**FH JOANNEUM**

**University of Applied Sciences**

# Cross Platform Development

# Possibilities and drawbacks of the Xamarin platform

**DI (FH) Norbert Haberl**

24. August 2015

# Table of Content

# Figures

# Tables

## Abstract Deutsch

Mobile Endgeräte werden ein zunehmend wichtiger Faktor - nicht nur in wirtschaftlicher Sicht, sondern auch in einer gesellschaftlichen Perspektive. Die Anzahl der Geräte nimmt verstärkt zu und erreicht mittlerweile nahezu jede Gesellschaftsschicht auf der Welt. Das ist mitunter ein Grund, warum mobile Applikationsentwicklung immer wichtiger wird. Um die Handhabung mehrere Plattformen zu erleichtern, hat man das Konzept von Crossplatform-Entwicklung eingeführt. Diese Arbeit beschäftigt sich mit den Konzepten und Ideen, die hinter diesem Ansatz stehen, wobei der Hauptfokus auf den Möglichkeiten liegt, die die Entwicklungsplattform Xamarin bietet, sowie auf den damit verbunden Einschränkungen.

# Abstract English

Mobile devices are becoming a more and more important factor - not only in an economic perspective but as well in a cultural perspective. Device numbers are increasing and the market penetration already reaches almost every level of society in the world. This is why the importance of mobile development is also increasing. To ease the handling of multiple platforms the concept of cross platform development was introduced. In this paper the detailed reasons for cross platform development are outlined and the possibilities and drawbacks of different approaches will be explained. The main focus lies on Xamarin as an example of a cross-compiled native application and its possibilities and drawbacks.

# 1 Introduction

Mobile applications are becoming a more and more integrated part in business models. What was a homepage in terms of state of the art company representation and the web shop in terms of e-commerce years ago is now the availability of a mobile application.

Smartphones outnumber traditional PCs and it is said that in future mobile devices phones and tablets will be the number one choice to access information in the internet. This is the reason why business models already now and especially in the future will depend on mobile applications that are available for all major platforms.

This paper aims to give an insight about current numbers and forecasts of the general smartphone industry sales numbers. Having a look at the Gartner hype cycle of mobile app development, the trends that are supposed to take place in this environment are shown and critically analyzed.

A goal of this paper is to introduce the key facts on mobile application development. Mobile application development is a diverse environment. It contains at least three target platforms: Android OS, iOS and Windows phone. Each of these platforms requires specific languages and tools to develop applications for them. As all three platforms are independent systems, it is challenging to share and reuse codes across them. Not only that there are usually three codebases necessary,but there are quite likely three different developers necessary to do so.This makes the development across multiple platforms time and cost intensive.

These are the drivers for cross platform development tools that aim to solve or at least to reduce the drawbacks that occur with the need to support multiple platforms. As this is a challenging field there are various possible approaches of how to overcome them. This paper describes some of the approaches and introduces an overview of possible technology stacks and tools.

The tool with the main focus on is Xamarin as an example of a cross compiled multiplatform tool that allows native user experience combined with a high code reuse percentage.

# 2 Definitions

The following section introduces the main terms which needto be clarified to gain a common understanding on how the terms will be used and which meaning is implied by them.

### 2.1.1 Application (App)

The term app is an abbreviation for the full term “application software”. In 2010 this term became popular and it was the word of the year back then by the American Dialect Society.cf.[1] Part of the success of modern mobile operating systems is the possibility to develop third party applications based on an operating system (OS) specific application programming interface (API). By this way the functionality of the OS can be extended and allows a rich set of features to be added to the smartphones. From games to business apps, there is an almost infinite pool of possibilities in terms of app development given. cf.[2]

### 2.1.2 Application Programming Interface (API)

An application programming interface or short API is a set of commands, functions, routines and protocols for building software applications. APIs allow interacting with the system on a high abstraction level. An API defines functionalities that are independent of their implementations which allow definitions and implementations to vary without compromising the interface. cf. [3]

### 2.1.3 Smartphone

According to Wikipedia a smartphone is a mobile phone with an advanced mobile operating system. Furthermore these phones combine the classic features of a cell phone such as phone calls and sending text messages with functions of popular devices such as a personal digital assistant, a media player or a GPS module for navigation purposes. cf.[4]

# 3 Mobile – App Development

It is said that mobile devices are the fastest-growing and most impactful innovation of our time. This leads to the forecast claiming that by 2018 the majority of internet users will access the internet via tablet or smart-phone. The usage of the traditional PC will be reduced to perform complex tasks only. cf. [5]

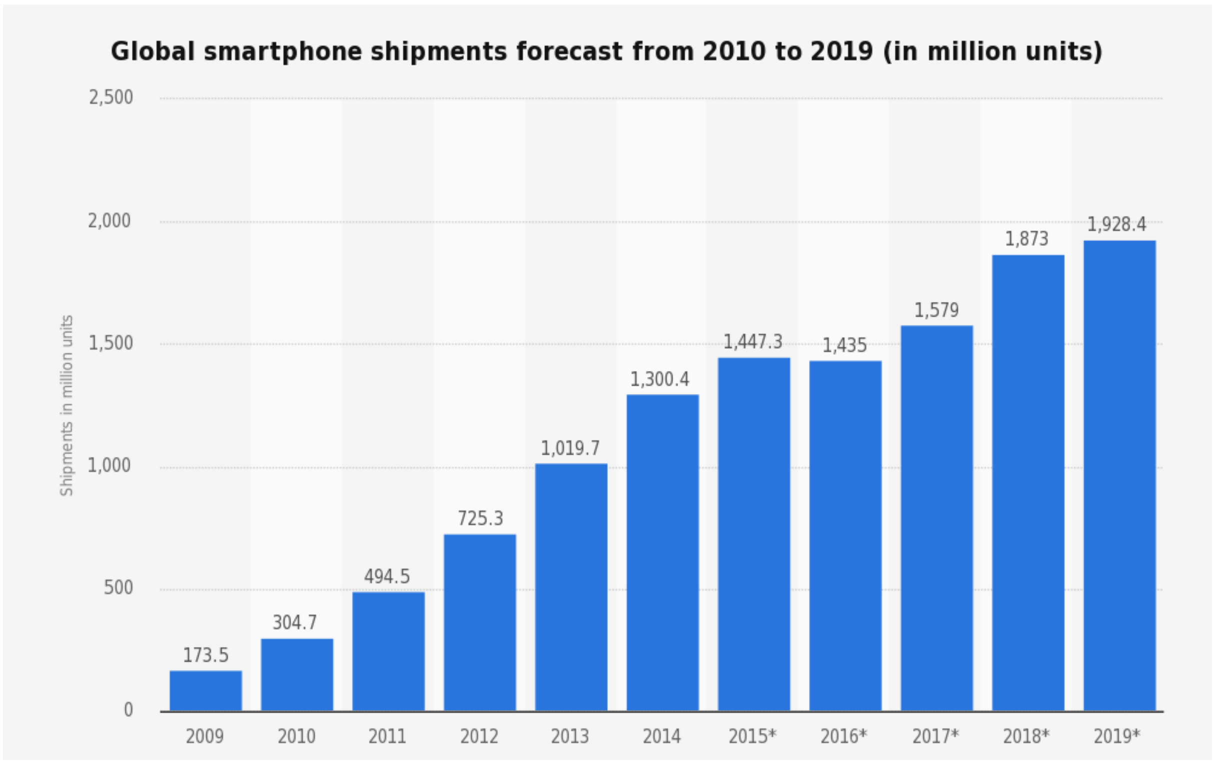


*Figure 1 Global smartphone shipments forecast from 2010 to 2019 (in million units) [6]*

According to [6] the shipments for smartphones are expected to reach over 1.7 billion units by 2018. This is an increase by a factor of ten compared to 2009s shipment numbers. By 2017 it is estimated that 34 % of the world's population will have a smartphone. The sales numbers have turned the market in a multi-billion dollar industry with revenue projected to top 272 billion U.S. dollars in 2015. cf. [6, 7]

The global success of this industry can be attributed to the growing popularity of several key operating systems. The two dominant systems are namely Android and iOS. By the end of 2013 these two held over 90 % of the global smartphone market share, with Android alone controlling almost 80 % as one can see in Table 1.This amounted to 226.1 million Android units and 51 million iOS units being shipped in the final quarter of 2013. The market domination of Android remains continuing with the shipment of Android phones in 2016 which is expected to hit around 680 million units. cf. [7]

*Table 1 Smartphone OS Market Share, Q1 2012 – Q1 2015 [7]*

| Period | | Android | iOS | Windows Pho-ne | BlackBerry OS | Others |
|---|---|---|---|---|---|---|
| **Q1 2015** | | 78.0% | 18.3% | 2.7% | 0.3% | 0.7% |
| **Q1 2014** | | 81.2% | 15.2% | 2.5% | 0.5% | 0.7% |
| **Q1 2013** | | 75.5% | 16.9% | 3.2% | 2.9% | 1.5% |
| **Q1 2012** | | 59.2% | 22.9% | 2.0% | 6.3% | 9.5% |

Figure 2 shows the development of the OS market share beginning in 2009. It is recognizable that iOS has held a constant market share since 2009. This can also be seen in Figure 2 which displays the market share of iOS from Q1 2012 until Q1 2015. The share has fallen from 23% to 15% in the last four years.

Contrary to this, Android has gained a massive popularity from 2009 until 2014. The market share increased over 20% fromQ1 2012 until Q1 2015 and holds almost 80% of the global market share now.

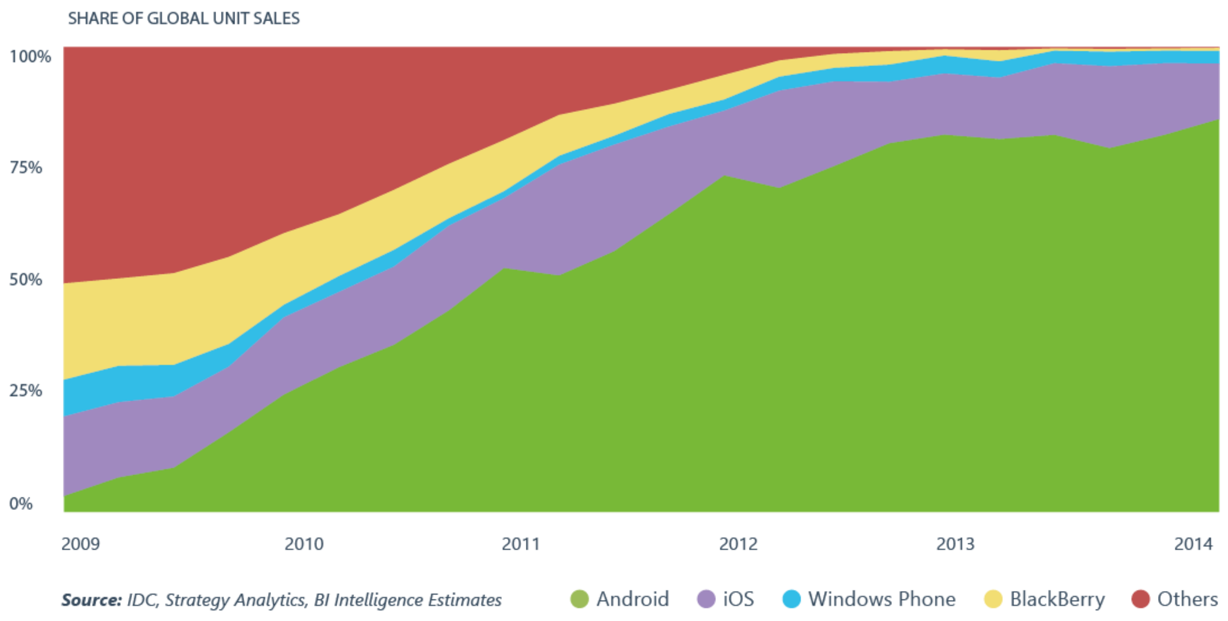


*Figure 2 Historical development, OS market share from 2009 until 2014 [7]*

According to Benedict Evans, a consultant for Andreesen Horowitz (a venture capital firm), the mobile market remakes the tech industry because it is so much bigger. Around four billion people buy a new smartphone every two years. In contrast to that only 1,6 billion PCs are replaced every five years. cf. [8] This can also be seen in Figure 3 that displays on the one hand the quarterly unit sales of PCs and on the other hand, beginning with 2007, the quarterly sales of smartphones which clearly show a shift in scale.

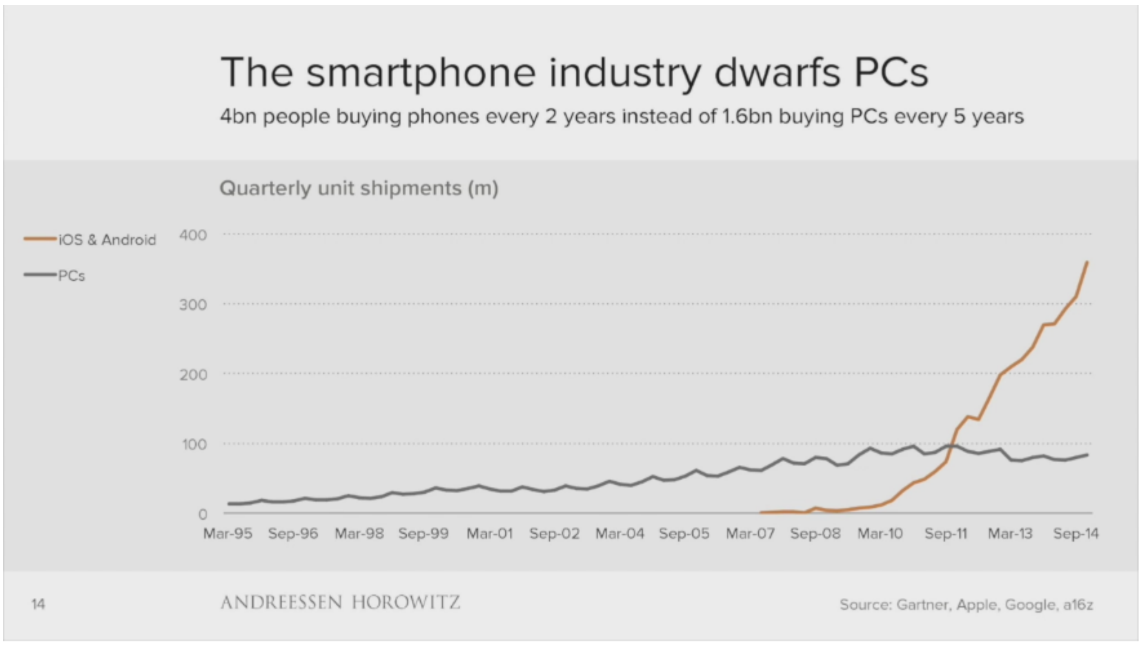


*Figure 3 The smartphone industry dwarfs the PC industry [8]*

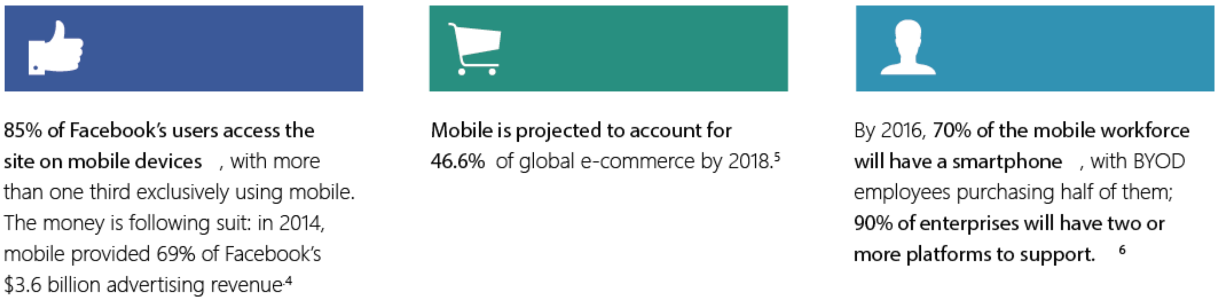
Smartphones replace the traditional PCs and therefore they are able to generate a high volume revenue stream for handset manufacturers, OS providers and last but not least for app developers. Evan states:

> *As of June 2014, mobile apps dominate the proportion of time spent online. [8]*

Context-aware mobile applications transform business processes, omnipresent access to personal and business information is available and in combination with the Internet of Things and wearables this technology is able to redefine and enhance our environment. cf. [9]

Having a look at Figure 4 that points out several key facts on going mobile makes clear why going mobile is not only an option anymore but  is becoming prerequisite to being successful: 69% of Facebook's 3.6 billion advertising revenue comes from mobile, 46.6% of global e-commerce is projected to be generated by e-commerce by

2018. By 2016 70% of the mobile workforce will have a smartphone and 90% of the enterprises will have to support two or more mobile platforms



**85% of Facebook's users access the site on mobile devices**, with more than one third exclusively using mobile. The money is following suit: in 2014, mobile provided 69% of Facebook's $3.6 billion advertising revenue[4]

**Mobile is projected to account for 46.6%** of global e-commerce by 2018.[5]

By 2016, **70% of the mobile workforce will have a smartphone**, with BYOD employees purchasing half of them; **90% of enterprises will have two or more platforms to support.** [6]

*Figure 4 Key facts on going mobile [9]*

This chapter has introduced key numbers and facts on mobile app development. The growth rates of the two dominant OS providers and the forecasts of smartphone sales for the next years were pointed out. Future prospects of computing will be mobile ones and as society is shifting away from a PC dominated world into a mobile device dominated world with a smartphone in everybody's pocket, developers will have to follow. cf. [9–11]

Nowadays smartphone users have diverse demands and needs regarding a smartphones` usage. Users expect to be able to check their flight status or their order history during a sales call to have instantly accessible information presented in a way that makes the most out of their devices` capabilities. Apps that fail to meet these demands are abandoned very fast and replaced by more promising options. More apps with even richer features running seamlessly on all platforms providing a rich and native usability experience are required. cf. [8, 9]

The following chapter will deal with concepts and technologies that allow to provide the user with the required features and furthermore the possibilities to reduce development time and cost will be pointed out.

# 4 The Gartner Hype Cycle View of Mobile Development

The 2014 Gartner Hype cycle for Mobile Applications and Development shows the current and future developments for key technologies in the mobile area. There are new movements such as: bring your own app (BYOA), mobile BPM, enterprise mobile app stores, mobile cloud or mobile social networks just to name a few.

Moreover, it covers technical topics such as Hybrid Mobile Development, HTML5, Mobile Web Apps and Native Mobile Development which are the more important topics in this paper and therefore will be dealt with in greater depth.

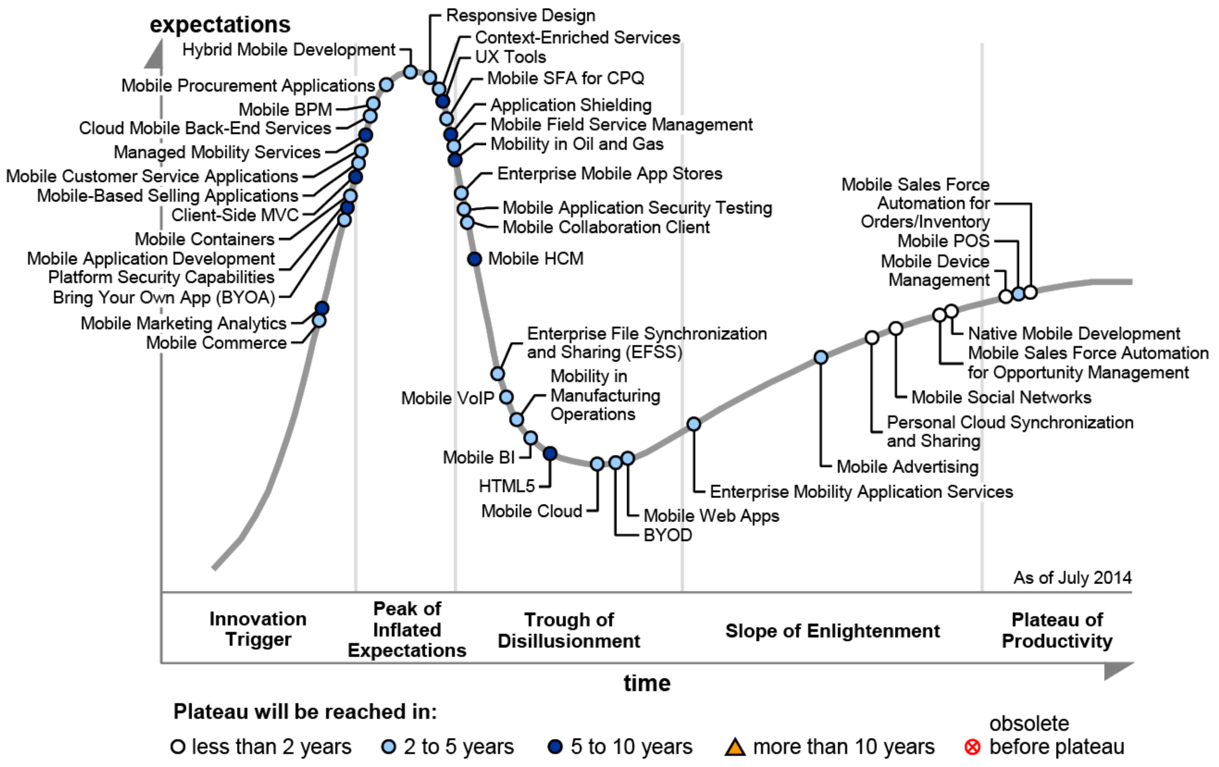


*Figure 5 Hype Cycle for Mobile Applications [12]*

The following section describes the technologies mentioned beginning with the most mature one (native app development) and continuing up to the one with the most expectations (hybrid mobile development).

As one can see in Figure 5 the topic of native mobile development is on its best way to leave the slope of enlightenment and enter the plateau of productivity. This has happened quickly in the last couple of years according to [12]

> *Hundreds of thousands of developers have developed native apps for today's dominant mobile OSs — Google Android and Apple iOS. [12]*

This approach has reached its maturity so fast because the large vendors Google and Apple provide a tool chain to support the development process from the design phase to the implementation phase on to testing and further to distribution and maintenance.

The category of mobile web apps can be found in the trough of disillusionment. A Mobile web app is an app that runs on a standard web server in combination with a mobile web browser. The technology behind is mostly a combination of HTML, Ajax and in recent developments HTML5 components. The major downside of this approach includes topics such as:

- Requires good network connection
- Restrictions in the usage of native OS features
- UI performance issues

In terms of behavior a mobile web app is similar to a rich internet application (RIA) only that it is tailored to fit the screen and performance restrictions of a mobile device. If reach and costs are key requirements for the app then mobile web applications are the way to go. cf. [12]

In the same trough HTML5 can be found; while HTML5 is not necessarily part of cross platform development or app development in general, it is important and influentially enough to mention it here.

> *HTML5 is a broad-based industry initiative that is the next step in the evolution of HTML. [12]*

While HTML5 is not one monolithic entity but rather a collection of many specifications, the key goal is to bring RIA-like capabilities into the language itself. This eliminates the need for third party plugins such as Adobe Flash or Silverlight which are an obstacle on mobile devices. Furthermore there is a wide adoption of hybrid architectures on mobile. HTML and other web technologies are used together with native-code wrappers to create apps which can be “compiled” and installed. This is a try to bridge the gap between HTML and native applications. This aspect is also dealt with separately in the hype cycle. cf. [12]

Gartner defines hybrid mobile development in the following way:

> *Hybrid mobile development allows developers to use Web skills and technologies to build mobile apps that can be loaded into an app store, downloaded to a mobile device and used like a native mobile app. [12]*

And further they state:

> *Hybrid technologies allow code reuse across multiple, incompatible mobile OSs and can yield apps that approach the look and feel of native applications. [12]*

Both statements sound enthusiastic and promising and this is quite likely because the topic of Hybrid Mobile Development is on the peak of inflated expectations in the Gartner Hype Cycle. This section has already mentioned some of the reasons why non-native application development is gaining popularity. The following sections will describe these reasons in more detail and point out the key points which need to be considered.

# 5 Native App Development

*Native mobile development is favored in the most demanding apps, where UI reactivity, graphics performance and processing speed are of paramount importance. [12]*

Each mobile device platform ships with a native programming language and a primary IDE to develop apps. For Android this is the programming language Java, and Google provides an integrated IDE with Android Studio that supports the developer in various ways, for example with code completion, pre-defined templates and ships as well with an Android device emulator. cf. [13, 14] Similar tool chains are available for Apple's IOS with Objective-C as a development language and XCode as an environment. This is also available for Windows Phone with C# as the programming language and with IDE Visual Studio as the first choice development environment. A short overview of these facts can be found in Table 2.

*Table 2 Quick facts - Native Application Development [13]*

| Android | IOS | Windows Phone |
|---|---|---|
| **Language:** Java<br><br>**Primary IDE:** Android Studio, Eclipse | **Language:** Objective-C<br><br>**Primary IDE:** XCode | **Language:** C#<br><br>**Primary IDE:** Visual Studio |

All of these tools enable a smooth development process of apps for a specific platform. Native applications are built from the ground up to fit a platform specific environment.

*The precise, platform-centered nature of native development means that these apps have no limits in terms of access to APIs and device features, performance optimization, and platform-specific best practices for user interface design. [13]*

To introduce an analogy here, to develop a native app is like writing an article in one's mother tongue. This enables the writers to use the full range of words and phrases (the API) and all the language specific beauty (device features). The language choice will be reflected in writing performance as well as in app performance.

While this sounds like a silver bullet, the problem with silver bullets is that they are very rare. The drawback in writing a book in one’s mother tongue (especially if it is some exotic language) is the limitation of the number of people the book will reach. The same is valid for app development. An app developed in ObjectiveC will only be available for iOS devices. The same applies to apps developed in C# for Windows phone - they will only be available for Windows Phones.

According to [13] this is one of the three major problems when developing mobile apps which are in detail:

1. Testing on multiple devices
2. Building native apps for multiple platforms
3. Lack of skilled mobile developers

As indicated by the survey in [13], 53% see “testing efficiently on many different hardware sets and screen sizes” as the biggest pain point. Followed closely by 50% who say that “developing native apps for multiple platforms” is an issue. The third most common obstacle is the availability of skilled mobile developers with 40%.

To develop native apps from scratch forces developers to be skilled not only with one language and one tool but as pointed out, to be able to develop for the three major platforms with three languages and three tool chains. This type of developer is barely available and this mostly means additional developers must be hired which is a valuable cost factor. cf. [11, 13]

Further downsides of this approach are [9]:

- Soloed development environments (multiple languages, tools, teams)
- Triples the effort to add or change features
- Slow release cycles

A common approach is to have a team for each platform; this adds a lot of administrative overhead as a feature request has to be communicated to all three teams. In addition, the coordination effort increases as the three teams need to stay in sync with the backend developers and because of different paradigms it might be hard to satisfy all three app developing teams from the backend side.

Point two goes hand in hand with this; solving a bug means to go through three code bases and search and fix it three times. Furthermore it should be considered that the platform can change and develop as well. For example with the introduction of the current version of Android Lollipop a major UI change has happened. Google intro-duced material design that had a new style guide.

# 6 Cross Platform Development

Cross Platform mobile Development faces four big challenges that need to be discussed:

**Problem 1: Different user-interface paradigms**

All three major platforms present the graphical user interface (GUI) in a similar way. All of them allow multi-touch interaction with the device. But in detail there are many differences. The navigation within the applications and pages is different, the conventions for the presentation of data as well as the ways to invoke and display menus. How to insert data and the touch gestures vary from system to system.

There are design guides available for each platform that suggest how the application is supposed to look like and how to behave in certain cases.[1] Users rely on this behavior and expect applications to follow these paradigms. cf. [15] Charles Petzold describes this in his book with the following words:

> *Each platform acquires its own “culture” of sorts, and these cultural conventions then influence developers. [15]*

**Problem 2: Different development environments**

Software development tasks are heavily supported by sophisticated integrated development environments (IDEs). Such IDEs exist for all three platforms, but they are different in their look and behavior cf. [13, 15].

- For iOS development, XCode on the Mac
- For Android development, Eclipse on a variety of platforms
- For Windows Phone development, Visual Studio on the PC.

**Problem 3: Different programming interfaces**

The APIs that are exposed on each platform depend on the underlying operating system. All three systems have different operating systems and therefore expose different APIs. Obviously they are similar in terms of what they can do but they use different names and concepts.

---

[1] https://developer.android.com/design/index.html
https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/
https://dev.windows.com/en-us/design

To toggle a user between two states, all three platforms provide an element but all three environments name them differently:

- In iOS it is a “view” called `UISwitch`
- In Android it is a “widget” called `Switch`
- On Windows phone it is called a “control” with the name `ToggleSwitchBut-ton`

This is just an example that shows, on the surface, where the differences are; as mentioned above it is not only about naming but also about programming and implementation concepts. cf. [15]

**Problem 4: Different programming languages**

The fourth difference is the programming language that is tied towards the development for a specific platform:

- Objective-C for iOS
- Java for Android
- C# for Windows Phone

They all have in common that they are object-oriented third generation languages but besides that the language concepts vary heavily. cf. [16, 17]

## 6.1 Reasons for Cross Platform Development

Intensive competition does not give much space for companies to be slowed down in launching new applications. Moreover, companies most likely need to support more than one mobile platform. This is where the idea of cross platform development comes in handy. The following list is derived from [18] and outlines the major reasons for using cross platform development tools:

- **Reduction of required skills** – Developers can use one single programming language independent of the target application
- **Reduction of coding** – With the paradigm of developing once and compiling many times for different OS the code base can be kept small.
- **Reduction of development time and long term maintenance costs**–This benefit comes with the fact that one single code basis needs to be maintained and failures and features do not need to be implemented / solved multiple times

- **Decrement of API knowledge** – It is not needed to know each of the individual APIs because the tools abstract them and provide one single API across all supported platforms.
- **Greater ease of development** – compared to building native apps for each operating system.
- **Increment of market share** – targeting multiple OS allows to reach more users which increases the chance to raise the Return on Investment (ROI).

One of the major driver for cross platform tools is the try to reduce costs on the developing side. Especially in the long run supporting multiple code bases is time consuming and challenging. This statement is supported by the following quote:

> *The economic constraints of native development are a major factor in the growing popularity of web apps, hybrid apps, code translators and Mobile Application Development Platforms […] [13]*

## 6.2 Cross Platform Tool Categories

There are various approaches possible when it comes to cross platform application development. An overview can be seen in Figure 6 which displays the categories ordered by complexity on one axis and the output on the other axis.

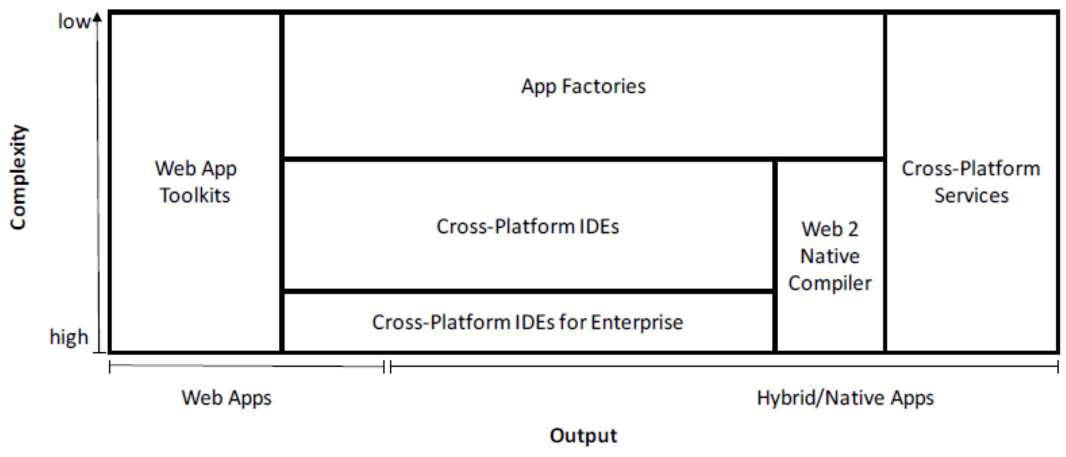


*Figure 6 Categories of cross-platform app development tools [11]*

The following sections will point out the most common approaches and give a brief overview about them.

### 6.2.1 Web App ToolKits

As seen in Figure 6 web apps and web app tool kits offer a complexity range from high to low but have always web pages as output that look and work well in a mobile browser.

> *[…] mobile web apps are just regular websites optimized to look good and function well on mobile devices and they can provide a quality app-like experience […] [13]*

The technology that is widely used in this area is classic frontend web development technology. This includes HTML, CSS and JavaScript which provide the logic behind the app. Further there are a lot of tools and libraries available that allow creating touch optimized apps. For example jQuery Mobile and Sencha Touch are two frameworks that provide UI components and logic for sliders, swipes and other touch controls which are common in mobile applications.

On the pro side of this technology stands the very common knowledge of mobile technologies which have been around already for years which means the developer availability is higher. A lot of people are able to develop webpages; to develop a mobile web app is just a special way of developing such a page. Another benefit this approach has is that as web apps only need a browser to be displayed and be used they are near-universal cross platform enabled. Whatever device enters the market as long as it has a touch display and a browser, the web app can be used from it. cf. [13]

The two major downsides of these apps are on the one hand that they are not able to utilize any of the devices built in sensors and are therefore very limited to what they can offer to the user. On the other hand as they are just web pages these apps cannot be distributed through common means such as the Apple App Store or Google's Android Marketplace. cf. [9]

### 6.2.2 App Factories

On the low side of the complexity App Factories can be found. These are environments that allow a "Drag & Drop" approach when creating new apps. The main target customers here are "non-developers" who want to create an app in a few hours or days. The apps are based on templates and are then compiled in native apps to run on various platforms. Examples for this are AppShed, AppYourself, Weever Apps, etc. [9, 11, 12]

### 6.2.3 Cross-Platform Integrated Development Environments

Cross-Platform Integrated Development Environments (CP-IDEs) follow the principle code once and deploy everywhere. This happens mostly by using the CP-IDEs own software development kit (SDK) which is then compiled into native apps. What this approach tries is to build an abstraction layer on top of the native API, which allows to program and to interact on one interface and the differences on the target platforms are handled from the CP-IDEs.

Generally, there can be seen two strategies in this environment. Some tools try to provide the most up-do-date development possibilities for their target platforms and focus on fewer platforms. Others try to be generalists that support as many platforms as possible. cf. [11]

Example frameworks that follow this approach are: Appcelerator, Embarcadero, V-Play and Xamarin.

### 6.2.4 Cross-Platform Integrated Development Environments for Enterprises

The focus of these tools lies on business environments. These tools support the workflow of app development and deployment in an enterprise environment. The benefit comes with a tighter integration into business systems such as ERP, CRM or shop systems. The providers include APIs for these systems within their environment and focus on business cases to enable easier development of business applications.

Frameworks that would fall into these sections are AnyPresence, AppConKit, IBM Worklight and others. cf. [11]

### 6.2.5 Cross-Platform Compilers

The key idea behind cross-platform frameworks is the approach: write once, run an-ywhere (WORA). This means that one single codebase can be used to generate the app as output for any supported platform. cf. [9] This can be seen in Figure 7. The tools bridge the requirements of native device APIs with the chosen programming language in order to be able to develop it with one single programming language. cf. [11] Examples that use this technology are Alchemo, Cocoon or Cordova.

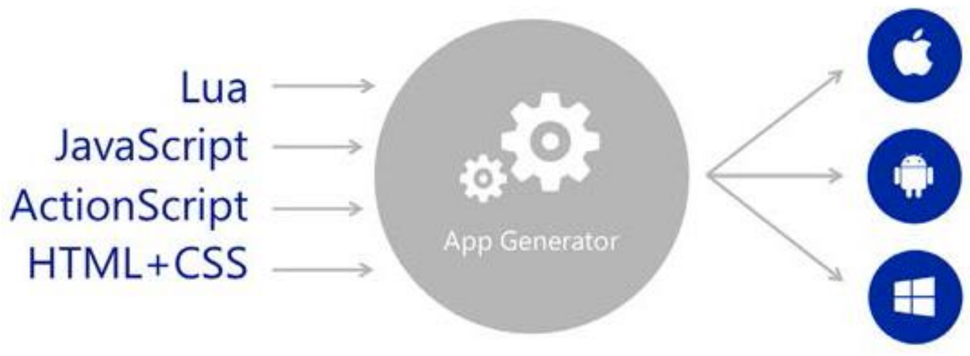


*Figure 7 Write once, run anywhere approach [9]*

## 6.3 Cross Platform Tools on a Closer Look

The following section introduces some of the most popular tools and points out their pros and cons in very general terms.

### 6.3.1 Appcelerator

JavaScript and a model-view-controller-framework (MVC) are the basis for each Appcelerator application. The development platform is enterprise-focused and comes with tools for analytics, performance management, monitoring and dedicated infrastructure. Appcelerator 4.0 is the latest version of the Appcelerator Platform. It includes Arrow, Hyperloop Abstraction Layer, Apple WatchKit, support for Windows native apps and a Platform marketplace. Moreover, a private cloud option for enterprise customers is offered which is especially important for customers who work with sensitive company data. cf. [19, 20]

*Figure 8 Appcelerator Logo*

#### 6.3.1.1 Pros

- Usage of native components is a performance win. Generally, the framework tries to normalize the UI across platforms.
- The use of JavaScript to normalize code across platforms enables developers to leverage existing skills on multiple target platforms.

- Besides the pure multiplatform features Appcelerator provides value-adds such as a Backend-as-a-Service (BaaS), app analytics which is interesting for business customers and a marketplace for 3rd party components to reduce the development amount of custom components.

#### 6.3.1.2 Cons

- The target platform SDKs needs to be managed locally. This requires a controlled build environment process, otherwise SDK version & build-related issues can become very time consuming.
- This point might be a pro but at the same time a con as the approach to normalize UI across platforms forces the developers to train on proprietary technology which cannot be used outside the framework

### 6.3.2 PhoneGap

One of the best known development tools in the cross platform community is PhoneGap. It is an Adobe product that is based on Apache Cordova which is an open source project. The whole tool suit is completely free to use which might be the reason why it is so popular. Adobe describes the product as the following:

> *Write a PhoneGap app once with HTML and JavaScript and deploy it to any mobile device without losing features of a native app. [21]*

The benefit is that it is standards-based and open-source and allows building cross-platform mobile apps with technologies such as HTML, CSS and JavaScript. This is shown in Figure 10 in an abstract way. Additionally, it supports the three major OS: Android, iOS and Windows Phone. cf. [21–23]

*Figure9 Phone Gap Logo [24]*

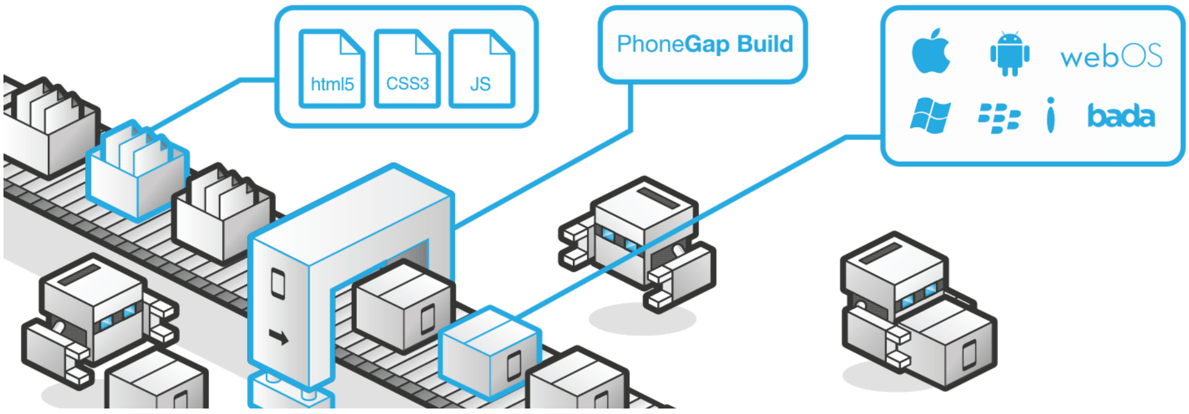


*Figure 10 Building Process of PhoneGap [21]*

### 6.3.2.1 Pros

- One of the major benefits of this framework as of all the frameworks that are built on standard web technology, is the fact that developers that are able to handle this technology are easier to find than specialists for platform specific developments. This reduces the costs for training and can enable a quick-to-market stance in companies ready to adopt it.
- Cordova apps install just like a native application and are therefore distributable through the app stores and can leverage the app store promotion abilities.
- The plugin architecture of Cordova allows handling the access to native device APIs in a modular way. There are already a lot of plugins which allows the developers to focus on their web-based development skills.
- Cordova is open source and free so there are no licensing costs.

### 6.3.2.2 Cons

- Open source often is a blessing and a curse at the same time. In the case of PhoneGap the custom plugins are available but the available ones might not support the needed platforms and further, bugs might not be resolved in a timely manner.
- If no plugin is available one need to write its own and this stands in contradiction to the main idea behind PhoneGap that allows an abstraction of the platform specific details.
- One of the major points of critique for PhoneGap apps has often been the performance drawback. Native UI will always outperform a hybrid solution but the

improvements made for WebViews and a strong hardware performance narrow down this gap.

### 6.3.3 Sencha Touch

Sencha touch is a JavaScript library / framework that is focused on features for developing mobile web applications that look and feel like native applications. It is based on web standards such as HTML5, CSS3 and JavaScript. The supported platforms are Android, iOS and Windows as well as Tizen and BlackBerry devices. Apps are developed in HTML5 and are accessible via the device browser and create a native-app-like experience. cf. [13]

The main enterprise product of Sencha is Ext JS5 which lets developers create HTML5 apps which can then be converted into native apps with PhoneGap. Sencha's HTML5-focused approach allows its apps to run across browsers as well as the latest touch-based devices and has attracted clients such as Google, CNN and Samsung.

*Figure11 Sencha Logo [10]*

#### 6.3.3.1 Pros

- The MVC approach that is built into Sencha Touch can be seen as an architectural benefit as well as a library of UI components, an extensible API and UI themes among other features.
- Native packaging via Apache Cordova/PhoneGap or Sencha's SDK is possible.

#### 6.3.3.2 Cons

- Performance can be an obstacle with apps written with Sencha Touch. This is caused by the same challenges Cordova/PhoneGap has to deal with. Especially if developers are writing inefficient JavaScript and DOM structures.
- Sencha provides its own technology stack and this approach creates a vendor lock-in.

- As soon as access to additional native APIs is necessary, custom plugins will be needed and they require specialized platform development skills.

### 6.3.4 Qt

Qt ("Cute") is a cross-platform development tool that targets a number of embedded, desktop and mobile platforms. The difference to most approaches is that Qt comes with its own development language named QML. QML is a CSS and JavaScript like language that is backed with an extensive amount of libraries and UI components written in C++. The tool is available as a free and open source version but can also be commercially licensed which offers support and other built-in components. cf. [25]

Qt

Code less.
Create more.
Deploy everywhere.

*Figure12 Qt Logo [26]*

#### 6.3.4.1 Pros

- Libraries for threading, networking and animations are shipped with the framework.
- A solid chain of development tools with Qt Creator IDE & Qt Designer are provided. Furthermore code profiling is available in the QML Profiler.
- For internationalization the tool Qt Linguist is provided. It allows shipping multiple languages within a single binary.

#### 6.3.4.2 Cons

- Qt offers a lot of libraries and functionality so it is challenging to learn it without external help.
- QObject and QWidget are not thread-safe, painting is not possible from threads
- QMake is outdated and complicated for more advanced projects cf. [25]

# 7 Xamarin

Xamarin is a cross-platform mobile application development solution that creates a unified environment for developers. cf. [17] The key factors that make the Xamarin framework different from other cross platform tools are: the handling of hardware features, the concept of code sharing and native bindings. These factors lead to good performance results and native user experience which is a difference compared to many other cross-platform tools, too.

*Figure 13 Xamarin Logo [27]*

The Xamarin approach is described as the following:

> *[…] building cross-platform native apps combines the essential characteristics of native apps-native UI, native performance, and native device access-with the efficiency and time-to-market advantage of code sharing [9]*

Generally, there are two approaches when developing mobile apps with Xamarin. On the one side one can use Xamarin.iOS / Xamarin.Android and on the other side there is the option of using Xamarin.Forms. Both approaches will be described in more detail in the following sections.

## 7.1 A single language for all platforms

The language of choice for Xarmarin is C# which is classified as a multi-paradigm programming language. C# code is traditionally imperative but can be flavored with declarative or functional programming paradigms. It is more or less a strongly-typed imperative object-oriented language which has been influenced by C++ and Java.

Over the years language concepts have been added such as generics, lambda functions, LINQ and asynchronous operations. And since April 3, 2014 an open source version of the .NET compiler Platform has been available. cf. [16, 17]

At a low level point of view the .NET framework provides features for C# basic data types such as integers, doubles, strings, etc. But .NET provides an enormous amount of libraries that are included within the .NET framework. They help developers dealing with common chores such as:

- Math
- Debugging
- Reflection
- Collections
- Globalization
- File I/O
- Networking
- Security
- Threading
- Web services
- Data handling
- XML reading and writing

Besides this rich set of features that the .NET framework offers, there are other reasons why Xamarin utilizes C# as a language of choice.

Back in 2000 shortly after Microsoft had announced the .NET framework, Miguel de Icaza and Nat Friedman (founders of Ximian) initiated an open-source project called Mono. The main idea behind Mono was to create an alternative implementation of both, the C# language and the .NET framework which could run on Linux.

In 2011 out of Ximian a new company was founded with the name Xamarin which adapted the open-source Mono to form the basis of Xamarin cross-platform mobile solutions.

In the very beginning the focus of Xamarin was on three basic sets of .NET libraries which are:

- Xamarin.Mac, also known as MonoMac
- Xamarin.iOS, also known as MonoTouch
- Xamarin.Android, known as Mono for Android

These three libraries form the basis of the Xamarin Platform. They implement .NET wrappings of native Mac, iOS and Android APIs. This allows developers to write applications in C# but to target native APIs of these three platforms with the great benefit of additionally having access to the whole .NET framework class libraries which is illustrated in the following picture and is discussed in more detail in the following paragraphs.

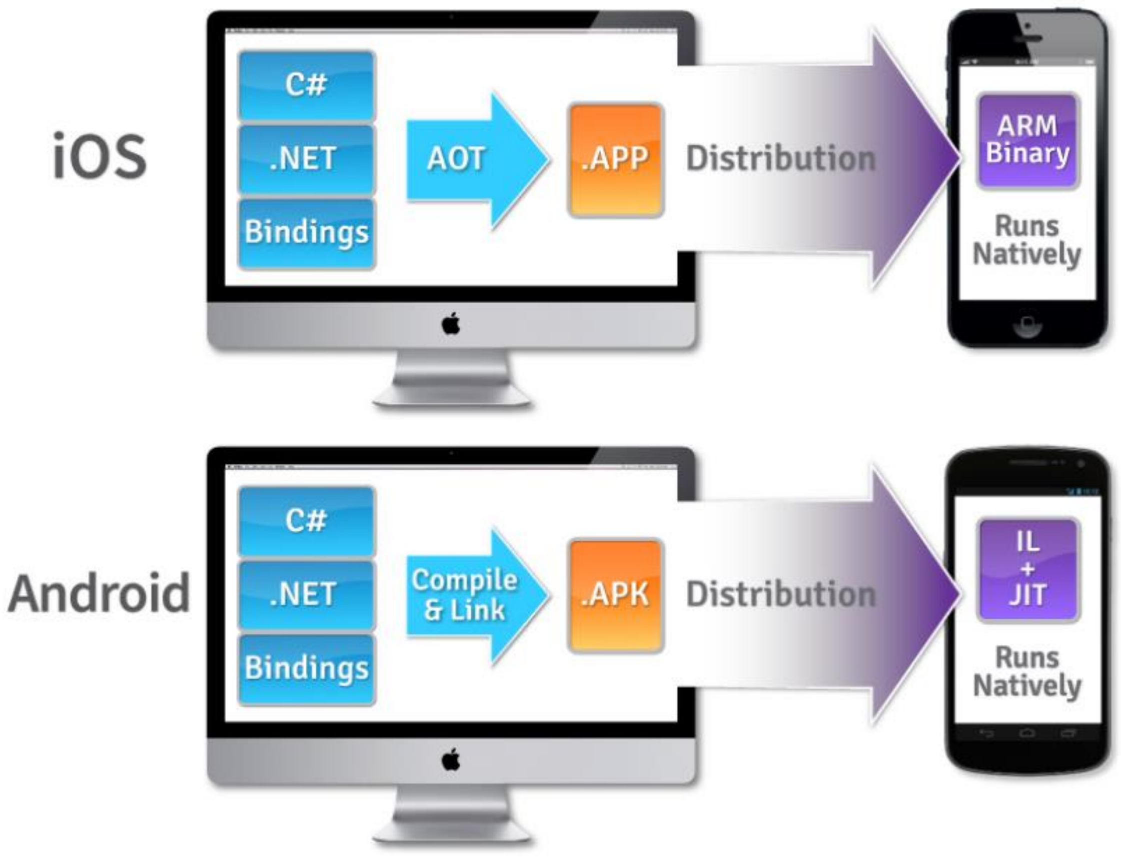


*Figure 14 The Xamarin architecture [27]*

## 7.2 Hardware Features

To provide a direct and identical link between the Xamarin developer platform and the native APIs, a native binding is created which allows to directly accessing the underlying iOS or Android SDK. This means that all the native objects can be used with the same names, same properties, attributes and methods. Xamarin does not re-implement the code in C# which allows the framework to release same-date releases for any new major mobile operating system updates cf. [9, 17]

Supported features are:

- Contacts
- Geolocation

- Compass, Gyroscope and Accelerometer
- Video and Audio
- Notifications

## 7.3 Code Sharing

One of the key features in building cross-platform applications is being able to share code across the applications running on various platforms. Here two approaches can be used, firstly shared projects and secondly portable class libraries. Because of having different systems underneath, this is rather complicated as different platforms often use a different sub-set of the .NET Base Class Library (BCL)

> ***Shared Projects*** *use a single set of files and offer a quick and simple way in which to share code within a solution and generally employs conditional compilation directives to specify code paths for various platforms that will use it (for more information see the Shared Projects article and the Setting up a Xamarin Cross-Platform Solution guide ).*
>
> ***PCL*** *projects target specific profiles that support a known set of BCL classes/features. However, the down side to PCL is that they often require extra architectural effort to separate profile specific code into their own libraries. For a more detailed discussion on these two approaches, see the Sharing Code Options guide.*

### 7.3.1 Benefits

- Centralized code sharing – this approach allows to write and test code in a single project. This functionality can then be consumed by other libraries or applications.
- Refactoring operations will affect all codes loaded in the solution.
- The benefit of PCL is that it can be easily referenced by other projects. Additionally, the output assembly can be shared to reference it in other solutions.

### 7.3.2 Disadvantages

- Because the same Portable Class Library is shared between multiple applications, platform-specific libraries cannot be referenced. [28]
- The Portable Class Library subset may not include classes that would otherwise be available in both MonoTouch and Mono for Android (such as DllImport or System.IO.File). [28]

Figure 15 shows a cross-platform application architecture that uses a portable class library to share code. Dependency is used to pass in platform-dependent features.

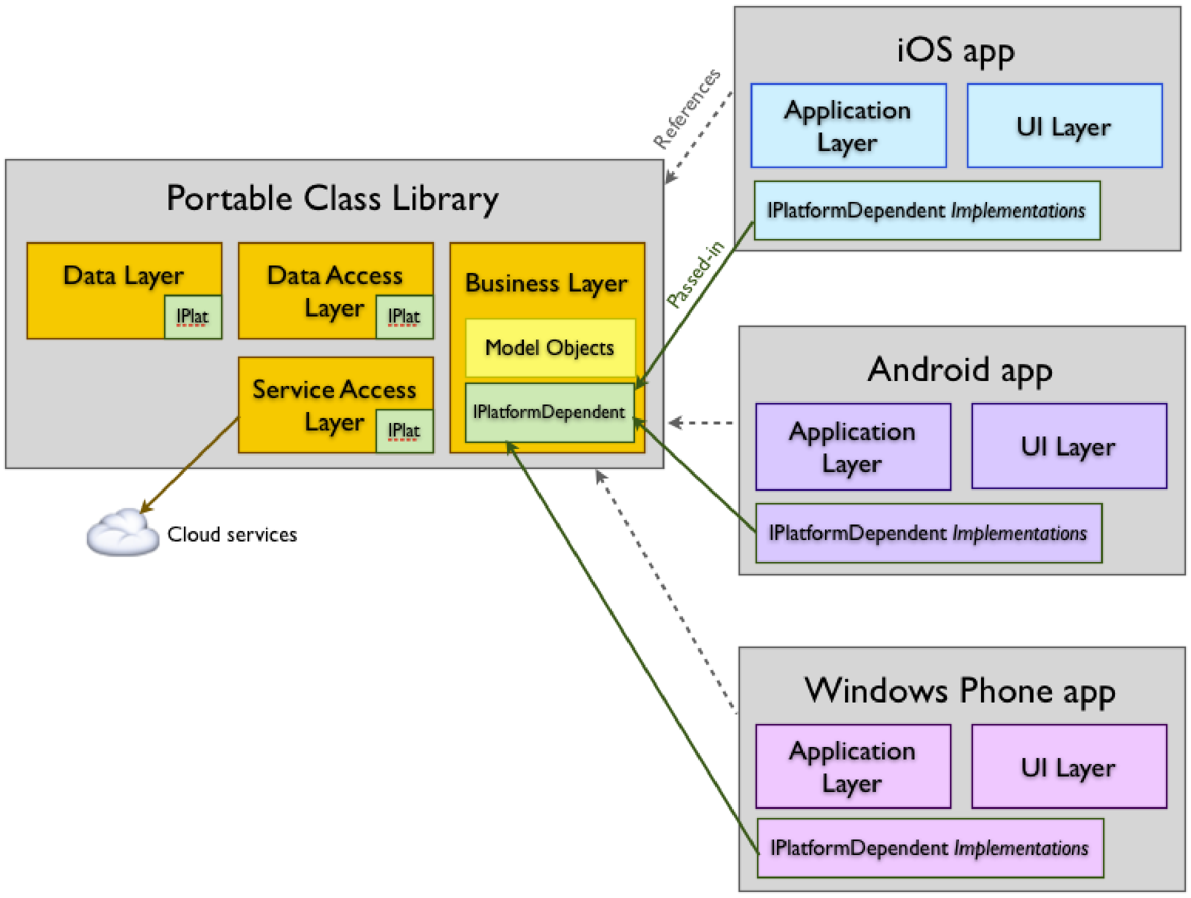


*Figure 15 Usage of a Portable Class Library Architecture [28]*

## 7.4 Native Bindings

Native bindings are a mechanism that allows the access to native libraries from within the Xamarin mobile application. The concept behind is to create a binding or mapping to the methods in C#. This is a semi-automated process where Xamarin is able to create many of these bindings by itself. But it does take some rework from the developer to make it work.

The major benefit of using native bindings is that it allows incorporating with third party libraries or libraries written in another language. An example would be an optical character recognition (OCR) library that is only available in C# which then could be included through native bindings. cf. [27]

## 7.5 The Xamarin development frameworks

There are two approaches to build a Xamarin application which are either based on Xamarin.Android / Xamarin.iOS or through the Xamarin.Forms framework.

Xamarin.iOS / Xamarin.Android therefore allows the developer to create separate user interfaces for each target operating system. This can be seen in Figure 16. The basis for the application is the so-called “Shared C# Logic” which contains all the business logic of the application, database or hardware access logic. cf.[29] On top of this the user interface will be implemented for each platform individually to allow full control of all the native features each platform has to offer. The main benefit of this approach is that although it is for 100 % managed C# code, it is able to access all user interface controls which the particular platform offers. This means there is no need to learn Objective C nor any Java code to develop native applications in both platforms. cf. [9]

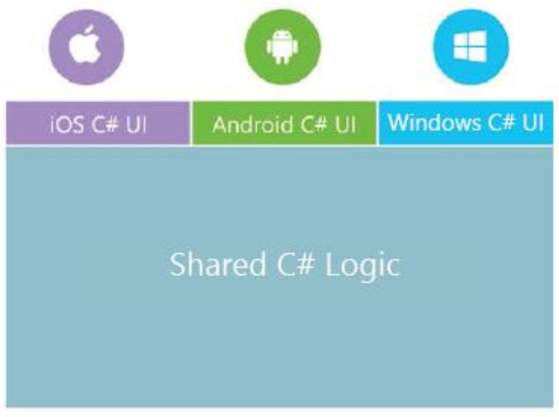


*Figure 16 1.1 Xamarin.iOS / Xamarin.Android architecture [9]*

Xamarin states that with this approach around 75 % of an average application code can be shared and therefore does not have to be implemented, tested and maintained several times. Of course, the exact code sharing percentage depends on the application type where game apps regularly require more specific user interface code than data driven applications. Nevertheless, this code sharing concept leads to a significant reduction of the development lifecycle and results in faster release schedules.

The second approach which is named Xamarin.Forms allows implementing the applications UI in 100 % shared code. As it can be seen in Figure 17the business logic or hardware layer works in the same way as with the classical UI approach. There is one shared C# code base that needs to be maintained. The difference is given by the circumstance that even the UI has a shared C# code base or furthermore, that it can also be developed as a declarative approach which is named XAML (Extended Application Markup Language). Every particular element of the UI controls are mapped to their native UI elements of the specific platform.

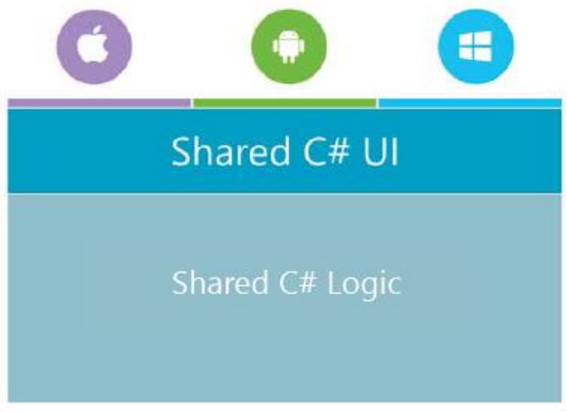


*Figure 17 Xamarin.Forms architecture [9]*

This approach as well allows creating native user experience on each platform that further adheres all the design principles for each platform individually. But by using the Xamarin.Forms approach one is not only limited to this provided abstracted UI elements, but it is still possible to access native UI components or functionality on each mobile platform with so-called platform renderers. cf. [9]

Figure18 shows screenshots of a Xamarin.Forms program that contains a Label, a Button, a Switch and a Slider. On all three devices these elements look different because the rendering happens platform-specific natively for each device.

What is especially interesting is that the tool includes special ToolBarItems which are mainly used for navigation purposes. Some of them have icons and some do not and the positioning of them is fully handled by Xamarin.Forms itself and renders to like it is expected for every platform.

As an example on the iPhone these are rendered with an element called UIBar-ButtonItem as the three icons and three buttons at the top of the page. On the Android the first three are rendered as items on an ActionBar, also at the top of the page. On Windows Phone they're realized as items on the ApplicationBar at the page's bottom. cf. [15]

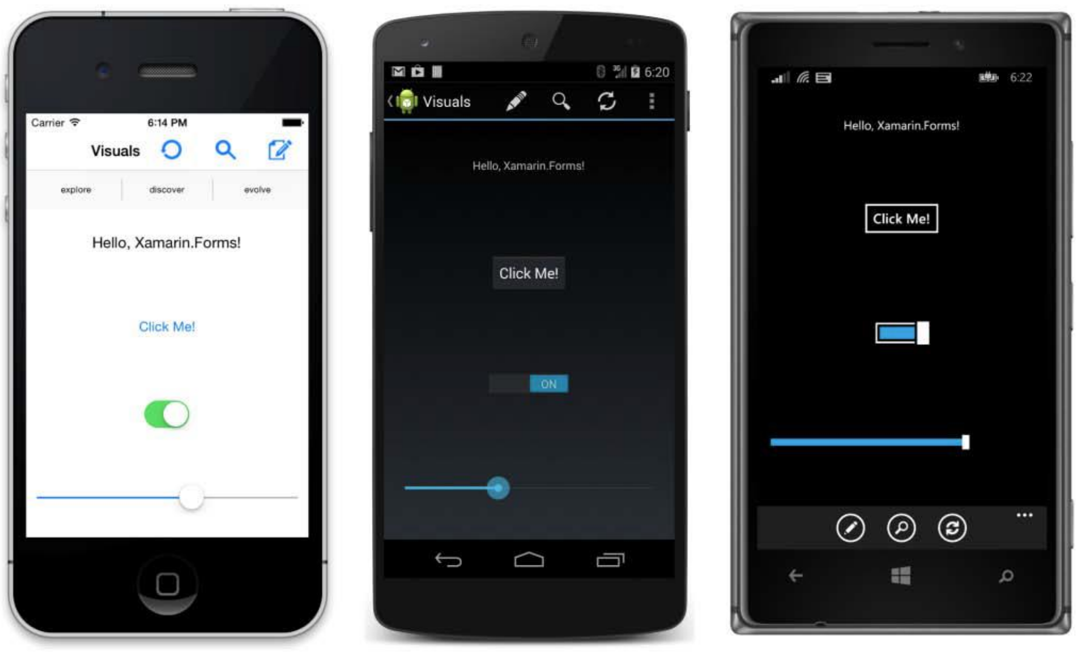


*Figure18 Xamarin.Formsapplication [15]*

## 7.6 A Closer Look at Xamarin Cross Compiling Technology

Xamarin.iOS provides for C# compilation to the native functions and also includes the core Mono .NET assemblies compiled specifically for iOS. Applications can be written in C# code in the Integrated Development Environment (IDE) of one's choice which are Microsofts preferred IDE Visual Studio or Xamarin Studio, but it must be compiled and deployed from a Mac using the MonoDevelop IDE. Unlike most Mono or .NET applications, Xamarin.iOS apps are statically compiled where compilation is accomplished through the Mono Ahead-of-Time (AOT) compilation facilities. The resulting assemblies are compiled to static code which results in limitations surrounding dynamic generation and execution. As a result Xamarin.iOS does not support .NET APIs that require dynamic code generation or execution. These include Reflection.Emit(), Generic Virtual Methods, and P/Invokes in generic types, among others. Full details are available from the official Xamarin documentation site at http://docs.xamarin.com/ios/about/limitations. It is also possible to bind to native iOS libraries to take advantage of existing or third-party components written for iOS. Xamarin supports binding to native Objective-C libraries using the .NET P/Invoke framework as long as the classes that need to be bound are not generics. cf. [30]

The Android System is based on the Java technology and so it is based on JIT (Just in time) compiling. Therefore Android apps run in a sort of virtual machine which is called the Dalvik Virtual Machine (Dalvik VM). Dalvik VM is optimized for devices with limited resources such as Smartphones and other mobile devices. Mono, which is the underlying technology of Xamarin itself, comes with a VM which is called the Common Language Runtime (CLR). Figure 19 illustrates that with using Xamarin.Android where both VMs are utilized at the same time. The technology that allows this to happen is on the one side the concept of peer objects that come with Xamarin and on the other side the feature called Java Native Interface (JNI) that allows calling non Java code from non-Java code such as C++ or C#.

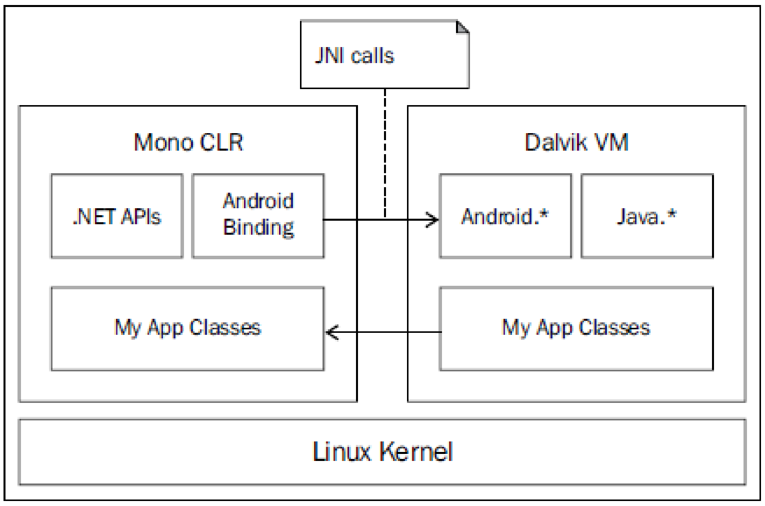


Figure 19 Mono Architecture on Android [29]

### 7.6.1 Java Native Interface Overview

Oracle defines the JNI as the following:

> *JavaTM Native Interface (JNI) is a standard programming interface for writing Java native methods and embedding the JavaTM virtual machine* into native applications. The primary goal is binary compatibility of native method libraries across all Java virtual machine implementations on a given platform. [31]*

Initially the idea behind JNI was in some cases that the ability of Java would not be enough to meet the needs of an application. Developers can use the JNI to write Java native methods to handle those situations when an application cannot be written entirely in Java. cf. [29, 31]

Some examples of this can be:

- The standard Java class library does not support the platform-dependent features needed by the application.
- A library is available in another language and it should be accessible by Java code.
- For performance reason a portion of the code that is time-critical needs to be implemented in a lower-level language.

The JNI interface allows performing the following tasks:

- Create, inspect and update Java objects
- Call Java methods
- Catch and throw exceptions
- Load classes and obtain class information
- Perform runtime type checking

### 7.6.2 Peer Objects

The second part of the architecture used by Xamarin.Android is the concept of peer objects. These objects are a pair of objects that work together to perform the tasks and actions needed in the Android app. One side of the peer is residing in the Mono CLR and the other peer resides in the Dalvik VM.

To enable this behavior Xamarin.Android is shipped with a set of assemblies which are called the Android binding libraries. For each Java class a corresponding class in the Android library is available and the methods on the binding classes act as wrappers to call corresponding methods on Java classes. These wrappers are commonly known as Managed Callable Wrappers (MCW). Whenever a C# class is created that inherits from one of the binding classes, a corresponding Java proxy class is generated at built time. The Java proxy contains a generated override for each overridden method in the C# class and acts as a wrapper to call the corresponding method on the C# class.

The creation of peer objects can be done on both sides, either in the Dalvik VM by the Android application framework or from the Mono CLR by overridden methods created by the developer. Via the Android.Runtime.IJavaObject.Handleproperty the reference that is held by each instance of a MCW can be accessed. Figure 20 shows the collaboration of the peer objects within the two environments. cf. [15, 27, 29]

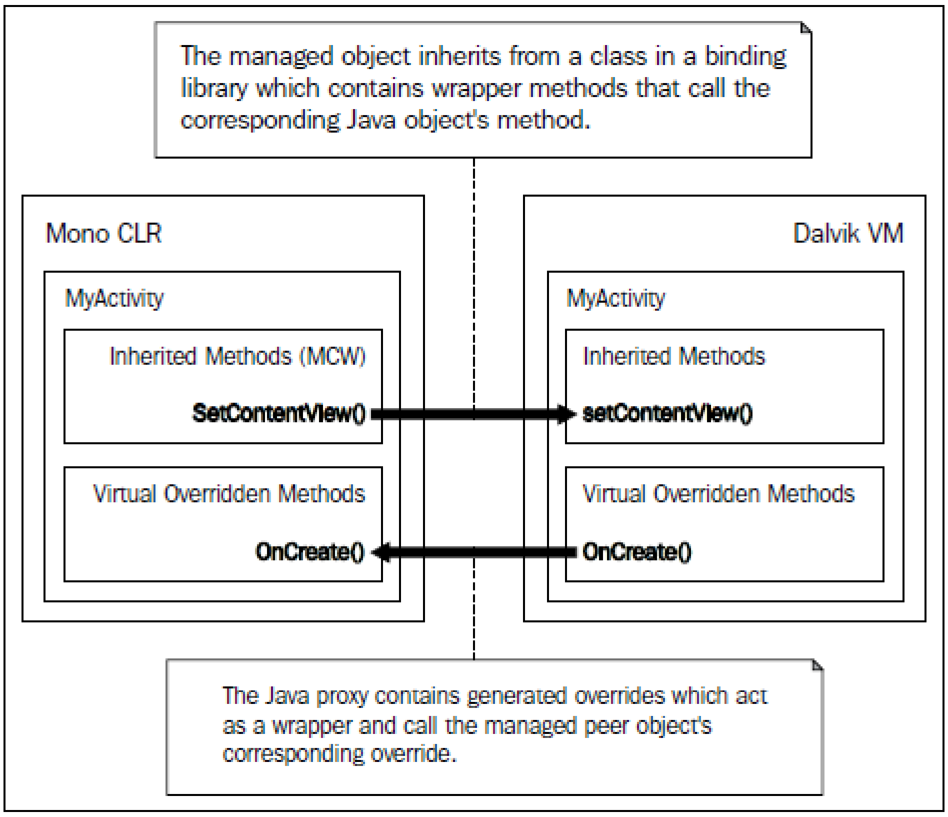

The managed object inherits from a class in a binding library which contains wrapper methods that call the corresponding Java object's method.
Mono CLR
Dalvik VM
MyActivity
Inherited Methods (MCW)
SetContentView()
Virtual Overridden Methods
OnCreate()
MyActivity
Inherited Methods
setContentView()
Virtual Overridden Methods
OnCreate()
The Java proxy contains generated overrides which act as a wrapper and call the managed peer object's corresponding override.

*Figure 20 Visualization of PeerObjects [29]*

### 7.6.3 Application Packing

Figure 21 displays the regular build and packing process as it is performed when using the native Android stack. The Android modules are compiled and packed into an .apk file. The apk file for each app contains the information necessary to run the application. This includes .dex files which are the .class files that are converted to Dalvik byte code, a binary version of AndroidManifest.xml as well as compiled resources (resources.arsc) and the un-compiled resources for the application. cf. [14, 27]

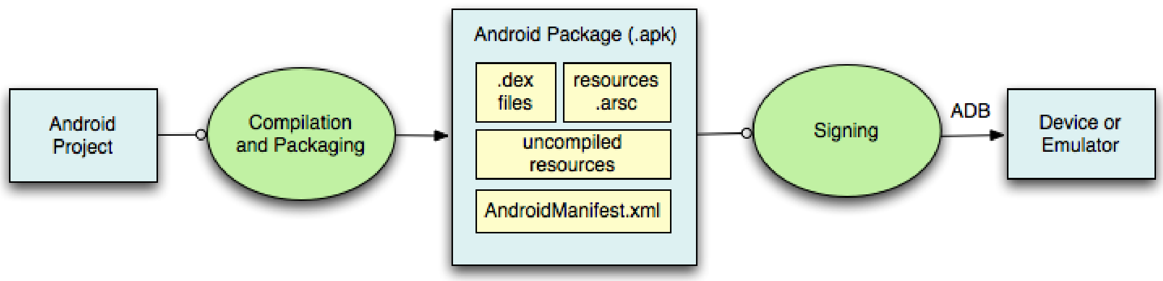


*Figure 21 Components involved in building a native Android app. [14]*

The Xamarin.Android apps follow the same structure. They contain the same parts:

- Dalvik executables (*.dex files)
- Resources
- Native libraries
- The application manifest

With the following additions:

- C# code is compiled into assemblies and stored in a top-level folder named assemblies
- Mono runtime libraries are stored along with other native libraries in the lib folder

### 7.6.4 Assemblies

The assemblies shipped with Xamarin.Android are the backbone of the whole Xamarin framework as they allow to use C# features during the development process. Xamarin.Android ships with an extended subset of Silverlight and desktop .NET assemblies. Together, these libraries provide the .NET runtime library support for developers, including namespaces such as System.IO and System.Threading. A list of assemblies that are available for the Android platform can be found in Table 3. cf. [16, 27]

*Table 3 List of Xamarin.Android Assemblies [27]*

| Assembly | Added | API Compatibility |
|---|---|---|
| **Mono.CompilerServices.SymbolWriter.dll** | 1.0 | Forcompilerwriters. |
| **Mono.Data.Sqlite.dll** | 1.0 | ADO.NET provider for SQLite. |
| **Mono.Data.Tds.dll** | 1.0 | TDS Protocol support; used for System.Data.SqlClient support within System.Data. |
| **Mono.Security.dll** | 1.0 | Cryptographic APIs. |
| **Mono.Android.dll** | 1.0 | This assembly contains the C# binding |

| | | |
|---|---|---|
| | | to the Android API. |
| **mscorlib.dll** | 1.0 | Silverlight |
| **OpenTK.dll** | 1.0 | The OpenGL/OpenAL object oriented APIs, extended to provide Android device support. |
| **System.dll** | 1.0 | Silverlight, plus types from the following namespaces:<br>• System.Collections.Specialized<br>• System.ComponentModel<br>• System.ComponentModel.Design<br>• System.Diagnostics<br>• System.IO.Compression<br>• System.Net<br>• System.Net.Cache<br>• System.Net.Mail<br>• System.Net.Mime<br>• System.Net.NetworkInformation<br>• System.Net.Security<br>• System.Net.Sockets<br>• System.Security.Authentication<br>• System.Security.Cryptography<br>• System.Timers |
| **System.Core.dll** | 1.0 | Silverlight |
| **System.Data.dll** | 1.0 | .NET 3.5 , with some functionality removed. |
| **System.Json.dll** | 1.0 | Silverlight |
| **System.Runtime.Serialization.dll** | 1.0 | Silverlight |
| **System.ServiceModel.dll** | 1.0 | WCF stack as present in Silverlight ALPHA QUALITY |
| **System.ServiceModel.Web.dll** | 1.0 | Silverlight, plus types from the following namespaces:<br>• System<br>• System.ServiceModel.Channels<br>• System.ServiceModel.Description<br>• System.ServiceModel.Web<br>ALPHA QUALITY |
| **System.Transactions.dll** | 1.0 | .NET 3.5; part of System.Data support. |
| **System.Web.Services** | 1.0 | Basic Web services from the .NET 3.5 profile, with the server features removed. |
| **System.Xml.dll** | 1.0 | .NET 3.5 |
| **System.Xml.Linq.dll** | 1.0 | .NET 3.5 |

### 7.6.5 Garbage Collection

As mentioned previously Xamarin.Android apps run in two VMs. This makes the task of garbage collection more challenging than with objects residing just in one. Through the close coupling between Xamarin.Android and Mono, the mechanisms for garbage collecting are used from Mono. There are two types of collections provided: minor collections and major collections. cf. [16, 32, 33]

- **Minor collections:** These collections are cheap and thus invoked frequently. They are responsible to collect recently allocated and dead objects and are invoked every few MB of allocations. Minor collections may be performed manually by calling: `GC.Collect(0)`

  **Major collections:** As the name suggests these collections are larger than the minor ones. This means they are more expensive in terms of CPU resources and are therefore called less frequently. They are actually only invoked when the memory for the current heap size is exhausted. Despite, it is possible to invoke them manually with `GC.Collect()` or`GC.Collect(GC.MaxGeneration).`

Within the Xamarin.Android world there are three types of objects around:

- **Managed objects:** These are any C# objects created from standard libraries. For example from the Mono runtime libraries. These are ordinary C# objects that have no connection to any classes from the Android bindings.
- **Java objects:** These are Java objects that reside in the Dalvik VM that were recreated as a part of some process, but they are not exposed to a managed object through JNI. These objects are collected as any other Java object.
- **Peer objects:** As mentioned earlier, peer objects are a managed object and Java object pair that communicates via JNI.

Peer-objects are the challenging types of objects to recycle. There are three steps necessary to perform this:

1. *All managed peers are eligible for Mono collection, meaning they are not referenced by any other managed objects. They have their JNI global reference replaced with a JNI weak reference. This allows the Dalvik VM to collect the Java peer if no other Java objects in the VM reference them. [16]*

2. *A Dalvik VM GC is invoked that allows the Java peers with weak global references to be collected. [16]*

3. *Managed peers with a JNI weak reference, as created in step 1, are evaluated. If the Java peer has been collected, then the managed peer is also collected. If the Java peer has not been collected, then it is replaced with a JNI global reference and the managed peer is not collected until a future GC. [16]*

The instance of a managed peer will live as long as it is referenced by managed code on the one side or its corresponding Java peer is referenced by any Java code.

By calling the `Dispose()` method on an object manually the connection between the peers is severed. This happens by freeing the JNI global reference which further allows each VM to collect the objects as soon as possible. cf. [15, 16, 33]

## 7.7 The Xamarin Ecosystem

Although Mono and now Xamarin focused on the core development of applications their toolset offers much more meanwhile. Beside the cross compiling technology Xamarin has established some major frameworks which are indispensable throughout a modern software development lifecycle (SDLC). Figure 22refers to a standard software development process which is also applicable for developing mobile applications. cf. [34, 35]

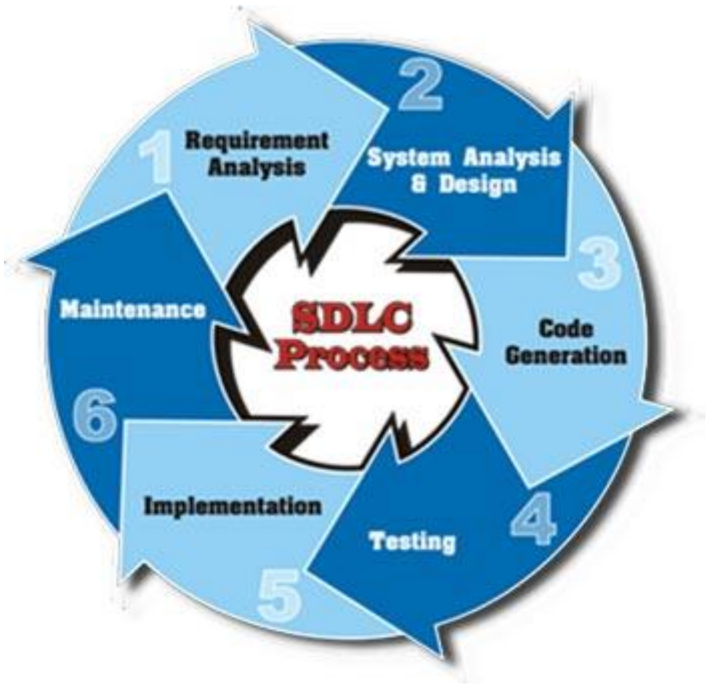


*Figure 22 Software development lifecycle [35]*

As it can be seen the SDLC is divided into six steps. Due to the fact that the requirement analysis is more conceptional than technical, the usage of Xamarin offers no additional value here, except the capabilities particular source control provide; in case of Xamarin also the Team Foundation Server by Microsoft can be used.

Depending if Xamarin.Android and Xamarin.iOS respectively is chosen all the capabilities of the visual designer can be used to design the UI. Within the Microsoft ecosystem the visual UI designers are well established and so they are within the Xamarin framework. Especially on Xamarin.iOS the concepts of storyboards are fully compatible regardless of which IDE has been taken, the Xamarin Studio or Microsoft's Visual Studio. Using Xamarin.Forms brings the disadvantage that there is no possibility to use any visual designer due to the restriction that the app has to be compiled natively to render the correct UI. Despite of this restriction, Xamarin.Forms has introduced a declarative approach of building the UI which is based on the Model View View model (MVVM) design pattern. Both approaches on both technologies allow a development based on  the Rapid Application Development (RAD) concept, so UIs can be created in a relatively short time and so can be replaced with classical wire framing tools. cf. [15, 29]

The value added of Xamarin relies on the code generation and in the previous chapters a detailed look has been given to explain the strengths and weaknesses. In particular the connection with the .NET framework and therefore the possibility to integrate every third party library written in C# gives developers a unique range of capabilities regarding the development of the actual code. The concept of portable class libraries (PCL) allows developers to use third party libraries within all applications which are capable of running C# applications. One additional major benefit of importance is also that all the productivity tools within the .NET stack can be used like code coverage, unit testing and refactoring. Several of these have been in use for more than ten years now and due to the open source initiative of Microsoft they are more reliable than ever before.

When it comes to testing a clear separation between Unit Testing and integration testing has to be made. Unit testing is rather easy to integrate through the many unit testing frameworks in the .NET ecosystem like NUnit, xUnit or the Microsoft build in MSTest engine. Therefore the business logic can be commonly tested with these frameworks. When it comes to UI testing or platform specifics, things differ. Due to the fact that the code has to be compiled to their native counterpart the common approach of unit testing is not possible because it is no .NET app anymore. But Xamarin has been built in support for integrating NUnit as a unit test engine and can therefore be integrated into the compiled app to test platform specifics. The drawback here is that there is no possibility to get real time feedback of these tests. They run inside the emulator or on the device itself so the connection to the IDE is broken and it is not capable of debugging anymore. When it comes to integration testing, Xamarin released a separate framework or even a platform to handle these scenarios which is called Xamarin Test Cloud. cf. [36]

Test Cloud offers the possibility to integrate unit tests with UI interaction with real time feedback to the development host. Therefore a small HTTP server is installed which is called from inside the application to collect feedback. Moreover, Test Cloud offers the potential to run these tests on physical devices also rather than only on the emulator. Test Cloud offers over 1000 devices tests can run on. Especially for the fragmentation of the Android platform this is essential. The drawback is that although one has subscribed to a fully commercial Xamarin license one has to pay separately for Test Cloud which starts at 1000 USD for 2 apps. cf. [36]

Implementation or the process of deployment is built in within the IDE of Xamarin, either if Xamarin Studio or Visual Studio is used. Before the app can be published to an app store, any app is runnable on the platform specific emulator. For Xamarin.iOS a Mac is needed although it can be started from a Microsoft Windows operating server. This emulator is the default one which comes with XCode. On Xamarin.Android two emulators can be used, either the default one which comes with the Java SDK or the Android player which is developed by Xamarin. When it comes to compiling the application several predefined build configurations are available to choose from, so all the cross compiling work is done behind the scenes without any need to configure your project. Publishing to an app store of choice is also built in within the IDE. Once an account on any app store is created either on Google Android orApple iOS the remaining steps for publishing are just to modify the project settings within the IDE and you can make use of the specific app store publishing features. So once the configuration is set, there are several build configurations to choose from and publishing on the app store of any of the preferred platforms is as easy as a click on a button.

In case of maintenance a software application, a common approach adds some kind of monitoring capabilities to get as much information as possible about the runtime behavior of the app. Xamarin therefore offers a platform which is named Xamarin Insights. cf. [37] With subscription to the commercial offer it is possible to access a huge amount of information through a web platform with adding code into the application. This approach is best known in the realization of Google’s analytics platform which offers similar functionality for web applications.

To sum up it can be said that Xamarin offers much more possibilities for a software developer than just a development framework. Beside the cross compiling technology, Xamarin supports all the major steps within the SDLC especially in regard to testing and monitoring.

# 8 Conclusion

Back in 2001 the primary goal of the Mono framework was to support the development of cross platform desktop and web applications, whereas the challenge today is to support devices beyond the desktop technology stack like smartphones, tablets and wearables.

So Migual de Icaza and members of the original Mono team founded Xamarin Inc. The primary goal was to bring the Microsoft .NET framework based on the C# language specifications to all of the major smartphone operating systems which are Apple iOS, Google Android and Windows Phone. Other than Appcelerator Titanium or Hybrid frameworks like the PhoneGap framework, Xamarin is based on a cross compiling technology as the major driver behind their technology stack. Through this technology it is possible to produce native applications with full support for the underlying APIs of the particular platform. Therefore in case of an Android application, Xamarin makes use of the established Mono runtime and uses JIT compiling. In case of iOS it is based on AOT compiling which produces a fully binary compatible application to their Objective-C counterpart.

This concept has some major advantages of developing mobile applications over traditional or other cross platform frameworks. First of all using one single codebase where the whole set of features of the C# language can be used offers all the same possibilities for mobile application development like for classical .NET development. This meansthat given language features like LINQ, ADO.NET or Lambda Expressions access to 3rd party libraries and well established tools like unit test frameworks, productivity tools and all C# compliant integrated development environments. Additionally,one gets a fully binary compatible application targeting the required platform which performs like their native counterpart or in some cases even faster. When using Xamarin forms as the UI framework it is sometimes easier to use the built-in components rather than the platform specific ones. Particularly if a developer is already familiar with the .NET framework and C# it is easy to build mobile applications.

But there are also some drawbacks especially if there are highly specific requirements for the UI or user interaction. Even if Xamarin.Android or Xamarin.iOSare used, which render natively, it will be even worse when using Xamarin.Forms for the graphical parts.

As they implement a new layer over the native environment there isno real control of what is generated as the final code that will be run on the device. That can lead to a lot of workarounds within the codebase. Another big point is the community around Xamarin. Although it is fast growing and established in the meanwhile, it is not comparable to the communities or libraries of their native counterparts. Especially when it comes to adapt to 3rd libraries into the Xamarin codebase like for example the OpenEar framework for Speech to text capabilities.

Finally, it has to be highlighted that beyond the software development process Xamarin also supports other areas of the classical software development life cycle. With Xamarin TestCloud they offer their own test platform like Calabash and with Xamarin Insights a fully-fledged monitoring solution is also available. Minor additions are the Xamarin Android player which is a, totally from scratch, rewritten Android emulator which performs faster than the native emulator which comes with the Java SDK.

Although it may seem that Xamarin is still relatively new on the market, it has recently celebrated its 4th birthday. But with the strong relationship to Microsoft they are on the way to becoming an emerging industry standard for enterprise mobile development. It is pronounced that Xamarin has humongous potential but that is yet to become truly visible. It is a powerful platform by default. With all the support from Microsoft and other key players on the market like SAP or Salesforce, it has a great chance to become a standard solution for cross-platform development very soon, as it is turned into an important component of the newest Visual Studio 2015 release.

# 9 References

[1] American Dialect Society, “App 2010 Word of the Year, as voted by American Dialect Society,” Pittsburgh, January 7th, 2010.

[2] P. Christenssons, *Tech Terms.* Available: http://techterms.com (2015, Jul. 26).

[3] Danny Dig and Ralph Johnson, “How do APIs evolve? A Story of refactoring,” (en), *Journal of the software maintenance ans evolution: Research and Practice*, 2006.

[4] Wikipedia, *Smartphone.* Available: https://de.wikipedia.org/w/index.php?oldid=144330788 (2015, Jul. 26).

[5] Janessa Rivera and Rob van der Meulen, *Gartner Says By 2018, More Than 50 Percent of Users Will Use a Tablet or Smartphone First for All Online Activities.* Available: http://www.gartner.com/newsroom/id/2939217 (2015, Jul. 26).

[6] statista.com, *Global smartphone shipments forecast from 2010 to 2019 (in million units).* Available: http://www.statista.com/statistics/263441/global-smartphone-shipments-forecast/ (2015, Aug. 04).

[7] IDC, *IDC: Smartphone OS Market Share.* Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp (2015, Jul. 26).

[8] Benedict Evans, “Mobile is eating the world,” Andreessen Horowitz Inc.

[9] Xamarin Inc, “Key Approaches for Mobile Success,”

[10] Bonnie Boardman, “No app for that? Write one!,” (en), *Industrial Engineer*, vol. 44, no. 3, pp. 44–48, 2012.

[11] reasearch2guidance, “Cross-Platform Tool Benchmarking 2014: Find the right tool for your app porject,”

[12] Gartner Inc, “Hype Cycle for Mobile Applications and Development 2014,”

[13] Dzone Research, “2014 Guide to Mobile Development,”

[14] Google Inc, *Building and Running Overview.* Available: https://developer.android.com/tools/building/index.html (2015, Aug. 04).

[15] Charles Petzold, *Creating Mobile Apps with Xamarin.Forms*. Redmond: Microsoft Press, 2015.

[16] M. Reynolds, *Xamarin Essentials*: Packt Publishing, 2014.

[17] J. Dickson, “Xamarin Mobile Development,” Grand Valley State University, Michigan, 2013.

[18] Manuel Palmieri, Inderjeet Singh, and Antonio Cichetti, Eds, *Comparison of Cross-Platform Mobile Development Tools*: IEEE, 2012.

[19] Jim Cowart, *Pros and Cons of the Top 5 Cross-Platform Tools.* Available: http://www.developereconomics.com/pros-cons-top-5-cross-platform-tools/.

[20] Stuart Parkerson, *Appcelerator Launches New Cloud Based Service for Building APIs.* Available: https://appdevelopermagazine.com/2612/2015/4/6/Appcelerator-Launches-New-Cloud-Based-Service-for-Building-APIs/ (2018, Aug. 04).

[21] Adobe Inc, *Take the pain out of developing mobile apps.* Available: https://build.phonegap.com/ (2015, Aug. 05).

[22] Prithal Shah, and Jacques Bourhis et al, *The Development of Mobile Applications using HTML5 and PhoneGap* on Intel® Architecture-Based Platforms.* Available: https://software.intel.com/en-us/articles/the-development-of-mobile-applications-using-html5-and-phonegap-on-intel-architecture-based (2015, Aug. 05).

[23] J. M. Wargo, *PhoneGap essentials: Building cross-platform mobile apps / John M. Wargo*. Upper Saddle River, NJ: Addison-Wesley, 2012.

[24] Adobe Inc, *PhoneGap.* Available: http://phonegap.com/ (2015, Aug. 05).

[25] Community, *Qt weak points and disadvantages.* Available: http://www.qtcentre.org/threads/26024-qt-weak-points-and-disadvantages.

[26] Qt, *Qt Main Page.* Available: http://www.qt.io/.

[27] Xamarin Inc, *Architecture.* Available: http://developer.xamarin.com/guides/android/under_the_hood/architecture/ (2015, Jul. 23).

[28] Xamarin Inc, *Introduction to Portable Class Libraries: Creating Reusable Cross Platform Library Projects.* Available: http://developer.xamarin.com/guides/cross-platform/application_fundamentals/pcl/introduction_to_portable_class_libraries/ (2015, Jul. 22).

[29] M. Reynolds, *Xamarin Essentials: Learn how to efficiently develop Android and iOS apps for deployment using the Xamarin platform*. Birmingham, UK: Packt Publishing, 2014.

[30] S. Olson, *Professional cross-platform mobile development in C#*. Hoboken, N.J.: Wiley, 2012.

[31] Oracle Inc, *Java Native Interface.* Available: http://docs.oracle.com/javase/7/docs/technotes/guides/jni/ (2015, Aug. 04).

[32] Mono-Project, *Generational Garbage Collection.* Available: http://www.mono-project.com/docs/advanced/garbage-collector/sgen/.

[33] Xamarin Inc, *Garbage Collection.* Available: http://developer.xamarin.com/guides/android/advanced_topics/garbage_collection/ (2015, Aug. 04).

[34] J. Lewis, *SDLC 100 success secrets: Software development life cycle (SDLC) 100 most asked questions, SDLC methodologies, tools, process and business models*. [United States?]: J. Lewis, 2008.

[35] Encoding Enhancers, *Software Development Life Cycle* (2015, Aug. 01).

[36] Xamarin Inc, *Xamarin Test Cloud.* Available: http://xamarin.com/test-cloud (2015, Aug. 01).

[37] Xamarin Inc, *Xamarin Insights.* Available: https://xamarin.com/insights (2015, Aug. 01).