RESEARCH ON CROSS-PLATFORM MOBILE APPLICATION DEVELOPMENT

(Android & iOS)

Final Report V 2.0

Digital Innovation

Saurav Aran
Intern at Royal Philips
June 10, 2013

GRADUATION REPORT

FONTYS UNIVERSITY OF APPLIED SCIENCES

HBO-ICT: English Stream

Data student:	
Family name , initials:	Aran, S
Student number:	2154056
project period: (from – till)	1/February/2013 – 31/July/2013
Data company:	
Name company/institution:	Royal Philips
Department:	Research
Address:	High Tech Campus 34, Eindhoven, The Netherlands
Company tutor:	
Family name, initials:	Bonné, P
Position:	Senior Designer
University tutor:	
Family name , initials:	Dorenbos, M
Final report:	
Title:	RESEARCH ON CROSS-PLATFORM MOBILE APPLICATION DEVELOPMENT
Date:	10/June/2013

Approved and signed by the company tutor: Patrick Bonné

Date: 10/June/2013

Signature:

Preface

This thesis report is written as a result of my graduation internship conducted at Philips Research, Eindhoven. This graduation internship is the final part of the Bachelor study in Information and Communication Technology (Software) at the Fontys University, Eindhoven. Besides this report, a project plan, demo iOS and Android applications, a multi-platform application (that runs on Android and iOS) and a research learning report/recommendation has been created.

Firstly, I want to thank all my teachers and classmates at the Fontys University for the wonderful environment they created during the 4 years long journey, making it much memorable and special. I express my gratitude and thanks to my university supervisor Mr. Marco Dorenbos for his continuous advices and his dual visits to the company during the graduation project.

Secondly, I thank all the Digital Innovation team members inside Philips who were very friendly and helpful during the graduation project. I thank from the core of my heart to my company supervisor Mr. Patrick Bonné for his support, guidance, encouragement and goal achieving mentorship. Also, I can't step back in thanking especially to Mr. Maurice Hebben for his continuous suggestions, sharing of information and friendliness throughout the project. My special thanks to Mr. Christiaan Tilman for his time and instruction regarding the iOS information during the project.

Finally, I thank my family and relatives for providing me such a wonderful chance to have an International experience in life. Their love, support and inspirations have all been playing a great role for the success in every step of my life.

Regards,

Saurav Aran

Table of Contents

Summary

With the advancement in technology, people tend further innovate their products to make it even more efficient, astonishing and easier to operate. Companies innovate their existing products to make them more useful by providing additional features to their customers. Mobile devices like Smartphones and tablets are portable and found in almost 1 billion populations in the world. Different applications are built in these mobile devices to provide the users entertainment, education, photography etc. Companies target these smartphones and tablets to provide applications to its customers for getting control and information to/from the products that are manufactured by the companies.

This project was conducted in DI (Digital Innovation) team inside Philips in Eindhoven that is engaged in the innovation of Philips products. The team tries to achieve applications on mobile devices for interacting with the Philips products but the challenging part is the fragmentation in the operating system (OS) in these mobile devices. The presence of different OSs like Android, iOS, Windows etc. makes developing application that target maximum numbers of customer costly and time consuming. Hence, there was a need of cross platform (multi-platform) approach in the app development that could easily eradicate such bottlenecks. With this approach just building an application once and using it across different platforms makes the application development time and cost efficient, and also provides service/access to a whole group of customers using different platform smartphones and tablets.

There are different existing cross platform mobile frameworks such as Xamarin, PhoneGap, RhoMobile etc. This project focuses on the use of Xamarin for the cross platform mobile application development. For the research activity, one of the Philips products called WakeUpLight (WUL) was used. Research questions like “How is it like to develop a cross platform application for the WUL using the Xamarin Studio Integrated Development Environment (IDE) that is able to control WUL’s functionalities using pre-existing ICP Client library?” and “A recommendation on the best model/approach/patterns for the maximum sharing of codes between the platforms in Xamarin” were received for the research. The goal of these research questions were to find out the efficiency in cross platform mobile application development for Philips using Xamarin.

The project planning, time management, Scrum and User Interface Design (UID) were done. Different tools and techniques like Xamarin Studio, Eclipse, X-code, C#, Java, Objective- C etc. were employed. Different online resources like Xamarin developer’s guide, YouTube, Stack Overflow, Philips internal documents etc. were used to conduct the research. The research resulted in recommendations and a demo cross platform mobile app built using Xamarin. The recommendation includes the use of MVVMCross (MVX) framework with Xamarin for a cross platform app development. Xamarin along with MVVMCross framework makes very good cross platform mobile application development strategies that helps in creating and managing large projects, provides robustness, app with native look and feel with excessive code sharing. In future, the implementation of ICP Client, which is the secure communication channel of Philips, can be demonstrated properly on Xamarin using MVVMCross framework.

It was a very interesting, educational, personal developing and wonderful experience in carrying out the graduation project at Philips with lots of pleasant people around.

Version History

Version #	Date	Comments
V 1.0	19/March/2013	First draft version containing Chapter 2 and 3. Received comments from supervisors
V 1.1	12/April/2013	Second draft version containing chapter 4. Received comments from supervisors
V 1.2	25/April/2013	Updated with chapter 1. Received comments from supervisors
V 1.3	14/May/2013	Updates with Chapter 8 and Preface.
V 1.3	24/May/2013	Updated with the remaining parts in the report. Received comments and suggestions from the supervisors
V 1.4	3/June/2013	Updated with respect to suggestions received and completed the report. Formatted the report according to the guidelines for writing report provided in the Fontys intranet. Semifinal version of the Report is ready. Final checkup is needed to be performed. Received comments and suggestions from the supervisors.
V 2.0	10/June/2013	Updates with respect to the suggestions/comments received from the supervisors. This is the final version of the report.

Glossary

AM:	Agile methodologies; one of the methods in Software development Life Cycle (SDLC).
Android:	Operating system by Google that runs on mobile devices.
App:	Application; software that runs on Operating System to carry out different tasks.
CPP:	Connected Product Platforms; Philips platform for its internet connected products.
DI group:	Digital Innovation group; a team formed from different Philips department for digital innovation.
ICP Client:	Internet Connected Products Client; Philips owned client for secure connection for CPP.
iOS:	iPhone Operating System; mobile Operating System owned by Apple Corp.
JNI:	Java Native Interface; a programming framework that enables Java code running in a JVM to call, and to be called by, native applications and libraries written in C, C++ .
JVM:	Java Virtual Machine; a virtual machine that can execute Java bytecode.
MVC:	Model View Controller; a pattern used during application development.
MVP:	Model View Presenter; a pattern used during application development.
MVVM:	Model View View-Model; a pattern used during application development.
MVX:	MVVMCross; a framework that implements the MVVM pattern for the application development.
NDK:	Native Development Kit; NDK is a toolset that allows implementing parts of app using native code languages such as C and C++.
NVM:	Non Volatile Memory; a component in ICP Client where different keys for the communication are stored.
PL:	Programming Language; Language particularly used in computers as instructions.
SDLC:	Software Development Lifecycle; process in the development of software.
SSIGR: Research.	Software Systems Integration Group Research; one of the groups inside Philips
UI:	User Interface; the place where user interaction takes with the machine/computers.
WUL:	Wake Up Light; Philips lamp product.

1 Introduction

At the present context, the use of mobile devices like Smartphones, tablets etc. have skyrocketed. People seem to be enjoying having a portable device and these devices have pretty much become a part of their life. Whether it is for communication, entertainment, health or educational etc. mobile devices facilitates with all these featured applications.

Philips, a multinational electronic company is stepping forward innovating its products. Innovation here refers in making its products accessible, viewing the product's information and being able to control them from mobile devices remotely. For example: Light bulbs when connected can be controlled (switch On/Off), enables to see the past activities (switched On/Off with time stamps) etc. via apps (applications) on the mobile devices.

Well, there are products meant to be innovated and there are mobile devices. So, what's the problem in here for this project to be initiated? Yes, there are mobile devices but, there exists different platforms (iOS, Android etc.) over these mobile devices. Hence, due to this fragmentation of platforms, making an application for a product to provide facilities to its maximum customer would require developing application for these several platforms individually. For example: It would require expertise in 4 different platforms, creating an app 4 times for an app of same functionality for iOS, Android, Windows and Blackberry. This will result to cost high amount of time and money. So, wouldn't it be nice if there exist possibility of creating application just once and use it across all the available platforms (cross platform) by making reuse of code to maximum across different platforms? Therefore, this project focuses on the research and development for making a cross platform application that suits Philips architecture. The research is mainly based on the use of Xamarin for the development of cross platform mobile applications for iOS and Android.

Chapter 2 provides information about the company, chapter 3 gives all the details about the project assignment and chapter 4 discusses about the methods, tools and techniques used during the project. Chapter 5 describes the kickoff activities. Thereafter, Chapter 6 and 7 provides information on Research questions conducted during the project and chapter 8 describes the project progress tracking. Finally, chapter 9 consists of conclusion(s) and recommendation(s).

2 The Company

This chapter describes about the company where the project was conducted.

This project is conducted at Royal Philips (Philips). Philips is a Dutch multinational electronics company established in 1891. It was founded by Gerard Philips and Frederik Philips. The current headquarter is in Amsterdam, The Netherlands. It has around 122,000 employers in more than 60 countries and Frans Van Houten is the current Chief Executive Officer (CEO) of the company. It is the largest manufacturer of lighting in the world measured by applicable revenues, as of 2012.

The company is mainly divided into three divisions: Philips Lighting, Philips Healthcare and Philips Consumer Lifestyle. Beside these divisions, Philips has a Corporate Sector that includes Philips Research. [1]

This project is carried under the DI (Digital Innovation) team of Philips, located at the High Tech Campus (HTC) in Eindhoven, The Netherlands.

2.1 Philips Research- Eindhoven

Philips Research Eindhoven is a corporate research organization of Philips that works for all innovation areas that includes development of future products and businesses for the support and maintaining the competitiveness of all the three main sectors of Philips in the current market.

It is located at the HTC area (Figure 2.1.1) where more than 100 other companies are established for the exchange of ideas between the companies, forming a Research and Development ecosystem.

Figure 2.1.1: Green region showing the aerial view of HTC area [2]

2.2 Digital Innovation (DI) team:

DI team consists of people gathered together from Software Systems Integration (SSI) group (one of the departments inside Philips Research), Philips Consumer Lifestyle and some other guest company like TASS (Total Specific Solutions) etc. The team is formed for the innovation of the Philips

products in the market with a slogan "Putting brains into our products, so consumers take them to heart". The team is engaged in connecting all the Philips products together so that, the operational information by its users can be collected and used by the Philips for better maintainability and improvement of Philips products to provide a greater user experience. For assisting this team with the research, a student is appointed.

2.3 Organization chart

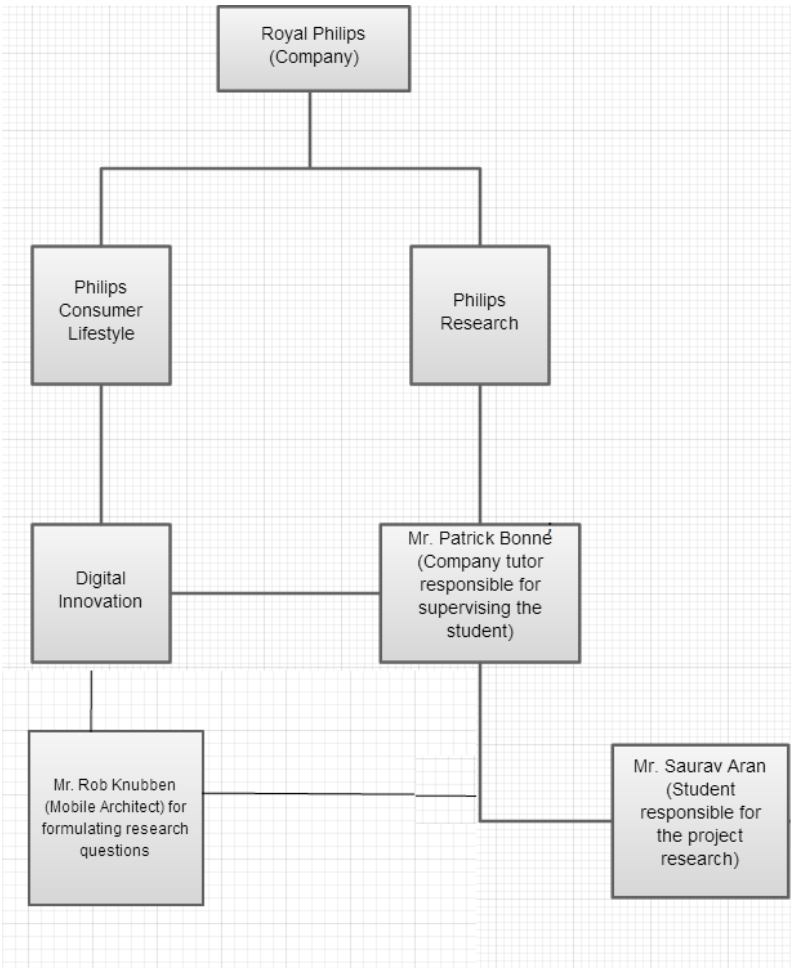


Figure 2.3.1: Organization chart

Figure 2.3.1shows the organization chart for the project. The chart shows only the divisions of Philips that have a connection with this project.

3 Assignment overview

This chapter gives information on the project assignment. The information consists of the analysis of the current situation, bottlenecks, purpose of the assignment, desired end situation, the precise command description and about the project plan.

The DI team is on the path of innovation for the company's existing products like Coffee machine, Home Cooker, Wake up Lights (WUL) etc. on the market. For the innovation of the products, they are trying to connect these Philips products. The platform for making these entire devices connected is called as Connected Product Platform (CPP). This CPP enables bidirectional data/information exchange ability to all the products from the Philips server.

Making the devices connected empowers the company to study the data received from these connected devices and help in analyzing the way its clients are using the products. This can help the company in taking proper steps for the further development of products, assisting its clients in case of any problems that arise in the products and preventing premature defects on the machine. For example: If a Coffee machine is connected, then the company is capable to see the statistics of all the data such as the operation history (time when the product was switched ON, OFF, refilled etc.) and hence can study the way/frequency of product usage on the basis of residential or geographical region. These statistics can also help in providing a better assistance to the customers in case the product copes with some defect. How about the coffee machine sending a notification to its owner when it's time for decalcifying[1] the machine? This helps in preventing the machine from early defects and overheating due to the accumulation of the calcium leftovers from water used on the machine providing the customers a greater satisfaction on the products. These products being connected allows them to be controlled and see its statistical data via different mobile devices (smartphones, tablets etc.) the customers own. This shows a very good example of Digital Innovation on a product. The customers get the power to control all its Philips devices under a single click of their finger on their mobile devices. The team also focuses on providing a simple, easy to learn mobile application with almost similar user experience (the way application interacts with the user such as notifications) and native look across different mobile platforms.

3.1 Current Situation

Some of the Philips products (for testing) are made connected. These devices are possible to control via mobile devices using the Philips secure client for connection called ICP Client. Multiple mobile applications with basic functionalities are built according to the number of target platforms. For example: WUL application for Android and iOS that is able to switch the WUL On or Off, increase or decrease the intensity of WUL etc.

There are different development platforms (Xamarin, Phone gap, service2media etc.) that are under research, distributed among different researchers inside Philips. This project focuses on Xamarin to check whether it fits the Philips architecture in providing a better option for cross platform application development.

3.2 Bottlenecks

Due to the existence of numerous mobile platforms (iOS, Android etc.) on the market, it is a tedious, time consuming and costly task for developing different applications with the same functionality feature on all these target platforms repeatedly.

[1] In coffee machines, a chemical process of removing the calcium leftovers from the water on the machine is called as decalcifying.

Also, the company's requirement for providing its customers, an application across different platforms for controlling the products with almost similar user experience and native look and feel is one of the most challenging tasks to fulfill.

3.3 Purpose of the assignment

This assignment's purpose is to help eliminate or reduce the bottlenecks that exist in the DI of the Philips. The help includes assisting the DI team by conducting investigation on the research questions that are defined by the DI team and providing recommendation on these research questions. These research questions focus particularly on the investigation on cross platform application development with native and similar user experience. The target platforms to research in this project are iOS and Android. Xamarin is the development environment used in this project. Research on Xamarin will be carried to find its use in the cross platform application development.

The research questions received are:

1st research question: How is it like to develop a cross platform application for the WUL using the "Xamarin Studio" Integrated Development Environment (IDE) that is able to control WUL's functionalities using pre-existing ICP Client library? [2]

2nd research question: A recommendation on the best model/approach/patterns for the maximum sharing of codes between the platforms in Xamarin. [3]

3.4 Desired end situation

Deliver of investigated information and results that consist of recommendations on the research questions defined by the DI team. It should contain brief useful information that meets the purpose of the assignment in developing cross platform application with native look and feel and similar user experience with a supporting proof of concept.

3.5 Command description

The usage of programming language (PL) and Integrated Development Environment (IDE) can vary with respect to the research questions received, which is suitable to accomplish the investigation. There are no special constraints set on methods, tools and techniques like Data Flow Diagrams (DFDs), Unified Modeling Language (UML), Prince 2 etc. to accomplish the project task.

Usage of ICP Client should be taken into account while carrying out the research and should meet the desired end situation. There is no need of comparing Xamarin with other tools for cross platform application development like PhoneGap, Titanium etc.

[2] Refer Chapter 5.1 (Kickoff) for ICP Client and Chapter 6.2 (1st Research question) for Xamarin
[3] Refer Chapter 7 (2nd Research question) for more information

3.6 Project plan

The first step in a project is preparing a project plan. Project plan is made in order to plot the activities to be performed and track those activities during the project to complete the project on time. (See Appendix B for more details on Project plan)

Different activities were planned to be performed during the project. The activities consisted of writing weekly notes to view the performed operations during the week, communication plan with the university tutor, creating and updating the Project plan, learning and developing demo applications for Android and iOS, executing research questions to investigate and provide recommendations, preparing the final report and feedbacks on it from the supervisors and finally preparing for the final presentation as seen in Figure 3.6.1. The project plan is updated after receiving each new research question, since Scrum was the adapted method for Software Development lifecycle (SDLC, detailed in Chapter 4.1.1). The time duration to execute the acquired research question is fixed by a proposal, which is calculated using different methods (see Appendix C to view the methods and calculation).

The total duration (contract) for the whole project is 25 weeks that is planned to consist of 3-4 research questions. But, the final report consists of 2 research questions because the final delivery time of the report is at the end of week 18 of the project i.e. graduation takes place before the end of the whole project. All the activities during the project were started and ended successfully as planned in the Project plan.

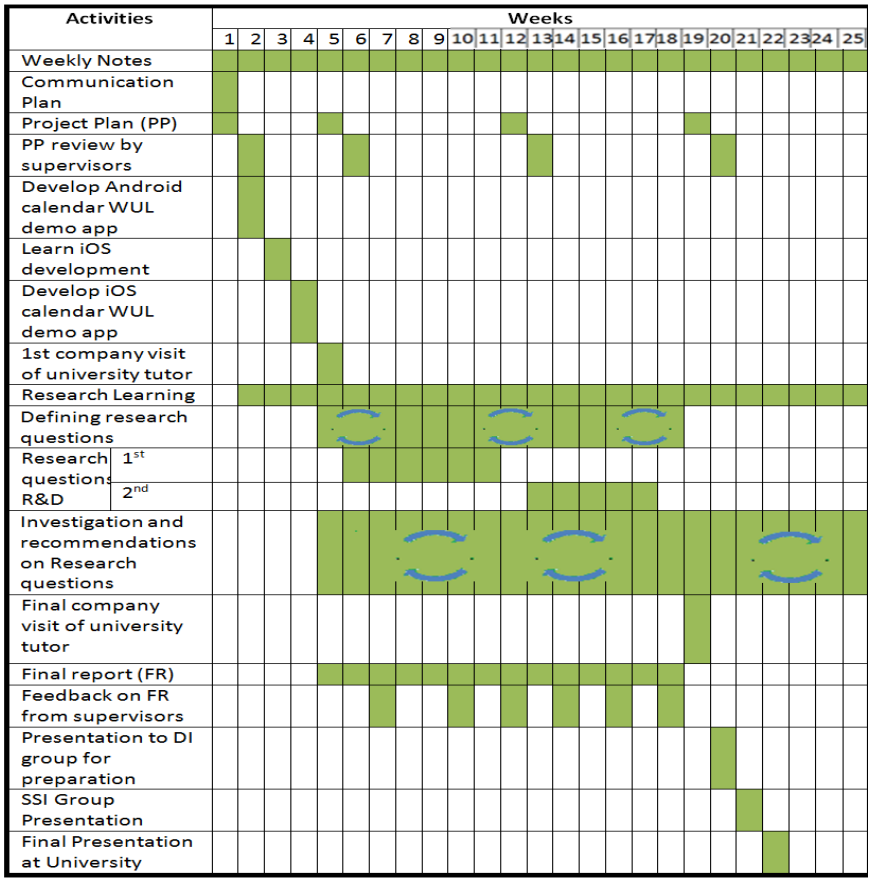


Figure 3.6.1:Activities to be performed during the project

4 Methods, Tools and Techniques

Different methods, tools and techniques were applied and used during the project. This chapter summarizes each of these in short.

4.1 Methods

4.1.1 Scrum in Software Development Life Cycle (SDLC)

The project being carried out in a research organization, the developmental activities carried during the project are in fact research oriented.

There are different existing models/methodologies for the SDLC like waterfall model, spiral model, agile method etc. In the traditional waterfall model, the phases (Analysis, Design, implementation, testing and maintenance) are planned and each of these phases occurs only after the previous phase to it completes. In this project, the Agile Methodologies (AM) of SDLC is applied. Unlike, waterfall model, AM uses feedbacks instead of planning. The feedback is driven by regular tests and releases of the evolving software/ research outputs. [3]

Among the different variations of AM, Scrum is the chosen method for the software development management in this project. Figure 4.1.1 shows the scrum SDLC framework. It is an incremental, iterative, flexible, holistic product development strategy where a development team works as a unit to reach a common goal". [4]

Scrum consists of Product Backlog, Sprint Backlog, Sprint and the Sprint result. The description of the product written in the form of a story is called as the Product Backlog. The product Backlog consists of rough estimates of both business and developmental values. This story in Product Backlogs when made into requirements to be implemented in the software is called as Sprint Backlog. For receiving the product that fulfills requirements in the Sprint Backlog, a time range of 1-30 days is assigned. This time boxed duration is called as the Sprint. After the completion of the sprint, it results in software that fulfills the requirement.

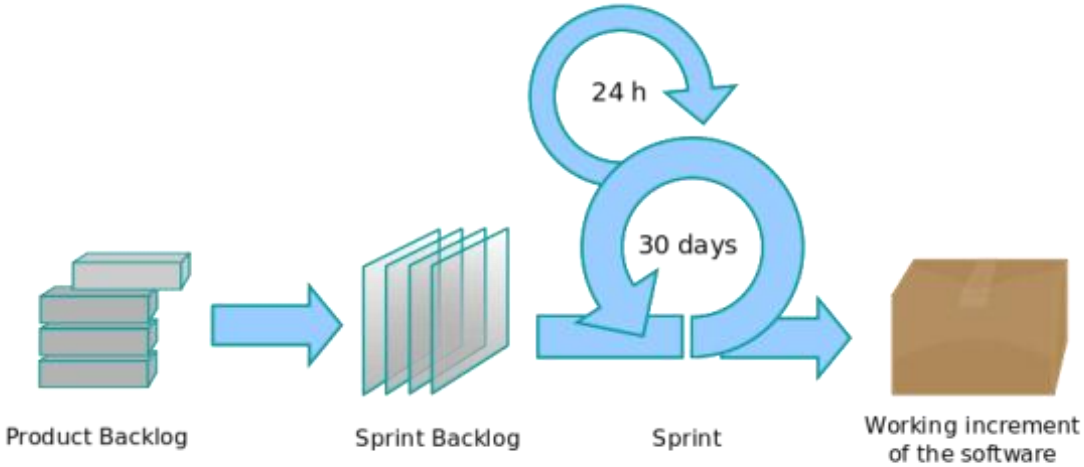


Figure 4.1.1:Scrum method in SDLC

After discussion and feedback on this version of result, next requirement will be carried out by iterating the sprint. The key benefit in this method is that, the customer can make a change in their minds (after a sprint) about their needs and wants. In this project, the time duration for each research question can be considered as the sprint in the scrum methodology.

4.1.2 User Interface Design (UID)

UID is the method in the software development that focuses particularly on the human machine interaction. It tries to increase the efficiency and simplicity in user experience with the machine. There are different phases in UID among which the phase of prototyping was particularly performed. Other phases like user analysis, usability testing etc. weren't executed. Figure 4.1.2 shows a prototype for sthe cross platform application that was made before starting the actual application development.

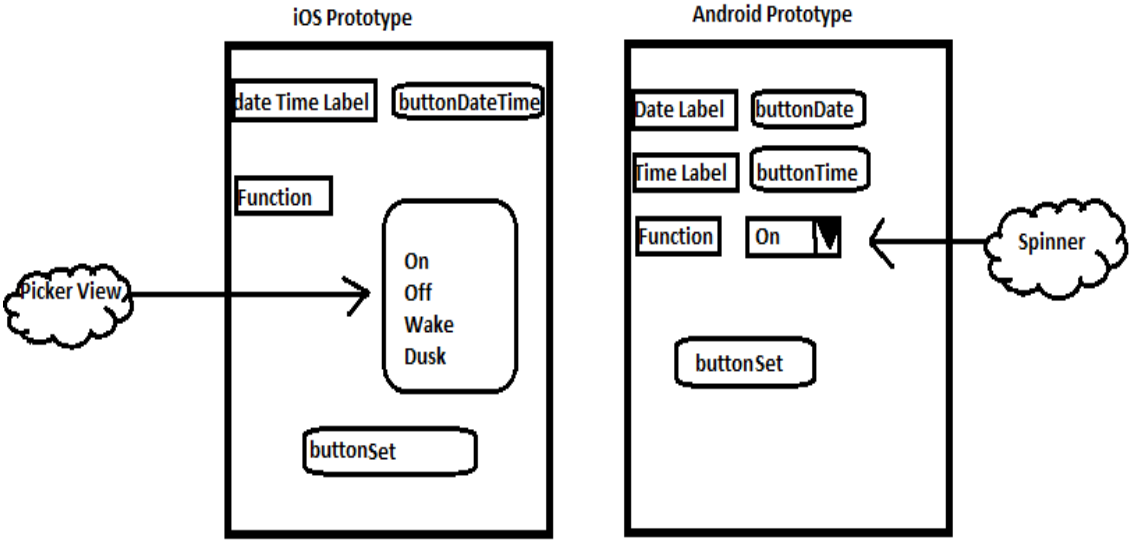


Figure 4.1.2: GUI-Prototype for the cross platform application

4.2 Tools

4.2.1 Microsoft Visio:

This tool was used for creating the UML diagrams (class diagram).

4.2.2 SmartBear SoapUI

SmartBear SoapUI allows easily and rapidly creating and executing automated functional, regression, compliance, and load tests. This tool was used to find out the XML format for sending the commands to the Philips CPP products via local Web services. [5]

4.2.3 Android Developer Tools (ADT) Bundle

ADT Bundle is a set of other tools that helps in the development of Android application. The ADT Bundle includes

a) Eclipse, which is an IDE helpful in the Android application development.
b) Android Software Development Kit (SDK) tools, which consists of other tools required for compiling, debugging, virtual Android device (emulator) etc.
c) Android Platform-tools.
d) Android system image for the emulator.

4.2.4 X-Code

X-Code is an IDE that is used in the application development for mac, iPhone and iPad.

4.2.5 Xamarin IDE Visual Studio

Xamarin is an IDE that uses C# as a primary language in the development of a cross platform mobile application. It consists of commercial products, namely Xamarin.Android (formerly: Mono for Android) and Xamarin.iOS (formerly: Monotouch). Xamarin.Android integrates with the Android to create Android applications. Xamarin.iOS integrates with the Cocoa and Cocoatouch framework to create iOS applications. The Xamarin compiler bundles the .NET runtime and outputs a native ARM executable, packaged as an iOS or Android applications. Using this IDE, both the Android and iOS applications can be built using C# PL and even lets share the codes across Android, iOS and Windows platforms. [6]
Visual Studio is and IDE for development of software applications from Microsoft. In this project it was used for Android and iOS mobile application development after the installation of Xamarin.iOS and Xamarin.Android plugins for Visual Studio.

4.2.6 Subversion (SVN)

While working in a project, to provide each member the latest version of the source codes/documents related to the project, SVN is used.

4.3 Techniques

4.3.1 Android & iPhone Operating System (iOS)

Android is the Operating System (OS) owned by Google. Android runs on mobile devices. iOS is the OS owned by Apple Inc. that runs on iPhone. The project research on multiplatform application development can be performed only by knowing these two OS.

4.3.2 Java & Objective-C Programming Language

Java and Objective-C are high level object-oriented programming language (PL). These are the primary PL that are needed to build an Android and iOS application respectively.

4.3.3 C & C#

C is a low-level[4] PL. Since the existing libraries of company are in native C/C++ language, to understand and use these libraries, the knowledge of C/C++ is needed.
C# is a high level object-oriented PL. This PL is used while working with Xamarin Integrated Development Environment (IDE).

4.3.4 Unified Modeling Language (UML)

UML is a standard way of visualizing the ideas, class diagrams, sequence diagrams etc. in software engineering.

4.3.5 Simple Object Access Protocol (SOAP)

SOAP is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks. It relies on Extensible Markup Language (XML) for its message format, and usually relies on other Application Layer protocols, most notably Hypertext Transfer Protocol (HTTP) or Simple Mail Transfer Protocol (SMTP), for message negotiation and transmission. [7] To send the commands to the Philips devices in a CPP, SOAP protocol can be used as one of the method.

[4] The low level language are the instructions that can be understood by the machine without the need of additional interpreter whereas high level language instructions need some type of interpreter to be understood by machines.

5 Kickoff

This chapter elaborates the processes followed before beginning to perform the research questions. This includes learning the architecture and creating two separate demo applications (native app[5]) for controlling WUL on iOS and Android target platforms.

5.1 Learning Philips architecture

To make the products connected, Philips has adapted an architecture called CPP (Connected Product Platforms) that consists of different elements that includes Data collection Portal, Data Receiver, control logic, Device Control Service (Figure 5.1.1) etc. These elements are responsible for the communication with the products remotely. These elements combine to form the CPP, also called Philips backend server. The learning for the architecture was done with respect to the information from the supervisor and the documents available on the DI team's SVN repository.

For this particular project, WUL was used as an item of Philips CPP product. WUL is a lamp that has basically the following functionalities:

- **On**: Switch the light on
- **Off**: Switch the light off
- **Increase**: Increase the intensity of light
- **Decrease**: Decrease the intensity of light
- **Wake**: Switching the light on by slowly increasing the intensity of light like a sunrise
- **Dusk**: Switching the light off by slowly decreasing the intensity of light like a sunset

Two WUL were placed in this CPP for research testing purpose, so that these functionalities can be executed remotely without physically touching the WUL. Currently, there are three ways for communication with this product. The ways are using webpage (yellow box), using Soap Interface (brown box) and using ICP Client (red box), as seen in Figure 5.1.1.

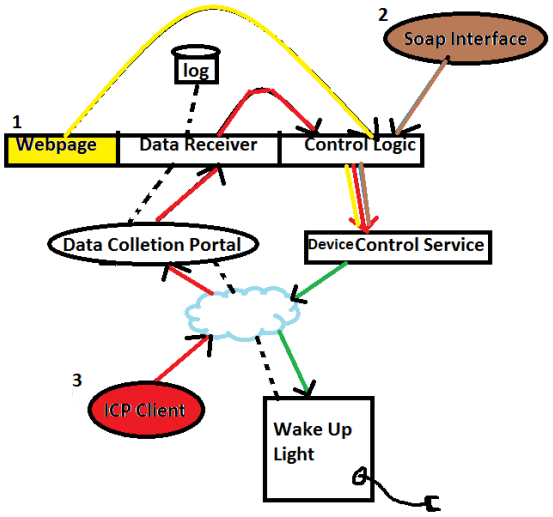


Figure 5.1.1: Possible ways to communicate to WUL

5.1.1 Using Webpage

A webpage is currently available which enables to send the commands to the WUL to switch it ON, OFF, Wake, Dusk, increase the intensity of light and decrease the intensity of light as seen in Figure 5.1.2.

5.1.2 Using SOAP message

The webserver also accepts SOAP requests over the internet to send the commands to the device specified in the request message (Figure 5.1.3).

[5] Application designed specifically to one target platform, built using the language that the target platform supports originally. Native apps have full access to their targeted platform and have high performance.

5.1.3 ICP Client

ICP Client is the Philips secure communication protocol which can be used to send and receive data to/from the CPP products remotely. The conceptual view of ICP Client can be seen in Figure 5.1.4 [8]. It realizes the functions and provides functional interfaces that allow a product or mobile device to access the DI back-end services.

Figure 5.1.2: Webpage for controlling WUL

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlso
   <soapenv:Header/>
   <soapenv:Body>
      <Command>
         <!--Optional:-->
         <lampId>?</lampId>
         <!--Optional:-->
         <command>?</command>
      </Command>
   </soapenv:Body>
</soapenv:Envelope>
```

Figure 5.1.3: Soap Envelope to send commands to WUL

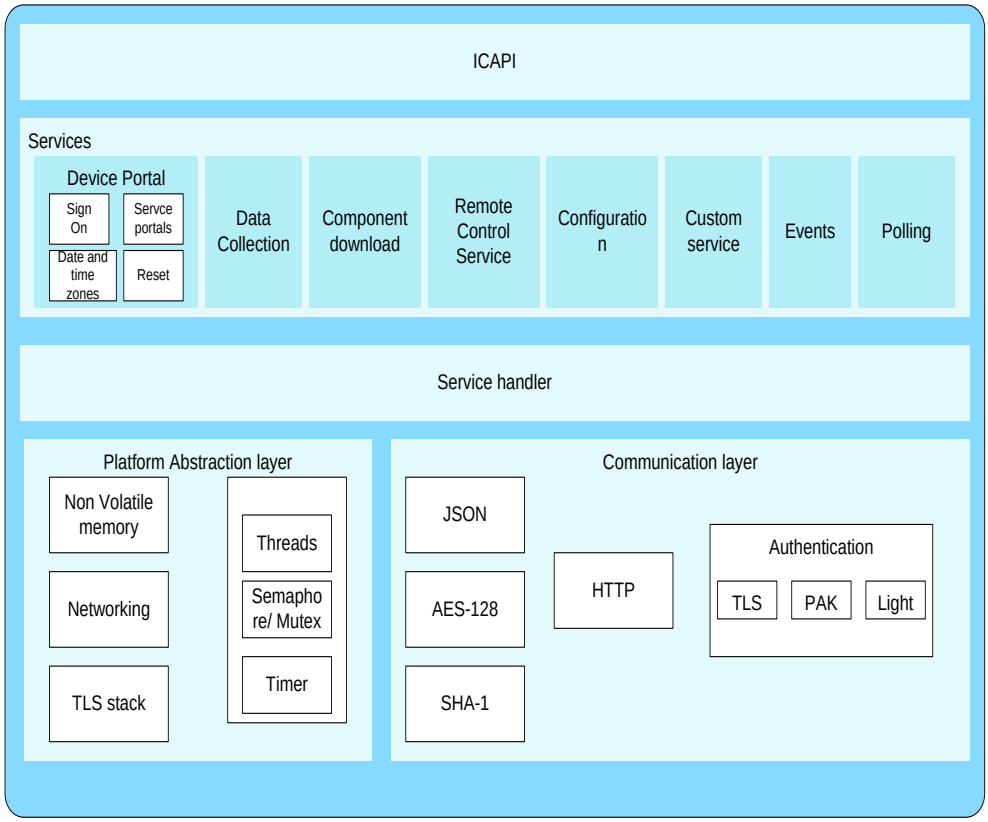


Figure 5.1.4: ICP Client Conceptual view [8]

5.2 Exploring Android and iOS

Before beginning to carry out research activities on cross platform application development for Android and iOS, the knowledge of developing app on these two specific platforms is essential. After gaining familiar with these two platforms, it will be a great relief in executing the research. Hence, firstly, it was assigned to develop a native iOS and a native Android alarm app that could send user selected commands to the WUL at the user specified date and time.

5.2.1 Creating Android and iOS alarm app to control WakeUpLight (WUL)

Being familiar with the Android development (Java and Eclipse) from the previous internship at Philips, it was quite an easy task to create an Android app. After gathering required information on Android development from the internet (developer.android.com, youtube.com, stackoverflow.com etc.), a simple paper prototype was firstly created on how the app will look like. Then the User Interface (UI) was designed and finally the Android application was built. The method used to send the command to the WUL is using SOAP. The application is able to let the user select the date and time and also, the user can select the command to be sent to the WUL (Figure 5.2.1). The application is able to trigger the user chosen function at user specified date and time. It also facilitates to view all the alarms that are set, with their details (date, time and function of alarm to be triggered). It displays a message in case the selected date and time is less than the current date and time.

With no previous experience with iOS at all, it started with the learning of Objective C language, using the X-Code IDE and building iOS application. After gathering the information required for iOS development from the internet (developer.apple.com, youtube.com, stackoverflow.com etc.), a paper prototype for the application was drawn, then the UI was designed and finally the iOS application was built. The method used to send the command to the WUL here is SOAP. It is able to trigger the selected function at user specified date and time by allowing the user to choose the date, time and function (Figure 5.2.2). It triggers the selected function instantly, in case, the date and time is less than the current date and time

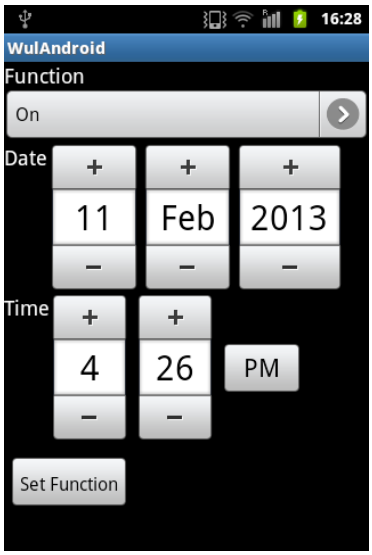


Figure 5.2.1: WUL alarm app for Android

Figure 5.2.2: WUL alarm app for iOS

5.2.2 Android-iOS comparison

After creating the two demo applications for controlling the WUL, some basic differences were to be noted on the basis of these two applications developed. The findings are as follows:

- The WUL alarm app on Android contains number of activities (with multiple xml layouts) and is an extended version of existing WUL app whereas the WUL alarm app on iOS is a completely new app built on a single View Controller (VC) which is similar to xml layout on Android. A story board can be used to see the overall VCs present in the application. The VC contains header files and a main file (where implementation is done). This VC can either be the default UIViewController class or a user created custom class.
- To start a new activity on Android apps, Intent can be used to load a new activity after which the Xml layout is loaded to the view. In iOS, a push or modal Segue can be used to open a new layout (basically a VC in iOS) within the application. Information between applications on Android can be passed using Intent whereas; IOS restricts this by limiting only through registered URL handlers. [9] No passing of information between apps was required while building the WUL alarm app on iOS and Android.
- In Android, starting a new activity from a different activity using Intent keeps all these activities in stack. Whereas, in iOS there are different types of VC and only if there is use of "Navigation controller" as the base VC, the screens/views are kept in stack using a Push Segue. Here, keeping in stack refers to the pause of the current screen and on top of that, a new screen is placed on, where both the activities are alive. If we return back from the stacked screen, the previous screen/VC which was paused will resume back closing the new screen that was opened. Using Modal Segue in iOS considers the new VC as a child VC of the current VC and the new activities cannot be kept in stack list. Hence, stacking is only possible using Navigation Controller VC as the base view where a Push Segue is used. Further VC can be a simple VC where a Push Segue is used to open a new VC and all these VC can be kept in stack. [10]
- For inserting a drop-down list in Android, Spinner was used to choose the list of functions. Whereas, in iOS, there is no availability of standard UI items for the dropdown list. However, as an alternative "Picker View" can be used that allows to select an item from list of available items. This UI item in WUL app is placed to let the user choose different functions like "ON", "OFF", "Wake" and "Dusk".
- For letting the user choose the date and time there is availability of "DatePicker" and "TimePicker" respectively on Android. In iOS, "DatePicker" can be used for the similar function. In the android DatePicker, the user was able to select the Year, but in the iOS Date Picker, there is no option for choosing the Year, if it shows both the Date and Time. However, it is possible to show the year, only if the date is shown. For WUL control, the Year isn't really needed because none of the user will really try to set their WUL light function for years ahead.
- The Android and iOS platforms both support working with multiple Threads (multithreading). During the development on Android, the started Thread ran even on the background but on iOS the Thread was suspended after entering the background mode. This can probably due to the reason of missing implementation for handling the background threads.

Hence, UI related similarities and differences on the two platforms (iOS and Android) were the most of the knowledge learnt in the development of this demo version of the WUL alarm app..

6 1st Research question (1st Sprint)

This chapter describes the steps performed with learning, development, problems and recommendations for the first research question received.

During the project, different question arise and recommendations on those questions should be provided on the basis of research conducted. Each research question is regarded as a sprint in the project. The first research question states: "How is it like to develop a cross platform application for the WUL using the Xamarin that is able to control WUL's functionalities using pre-existing ICP Client library? "Appendix D provides detailed information for the 1st Research question.

6.1 Time proposal

When a task is given to be performed, it consumes certain time. This time duration needs to be estimated before starting the task. For the first research question, instead of just assigning a time period blindly for performing the research question, different methods were applied in calculating the time span for performing the task. The methods might not be efficient but was still chosen to determine the time span which might be nearly same to a random time span proposal. The proposal of 6 weeks was made and was accepted. (See Appendix C for the methods and proposal calculation). After the proposal acceptance, learning and gathering information on Xamarin Studio was performed.

6.2 Information gathering and learning

6.2.1 Information sources:

The first research question related to the Xamarin needed to be investigated. There are number of sources for collecting information and learning on Xamarin, among which three sources are the most helpful ones. Firstly, the official developer's guide for Xamarin [11] provides almost all the necessary documentation for the application development on Xamarin. Secondly, YouTube provides numerous tutorials and seminars in the learning/development on Xamarin [12]. Finally, Stack Overflow network consists of number of questions and answers that are very helpful in learning and solving problem while developing using Xamarin. [13]

6.2.2 Learning on Xamarin:

Xamarin is a software developer company that provides tools for mobile application development. One of their products is Xamarin Studio. It is a commercial IDE from Xamarin that is used to build native/cross platform mobile applications across Android, iOS and Windows platforms with C#. Only the iOS and Android platforms will be discussed. This IDE is supported on both Windows and Mac OS. The company's main product is Xamarin.Android (formerly Mono for Android) and Xamarin.iOS (formerly MonoTouch), combined called Xamarin twins. They are required for developing Android and iOS applications, built on top of Mono (an open-source version of the .NET Framework based on the published .NET ECMA standard), as seen in Figure 6.2.1, including memory management, reflection, and the .NET base class libraries. Saying just Xamarin, Implicitly represent its two products Xamarin.Android and Xamarin.iOS.

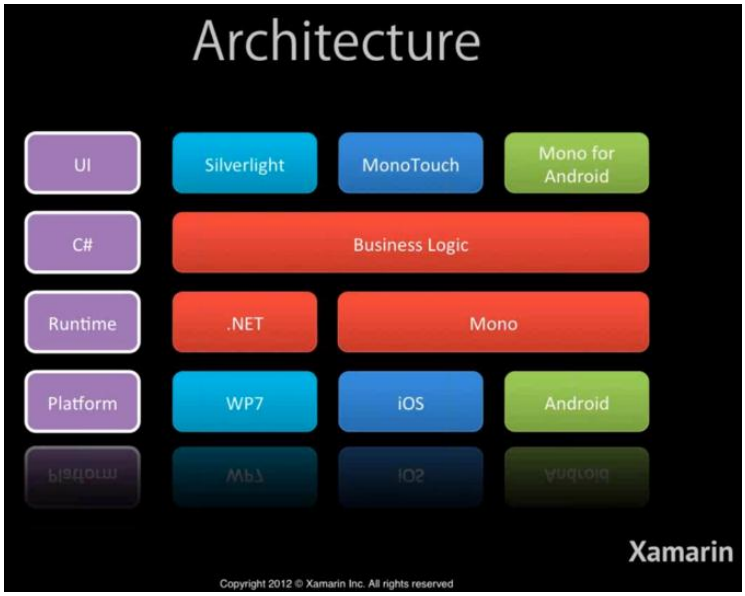


Figure 6.2.1: Xamarin Architecture

Xamarin's Ahead of Time (AOT) compiler compiles Xamarin.iOS apps to produce native Arm binary (.app) suitable for Apple's app store and Xamarin's compiler compiles Xamarin.Android apps down to Intermediate language and finally produces native binary output (.apk) for Android by taking advantage of Just in Time (JIT) compilation right on the Android device. As Xamarin compiles the app to a native binary (not cross-compiled/interpreted), it gives users brilliant app performance for even the most demanding scenarios like high frame rate gaming and complex data visualizations. Also, Xamarin exposes all the Application Programming Interfaces (APIs) available in iOS and Android to the developer as C# class libraries. The apps built are still native because, they don't abstract away the native platform UI APIs instead provide a bindings to the native APIs and provide access using C# library. [14] Hence, Xamarin being able to provide development of native featured (User Interface & performance) application and provide the ability to develop the cross platform application, it seems to be a good and very efficient IDE to choose regardless of its limitations (most of which are removed from the 4.1 release) on Xamarin.Android [15] and Xamarin.iOS. [16]

Note: The Xamarin IDE used in this research is Version 4.0.2 with Xamarin.Android (Version: 4.6.0) and Xamarin.iOS (Version: 6.2.1.201)

The application development using Xamarin studio for both iOS and Android is only possible on Mac OS, whereas, Xamarin studio on Windows doesn't support iOS development. However, to develop both Android and iOS applications, Visual Studio can also be used by installing the Xamarin.iOS and Xamarin.Android plugins for the Visual Studio. Using the Visual Studio for the iOS app deployment, a network Mac OS is required.

Simple Android and iOS native applications were built firstly in Xamarin. The apps contained basic functionality for clicking buttons and showing data information with respect to the click, just to be familiar with the Xamarin environment. Next step was to learn and create cross platform application on Xamarin.

6.3 Cross Platform app development using Xamarin

After being familiar with native app development in Xamarin, cross platform application was needed to be built to perform the first research question. Hence, the architecture for Xamarin cross platform development is to be learnt, prototypes and UML designs were to be created and finally the cross platform app for controlling the WUL is to be built.

6.3.1 Architecture

For building a cross platform app in Xamarin, firstly the strategy needed to be known. The architecture for the cross platform application can be seen in figure 6.3.1. The architecture clearly shows two portions separated. First portion is the View of the application that contains the UI (User Interface) layer and the Application layer. This part is specific to each platform target i.e. separate UIs need to be made for iOS, Android and Windows if these three platforms are targeted. This provides the reason for the native look and feel to the cross platform application. The second portion is the shared code of the application that contains the business layer, data layer, data access layer, service access layer. These are the layers that can be shared across the different platforms that make the application a multi-platform application. Hence, a cross platform application targeting Android and iOS will have one shared project (directory), one Xamarin.Android project and one Xamarin.iOS project. (See Appendix D for more detail)

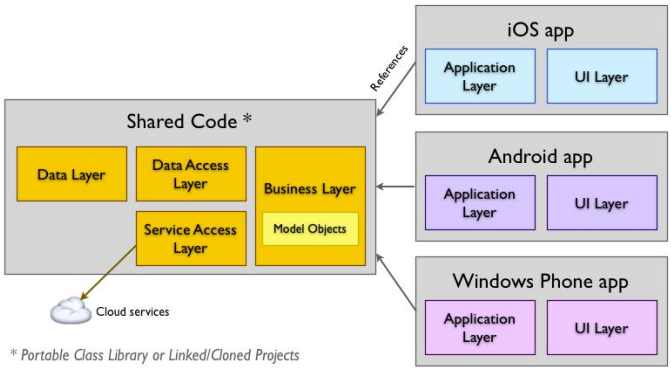


Figure 6.3.1: Xamarin Cross platform architecture

6.3.2 Prototyping & UML

After learning the architecture and before the application development, certain shape needed to be given to application and planned ahead. Hence, the UI prototype design that matches the similarities in between the platforms prepared can be seen in figure 6.3.2.

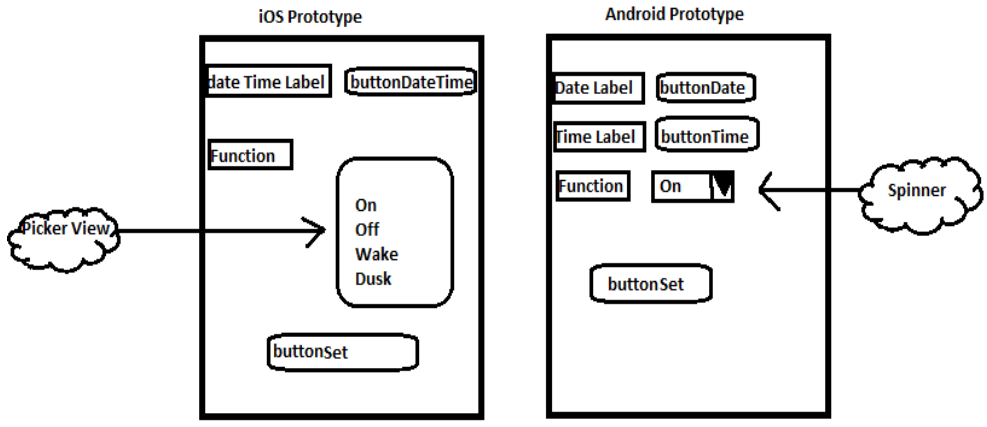


Figure 6.3.2: Cross platform UI prototype for iOS and Android

After designing the View part for the cross platform application, there was a need for designing the shared part and hence simple WULCustTimer can be seen in Figure 6.3.3 which consists of shared code across the platforms.

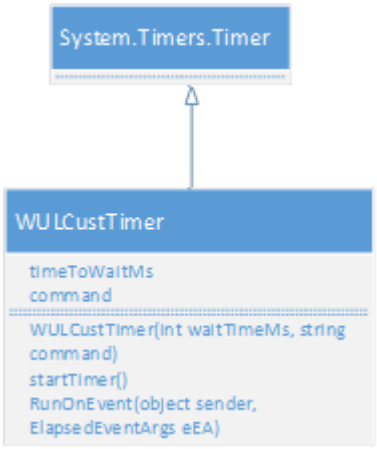


Figure 6.3.3: Shared Code UML

6.3.3 Cross platform implementation

Following the prototype and UML design, a cross platform application (Figure 6.3.4) was ready that shared codes between the two platforms (iOS and Android). Since, being a simple app, it only shared around 50 percent of codes. However, the shared code can be increased to greater extent with respect to the type of application built. This application lets the user select date and time, select a function and set an alarm. However, till this step the app has the ability only to trigger the info of the chosen function in the form of message, at user selected date and time. The next step was to really make the application to be able to trigger and send the function command remotely to the WUL.

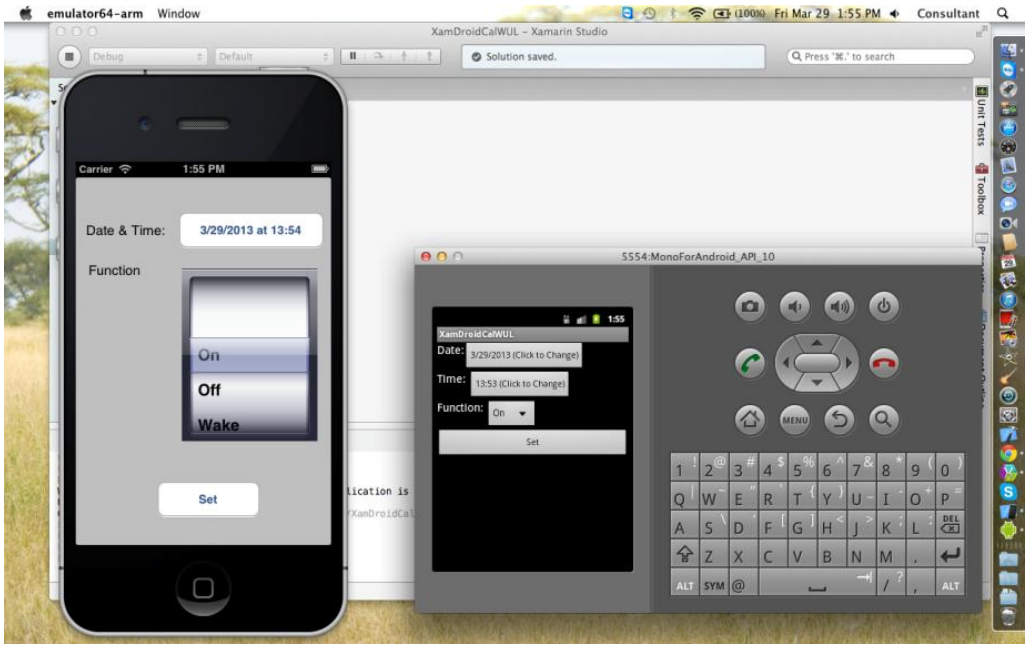


Figure 6.3.4: Cross platform WUL application running on iOS and Android

To send the commands to WUL, there were three methods as described in chapter 5.1. Webpage allows sending command directly from webpage. Soap was used to build native WUL application in chapter 5.2.1. For this application, as stated in the first research question, ICP Client needed to be used for sending the remote commands to the WUL.

6.3.4 Using ICP Client

The ICP Client library is originally written in native C language. Different versions of ICP Client library are available, built for Android, iOS and Windows. Java, Objective-C and C# wrappers are created on the native C ICP client library to be used in Android, iOS and Windows respectively. (See Appendix D for more details):

Xamarin supports the use of existing Java and iOS libraries. The binding feature in Xamarin can be used to access the Objective-C library as a C# library. [17]

- To reuse the Java libraries, it can be done in three ways: [18]

a) Create a Java Bindings Library – With this technique, a Xamarin.Android project is used to create C# wrappers around the Java types. A Xamarin.Android application can then reference the C# wrappers created by this project, and then use the .jar file.
b) Java Native Interface – The Java Native Interface (JNI) is a framework that allows non-Java code (such as C++ or C#) to call or be called by Java code running inside a JVM.
c) Port the Code – This method involves taking the Java source code, and then converting it to C#. This can be done manually, or by using an automated tool such as Sharpen.

Since, the ICP Client library obtained is originally native and wrappers are created around it to run on different platforms. Hence, instead of creating another C# wrapper using the methods stated above, instead, the native-C ICP Client can be used, using the Platform/Invoke (P/Invoke) mechanism supported in C# for gaining access to the native C functions present in the library.

The working mechanism of the ICP Client library to send commands to the WUL is:

a) Initialization must be done firstly. ICP Client Init()
b) Secondly, Sign On is performed. ICP Client SignOn()
c) The callback for the SignOn should be handled.
d) Finally, commands can be sent to the WUL by setting the data to upload and executing the command. (Data collection Portal)

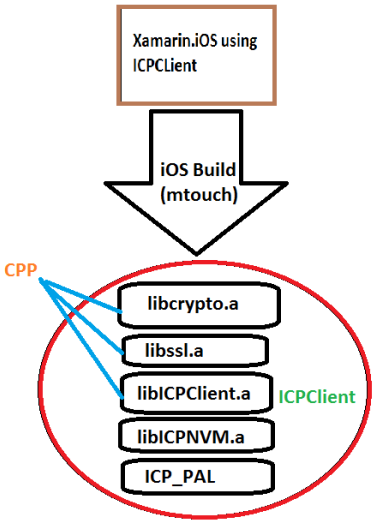


Figure 6.3.5: ICP Client architecture dependencies

As seen in Figure 6.3.5, the ICP Client is further dependent on different other libraries like libssl, libcrypto etc. The ICP Client, at the initial phase of its development by the Philips team referenced the NVM, whose implementations were done implicitly inside the ICP Client library in native C.

Implementing the ICP Client compiled for Android in the Xamarin.Android project, it was unable to consume the function. Ignoring the Java Native Interface (JNI) functions and trying to call the C library functions failed in this case. It was found out that, there have been changes in the architecture during the course of time related to the implementation of NVM in the ICP Client. This inability to consume the ICP Client function was due to the reason that, NVM interface in linked to the JNI and the NVM calls should be implemented separately by the application to fit to the new architecture. Implementing the NVM calls were out of the scope of this project and hence this step was skipped.

Moving forward to use the ICP Client compiled for iOS in Xamarin.iOS. In the iOS compiled version of ICP Client library, the same problem occurred which showed the missing implementations for the NVM functions as seen in Figure 6.3.5.

```
Process exited with code 1, command:
/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Devel
Undefined symbols for architecture i386:
  "_icpControlServiceGetThreadStatus", referenced from:
      _ProcessSubscribeEvents in libICPClient.a(icpSubscribeEvents.o)
  "_icpInvokeControlService", referenced from:
      _ProcessSubscribeEvents in libICPClient.a(icpSubscribeEvents.o)
  "_icpPAL_NVM_GetApplicationIdentity", referenced from:
      _icpConstructProvisioningBodyFunc in libICPClient.a(icpProvisioning.o)
  "_icpPAL_NVM_GetDeviceIdentity", referenced from:
      _icpConstructProvisioningBodyFunc in libICPClient.a(icpProvisioning.o)
  "_icpPAL_NVM_SetProductInfo", referenced from:
      _icpSetCredentials in libICPClient.a(icpProvisioning.o)
  "_icpPAL_NVM_SetPropertyByteArray", referenced from:
      _icpSetCredentials in libICPClient.a(icpProvisioning.o)
  "_icpPAL_NVM_SetPropertyString", referenced from:
      _icpSetCredentials in libICPClient.a(icpProvisioning.o)
ld: symbol(s) not found for architecture i386
collect2: ld returned 1 exit status

error MT5201: Native linking failed. Please review user flags provided to gcc: "-L/
```

Figure 6.3.5: Missing NVM implementation

Hence, the change in the architecture in the ICP Client became an unexpected bottleneck to see the consumption of use of ICP Client in the Cross platform application using Xamarin. Hence, the implementation of ICP Client library in Xamarin project was unsuccessful and the reason was identified.

The complicated looking ICP Client library was unsuccessful to be used in the Xamarin project due to the change in the architecture and the step chosen to implement it using the P/Invoke mechanism. So, a test was conducted to prove whether a native library can be used in Xamarin project or not. Hence, a simple native C library consisting of calculation functions like multiply and add was made. Then, using the Android Native Development Kit (NDK) toolset, the native C library was built for Android. The ICP Client we use in the project consist the same properties as this test library. Using the C# P-Invoke mechanism, it was possible to consume the function of this test library and therefore, it was proved that Xamarin supports the use of native-C library.

6.4 Result and recommendation (1st Sprint)

The problem in the NVM implementation was detected that lead in the inability to use the ICP Client library in the Xamarin project for the current moment. The result from the first research question is the information collected over Xamarin for the cross platform application development and the use of the ICP Client library in Xamarin.

Despite the inability to use the ICP Client library on Xamarin, the recommendation would be to make use of Xamarin for the cross platform application development because of the following reasons:

- **Native look and feel**
  Xamarin provides a cross platform application development with a native look and feel. It's not only the look but it is in fact native. The cross platform application created in Xamarin has a fully native look and feel. The UI elements have native look and feel.
- **High Performance app**
  Since, due to the direct full native API possible through the one to one mapping provided by the Xamarin, the performance capability is high as in the native application. The performance testing wasn't carried out but, as a normal user, no performance difference between this cross application built in Xamarin and the native application built during the kickoff (Chapter 5) were noticed. The UI response while interacting with the UI was similar.
- **Support for existing libraries**
  The existing C#, native C, Java, Android libraries can be used in Xamarin. This helps in the application development quite more efficient and faster. A native-C library built for specific platform (Android) was used in Xamarin to see this.
- **Excessive codes sharing**
  The UI can be separated and the business logic, data layer and network communication codes can be shared across platforms using the same language (C#) and .NET, including memory management, reflection, and the .NET base class libraries. Codes were shared during the development of the cross platform WUL application in Xamarin.
- **Cost effective**
  Though the Xamarin is a commercial product pricing ranging free- $1899 per user per platform, but in terms of the output received through the use of this product, costs can be saved in a long term.
- **Time efficient**
  Due to the code sharing ability across the platforms in the cross platform application development, lot of time can be saved in the development of application for different platforms.

  After achieving the stable version of ICP client library, different possible methods as described in 6.3.4 can be used to make the library workable to send remote commands to the Philips Products. Despite the inability to make use of ICP Client caused due to some missing implementation, creating a cross platform application using Xamarin is really a great tool for the development. Therefore, a recommendation to use Xamarin for the cross platform application development is made.

7 2nd Research question (2nd sprint)

This chapter describes the research learning, experience and recommendations for the second research question received.

After carrying out the first research question, pretty interesting techniques for developing a cross platform application was seen. However, a correct template for the development wasn't clear enough. Hence, to make the application of a common architecture, a need for an efficient pattern/model was raised. Hence, the second research question was defined out to make the development more efficient. The 2nd research question states: "A recommendation on the best model/approach/patterns for the maximum sharing of codes across the platforms in Xamarin."

Note: The Xamarin IDE used in this research is Version 4.0.2 with Xamarin.Android (Version: 4.6.7) and Xamarin.iOS (Version: 6.2.5.2). Appendix D provides detailed information for the 2nd Research question.

7.1 Time proposal

Like in the first research question, a time proposal for carrying out the task needed to be given. Using methods for calculating time (see Appendix C), a proposal of 5 weeks was made. After the proposal acceptance, learning and gathering information on patterns was performed.

7.2 Information gathering and Learning

7.2.1 Information sources

The investigation on the second research question was related to the recommendation of patterns. Lots of internet sources were referred for learning and the most helpful were:

- Joel.inpointform patterns differences: http://joel.inpointform.net/software-development/mvvm-vs-mvp-vs-mvc-the-differences-explained/
- Codeproject article on MVC and MVP: http://www.codeproject.com/Articles/288928/Differences-between-MVC-and-MVP-for-Beginners
- Msdn MVVM info: http://msdn.microsoft.com/en-us/library/gg405484(v=pandp.40).aspx
- Slodge blog on MVVMCross: http://slodge.blogspot.co.uk/search?updated-min=2013-01-01T00:00:00Z&updated-max=2014-01-01T00:00:00Z&max-results=50
- Github MVVMCross repo: https://github.com/slodge/

7.2.2 Learning Patterns and framework

During this sprint, lots on information on patterns and its different types were collected. Patterns are the design, to make the application development efficient by the separation of Views or User Interfaces (UI) and the actual business logics of the application. This enables to freely make changes in the user interface design without the change in the business logic implemented. It also increases testability, maintainability and extensibility of the software application built.

There are different types of patterns among which, most used patterns are Model View Controller (MVC), Model View Presenter (MVP) and Model View ViewModel (MVVM). In general, the View and the Model are the same across these different patterns. The view consists of the codes for the UI rendering and the Model consists of all the business logic/data. The different MVC, MVP and MVVM patterns concerns on the interaction of the Controller, Presenter and the ViewModel (VM) respectively with the View and the Model (See Appendix D for detailed information).

After the patterns learning process, there was a need to choose the model to be used in the Xamarin that could help in increasing the efficiency of code sharing strategy. After a lot of information

collection, MVC and MVVM patterns were seen to be applied in Xamarin by MonoCross and MVVMCross frameworks respectively. MVVMCross is built from the experience of MonoCross (a long ago Monocross fork[6] transformed from MVC to MVVM) and it is the most active framework at the current moment. Hence, further investigation on MVVMCross framework was decided to be carried out, that implements MVVM pattern.

7.2.3 MVVMCross (MVX) framework

MVX is an Open source cross platform framework for MonoTouch (Xmarin.iOS), MonoDroid (Xamarin.Android), WP7 and WinRT. It utilizes the MVVM C# pattern for the cross platform native application development. It was started by Stuart Lodge in November 2011. This framework makes extensive use of Portable Class library (PCL). [19] [20]

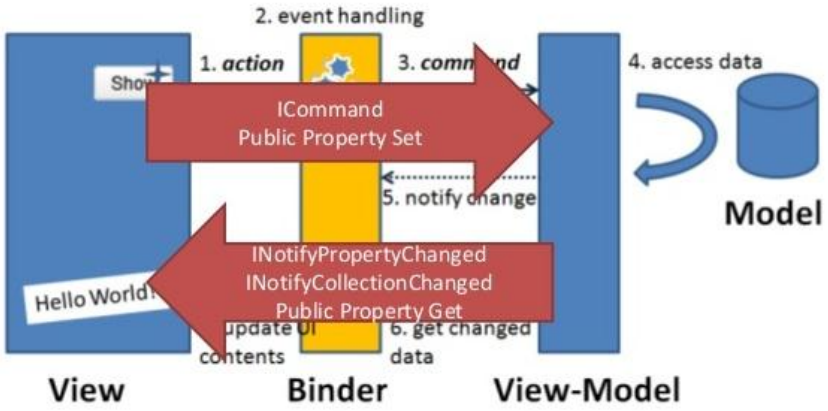


Figure 7.2.1: MVX implementation

MVX consists of three components: View (V), View-Model (VM) and the Model (M). View consists of all the UI rendering codes and the implementation of platform specific codes like push notifications in iOS or File I/O access across platforms. Model consists of all the business logic and services like database, web services. The VM acts as a mediator between the View and the model and consists of public properties that bind[7] with the View using the Binder provided by MVX framework. The use of MVX framework can be done after certain setup by adding classes and references (see Appendix D) needed for the framework to perform some automatic tasks like binding. Here, the View and the VM bind that makes instant update of the view when there is change in the public properties in the VM.

The main part of the MVVMCross framework is ICommand and INotifyProperty changed. Whenever there is certain action in the View (UI interaction), the action binds with the ICommand in the VM which then performs some defined manipulation. This manipulation in turn might make changes in the Model. Finally, after the manipulation, INotifyPropertyChanged is triggered. Finally, this will update/render the View accordingly. In this way, it can be seen that there is a decoupling of View and the Business logic, which are connected with a very thin layer using the MVX framework. (Figure 7.2.1)

Using the MVX framework helps to make an extensive sharing of codes between the platforms. The Views are implemented individually with the number of targeted platforms and shared codes can be reused using the PCL across the platforms. Like a simple Xamarin cross platform application solution, for an MVX application solution in Xamarin targeting Android and iOS would contain three projects:

[6] Copying the original open-source project in GitHub to make reuse or extension of it.
[7] Connected; changes with respect to the change in the other source it is connected to.

One Android project with Android View (View.Android), one iOS project with iOS View (View.iOS) and the one PCL (PCL.Core) project. This PCL.Core contains all the shared codes enclosing the Model and the VM.

The PCL used in MVX is of Profile 104. This consists of only the portable codes and cannot be used to implement the platform specific[8] implementations. This profile defines a small subset of .Net (.Net 4.5) that contains parts of the assemblies for:

- mscorlib
- System.Core
- System.Net
- System.Runtime.Serialization
- System.ServiceModel
- System.Windows
- System.Xml
- System.Xml.Linq
- System.Xml.Serialization

This profile of PCL consists of APIs available on all platforms. Hence, the cross-compatibility will be supported to maximum and the PCL can be reused easily again without the fear of containing any unsupported APIs (accidently used codes) in the shared code across platforms. Simple .Net libraries can't be referenced across different platforms (other than its compiled platform) but, the PCL works across different platforms easily. Hence, it looks reasonable for using PCL in MVX framework for the development of cross platform applications.

7.2.4 Different cases during MVX development:

During the MVX application development, different things related to the idea of implementing certain cases occur. Some of the cases learnt are as follows:

a) Multiple screens

Cross platform applications with multiple screens can be too created using the MVX framework. Since, the Views and the VM bind together using the MVX framework, VM is responsible for which screen to load. So, when there is a need of certain screen, the VM binded with its view can be activated. For example: When the application starts, the first VM activates which loads its binded view (first screen). So, to open a different screen when a button-click on this first screen, the click event of the first screen (button) is binded to the first VM. From this first VM, the SecondViewModel (VM) that is binded with the second Screen can be activated (as shown below). Hence, activating the SecondViewModel (VM) from the first VM will enable the app to show the second screen. This way, multiscreen applications can be built from MVX through VM transitions rather than from views.

```
public void GoToSecondVM ()
        {
          Base.ShowViwModel<SecondViewModel>();  //base class method
        }
```

Sequentially, adding ICommand's to ViewModel's, binding ICommand's to Views (Buttons), adding multiple VM, adding multiple Screens (View), finally navigating between the screens (View) using VM make a multiscreen MVX application.

[8] Related to specific platforms like the UI and device APIs (Bluetooth, Camera etc.)

b) Passing information between the screens

At certain point during the application development, there arises the need of passing information (data) between the screens. This can be accomplished simply by passing the required data using the VM (binded with the active screen) that activates the other VM (binded to the next screen) while loading next screen as below:.

```
//Method to go to the second VM
public void GoToSecondVM ()
        {
          Base.ShowViwModel<SecondViewModel>(new{passData="Hello"});
//base class method
        }
```

The reserved Init() function in VM (second) as seen below can be used to pass the information from the VM (first) above.

```
//VM reserved Init() method implemented in the second VM to pass data
Public void Init (string passData)
{
SetProperties= passData;
}
```

c) Gestures

As gestures relate to the platform specific implementation, they reside on the View project of the MVX application and they are implemented in each platform separately. There are various ways to implement gesture recognizer. For example, in iOS a tap gesture recognizer implementation would look something like this:

```
Var gesture= nwe UITapGestureRecognizer(()=>
{
//The operation to be performed on the tap gesture
})
View.AddGestureRecognizer(gesture);
```

Where, View is the main view where the gesture is to be implemented

d) NuGet/Plugin (Native abstraction)

MVVMCross NuGets are the packages available for an easy development. They contain a template (default Classes and assemblies) for the MVVMcross application development. These also contain the implementations of the Native APIs that can be shared across platforms. These NuGet packages (Plugin) can be added and used in the Xamarin projects. The MVX framework adapts the PCL project but the PCL doesn't support the platform specific implementations like accessing Cameras, Bluetooth etc. For this reason to implement the non-supporting codes in PCL the existing plugins can be used in the MVVMCross or a custom plugin can be created to be used to access the native APIs in PCL. However, for a custom plugin the implementation needs to be done on each number of targets. The following steps can be followed for creating a custom plugin:

1. Declaring common functionality (an interface[9])
   An interface can be declared in the PCL Core project that consists of the functionalities to be used which should be registered as a MVX plugin.
2. Writing platform specific implementations
   The platform specific implementations for the specific platforms can be done in the view project (iOS and Android) which implements the interface declared in the PCL Core project.
3. Using the interface and not the implementation
   The interface is declared and the implementation is done in the platform specific view project. Now, for consuming the platform specific functions can

e) Xamarin.Mobile

Xamarin.Mobile is a library that exposes a single set of APIs for accessing common mobile device functionality across iOS, Android, and Windows platforms. This increases the amount of code developers can share across mobile platforms, making mobile app development easier and faster. Xamarin.Mobile currently abstracts the contacts, camera, and geo-location APIs across iOS, Android and Windows platforms. Future plans include notifications and accelerometer services. [21] It maps to native implementation on each platform.

Figure 7.2.2: Xamarin.mobile

For Example, to use contacts on Android and iOS, we can use:

```
Var book= new AddressBook(){PreferContactAggregation=true};
Foreach(Contact c in book.Where(c=>LastName== “search keyword”)){
Console.WriteLine(C.DisplayName);
Foreach(Phone p in c.Phones)
Console.WriteLine(“Phone: ”+p.Number);
Foreach (Email e in c.Emails)
Console.WriteLine(“Email: “+e.Address);}
```

[9] Interface is a type that exposes the behavior of the services that needs to be provided. It doesn’t contain any data but it contains only the behaviors.

Xamarin.Mobile can be a boon in the cross platform development using Xamarin and was planned to be used in the cross platform development. But, when adapting the MVVMCross framework, even Xamarin.Mobile seems less important. MVVMCross provides plugin features containing lots of single API access (functions) that Xamarin.Mobile includes (except for Contacts) and many other functions that Xamarin.Mobile doesn't include, like Accelerometer, Bluetooth etc. Hence, MVVMCross plugins are superior to the Xamarin.Mobile. Therefore, looking at the existing plugins and ability to create a custom plugin in MVVMCross, the use of Xamarin.mobile can be skipped and MVVMCross plugins can be used instead.

f) Native libraries

Getting the Platform/Invoke (P/Invoke[10]) mechanism work (Appendix D. Research Question 1. II.D2), to use the functions in the native libraries in the PCL, then the same procedure can be followed as mentioned above in using Plugin (7.2.4.d).

g) Features

The advantages of MVVMCross include: [22]

- Binding support on iOS (XIB) and Android (XML).
- A lot of app behavior gets put into VM and is reusable across platforms.
- Since VM are platform-agnostic unit tests can be written that covers most of the app's behavior.
- View models are shared, but views remain totally native.
- The framework itself is very extensible, allowing for a lot of customization when needed.
- Lots of most demanding scenarios, like high frame rate gaming and complex data visualizations have been built using the Xamarin and MVVMCross framework. For example: Kinect Star Wars, Aviva Drive, The CrossBox DropBox Client etc.

7.2.5 Experience with MVX development

After learning and gathering information about the MVX, it was turn to use the framework to be applied in the same application that was built in the 1st sprint i.e. An alarm application that could send commands to WUL at user selected date and time. To begin with the implementation, architecture for this MVX WUL alarm application was made as seen in Figure 7.2.3. The UI remains the same for this application but there adds different things like binding, ICommand etc.

[10] A mechanism in C# to access functions written in native (C) languages from C#

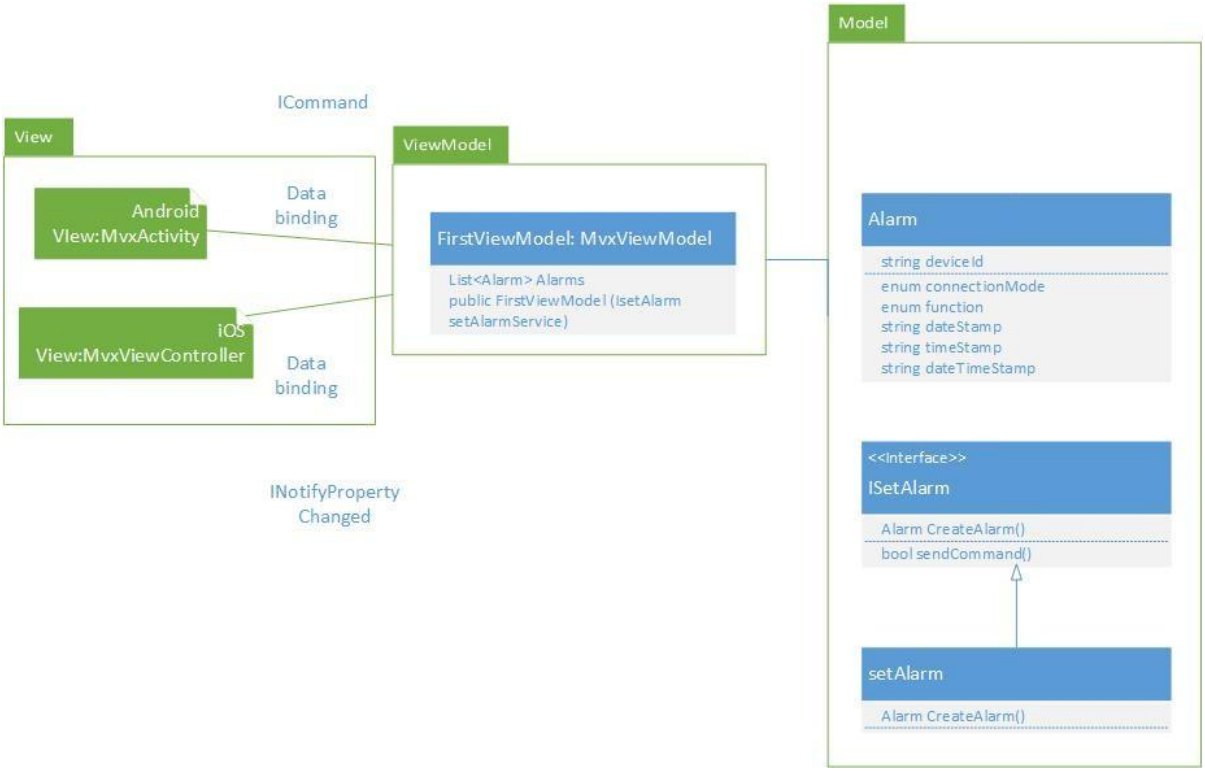


Figure 7.2.3: MVX architecture for the WUL Alarm app

After designing the architecture for the app, implementation was carried out and several things were experienced. The biggest problem is the setting up the MVVMCross framework. To utilize the MVX framework, NuGet packages are used officially in VisualStudio IDE using the Xamarin.iOS and Xamarin.Android plugin. Using the Visualstudio for the development, the PCL support for the Xamarin.iOS and Xamarin.Android frameworks doesn't exist by default. Hence, tweaks must be done in order to make use of them [22]. Even following the tweaks didn't work out in this project. The NuGet package for using the MVVMCross requires NuGet 2.5 which isn't yet finished/ released for Xamarin Studio on Mac. Hence, unable to use the NuGet, all the references, required classes should be done manually on Mac [23]. Even after the modeling of the application, due to these problems at the present context, while trying to manually work with the assemblies for the MVVMCross framework couldn't be set properly and consumed more time than estimated. Therefore, the straight forward getting started with the MVVMCross isn't yet perfect. Hence, though MVVMCross being powerful, the initial setup is really the main bottleneck during the development.

The main advantage of using MVX can be clearly seen in bigger projects while managing, sharing and testing. In smaller projects it only makes things complicated rather than making life easier. The previously built WUL application in the 1st research question architecture (using only Xamarin) to adapt the MVX is quite a small project. Hence, MVX isn't that efficient in in this case that showed only significant improvements while sharing codes but adding much more complications of binding stuffs.

7.3 Result and recommendation (2nd Sprint)

From the 2nd sprint research activity, MVVMCross framework that implements MVVM pattern was discovered to be used in Xamarin for increasing the shared code between the platforms. Xamarin with MVVMCross framework is a very powerful combination for the cross platform mobile application development. These two combined boosts up the cross platform application development process by giving a single language of C# for the development across different platforms. They enable shared codes across different platforms, provide native look and feel to the cross app, support native and existing libraries, high performance apps and large applications have a common architecture. However, for the cross platform support, the use of different IDEs (Xamarin, MonoDevelop, VisualStudio) for the development using Xamarin components and MVVMCross framework make a tiresome work to find the correct path. But once, the correct path is known and setup, it makes the whole life easier for the cross platform application development. Hence, using this framework is efficient in larger projects rather than in smaller ones. To conclude, a native-cross platform application can be built with excessive sharing of codes across platforms and a clear decoupling of UI and the business logics can be done using Xamarin and MVVMCross framework combined.

Therefore, the recommendation would be to make use of MVVMCross framework (MVVM pattern) for the development of cross platform mobile application. Despite of the disability to set up the framework for use, based on the research information on the features and supports of Xamarin and MVVMCross, they should be applied in the cross platform mobile application development.

8 Progress Tracking

There are different activities planned to be performed during a project. This chapter describes the different ways of tracking of these activities during this project.

8.1 Mentor meeting

Whenever any suggestion was required, when there was need of knowing the status of the project, definition of a new research needed to be done, result on the research question is to be given, a meeting was planned with the company mentor. This meeting was really helpful and educational too due to the suggestions received during the meetings.

8.2 Scrum Standup meeting

Each Tuesday and Thursday at 11:45 AM, a scrum standup meeting takes place. During this meeting, each member in the team gives a short description on what the member did previously, is doing currently and has planned next. This meeting, though being short, is fruitful and gives a proper idea in exchanging information between the team members.

8.3 Documented status

The position of the project was viewed every week in the Project phasing diagram (Figure 3.6.1 or see Appendix B: Project Plan). Since, there were different sub-activities defined while carrying out the research questions. These were tracked using a different table.

The first research question was planned to be performed started week 6 and ending week 11 of the project. The second research was planned to be performed started week 13 and ending week17 of the project. At the end of each activity during the research, a status was assigned to the activity, some comments on the activity and a decision whether to step the next activity or not was taken. Changes could be made in the plan if any findings during an activity need any change. (For more details, see Appendix C. Time proposal for Research Questions). The sub-activities planned within the research questions were successfully adapted and completed (Table 1 and Table 2). However, in research sometimes we meet some unexpected things as in week 17 (see Table 2), where the planned activity was changed due to the unfit with the context.

8.3.1 Tracking the first research question

Weeks	Activities	Status & comments at the end
Week 6	Learning Xamarin studio to develop a simple Native iOS (HelloXamiOS) and Android (HelloXamDroid) apps in C# for the first 1 week that consists of simple UI buttons and shows some text messages on the display.	Done. Gained basic knowledge on Android and iOS app development on Xamarin that uses C#. Next activity can be stepped as planned.
Week 7	Learning and gathering information to be able to develop a simple cross platform application on Xamarin Studio. Simple cross platform app refers in making use of shared codes between the different platform projects.	Done. The techniques for developing a simple cross platform application is acquired and hence next activity can be adapted
Week 8	Develop a cross platform	Done. A simple cross platform application is created

	XamWULCal app that can trigger a method (that shows an alarm message on the display) at user chosen date and time.	that fulfills the activity for Week8. Hence, Week 9 can be stepped further.
Week 9	Investigation on the use of native iOS and Android libraries on Xamarin.	Done. The methods and techniques for the usage of native iOS and Android libraries on Xamarin were known. However, to successfully complete the week 10 activity, additional support is needed to perform the activity due to no previous experience in usage of library written in different language than the platform to be used in.
Week 10	Importing the ICP Client library on the XamWULCal built in week 8.	Done. The C# version of ICP Client library (C# wrapper around C) was provided and hence simply added the ICP Client library to the project. Next, activity can be stepped and the C# wrapper class can be used in Xamarin to access the function of ICP Client library.
Week 11	Finish the XamWULCal app that enables to send commands to the WUL via ICP Client at user specified date and time.	Not possible for this moment. The ICP Client (compiled for Android and iOS) was imported to Xamarin.Android and Xamarin.iOS project in order to send commands to WUL. But, due to the uneven version of libICPClient where the NVM doesn't implement all the functions referenced by libICPClient, it wasn't possible to send commands to WUL for this moment. Hence, the task was carried out as planned but the result didn't appear as expected, to be able to send commands to WUL via ICPClient.

Table 1: Tracking Research question 1

8.3.2 Tracking the second research question

Weeks	**Activities**	**Status & comments at the end**
Week 13	Learning about Windows Presentation Foundation (WPF) patterns/model.	Done. Gathered information on WPF and shall proceed to next step.
Week 14	Learning different types of WPF models and their uses in creating different types of applications. Viewing the models adapted on existing Xamarin example applications.	Done. Learnt about MVC, MVP and MVVM. Frameworks like MonoCross (uses MVC) and MVVMCross (uses MVVM) are popular for the cross platform application development in Xamarin. Next, step can be taken.
Week 15	Learning about Xamarin.mobile	Done. It was learnt that Xamarin.mobile provides single set of API access. But, this library cannot be used directly while adapting the MVVMcross framework. Hence, instead the available plugin in the Nuget can be used that provides many more API accesses besides GPS and accelerometer. Next step can be taken.
Week 16	Choosing the most effective model for Xamarin Cross platform application development that makes high reuse/sharing of codes and designing a prototype cross platform app (with respect to the	Done. MVVMCross framework chosen. The design of the app is designed as the same app created before. i.e. WULCalendar app but using the MVVMCross framework. Since, the WULCalendar app doesn't make use of any APIs present in Xamarin.mobile, the next step can be replaced with the implementation of the application.

	chosen model).	
Week 17	*Implementing Xamarin.mobile to fit the design in Week 16.*	*Changed. Xamarin.mobile doesn't fit the design and also better option (MVVMCross plugin) was found.*
Week17	implementation of MVVMCross framework	Not completed. Due to the unexpected setting up (tweaks) required for the framework deployment and fragmentation in IDE for the development support, the task for implementing the framework couldn't be completed successfully.

Table 2: Tracking Research question 2

9 Conclusion and recommendation

This chapter consists of the final conclusion and recommendation summarized from the results and recommendations from the 1st research question (see Chapter 6.4) and 2nd research question (see Chapter 7.3).

The goal of the project to research and provide recommendation on the received research questions and to minimize the problems that rose during the mobile application development across Android and iOS is successfully met.

Firstly, a proof of concept was delivered by running a cross platform alarm application on Android and iOS using Xamarin. It positively showed the sharing of codes across the platforms while keeping the user experience still native. A recommendation to make use of Xamarin for the cross platform application development is given.

Secondly, MVVMCross is recommended to be the best framework that implements the Model View ViewModel (MVVM) pattern. The recommendation is based on the research learning (Appendix D) on the framework.

To recapitulate, Xamarin with MVVMCross framework is a very powerful combination for the cross platform mobile application development. Hence, the recommendation is to make the use of Xamarin (Xamarin.Android and Xamarin.iOS) and MVVMCross framework (that implements the MVVM pattern) for the cross platform application development.

Evaluation

Here, I describe my personal experiences on how I tried to eliminate the bottlenecks, my weakness and strength, the learning for future career, the working environment and the memorable things during the project journey at the company.

During the project period, the problems that arose were firstly tried to tackle using the internet. Then, secondly I took help from my supervisor when got stuck on that problem. The help provided in the form of suggestion was really a great relief. I learned a lot during the project and also found out some of my weaknesses and strength. My greatest weakness was a lack of communication with people due to my shy nature, but there were friendly people around to make me more comfortable and hence I tried to improve slowly to increase the communication. My strong point was the ability to give my best effort on my received task and the creativity skill (methods in Appendix D), though not very efficient. The everyday blogging about the carried out activities helped a lot in reviewing/refreshing the past knowledge gained and also in writing the final report.

Related to mobile development before starting the project, I only had the knowledge of Android development from my previous internship and university. The iOS world was totally new for me. Moreover, the cross platform development sounded like a magic to me before this project. But now at the end of this project, I have gained the knowledge of mobile application native and cross platform application development targeting Android and iOS platforms. I can really make this mobile application development proficiency as my future career.

Working with the DI team at Philips is really one of the greatest opportunities I had in my life. The way of working and the characteristics of people around in the DI team is really a pleasant entity to have at work. The soothing working environment really makes the work a fun rather than a tedious job, where everyone is ready to help you with their best effort. The first memorable moment working at Philips is the project and sport interactivity with my company supervisor that I never experienced before and it was the best. The second remarkable moment was the saying from my company supervisor “I try to push you from your limit, but don’t get demotivated”. This saying when understanding deeply really encouraged me to perform my task. Sometimes during the project, I dived deep inside and completely lost in search of the result. At that moment, my company supervisor helped me pull out from there and placed back in a comfortable situation. The most important lesson I learnt from him is “Don’t focus on the result. Plan and follow the process, result is the end of the process.” The final remarkable moment at the company was the continuous helpful suggestion and inspiration from Mr. Maurice Hebben (DI team member). Hence, working at Philips was educational, fun, social, motivational, future career developing and a wonderful experience.

References/Literatures

[1] "Wikipedia info on Philips," [Online]. Available: https://en.wikipedia.org/wiki/Philips.

[2] "Eindhoven.nl info on the HTC planning," [Online]. Available: http://www.eindhoven.nl/ruimtelijkeplannen/plannen/NL.IMRO.0772.80022-/NL.IMRO.0772.80022-0104/t_NL.IMRO.0772.80022-0104_1.2.html.

[3] "Wiki info on SDLC," [Online]. Available: https://en.wikipedia.org/wiki/Software_development_process.

[4] "Wiki Info in Scrrum method in SDLC," [Online]. Available: https://en.wikipedia.org/wiki/Scrum_(development).

[5] "SmartBear SoapUI official site," [Online]. Available: http://www.soapui.org/About-SoapUI/what-is-soapui.html.

[6] "Xamarin Official website," [Online]. Available: http://xamarin.com/.

[7] "Wiki Info on SOAP," [Online]. Available: https://en.wikipedia.org/wiki/SOAP.

[8] A. GJ, "ICP Client API.docx (Philips Internal Document)".

[9] "passing data between ios apps," [Online]. Available: http://kotikan.com/blog/posts/2011/03/passing-data-between-ios-apps.

[10] "VC Push Segue and Modal segue," [Online]. Available: https://www.youtube.com/watch?feature=endscreen&NR=1&v=Ab5jyXihwRM.

[11] "Xamarin Guides documentation," [Online]. Available: http://docs.xamarin.com/guides.

[12] "Youtube with Xamarin search," [Online]. Available: http://www.youtube.com/results?search_query=xamarin&oq=xamarin&gs_l=youtube.3..35i39l2j0l8.1064.1924.0.2358.7.7.0.0.0.0.76.377.7.7.0...0.0...1ac.1.11.youtube.LBBmLud0S4U.

[13] "StackOverflow with Xamarin tag," [Online]. Available: http://stackoverflow.com/questions/tagged/xamarin.

[14] "Xamarin Seminar by Greg Shakles," [Online]. Available: https://www.youtube.com/watch?v=WkNbRUqnSSc.

[15] "Xamarin.Android limitations," [Online]. Available: http://docs.xamarin.com/guides/android/advanced_topics/limitations.

[16] "xamarin.iOS Limitations," [Online]. Available: http://docs.xamarin.com/guides/ios/advanced_topics/limitations.

[17] "Xamarin Doc on Objective C library integration," [Online]. Available: http://docs.xamarin.com/guides/ios/advanced_topics/native_interop.

[18] "Xamarin Doc onJava Integration on Xamarin," [Online]. Available: http://docs.xamarin.com/guides/android/advanced_topics/java_integration_overview.

[19] "Github repo for MVVMCross," [Online]. Available: https://github.com/slodge/MvvmCross.

[20] S. Lodge, "Blogspot on MVVMCross Tutorial," [Online]. Available: http://slodge.blogspot.co.uk/search?updated-min=2013-01-01T00:00:00Z&updated-max=2014-01-01T00:00:00Z&max-results=50.

[21] "Xamarin.Mobile," [Online]. Available: http://xamarin.com/mobileapi.

[22] "Nabble.com forum on MVVMCross experience," [Online]. Available: http://monotouch.2284126.n4.nabble.com/MonoCross-and-or-MVVMCross-experiences-td4657938.html.

[23] "PCL setup in visualstudio for Xamarin projects," [Online]. Available: http://slodge.blogspot.co.uk/2013/04/my-current-pcl-setup-in-visual-studio.html.

APPENDICES

A. Graduation Project Survey

1. Describe the problem analysis:
   (What is the reason for the internship company to initiate this assignment? What is for the company the added value of this assignment? Can you describe the starting situation and starting points: introduction and problem definition?)
   - The company wants to investigate on developing multiplatform apps according to the company's Digital Innovation (DI) architecture. Documentation and a running multiplatform application. This will provide information on their current architecture and their customer can gain a better user experience through multiplatform app.

     There are different devices of company and there is need of multiplatform application which can be used to interact with those devices and learning the DI architecture of the company is the starting situation. While building the multiplatform application, missing of architectural information can lead in some problem or it can be that, the multiplatform application doesn't support the company's whole DI architecture.

2. Describe the assignment.
   (Especially the objectives results to be delivered and final products to be realized. Also indicate what you want to achieve for the internship company. Give a clear description of the graduation assignment.)
   - The main goal of the assignment is to investigate and document on developing mobile apps according to the Philips Digital Innovation architecture with a supporting running example.

     From the internship, I want to achieve better knowledge in multiplatform mobile applications and connecting these applications with lifestyle products.

3. What is the research component of this assignment?
   (What are your research topics? If necessary, draw up a research plan.)
   - The research component is to investigate how to develop a (mobile) App, based on the company's DI architecture that can be targeted for the different mobile platforms (Android, iOS, Windows-Mobile, BlackBerry-OS, and maybe more).
4. What are the methods and tools?
   (What operating procedure and means can you make use of, during the internship period at the company? What facilities will be made available by the company?)
   - Not yet known about all the methods, tools and facilities.

5. How and by whom will you be guided by the company?

   - By Mr. Patrick Bonné, providing information and requirements, those are needed to carry out the assignment successfully.

6. What fields of Study play an important factor in realizing the assignment?
   (For example, information analysis, design, realization, monitoring and security.)
   - Project planning, feasibility study.
   - Analysis, Requirements definition.
   - Mobile computing.
   - Digital Innovation
   - Consumer lifestyle
   - Information & Software development

B. Project Plan (PP)

B.1. Introduction

Connectivity for consumer lifestyle is a project that is carried under the Digital Innovation (DI) development program of Philips to connect life style products. This enables to control the devices remotely to provide better user experience by combining the functionalities of different devices in an intelligent way. For example: Switching the lights ON and OFF or increase and decrease the intensity of light through smartphones or automatically with respect to the time settings.

The project assignment includes investigating on how to develop a mobile application based on the Philip's DI architecture, targeted for different mobile platforms such as Android and iOS (iPhone OS). It includes providing recommendations on the research questions based on the project using the investigated information and demo example application.

This Project Plan (PP) contains the project statement, project phasing, project management plan and the organization chart that will help in understanding the project in detail. This will also help people engaged in the project (student, supervisors) in future, to keep track of the project path and situation.

B.2. Project Statement

B.2.1. Formal client

Mr. Patrick Bonné will be performing the role of a client (supervisor) and Mr. Rob Knubben will be asking the research questions needed for the project carried.

B.2.2. Project leader

Mr. Saurav Aran will act as the project leader for carrying out the project and also responsible for communication between the company and the university.

B.2.3. Current situation

The everyday lifestyle products manufactured by the company such as Coffee machines, Wake up lights, Home cooker etc. are the Digital Innovation (DI) products. These products are possible to control or send information via the use of smartphone and have the ability to get connected to the company's main server.

The product innovation is assumed to be near to the release phase.

B.2.4. Project justification

This project was started in order to assist the DI team. The assistance includes researching on the best use of the cross-platform applications for the company according to its DI architecture. Different research questions will be presented which needs to be investigated and given recommendations with reasoning and example cross-platform application.

The current challenge for the client is the presence of different mobile platforms (Android, iOS, Windows, BlackBerry-OS etc.) devices (smartphones, tablets) from where the products are to be connected. The client wants to achieve the possible best way to make the use of their products through these different platform smart devices. Also, the cost effectiveness, speed and maintainability are the major challenges.

B.2.5. Project product

The main task of the project is to investigate how to develop an app (mobile), provide recommendation with respect to the research questions and to build/extend existing example application according to the company's DI architecture that works at least on Android and iOS devices. There can be 3-4 research questions during the whole project.

The 1st research question: How is it like to develop a cross platform application for the WUL using the "Xamarin Studio" Integrated Development Environment (IDE) that is able to control WUL's functionalities using pre-existing ICP Client library? The recommendation consist brief useful information on the approach.

The 2nd research question: A recommendation on the best model/approach/patterns for the maximum sharing of codes between the platforms in Xamarin. Xamarin.mobile can be used to support the recommendation.

B.2.6. Project deliverables and non-deliverables

a) Project Plan. Design Document & User Requirement Specification (URS) isn't required. However, during the project, some parts will be included in chapter 2.vi.c.
b) The source code of extended demo application (no User's manual required) and the weekly notes.
c) The recommendations with respect to the research questions including working prototypes.
d) A final report (University report format) consisting of detailed description and learning on the project.

B.2.7. Project constraints

- The use of ICP (Internet Connected Products) client which is used to connect the Philips products and the smart devices via the Philips back-end server should be taken into account during the whole project.
- End user experience should be unchanged to maximum extent keeping it similar.
- Fixed time duration of 6 months starting Feb 1st - July 31st, 2013.

B.2.8. Project risks

Time, money and quality:

- Risk Level: High and can have a high chance of occurring.
- Condition and result: The project can run out of time and also the money is limited to provide a certain project quality. This can cause a serious effect on the project.
- Prevention: Following time-boxing and reaching the milestones as planned.
- Alternative: Time and money are constant and hence the quality will decrease.

Deprive of Project materials:

- Risk Level: Low and have a minimum chance of occurring.
- Condition and result: During the project if the materials (e.g. different platform devices) needed for the project are not provided in time, the project can be affected.
- Prevention: Providing the required project materials on time.
- Alternative: Using simulators in case of unavailability of the devices. The project group may cope with this risk.

B.3. Project Phasing

Basically the project phasing (Tasks) can be divided into the following as below:

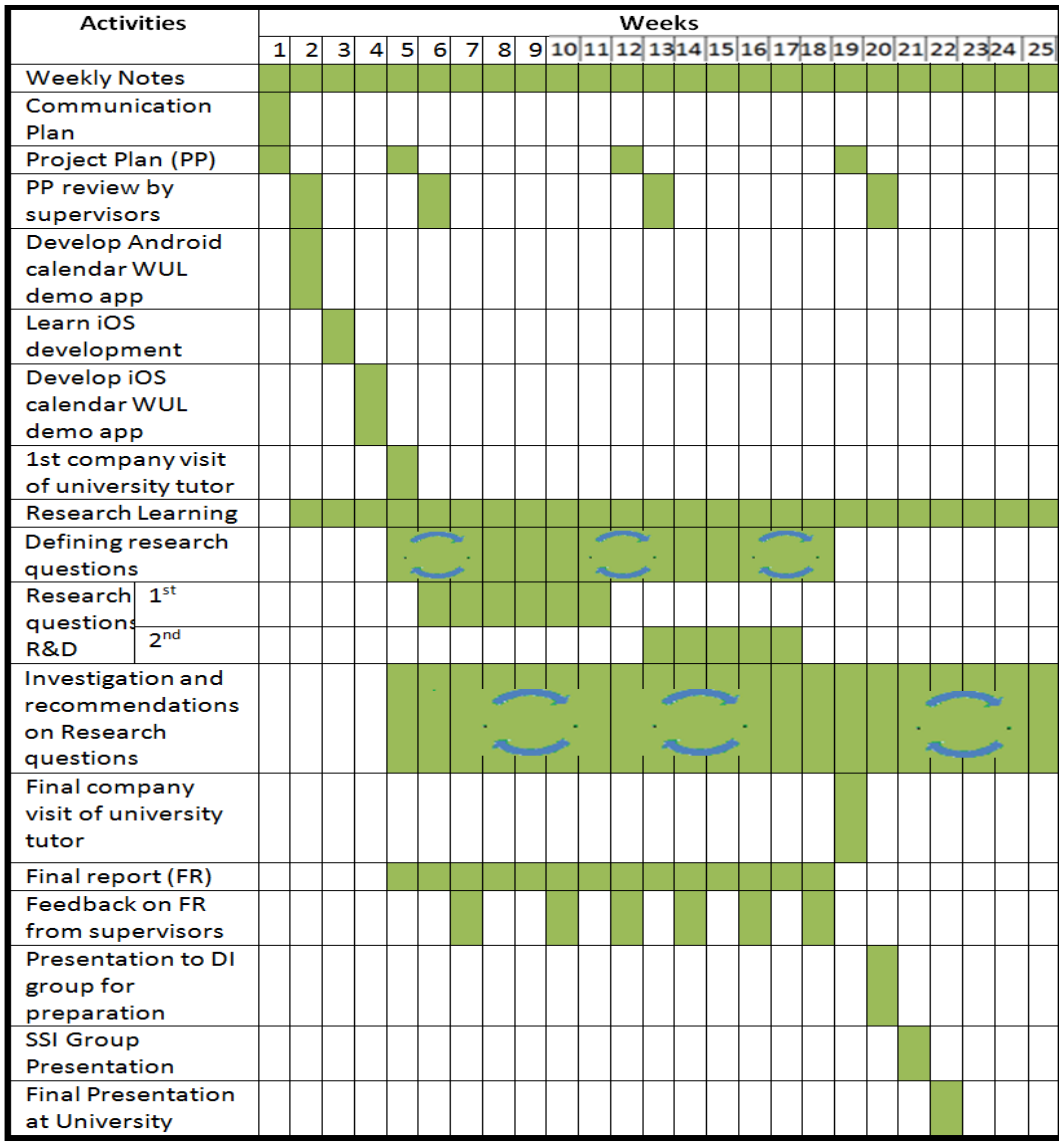


Figure B.3: Project phasing

There are different activities to be carried out which can be represented as the phases in the project as seen in Figure B.3.

Weekly Notes: The weekly notes will be the collection of notes written everyday about the activity and learning. This will last till the end of the project.

Communication Plan: Since, this is just the documentation describing the method of communication with the company and university tutors. It will just consume the first week of the project.

PP: As, we aren't following the waterfall model for the project; the PP will be updated after every 1 month (approx.) i.e. week 1, 5, 12 and 19. The PP reviews by the supervisors will be done a week after the PP is updated i.e. week 2, 6, 13 and 20.

Develop Android calendar WUL demo app: This part is basically for learning and refreshing the knowledge on Android platform. The task will be to develop/extend an Android app that lets the user to control the WUL at user specified date and time (like an alarm setting). Due to the existing knowledge on Android platform, the demo app development on the Android is only planned for a week.

Learn iOS development: Since, the iOS platform being a totally new platform, the learning time for making a demo application version of calendar app on iOS, one week is assigned.

Develop iOS calendar WUL demo app: The demo app will be developed in a week after the learning is done. The feature of the app is same as developed app in the Android platform in week 2

1st & final company visit of university tutor: The 1st visit of the university tutor to the company for a detailed info on the graduation internship project information is planned on week 5. The final visit for the completion of the graduation project is planned on week 19.

Research Learning: Learning is required almost throughout the project so as to be able to work on the project.

Defining research questions: Research questions will be defined during the project multiple numbers of times that will last till week 18 starting from week 5. (See Appendix C for the Time proposal duration calculation/method adapted for the 1st and 2nd research questions and the sub-activities).

- **1st Research question:** The first research question is defined in week 5. The research and development on the 1st research question will start from week 6 and end in week 11 with duration of 6 weeks.
- **2nd Research question**: The second research question is defined in week 12. The research and development on the 2nd research question will start from week 13 and end in week 17 with duration of 5 weeks

Investigation and recommendation on research questions: Since, there arises different new research questions while performing the project, this phase is iterated number of times during the project.

Final report: The Final report writing is planned to start from week 5 till week 19. The review and feedback on the report is planned to be done by the supervisors on week 7, 10, 12, 14, 16 and 18. This is done so as to have a good final report as per the university's standard. The deadline for handing in the final report is June 10, 2013.

Final presentation: The final presentation of the graduation project at the university is scheduled to be held on week 22 (July 3, 2013). Before the final presentation at the university, a preparation presentation is planned on week 20 to the local members in the team and the final presentation for the Software System Integration Group (inside company) on week 21.

In the phasing (Figure 1) we can see the final report and the final presentation ending before the project and some other activities because the project contract is longer than the graduation time.

B.4. Project Management Plan

The project management plan on the MOSQUITO (Money, Skills, Quality, Information, Time and Organization) aspects are as follows:

B.4.1. Money

The materials and resources required for the project will be provided by the company. The project fine will be in terms of grading (marks) the student at the project end, since it is a graduation internship carried out for educational purpose.

B.4.2. Skills

Tasks	Skills Required
Weekly Notes	Active and a good skill of noting the activities carried out.
PP	Knowledge of planning the project and foresee the whole project beforehand that will help in evaluating the project.
Demo Application	Knowledge of programming for Android and iOS application development that includes Java, Objective-C & prototyping.
Research Learning	Ability of learning new skills, self-questioning and acquiring the new terms is required. Also, ability to learn and implement new programming functionality according to research questions is needed.
Investigation and recommendations on Research questions	The skill of comparing different research learning, current information and a good ability to give proper recommendation that would best fit the project context. Also, knowledge of Android and iOS application development is required.
Final report	A good skill of elaborating how the project was conducted and what learning were performed that suits according to the University report format.
Final Presentation	Having a good overview of the whole project and ability to describe them in a simple way, assuming himself/herself as an audience.

Table B.4.2: Skills required in the project

In B.4.2, we can see the skills required to perform different tasks during the project.

B.4.3. Quality

In the project, quality refers to the level of investigated documentation and information provided on the research questions of the project. The recommendation includes descriptions, advantages, disadvantages and brief information on the research questions. The quality result too refers to the built/extended mobile application supporting the number of different platforms.

B.4.4. Information

	PP	WN	URS	PSC	RR	FR
Mr. Patrick Bonné	R, Di, A, Ar	R, Ar	R, Di, A, Ar	R, Di, A, Ar	R, Di, Ar	R, Di, A, Ar
Mr. Rob Knubben					R,Di	
DI team	IP					
SSIGR	IP					
Mr. Marco	R, Di, A, Ar	R, Ar	R, Di,	-	R, Di, Ar	R, Di, A,

Dorenbos			Ar			Ar
Student	S, Di, Ar	S, Ar	S, Di, Ar	S, Di, Ar	S, Di, Ar	S, Di, Ar

Table B.4.4: Information Table

Legend

SSIGR:	Software Systems Integration group Research		
PP:	Project Initiation Document	**S:**	Send
WN:	Weekly Notes	**R:**	Receive
URS:	User Requirement Specification	**Di:**	Discuss
PSC:	Project Source Code	**A:**	Approve
RR:	Research Recommendation	**Ar:**	Archive (conditions may apply)
FR:	Final Report	**IP:**	Internal presentation

B.4.5. Time

Tasks	**Time**
Weekly Notes	Written during the whole project period
PP	5 weeks
Demo Application	3 weeks
Research Learning	25 weeks
1st Research question	6 weeks
2n Research question	5 weeks
Investigation and recommendations on Research questions	22 weeks
Final report	15 weeks
Presentation preparation	4 weeks

Table B.4.5: Time required in the project for different activities

Table B.4.5 shows the time distributed/required for the activities in the project. The total time for the whole project is 26 weeks and some tasks are performed in parallel to each other as seen in Figure B.3.

B.4.6. Organization

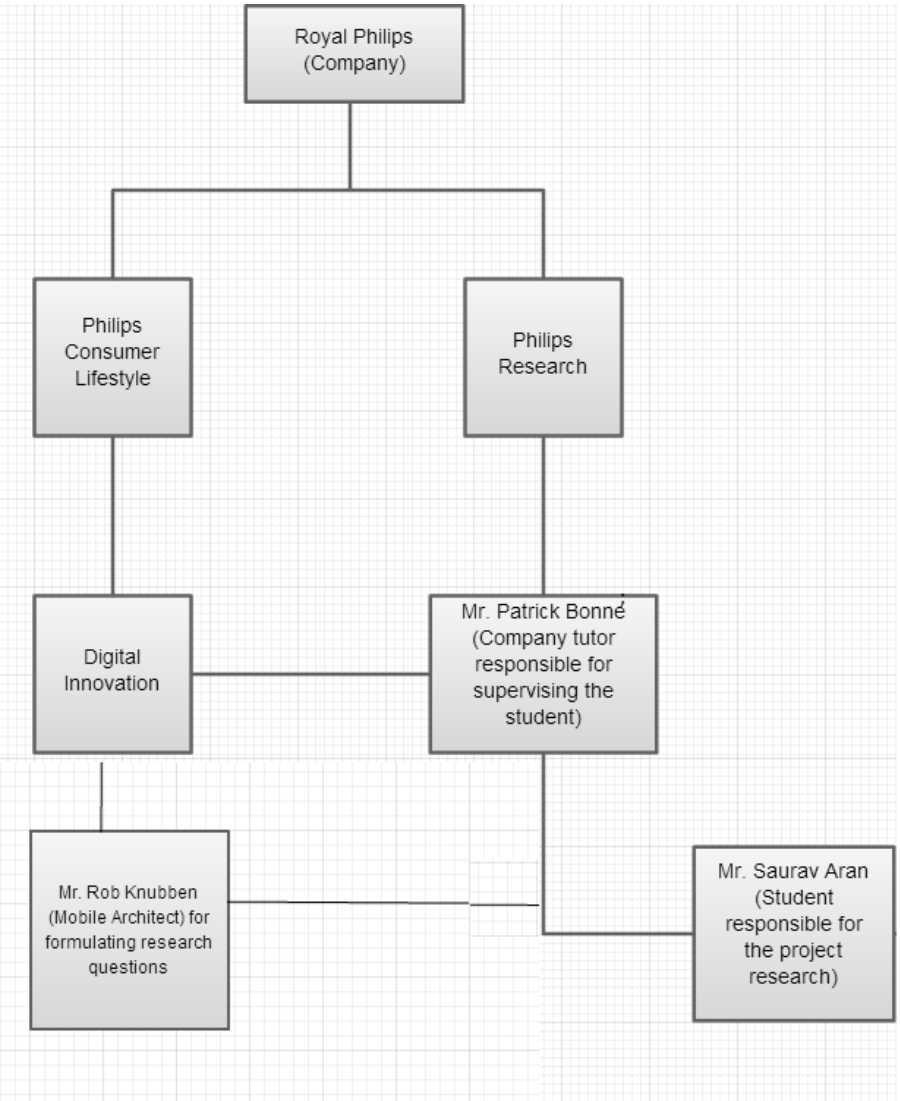


Figure B.4.6: Organogram of the company

Figure B.4.6 shows the organogram of the company where it indicates the position of the intern student at the company. It only shows the divisions of the company related to this project.

B.5. Version History

Version No.	Date	Comments
Ver. 1.0	Feb 5-6	Remarks/comments by Mr. Patrick Bonné
Ver. 1.1	Feb 7	Further remarks by Mr. Patrick Bonné
Ver. 1.1	Feb 19	Comments and suggestions by Mr. Marco Dorenbos
Ver. 1.2	Feb 20-25	Renamed Project Plan to Project Initiation document (PP) and updated with respect to comments by Mr. Marco Dorenbos.
Ver. 2.0	March 8	Research question 1 information added on Project product, project phasing and Time
Ver. 2.1	April 24	Updated with Research question 2 information on Project product, project phasing and time

C. Time proposal for Research Questions

C.1. Introduction

This document gives information on four self-created methods that can be used for setting time duration to multiple tasks to be executed in a project. The time proposal for each research questions on the basis of the chosen method can be found thereafter. The activity tracker with comments (if it was finished successfully or not) during the execution of each research question can also be seen.
The first research question was received in week 5 of the project and the second research question was received on week 12 of the project.

C.2. Time Proposal calculation methods (self-created methods)

Time distribution for a task in a project can be done in different ways. Some of them are Average time based, Weight based. Activities time distribution based and Average Hybrid method.

Task: Research questions to be performed

Activities: The research learning and development to be carried out in each task.

C.2.1. Average time based

The time distribution in this method is done by calculating the average of project time and number of tasks (work) to perform. The activity time is equal for all the tasks. This method is suitable for achieving at least some superficial results on all the activities to be carried out in a project. The Total time for each activity can be calculated as:
TaskTime(Tt)=Total time(T)/Number of tasks(N). (Result rounded to its lower whole number)
(If the numbers of task to be done are unfixed to some range, an average of total tasks can be taken.)

- Pros:

a) All the activities in a project will get equal priority and hence the final result will at least contain the output from all the tasks that needs to be performed in a project.

- Cons:

a) If a task consists of tough and harder activities, this method restricts from achieving a good result causing an unsatisfactory or unfinished output for that particular activity.
b) The level of details on a result might be unsatisfactory.

C.2.2. Weight based

This method is based on the weight (alpha) of the activities to be carried out in a project. Here, weight refers to the deepness and seriousness of the result to receive that ranges from 0-1. The weight to all tasks needs to be distributed in such a way that the sum of all the weights of total tasks results 1. This method is suitable for achieving deeper results on particular important activities than other tasks in a project. The Total time for each activity can be calculated as:

Tiwt= weight of Task (alpha) Total time (T)*
Where, TiWt is the Time for the Task on the basis of weight.
(In case, numbers of tasks are unfixed to some range, an average of total tasks can be taken.)

- Pros:

a)The result obtained on the weighted task will be of greater detail than other tasks.

- Cons:

a)The tasks with lower weight are performed less and can influence to an overall project result.

b) The weight distribution either done by the project members either by series of project group talks and discussions or a simpler prediction might not be that effective and correct. Due to this, some tasks that might carry high weight might be underweighted and hence the result on that underweighted task can be unsatisfactory.

C.2.3. Activities time distribution based

This method lists all the specific activities that need to be carried out in all the tasks and an estimated time duration is specified to each activity. The total time for a task is the sum of time for all the activities in that task. This method provides enough time for all the activities to be performed and especially suitable to define a project time.

Total number of tasks in a project= n; Total number of activities in a task=m
Task1Time= Task1Activity1Time+ Task1Activity2Time+……+Task1Activity (m-1) Time+ Task1ActivitymTime
Task2Time= Task2Activity1Time+ Task2Activity2Time+……+Task2Activity (m-1) Time+ Task2ActivitymTime
……

Task(n-1)Time= Task(n-1)Activity1Time+ Task(n-1)Activity2Time+……+Task(n-1)Activity(m-1)Time+ Task(n-1) ActivitymTime
TasknTime= TasknActivity1Time+ TasknActivity2Time+……+TaskmActivity (m-1) Time+ TasknActivitymTime

- Pros:

a) All the activities in a task are well defined and provide sufficient time to all the activities in the tasks to be carried out.

- Cons:

a) The sum of task duration can be longer than expected and can cross the project duration estimated.

b) The activities list in a task might not be complete during this planning and hence the time distribution can go wrong in the middle of the project. These unknown activities that arise make a pitfall in this method.

c) If the project duration is fixed, the tasks to be carried out at the later phase of project might not be performed due to the reserved time by the initial tasks.

C.2.4. Average Hybrid method

In this method, an average duration for a Task obtained from all the three methods (Average time based, Weight based & Activities time distribution based) rounded to its lower whole number. This method is suitable for getting the time duration for a task when all the tasks needed to be performed are not specific and not properly defined activities in a project.

- Pros:

a) Since, it uses the average time duration of all the method the chance of failure is minimum.

b) Gives a relief sensation for planning time duration in case of unspecific future tasks to be performed in a project.

c) The final output contains average requirements specified in all tasks.

- Cons:

a) An unhealthy way of determining the time duration for a task in a project that can lead to unknown problem of total time distribution. The duration for the final task in a project can have a chance of getting a time duration which is not available in the project and hence need to be compromised.

C.3. Time proposal for the 1st research question

First of all, the time duration using each method will be calculated. Finally, a decision on which method to choose will be decided and the time proposal for carrying out the 1st research question.

C.3.1. Calculation

a) Average time based:

- Number of weeks left (Total Time): 20
- Number of research questions: 3-4
- Average number of research questions: (3+4)/2=3.5
- Number of weeks per research question =20/3.5= 5.7 weeks

Hence, using the Average time based calculation method each research question will get a total time of 5 weeks. Therefore, research question 1 will get duration of **5 weeks**.

b) Weight based:

- Total time (Number of weeks left): 20
- Total number of Tasks (Number of research questions): 3-4
- Average Tasks (Average number of research questions): (3+4)/2=3.5

Since, all the research questions are not yet known and hence cannot be assigned a weight to all the tasks to sum up 1, a prediction of the weight of Task1 (research question) should be given which is "0.35". Therefore, *alpha1= 0.35*. The prediction consists of the following reasons:

- The start activity can be the foundation activity in a project and hence, will gain a weight more than the average weight (average weight =1/ Average tasks=1/3.5=0.28)
- There consists of many new learning and development like developing a cross platform application on Xamarin Integrated Development Environment (IDE).
- The ICP Client library seems to be more challenging for the cross platform development.

Therefore, the Time for Task 1(Research question 1) = alpha 1*Total time=0.35*20= 7 weeks.

Hence, using the Weight based calculation method, research question 1 will be performed for **7 weeks.**

Using this technique in this project, we have a total weight of 0.65 (1-Task1 weight=1- 0.35) remaining for other 2-3 continuing research questions.

c) Activities time distribution based:

Since, the project time is already defined and we have 20 weeks of time left, this method cannot be applicable. Also, all the tasks (research questions) and activities to be carried out are unknown which makes this method unsuitable for this project. However, we can try giving some duration to the possible activities for Task1 (research question 1).

Possible activities with expected time duration:

I. Learning about Xamarin and using it: 1 week
II. Developing simple Android and iOS application in Xamarin: 2 weeks
III. Making a cross platform application in Xamarin: 2 weeks
IV. Learning to make use of a library in Xamarin: 1 week
V. Using the ICP client library in the project: 2.5 weeks

Therefore, total time for task 1=Sum of Activity Time (I+II+III+IV+V) =1+2+2+1+2.5= 8.5 weeks

Hence, using the Activities time distribution based calculation method; research question 1 will get total time of **8 weeks**.

d) Average Hybrid method:

- Average time based duration= 5 weeks
- Weight based duration= 7 weeks
- Activities time distribution based duration= 8 weeks
- Time for Task 1 (1st research question) = (5+7+8)/3=20/3=6.6 weeks

Hence, using the Average Hybrid method calculation method, research question 1 will get total time duration of **6 weeks**.

C.3.2. Proposal

Since, the future research questions are unspecific, looking at the advantages and the week calculation resulting in 6 weeks that seem to be suitable in term of total project duration of 20 weeks, I propose to use the Average Hybrid method for the 1st research question.

Therefore, the research question 1 will be planned for 6 weeks starting from 11/March/2013 (Week 6 of the project) to 19/April/2013 (Week 11 of the project). The research recommendation will be provided on 22/April/2013 (Week 12 of the project).

(In case, the 1st research question is completed before the end planning date (19th April), we shall start with the 2nd Research question.)

C.3.3. List of Activities and tracking them

At the end of each week, a check is performed on the activity carried out during the week and a go/no-go decision will be taken for the next week activity.

Weeks	Activities	Status & comments at the end
Week 6	Learning Xamarin studio to develop a simple Native iOS (HelloXamiOS) and Android (HelloXamDroid) apps in C# for the first 1 week that consists of simple UI buttons and shows some text messages on the display.	Done. Gained basic knowledge on Android and iOS app development on Xamarin that uses C#. Next activity can be stepped as planned.
Week 7	Learning and gathering information to be able to develop a simple cross platform application on Xamarin Studio. Simple cross platform app refers in making use of shared codes between the different platform projects.	Done. The techniques for developing a simple cross platform application is acquired and hence next activity can be adapted
Week 8	Develop a cross platform XamWULCal app that can trigger a method (that shows an alarm message on the display) at user chosen date and time.	Done. A simple cross platform application is created that fulfills the activity for Week8. Hence, Week 9 can be stepped further.
Week 9	Investigation on the use of native iOS and Android libraries on Xamarin.	Done. The methods and techniques for the usage of native iOS and Android libraries on Xamarin were known. However, to successfully complete the week 10 activity, additional support is needed to perform the activity due to no previous experience in usage of library written in different language than the platform to be used in.
Week 10	Importing the ICP Client library on the XamWULCal built in week 8.	Done. The C# version of ICP Client library (C# wrapper around C) was provided and hence simply added the ICP Client library to the project. Next, activity can be stepped and the C# wrapper class can be used in Xamarin to access the function of ICP Client library.
Week 11	Finish the XamWULCal app that enables to send commands to the WUL via ICP Client at user specified date and time.	Not possible for this moment. The ICP Client (compiled for Android and iOS) was imported to Xamarin.Android and Xamarin.iOS project in order to send commands to WUL. But, due to the uneven version of libICPClient where the NVM doesn't implement all the functions referenced by libICPClient, it wasn't possible to send commands to WUL for this moment. Hence, the task was carried out as planned but the result didn't appear as expected, to be able to send commands to WUL via ICPClient.

C.4. Time proposal for the 2nd research question

Since, Average Hybrid method was the chosen method for the time management in the first research question, the same method will be used in determining the time required for the second research question.

Number of weeks left (Total Time): 13
Number of research questions: 2-3
Average number of research questions: (2-3)/2=2.5

To use the Average Hybrid method, first we need to calculate the time span obtained from other three methods (Average time based, Weight based and Activities time distribution based).

C.4.1. Calculation

a) Average time based:

Number of weeks per research question =13/2.5= 5.2 weeks

Using the Average time based calculation method each remaining research question will get a total time of 5 weeks**.** Therefore, research question 2 will get duration of **5 weeks**.

b) Weight based:

Firstly, we need to give a weight value to alpha. The sum of total value of alpha in the project is 1 and the first research question already had the weight of 0.35. Hence, we only have 0.65 (1-0.35) of the weight left for the total project (remaining 2.5 research questions). Providing an alpha value of 0.31 (alpha2=0.31) to this research question which is a little bit more than the average weight (0.28),

Therefore, the Time for Task 2(Research question 2) = alpha 2*Total time=0.31*13= 4.03 weeks.

Using the Weight based calculation method; research question 2 will be performed for **4 weeks**.

We have a total remaining weight of 0.34 (1- 1st Research question weight – 2nd Research question weight=1- 0.35-0.31) for other 1-2 continuing research questions.

c) Activities time distribution based:

Possible activities with expected time duration:

I. Learning about application design patterns: 2 weeks
II. Learning Xamarin.mobile: 1 week
III. Choosing the best pattern suitable for Xamarin app development and designing a cross platform app with respect to chosen pattern: 1 week
IV. Implementing Xamarin.mobile to fit the application: 1 week
V. Finalizing the cross platform model with maximum usage of code sharing in Xamarin: 1 week

Therefore, total time for task 2 =Sum of Activity Time (I+II+III+IV+V) =2+1+1+1+1= 6 weeks

Using the Activities time distribution based calculation method, research question 2 will get a total time of **6 weeks**.

d) Average Hybrid method

Average time based duration= 5 weeks
Weight based duration= 4 weeks
Activities time distribution based duration= 6 weeks

Time for Task 2 (2nd research question) using Average Hybrid method = (5+4+6)/3=15/3= 5 weeks

Hence, using the Average Hybrid method calculation method, research question 2 will get total time duration of **5 weeks**.

C.4.2. Proposal

Therefore, the research question 2 will be planned for 5 weeks using the same Average Hybrid method chosen in the 1st research question. The time range is starting from 29/April/2013 (Week 13 of the project) to 31/May/2013 (Week 17 of the project). The research recommendation will be provided on 3/June/2013 (Week 18 of the project).

C.4.3. List of Activities and tracking them

At the end of each week, a check is performed on the activity carried out during the week and a go/no-go decision will be taken for the next week activity.

Weeks	**Activities**	**Status & comments at the end**
Week 13	Learning about Windows Presentation Foundation (WPF) patterns/model.	Done. Gathered information on WPF and shall proceed to next step.
Week 14	Learning different types of WPF models and their uses in creating different types of applications. Viewing the models adapted on existing Xamarin example applications.	Done. Learnt about MVC, MVP and MVVM. Frameworks like MonoCross (uses MVC) and MVVMCross (uses MVVM) are popular for the cross platform application development in Xamarin. Next, step can be taken.
Week 15	Learning about Xamarin.mobile	Done. It was learnt that Xamarin.mobile provides single set of API access. But, this library cannot be used directly while adapting the MVVMcross framework. Hence, instead the available plugin in the Nuget can be used that provides many more API accesses besides GPS and accelerometer. Next step can be taken.
Week 16	Choosing the most effective model for Xamarin Cross platform application development that makes high reuse/sharing of codes and designing a prototype cross platform app (with respect to the chosen model).	Done. MVVMCross framework chosen. The design of the app is designed as the same app created before. i.e. WULCalendar app but using the MVVMCross framework. Since, the WULCalendar app doesn't make use of any APIs present in Xamarin.mobile, the next step can be replaced with the implementation of the application.
Week 17	*Implementing Xamarin.mobile to fit the design in Week 16.*	*Changed. Xamarin.mobile doesn't fit the design and also better option (MVVMCross plugin) was found.*
Week17	implementation of MVVMCross framework	Not completed. Due to the unexpected setting up (tweaks) required for the framework deployment and fragmentation in IDE for the development support, the task for implementing the framework couldn't be completed successfully.

D. Research Learning and recommendations

CONTENTS

1 Introduction

This research is carried out during the graduation internship that started from February/1/2013 and ended on July/31/2013. The research questions are supposed to be defined starting from week 5 and end on week 18 of the project Phasing as stated on the Project Plan.

During the whole internship period, a total of 3-4 research questions are estimated to be defined and investigated. This document consists of the findings, learning and recommendation on the basis of these research questions.

2 Research question 1

The first research question was received on week 5 of the project phase. The research question 1 stated "How is it like to develop a cross platform application for the WakeUpLight (WUL) using the "Xamarin Studio" Integrated Development Environment (IDE) that is able to control WUL's functionalities using pre-existing ICP Client library?" The output for this research question should contain brief useful information on cross platform application development using Xamarin Studio. The research will be dealing with only two platforms: Android and iOS

I) Planning

The start of the research question was preceded by a time duration proposal (refer "Time Management for Research Questions.doc") of 6 weeks for this 1st research question. After the proposal was made and approved, a planning for possible activities during the research question was done. Finally, these listed tasks were performed during the research period of the 1st research question and the research information acquired is noted.

The research consisted of learning a totally new development environment for mobile application. The activities were to be planned, in order to make sure the result obtained at the end of 1st research question is on specified time and to avoid getting lost during the research journey. The planning for the lists of possible activities to be carried out is as follows:

- Week 6: Learning Xamarin studio to develop a simple Native iOS (HelloXamiOS) and Android (HelloXamDroid) apps in C# for the first 1 week that consists of simple UI buttons and shows some text messages on the display.
- Week 7: Learning and gathering information to be able to develop a simple cross platform application on Xamarin Studio. Simple cross platform app refers in making use of shared codes between the different platform projects.
- Week 8: Develop a cross platform XamWULCal app that can trigger a method (that shows an alarm message on the display) at user chosen date and time.
- Week 9: Investigation on the use of native iOS and Android libraries on Xamarin.
- Week 10: Importing the ICP Client library on the XamWULCal that is built in week 8.
- Week 11: Finish the XamWULCal app that enables users to send commands to the WUL via ICP Client at user specified date and time.

II) Learning, Development & investigations on Xamarin

In general, Objective C using X-Code Integrated Development Environment (IDE) and Java using Eclipse IDE is used to develop a native iOS and Android app respectively. To make a native app on different platforms, it is required to recode the whole app totally per platform, which is time consuming as well as inefficient in terms of cost. There exist some other frameworks (e.g. PhoneGap), that helps in preventing these issues. However, there arise different problems related to accessing the device's native Application Programming Interfaces (APIs) and also the apps built using these frameworks lack native user experience.

Philips targets to provide a native experience (look and feel) to its customers and hence, taking such things into account and to save time and cost for the application development in different environment, Xamarin Studio [11] seems to be a good step for the research. The learning and investigation topics below flow in systematic as in the planning above.

This chapter consists of learning about Xamarin Studio, Android & iOS native application development and finally a cross platform application.

A) Xamarin studio

Xamarin is a commercial IDE that is used to build native/cross platform mobile applications across Android, iOS and Windows platforms with C#. Only the iOS and Android platforms will be discussed. This IDE is supported on both Windows and Mac OS. It consists of Xamarin.Android (formerly Mono for Android) and Xamarin.iOS (formerly MonoTouch) for developing Android and iOS applications respectively, built on top of Mono (an open-source version of the .NET Framework based on the published .NET ECMA standard), including memory management, reflection, and the .NET base class libraries, as seen in Figure II.1.

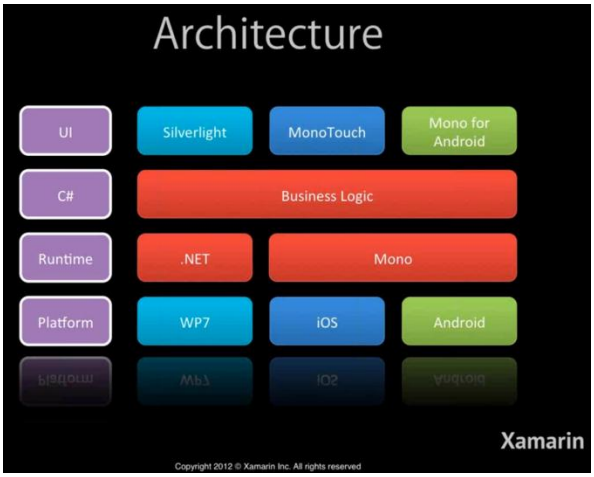


Figure II.1: Xamarin Architecture

Xamarin's Ahead of Time (AOT) compiler compiles Xamarin.iOS apps to produce native Arm binary (.app) suitable for Apple's app store and Xamarin's compiler compiles Xamarin.Android apps down to Intermediate language and finally produces native binary output (.apk) for Android by taking advantage of Just in Time (JIT) compilation right on the Android device. As Xamarin compiles the app to a native binary, not cross-compiled, and not interpreted, it gives users brilliant app performance for even the most demanding scenarios like high frame rate gaming and complex data visualizations. Also, Xamarin exposes all the Application Programming Interfaces (APIs) available in iOS and

Android to the developer as C# class libraries. The apps built are still native because, they don't abstract away the native platform UI APIs instead provide a bindings to the native APIs and provide access using C# library. [14] Hence, Xamarin being able to provide development of native featured (User Interface & performance) application and provide the ability to develop the cross platform application, it seems to be a good and very efficient IDE to choose regardless of its limitations on Xamarin.Android [15] (most of which are removed from the 4.1 release) and Xamarin.iOS [16].

Note: The Xamarin used in this research is Version 4.0.2 (build 18)

The application development using Xamarin studio for both iOS and Android is only possible on Mac OS. Whereas, Xamarin studio on Windows doesn't support iOS development. However, to develop both Android and iOS applications, Visual studio can be used by installing the Xamarin.iOS and Xamarin.Android plugins for the Visualstudio. Using the visual studio for the iOS app deployment, a network Mac OS is required.

B) Helloworld native mobile Application development

For the development of iOS and Android applications on Xamarin, the pre-knowledge of iOS and Android native development on X-Code and Eclipse respectively helps to understand the common platform specific terms easily. Android development, iOS development and then a short conclusion on the native mobile application development using Xamarin will be discussed hereafter.

1. Android Development

The Android development that will be discussed here is a very basic step that contains just adding of some widget class (Button) on the application layout and making it run. This is actually to see the native look of the Android app and to gain an idea to implement some actions to these buttons

a) To start with the Android development on Xamarin, creating a new C#-> Android Application project sets up a default application to the user with simple application that counts and displays the number of click on the button present.

b) To create and add permissions to the Android Manifest, right clicking the project ->Options ->Android Application, and adding the necessary permissions. This will allow the application to use the services and access protected parts of device APIs like Camera, GPS, Bluetooth etc. or even restricting to the minimum Android API level to use the application and many more permissions.

c) To create a layout for the application and add other UI items (i.e. android.widget) can be done in two ways:

- One way is simply dragging and dropping the UI items available on the ToolBox on the Resources->layout->Main.amxl. The implementation for the events and actions carried out on this user interface (Main.amxl layout) by the user can be done on MainActivity.cs. This MainActivity.cs consists of Activity 1 class (default) which is a derived class of Activity.

  For example: If a button is added on the layout (Main.axml) with an ID myButton, a reference to this button, and a click event to button to show a text on the top of itself can be done by:

  Button btnRef=FindViewById<Button>(Resource.Id.myButton);//reference to the button

button.Click += delegate {btnRef.Text = "Message Displayed on the button";} //triggering action when a click is done on the button

- The other way is to create it dynamically i.e. creating a new instance of Layout class and adding items to it as we normally do in C#.

  For Example: To create a new layout with vertical orientation, adding a button on it and displaying the layout can be done by:

  var layout = new LinearLayout (this); // new instance of layout
  layout.Orientation = Orientation.Vertical;//setting up the property
  var btnPress = new Button (this);//new instance of Button
  layout.AddView (btnPress);///adding the new button to the layout
  SetContentView (layout);//displaying the layout as the content view

Figure II.2: Native Android HelloWorld app built using Xamarin

So, we can say that Xamarin.Android provides all the native field, properties and methods for this instance of Button to the user in the form of C# class library. This button is a totally native Android Button, as seen in Figure II..

d) Similarly, it is possible to access all other Android APIs to build a native Android application using Xamarin in C#.

2. iOS Development

The iOS development that will be discussed here is a very basic step that contains just adding of Button and label on the application layout and making it run. This is actually to see the native look of the iOS app and to gain an idea to implement some actions to these buttons.

a) To start with the Android development on Xamarin, creating a new C#->iPhone-> Single View Application project sets up an empty iOS application.

b) For maintaining the compatibility of the application, info.plist can be changed accordingly.

c) The layout of the application can be designed either (uses Xamarin.iOS UI Kit) using a source code editor (not tried) or by using the X-Code interface builder. To use the X-Code Interface Builder (IB) to design the layout, right click the class with .xib extension (storyboard)-> OpenWith-> XCode Interface Builder. This will load the X-code and lets us view/create layouts/widgets for the application. All the changes made in the layout from XCode IB (should have knowledge of X-CodeIB for creating Referencing outlets and Sent Events) are automatically updated with the new changes in the Xamarin iOS project. The .designer.cs (sub tree on the layout/ViewController class derived from

Figure II.3: Native HelloWorld iOS app built using Xamarin

UIViewController) consists of the Outlets added as:

[Outlet]
MonoTouch.UIKit.UIButton btnClickOutlet { get; set; } //a button
[Outlet]
MonoTouch.UIKit.UILabel lbl2 { get; set; }// a label

With actions defined to it as:

[Action ("btn1Pressed:")]
partial void btn1Pressed (MonoTouch.Foundation.NSObject sender);

which can be consumed by:

partial void btn1Pressed (NSObject sender)
{this.lbl2.Text=" btn 1 clicked.";}// displaying message when btn1 clicked

However, the actions to the outlets can also be made in Xamarin as mentioned earlier. The Main.cs loads this ViewController when the application starts.

In Figure II.3, we can see that the application layout containing buttons and labels have native look and feel (in fact, they are native) and were accessed as a C# class library.

d) Similarly, it is possible to access all other iOS APIs to build a native iOS application using Xamarin in C#.

3. Conclusion

a) A totally native application (full platform SDK access) for both Android and iOS can be built using Xamarin's Xamarin.Android and Xamarin.iOS respectively.
b) All the native APIs present can be totally used and accessed through Xamarin's libraries in C# that are mapped to all those native APIs.
c) It's not only the native look on the application built on Xamarin, but in fact, the UI items are all native components from each platform.

C) Cross-platform mobile application development

Earlier, native iOS and Android applications were built using the Xamarin. Now, it's time to learn and then develop a cross platform application so as to find out how we can be efficient in terms of time, cost and user experienced using the cross platform mobile app development feature of Xamarin. Learning, development and then a conclusion on cross platform mobile application using Xamarin IDE will be discussed hereafter.

1. Learning

At first, the word "Cross platform mobile application" sounds like a single universal application (like a web application) that can be deployed and run on different mobile platforms. Personally, before exploring to the Xamarin world, I assumed the same and initially thought of developing a single universal application project on Xamarin to build an application that runs on all mobile platforms. But this is not the real case here in Xamarin. Instead, cross platform mobile app actually referred in the sharing of codes between different platforms and still, we need to build multiple application projects on Xamarin according to the number of platforms we target i.e. There should be at least two different application projects if we want to build a cross platform application for Android and iOS; and also should either contain a different shared library project or a file shared between platforms. Hence, cross platform development means, writing codes once and using it in each of this different platform's application project without rewriting the code.

There can be an interrogation like "If we can share codes between all the platforms, why not to build a single application by sharing all the codes and finally compile it to run on all platforms instead of creating different applications for each platform?" Yes, it is true that we can share codes between the platforms, but the truth is that, not all the codes can be shared. There are several platform specific features that don't exist/support/differ between platforms that lead in preventing total sharing of codes between the platforms. For example: the way of notifying users by an application on Android is different in Android, iOS and Windows.

Xamarin cross platform application uses a simple architecture as seen in Figure II.

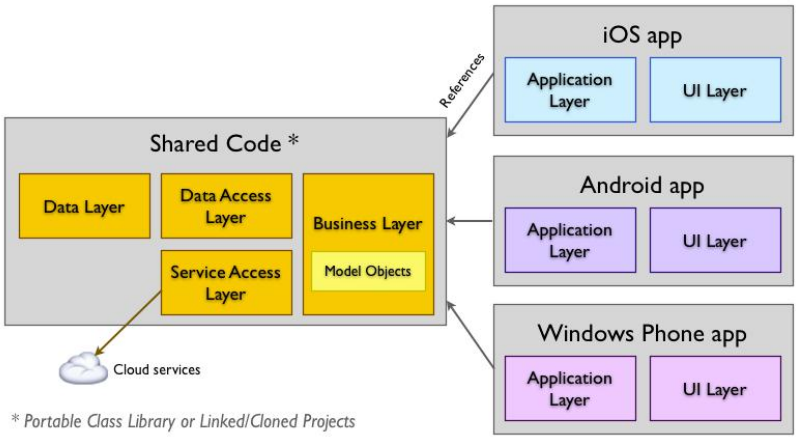


Figure II.4: Xamarin Cross platform architecture

Figure II shows that the User Interface (UI) layer that consists of all the user facing layer like screens, widgets etc. in each target platforms and the App layer that consists of code that is application specific aren't shared. These two layers are implemented separately with respect to the number of target platforms. The shared layers that can be reused across the platforms are Business Layer (BL), Service Access Layer (SAL), Data Access layer (DAL) and the Data Layer (DL). The presence of shared layers depend on the type of application to be built for example: if an application doesn't make use of network resources, the Service Access Layer may not exist.

The sharing of the codes between the platforms can be done by the different ways. One method is File Linking. In this method, the code (file/class/) that is to be shared lies in same/different directory and only a reference of the file meant to be shared is provided to each platform application project. Finally, it is compiled together with the platform specific project (including unshared layers) to create platform specific application. Any changes in one of the code will synchronize to change in every project. Other method for sharing codes is creating a platform specific library project or creating Portable Class libraries (PCL) which can be consumed across different platforms. Instead of providing a reference, the files to be shared can be cloned across different projects too. Microsoft Project Linker can also be used which keeps multiple project files synchronized as source files are added to each. These are all the methods that can be used in sharing codes while developing a cross platform mobile application using Xamarin.

For developing a cross platform app, following these mentioned steps will help to achieve some relief in the development:

a) Making a prototype for both platforms that consist of application characteristics that are universal like feature selection via tabs or menus, lists of data and scrolling, navigating back etc. This helps in maintaining the user experience to a greater extent across different platforms.
b) Determining device-specific features like Camera, GPS, Accelerometer etc. to take advantage of each platform. This can alter user experience to some extent.
c) Defining the shared code, building the UI for each platform project and finally start with application implementation.

2. *Cross-platform development*

To start with the development of a cross platform app, simple prototypes for both the platforms was designed for both the iOS and Android UI layer, as seen in Figure II.. Android consists of two buttons, one for selecting Date and the other for Time, as it doesn't support like a DateTime Picker in iOS. However, on iOS, two different buttons can be used to let the user select Date and Timer separately like on Android. A Spinner is used to show the list of WUL functions on Android and a PickerView is used on iOS.

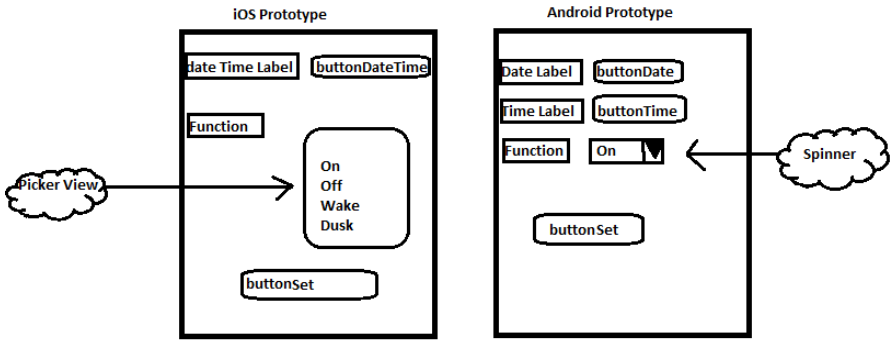


Figure II.5: Prototype for the Cross platform app

This application doesn't use any of the device's hardware (Camera, Bluetooth etc.) but instead uses a .NET framework's base class library (System.Timers.Timer) that will be coded once and reused across multiple platforms. Following the cross platform architecture as in Figure II, a Xamarin Solution (XamCalWUL) for this development will contain three project, one for implementing the iOS UI layer (XamiOSCalenWUL), second for implementing the Android UI layer (XamDroidCalWUL) and the last one for shared project (XamCore). The XamCore contains a class inherited from the Timer coded once and used across different platforms, as seen in Figure II.5. To display debug message across Android and iOS, conditional compilation is too done that allows running a main UI Thread to display the message on the target platform. For example, a conditional compilation that shows a log message "Function Triggered from Android" while running on Android:

```
#if __ANDROID__
Console.WriteLine("Function Triggered from Android");

#else {Console.WriteLine("Function Triggered from other devices");}
```

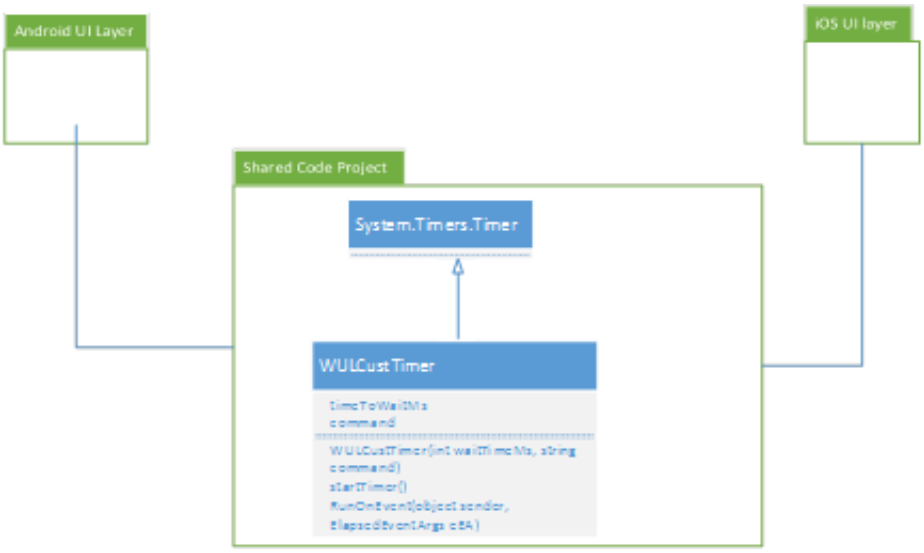


Figure II.6: XamCalWUL architecture

After designing and implementing the UI layer for both the platforms, the shared code was used to trigger the WUL function in user specified Date and Time by simply cloning the file "WULCustTimer.cs" Class across iOS and Android projects (Due to the less amount of shared code present, the Core project was removed and the class was cloned in each project instead)

The cross application built contains little amount of shared code (WULCustTimer.cs) due to its simplicity in functionality to only trigger a Timer. The application built can be seen in Figure II.7.

To access the device hardware APIs like Camera, Bluetooth etc., Xamarin provides a binding to the Native APIs. However, if we want to perform an increase in shared code across the platforms and make application development even faster, Xamarin.mobile is the perfect library to be used. Xamarin.mobile exposes a single set of APIs for accessing common mobile device functionality across iOS, Android, and Windows platforms. This library abstracts the contacts, camera, and geo-location APIs across all these platforms. Future plans include notifications and accelerometer.

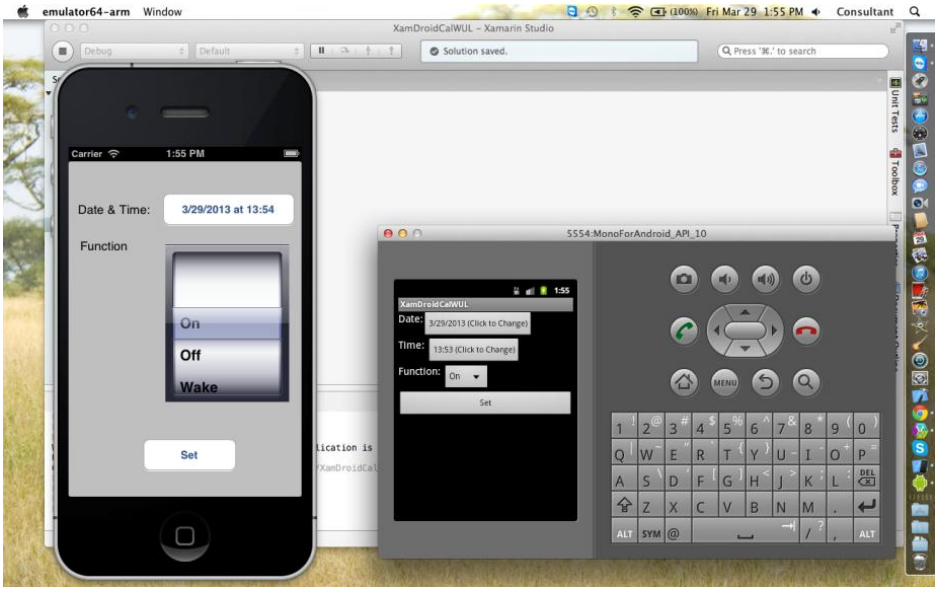


Figure II.7: Cross platform applications (XamiOSCalenWUL & XamdroidCalWUL) built in Xamarin

3. Conclusion

a) Cross-platform development consists of multiple projects with different platform specific UI layer and App layer implemented. The shared part can be the BL, SAL, DAL and DL.
b) The sharing of codes can be done by the method of File Linking into each/separate app project or by cloning the files to be shared or using Microsoft Project Linker available only for Windows.
c) Making prototypes, determining device-specific features and defining shared code ahead will help to relief in building a cross platform application that provides a similar user experience on all platforms.
d) Xamarin.mobile exposes a single set of APIs for accessing common mobile device functionality across different platforms increasing the amount of shared code.

D) Using existing native C/C++, Java and Objective-C libraries

The cross platform application built so far has the ability to trigger a message display the function chosen by the user at user selected date and time. Now the task of using an ICP Client to send remote commands (ON, OFFF etc.) to the WUL device, instead of displaying the message, is yet to be accomplished.

1. Learning

At the present context, we have native libraries available written in C, Objective-C (C containing a wrapper for iOS) and Java (C containing wrapper for Android) that can be used in iOS and Android platforms respectively. But to make use of any libraries; it is only possible to access C# libraries in Xamarin. It is not practically possible to recreate all the existing native libraries to be used in C#. Hence, there should be some tweaks so as to simplify the usage of existing native, Java and Objective C libraries.

To be taken into account, Xamarin.iOS makes use of only static libraries due to the restriction in the Apple store as iOS limits the Just In Time (JIT) compilation and only supports Ahead Of Time (AOT) compilation. Whereas no such restriction is yet seen on Android store (Google Play).

i. We can use the binding feature in Xamarin to access the Objective-C library as a C# library. [17]

ii. To reuse the Java libraries as C# library, it can be done in three ways: [18]

1) Create a Java Bindings Library – With this technique, a Xamarin.Android project is used to create C# wrappers around the Java types. A Xamarin.Android application can then reference the C# wrappers created by this project, and then use the .jar file.
2) Java Native Interface – The Java Native Interface (JNI) is a framework that allows non-Java code (such as C++ or C#) to call or be called by Java code running inside a JVM.
3) Port the Code – This method involves taking the Java source code, and then converting it to C#. This can be done manually, or by using an automated tool such as Sharpen.

By the above methods, it is possible to use the libraries in Xamarin written in Objective C and Java languages for developing Xamarin.iOS and Xamarin.Android application respectively. However, on the basis of information collected from a forum post [19], it was found that binding a native library and making it to be used as C# class library will only allow the native platform application of that library on Xamarin. For example: If we bind the Android's ICP Client library to be used as a C# class library, it can only be used in Xamarin.Android project application but not on Xamarin.iOS application and vice versa. This makes a double task to bind the other library too. Instead, if we bind a native C/C++ library (not compiled for specific platform), it will allow to be used in both the platforms. Therefore, the C/C++ ICP Client library (if available) can be used to bind and make accessible as a C# class library on Xamarin.

iii. For binding the native C/C++ library, we can follow some of the many methods as follows:

1) Creating a wrapper around the C/C++ library using different available tools like Simplified Wrapper and Interface Generator (SWIG) [20], CXXI (pronounced as Sexy) [21] can be used. SWIG seems to be more efficient than CXXI due to the reason CXXI is reflection based and not suitable for Ahead of Time Compilation (AOT) features needed in Xamarin.iOS application. [22]
2) Using Platform Invocation Services (P/Invoke), C# DllImport (using System.Runtime.InteropServices) enable interop with DLL files and lets us use a C/C++ DLL dynamic link library, or custom legacy DLL- even one we can't rewrite but have the ability to modify. This can also be a good alternative for using the C/C++ ICP Client library in the Xamarin project.

In order to use the ICP client library on the Xamarin project, different methods and techniques as explained above can be used. However, the exact information on the type (Language) of ICP client library available is so far still unclear to me, whether the ICP Client library available is only native Android/iOS or C++ or C.

To conclude the learning on using different types of existing libraries on Xamarin, Java, Objective C and native C/C++ libraries can be used on Xamarin and accessed as a C# class library by applying different techniques.

2. *Importing and using a simple Test Native C shared library (.so) in Xamarin*

The existing ICP Client library available is of four types. One is a C library, second is the C library wrapped with a wrapper to be used in Android (Java), the third type is a C library wrapped with a wrapper to be used in iOS (Objective C) and the fourth is the C# ICP Client library compiled for Windows (C# wrapper around C).

On this information, instead of trying to import and use the complicated ICP Client library at the beginning, a simple Test C shared library (.so that is built using NDK-build) will be imported and consumed in the Xamarin.Android Project using the P/Invoke mechanism.

Let's assume we have a native shared library called MyTest.so (C header and source compiled using ndk-build on Eclipse) and we want to use it in the Xamarin.Android project. The MyTest.so consists of a single function

int MyTest_GetValue();

Now, we need to use this function on Xamarin.Android project. Here are the steps so as to succeed:

Step 1: Creating a new folder inside the Xamarin.Android project called lib and sub-folder armeabi. Copying the .so library to be used inside the armeabi folder. [23]

Step 2: Setting the properties of the library.so (imported library) **Build action** to "AndroidNativeLibrary" and **Copy to output** to "Always Copy".

Step 3: (Working in Xamarin.Android Class eg: MainActivity.cs)

- Including the namespace InteropServices by "*using System.Runtime.InteropServices;*"
- Using the standard DllImport in the project to import the native library as below: *[DllImport("LibTest.so")] public extern static int MyTest_GetValue();*// with exact Functtion Name, Type & Params in the .so Lib.

Step 4: Consuming the function above (MyTest_GetValue()) in the application.

For Example:

int value= MyTest_GetValue();

Console.Writeline(value.ToString());

As in the above mentioned ways, the functions in the library can be used by using the P/Invoke (Platform Invocation Service) that allows managed code (C#) to call unmanaged functions that are implemented in a DLL in Xamarin.Android project.

Hence, we can see that Native libraries can be successfully imported and used in Xamarin.Android projects. (Testing a native library on Xamarin.iOS project wasn't performed. However, directly implementing the native ICPClient library on Xamarin.iOS project will be performed)

3. *Implementing ICP Client library functions on Xamarin Project to send remote commands to the WUL*

There are different variations of ICP Client library that can be used in the Xamarin Project. The Java and Objective C variation of ICPClient library can be used in Xamarin.Android and Xamarin.iOS project respectively by creating a binding to these libraries in Xamarin. Since, these ICPClient libraries available are already a wrapper (Objective-C/Java) around C. Hence, creating another wrapper (C#) will be a wrapper around a wrapper. Hence, instead of using these libraries, we tend to use the native Core library (C) and also C# wrapper of the ICP Client is already available for the use in the project. The native C ICP Client library (libICPClient compiled for Android/iOS) will be imported to the Xamarin.Android and Xamarin.iOS project and it will be used for sending remote commands to the WUL through the C# wrapper available in Xamarin.

According to the architecture of ICP Client library (ICPClient-API.docx), IcpClientInit, IcpClientSignOn, Sign on callback and Data Upload are the necessary functions and callbacks to be used for sending a remote command to the WUL. However, in the wrapper code, the implementation for Data Upload (data collection portal) isn't seen. The first step would be to Initialise (Init), sign on (SignOn) and handling callback. After being able to consume important functionalities from the native ICP Client library, consumption of the data collection portal (UploadData) can be carried out further.

i. ICP Client on Xamarin.Android

To start in Xamarin.Android, the import of Android compiled ICP Client library (libICPClient) to Xamarin.Android project (XamDroidCalWUL) was done. The version of ICP Client used is an older version so as to eliminate problems related to Non-Volatile Memory (NVM) as lots of changes related to NVM are made in the new version of ICP Client. Other required libraries (libcrypto.so, libssl.so and libicpnvm.so) that are referenced from the ICPClient library are too imported to the project. Performing the same method as carried out in the Test (Research question 1.II.D.2) to make use of icpClientInit and icpClientSignOn functions, the function didn't seem to get access from the library with a null reference exception. Here, what we do is calling the C library functions and ignoring the Java Native Interface (JNI) functions. According to Mr. Aravind Gundumane, the null reference might be due to JNI interface for NVM where values can only be provided from Java application. The only way is to compile out JNI layer and expose C NVM interface to application. Hence, for a moment it wasn't possible to use the ICP Client library in Xamarin.Android.

ii. ICP Client on Xamarin.iOS

Moving further with using the ICP Client on Xamarin.iOS project (XamIOSCalenWUL), the available static (since iOS only supports static libraries) latest ICPClient library (libICPClient.a) compiled for iOS is imported. Also, other available libraries referenced by ibICPClient.a (libcrypto.a and libssl.a) are imported to the project. The library too references to the NVM functions, but only the C source and header file is available for NVM (older version). Hence, it should be first compiled in X-Code to create a static library for different architectures. For now, we can build the i386 architecture meant for simulators. Creating a new project (Cocoa Touch Static Library) on X-code and adding the NVM C source and header to the project, a static library for i386 architecture

(libICPNVM.a) was built. Now, this library can be used in Xamarin.iOS project which will be referenced by the libICPClient. Even after importing this library to the Xamarin.iOS project, there exists failure in referencing the functions like icpPAL_NVM_SetProductInfo, icpPAL_SetPropertyByteArray etc. According to Mr. Chris Tilmans, the reason behind this failure was due to the different versions of ICP Client library and the NVM (source and header files). All the functions that are referenced by the new version of libICPClient.a aren't implemented in the imported version of NVM, which is pretty much old. The design and implementation of ICPClient and NVM is changed at the current time than implemented previously. The older version of NVM is fully implemented in C and the older version of ICPClient library only referenced the limited functions present at that moment in NVM. But, the newer version of ICP Client consists of different other references to functions which the current version of NVM thereafter references to other library written in Objective-C. This causes a disruption in using the newer version of NVM either here in Xamarin.iOS project. Due to this reason, for the current moment, the ICP Client library cannot be used in Xamarin.iOS project for sending commands to the WUL.

4. *Conclusion*

1) Java, Objective-C and Native libraries (C/C++) can be used in Xamarin projects for the development of Android and iOS applications.
2) A test native shared library compiled for Android was used on Xamarin.Android project to consume its underlying functions to prove the usage of native libraries on Xamarin studio. This test supports to some extent that native ICP Client library can be too used in Xamarin. However, due to the complicated underlying implementations (of which I am unaware) in ICP Client library, it might not be completely possible.
3) Instead of creating a wrapper around a wrapper, it was chosen to use the core ICPClient library compiled for different platforms (iOS and Android).
4) Trying to access the functions on ICP Client library compiled for Android, resulted a null reference failure, indicating a direct non-access to the C functions present in the library.
5) Due, to the difference in the version of ICP Client library and the NVM, at the current moment and change in the architecture/implementation of the required libraries over time, it is not possible to use the ICP Client library to send commands to the WUL at the current moment using the native library.

III) Conclusion(S) & Recommendation(S) on Research question 1

Xamarin allows development of native/cross platform mobile applications across Android, iOS and Windows platforms with C#, including memory management, reflection, and the .NET base class libraries. It gives users brilliant app performance for even the most demanding scenarios like high frame rate gaming and complex data visualizations, since the apps built are native. The apps built have native look and feel and can access all the APIs that are exposed by the devices. The ability to provide sharing of codes even between the different platforms is the most fascinating feature of Xamarin. To facilitate increase in sharing of codes, it also provides different libraries, for example: Xamarin.mobile that exposes a single set of APIs for accessing common mobile device functionality across different platforms. There exist different limitations on Xamarin which weren't faced/ recognized during the research question 1.

Different existing libraries (Java/Objective-C/Native) are supported to be used on Xamarin for the development of mobile applications which was also proved by conducting a use of a simple test library (for Xamarin.Android project). However, regarding the use of ICP Client library on Xamarin to send commands to WUL due to the confusions in different versions of ICP client and other dependent libraries available, the change in the architecture/implementation of the libraries over time and lack of complete information of the library, it wasn't possible to use the native library on Xamarin at the current moment. However, different steps can be followed next so as to try to make ICP Client library workable on Xamarin:

a) Making a decision on which version of Core ICP Client and other referencing libraries to use and how to approach it. On the basis of the decision from discussions, trying to make use of the Core ICP Client library (native C) compiled for iOS and Android in Xamarin project. OR
b) Using the existing Java/Objective-C ICP Client library to make it able to be used in Xamarin project. Though these libraries already being a wrapper, a C# bindings can be done on top of it to be used in Xamarin. This will include some automatic bindings by Xamarin and manual works.

To conclude, Xamarin seems to be a good IDE for the development of both native and cross platform mobile applications, taking care of Philips requirements of providing a faster application with native look and feel to deliver a great user experience to the users. Also, time saving, cost saving and ability to reuse existing libraries for the cross platform mobile application development are really the worthy values to gain after embracing this IDE.

Possible future works:

a) Using a Soap protocol to send commands to WUL instead of ICP Client.
b) Trying to make the ICP Client workable on Xamarin to send remote commands to the WUL.
c) Making the use of Xamarin.mobile to provide a maximum share of codes for the access of device APIs like Camera, Bluetooth etc.
d) Collecting information on open source framework called MonoCross for the crossplatform mobile application development.

3 Research question 2

The second research question was received on week 13 of the project phase. The research question 2 stated "A recommendation on the best model/approach/patterns for the maximum sharing of codes between the platforms in Xamarin." The output for this research question should contain brief useful information on the efficient pattern that can be used in Xamarin to increase the use of shared code. The research will be dealing with only two platforms: Android and iOS

I) Planning

The start of the research question was preceded by a time duration proposal (refer "Time Management for Research Questions.doc") of 5 weeks for this 2nd research question. After the proposal was made and approved, a planning for possible activities during the research question was done. Finally, these listed tasks were performed during the research period of the 1st research question and the research information acquired is noted.

The research consisted of learning different type of patterns. The activities were to be planned, in order to make sure the result obtained at the end of 2nd research question is on specified time and to avoid getting lost during the research journey. The planning for the lists of possible activities to be carried out is as follows:

- Week 13: Learning about Windows Presentation Foundation (WPF) patterns/model.
- Week 14: Learning different types of WPF models and their uses in creating different types of applications. Viewing the models adapted on existing Xamarin example applications.
- Week 15 : Learning about Xamarin.mobile
- Week 16: Choosing the most effective model for Xamarin Cross platform application development that makes high reuse/sharing of codes and designing a prototype cross platform app (with respect to the chosen model).
- Week 17: Initially planned- Implementing Xamarin.mobile to fit the design in Week 16. Changed the activity at the end of week 16 to- Further implementation of MVVMCross framework Reason: Due to unfit of this activity to the context. The application that was designed using the MVVMCross framework doesn't make use of any APIs (contacts, camera etc.) provided by the Xamarin.mobile. In case, if there is need of such APIs, MVVMCross framework provides plugins for such APIs that can be used easily.

II) Learning on Patterns and implementation approaches

According to the planned activities second research question deals about learning the different kinds of patterns as there is a need of providing recommendation on the use of pattern for the cross platform application development.

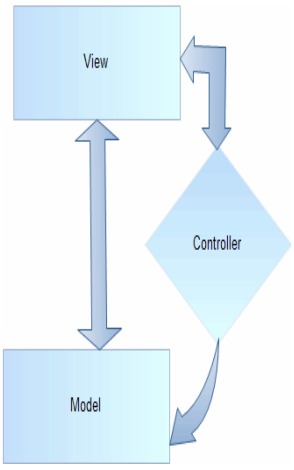
A) Patterns

Patterns are the frameworks designed in order to make the application development efficient by the separation of Views or User Interfaces (UI) and the actual business logics of the application. This enables to freely make changes in the user interface design without the change in the business logic implemented. This technique also allows UI designers independently design the UI part and let the software engineers implement the real business logic of the application. It also increases testability, maintainability and extensibility of the software application built.

The decoupled views and the business logic are interconnected (binded) to each other by a thin layer. This thin layer is designed form different kinds of patterns. Among many existing patterns, the mostly used patterns are Model View Controller (MVC), Model View Presenter (MVP) and Model View ViewModel (MVVM). Across all these patterns, the View and the Model are pretty much the same but simply changes in the connectivity technique between these elements either through the Controller in the MVC, the Presenter in MVP and the ViewModel in MVVM. [12] These patterns is not only adapted for developing windows form or web applications but it is also used in the software development on other platforms according to the fit.

1. Model view controller (MVC)

View - Renders data - Receives events(like button press, selection change) and presenter as appropriate - Have basic validations(like invalid email, empty text,etc) - May retrieve data from Model directly
Controller - Supports View in complex decision making tasks. - Changes view when necessary - Communicate with Model - Complex validations(which involves database query) - Queries model - Retrieves data from model, formats it and send it to the view
Model - Communication with DB layer - Raising appropriate event when dataset is generated.

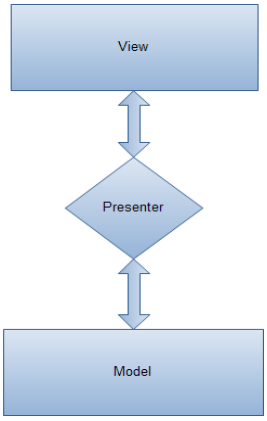


When to use MVC? [13]

- Use in situations where the connection between the view and the rest of the program is not always available (and you can't effectively employ MVVM or MVP).
- This clearly describes the situation where a web API is separated from the data sent to the client browsers. Microsoft's ASP.NET MVC is a great tool for managing such situations and provides a very clear MVC framework.

2. *Model view presenter (MVP)*

View - Renders data - Receives events(like button press, selection change) and presenter as appropriate - Have basic validations(like invalid email, empty text,etc)
Presenter - Supports View in complex decision making tasks. - Communicate with Model - Complex validations(which involves database query) - Queries model - Retrieves data from model, formats it and send it to the view
Model - Communication with DB layer - Raising appropriate event when dataset is generated.



When to use MVP? [13]

- Use in situations where binding via a data-context is not possible.
- Windows Forms is a perfect example of this. In order to separate the view from the model, a presenter is needed. Since the view cannot directly bind to the presenter, information must be passed to it view an interface (IView).

3. *model view viewmodel (MVVM)*

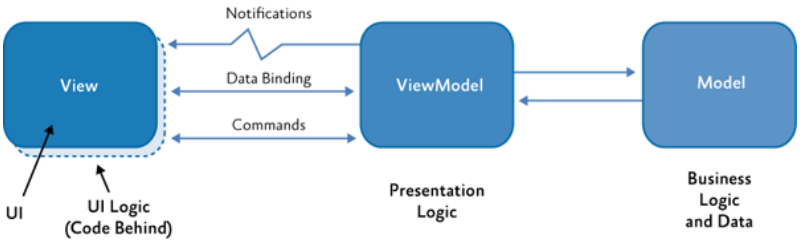


Figure II.8: MVVM Pattern

As seen in the above Figure II.8, the ViewModel (VM) acts as the mediator between the Model and the View. There is data binding, Commands and Notifications present between the View and the VM. These elements make the interconnection between the View and the VM. Whatever events occur in the View is sent to the VM. The VM then changes the data in the Model and after the task is done, the VM sends notification to the View what to do (display).

When to use MVVM? [13]

- Use in situations where binding via a data-context is possible. Why? The various IView interfaces for each view are removed which means less code to maintain.
- Some examples where MVVM is possible include WPF and javascript projects using Knockout.

B) Frameworks in Xamarin

After learning about different types of patterns, a suitable framework for Xamarin needs to be adapted. Searching the frameworks on the internet, two types of popular frameworks were found to be used on Xamarin; MonoCross and MVVMCross (MVX). The MonoCross adapts the MVC pattern whereas MVX adapts MVVM pattern.

MVX is very active at the current moment and has lots of incredible features (discussed in C). In fact, MVX is developed from the long ago Github fork of MonoCross project, shifting from MVC to MVVM pattern. Hence, being developed from experience of MonoCross, MvvmLight (MVVM tool in C#) and being the most active framework at the present context, MVX framework was the chosen framework (MVVM pattern) to be adapted in Xamarin. Further research step consists of collecting information and investigation on MVX.

C) MVVMCross (MVX)

MVX is an Open source cross platform framework for MonoTouch (Xmarin.iOS), MonoDroid (Xmarin.Android, WP7 and WinRT). It utilizes the MVVM C# pattern for the cross platform native application development. It was started by Stuart Lodge in November 2011. [14] [15] This framework makes extensive use of Portable Class library (PCL).

1. Architecture using MVX

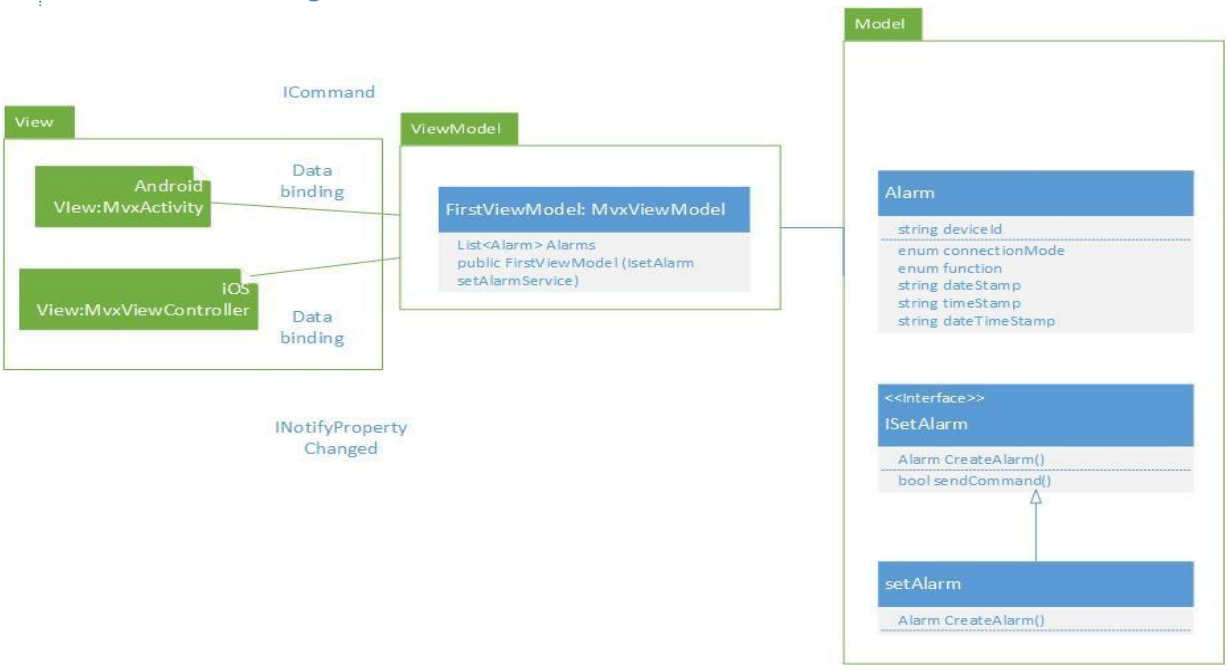


Figure II.9: MVVM .Net Implementation

The MVVMCross framework makes the use of MVVM pattern as seen in Figure II.9 for a WUL Alarm app. It consists of View, View-Model (VM) and the Model. It represents the state and behavior of the presentation independently of the GUI controls used in the interface. To revise, View consists of all the UI rendering codes and the implementation of platform specific codes like push notifications in iOS or File I/O access across platforms. Model consists of all the business logic and services like database, web services. The VM acts as a mediator between the View and the model.

ICommand and INotifyPropertyChanged play a great role in the communication between the View and the VM. They lie on the VM implementation. They contain all sorts of triggers to actions (clicks) in the View and the update of the view. The actions on View that are binded with the ICommand present in the VM, it performs specific operations that sets the different properties present in the VM. When there is a change in the properties, using the INotifyPropertyChanged event, the View elements binded to that changed properties updates according to the updated data in the VM.

ICommand can be seen as below which is triggered on the click action of a button on the View. The button's click action is binded to the ICommand btnClickAction.

```
Private System.MvvmCross.ViewModels.MvxCommand  btnClickAction;
Public System.Windows.Input.ICommand BtnClickAction{
 get{btnClickAction=btnClickAction ?? newCirrious.MvvmkCross.ViewModels.MvxCommand (ClickOperation);
return btnClickAction;}}

Private Void ClickOperation(){
//set or change properties that will in turn activate the INotifyPropertychanged event to update the view
SetPproperty= someValue;}
```

The INotifyPropertyChanged (RaisePropertyChanged) can be seen as below which updates the value of the setProperty (binded to some textview's or buttons text) on the view.

```
Private string setProperty;
Public String SetProperty{
get{return setProperty;}
set{setProperty=value; RaisePropertyChanged(()=>Hello);}}
```

MVX framework has lots of features (Research Question 2.II.C.3) cross platform application development.

2. Implementing the MVX in cross platform application.

Implementing the MVX in the Xamarin application also contains multiple projects in a solution project, like in the cross platform approach while using just Xamarin. That means 3 projects for targeting two platforms (Android & iOS in this research). Since, MVX makes extensive use of Portable Class Library (PCL), one project is the PCL project (core project), which consists of all the shared codes across the platforms. The other two projects consist of the View implementation for Android and iOS consisting of the UI implementation and platform specific implementation that cannot be implemented in the PCL project. The information is collected on the basis of learning from the n+1 tutorial found on the YouTube and Code Project by Stuart Lodge. [16] [16].

a. PCL project (Core Project)

The PCL project used in the implementation of MVX framework is a Portable Class Library of Profile 104. This consists of only the portable codes and cannot be used to implement the platform specific implementations. This profile defines a small subset of .Net that contains parts of the assemblies for:

- mscorlib
- System.Core
- System.Net
- System.Runtime.Serialization
- System.ServiceModel
- System.Windows
- System.Xml
- System.Xml.Linq
- System.Xml.Serialization

Profile104 consists of the all the necessities needed to build the MVVM applications. It doesn't support the conditional compilation for platforms and restricts to use it. While creating a PCL project in Visual studio, the default PCL might not be of this profile. Also, the target frameworks in the PCL (in Visual Studio) might not contain Android and iOS. Hence, for creating the required PCL to work for MVX, some tweaks to make the Profile 104 [17]and to include also Android and iOS target framework [18] need to be performed (currently).

A general MVVMCross Core PCL project consists:

- Assemblies reference- The reference to MVVMCross assemblies [19] need to be provided. For examples: Cirrious.CrossCore.dll (core interfaces and concepts including Trace, Inversion of Control (IoC) and Plugin management) and Cirrious.MvvmCross.dll (Mvvm classes including base classes for the MvxApplication and our MvxViewModels) etc.

- Model: Business logic or Services (Interface and implementation pairs) and database

- View-Model- Inherited from MvxViewModel, uses the services and contains public properties that call RaisePropertyChanged Event.

- packages.config: This is an external file with packages of MVVMCross that resides in all the three projects (Core PCL, Android and iOS) targeting iOS and Android as follows:

```
<?xml version="1.0" encoding "utf-8"?>
<packages>
<package id="MVVMCross.HotTuna.CrossCore" version ="3.0.4" targetFramework="portable-
win+net45+MonoAndroid16+MonoTouch40+sl40+wp71"/>
<package id="MVVMCross.HotTuna.StarterPack" version= "3.0.4" targetFramework="portable-
win+net45+MonoAndroid16+MonoTouch40+sl40+wp71"/>
 </packages>
```

- App.cs: Need to add App.cs to the root namespace (folder). Contains the application wiring, including the start instructions. It inherits MvxApplication, registers the services and registers special object for the MVX framework (IMvxAppStart).

b. Android Project (Android View)

The Android project consists of all the code implementations/Xml that renders the Android UI (to provide native UI) and the platform specific implementations like File I/O access.

A general MVVMCross Android project consists of:

- Normal Android application constructs: such as Assets folder, Resources folder and the activity class.

- Assemblies reference: The reference to the Core PCL project needs to be provided, as well as to the MVVMCross assemblies. For example: Cirrious.CrossCore.dll (core interfaces and concepts including Trace, IoC and Plugin management), Cirrious.MvvmCross.Binding.dll (DataBinding classes which is mainly used from XML), Cirrious.MvvmCross.dll (Mvvm classes - including base classes for our views and viewmodels), Droid specific version references (like Cirrious.CrossCore.Droid.dll Cirrious.MvvmCross.Binding.Droid.dll, Cirrious.MvvmCross.Droid.dll) etc.

- MvvmCross Android Binding Resource File: The MvxBindingAttributes.xml seen in figure II.10 should be added to the /Resources/values folder. This is required to use the binding and other features by the MVVMCross.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <declare-styleable name="MvxBinding">
    **<attr name="MvxBind" format="string"/>**
    <attr name="MvxLang" format="string"/>
  </declare-styleable>
  <declare-styleable name="MvxListView">
    <attr name="MvxItemTemplate" format="string"/>
    <attr name="MvxDropDownItemTemplate" format="string"/>
  </declare-styleable>
  <item type="id" name="MvxBindingTagUnique"/>
  <declare-styleable name="MvxImageView">
    <attr name="MvxSource" format="string"/>
  </declare-styleable>
</resources>
```

Figure II.10: MvxBindingAttributes.xml

- Setup.cs: This class is added in the root namespace. It performs the initialization of the MvvmCrossframework and your application, including:
  - the Inversion of Control (IoC) system
  - the MvvmCross data-binding
  - our App and its collection of ViewModels
  - our UI project and its collection of Views

Most of this functionality is provided automatically. Within the Android UI project, all we have to supply are App, business logic and ViewModel content and some initialization for the Json.Net plugin and for the navigation mechanism. However, the use of Json is not mandatory and other serialization mechanisms can be adapted either.

- Views and Bindings: After the general structure is ready, we can start constructing the UI for our application to consume the PCL core project. The binding between the UI and the ViewModel created can be done using the MvxBind. For example, if we need to bind the Text property of a TextView in the Android view, with the string property "TextPropInVM" in the Core PCL, then we can simply bind them as in Figure II.11. The term "local" seen is referenced from MvxBindingAttributes.xml as seen in Figure II.10.

```
<TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        local:MvxBind="Text TextPropInVM " />
</LinearLayout>
```

Figure II.11: Binding Android View and Core PCL VM

A general MVVMCross Android application is ready after creating the view and binding it to the VM of the core project. Hence, it can be seen that just after binding the view to the VM of the Core PCL project, makes an app ready and proves that the UI and the business are really decoupled to maximum extent while making great use of Core PCL (shared codes).

c. iOS Project (iOS View)

The iOS project consists of all the code implementations/View Controllers that renders the iOS UI (to provide native UI) and the platform specific implementations like push notifications.

Like in the Android project, a general MVVMCross iOS project consists of:

- Normal Android application constructs: The default constructs while creating an iOS app like the Resources folder the info.plist 'configuration' information, the AppDelegate.cs class, the Main.cs class and the MyViewController.cs class. The AppDelegate provides a set of callback that iOS uses to inform about events in the application's lifecycle. To use this AppDelegate within MvvmCross, we need to modify it so that it inherits from MvxApplicationDelegate instead of UIApplicationDelegate:

- Assemblies reference: The MVVMCross assemblies and Android Specific assemblies for MVVMCross are referenced.

- Setup.cs: The Setup in iOS performs the same task as in the Android.

- Views and Bindings: The view items in the View Controller (VC) can be created using Xcode Interface Builder (XIB) in Mac (Xamarin Studio) or even programmatically (both Xamarin and VisualStudio). However, the VC should inherit the MvxViewController as seen in figure II.12, instead of the default UIViewController. The VM related specifically to this VC is too assigned, so that the specific view load that is related to its VM.

```
public partial class TipView : MvxViewController   {
        public new TipViewModel ViewModel        {
            get { return (TipViewModel)base.ViewModel; }
            set { base.ViewModel = value; }        }
          .....     }
```

Figure II.12: iOS View Controller

After the outlets (object reference to the UI components) are created from the XIB, bindings can be made between the view items and the VM properties when the VC loads. Binding a UILabel's text property to the string property TextPropInVM in the Core VM can be seen in Figure II.13.

```
public override void ViewDidLoad ()
        {
            base.ViewDidLoad ();

            this.CreateBinding (this.TouchLabel).To( (PCLCoreVM vm) =>
                  vm. TextPropInVM ).Apply();
        }
```

Figure II.13: iOS Binding

Similar to MVVMCross Android app, general MVVMCross iOS application is ready just after creating the view and binding it to the VM in the core project. This too shows the decoupling of the View and the VM and making extensive use of shared codes using the PCL.

3. different Concepts in MVVMCross app development

a) Multiple screens

Cross platform applications with multiple screens can be too created using the MVX framework. Instead of using Intents from one screen to open other screen, MVX application follow a different strategy. Since, the Views and the ViewModels are binded together using the MVX framework, VM are responsible for which screen to load. So, when there is a need of certain screen, the VM binded with its view can be activated. For example: When the app start, the first VM activates which loads its binded view (first screen). So, to open a different screen when a button-click on this first screen, the click event of the first screen (button) is binded to the first VM (Figure II.13). From this first VM, the SecondViewModel (VM) that is binded with the second Screen can be activated (Figure II.14). Hence, activating the SsecondViewModel (VM) from the first VM will enable the app to show the second screen. This way, multiscreen applications can be built using MVX using the VM transitions.

```
public void GoToSecondVM ()
        {
          Base.ShowViwModel<SecondViewModel>();  //base class method
        }
```

Figure II.14: Activating SecondViewModel from a VM

Technically, adding ICommand's to ViewModel's, binding ICommand's to Views (Buttons), adding multiple view models, adding multiple views, finally navigating between views/view models make a multiscreen mvx application.

b) Passing information between the screen

At certain point during the application development, there arises the need of passing information (data) between the screens. This can be accomplished simply by passing the required data using the VM (binded with the active screen) that activates the other VM (binded to the next screen) while loading next screen as below:.

```
//Method to go to the second VM
public void GoToSecondVM ()
        {
          Base.ShowViwModel<SecondViewModel>(new{passData="Hello"});
//base class method
        }
```

Using the new keyword above creates a new anonymous class and the compiler generates that as an internal class so we need to use it externally. Hence, we need to add assembly attributes in the AssemblyInfo.cs as:

```
 [assembly: InternalISVisibleTo("Cirrious.MvvmCross")]
```

The reserved Init() VM method as seen below can be used to pass the information between the information as seen above.

```
//VM reserved Init() method implemented in the second VM to pass data
Public void Init (string passData)
{
SetProperties= passData;
}
```

c) Gestures

As gestures relates to the platform specific implementation, they live on the View project of the MVX application and they are implemented in each platform separately. There are various ways to implement gesture recognizer. For example, in iOS a tap gesture recognizer implementation would look something like this:

```
Var gesture= nwe UITapGestureRecognizer(()=>
{
//The operation to be performed on the tap gesture
})
View.AddGestureRecognizer(gesture);
```

Where, View is the main view where the gesture is to be implemented

d) Plugin/ NuGet (Native abstraction)

MVVMCross NuGets are the packages available for an easy development. They contain a template (default Classes and assemblies) for the MVVMcross application development. These also contain the implementations of the Native APIs that can be shared across platforms. Since, the MVX framework adapts the PCL project but the PCL doesn't support the platform specific implementations. For this, we can use the existing plugins in the MVVMCross or create a custom plugin to be used to access the native APIs in PCL. However, for a custom plugin the implementation needs to be done on each number of targets.

1. Declaring common functionality (an interface)
   An interface can be declared in the PCL Core project that consists of the functionalities to be used which should be registered as a MVX plugin.
2. Writing platform specific implementations
   The platform specific implementations for the specific platforms can be done in the view project (iOS and Android) which implements the interface declared in the PCL Core project.
3. Using the interface and not the implementation
   The interface is declared and the implementation is done in the platform specific view project. Now, for consuming the platform specific functions from the PCL, the interface implemented by the View can be used.

e) Xamarin.Mobile

Xamarin.Mobile is a library that exposes a single set of APIs for accessing common mobile device functionality across iOS, Android, and Windows platforms (Figure II.15). This increases the amount of code developers can share across mobile platforms, making mobile app development easier and faster. Xamarin.Mobile currently abstracts the contacts, camera, and geo-location APIs across iOS, Android and Windows platforms. Future plans include notifications and accelerometer services. [5] It maps to native implementation on each platform.

Figure II.15: Xamarin.mobile

To use contacts on Android and iOS, we can use:

```
Var book= new AddressBook(){PreferContactAggregation=true};
Foreach(Contact c in book.Where(c=>LastName== “search keyword”)){
Console.WriteLine(C.DisplayName);
Foreach(Phone p in c.Phones)
Console.WriteLine(“Phone: ”+p.Number);
Foreach (Email e in c.Emails)
Console.WriteLine(“Email: “+e.Address);}
```

Xamarin.Mobile can be a boon in the cross platform development using Xamarin. But, when we adapt an MVVMCross framework, even Xamarin.Mobile seems ligther as MVVMCross provides plugin features containing lots of functions (except for Contacts) that Xamarin.Mobile includes and other many functions that Xamarin.Mobile doesn’t include like Accelerometer, Bluetooth etc. Xamarin.Mobile is not portable code - it can't be called directly from PCLs. The first version of Xamarin PCL support is supposed to be due very soon. However, it can be used in the PCL by making some tweaks. But, instead of using the Xamarin.mobile, looking at the plugins for the APIs already existing in the MVVMCross, Xamarin.mobile can be skipped to be used while adapting the MVVMCross framework.

f) Native libraries

Getting the P/Invoke mechanism work (Appendix D. Research Question 1. II.D2), if we want to use the functions in the native libraries in the PCL, then we can follow the same procedure as mentioned above in using Plugin (3.d).

4. features

The advantages of MVVMCross include: [20]:

- Binding support on iOS (XIB) and Android (XML)
- A lot of app behavior gets put into VM and is reusable across platforms
- Since VM are platform-agnostic unit tests can be written that covers most of the app's behavior
- View models are shared, but views remain totally native
- VM can be customized on how to present based on the needs (such as showing multiple at once for a tablet version)
- The framework itself is very extensible, allowing for a lot of customization when needed.

5. Exisiting applications using this framework

Lots of most demanding scenarios, like high frame rate gaming and complex data visualizations have been built using the Xamarin and MVVMCross framework as below: [21]

- Kinect Star Wars - http://www.youtube.com/watch?v=MXPE2iTvlWg
- Aviva Drive - http://www.aviva.co.uk/drive
- Origo Foci-Eb 2012 - http://slodge.blogspot.co.uk/2012/10/origo-foci-eb-2012-example-mvvmcross.html
- The CrossBox DropBox client - https://github.com/runegri/CrossBox
- The Blooor shopping list app - https://github.com/Zoldeper/Blooor
- Various Conference apps - SQLBitsX, DDDSW, LondonAzure, https://github.com/slodge/MvvmCross/tree/vnext/Sample%20-%20CirriousConference

6. bottleneck:

The biggest problem is the setting up the MVVMCross framework. To utilize the MVX framework, NuGet packages are used officially in VisualStudio using the Xamarin.iOS and Xamarin.Android plugin. Using the Visualstudio for the development, the PCL support for the Xamarin.iOS and Xamarin.Android frameworks doesn't exist by default. Hence, tweaks must be done in order to make use of them. [17]. Even following the tweaks didn't work out in this project. The NuGet package for using the MVVMCross requires NuGet 2.5 which isn't yet finished/released for Xamarin Studio on Mac. Hence, unable to use the NuGet, all the references, required classes should be done manually on Mac. Even after the modeling of the application, due to these problems at the present context, while trying to manually work with the assemblies for the MVVMCross framework [18] couldn't be set properly. The straight forward getting started with the MVVMCross isn't yet perfect. Hence, though MVVMCross being powerful, the initial setup is really the main bottleneck during the development.

III) Conclusion and recommendation

Xamarin with MVVMCross framework is a very powerful combination for the cross platform mobile application development. These two combined boosts up the cross platform application development process by giving a single language of C# for the development across different platforms. They enable shared codes across different platforms, provide native look and feel to the cross app, support native and existing libraries, high performance apps and large applications have a common architecture. However, for the cross platform support, the use of different IDEs (Xamarin, MonoDevelop, VisualStudio) the development using Xamarin components and MVVMCross framework make a tiresome work to find the correct path. But once, the correct path is known, it makes the whole life easier for the cross platform application development. To conclude, a native-cross platform application can be built with excessive sharing of codes across platforms using Xamarin and MVVMCross framework together.

The recommendation would be to make use of Xamarin and MVVMCross framework for the development. Despite of the disability to set up the framework for use and some overheads, based on the research information on the features and supports of Xamarin and MVVMCross, they should be applied in the cross platform mobile application development.

BIBLIOGRAPHY

[1] "Xamarin Guides documentation," [Online]. Available: http://docs.xamarin.com/guides.

[2] "Xamarin Seminar by Greg Shakles," [Online]. Available: https://www.youtube.com/watch?v=WkNbRUqnSSc.

[3] "Xamarin.Android limitations," [Online]. Available: http://docs.xamarin.com/guides/android/advanced_topics/limitations.

[4] "xamarin.iOS Limitations," [Online]. Available: http://docs.xamarin.com/guides/ios/advanced_topics/limitations.

[5] "Xamarin Doc on Objective C library integration," [Online]. Available: http://docs.xamarin.com/guides/ios/advanced_topics/native_interop.

[6] "Xamarin Doc onJava Integration on Xamarin," [Online]. Available: http://docs.xamarin.com/guides/android/advanced_topics/java_integration_overview.

[7] *Stackoverflow forum on Native library uses on Xamarin.*

[8] "Swig tool for C/C++ Binding," [Online]. Available: http://www.swig.org/.

[9] "Blog on bridging C++ and C# using CXXI," [Online]. Available: http://tirania.org/blog/archive/2011/Dec-19.html.

[10] "Blog on using SWIG for C++ binding," [Online]. Available: http://blog.reblochon.org/2013/01/c-bindings-for-monotouch-using-swig.html.

[11] "Xamarin Docs on using Native libraries on Xamarin.Android," [Online]. Available: http://docs.xamarin.com/guides/android/advanced_topics/using_native_libraries.

[12] [Online]. Available: http://www.codeproject.com/Articles/288928/Differences-between-MVC-and-MVP-for-Beginners.

[13] [Online]. Available: http://joel.inpointform.net/software-development/mvvm-vs-mvp-vs-mvc-the-differences-explained/.

[14] [Online]. Available: http://slodge.blogspot.co.uk/2012/12/mvvmcross-video-presentation-xaminar.html.

[15] [Online]. Available: https://github.com/slodge/MvvmCross.

[16] "n+1 MVVMCross tutorial by Stuart Lodge," [Online]. Available: http://www.youtube.com/user/MrHollywoof?feature=watch.

[17] [Online]. Available: http://www.codeproject.com/Articles/566191/MvvmCross-v3-Writing-a-First-App.

[18] [Online]. Available: https://www.youtube.com/watch?v=UC2r4mmj3UI&feature=youtube_gdata.

[19] [Online]. Available: http://slodge.blogspot.de/2012/12/cross-platform-winrt-monodroid.html.

[20] [Online]. Available: https://github.com/slodge/MvvmCross-Binaries/tree/master/VS2012/bin/Debug/Mvx.

[21] [Online]. Available: http://monotouch.2284126.n4.nabble.com/MonoCross-and-or-MVVMCross-experiences-td4657938.html.