CHALMERS

Evaluating the Potential of Developing Cross-Platform Mobile Applications

Master of Science Thesis

Mikhail Yakubovich

Department of Computer Science & Engineering
Computer Systems and Networks
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2013

Evaluating the Potential of Developing Cross-Platform Mobile Applications

Mikhail Yakubovich

Examiner: Sally McKee

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

[Cover: an explanatory caption for the (possible) cover picture with page reference to detailed information in this essay.]

Department of Computer Science and Engineering
Göteborg, Sweden September 2013

Abstract

The mobile market is rapidly growing along with the constant increase in handheld devices per household. This trend affects IT consultancy companies whose clients often request applications for mobile operating systems such as iOS, Android and Windows Phone. Such companies strive to decrease the development time and present the customer with a satisfying solution that is delivered on time. In order to decrease the time-to-market and thus the cost of the final product, companies seek to develop applications independent of the target mobile operating system by using a cross-platform approach. This approach can eliminate the increased effort that normally comes with developing a separate application for each mobile operating system, providing a more efficient solution.

This master thesis investigates two common cross-platform tools (CPTs), namely, Xamarin and Unity. In addition we examine the approach of sharing a core code base of a mobile application between multiple platforms. The purpose is to determine which alternative CPT is more suitable for the development of portable mobile applications. The Analytic Hierarchy Process methodology is used to give a high-level quality analysis of the CPTs and to select the most suitable candidate to use to develop the target application.

Acknowledgments

First of all, I would like to express my gratitude to Johan Sanneblad, Consultant Manager at HiQ for his supervision during this master thesis. I also feel grateful for all the help, feedback and guidance given by my examiner a prof Sally McKee. Last, but not least, I thank my family for all the support given while I was working on this project.

Table of Contents

1. Introduction

As the number of devices per household steadily increases, the mobile market continues to grow at a rapid pace. IT companies are trying to keep up with this development as their clients request mobile applications that have to be produced quickly but still can deliver high quality and customer satisfaction. With the introduction of cross-platform mobile app development tools (CPTs), companies are able to cut costs during the development process and at the same time deliver the product faster to the client.

Targeting more than one platform normally requires developing a corresponding application for each mobile operating system. However, this approach means that the development time and hence the cost of the product will increase. A cross-platform approach, on the other hand, helps to solve this problem by developing a single code base that supports multiple platforms. Another benefit of CPTs is that they allow changes to be made faster to portable mobile applications, as only one single code base needs to be modified.

With the high demand for cross-platform solutions, it is crucial to find the efficient Cross Platform Tool that can provide the best results for the type of application that is to be developed. This master thesis therefore investigates two common CPTs, Xamarin and Unity. It also analyzes a third approach in which the mobile applications share a core code base. The purpose of this thesis is to determine which of these three alternatives is most suitable for the development of a given standard application. To this end we use the Analytic Hierarchy Process methodology to perform a high-level quality analysis of the CPTs.

To define the problem at hand, first I define a number of research questions:

- Which approach is the most suitable for cross-platform mobile development among the evaluated tools and methods Xamarin, Unity and the alternative of sharing a core codebase?
- Does the answer differ when criteria such as security, reusability, functionality and documentation are evaluated?

1.1 Scope

Since a complete analysis of several software frameworks is not feasible under the timing constraints that apply to this master thesis work, it is crucial to define the scope of the evaluation. The target mobile operating systems are limited to iOS and Android OS. The tools and methods that we evaluate are therefore limited to three approaches: Xamarin, Unity and the alternative of sharing a core code base of the mobile application. The code base, written in C++, can be part of iOS as well as Android application.

We developed the proof-of-concept application for the purpose of the evaluation using each of the alternatives. Since most modern mobile software applications utilize networking and database access functionalities, the proof-of-concept application conforms to the following functional requirements that represent the target application domain:

- **Networking**: the application must download and parse an RSS;
- **Data Access**: the data must be persistently stored in the application bundle;
- **User Interface**: the application presents the stored data in a table view controller;
- **Security**: the application provides secured data transfer from the server.

The evaluation criteria used in AHP methodology:

- **Documentation** the ease of obtaining information about functionality, API, best practices, code reuse, etc.
- **Functionality** with respect to networking, data access and user interface.
- **Security** with respect to language and overall application security.
- **Reusability** the ease or difficulty of reusing the application logic/code for different mobile platforms.

1.2 Methodology

We first investigate approaches for cross-platform mobile development and previous efforts made to compare the various techniques. I then identify a methodology that could assess the different approaches given a certain set of attributes. I compare several methodologies for a decision-making problem and choose an Analytic Hierarchy Process methodology for evaluation and selection of the best candidate tool/method.

1.3 Motivation

This master thesis work has taken place at HiQ, an IT consultancy firm specializing in communications, software development and business-critical IT. The aim of this thesis is to increase the knowledge of cross-platform mobile development that has great potential for the efficient development of future Android and iOS projects. The results of the thesis work, being conducted with HiQ’s needs in mind, are equally relevant to other companies seeking to provide similar expertise to its clients.

2. Background

During the last few years, a new set of platforms for mobile development has appeared that offers an alternative to the standard way of developing mobile applications. Developers are no longer limited to using Objective-C or Java in order to create iOS/Android applications but can instead make use of solutions where a single language is used to generate native applications for iOS and Android.

We first compare and contrast the two products we evaluate Xamarin and Unity with an approach that shares a core code base, that relying on the programmer to implement a portable core by using basic language and library functionality. The most important distinguishing features of each approach are:

- how networking communication is performed (including how web service technologies are integrated)
- how databases are accessed
- how user interactions is managed

2.1 Sharing a core code base

Another method is to develop two separate applications using native SDKs, one for iOS and one for Android, where both applications share a single code base to perform networking and data access logic.

Although Objective-C is the premier language for iOS development, it is common practice to use C/C++ for CPU-intensive computations to achieve better performance enhancements. Similarly, although Java is the preferred language for Android development, C++ code can still be compiled using the Native Development Kit (NDK) [7].

The use of C++ code in Objective-C projects is facilitated by the introduction of the new primary clang compiler for Apple. Even before clang, when GCC was the main compiler, developers often incorporated C++ libraries in their iOS projects [6].

Most Android applications do not need to use NDK, and, in fact, Google recommends against using it. However, typical candidates for the NDK are CPU-intensive operations that do not require a lot of memory. The drawback of using NDK is the added application complexity, which should be balanced against any possible performance benefits [7].

Several C++ libraries can be used for developing a shared code base. Boost and POCO libraries aggregate functionality important to the cross-platform applications targeted in this work.
POCO is a collection of open source C++ libraries similar to the .NET Framework, Apple's Cocoa framework or the Java Class Library. POCO provides modern ANSI/ISO Standard C++ and

uses C++ Standard Library [11]. The main library components of POCO have been ported to both Android and iOS to allow cross-platform features and platform abstraction [12]. Figure 1.2 gives an overview of the POCO library collection.

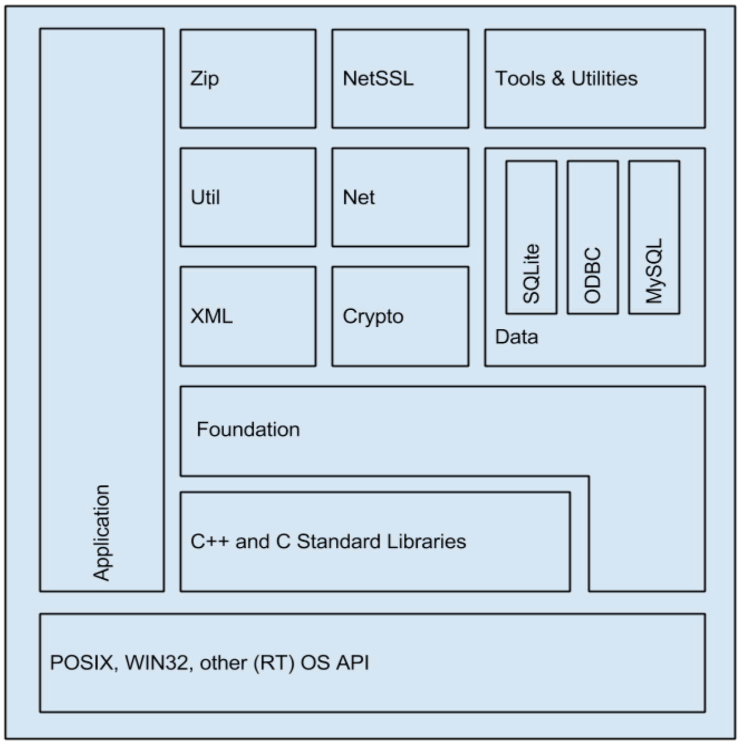


Figure 1.2 – An Overview of the Poco C++ Libraries

Networking

The .NET library of POCO enables the development of network-based applications. Common network protocols such as TCP, UDP, ICMP, raw and multicast sockets are available. Furthermore, POCO provides secure sockets for use with the NetSSL. Additional features on the client side include FTP to send and receive files and SMTP to receive mail from a POP3 server [17].

Data Access

POCO Data provides a database abstraction layer so that users can store and retrieve data. The database abstraction layer works over different SQL databases, including SQLite, which is the database used on both iOS and Android [20].

For the purpose of this thesis, we selected POCO library in place of Boost because it provides all the functionalities that are needed in the types of applications we target. Additionally, Boost has not yet been regression-tested for Android and iOS, although techniques exist to build Boost for these platforms.

2.2 Xamarin

Xamarin applications are written in C# and use the base class library (BCL) of .NET to enable common functionality such as database interaction and file operations. Xamarin offers a run-time environment and a set of libraries that work across iOS, Android and Windows Phone mobile platforms. Up to 90% of the code can be shared when written for the three major mobile platforms. The user interface remains native to each environment while the business logic of the application is shared. In addition to C#, third party code can also be used, as Xamarin can directly invoke Objective-C, Java, C and C++ libraries.

Xamarin offers two commercial products, Xamarin.iOS and Xamarin.Android, which are built on top of Mono the open-source version of the .NET framework, which is a well-established platform available for common operating systems including Linux, Unix, FreeBSD and Mac OSX [5].

Xamarin.iOS uses an Ahead-of-Time (AOT) compiler that directly produces native ARM assembly code. Xamarain.Android, on the other hand, compiles down to Intermediate Language (IL) that, in turn, is Just-in-Time (JIT) compiled to native assembly at the time when the application starts. Both commercial products fully utilize a run-time that handles memory allocation, garbage collection and underlying platform interoperability [5].

Networking

Another important feature that Xamarin provides is the ability to integrate various web service technologies, an increasingly common requirement in mobile development. Xamarin provides a wide range of support for these technologies, including built-in and third-party APIs to offer integration with SOAP, RESTful and WCF services [16].

Data Access

Xamarin enables data access functionality using either SQLite or ADO.NET libraries. The SQLite database is available both in Android and iOS but the native methods to access the database differ. In Xamarin.iOS and Xamarin.Android the native APIs offers ability to share code between the platforms other than through SQL queries. To achieve 100% code sharing, the ADO.NET library can be used from both Xamarin.iOS and Xamarin.Android [19].

User Interface

Xamarin gives full access to all native user interface APIs. When developing business logic with Xamarin Studio, the UI can be created using Apple's IDE XCode and Interface Builder. Native iOS applications can be created with Xamarin.iOS, which includes the same UI controls as when creating an application in Objective-C with XCode. For iOS UI development, the MonoTouch.UIKit API can be utilized. Android provides Android.Views, which takes advantage of the Xamarin UI designer. Mono.Android.dll is the assembly that contains the C# binding to the Android API, while MonoTouch.dll contains the C# binding to the CocoaTouch API for iOS [21]. In addition, Mono also supports security through Mono.Security.dll and System.dll with System.Net.Security and System.Security.Cryptography [22].

2.3 Unity

Unity is a game development platform specialized for efficient multiplatform publishing on Xbox, PC, Android and iOS. Unity provides a rendering engine for the creation of interactive 3D games. Unity further supports development for standard desktop environments as well as for game consoles such as Xbox 360, PlayStation 3 and Wii U [13].

Networking and Data Access

Like Xamarin, Unity uses an open-source version of the Mono .NET framework. .NET class libraries can be used in scripts written for Unity. This enables networking and data access capabilities and a large range of web services [18]. The Mono run-time also offers security capabilities through libraries such as Mono.Security.dll and System.dll, as is the case for Xamarin [22].

User Interface

NGUI is one available user interface system for Unity. It also functions as an event notification framework for Unity, with features such as full inspector integration and support for lighting, refraction, clipped panels and a built-in localization system. NGUI is written in C# and provides full support for Android, iOS and Flash. Template widgets can be created with UI components such as buttons, input fields [14].

3. Software Evaluation Methodologies

This chapter discusses methods used for decision making when different alternatives are present. The most popular multi-criteria decision making methods are: the Weighted Sum Model, the Weighted Product Model and Analytic Hierarchy Process. For the purpose of this analysis, we choose the Analytic Hierarchy Process method because of its ability to decompose a problem into its constituents parts and the possibility to check for inconsistencies. The selected technique, the Analytic Hierarchy Process, is then used to give a high-level quality analysis of the CPTs. This facilitated the decision in finding the best candidate approach for the target application domain.

3.1 Multi-Criteria Decision Analysis

Multi-criteria decision analysis (MCDA) is a method that helps to ease the process of decision-making. It can be utilized when the problem can be divided into smaller parts that are easier to understand and analyze. Multi-criteria decision-making (MCDM) is often applied in real-life situations by governments and industries to evaluate conflicting criteria. In the simplest case, the criteria can be represented in the same unit, such as cost. A more realistic scenario expresses different criteria in a number of unique dimensions such as time, cost, weight and impact. The various dimensions present a complex problem that can be helped by a set of MCDM methods [1].

3.2 Analytic Hierarchy Process

The analytic hierarchy process (AHP) is a MCDM method. AHP requires establishing of a goal and a hierarchy of criteria for the problem of which a decision needs to be made. AHP breaks down the stated problem into subcomponents with respect to the overall goal. The specified hierarchy of criteria is subsequently used to evaluate alternative solutions and finally to reach a decision of the preferred option [2].

AHP can be used in many fields including software engineering. The top-level decision criteria for a typical problem within this area could, for example, be functionality and reusability, and the next level could evaluate these with respect to attributes like UI, DB, etc. The AHP can in this case give a high-level quality analysis of the software system that is being evaluated [3].

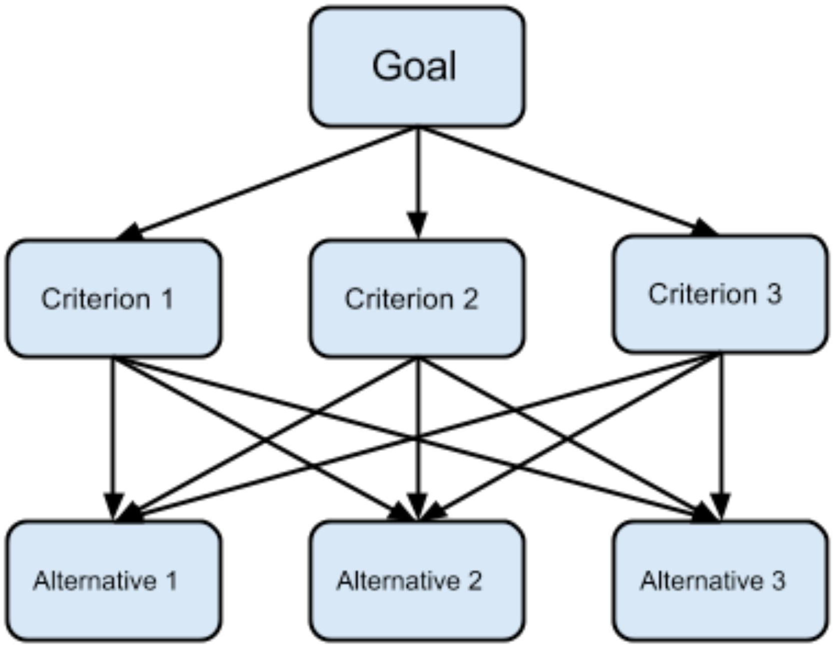


Figure 3.1 - The Hierarchy of AHP

The relevant data are generated via a set of pairwise comparisons. First a user derives weights needed for the decision criteria where criteria, with higher significance have larger weights. Each criterion will thus have weights assigned to it indicating its relative level of importance. Table 3.1 shows the relative importance scale ranging from one to nine. In the case where criterion A is compared to criterion B and A is strongly more important than B, a weight of five is assigned to the pairwise comparison [2].

Definition	**Intensity**
Equal importance	1
Moderately more important	3
Strongly more important	5
Very strongly more important	7
Extremely more important	9
Intermediate values	2, 4, 6, 8

Table 3.1 The Scale for Pairwise Comparison

The general strategy of the AHP methodology is:

- to determine the relative weights of the attributes,
- to compare the alternatives on each attribute and
- to aggregate weights and to produce the final quality values.

In the following chapters, the methodology will be described in detail in relation to the cross-platform tools under evaluation.

4. Selection of Evaluation Criteria

In this section we define evaluation criteria for the decision-making problem. We decompose top-level criteria into sub-criteria, each of which I analyze independently. After the hierarchy is built, we compare the criteria on the same hierarchy level with respect to their impact on the higher-level elements. The higher-level element for the top-level criteria is the goal and the higher-level element for sub criteria is their corresponding top-level criterion.

4.1 Managing Stakeholders

Deciding how to weight the criteria is nontrivial. Many stakeholders involved at various stages in a project cycle might have very different opinions about the importance of a particular criterion. A range of people involved in the project can represent such stakeholders; they are often developers, project managers, architects and customers. To manage the diverse views from these stakeholders, a person with leadership must be involved in the evaluation process.

We chose a team of experienced mobile developers from HiQ to take part in the decision-making of this thesis. Three senior developers played the role of stakeholders for the project. The personal experience of the author of this thesis was additionally used to assign weights that were acceptable to all parties.

4.2 Evaluation Criteria

Documentation, Functionality, Security, and Reusability are the top-level evaluation criteria of the decision-making problem.

With respect to documentation, we examine how easy it is to obtain information about the functionality, and whether the API is clearly described. We assign to functionality the sub-attributes networking, data access and user interface. First of all, since the stated problem concerns cross-platform development, it is important to find out whether the evaluated CPTs provide the functionality needed to implement a portable application. The third criterion, Security, refers to the resistance of the application towards hijacking, attacks, and other security breaches. The last attribute, Reusability, concerns the ease of using the application code on another mobile platform with respect to the sub-attributes networking, data access and user interface.

By decomposing the functionality and reusability, it is possible to achieve a more detailed analysis of the specific criterion. Unity, for instance, may satisfy a certain sub-attribute while giving poor results for another. At the same time, Xamarin may provide satisfying results for other second-level criteria.

Figure 4.1 shows a summary of the AHP setup for the given problem.

Problem	
To find the most optimal alternative approach for the development of cross-platform mobile applications	
Alternatives	
Xamarin, Unity, sharing a core code base	
Criteria	
Documentation	
Functionality	Networking
	Data Access
	User Interface
Security	
Reusability	Networking
	Data Access
	User Interface

Figure 4.1 - The AHP Setup

After the problem is set up, the relative weights of each of the comparison attributes must be determined.

4.3 Weighting the Evaluation Criteria

Figure 4.2 shows the relative comparison of the top-level criteria. We evaluate the relative importance of the elements with respect to the goal, which is to find the optimal alternative approach for the development of cross-platform mobile applications.

	Documentation	**Functionality**	**Security**	**Reusability**
Documentation	1	1 / 7	1 / 5	3
Functionality	7	1	3	9
Security	5	1 / 3	1	5
Reusability	1 / 3	1 / 9	1 / 5	1

Figure 4.2 - The Relative Importance Table for the Top-level Attributes

All elements that are part of the main diagonal (from the upper-left cell to finishing at the lower-right cell) are assigned the value 1 since each attribute is relatively compared to itself. By definition, the values of cells above the main diagonal in the table are mathematical inverses of cells that lie below the main diagonal. As an example, because security is strongly more important than reusability, the value of 5 is assigned. The relative importance of reusability versus security, on the other hand, got the value of 1/5, which is the mathematical inverse of 5 [2].

Next we compare second-level attributes. Figure 4.3 shows the relative comparison of the sub-attributes Networking, Data Access and User Interface for the functionality criterion.

Functionality	Networking	Data Access	User Interface
Networking	1	7	5
Data Access	1 / 7	1	1 / 3
User Interface	1 / 5	3	1

Figure 4.3 - The Relative Importance Table for the Functionality sub-attributes

Figure 4.4 shows similar comparisons of sub-attributes as Figure 4.3, but with respect to the reusability criterion.

Reusability	Networking	Data Access	User Interface
Networking	1	1	1 / 3
Data Access	1	1	1 / 3
User Interface	3	3	1

Figure 4.4 - The Relative Importance Table for the Reusability sub-attributes

4.4 Calculating the Priority Vector for the Criteria

The next step is to calculate the priority vector by summing each column and then dividing the relative importance value by the sum of the column. In the last step we calculate the average of each row. Figure 4.5 shows the first step of calculating the priority vector. Figure 4.6 displays the results after the second step where each entry of the table has been divided by its column sum. This generates the priority vector that is summed up for each top-level attribute in Figure 4.7. In relation to the below calculation, each value of the priority vector will generate results between 0 and 1. Likewise, the sum of all values in any priority vector equals to 1 [2].

	Documentation	Functionality	Security	Reusability
Documentation	1.00	0.14	0.20	3.00
Functionality	7.00	1.00	3.00	9.00
Security	5.00	0.33	1.00	5.00
Reusability	0.33	0.11	0.20	1.00
Sum	13.33	1.59	4.40	18.00

Figure 4.5 - Calculating the Priority Vector: Step One

Documentation	0.08	0.09	0.05	0.17
Functionality	0.53	0.63	0.68	0.50
Security	0.38	0.21	0.23	0.28
Reusability	0.03	0.07	0.05	0.06
Pr. Vector	0.09	0.58	0.27	0.05

Figure 4.6 - Calculating the Priority Vector: Step Two

Documentation	0.09
Functionality	0.58
Security	0.27
Reusability	0.05

Figure 4.7 - AHP Priority Vectors for the Top-level Attributes

Calculating the Functionality sub-attributes priority vector proceeds similarly. Figure 4.8 and 4.9 present the results from step one and two. Figure 4.9 gives the final priority vector of the functionality second-level attributes.

	Networking	Data Access	User Interface
Networking	1.00	7.00	5.00
Data Access	0.14	1.00	0.33
User Interface	0.20	3.00	1.00
Sum	1.34	11.00	6.33

Figure 4.8 - Calculating the Priority Vector: Step One

Networking	0.74	0.64	0.79
Data Access	0.11	0.09	0.05
User Interface	0.15	0.27	0.16
Pr. Vector	0.72	0.08	0.19

Figure 4.9 - Calculating the Priority Vector: Step Two

Functionality	
Networking	0.72
Data Access	0.08
User Interface	0.19

Figure 4.10 - AHP Priority Vectors of the Functionality sub-attributes

We again follow the same approach, this time for the Reusability top-level criterion. The sub-attributes are again set to Networking, Data Access and User Interface.

	Networking	Data Access	User Interface
Networking	1.00	1.00	0.33
Data Access	1.00	1.00	0.33
User Interface	3.00	3.00	1.00
Sum	5.00	5.00	1.67

Figure 4.11 - Calculating the Priority Vector: Step One

Networking	0.20	0.20	0.20
Data Access	0.20	0.20	0.20
User Interface	0.60	0.60	0.60
Pr. Vector	0.20	0.20	0.60

Figure 4.12 - Calculating the Priority Vector: Step Two

Reusability	
Networking	0.20
Data Access	0.20
User Interface	0.60

Figure 4.13 - AHP Priority Vectors of the Reusability sub-attributes

At this point, we determined the relative weights and performed a pairwise comparison for the top and sub-level attributes. In the next chapter, we compare the alternative on each attribute. Finally, we aggregate weights and produce the final quality values.

5. Evaluation and Results

This chapter evaluates the three chosen approaches Sharing a core code base, Xamarin and Unity following the presented methodology. More specifically, it gives an analysis of which tool is most suitable for each evaluation criterion to facilitate the decision making of selecting the best candidate cross-platform approach for the target application domain. We base the comparison results on the experience gained while working on the proof-of-concept applications for each approach.

5.1 Documentation

The results from the three approaches for documentation are listed in Figure 5.1. We can here see that Xamarin received the best result with a ranking vector of 0.64. A brief discussion of each of the cross-platform methods is given below to explain the results.

Documentation	Xamarin	Unity	Sharing a core code base	Ranking Vector
Xamarin	1	3	7	0.64
Unity	1 / 3	1	5	0.28
Sharing a core code base	1 / 7	1 / 5	1	0.07

Figure 5.1 – Results for Documentation

Sharing a core code base
Despite the fact that the Poco library provides adequate documentation, it is very difficult to find out-of-the-box solutions using the Internet. Furthermore, the community is not big, which means that it may be time consuming to find a solution to a specific problem while developing. The core C++ library subsequently gets a rather dissatisfying score for the criterion of documentation and accessibility with a very low ranking vector of 0.07.

Xamarin
Xamarin is a commercial product with great documentation, API reference and also provides extensive guides and tutorials to the developer where sample applications and videos are included. As mentioned above, Xamarin received the best results for the given criterion with a ranking vector of 0.64.

Unity
The cross-platform tool Unity is very well documented and provides best practices and code samples to the developer. What decreases the ranking for this approach is NGUI, the UI system for Unity, for which the documentation lacks many out-of-the box solutions and samples. The ranking vector for Unity therefore lands on a score of 0.28.

5.2 Functionality

This subsection compares sharing a core code base, Xamarin and Unity on the Functionality criterion.

Networking

The next set of criteria up for evaluation is functionality with the sub-criterion networking. The results are presented in Figure 5.2 where we can see that Xamarin and Unity received the same ranking vector of 0.43.

Networking - Functionality	Xamarin	Unity	Sharing a core code base	
Xamarin	1	1	3	0.43
Unity	1	1	3	0.43
Sharing a core code base	1 / 3	1 / 3	1	0.14

Figure 5.2 – The Results for Sub-Attribute Networking in Relation to Functionality

Sharing a core code base
Despite the fact that the POCO library provides an adequate API for development of network-based applications, it obtained less satisfying results. This was due to the experienced difficulty during setup when building the necessary POCO networking libraries as well as during linking of the resulting libraries for the iOS and Android applications. The result for sharing a core code base only received a ranking vector of 0.14.

Xamarin
Xamarin obtained the highest score in the pair-wise comparison for the networking attribute due to Xamarin's Networking API. By looking at the API, it is clear that it would be fully feasible and simple to implement the desired networking functionality specified for the demo application. As mentioned in Chapter 2, integrating web services into mobile applications is a common scenario and Xamarin provides support for this feature, thereby increasing the possibilities for network communication and data exchange.

Unity
As can be seen in table 6.2, Unity obtained the same ranking vector as Xamarin. This can be explained by the fact that both tools use Mono assemblies for implementing networking functionality as described in chapter 2. The ranking vector for Unity regarding a functionality of the attribute networking was therefore also given a score of 0.43.

Data Access

Another evaluated second-level attribute for functionality is data access. The obtained ranking vectors for this criterion is illustrated in Figure 5.3 where we can see that Xamarin and Unity again are given the same score and share the top position.

Data Access - Functionality	Xamarin	Unity	Sharing a core code base	
Xamarin	1	1	5	**0.45**
Unity	1	1	5	**0.45**
Sharing a core code base	1 / 5	1 / 5	1	**0.09**

Figure 5.3 - The Results for Sub-Attribute Data Access in Terms of Functionality

Sharing a core code base
Although it is feasible to develop a Data access abstraction layer for iOS and Android using the API of POCO Data, it is not 100% possible to share this logic between two platforms and hence to write 100% cross-platform code. This is due to the fact that sandboxing mechanisms on iOS and Android are different. Furthermore, database file access logic will be different on iOS and Android since the files are stored in different locations on the file system. The ranking vector for data access is thus rather low for the core C++ method that obtained a result of 0.09.

Xamarin
As mentioned in the background chapter, data access functionality for Xamarin can be implemented using either SQLite or ADO.NET libraries where the latter allows 100% code sharing between Android and iOS. This is the reason why Xamarin receives the highest score for data access in terms of functionality. To be more exact, a ranking vector of 0.45 was obtained for this cross-platform technique.

Unity
To make use of an SQLite database on a Unity project that targets both iOS and Android can be quite a challenging task that requires deep knowledge of the Unity environment. One will need to develop a Unity plugin for iOS and Android in order to enable this type of functionality. As an alternative, a commercial plugin can be purchased that will provide an easy-to-integrate solution. A score of 0.30 was given to Unity for the data access attribute.

User Interface

The second-level attribute user interface for functionality was also evaluated and obtained the scores that are displayed in Figure 5.4. As can be seen, the sharing a core code base approach received the highest ranking vector of 0.63 as it presented the best UI capabilities.

User Interface - Functionality	Xamarin	Unity	Sharing a core code base	
Xamarin	1	3	1 / 3	**0.26**
Unity	1 / 3	1	1 / 5	**0.11**
Sharing a core code base	3	5	1	**0.63**

Figure 5.4 – Ranking Vectors for the Attribute User Interface

Sharing a core code base
For the sharing a core code base, the actual UI is developed using native libraries and SDKs. Only the core functionality is shared while the rest is left for the native environment. This approach got the highest ranking for the user interface criterion because it assumes that all UI logic is developed using native iOS and Android frameworks. In fact, only networking and data access logic is shared among the platforms that can be provided by the C++ Library. The use of native APIs for UI is always the most sufficient way in terms of functionality.

Xamarin
Xamarin lets the developer create applications with a native look by using the same UI controls as used in standard iOS/Android development. However, one main obstacle will always be the delay that occurs when a vendor such as Google or Apple adds new functionality. The cross-platform framework then needs to be updated to include the new feature into its API, and this process can take time.

Unity
For the Unity approach, the NGUI kit is used to develop UI controls. This means that one cannot achieve 100% look and feel of the native environment. Furthermore, NGUI may present more obstacles during the implementation since the library is not so widely used as the native GUI libraries. The UI score for Unity is thus only given a ranking vector of 0.11.

5.3 Security

This subsection compares sharing a core code base, Xamarin and Unity on the Security criterion.

The results for the criteria security can be seen in Figure 5.5. It is here visible that the approach of sharing a core code base gets far better results with a ranking vector of 0.60 than Unity and Xamarin that both were given a value of 0.20 for the security attribute.

Security	Xamarin	Unity	Sharing a core code base	
Xamarin	1	1	1 / 3	0.20
Unity	1	1	1 / 3	0.20
Sharing a core code base	3	3	1	0.60

Figure 5.5 – Ranking Vectors for the Security Attribute

Sharing a core code base
Despite the fact that each language such as C, C++ and Objective-C has its own vulnerabilities that can be exploited to attack the application developed using a specific language, the proper use of each language can to a large extent mitigate the risk of security vulnerabilities.
The alternative received the high security ranking vector due to the fact that it can provide a low level access to the System API. As an example, it can be used to check a process state flag that indicates if the process being traced is set by the kernel when an application is being debugged. When combined with language-based security features of either Java, Objective-C and C/C++, the best security solution can be provided.

Xamarin
Xamarin uses the Mono run-time that implements the ECMA Common Language Infrastructure (CLI). This approach received a lower ranking vector than sharing a core code base. The reason for this is that Mono cannot provide similar protection techniques that otherwise can be implemented by using the low level API.

Unity
Unity is also using the Mono run-time environment, similar to Xamarin. It received the same ranking vector of 0.20 as Xamarin since it does not use the native API.

5.4 Reusability

This subsection compares sharing a core code base, Xamarin and Unity on the Reusability criterion.

Networking
One of the second-level attributes for reusability is networking. All three technologies received the same ranking vector of 0.33 for this criterion as stated in Figure 5.6. All technologies received the same values because the network logic can be reused both on iOS and Android regardless of approach.

Networking - Reusability	Xamarin	Unity	Sharing a core code base	
Xamarin	1	1	1	0.33
Unity	1	1	1	0.33
Sharing a core code base	1	1	1	0.33

Figure 5.6 – Ranking Vectors for Networking in terms of Reusability

Data Access
Another sub-attribute for reusability is data access. The results from this pair-wise comparison are shown in Figure 5.7 where we see that Xamarin and Unity obtained the same ranking vector of 0.43. The approach of sharing a core code base got a lower score of 0.14 due to the fact that the data access logic will differ on iOS and Android when using POCO because of different sand-boxing mechanisms. This will impact the reusability since the code cannot fully be shared; around 10% of the data access code will always be platform specific.

Data Access - Reusability	Xamarin	Unity	Sharing a core code base	
Xamarin	1	1	3	0.43
Unity	1	1	3	0.43
Sharing a core code base	1 / 3	1 / 3	1	0.14

Figure 5.6 – Results for Data Access in terms of Reusability

User Interface

Lastly, the sub-attribute user interface was evaluated for the criterion reusability. As described in Figure 5.8, Unity achieved the highest ranking vector. This depends on the fact that the UI logic when using Unity together with NGUI can be shared among the mobile platforms. Neither Xamarin, nor the C++ core library can offer shared UI logic.

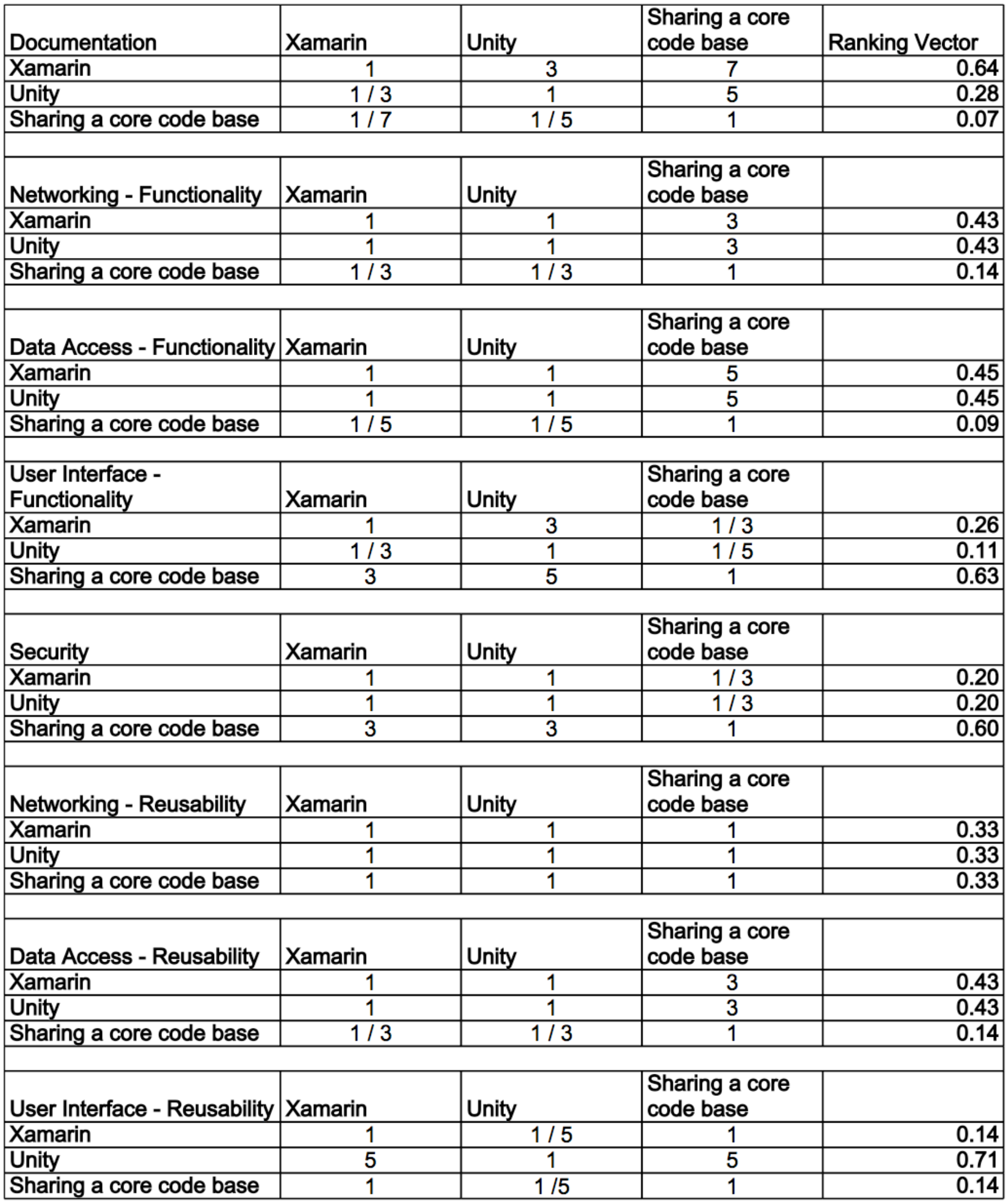
User Interface - Reusability	Xamarin	Unity	Sharing a core code base	
Xamarin	1	1 / 5	1	**0.14**
Unity	5	1	5	**0.71**
Sharing a core code base	1	1 /5	1	**0.14**

Figure 5.7 – Ranking Vectors of UI in terms of Reusability

5.5 Results

This subsection aggregates all the intermediate data, the relative weights of the criteria as well as the relative weighs of the alternatives, and produces the final evolution metrics.

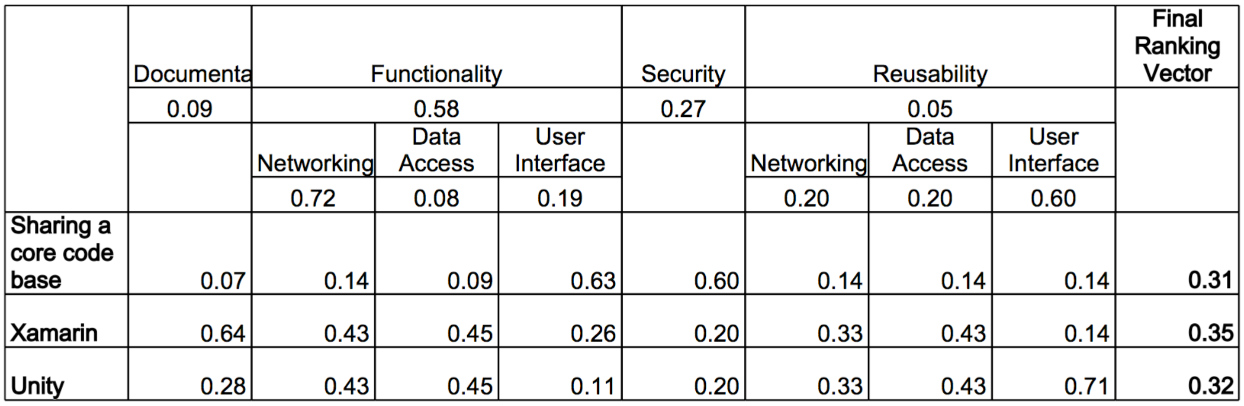
Figure 5.9 shows the results of a comparison analysis of every alternative solution on each of the low-level attributes as well as the ranking vector of all alternatives.



Documentation	Xamarin	Unity	Sharing a core code base	Ranking Vector
Xamarin	1	3	7	0.64
Unity	1 / 3	1	5	0.28
Sharing a core code base	1 / 7	1 / 5	1	0.07
Networking - Functionality	Xamarin	Unity	Sharing a core code base	
Xamarin	1	1	3	0.43
Unity	1	1	3	0.43
Sharing a core code base	1 / 3	1 / 3	1	0.14
Data Access - Functionality	Xamarin	Unity	Sharing a core code base	
Xamarin	1	1	5	0.45
Unity	1	1	5	0.45
Sharing a core code base	1 / 5	1 / 5	1	0.09
User Interface - Functionality	Xamarin	Unity	Sharing a core code base	
Xamarin	1	3	1 / 3	0.26
Unity	1 / 3	1	1 / 5	0.11
Sharing a core code base	3	5	1	0.63
Security	Xamarin	Unity	Sharing a core code base	
Xamarin	1	1	1 / 3	0.20
Unity	1	1	1 / 3	0.20
Sharing a core code base	3	3	1	0.60
Networking - Reusability	Xamarin	Unity	Sharing a core code base	
Xamarin	1	1	1	0.33
Unity	1	1	1	0.33
Sharing a core code base	1	1	1	0.33
Data Access - Reusability	Xamarin	Unity	Sharing a core code base	
Xamarin	1	1	3	0.43
Unity	1	1	3	0.43
Sharing a core code base	1 / 3	1 / 3	1	0.14
User Interface - Reusability	Xamarin	Unity	Sharing a core code base	
Xamarin	1	1 / 5	1	0.14
Unity	5	1	5	0.71
Sharing a core code base	1	1 /5	1	0.14

Figure 5.9 - The Ranking Vector of All Alternatives

After the ranking vector for criteria and alternative is computed, the next stage is to aggregate all the intermediate data and to produce the final calculations. Figure 5.10 shows the intermediate results as well as the Final Ranking Vector.



	Documenta	Functionality			Security	Reusability			**Final Ranking Vector**
	0.09	0.58			0.27	0.05			
		Networking	Data Access	User Interface		Networking	Data Access	User Interface	
		0.72	0.08	0.19		0.20	0.20	0.60	
Sharing a core code base	0.07	0.14	0.09	0.63	0.60	0.14	0.14	0.14	**0.31**
Xamarin	0.64	0.43	0.45	0.26	0.20	0.33	0.43	0.14	**0.35**
Unity	0.28	0.43	0.45	0.11	0.20	0.33	0.43	0.71	**0.32**

Figure 5.10 - Intermediate Results and the Final Ranking Vector

To calculate the Final Ranking Vector for each alternative, we derived the weighted sum of the alternative's attribute ranking.

As described in Figure 5.10, the approach of sharing a core code base got the final value of 0.31, while Xamarin and Unity got 0.35 and 0.32 respectively.

Now we can interpret the final ranking values. We can compare each build on a percentage basis, because the sum of all values in the priority vector equals to 1. We can say that Xamarin is 3 percent (0.35 – 0.32 = 0.03) better than Unity or 4 percent (0.35 – 0.31 = 0.04) better than the approach of sharing a core code base. However, this interpretation is not fully accurate because it only applies to the current case when three alternatives are compared. If we consider a fourth alternative for the analysis, the final ranking vector would change. Xamarin would in this case be less than 3 percent better than Unity and less than 4 percent better than sharing a core code base. When comparing three alternatives, if a difference is bigger than 0.20, it can be considered as a significant difference. A difference is not significant if it is less than 0.05.

Since all three approaches got the final ranking values with differences less than 0.05, we can say that all of them are good candidates for developing the cross-platform application for the target application domain.

6. Conclusion

Several alternatives were evaluated during this master thesis that were put in relation to the target application domain, which is more or less a skeleton for a typical mobile client product. The proof-of-concept applications were developed after the functional requirements Network, Data Access and User Interface; these attributes later facilitated the comparison between the three cross-platform approaches Xamarin, Unity and sharing a core code base.

Similar final results were obtained for the three cross-platform tools, which means that all of them are potentially promising candidates. However, in a real life situation, the actual applications may have more strict demands on specific parts of the functionality. For example, an application such as Skype has high demands on the functional requirements Networking and Database Access as well as Security. The best candidate in this case will be to share a core code base where the C++ language is used to develop the cross-platform core. In a second scenario, Unity would be the preferred cross-platform tool to use when developing an application with custom UI and 3D effects that would be used together with NGUI. In yet another scenario, Xamarin may be the best choice if the application lacks strict requirements in terms of GUI or Networking/Data Access.

The results of the final comparison can also differ if other attributes are selected. This may depend on the resources that the company has. For example, if the company has access to a .NET developer that is available to work on a given project, Xamarin or Unity may be the best choice at the time.

Software Engineering is a dynamically changing industry. Technologies that fail to show potential today can have a lot more to offer in half a year. The purpose of this master thesis has been to give an indication of which technique IT companies should use to find the best technology for mobile cross-platform development. The results should be considered as a reflection over the current situation on the market rather than solid and unchangeable over time.

References

[1] NC State University. *Multi-Criteria Decision Analysis*. 2011. Web [2013-06-17]. URL: http://www.ncsu.edu/nrli/decision-making/MCDA.php

[2] Triantaphyllou, E & Mann, S. *Using the Analytic Hierarchy Process for Decision Making in Engineering Applications: Some Challenges*. Inter'l Journal of Industrial Engineering: Applications and Practice, Vol. 2, No. 1, pp. 35-44, 1995.

[3] McCaffrey, J. *The Analytic Hierarchy Process*. 2005. Web [2013-06-18]. URL: http://msdn.microsoft.com/en-us/magazine/cc163785.aspx

[4] Staessen, M. *A comparative study of cross-platform tools for mobile application development*. 2013. Master Thesis for Katholieke Universiteit Leuven.

[5] Xamarin Developer Center. *Introduction to Mobile Development*. 2013. Web [2013-07-01]. URL: http://docs.xamarin.com/guides/cross-platform/getting_started/introduction_to_mobile_development

[6] Jordan, P. *Strategies for Using C++ in Objective-C Projects (and vice versa)*. 2012. Web [2013-07-01]. URL: http://philjordan.eu/article/strategies-for-using-c++-in-objective-c-projects

[7] Android Developer. *Android NDK*. 2013. Web [2013-07-01]. URL: http://developer.android.com/tools/sdk/ndk/index.html

[8] Boost. *Boost C++ Libraries*. 2007. Web [2013-07-02]. URL: http://www.boost.org/

[9] Cogswell, J. *Comparing the C++ Standard and Boost*. 2013. Web [2013-07-02]. URL: http://slashdot.org/topic/bi/comparing-the-c-standard-and-boost/

[10] Poco. *Poco C++ Libraries Features*. 2013. Web [2013-07-02]. URL: http://pocoproject.org/features.html

[11] Poco Project. *Introduction: a Guided Tour of the POCO C++ Libraries*. 2013. Web [2013-07-02]. URL: http://pocoproject.org/docs/

[12] Sourceforge. *Poco C++ Libraries, Cross-platform C++ libraries with a network/internet focus*. 2013. Web [2013-07-02]. URL: http://sourceforge.net/projects/poco/

[13] Unity 3D. *Unity for Mobile Development*. 2013. Web [2013-07-02]. URL: http://unity3d.com/unity/multiplatform/mobile

[14] Tasharen Entertainment. *NGUI: Next-Gen UI kit*. 2011. Web [2013-07-02]. URL: http://www.tasharen.com/?page_id=140

[15] Developer Force. *Editing Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options*. 2013. Web [2013-07-02]. URL: http://wiki.developerforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options

[16] Xamarin Developer Center. *Introduction to Web Services, Consuming and Configuring Web Services in Xamarin's Mobile Platform*. 2013. Web [2013-07-03]. URL: http://docs.xamarin.com/guides/cross-platform/application_fundamentals/introduction_to_web_services

[17] Applied Informatics. *Introduction: A Guided Tour of the POCO C++ Libraries*. 2013. Web [2013-07-03]. URL: http://www.appinf.com/docs/poco/00100-GuidedTour.html

[18] Unity 3D. *Mono Compatibility*. 2013. Web [2013-07-03]. URL: http://docs.unity3d.com/Documentation/ScriptReference/MonoCompatibility.html

[19] Xamarin Developer Center. *Part 5 – Practical Code Sharing Strategies*. 2013. Web [2013-07-03]. URL: http://docs.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/part_5_-_practical_code_sharing_strategies

[20] Poco Data Library. *POCO Data User Guide*. 2013. Web [2013-07-03]. URL: http://pocoproject.org/docs/00200-DataUserManual.html

[21] Xamarin Developer Center. *Building Cross Platform Applications: Best Practices for Developing Mobile Applications with Xamarin*. 2013. Web [2013-07-04]. URL: http://docs.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications

[22] Mono. *The Mono Runtime*. 2013. Web [2013-08-05]. URL: http://www.mono-project.com/Mono:Runtime