that the buttons are rounded in the iOS interpretation and that the Android interpretation has rounded buttons. This would be determined by the how the operating system received information that was originally written in XAML. Another striking difference that shows up from the XAML as well is that one of the “clear” buttons is in strict uppercase and one has a single uppercase “C” and the rest are lowercase letters. That information was determined by the following code in XAML:

```
<Button BorderRadius="30" Grid.Row="1" Grid.ColumnSpan="3" Text="Clear" FontSize="30" TextColor="#fff" BackgroundColor="#808080" Clicked="OnClear" HorizontalOptions="Fill"/>
```

Observe that the intended text was only reproduced in the iOS variant of the application. This reveals the undesired effect that Xamarin Forms does not hint at on their website. Due to the way

that the cross-platform code is compiled, the code that you write is not what ends up running on the device. When the terminology of "native" is used by Xamarin, this only means that the FINAL running code is native to the platform that it is being executed on. This does not mean that the code that is written is run as native on the platform specified. These observations what are called "side effects" and they can vary in their impact. In this instance there were no side effects that were detrimental to the functionality of the application, rather they simply effected the look and feel of the calculator. This did not hinder the performance of the calculators and they still functioned as expected based on the desired functionality outlined in the introduction.

Android Development

The intention of this development process was to produce the same calculator functionality shown from the Xamarin Forms development process. Native Android applications are written Java and XML. This relates rather directly with the Xamarin process as Java and C# are some of the most similar languages in existence. Also, the fact that XAML is a subset or a derivation of the wider accepted XML made these projects very close. The process started by installing Android Studio. This technology was chosen because it has been the most common way to develop an Android app for operating system's existence. It is also the platform suggested by Google, the owner of the Android operating system. Since Android can be developed on either Windows 10 or Mac OSX High Sierra, Windows 10 was selected to be the platform development would take place on for this portion of the research. Android Studio was just as straightforward of an installation process and utilized the previously installed Android SDK's. This sped up the process though it is important to note that they need to be installed at some point to produce Android Applications. Though the installation process of Android Studio mimicked the process of installing Visual Studio, none of the experiences that went into the actual development process were similar. The Android experience felt unnecessarily complicated and upon the creation of the first test project there were some terrible problems with getting Gradle, an open-source build automation system, to work. Once the project was updated to only support the newest versions of Android, the development process was able to gain traction. Android Studio offered the ability to create the GUI for the application using drag and drop features. This initially seemed to be a great addition to the development process, but it quickly overcomplicated the process. When items are placed into the design tool, they become fixed. This offers no benefits when it comes to scalability on a platform such as Android. With diverse devices that need to be supported, the limitations on the design tool, at least to a new user, seemed extremely disappointing. Another element of design that seemed to be missing from the Android development was the ability to manipulate elements in the XML using Java. The final application ended up having to have Java code for each of the XML buttons rather than working with them directly. This added complexity to the Java code and tightened the coupling. What resulted was a working app without surprises. The questions that come up are that of sustainability. The unfortunate reality is that in order to produce a working application, the project had to be brought up to a very recent version of Android. This isolated over half of Android devices from being able to download and use this calculator application. The final look for the native Android application is found at Figure 4 below.

Figure 4. Native Android deployment

It's obvious from looking at Figure 4 that it has a less desirable interface. This interface was also configured directly for the Nexus 5X used for the Xamarin Android application. The buttons are once again square but they are laid out in a less desirable configuration. The application designer in Android Studio fought the scaling of pre-defined buttons. This lead to them remaining in their default sizes. The signature of the native development of this application can be seen at the top of the application that displays the name of the application. This is part of the default configuration for the application that was not displayed on the Xamarin implementation.

iOS Development

The intention of developing another application for iOS natively was to provide the same functionality using Swift 3. Native iOS application are written using Swift, a replacement to the previously used Objective C language. This language is the farthest step away from the other languages used in the development process of Xamarin Forms and native Android. Surprisingly, the development process of the native iOS application came with the least surprises. The environment as specified earlier, was on a Macintosh running Mac OS High Sierra. Developing for iOS requires the use of a Mac and they provide a development tool called XCode. This installs all the necessary VMs and SDK's to develop for iOS. The process was by far the most painless. The latest version of XCode does not offer a direct method for developing a graphical interface through languages such as XML or XAML. For graphical interface development to take place, XCode offers a much more powerful designer than Android Studio. This interface offers the user the ability to select from several fields and drag them into place. Scalability is also on the forefront of development as the different iPhone sizes can be cycled through and adjustments can be made for each different implementation. This process is made easier because of the limited amount of devices that have to be kept in mind when developing applications. Once the interface was created, Apple's developers guides show how you can drag between the interface and the Swift code. This

creates a hook into the interface. These hooks can then be used to work directly with the interface using the code. These interactions were seamless. This type of interaction also created much less code because XCode handles all the responsibility for connecting the interface to the code responsibilities for the application. The final iOS implementation of the calculator application is below as Figure 5.

Figure 5. Native iOS deployment

As can be seen from Figure 5, the interface is very clean and has an intentional layout. The look closely mimics the native iOS calculator as intended and has all the functionality of the Xamarin Forms calculator and the native Android calculator. The native iOS deployment also provided as seamless deployment to the iPhone X used for testing. XCode immediately recognized the device and did a quick build for deployment. The only hiccups during the deployment process was the built-in security feature to iOS for unsigned applications. Since this research was approached without paying for developer licenses from Apple or Google. The deployed applications cause the operating systems to give warnings about running the applications and requires the user to go enable access to the applications directly.

Conclusions

Xamarin vs Native Android

Throughout the research process there were very different strengths for each of the different development formats. Xamarin Forms was a surprising success. There are clearly some drawbacks with working on an application in a language that is not the original intention for developing applications for either the iOS or Android platforms, but the process has proven to be a powerful one. The XAML development for the front end proved to be extremely an extremely straightforward process. This is a stark contrast to the native Android development process. XML was not nearly as powerful as XAML for Xamarin development. The C# code was able to directly manipulate and pull data from the XAML. This heavily reduced the amount of C# code that needed

to be written because XAML and C# can carry the same responsibilities. XML, however, lacks the capability of being as flexible as its XAML counterpart. In the native Android development process, XML is strictly limited to representing the front end and is not intended to have functional capability on its own. This hinderance caused a severe increase in the amount of time to deploy the Android application vs the Xamarin Android application. Bearing in mind that once the Xamarin Forms application had been written, it was available on both platforms. The entire Xamarin development process ended up being a few hours shorter than the Android counterpart. This means that for a novice mobile developer, Xamarin offers a solution for three platforms that can be developed in the time that it takes to develop a single, identical Android application. The platform is powerful enough that it doesn't give a lot of reason to develop natively for Android unless it's absolutely necessary. Highly graphical games or anything that needs to take advantage of hardware acceleration may see benefit to native development for Android, but your mileage may vary. As far as the native code vs compiled into native code debate, it is easier to fault the operating system for these types of missteps. Since the compiler used for the C# code is a JIT or Just in Time compiler, it goes through the same process as the Java code would. The only reason that the desired look will differ between an iOS and Android implementation of an application is because of the interpretation of the operating systems. The buttons on the natively developed Android application turned out the same as their Xamarin counterparts. Also, the casing concern for the "Clear" button was solved by examining the following code from the native Android XML:

```
<Button
   android:id="@+id/buttonClear"
   style="@style/Widget.AppCompat.Button.Borderless"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_above="@+id/buttonSeven"
   android:layout_alignStart="@+id/editText"
   android:background="@color/ClearColor"
   android:text="Clear"
   android:textColor="@color/ButtonColor"
   android:textSize="20sp" />
```

Notice, the "android:text="Clear" field for the Clear button. Also, notice the amount of information required for this clear button over the code previously shown for the clear button. The operating system simply interprets the Android text into an uppercase format on its own. Therefore, it is no fault of Xamarin Forms for these inconsistencies. The clear victor for the Xamarin Forms vs Android debate according to this research would be Xamarin Forms.

Xamarin vs iOS

The Xamarin Forms vs iOS portion for this research is inconclusive. The reason for this is because there were no surprises when it came to iOS deployment of the Xamarin Application. The behavior was as expected and the functionality was solid. The native development process for iOS was also seamless. The development process for a single iOS application was quicker than the Xamarin Forms implementation. However, a developer looking to deploy to multiple platforms will benefit greatly from the cross-platform capability of Xamarin Forms. This research was able

to confirm the ease of development for iOS applications in both the process of Xamarin Forms and native iOS development via Swift.

Future Research

The area of mobile development would greatly benefit from continued development in this area. The fact is that Xamarin will continue to develop this product. As this product continues to develop, there will be more tools available to make applications better. This product also makes the entire process of getting an application to market much faster. This is supplemented by the fact that a single technology needs to be learned to successfully develop these applications. If companies begin to adopt Xamarin Forms to develop their applications, there will no longer be a need to have individuals specialize in either Android or iOS development. The question that needs to be answered in the future is whether Apple and Google will support this type of development. Apple has consistently sought to remove "templated applications" from the market and there is some discussion as to whether Xamarin Forms will eventually fall into that form of application development.

Resources

Xamarin. (n.d.). Retrieved December 20, 2017, from https://developer.xamarin.com/guides/xamarin-forms/