Haaga-Helia

Xamarin as a tool for cross-platform mobile development

Oleksandr Gridin

Bachelor's Thesis
Degree Programme in
Business
Information Technology
2015

Haaga-Helia

Abstract

Author Oleksandr Gridin	**Year of entry** 2013
Title of report Xamarin as a tool for cross-platform mobile development	**Number of report pages and attachment pages** 21
Advisor Juhani Välimäki	

Xamarin as a tool for mobile development was chosen as a topic for the thesis work because of its fast pace of growth. This technology was founded in May 2011 and now counts more 1.25 million developers who have already proven its worth. With help of this technology mobile development process will come to the new qualitative level where crappy software won’t exist anymore.

The main goals for this project were to show Xamarin’s power with creating cross-platform mobile application and to prove its portability concept by making research that is proven by created prototype for Android and Windows Phone.

As a conclusion it is clearly stated that Xamarin has huge power for code sharing between mobile platforms. The project that has been created during thesis work fully proves main concepts.

Keywords
Xamarin, mobile development, cross-platform development, C#, MVVMCross

Table of contents

Terms and Abbreviations

MVVM	Model-View-View Model
VS	Visual Studio
IDE	Integrated development environment
TFS	Team Foundation Server
SVN	Subversion
SDK	Software Development Kit
UI	User Interface
XML	Extensible Markup Language
CRUD	Create Read Update Delete
WCF	Windows Communication Foundation
JS	Java Script
PCL	Portable Class Library
MVP	Minimum Value Product
OOP	Object Oriented Programing
IoC	Inversion of Control
DI	Dependency Injection

1 Introduction

1.1 Phenomenon and needs

Huge competition in the mobile development market is no longer allows to run applications focused on one operating system, which puts developers to face a difficult choice. They have to choose between few different approaches:

- developing native applications for each platform (using native languages for each of them);
- developing a mobile site using HTML5;
- to build applications using frameworks for cross-platform development.

Cross-platform development approach gives list of benefits:

- Great for the applications that must contain a large amount of multi-platform code, solving the duplicating problem of the logic
- Application will be developed in a short time under some platforms. Code sharing across multiple platforms allows significantly increase development speed.
- Prototypes for Android and iOS can be created without knowing Java or Objective-C.
- When having start-up, it's not needed to implement difficult features, and cross-platform saves valuable time.

1.2 Goal of this research project

The goal of this thesis is to investigate how applications' code using Xamarin technology can be ported between different mobile platforms. And another project target is to prove or disapprove Xamarin's statements about possible percentage of code sharing.

1.3 Scope of this thesis

The scope of this thesis is a research for Xamarin and actual mobile application Windows Phone and Android.

1.4 Out of scope

For this particular thesis project, has been made research about Xamarin and cross-platform development but because application for the 3 platforms is a huge project, there was nothing researched and done for iOS.

2 Background theory – Xamarin, and cross-platform

2.1 Xamarin’s ancestors

Xamarin was founded in May 2011 what meant that it is very yang technology but despite its age already very popular around developers around the world. According to the official website it is used by 1,134,502+ developers by the time when this paper was created. Need for such a powerful tool was created long time ago with widely known frameworks (such as PhoneGap), what proposes the development of cross-platform mobile applications using HTML5 and JavaScript. The idea is that application is created as a normal mobile website using appropriate JS libraries, for example, Jquery Mobile. Then, all this is packed in some sort of the container which for the user appears like a native application. This kind of approach has number of disadvantages:

- There is no access to native elements of UI. So if you want to use the standard button for the iPhone, you have to draw it, and then create for the environment;
- The API get trimmed to work with the platform. Thus, certain features that is platform specific will be unavailable for the development;
- Physically the application runs inside the phone browser (actually inside WebView browser) what means: low productivity and huge display problems;
- The application is unavailable without internet connection.

So all of this was a starting point for creation powerful cross-platform tool using aspects of native mobile development process.

2.2 What is Xamarin

Xamarin - is cross-platform framework for mobile development (iOS, Android, Windows (Phone and Store)) using C#. The idea is very simple, you write your code in your favorite language, with all the usual language features for you like LINQ, lambda expressions, and Generics and asyncs. At the same time with all of the listed, you have a full access to all SDK specific features and ability to create a native UI. As outcome you are getting an application that is no different from native and has the same performance as a native.

The framework consists of several parts:

- Xamarin.IOS - class library for C#, gives developers access to the iOS SDK;
- Xamarin.Android - class library for C#, gives developers access to the Android SDK;

- Compilers for iOS and Android;
- IDE Xamarin Studio;
- Plugin for Visual Studio.

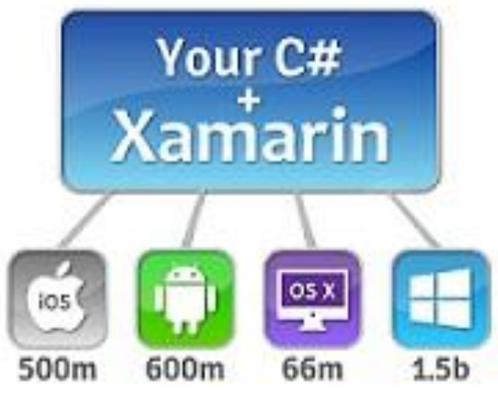


Figure 1. Xamarin structure

Xamarin is based on the open-source implementation of the platform for .NET - Mono. This implementation includes its own C# compiler, runtime environment, as well as basic .NET libraries. The purpose of the project is allowing to run programs written in C#, on operating systems other than Windows such as Unix-systems, Mac OS and others. It is important that development of Xamarin is done by the same people who did development of Mono. And the most important it is not a Microsoft.

In terms of execution of the applications between iOS and Android, there is one key difference, it is the way how apps are precompiled. For the execution of applications Android uses a virtual Java-machine Dalvik. Native applications are written in Java, compiled into a certain intermediate byte code that is interpreted by Dalvik into processor commands at the execution time of the program. This is so-called just-in-time compilation (compilation on the fly). In the iOS is used another compilation model - Ahead-of-Time (compilation before execution). Xamarin includes this difference by providing separate compilers for each platform, what allows the output is real, native applications that run outside of the browser context and can use all the hardware and software resources of the platform.

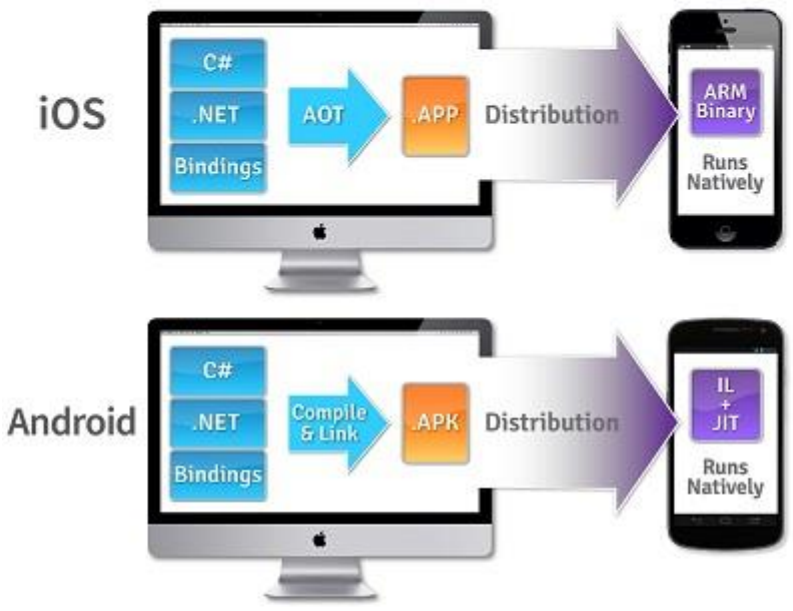


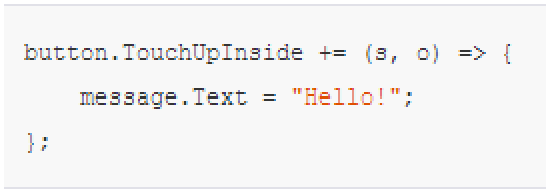
Figure 2. Compilation process on iOS and Android

For the iOS this approach is simple because there is no virtual machine so code just need to be a precompiled into machine before run.

For Android everything is more complicated. When the application is compiled, it is converted C# code into an intermediate bytecode that is clear for the virtual machine Mono and then this virtual machine is also added to the application packaged. Dalvik and Mono are written in C and run on top of the Linux core (as we remember that Android is based on Linux). So when the application on Android runs, both virtual machines start to work side by side and share date through special wrappers.

Let’s go a bit deeper with Xamarin.iOS and its library monotouc.dll, because it shows better difference in writing code rather than Xamarin.Android and its library monodroid.dll, because native Java is very much closer to the C#.

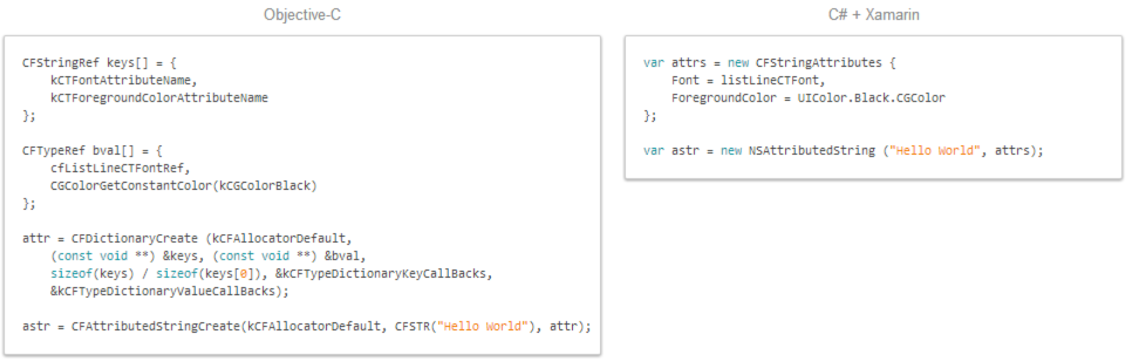
Monotouch.dll class library provides access to all features of the iOS SDK. For the developer this is just a set of C# classes with a good annotation. Within these classes used developed by the Xamarin engineers’ mechanisms of binding to the native classes and methods. It is important that the same mechanism can be used for the binding to any of the libraries written in the Objective-C. Most of the classes and methods are named the same as in the original iOS SDK, however, there are sometimes exceptions (in this case should be used the Xamarin‘s documentation to search for the original name). C# classes use event mechanisms that allows to write beautiful and compact event handlers using lambda expressions:



```
button.TouchUpInside += (s, o) => {
    message.Text = "Hello!";
};
```

Figure 3. Lambda expression example in C#

And if to check example code written in C# it is evident that code looks much more readable and reliable.



```
CFStringRef keys[] = {
    kCTFontAttributeName,
    kCTForegroundColorAttributeName
};

CFTypeRef bval[] = {
    cfListLineCTFontRef,
    CGColorGetConstantColor(kCGColorBlack)
};

attr = CFDictionaryCreate (kCFAllocatorDefault,
    (const void **) &keys, (const void **) &bval,
    sizeof(keys) / sizeof(keys[0]), &kCFTypeDictionaryKeyCallBacks,
    &kCFTypeDictionaryValueCallBacks);

astr = CFAttributedStringCreate(kCFAllocatorDefault, CFSTR("Hello World"), attr);
```

```
var attrs = new CFStringAttributes {
    Font = listLineCTFont,
    ForegroundColor = UIColor.Black.CGColor
};

var astr = new NSAttributedString ("Hello World", attrs);
```

Figure 4. Sample code of creation a string with attributes in Objective-C and C# + Xamarin

For asynchronous development Xamarin provides an opportunity to use the classes from the System.Threading.Thread and System.Threading.ThreadPool namespaces, and a full range of opportunities offered by Task Parallel Library.

2.3 UI development

For each platform Xamarin provides the ability to use native UI development tools and native user interface elements. For Android UI creation may occur directly in the code, or by using declarative approach interface description into XML. For iOS it is ether a code, or native interface design tools, such as different xib-files or a single large Storyboard. Editing these files takes place in the familiar for the iOS developer environment called XCode. For the development of iOS-applications, you need Mac for couple reasons:

- For editing the UI environment XCode;
- The applications require debugging simulator iPhone / iPad, which is also available only on the Mac.

2.4 Portability

According to the developers and Xamarin's marketing campaigns Xamarin is a cross-platform development, what means that the application is written once and can be run on different mobile platforms.

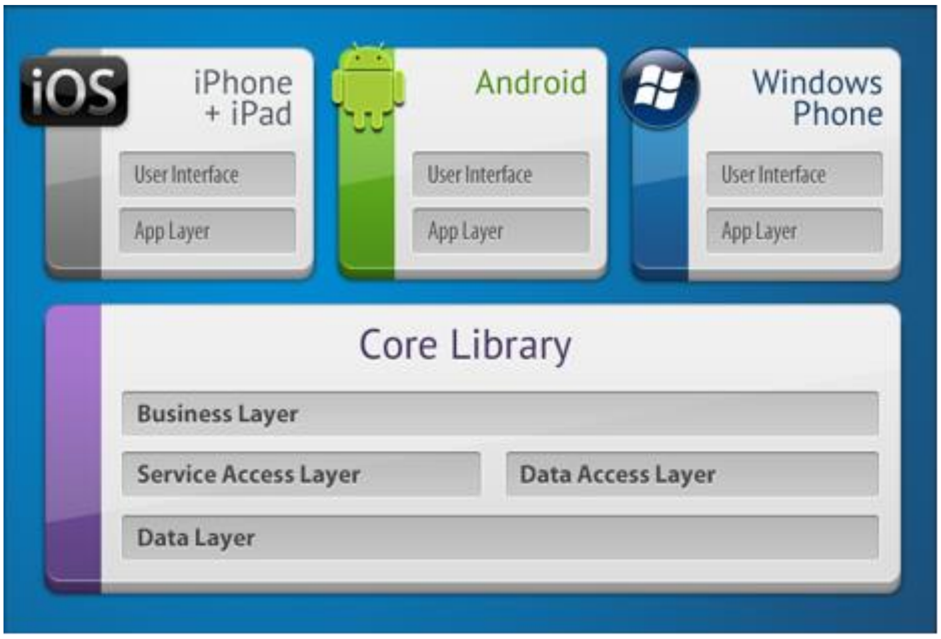


Figure 5. Cross-platform layers

But these rules work not for all code. It means that for each of the platforms, you will need to implement your own UI layer. The code that is responsible for the appearance of the application, needs to be written for each platform separately. This is the price for the possibility to work with platform specific UI. The application can be split into layers listed below:

- Data Layer (DL) - Storage of data, for example, SqlLite database or xml-files;
- Data Access Layer (DAL) - Wrap over the repository for CRUD-Operations;
- Business Layer (BL) - Layer that contains business logic of the application;
- Service Access Layer (SAL) - Layer that is responsible for interaction with remote services (Rest, Json, WCF);
- Application Layer (AL) - The layer containing the platform-specific code that depends on libraries monotouch.dll or monodroid.dll;
- User Interface Layer (UI) - User Interface Layer.

According to the main Xamarin guidelines portability supports on layers that listed above the AL. However, with latest libraries it is possible to create even UI layer as a portable.

2.5 Cross-platform solutions for Xamarin

2.5.1 MVVMCross

MVVMCross is a cross-platform solution that enables developers to create cross-platform mobile applications. MVVMCross works exactly how it described in the chapter 2.3. Basically it is add-on(plugin) that gives to the developers some kind of frame(pattern) to make project looks more readable and supportable with help of MVVM development pattern and portable with connection to the Xamarin. Main advantage of this library is that there are no limitations with UI layer. It is possible to create different designs for different platforms.

2.5.2 Xamarin.Forms

Xamarin.Forms is also a cross-platform solution that enables developers to create cross-platform mobile applications. However, this plugin gives more portability to the developers. Like it was described in the chapter 2.3. Xamarin gives possibility to share all application layers except UI layer while Xamarin.Forms allows developers to share also UI layer by using specially created cross-platform UI controls. Despite on raising percent of portability this library has big amount of implementation limitations:

- UI on each platform has the same design just with native outlines;
- At the moment Xamarin.Forms has quite small number of standard UI controls what bring development process to quite usual approaches;
- According to the bug reports some of the controls has weird behavior what lead to the additional work around;
- In the VS intellisence is very buggy, can give wrong information about errors and warnings;
- Because this is quite new library note much documentation exist and quite many "easy pointed" bugs exist.

As a main plus of this library that creation of the application takes much less time what very important when it's needed to create prototype.

2.6 Xamarin development environment

2.6.1 Xamarin studio

Xamarin Studio - a cross-platform IDE, which works on both Mac OS X, and on Windows. It looks very simple and friendly, but by the simple interface hides a very powerful tool, which included a lot of features, familiar to us in the Visual Studio, such as:

- Nice syntax highlighting;
- Code completion (including the ability to simultaneously import namespaces);
- Easy universal search for file names, types, classes, and members;
- Created possibility to navigate through the project, such as: jump to the description of the class, jump to the base class, list of the places where classed is used;
- Various mechanisms refactoring;
- Well-developed debug mechanisms, including view current value of the variable when you hover pointer on it;
- Native integration with version control systems: SVN, Git and TFS (for TFS you need third-party tools);

But it also has negative aspects. Here are few examples from Xamarin Studio bug track:

- Hotkeys (including copy-paste) are only in English layout. Developers are aware of this problem. There is a report in bag track created;
- From time to time, when you try to put a breakpoint Studio freezes. Despite on the availability of the auto save mechanism, it is a bit frustrating.
- When using the built-in integration with SVN adding new files to the project is not tracked automatically. It means change in the .csproj file will be saved but actual file won't. For each file you need to add it manually to the repository.
- Sometimes the project fails to be compiled. Every time when it happens you need to restart Xamarin studio.

2.6.2 Visual studio

Xamarin offers the opportunity to have development in the Visual Studio after installing a special plug-in (for versions below 2015, VS 2015 has Xamarin plugin in general install), which is available in the business-license that is paid, but you can try it with monthly trial. There is huge amount of pluses:

- Visual studio very old development environment and plenty of developers familiar with it;
- You can use all power of the VS such as ReSharper and other favorite plugins;

And now a bit about some negative aspects. Because VS is very old and famous tool support answer very fast and they fix reported bugs also quite fast. So it is quite difficult to find something that stands in the bug report long time to mentioned it here.

2.7 IoC and DI

Inversion of Control it is an abstract concept, a set of guidelines for writing a weakly bounded code. The point of which is that each component of the system should be as isolated from other as possible without relying on the specific implementation of the other components.
Dependency Injection it is a way how this IoC implemented. According to the approach of inversion of control if we have a client who uses a service, it should be done not directly but through an intermediary, some kind of outsourcing as it shows on the figure below.

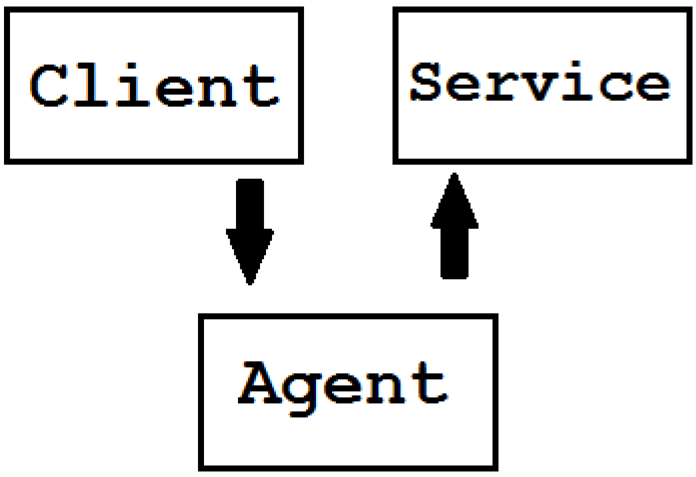
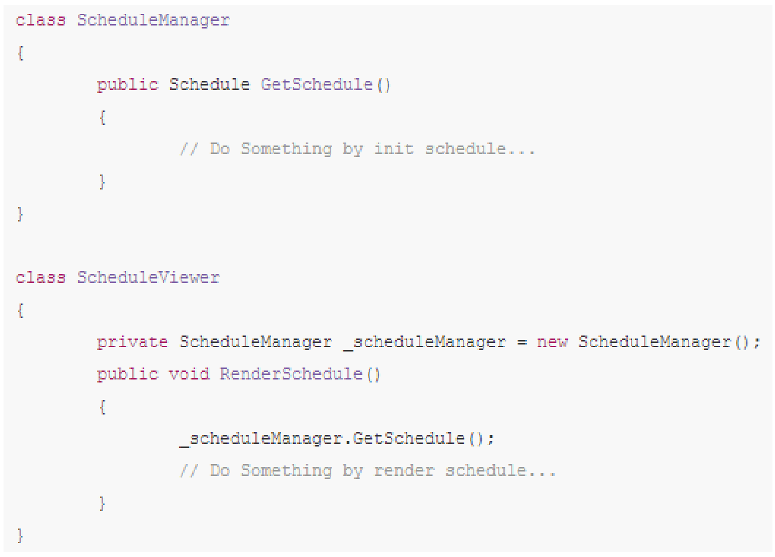


Figure 6. Way how IoC and DI work

The way how technically it will be implemented defines each of the implementation of the IoC approach. As an example let’s look into this. Let's say there is a class that creates a schedule, and another class interpret it (there are usually lots of them, each for platform: one for desktop applications and another for the web, etc.). If there is nothing known about IoC, DI, it will be written like this:

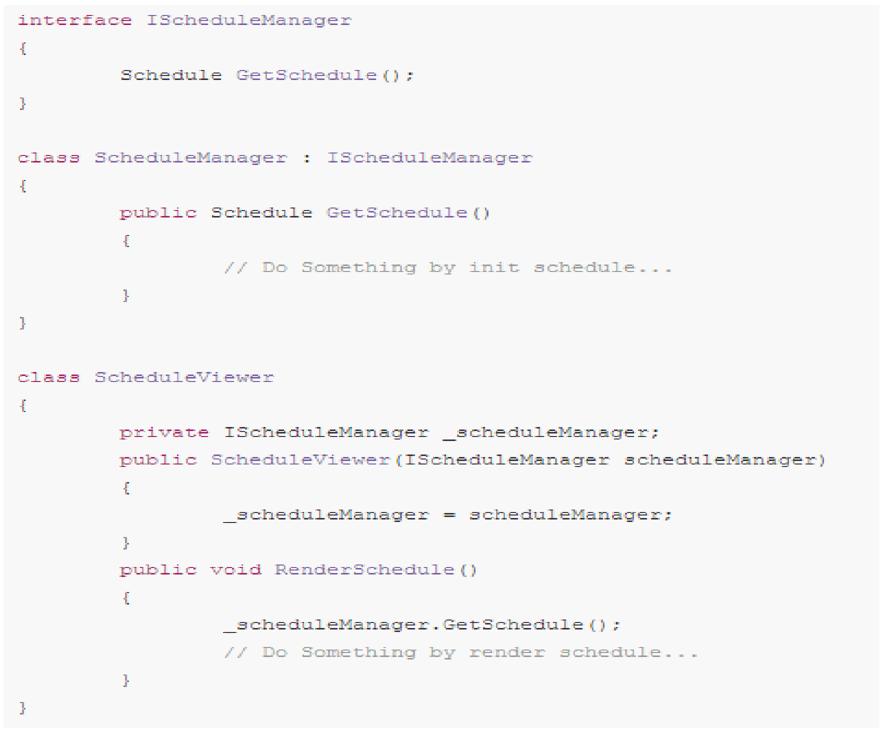


```
class ScheduleManager
{
        public Schedule GetSchedule()
        {
                // Do Something by init schedule...
        }
}

class ScheduleViewer
{
        private ScheduleManager _scheduleManager = new ScheduleManager();
        public void RenderSchedule()
        {
                _scheduleManager.GetSchedule();
                // Do Something by render schedule...
        }
}
```

Figure 7. Example without IoC and DI

Looks all good, code solves the problem, but what if there is a need later to change the implementation schedule manager and / or to have a few of these managers and dynamically replace them. It means that later will be need to change something in ScheduleViewer, and thus test it again. Fortunately, developers are lazy people in a good sense, and do not like to do the same thing twice. By using dependency injection, in order to break this bound it is good to make a connection between these classes weaker, adding a layer with interface IScheduleManager. For this will used one of the DI technology, named Constructor Injection:



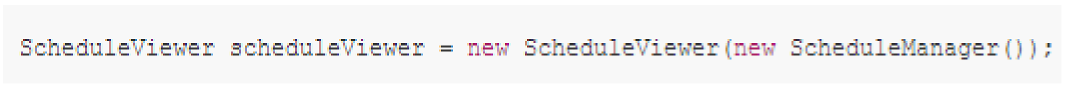
```
interface IScheduleManager
{
        Schedule GetSchedule();
}

class ScheduleManager : IScheduleManager
{
        public Schedule GetSchedule()
        {
                // Do Something by init schedule...
        }
}

class ScheduleViewer
{
        private IScheduleManager _scheduleManager;
        public ScheduleViewer(IScheduleManager scheduleManager)
        {
                _scheduleManager = scheduleManager;
        }
        public void RenderSchedule()
        {
                _scheduleManager.GetSchedule();
                // Do Something by render schedule...
        }
}
```

Figure 8. Example with IoC and DI

And then where there is a need to use ScheduleManager for displaying schedules can be done like this:



```
ScheduleViewer scheduleViewer = new ScheduleViewer(new ScheduleManager());
```

Figure 9. Class call

3 Research plan

First of all, for this I am planning to go through Xamarin's manuals and community, some video tutorials from MVVMCross founder Stuart Lodge. Also to be able to work with Xamarin development I will need to familiarize myself with VS platform, C# programing language and MVVM development pattern. This all will help me to create application following MVP with the features that answer thesis questions.

Whole my development process I will divide into 4 logical parts that will be created in the listed order:

1. Creation PCL (core project with server and application layers);
2. Services creation for server calls;
3. Creation Windows Phone UI;
4. Creation Xamarin.Android UI.

4 Analyzing requirements for Xamarin application

As a target for this project was to answer on the question about code portability. In a way to do these requirements there were done a big peace work with both theoretical and practical approaches. So show better picture lets go through each of these approaches separately.

4.1 Theoretical requirements

As a starting point for this project I took Xamarin website where I read basics about it. With help of this website I got a main idea of what do I need to learn or at least familiarize myself. The list for the is pretty big and consist from this aspects:

- C#
- Visual Studio
- MVVM
- MVVMCross
- Xamarin libraries (monotouch.dll and monodroid.dll)

Despite on a fact that this list includes tons of theoretical materials, I did my best on a way to go through them all. Before I started this thesis work, I already had an experience with all of this aspect from the practical point of view. In overall it was much easier for me go with this project than for any other person without background in this field. However, because I had mostly practical experience, I found lots of interesting theoretical facts.

Now step by step about theory. C# is OOP programing language that gives possibility to write clear and reasonable code. VS is the environment where C# is used for creating different kind of applications. MVVM is development pattern that gives more flexibility with writing and maintaining actual code. MVVMCross is cross-platform solution that utilized Xamarin libraries (monotouch.dll and monodroid.dll) for creation portable layers for the application.

For the C#, VS and MVVM MSDN was more than enough. However, for MVVMCross I had to use some different study methods. Founder of this library is Stuart Lodge created great blog about his library that includes video tutorials. With help of this information I was able to get all needed theory to be able to work with my project.

4.2 Practical requirements

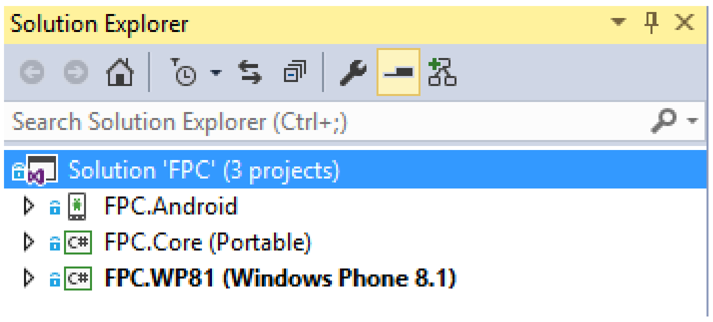
In this particular project practice fully meets theory. So basically for practical requirements were used almost the same web resources except the biggest developers' community Stack Overflow and project samples on the GitHub. With help of this 2 resources I was able to find out all answers for the question concerning actual development. Before creating application, I went through MVVMCross samples on the GitHub and list of plugins that can be used within the project. This helped me to create MVP and make sure that there is all functionality that I would like to implement already exists and support cross-platform ideology. Stack Overflow was used in everyday development process to me with finding out correct solutions and solve problems.

4.3 Proving the concept

According to the research results and supported project I can stated that it is true, around 75% of the code can be shared for the creation mobile applications for major platforms Android, iOS, Windows Phone with help of Xamarin technology. In order to get this high percentage developer must follow strict project architecture and use quite weak (at the moment) amount of plugins that have Xamarin implementations as on the native platforms. Another thing, developer should have quite good skills in C# programing language and be familiar with most of its features, otherwise creation cross-platform solution will be the same as mission impossible.

5 Architectural design for Xamarin application

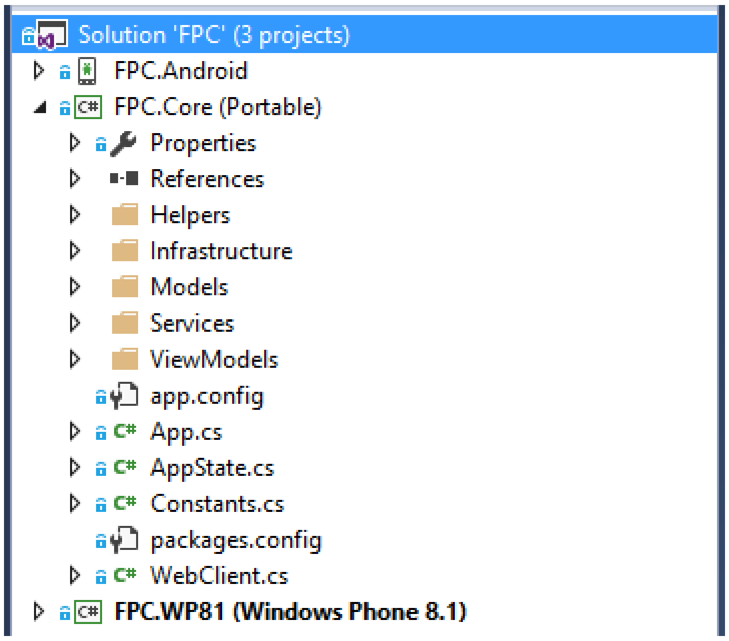
The final version of the project has sharp structure that was followed during whole development process. Main idea was to separate solution into 3 logical parts (projects within the solution). One for core project that includes business logic, services for the server calls, convertors. Another two is for the platform specific features for the Android and Windows Phone and their UIs.



Solution 'FPC' (3 projects)
FPC.Android
FPC.Core (Portable)
FPC.WP81 (Windows Phone 8.1)

Figure 10. Solution structure

The most important part is core project, so called PCL project. It is important because it includes both business logic and server calls that is used on each platforms during deployment. This project also has logical architecture in a way to easy maintain, improve and navigate within it. Here are most interesting 3 folders: Models, View Models and Services.



```
Solution 'FPC' (3 projects)
  FPC.Android
  FPC.Core (Portable)
    Properties
    References
    Helpers
    Infrastructure
    Models
    Services
    ViewModels
    app.config
    App.cs
    AppState.cs
    Constants.cs
    packages.config
    WebClient.cs
  FPC.WP81 (Windows Phone 8.1)
```

Figure 11. PCL structure

Models are both Data Transfer Objects or just an objects that used for storing information pulled or pushed from the server or share information between View Models. Services is used according to the IoC and DI practices in a way to create as weakly bounded code as it is possible. Mainly it works by writing some sort of fuse called Interface that calls from the View Model or any other classes and realization of the interfaces written in the different classes what gives huge flexibility for the developers if the need for example change cloud service, they will need only to modify one class. And last but not least is View Models. Classes with this endings having all of the business logic for each view (not always but mostly) that displayed on the UI. Like it was stated in the theory part, business logic is that layer that shares between mobile clients. Inside of this classes also exists properties and commands. Those 2 things are used for binding between views and View Models in a way to fully separate it from each other, that no logic will exist on a view.

As a result of this architectural structuring of the project we got all business logic inside core project, UI description for each platform is also in the separate projects. This way gives possibility for the developer easy to navigate, maintain and update project. And another good reason, it is a good structure for the people who just came to work with the project because it has clear structure.

6 Evaluation

Main goal of this thesis was to prove or disapprove with Xamarin's marketing campaigns statement that with help off this technology, 75% of the code can be ported to another mobile platform. As it said before, according to my research and project, I can state that it is true. With some minor difficult that are described in the chapter 7, developers can meet this announced performance. Results are valid according to the community and finalized project.

In my personal opinion sources that I've been using like, Stack Overflow, Xamarin Community, GitHub and MSDN are great. With help of this resources developer can forget about libraries and because most of the technologies are widely used most of the problems were solved, it is just needed to spend some time to find answers for the needed questions.

Methods and tools that I picked for this thesis project fully proved themselves. For getting answers for my question I've been using online communities and resources what was brilliant. And another method was empirical(practical) where I created actual project to support my research. As a main tool I used Visual Studio while another option was Xamarin studio. There are 2 main reason for this selection. First, I am working with VS from it 2008 version, so I have quite good experience about it. Another Reason is package manager control. VS has both NuGet and Xamarin package control while Xamarin Studio has only its own what narrow range of the plugins that can be used within the project.

6.1 Further research

For this particular thesis work I haven't done anything for the iOS platform. There are few reasons. First development for the iOS requires Mac OS (MacBook or so) or virtual machine that can build projects. There are no free build machines what makes development for the iOS quite expensive. Second that I didn't have experience for the iOS UI development and don't know how native SDK function.

If we talk about technology, they keep good track on reaching 100% code portability (now 75% according to the official website). Latest Xamarin.Forms solution is a good example of it. While previous solution could port only up to 75% of code, like all layers except UI layers, Xamarin.Forms supports UI sharing also. This approach is even more strict that MVVMCross, however, they are doing their best in a way to reach 100% code portability. Maybe after a while we will see more flexible solution for cross-platforming from them.

Biggest problem for the Xamarin is that they are creating cross-platform solution based on what Android and iOS platforms has. Because Android and iOS constantly improve their functionality, Xamarin always one step back. I think solution can come later when Xamarin will sign partnership agreement with Apple and Google, and they will work with the functionality at the same time, like they did with Microsoft.

Amount of the functional libraries for different cases should be bigger. Every time when there is need for implementing some functionality, exists risk that it will very difficult or includes many hours for work around, creation additional adapters for each platform separately what eventually brakes down portable ideology. Also Windows Phone is very tricky because unfortunately it doesn’t have many implementations for famous libraries. Sometimes it is very difficult to find Xamarin’s implementation for some native libraries.

7 Summary

With help of this thesis project I have successfully created a mobile prototype using Xamarin technology for the Android and Windows Phone. During this project I met plenty of difficulties, such as lack of programing skills within the cross-platform solution, lack of features and libraries that Xamarin can offer. Despite of this difficulties I was able to answer thesis question that Xamarin really is a great tool. It gives possibility for the developer to create mobile cross-platform application with minimum resources needed for the project. One skillful team can easily carry development for all of 3 major platforms (with native development 1 team per platform that creates the same business logic for each application) what is much cheaper from the money perspective.

In my personal opinion, Xamarin is excellent tool, that has a great future. Most of the applications that we are having in the application stores has the same instances for each platform, because all of the companies would like to create services for each platform covering all kinds of users. That’s why it is so important in the technology world, that developers are not repeating the same apps to the different platforms, but having time to really invent something bigger and more valuable for the mobile society.

It is not fully proven by the developers, but eventually it will be and then will spread even more around the world to prove its worth.

Bibliography

BlogSpot. MVVMCross N+1 table of contents. URL: http://mvvmcross.blogspot.fi/ Accessed 15.11.2015.

GitHub. MVVMCross sources. URL: https://github.com/MvvmCross/MvvmCross Assessed: 15.11.2015.

GitHub. MVVMCross tutorials. URL: https://github.com/MvvmCross/MvvmCross-Tutorials Assessed: 15.11.2015.

HabrHabr. Cross-platform development for mobile with Xamarin. URL: http://habrahabr.ru/post/172121/ Accessed 13.11.2015.

HabrHabr. Working with the Xamarin: experience of work in 2 different projects. URL: http://habrahabr.ru/company/icl_services/blog/270051/ Accessed: 11.11.2015.

HabrHabr. IoC, DI. URL: http://habrahabr.ru/post/131993/ Accessed: 27.11.2015.

Xamarin. Getting Started with Android. URL http://developer.xamarin.com/guides/android/getting_started/ Accesses: 5.10.2015

Xamarin. Getting Started. URL: http://developer.xamarin.com/guides/cross-platform/getting_started/ Accesses: 5.10.2015

Xamarin. Limitations. URL: http://developer.xamarin.com/guides/android/advanced_topics/limitations/ Accessed 15.11.2015.

YouTube. Stuart Lodge channel. URL: https://www.youtube.com/user/MrHollywoof Accessed: 16.11.2015