Charles University in Prague

Faculty of Mathematics and Physics

MASTER THESIS

Petr Čečil

Cross-platform Mobile Development

Department of Software Engineering

Supervisor of the master thesis: RNDr. Filip Zavoral, Ph.D.

Study program: Informatics (N1801)

Specialization: ISS (2612T043)

Prague 2015

I would like to thank my supervisor Filip Zavoral for his supervising and for his time, ideas and advice.

I declare that I carried out this master thesis independently, and only with the cited sources, literature, and other known sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague on 3rd December 2015 signature

Název práce: Multiplatformní mobilní vývoj

Autor: Petr Čečil

Katedra / Ústav: Katedra softwarového inženýrství

Vedoucí diplomové práce: RNDr. Filip Zavoral, Ph.D., Katedra softwarového inženýrství

Abstrakt: Práce se bude zabývat problémy vývoje multiplatformního software pro mobilní zařízení. Autor bude analyzovat stávající frameworky a nástroje; poukáže na jejich silné a slabé stránky. Na základě této analýzy bude pilotní aplikace navržena a realizována pomocí nejlepšího frameworku. Zvláštní pozornost bude věnována metodice multiplatformního vývoje.

Klíčová slova: Multiplatformní, mobilní, vývoj software

Title: Cross-platform Mobile Development

Author: Petr Čečil

Department / Institute: Department of Software Engineering

Supervisor of thesis: RNDr. Filip Zavoral, Ph.D., Department of Software Engineering

Abstract: This thesis will address problems of cross-platform software development for mobile devices. The author will analyze existing frameworks and tools; he will point out their major strengths and weaknesses. Based on this analysis, a pilot cross-platform application will be designed and implemented using the most versatile framework. Particular attention will be paid to the methodology of cross-platform development.

Keywords: Cross-platform, mobile, software development

Contents

1 Introduction

As organizations expand their applications to mobile devices, finding a suitable mobile framework is crucial. Smartphone devices are the fastest growing technological innovation of our time. Their adoption is ten times faster than the PC boom in the 80s, two times faster than the Internet boom in the 90s and three times faster than the last social networking explosion [1]. Smartphone sales outgrew PCs in less than two years [2] and the rise of tablets was even quicker.

According to the American information technology research company Gartner: *By 2016, 70% of the mobile workforce will have a smartphone, and 90% of enterprises will have two or more platforms to support* [3]. The impact is already very significant, e.g. in 2013, Facebook experienced 54% year-over-year growth in active mobile users, which generated 30% of their revenue growth [4].

1.1 Device Platforms

Every smartphone (or mobile device with communications capability) usually contains two mobile operating systems: primary end-user software platform and low-level real-time system operating hardware. The second low-level operating system is stored in firmware and runs on a baseband processor[1]. This software is proprietary and closed and often contains security vulnerabilities [5], usually because of poor maintenance and absence of updates from the manufacturer. Vulnerabilities are sometimes exploited by users or developers to gain root access to the device system and by that they can get more from their devices.

The main problem complicating development is that there are three different mobile platforms: Android, iOS, and Windows Phone.

[1] A baseband processor is a device (or chip) in a network interface managing all antenna (radio) functions

1.1.1 Android

Android by Google Inc. is based on the Linux Kernel. It is the most popular operating system on smartphones [6], even though it is not a popular system on PCs. Although the Android operating system is an open-source software, standard distribution from Google contains a lot of bundled closed-source software.

Until the release 2.0 (Eclair), it was used only on mobile phones. Android 2.x versions were also used on tablets, and version 3.0 (Honeycomb) was the first release optimized for devices with larger screens, particularly tablets. The current version is 6.0.

Android as an open-source project has an active community of developers who used its source code to develop their modified versions of the operating system. CyanogenMod [7] is very popular in open source community. FlymeOS and MIUI are partially closed-source systems developed by Chinese electronic companies Meizu and Xiaomi for their smartphone products.

1.1.2 Apple iOS

Previously known as iPhone OS, iOS was created by Apple Inc. for their new line of smartphones in 2010. It is a closed source proprietary OS based on open source Unix Darwin core OS. Currently, all iOS devices are being developed by Apple.

1.1.3 Windows Phone

In 2010, Windows CE/Mobile OS, created by Microsoft, was replaced by Windows Phone, mostly because user interface (UI) was not optimized for touch input by fingers. The user interface is derived from Microsoft's Metro design language[2] and

[2] Metro is a typography/geometry design language created by Microsoft for user interface, it evolved in Windows Media Center and Zune and was formally introduced during the unveiling of Windows Phone 7 [47]. Its new official name is: 'Microsoft design language'."

mainly consists of multiuse tiles replacing classic icons used in Android or iOS. Windows Phone devices are primarily made by Microsoft Mobile (formerly Nokia), HTC and Samsung. In 2015, Microsoft announced that universal Windows 10 Mobile will replace Windows Phone.

From a business point of view, the mobile landscape is very heterogeneous and more fragmented than the PC era. From 2009 to 2012, Android raised from 4% smartphone market share to 66%. On the other hand, iOS users are spending twice as much time on devices [8]. Therefore, mobile devices are vital as a development platform.

1.2 Goals

Based on the original assignment, these goals were chosen for the thesis:

- Analysis of existing cross-platform frameworks and tools. There are generally two competing frameworks for multiplatform development. These details are described in section 2.
- Creating methodology for cross-platform development (based on the analysis). When a proper framework is chosen, there are still a lot of architectural and design decisions based on framework features. These decisions are discussed in section 3. This section also contains author's own experiences with mobile development.
- Implementation of a pilot cross-platform application. Based on the analysis and methodology, a multiplatform pilot application was created. Section 4 contains the description of the development and architecture.

2 Analysis of existing frameworks

Some basic usability factors should be defined before comparing frameworks, because choosing a good framework influences the final application usability tremendously.

Many factors contribute to the decision: whether users will use an app, including design, reliability, performance and overall good usability. Users expect that the mobile application makes contextually-relevant information instantly accessible and it should deliver this content promptly and intuitively. Applications that do not fulfill this requirement might get uninstalled quickly.

Here are some of the basic application design guidelines: The developer should use standard native interface controls to conform to each platform design conventions and vendor's guidelines, so users can quickly adapt to an interface. The app should also offer optimal performance using platform-level acceleration and have full access to underlying platform/device functionality.

2.1 Native Tools

The most common technique used today for creating mobile apps for multiple mobile device platforms is writing the app from scratch for each vendor. Apple iOS applications are very fast, but usually developers are stumbling with Objective-C concepts such as pointers and difficult debugging, which differs heavily from high-level languages. Therefore, Apple created Swift language [9] that adopts more high-level concepts from C# and Java. Apple also provides Xcode [10], an integrated development environment (IDE), a suite of development tools for creating software also for iOS. It contains built-in Interface Builder for constructing graphical user interfaces.

Until version 4.4, Android used Dalvik [11] -- a process virtual machine[3] to run apps. Dalvik was using a just-in-time compilation of Java bytecode. This late compilation contributed to inadequate performance. In Android 4.4, a new ART (Android Runtime) environment was introduced. ART used new ahead-of-time compilation (during installation) of apps allowing faster runtime. It became the only runtime option in Android 5. Android Studio is an IDE for developing on the Android platform; it contains a WYSIWYG[4] editor for real-time app rendering.

Android always used reimplemented Java libraries. From my point of view, the biggest drawback is that "Android Java" is always an about two-years-old subset/implementation of standard Oracle Java libraries.

Microsoft's Windows Phone .NET usually has a faster development cycle than Android, because it uses .NET natively. For example, the new version of C# 6 compiler can be used on mobile devices the day it was introduced for the .NET stack. Windows Phone apps are based on XNA, a Windows Phone specific version of Silverlight [12], the Windows Phone App Studio [13], or the Windows Runtime [14]. These technologies allow developers to develop apps simultaneously for Windows Store (desktop and tablet Windows) and for Windows Phone. Apps are usually designed and tested in Microsoft's Visual Studio.

Although each platform vendor provides free tools for mobile development, some (usually small) payments are required for a platform developer license (to allow publishing of the app). A summary of native tools is described in Table 1.

---

[3] A virtual machine is an abstract computing machine that can run a Java program compiled into Java bytecode.

[4] A 'What You See Is What You Get' editor is a system in which the content's appearance during editing is very close to the final product

Table 1 - Summary of native mobile tools

platform	iOS	Android	Windows Phone
runtime	machine code	Dalvik/ART	.NET
tools	• Objective-C • Swift	• Java • Android XML	• C# • C++ • XAML • HTML5

2.1.1 Strengths and Weaknesses of Native Development

Because native apps run on platforms designed for them, there are usually no performance problems caused by the framework. Native solutions also have a large community and tools for development are very mature. On the other hand, there is a need for high platform dependent knowledge, and the biggest drawback is that the developer cannot reuse any code at all (including business logic). A partial solution that is usually used is moving most of the business logic on a web server as most apps today need some online backend anyway. But this impacts the performance profoundly.

2.2 Apache Cordova

Cordova (previously named PhoneGap) [15] is an open source hybrid[5] mobile application framework that uses a native wrapper to run interpreted HTML5/CSS3/JavaScript code. Access to particular system resources is granted through platform-specific wrappers included in Cordova. The resulting application is hybrid, meaning that it is partially native and partially displayed in the application's web view (as shown in Figure 1). JavaScript notifies the browser using an

[5] Hybrid apps are neither truly native nor purely web based.

asynchronous call, and the browser calls internal commands; then it can return requested information in a JS callback[6].

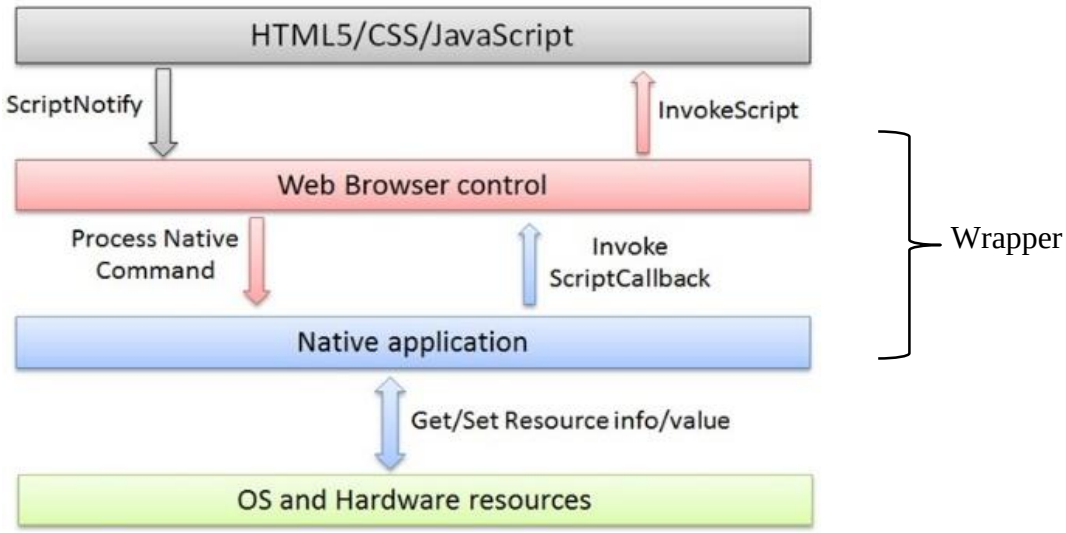

Figure 1 - Apache Cordova architecture [16]

The framework can be extended with native plug-ins allowing developers to add functionality called from JavaScript. HTML5 now provides access to hardware. However, browser support for HTML5 based device access is not consistent across mobile browsers. To overcome these limitations, the framework embeds HTML5 support into an in-app web browser (independent of a device browser). Table 2 summarizes framework behavior on different platforms.

At least two existing frameworks use this open source solution: Adobe's PhoneGap [17] and Visual Studio Tools for Apache Cordova [18]. PhoneGap is a productized version and a complete ecosystem built on top of Cordoba. VS Tools adds an installer for SDK, tools and libraries and also build and debug tools. It also has a library for

[6] JavaScript statements are executed line by line. However, the next line of code can be run even when the previous required statement is not finished. To prevent this, JavaScript allows callback functions. A callback function is executed when the current required statement is finished.

integrating the app into MS Azure[7] cloud backend, Microsoft Azure Mobile App Services [19].

Table 2 - Summary of Apache Cordova framework tools

platform	iOS	Android	Windows Phone
runtime	Cordova (iOS wrapper)	Cordova (Android wrapper)	Cordova (WP wrapper)
tools	• HTML5/JavaScript		

2.2.1 Strengths and Weaknesses of Apache Cordova

The most obvious advantage of this framework is no need for emulators at early stages of development. This benefit simplifies the presentation of demos and makes progress faster as it eliminates deployment on an emulator (the app can be run in a regular browser). Also, it is free and open source; no initial investments are needed. No particular platform knowledge is required, just HTML5 and JavaScript (at least at the beginning).

However, there is no guarantee that application development will be successful. Adobe Systems warns that apps built using PhoneGap/Cordoba might be rejected by Apple for poor performance and "exotic" look [20]. JavaScript is not compiled, so the developer cannot benefit from compile time errors. JavaScript has a poor performance and touch delay with limited mobile device resources. Apps that need offline mode might be hard to create, as they need a native plugin for every platform. Until today, some hardware wrappers implementations are still missing or incomplete. There is a very clear chance that companies will need to pay for the commercial solution anyway, because some parts of the framework are not yet implemented, or for performance reasons.

[7] Microsoft Azure is a cloud computing platform and infrastructure used for deploying apps into the global network of datacenters.

2.2.2 Performance of Apache Cordova

Native performance is hard to achieve with frameworks that interpret code in runtime. Applications written in Cordova have a long loading time, poor performance, and they are almost unusable with highly CPU-bound work. The good thing is that the application package size is considerably smaller than Xamarin apps [21].

In 2013, high profile companies such as LinkedIn and Facebook abandoned HTML5 mobile apps, with Mark Zuckerberg saying: "*Betting completely on HTML5 is one of the, if not THE biggest strategic mistake we've made*" [22]. Now they both use native solutions.

2.3 Xamarin

The Xamarin Platform [23] has two parts. It consists of Xamarin.Android (formerly known as MonoDroid) and Xamarin.iOS (or MonoTouch). Development resembles native development using native tools. But because it uses C#, the developer is allowed to reuse business logic and backend between platforms.

Xamarin basically wraps iOS, OSX, Android, and Windows Phone platforms to create a unified way of accessing them through .NET (as shown in Table 3). This architecture can be a very complex problem because underlying technologies evolve and change quite frequently. For example when iOS introduced 64bit support Xamarin needed one year to compensate this. This quick development resulted in a lot of bugs that might frustrate end-developers [24].

Table 3 - Summary of the Xamarin framework

platform	iOS	Android	Windows Phone
runtime	ARM assembly code	native assembly	Windows Phone .NET
tools	• C# / XAML		

In 2014, Xamarin released a new technology called Xamarin.Forms which allows more code reusability because of using the same UI code for all platforms. The biggest weakness is a missing UI designer for Xamarin.Forms.

The overview in Figure 2 displays a few building blocks: Xamarin supports both Visual Studio (not in cheaper versions) and Xamarin Studio IDE. Xamarin Studio is the only option with cheaper licenses of Xamarin, and it does not have as many features as Visual Studio and also it is not compatible with VS extensions such as ReSharper. However, Xamarin Studio can run on Mac computers (which is not possible for Visual Studio).

Both IDEs support pre-build components from Xamarin Component Store. This store is not as good as Visual Studio NuGet because it doesn't support automatic updates of packages. But thanks to the open source community, the packaging system NuGet quickly compensated and most of the components are also available through NuGet.

Figure 2 - Xamarin Platform Overview [25]

2.3.1 Strengths and Weaknesses of Xamarin

The framework is very fast; apps run nearly as fast as native apps (this topic is discussed in more detail in section 2.3.2). Because the .NET support potential to reuse server code is significant, eg. some of the Business logic can also be used in mobile clients if the Internet is not accessible. Nearly maximum code reuse can be achieved when using Xamarin.Forms. As a side effect, this technology reduces developer or

typo-induced bugs. The framework displays standard native user controls that conform to the platform's design conventions. As an essential feature, the application has a 100% access to underlying platform-specific capabilities and hardware acceleration. Recently, it was also made possible to use the ASP.NET Razor templating engine [26] for generating HTML in applications.

Because Xamarin is in rapid development, frequent updates during development are needed. The developer should also be aware of memory leaks because of hidden bugs in UI implementation. Xamarin.Forms does not have a UI designer; the only way to view UI is a compilation. Large parts of Xamarin API is protected, which leaves tiny space for workarounds in case of bugs in the framework. It should be noted that Xamarin.Forms are still a bit slower than Xamarin Platform device development [21]. There is also a licensing problem. The cheapest license for small developers doesn't include support for Visual Studio, which can be limiting for starting projects.

2.3.2 Performance of Xamarin

Performance tests on Android and iOS were compared in [21] [27]. Xamarin.Android is faster than Dalvik native apps or ART in almost all cases [28], and Xamarin.iOS is comparable to native Objective-C [21]. Although Xamarin.Forms is a little slower than the Xamarin Platform [21], they are still faster or equal compared to full-native solutions. It is more obvious in the comparison between Android and Xamarin. This is, in my opinion, because .NET or Mono runtime is more optimized than Google Java runtime. Figure 3 depicts a performance test of the basic binary tree, associative array implementation, and N-body simulation of the Jovian planets benchmark [29] in Java and C#. Source code can be found on GitHub [27]. Devices used for testing were HTC Flyer tablet with Android 2.0.3 and Iconia Tab (Android 4.0.3).

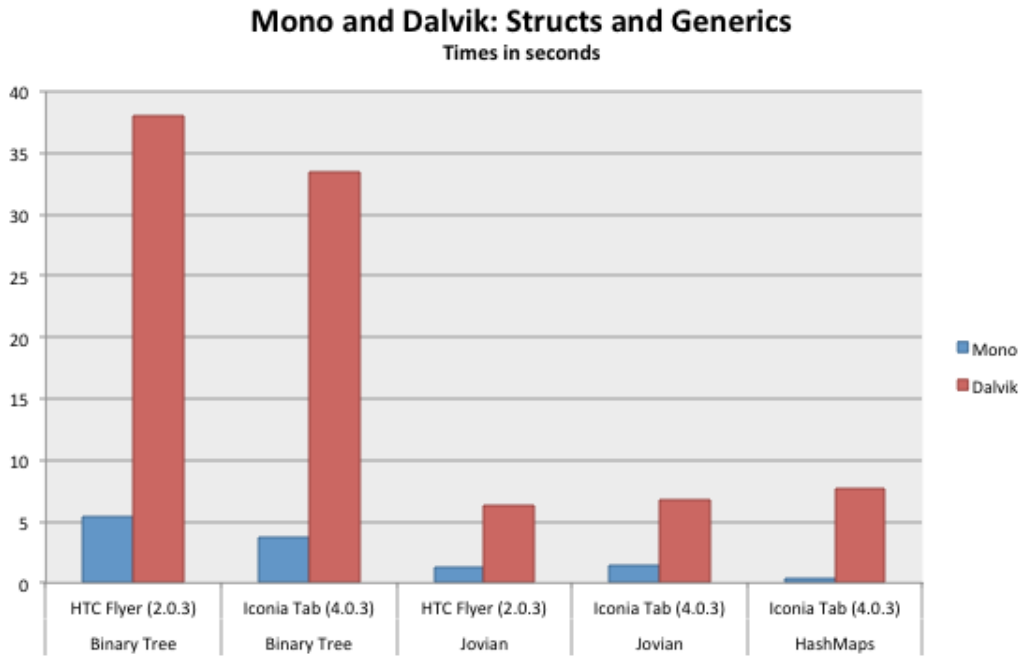

Figure 3 - Mono / Dalvik comparison [27]

The minimal size of a Xamarin app is about 12-16 MB because of the size of the framework. However, with bigger "real" apps, it is anticipated that the difference will be less significant [21]. In my experience, a big difference can be achieved using linker optimization, as the entire .NET framework is quite large.

2.4 Other tools

Other tools such as Telerik Platform Icenium [30] were not discussed, because they are based on Apache Cordova and they share the same advantages and disadvantages. Most of the fully native complex universal frameworks are still in alpha stage, with the only exception of Xamarin.

2.5 Conclusion

Today, the most used method for multiplatform mobile development is a separated native development for every platform. This process requires a big team of specialized experienced developers allowing multiplatform releases. Cordova represents one way to avoid these costs. But sooner or later, it is necessary to rewrite applications to a native code completely or using Xamarin because of performance.

Cross-platform frameworks usually support the lowest common subset of UI and hardware API across all of their supported platforms. Developers must limit their apps to API supported by the framework, which usually leads to frustrated users. Another possibility is to use platform specific languages and end up with a hard-to-maintain fragmented code.

Xamarin exposes 100% of the underlying platform, and there is a smaller chance that some framework limitation will be reached. There is also a student program that allows students to acquire a full license of Xamarin for free. These are the reasons why Xamarin was chosen for the pilot application.

3 Methodology of Cross-platform Development with Xamarin

Xamarin.Android and Xamarin.iOS are built on top of Mono [31], an open source version of .NET Framework. iOS applications are compiled directly to ARM[8] assembly code. Android apps were traditionally compiled to bytecode and then Just-in-Time compiled to native assembly when the app launched. New Android 5 ARM runtime also introduced AOT compilation, as described in chapter 2.1.

Xamarin apps are compiled using Xamarin Mobile Profile, which is a subset of .NET 3.5 BCL, a Base Class Library with a set of basic types, functions and features of .NET. Xamarin Profile names are MonoTouch [32] for iOS and Mono.Android [33] for Android. In addition to BCL, they contain iOS SDK and Android SDK libraries allowing to access mobile system resources.

3.1 Introduction to Dependency Injection

In the multiplatform app, it is good practice to decompose platform specific implementations into services with some defined Interface. There is a design pattern created for this called Dependency injection [34]. An injection is defined as passing dependency to an object that can use it. This responsibility is delegated to a dependency resolver (injector). Whenever a developer needs an instance of dependency, it must be obtained through this resolver; this practice is required because

[8] ARM is a family of Reduced Instruction Set Computing (RISC) architectures for processors. Compared to typical (Complex Instruction Set Computing - CISC) x86 processors, these require fewer transistors. This simple design (which also reduces heat and power use) is desirable for portable devices such as smartphones.

it follows the single responsibility principle[9]. The resolver also takes care of the lifespan of the object (e.g. singletons).

Xamarin.Forms have a dependency resolver called `DependencyService` that allows shared code to access platform specific code. There are three parts of `DependencyService` usage:

- **Interface** – An interface in shared code should be defined. It describes the functionality of platform specific code.
- **Registration** – An implementation of an interface needs to be registered so that `DependencyService` can create instances of it. This registration can be easily done by adding `Dependency` attribute into the implementation.
- **Location** – By calling a dependency resolver (`DependencyService.Get<>`), the client receives an instance of dependency. This location concept also allows shared code to access platform-specific code.

An example of usage is described in Listing 1. Interface `ITextToSpeech` has only one method, Android implementation is registered by attribute, and when the Speak method is required, it can be called through `DependencyService.Get<ITextToSpeech>`.

[9] The single responsibility principle states that every class should have responsibility over a single part of functionality of software, in this case – dependency resolution

```
// interface
public interface ITextToSpeech
{ void Speak (string text); }

// registration
[assembly: Xamarin.Forms.Dependency (typeof (TextToSpeech_Android))]
public class TextToSpeech_Android : Java.Lang.Object, ITextToSpeech,
TextToSpeech.IOnInitListener
{ /*implementation*/ }

// location
DependencyService.Get<ITextToSpeech>().Speak("Hello");
```

Listing 1 - Example of DependencyService usage

3.2 Code Sharing

The biggest advantage of cross-platform frameworks is code-sharing. It lowers the amount of redundant code, and with less code redundancy there is a lower chance of inducing new logical bugs. The goal of code-sharing strategies is to support the architecture shown in Figure 4. Single codebase can be used by different platforms. This Figure illustrates code structure in Visual Studio projects. Data layer uses a local database, service access layer accesses services from the network and business layer contains local application logic. Data, service access, and business layers are shared across platforms, and the only platform specific code is in platform-specific projects.

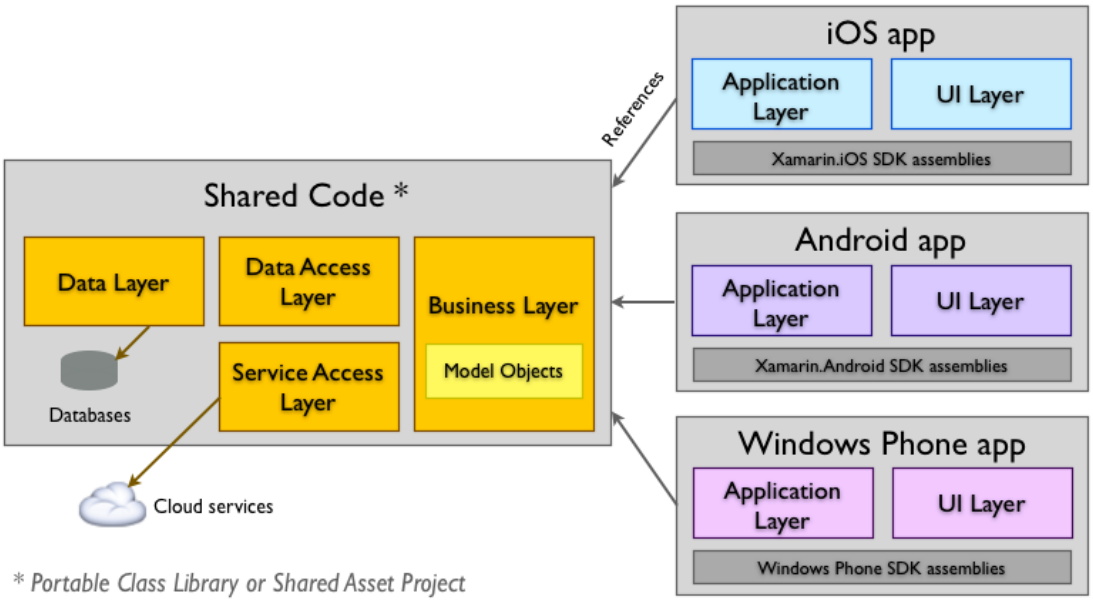

Figure 4 - code-sharing architecture [35]

There are three basic methods of code sharing between platforms: Shared project, Portable class library and the combination of both. These three methods are described in next chapters.

3.2.1 Shared Project

Visual Studio 2013 Update 2 introduced a new type of project template - Shared Project. It is not compiled as a dll but as a part of the parent application. Compared to class library, Shared Project also allows sharing of code different than C# (JavaScript, XAML[10], etc.). Also, with a shared library, the developer is always able to debug all of the code.

Shared project methods use Visual Studio's Share Project to organize source code and uses `#if` compiler directives or Xamarin.Forms blocks to manage platform specific requirements. Because of this, each project includes almost all shared source files as shown in Figure 5.

3.2.1.1 Advantages

This method allows code sharing between platforms without creating additional libraries. The code can be branched by compiler directives (e.g. `#if __ANDROID__`) or by a Dependency Injection. The application can use platform-specific libraries. Code from Shared Project can be accessed from platform specific app layer.

3.2.1.2 Disadvantages

Shared Project has no output assembly, so it cannot be shared as a separate DLL. Some tools (e.g. C# language resources) are not available for Shared Projects. As Shared Project is quite a new feature, some VS tools (e.g. ReSharper) show false errors when

[10] XAML - Extensible Application Markup Language is a XML based user interface language used in Windows Phone and WPF desktop applications.

using Shared Projects. Refactoring with “inactive” compiler directives will not update the code.

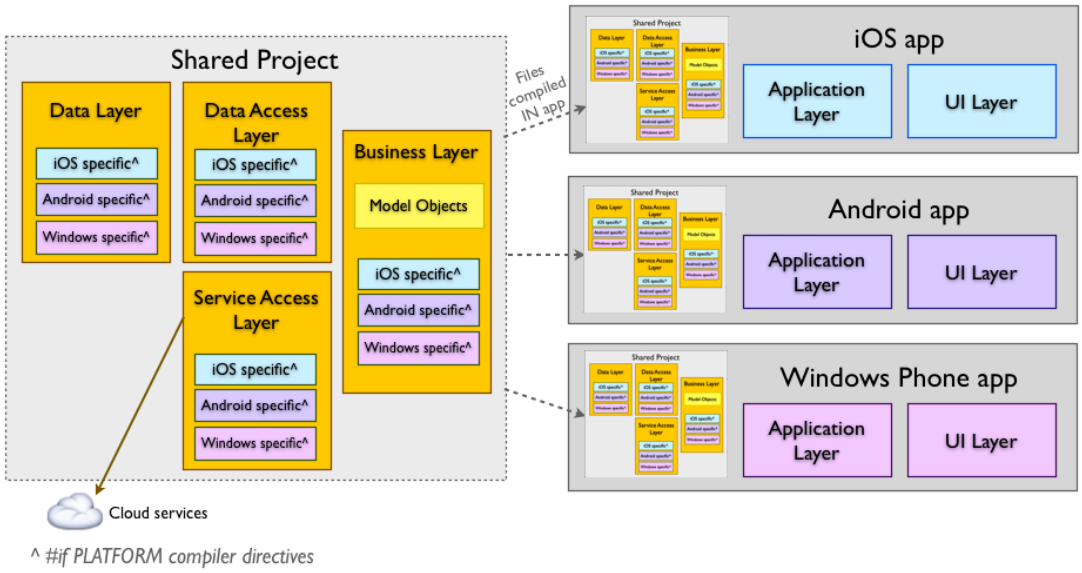

Figure 5 - Shared Asset Project [35]

3.2.2 Portable Class Library

Another strategy is to use Portable Class Library (PCL) and Dependency Injection (DI) for platform-specific functionalities. The diagram in Figure 6 shows the architecture of a cross-platform app using a PCL to share code as dll and Dependency Injection to pass in platform-dependent services. When a particular feature is needed, an instance of e.g. Data Layer, the DI resolver returns a platform dependent implementation.

3.2.2.1 Advantages

Dependency Injection can branch the code. Refactoring will update all affected references. Also, shared library can be compiled and used as DLL, which makes code sharing very centralized.

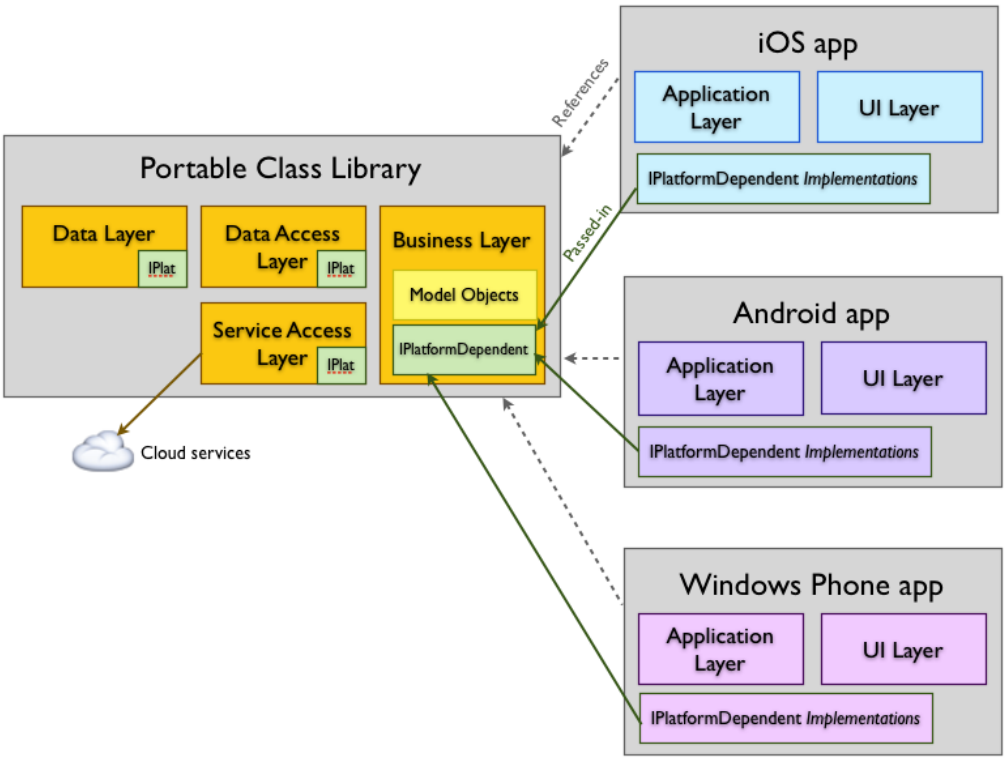

Figure 6 - Portable Class Library [36]

3.2.2.2 Disadvantages

Only subset of .NET is available for PCL, therefore the developer cannot use all classes otherwise available in Mono.Touch and Mono.Android. Because compiler directives cannot be used, platform specific libraries cannot be referenced in shared code. This requires to decompose all platform specific code into services and always use a dependency injection. This forced decomposition can be an advantage for large parts of an application, but e.g. for single line of code it is unnecessary over-engineering.

3.2.3 Combination of Shared Project and Small PCL

Because the Shared project template is quite new, not all of the features are available (e.g. localization files). Because of this limitation, a combination of PCL and Shared project can be used. Most of the code can be put into Share Asset Project, and C# language resource files can be in a small PCL library, leveraging all advantages of both solutions. This method was chosen for the pilot application in this thesis.

3.3 Xamarin.Forms

Another goal is to also increase codeshare (and minimize redundancy) in the UI code. This redundancy is the reason why Forms framework was created by Xamarin. Xamarin.Forms is a cross-platform UI toolkit built on top of the existing Xamarin framework that allows rapid cross-platform development. Basically the developer only needs to create the UI once. It will look native on every platform.

The project can be based on Shared project or PCL. iOS, Android, Windows Phone and lately also Windows 10 [37] are all supported platforms.

3.3.1 Controls Reference

Controls for UI can be defined in XML language XAML or directly using C# code. C# code has better debugging support in Visual Studio, because it is strongly typed[11].

There are four main groups of controls used to create the user interface of a Xamarin.Forms app: Pages, Layouts, Views, and Cells. At runtime, each control is mapped to a native equivalent by calling a platform dependent renderer.

3.3.1.1 Page

Page class is a primary building block of the screen and contains a single child. It corresponds to View Controller on iOS, Activity on Android and Page on Windows Phone. There are 5 types of Pages: ContentPage, MasterDetail, Navigation, Tabbed and Carousel. In Figure 7, there are examples of all Page types.

[11] Strongly typed languages do not have a precise definition, but they are vaguely defined as languages with the presence of type safety and static type checking – this allows to generate type errors during compile time whenever passed type (e.g. into a function) does not match expected type.

ContentPage is a base class for all pages with only some basic features. The MasterDetail page consists of the master page usually containing the navigation menu, and a detail page containing the actual content. The most usual type of page is NavigationPage. It has a floating toolbar on the top (with an optional menu button) to navigate back. For pages with multiple subpages, TabbedPage or CarouselPage can be used.

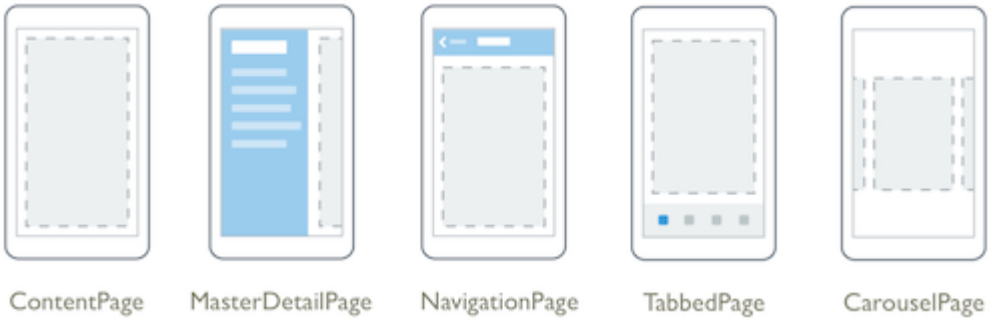

Figure 7 – Main Xamarin.Forms Page types [38]

3.3.1.2 Layout

Layout is used as a container for other controls. It is typically used for positioning and sizing child elements, as shown in Figure 8.

Basic layout type is a StackLayout. It contains logic to set the position of child elements into a vertical or horizontal stack and allows scrolling. An AbsoluteLayout adds absolute position into the elements' properties. Using GridLayout, a table-like alignment can be created. ContentView and the Frame are the basic rectangles for aligning objects. ScrollView can add a scrolling feature into most objects.

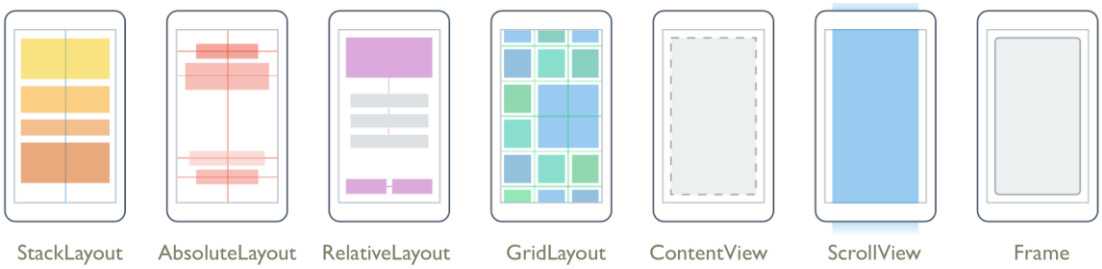

Figure 8 - Xamarin.Forms Layout types [38]

3.3.1.3 View

View is used as the base class of all visual objects such as buttons, labels, and all other controls. The most common example is a ListView. Figure 9 shows a preview of different behavior on all three platforms. It is a bindable[12] list with cell structure defined by Cell class. Bindable property DataSource fills the list with data/content. Layout is a subtype of View.

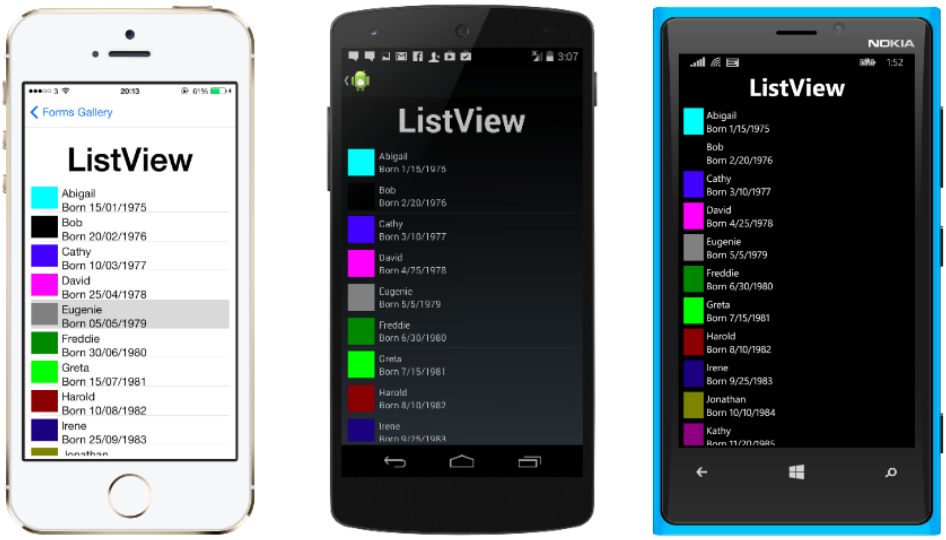

Figure 9 - Xamarin.Forms ListView on different platforms [38]

3.3.1.4 Cell

Cells are specialized elements used for (bindable) child items of tables or lists. They describe how each item of the list will be drawn. It is a template for creating a visual element.

[12] Data bindings allow properties of two objects to be linked. Change of one object causes the change of the other one. It can be defined as two-way binding or only one-way binding.

3.3.2 Native and Custom Control Renderers

Xamarin.Forms components are rendered using native controls on the target platform. This approach allows apps to retain the look that conforms to each platform's design conventions.

The framework also allows developers to create or override custom control renderers to provide flexibility to change control's look and behavior. Each renderer is added to a similar platform project to be compiled only on the target platform. Finally, the renderer is attached to assembly using the `assembly:ExportRenderer` attribute.

3.3.3 Localization

One of the biggest advantages of the Xamarin platform is a mechanism for localization of applications using standard .resx files [39]. The standard .resx resource file consists of XML entries, which specify objects and strings inside XML tags. Visual Studio provides a visual editor of .resx files, but files can also be edited directly as text files. Localization uses standard .NET classes in `System.Resources` and `System.Globalization` and .resx files containing translated strings provide strongly-typed access to translations. Native solutions are usually much worse. E.g. in iOS, English strings in the code are used as keys for all other translations – which is a horrible idea. A change in the English translation needs to be manually fixed in all translations.

3.3.3.1 Platform-Specific Localization

Sometimes it is also required to localize images used by UI. Both Android and iOS applications have Resources directory for translated images, for Windows Phone it is called Assets. The developer still needs to use native translation files for the application name as well.

3.4 Asynchronous Caveats

The developer should avoid performance bottlenecks and enhance the responsiveness of implementation by using asynchronous programming. C# version 5 introduced two new keywords for asynchronous operations: `async` and `await`. These are keywords that allow a simple code using Task Parallel Library [40] to execute long running operations in a different thread and quickly access results on completion. It is designed to leave delicate work to the compiler so the developer can keep a code that still looks synchronous, as shown in Figure 10. When a synchronous call is used, the thread is blocked until the operation completes. When an asynchronous call is used, a different thread is created for a long running task.

The previous paragraph describes an ideal case. In real life when the developer does not use asynchronous calls in C# correctly, he can create race conditions (hard to debug). In my opinion, in asynchronous programming, very thorough testing of the application is essential.

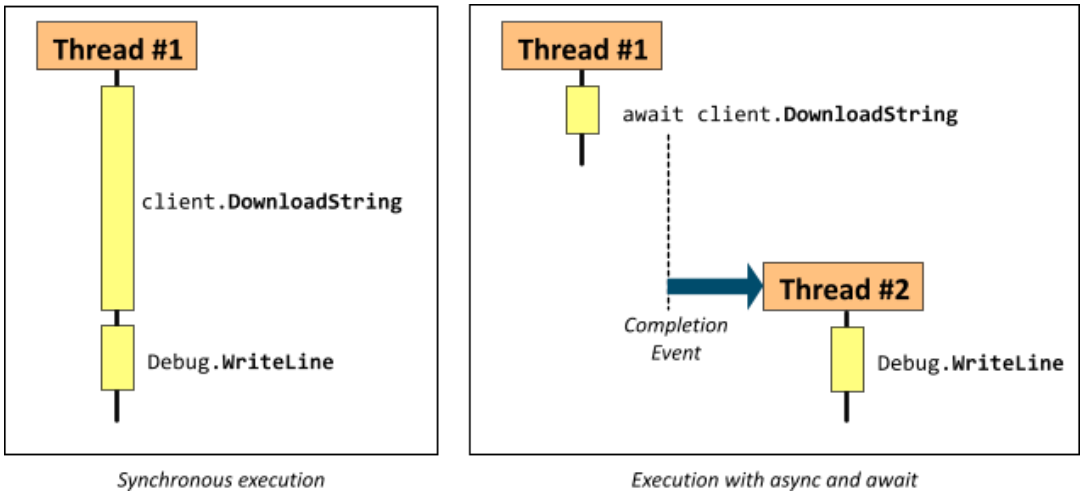

Figure 10 – example of async execution [41]

3.4.1 Xamarin.Forms MessagingCenter

Because of the high level of abstraction introduced with Xamarin.Forms, a new method of communication between different screens of the application was instroduced. Xamarin added a messaging service to send and receive messages. This service also helps with synchronization of some asynchronously running components.

`MessagingCenter` enables communication of components without knowing anything about each other. There are two static methods used with messaging:

- **Subscribe** – Listens to messages of a particular signature, consisting of string name and type of sent object. Multiple listeners can be set for the same signature. I recommend implementing a custom extension called `SubscribeOnce` to prevent unintended multiple subscriptions. The `Unsubscribe` method is also available.
- **Send** – Sends a message with a signature, additional arguments can be passed.

3.5 Memory management

One of the biggest problems of mobile development is a very limited memory. Cell phones have very limited memory, and the developer needs to make sure that it is used effectively. This part is based on the article: Xamarin's best practices [42].

Mono is a garbage collected environment, but effective memory management is not only about releasing memory quickly. The developer should think about loading large objects into memory, e.g. displaying images in full resolution can fill the memory completely. During optimization of cross-platform code, it is an absolute necessity always to measure real memory costs on particular devices, because no emulator will provide an accurate representation of the device.

3.5.1 Garbage collection with Xamarin

Managed languages use garbage collectors to empty or reclaim memory no longer in use. Xamarin's garbage collector is named SGen. It uses three heaps:

- **Nursery** – small heap for new objects
- **Major Heap** – for long-running objects
- **Large Object Space** – for objects larger than 8000 bytes

The biggest advantage of SGen is that it performs small garbage collections on these heaps, and this reduces the impact on application performance, because big garbage collections can freeze the application.

3.5.1.1 Strong and Weak References

To manage objects on the heap, garbage collector needs to determine whether the application has a strong/weak reference to the object. When an app can utilize an object on the heap, then it is a strong reference and the object's memory cannot be reclaimed. On the other hand, when the application does not have a direct reference to the object, it is a weak reference, and these objects can be safely removed by the collector. This clean-up is useful for objects that require a big amount of memory, such as bitmaps.

The diagram in Figure 11 shows the difference between strong and weak references. Blue objects are in direct use by the application, purple ones are not in immediate use. A garbage collector can dispose (purple) weak reference objects and reclaim memory if needed.

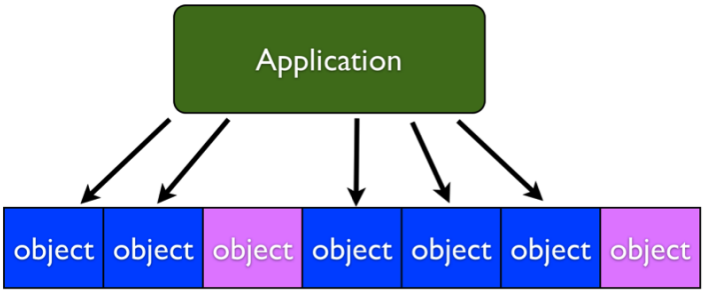

Figure 11 – strong (blue) and weak dependency of memory objects [42]

Every developer should consider disposing of limited resources (e.g. Network connections, large blocks of memory, file descriptors).

Dispose pattern is a design pattern that is used to handle resource clean-up and prevent resource leaks in garbage collected environments. It gives an object a method called Dispose which frees all resources an object holds. In .NET it is done by implementing the `IDisposable` interface. The `IDisposable` pattern can also be used for wrapping an object by `using` statement. An example of this statement is depicted in Listing 2.

```
using (var image = UIImage.FromFile ("image.png")){
    Draw (image);
}
```

Listing 2 - Example of using statement [42]

For example, it is very easy to consume all memory with just a few images because native Image objects contain only 20 bytes of memory, but a part of these 20 bytes contains a pointer to the actual image data, which might be megabytes. But from a garbage collector perspective, only 20 bytes are consumed. In these cases, a Dispose pattern is recommended.

3.5.2 Linker and package size

To reduce the size of an application, Xamarin contains a Linker optimization as part of the build tools. During the build, Linker performs a static analysis of code to determine which types/members are used by the app and removes unused ones.

In Xamarin there are three Linker optimization behaviors: Do not link, Link SDK and Link All.

- Do not Link – no Linker optimization. Default for debugging due to performance reasons.
- Link SDK Only – will reduce the size of Xamarin assemblies only (by removing unused ones), user code will be unaffected.
- Link All Assemblies – aggressive optimization, targets SDK assemblies and reduces the size of the assembly. It can result in consuming less memory.

Link All Assemblies requires more testing of the app because the linker can be too aggressive and remove code called e.g. by reflection (reflection allows class call by its string name). `[Preserve]` attribute is also available for forced preserving of a type (it can be added to the class to prevent linker from removing the class).

Package size can be a problem on platforms that use AOT compiled (iOS, ARM Android) code because bytecode is more compact than native code. Native coding of packages also creates some problems with generics because it requires native

compilation of every type parameter possibility (if linker optimization is not used). The following steps can help with reduction of package size:

- Use linker, if the linker is not set, almost full .NET stack can be compiled
- The package should be built for a single architecture and with disabled Debug flag
- Use smaller LLVM[13] compiler (slower, but produces a smaller package)

3.5.3 Performance

Application UI should be independent of background tasks. UI lags are problematic for mobile devices because users consider them as bugs. Therefore, heavy computation tasks should always be queued as background operations. When a process completes, it can notify the user interface thread using an asynchronous callback. Android and iOS do not provide thread-safe API. This problem means that user interface should be invoked only on the UI thread (this also includes Xamarin.Forms messages). The developer should consider the possibility to cancel background operations in UI, for long running tasks or when changing context (e.g. hitting the back button). Xamarin recommends using the Task Parallel Library for asynchronous calls.

3.5.4 Memory leaks

Some of Xamarin.Forms controls are more prone to create memory leaks than others. E.g. Frame, if it is used in ListView, can get a little “leaky”. The current version of the Xamarin framework can contain hidden bugs, and the only way to avoid them is thorough testing of the application on every platform.

[13] LLVM is a higly optimised compiler now also used in iOS

3.6 Summary

This chapter provides a lot of recommendations I learned during the development of the pilot application. It can help new developers with starting their project. Particular attention should be paid to parts 3.2 Code Sharing and 3.5 Memory management because these topics are the fundamentals of mobile development.

4 Pilot Application

Even in small companies today, it is more and more common to use a project management tool of some kind to track projects or tasks. This tracking enhances productivity and also enables more accurate invoicing. The usual productivity of web-based tools, such as Redmine [43] or TFS [44], was created for IT companies and has too many unnecessary features. Too many features can be counterproductive for mobile applications, and I think that is the reason why mobile apps for Redmine or TFS are virtually non-existent.

The goal of the pilot application was to create a mobile frontend to an online productivity tool called "Kraftil.cz" [45] (using initially Redmine as a backend). The application is designed to operate on smartphones running iOS, Android, or Windows Phone. The user can document the progress of projects and tasks. The manager can assign tasks to employees or colleagues and monitor the progress of work in real time. This progress tracking can also help the planning of operations. Content created by progress tracking can also be used as a company's web page content.

4.1 Application Design

Data layer called `Remap.Data` uses very basic ORM[14] SQLite and model classes in `Remap.Models`. The presentation layer is represented by `Remap.Views` and `Remap.ViewModels`, adhering to the modern MVVM design pattern [46]. Figure 12 shows the deployment diagram of an application.

MVVM is an acronym for the Model-View-ViewModel design pattern. Model is a class with limited functionality, usually populated from a database or web service. The logic of the application is in ViewModel. It notifies View of changes in Model and

[14] Object-relational mapping is a programming technique for converting data between a relational database and an object-oriented programming language.

ensures data consistency. At the front end, there is a View. It describes the layout and displays data from the ViewModel.

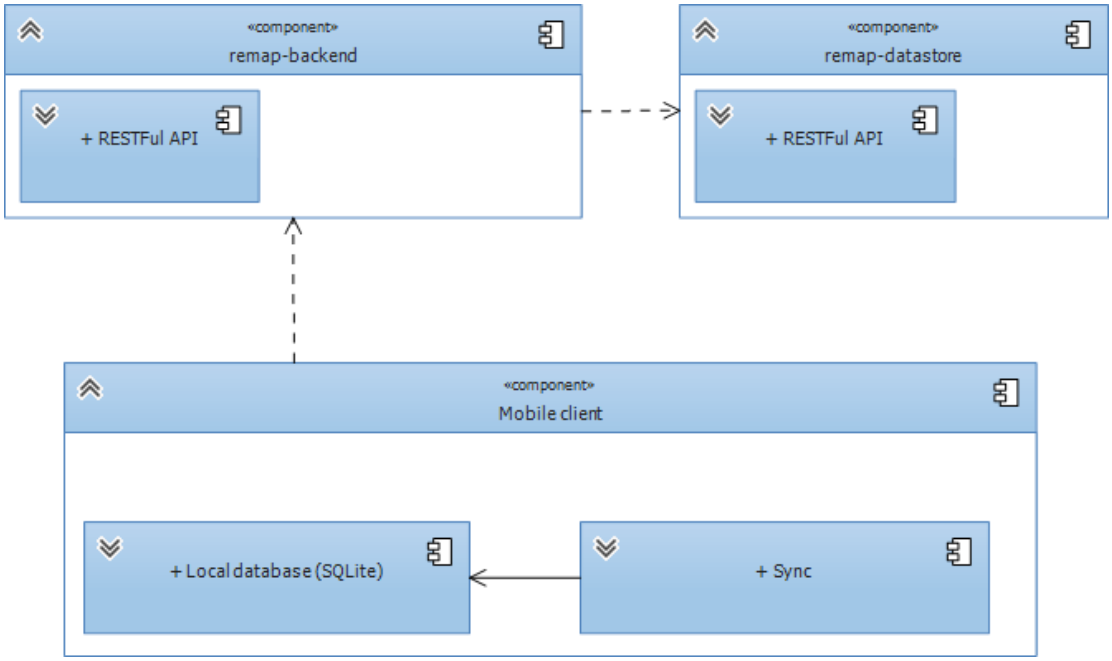

Figure 12 - Deployment diagram of complete client-server architecture

Implementation of MVVM design pattern is shown in Figure 13. Views use bidirectional data binding to show data from ViewModels and ViewModels reads data from Data layer using Models. Data layer uses synchronization to maintain data consistency between the SQLite database and a remote web service accessed through class RemapProxy.

The backend is an HTTP server with a RESTful[15] API and the application saves downloaded data into a local data store on the SQLite file. Therefore the app can work with rare connection outtakes and minimizes the amount of transferred data. Server backend is not a part of this thesis.

[15] Representational state transfer (or REST) [66] is a state-less architectonic style of web applications. It allows to create a web interface for the server backend using simple HTTP verbs (GET, POST, PUT, DELETE, etc.). Systems that conform to this style are called RESTful.

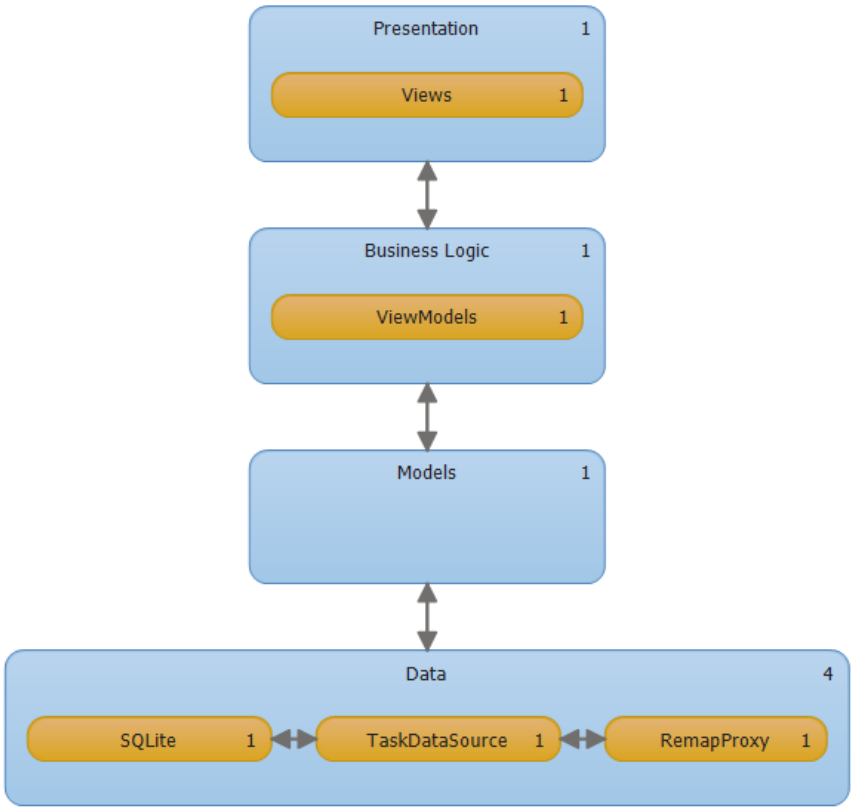

Figure 13 - Layer diagram of MVVM design pattern implementation

4.1.1 Synchronization of local database with API server

Synchronization was one of the fundamental features of the application. In a mobile application, a developer cannot always rely on a 100% working Internet connection. A local SQLite database was used with additional local unique database keys. This design means that database rows in the application have both local keys and remote keys. When the connection is lost, an item can be still saved into the local database using the local unique key.

The main purpose of mobile application's database is to cache server queries putting a lower load on a mobile connection. SQLite ORM [47] was used to minimize the usage of hard-coded SQL queries. Later, for reducing the number of requests, batch saving was introduced. The application waits a few seconds for all changes created by the user and then it saves all changes in one batch request to a server.

4.1.1.1 API Backend Notes

Server API is not part of the thesis, and it was created by a different team in Java. It allows storing all smartphone app data on the server, and it was built on top of a Redmine [43] project management application written in Ruby on Rails. This backend choice was proven as wrong because of limitations of the backend system. It started to limit the development of more sophisticated features very quickly (e.g. file management). In the future, it is planned to replace the Redmine backend entirely.

4.1.2 Deployment

An app marketplace is a type of a digital distribution platform for mobile devices. Marketplaces are typically online stores where users can find desired content. Many app stores require some (usually automatic) approval process.

Apple iOS applications are deployed to the AppStore [48]. Android has Google Play [49], and the WP store is called Windows Phone Store [50]. All deployment stores are well documented and relatively easy to use. Automatic deployment/signing scripts were created in this thesis for speeding up the delivery of new versions. Appendix B contains a detailed description of automatic Android deployment.

4.1.3 Used Libraries

There are a lot of choices to make when starting a new project. The development community moves really fast, and new libraries appear every day. The following list can help the new developer with those decisions.

4.1.3.1 SQLite.NET ORM

Object Relational Mapping allows the developer to save and retrieve objects from a database without writing SQL statements. SQLite.NET [47] is a very basic ORM library recommended by Xamarin for data storage for cross-platform apps.

4.1.3.2 XLabs.Forms

XLabs.Forms [51] is an open source project that consists of controls and services for Xamarin.Forms projects. It contains:

- IOC and a view factory
- Application services for hardware interaction
- New controls, e.g. calendar, image button, checkbox

The problem is that it contains a lot of bugs, and development is quite slow now. However, some of the controls are still unique, and better replacement does not exist. A relatively new alternative to XLabs.Forms is discussed in the next section.

4.1.3.3 Acr

An Acr [52] library is an alternative to XLabs.Forms with more reliable Camera services and Dialog service. In a future version of the pilot application, there is a plan to replace more parts of XLabs.Forms with features from this library.

4.1.3.4 RestSharp

RestSharp [53] is a very powerful strongly typed REST and HTTP API Client. During development of the practical part of this thesis, some GitHub pull requests were created for enhancement of this library. Also coding using RestSharp is much shorter and more self-explaining than using the Microsoft's `System.Net.Http.HttpClient`. There is also an alternative to RestSharp called ServiceStack, but RestSharp is easier to use.

4.1.3.5 MR.Gestures

Implementing all hand gestures for Xamarin.Forms from scratch can be a very long process; so I decided to use the MR.Gestures [54] library.

4.1.3.6 Xamarin Insights

Insights [55] is a library from Xamarin that automatically sends crash reports to an online bug tracker; this is a very useful library.

Deployment on different devices can produce bugs hidden to the developer. By automatic sending of bug reports to Insight's web pages, every bug can be discovered. Insights send the whole call-stack, device info, and user defined values.

4.2 User Experience Design

The user interface (UI) is a critical feature for smartphone applications. Users usually give the app a few minutes and if they don't understand it, they will never use it. Well prepared and tested user interface can make a big difference.

For Android, it was chosen to deliver the user interface in a new Material Design [56]. Material Design is a design language developed by Google. It was first used in a new version of personal assistant Google Now [57]. The design resembles natural surfaces and edges providing meaning about what the user can touch. This design is the default design for Android 5 applications. For older versions of Android, there was an AppCompat v21 [58] library created by Google allowing the usage of Material Design and a feature set from Android 5 Lollipop. For C#, there was an open-source implementation called `NativeCode.Mobile.AppCompat` [59]. This library was made obsolete since Xamarin.Forms 1.5 was released, because it now supports AppCompat Material Theme natively.

Material design introduced a lot of new features. One of the most popular is the "Floating action button" (FAB). It is used for the most promoted action (usually the add button). FABs are circle icons floating above the UI and have motion behaviors that include morphing, launching, and a transferring anchor point. In applications, they are used for most used actions on every page, so the user does not need to find them in menus. In Figure 14, an excerpt from an original UI mock up is depicted.

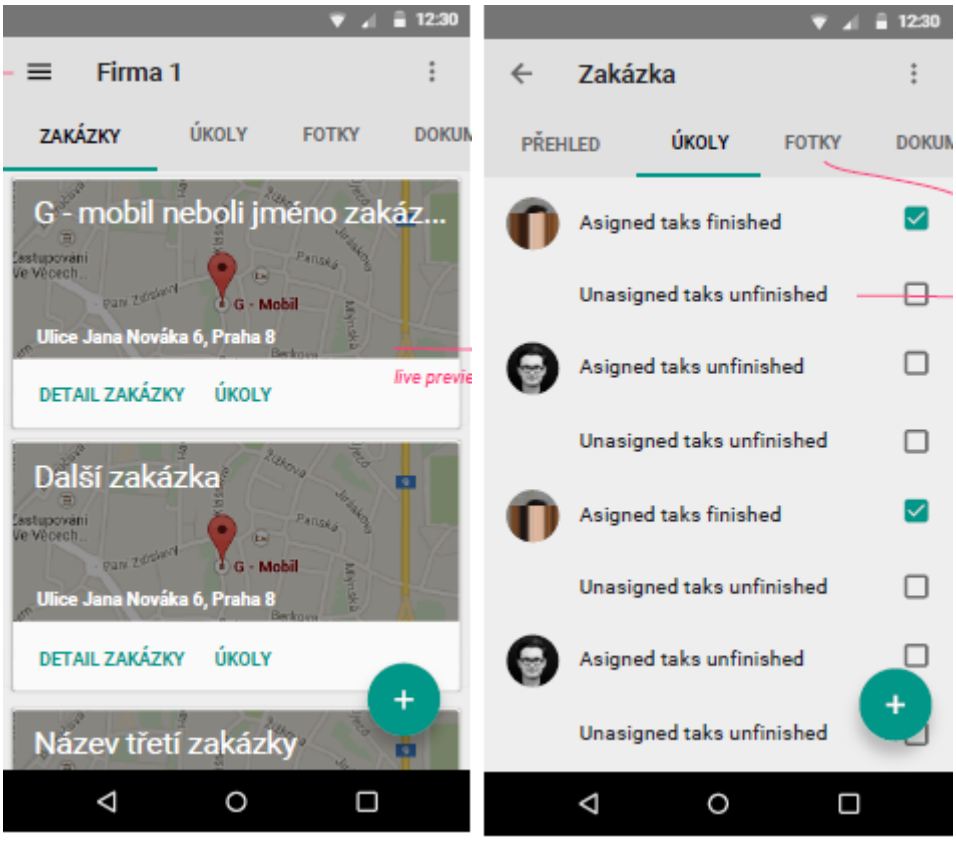

Figure 14 - Mockup of user interface in Android's Material design

Another interesting component is CardView. Cards are a practical means of displaying content composed of different elements. They're also well-suited for showcasing elements whose size or supported actions vary, such as photos with captions of variable length. They are used as base objects for showing information about a project. An example of such project cards is shown on the left side of Figure 14.

4.3 Implementation Details

Development started with a minimal set of features, and new items were added by feedback from first users. In the last part of the development, there was a stabilization phase – testing and fixing of hidden bugs.

4.3.1 Minimal Feature Set

The application was designed as a project management application for small companies, displaying all of the progress tracking in one place. It should allow to control workers through the mobile app and to facilitate management of contracts,

tasks, and documents. Results can be used to simplify the reporting of costs (e.g. by photographing receipts of purchase). These features were the original ones for the first release in 2015.

Content created in the application can be used to create a company website with a simple presentation of completed projects. It would allow creating a website without any programming knowledge. The application also contains basic website editing tools (as a hybrid web view). These additional features were not part of the thesis.

4.3.1.1 Project Roadmap

Development was conducted in 12 Sprints, each 14 days long. Each Sprint began by planning the session and in the middle of each Sprint there was one more control meeting. This methodology is based on the management of agile development planning.

4.3.1.2 Implementation Notes

User login was implemented using Google services [60] to allow the usage of user's Google accounts. As an alternative (and platform-independent) option, it was permitted to log in through one-off generated codes sent to email. A list of active projects (task groups) and tasks was created with the complete CRUD (Create, Update, and Delete) interface. Tasks were allowed to have image attachments.

User feedback later drove adding new features. These items include an image gallery, comments, and a journal of all changes done on a task.

Debugging was easiest on Windows Phone devices, because of the high speed of deployment. Android devices were a little slower, but they still remain my most favorite platform because of good documentation. Debugging was most difficult on iOS devices; they require build server on iMac connected to Visual Studio through the network. The mobile device for deployment must be also connected to an iMac computer. For development, I would recommend acquiring a small Apple computer such as Mac mini [61].

5 Conclusion

This thesis certainly does not cover every aspect of multiplatform development, but I think it can help starting developers. Regular web or desktop development can be very different compared to smartphone development. Limited memory, need for a precise user interface, debug, and lower tolerance for bugs can make development more challenging.

From my point of view, application UI design was the hardest part. User interface was refactored about three times. The biggest problem was debugging on different devices. An application behaves differently on an emulator and the device. This behavior also differs on the same platform but a different hardware device (for example on Android devices). Also, compiling and deployment for debugging on devices is slower than x86 development, which I have more experience with.

5.1 Goal Fulfilment

The main objectives of section 1.2 that were fulfilled in this thesis are:

- Analysis of existing frameworks. It contains my experience with existing multiplatform frameworks and reasons why I chose the Xamarin framework for the pilot application.
- Methodology of development. It describes caveats I encountered when I was working with the Xamarin framework.
- Application design and development. It is a documentation of the development of the pilot app.

There are some known issues. Sometimes the “hamburger” button (for displaying master-menu) on the main master-detail page disappears. I identified it as a framework bug and reported it to Xamarin (no workaround found). The color scheme of Windows Phone sometimes resembles the Android application too much, since the Android application was created first. Almost all unexpected errors are automatically documented. If an application crashes, it will send a bug report to Xamarin Insights with the whole call stack.

5.2 Future Work

In the future, there is a plan to implement offline file management, which can delay upload in case the Internet is not accessible. This control will allow the app to work entirely offline.

The Android application is now deployed in open beta at Android Store. In the future, hopefully, the app will fulfill very high requirements of production code and will also be deployed publically on Windows Store and Apple AppStore.

References

[1] **Farago, Peter.** *iOS and Android Adoption Explodes Internationally.* [Online] Flurry.com, 2014. [Cited: 4 13, 2015.] http://www.flurry.com/bid/88867/iOS-and-Android-Adoption-Explodes-Internationally.

[2] **Ballvé, Marcelo.** Smartphones Are Outselling PCs Two-To-One. *Business Insider Australia.* [Online] 2012. [Cited: 4 13, 2015.] http://www.businessinsider.com.au/smartphones-outpacing-pcs-two-to-one-2012-11.

[3] **Gartner.** Two-Thirds Of Enterprises Will Adopt Mobile Device Management Solution . *CRN.* [Online] 2012. [Cited: 4 13, 2015.] http://www.crn.in/crn/news/292860/gartner-thirds-enterprises-adopt-mobile-device-management-solution.

[4] **Olanoff, Drew.** Facebook's Monthly Active Users Up 23% to 1.11B; Daily Users Up 26% To 665M; Mobile MAUs Up 54% To 751M. *TechCrunch.* [Online] 2015. [Cited: 7 14, 2015.] http://techcrunch.com/2013/05/01/facebook-sees-26-year-over-year-growth-in-daus-23-in-maus-mobile-54/.

[5] **Holwerda, Thom.** The second operating system hiding in every mobile phone. *OSNews.* [Online] 11 12, 2013. http://www.osnews.com/story/27416/The_second_operating_system_hiding_in_every_mobile_phone.

[6] **Manjoo, Farhad.** A Murky Road Ahead for Android, Despite Market Dominance. *The New York Times.* [Online] 5 27, 2015. [Cited: 5 27, 2015.] http://www.nytimes.com/2015/05/28/technology/personaltech/a-murky-road-ahead-for-android-despite-market-dominance.html?_r=0. ISSN 0362-4331.

[7] **CyanogenMod open-source community.** CyanogenMod | Android Community Operating System. *Cyanogenmod.org.* [Online] 2015. [Cited: 11 15, 2015.] http://www.cyanogenmod.org/.

[8] **Netmarketshare.com.** Market share for mobile, browsers, operating systems and search engines | NetMarketShare. *Netmarketshare.com.* [Online] [Cited: 7 14, 2015.] https://netmarketshare.com/.

[9] **Apple.** Swift - Apple Developer. *Developer.apple.com.* [Online] 2015. [Cited: 10 12, 2015.] https://developer.apple.com/swift/.

[10] —. Xcode - IDE - Apple Developer. *Developer.apple.com.* [Online] 2015. [Cited: 10 5, 2015.] https://developer.apple.com/xcode/ide/.

[11] **Google.** Dalvik bytecode | Android Open Source Project. *Source.android.com.* [Online] 2015. [Cited: 9 1, 2015.] https://source.android.com/devices/tech/dalvik/dalvik-bytecode.html.

[12] **Microsoft.** Microsoft Silverlight. *Microsoft.com.* [Online] 2015. [Cited: 9 4, 2015.] https://www.microsoft.com/silverlight/.

[13] —. Windows App Studio–Free Tool to create apps in Windows Stores. *Appstudio.windows.com.* [Online] 2015. [Cited: 9 8, 2015.] http://appstudio.windows.com.

[14] —. Windows Dev Center. *Dev.windows.com.* [Online] 2015. [Cited: 9 4, 2015.] https://dev.windows.com/.

[15] **Apache.** Apache Cordova. *Cordova.apache.org.* [Online] 2015. [Cited: 10 1, 2015.] https://cordova.apache.org/.

[16] **Semtech Solutions Ltd.** An introduction to Apache Cordova. [Online] 2 26, 2014. [Cited: 10 2, 2015.] http://www.slideshare.net/mikejf12/an-introduction-to-apache-cordova.

[17] **Adobe Systems.** PhoneGap. *Phonegap.com.* [Online] 2015. [Cited: 10 17, 2015.] http://phonegap.com/.

[18] **Microsoft.** Visual Studio Tools for Apache Cordova. *Visualstudio.com.* [Online] 2015. [Cited: 10 19, 2015.] https://www.visualstudio.com/cs-cz/features/cordova-vs.aspx.

[19] —. Mobile App Service | Microsoft Azure. *Azure.microsoft.com.* [Online] 2015. [Cited: 11 2, 2015.] https://azure.microsoft.com/en-us/services/app-service/mobile/.

[20] **Adobe Systems.** Creating apps with PhoneGap: Lessons learned. *Adobe Systems.* [Online] 09 17, 2012. http://www.adobe.com/devnet/phonegap/articles/creating-apps-with-phonegap-lessons.html.

[21] **Ford, Kevin.** Mobile Development Platform Performance (Native, Cordova, Classic Xamarin, Xamarin.Forms). [Online] 12 23, 2014. http://windingroadway.blogspot.it/2014/12/mobile-development-platform-performance.html.

[22] **McCarty, Brad.** Mark Zuckerberg Discusses Facebook's IPO Regrets and Mistakes. *The Next Web.* [Online] 2012. [Cited: 4 13, 2015.] http://thenextweb.com/facebook/2012/09/11/zuckerberg-the-performance-stock-obviously-disappointing/.

[23] **Xamarin.** Mobile Application Development to Build Apps in C# - Xamarin. *Xamarin.com.* [Online] 2015. [Cited: 9 12, 2015.] https://xamarin.com/platform.

[24] **Ford, Kevin.** Choosing The Best Mobile Technology (White Paper). [Online] 11 2014. http://magenic.com/Portals/0/Magenic-White-Paper-Choosing-the-Right-Mobile-Technology.pdf.

[25] **Xamarin.** Xamarin White Paper - Key Strategies for Mobile Excellence. *Xamarin.* [Online] [Cited: 7 14, 2015.] http://cdn1.xamarin.com/webimages/assets/Xamarin_Whitepaper-Key_Strategies_for_Mobile_Excellence.pdf.

[26] **Microsoft.** aspnet/Razor. *GitHub.* [Online] 2015. [Cited: 10 14, 2015.] https://github.com/aspnet/Razor.

[27] **Icaza, Miguel de.** Android Ported to C#. [Online] 5 1, 2012. blog.xamarin.com/android-in-c-sharp/.

[28] **Bogatov, Egor.** Xamarin.Android vs Dalvik (or ART) test. [Online] https://github.com/EgorBo/Xamarin.Android-vs-Java.

[29] **Debian.** n-body description (32-bit Ubuntu one core) | Computer Language Benchmarks Game . *Benchmarksgame.alioth.debian.org.* [Online] 2015. [Cited: 10 4, 2015.] http://benchmarksgame.alioth.debian.org/u32/nbody-description.html.

[30] **Telerik.** Telerik Platform. [Online] http://www.telerik.com/platform.

[31] **Mono.** Home | Mono. *Mono-project.com.* [Online] 2015. [Cited: 11 4, 2015.] http://www.mono-project.com/.

[32] **Xamarin.** Available Assemblies - Xamarin iOS. *Developer.xamarin.com.* [Online] 2015. [Cited: 10 7, 2015.] https://developer.xamarin.com/guides/ios/under_the_hood/assemblies/.

[33] —. Assemblies - Xamarin Android. *Developer.xamarin.com.* [Online] 2015. [Cited: 10 4, 2015.] https://developer.xamarin.com/guides/android/under_the_hood/assemblies/.

[34] **Fowler, Martin.** Inversion of Control Containers and the Dependency Injection pattern. *martinfowler.com.* [Online] 2004. [Cited: 11 1, 2015.] http://martinfowler.com/articles/injection.html.

[35] **Xamarin.** Sharing Code Options - Xamarin. *Developer.xamarin.com.* [Online] 2015. [Cited: 4 12, 2015.] http://developer.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/sharing_code_options/.

[36] —. Introduction to Portable Class Libraries - Xamarin. *Developer.xamarin.com.* [Online] 2015. [Cited: 4 12, 2015.] http://developer.xamarin.com/guides/cross-platform/application_fundamentals/pcl/introduction_to_portable_class_libraries/.

[37] —. Xamarin.Forms for Windows Preview. *Developer.xamarin.com.* [Online] 2015. [Cited: 4 13, 2015.]

[38] —. Xamarin.Forms Controls Reference - Xamarin. *Developer.xamarin.com.* [Online] 2015. [Cited: 7 15, 2015.] http://developer.xamarin.com/guides/cross-platform/xamarin-forms/controls/.

[39] **Microsoft.** Resources in .Resx File Format. *Msdn.microsoft.com.* [Online] 2015. [Cited: 10 12, 2015.] https://msdn.microsoft.com/cs-cz/library/ekyft91f%28v=vs.90%29.aspx.

[40] —. Task Parallel Library (TPL). *Msdn.microsoft.com.* [Online] 2015. [Cited: 10 2, 2015.] https://msdn.microsoft.com/en-us/library/dd460717%28v=vs.110%29.aspx.

[41] **Nokia.** Asynchronous Programming For Windows Phone 8 . [Online] [Cited: 1 9, 2015.] developer.nokia.com/community/wiki/Asynchronous_Programming_For_Windows_Phone_8.

[42] **Xamarin.** Memory and Performance Best Practices - Xamarin. *Developer.xamarin.com.* [Online] 2015. [Cited: 4 9, 2015.] http://developer.xamarin.com/guides/cross-platform/deployment,_testing,_and_metrics/memory_perf_best_practices/.

[43] **Lang, Jean-Philippe.** Overview - Redmine. *Redmine.org.* [Online] 2015. [Cited: 11 2, 2015.] http://www.redmine.org/.

[44] **Microsoft.** Team Foundation Server | Visual Studio. *Visualstudio.com.* [Online] 2015. [Cited: 11 13, 2015.] https://www.visualstudio.com/products/tfs-overview-vs.

[45] **Zen systems s.r.o.** Kraftil.cz - Web cards for self-employed people and small firms. *Kraftil.cz.* [Online] 2015. [Cited: 11 2, 2015.] https://kraftil.cz/?lang=en.

[46] **Microsoft.** The MVVM Pattern. *Msdn.microsoft.com.* [Online] 2015. [Cited: 11 4, 2015.] https://msdn.microsoft.com/en-us/library/hh848246.aspx.

[47] **Xamarin.** Part 3 - Using SQLite.NET ORM - Xamarin. *Developer.xamarin.com.* [Online] 2015. [Cited: 11 25, 2015.] https://developer.xamarin.com/guides/cross-platform/application_fundamentals/data/part_3_using_sqlite_orm/.

[48] **Apple.** App Store. *Apple.com.* [Online] 2015. [Cited: 11 2, 2015.] www.apple.com/appstore.

[49] **Google.** Google Play. *Play.google.com.* [Online] 2015. [Cited: 11 2, 2015.] https://play.google.com/.

[50] **Microsoft.** Windows Phone Apps - Microsoft Store. *Windowsphone.com.* [Online] 2015. [Cited: 11 2, 2015.] http://www.windowsphone.com/store.

[51] **XLabs.** XLabs/Xamarin-Forms-Labs. *GitHub.* [Online] 2015. [Cited: 11 12, 2015.] https://github.com/XLabs/Xamarin-Forms-Labs.

[52] **Ritchie, Allan.** aritchie/userdialogs. *GitHub.* [Online] 2015. [Cited: 11 2, 2015.] https://github.com/aritchie/userdialogs.

[53] **Restsharp.org.** RestSharp - Simple REST and HTTP Client for .NET. *Restsharp.org.* [Online] 2015. [Cited: 11 2, 2015.] http://restsharp.org/.

[54] **Rumpler, Michael.** MR.Gestures - Handle all the touch gestures in your Xamarin.Forms mobile apps. *Mrgestures.com.* [Online] 2015. [Cited: 11 12, 2015.] http://www.mrgestures.com/.

[55] **Xamarin.** Mobile App Tracking, Monitoring, and Crash Reporting — Xamarin Insights. *Xamarin.com.* [Online] 2015. [Cited: 11 13, 2015.] https://xamarin.com/insights.

[56] **Google.** Introduction - Material design - Google design guidelines. *Google design guidelines.* [Online] 2015. [Cited: 11 4, 2015.] http://www.google.com/design/spec/material-design/introduction.html.

[57] —. Google now. *Google.com.* [Online] 2015. [Cited: 10 2, 2015.] http://www.google.com/landing/now/.

[58] —. Maintaining Compatibility | Android Developers. *Developer.android.com.* [Online] 2015. [Cited: 11 2, 2015.] http://developer.android.com/training/material/compatibility.html.

[59] **Pham, Mike.** nativecode-dev/oss-xamarin. *GitHub.* [Online] 2015. [Cited: 10 2, 2015.] https://github.com/nativecode-dev/oss-xamarin.

[60] **Google.** Google Services | Android Developers. *Developer.android.com.* [Online] 2015. [Cited: 11 1, 2015.]

[61] **Apple.** Mac mini - Apple. *Apple.* [Online] 2015. [Cited: 11 4, 2015.] http://www.apple.com/mac-mini/.

[62] **Google.** Google Play Developer API. *Google Developers.* [Online] 2015. [Cited: 4 9, 2015.] https://developers.google.com/android-publisher/.

[63] —. Google.Apis.AndroidPublisher.v2 Client Library. *Nuget.org.* [Online] 2015. [Cited: 4 9, 2015.] https://www.nuget.org/packages/Google.Apis.AndroidPublisher.v2/.

[64] **Cleary, Stephen.** Async/Await - Best Practices in Asynchronous Programming. *MSDN Magazine.* 2013, 3.

[65] —. *Concurrency in C# Cookbook.* s.l. : O'Reilly Media, Inc., 2014. 1449367569, 9781449367565.

[66] **Lefevers, Jason.** A Walkthrough the History of the Metro UI. *Windows Phone Metro.* [Online] 9 15, 2013. [Cited: 10 1, 2015.]

https://windowsphonemetro.wordpress.com/2011/07/11/a-walkthrough-the-history-of-the-metro-ui/.

[67] **Fielding, Roy Thomas.** Chapter 5: Representational State Transfer (REST). *Architectural Styles and the Design of Network-based Software Architectures.* Irvine : University of California, 2000.

Appendix A – Contents of Online Attachment File

An attached file has the following structure:

Sources (folder) – source codes of Pilot Application

- Remap
  - Remap.Architecture – sources of architecture diagrams
  - Remap.Common – sources of shared BCL library
  - packages – NuGet packages
  - Remap – sources of the actual pilot application
    - Remap – shared project
    - Remap.Android
    - Remap.iOS
    - Remap.WinPhone
- RestSharp – sources of RestSharp library from GitHub

Deploy (folder) – deploy packages for all 3 platforms

- Android (.apk package) – also deployed as public beta on Google Play:
  - https://play.google.com/store/apps/details?id=zensys.remap
- Windows Phone 8.1 (.xap package) – beta in certification process on Windows Phone Store (as 'Kraftil'), it is also possible to install from SD card:
  - http://www.windowsphone.com/en-us/how-to/wp8/apps/how-do-i-install-apps-from-an-sd-card
- iOS (.ipa package) – binary package

Appendix B – Android, automatic package upload to Google Play

There is a Google Play Developer API for managing apks on Google Play. This appendix is a small example showing how to build a .NET command line application to automatically upload a new version of an apk to Google Play. It uses Google Play Developer API [62] and AndroidPublisher NuGet package [63].

In Google Developers Console:

- register project
- enable Google Play Android Developer API for the project
- create a Service account and download .p12 certificate, get a certificate pass and Service account email

In Google Play Developer Console:

- Give the Service Account email necessary rights via “Settings / User Accounts & rights.”

The command line application is quite straightforward; it is based on Google's Java example:

- it authorizes itself on AndroidPublisherService using service account certificate
- creates a new Edit
- using this Edit, it uploads the new apk
- using the apk version code, it assigns the apk to beta and adds the message to changes
- then it commits all changes to Edit

The deploy manager can run this application like this:

```
ApkUploader.exe "* bug fixes."
```

The C# code of ApkUploader:

```
// Load the certificate.
var certificate = new X509Certificate2(
    Path.GetDirectoryName(Assembly.GetEntryAssembly().Location) +
certPath,
    certPass,
    X509KeyStorageFlags.Exportable);

// Create the ServiceAccountCredential.
var serviceAccountCredential = new ServiceAccountCredential(
    new ServiceAccountCredential.Initializer(serviceAccountEmail)
    {
        Scopes = new[] {
AndroidPublisherService.Scope.Androidpublisher }
    }.FromCertificate(certificate));

// Create the AndroidPublisherService.
var androidPublisherService = new AndroidPublisherService(new
BaseClientService.Initializer
{
    HttpClientInitializer = serviceAccountCredential,
    ApplicationName = "ApkUploader"
});

// Create a new edit to make changes to your listing.
var edit = androidPublisherService.Edits.Insert(null /** no content
*/, packageName).Execute();

// Upload new apk to developer console
var upload = androidPublisherService.Edits.Apks.Upload(
    packageName,
    edit.Id,
    new FileStream(
        apkPath,
        FileMode.Open),
    "application/vnd.android.package-archive"
    );
upload.ResponseReceived += (apk) =>
{
    if (apk == null)
        return;

    // Assign apk to beta track.
    var apkVersionCodes = new List<int?> { apk.VersionCode };
    var updatedTrack =
        androidPublisherService.Edits.Tracks.Update(new Track() {
VersionCodes = apkVersionCodes }, packageName, edit.Id,

EditsResource.TracksResource.UpdateRequest.TrackEnum.Beta).Execute()
;

    // Update recent changes field in apk listing.
    var newApkListing = new ApkListing { RecentChanges =
recentChanges };
    androidPublisherService.Edits.Apklistings.Update(newApkListing,
packageName, edit.Id, apk.VersionCode.GetValueOrDefault(), "cs-
CZ").Execute();
```

```
    // Commit changes for edit.
    var commitRequest =
androidPublisherService.Edits.Commit(packageName, edit.Id);
    var appEdit = commitRequest.Execute();
};
var result = upload.Upload();
```