A PROJECT REPORT

ON

"CONTRIBUTION TO XAMARIN: GENERATING XAML CODE BY PROVIDING GUI"

Submitted by

ANAGHA CHIPLUNKAR : [Exam Seat No: B120208507]
NEHA DHAKNE : [Exam Seat No: B120208535]
GAURI DUDHMANDE : [Exam Seat No: B120208514]
:

Under the guidance of

Prof. Suchitra Pakale

(Assistant Professor, Department of Information Technology)

DEPARTMENT OF INFORMATION TECHNOLOGY

MKSSS'S CUMMINS COLLEGE OF ENGINEERING FOR WOMEN, PUNE

2017-18

AFFILIATED TO

SAVITRIBAI PHULE PUNE UNIVERSITY

A PROJECT REPORT

ON

"CONTRIBUTION TO XAMARIN: GENERATING XAML CODE BY PROVIDING GUI"

Submitted to

SAVITRIBAI PHULE PUNE UNIVERSITY

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

INFORMATION TECHNOLOGY

by

ANAGHA CHIPLUNKAR : [Exam Seat No: B120208507]
NEHA DHAKNE : [Exam Seat No: B120208535]
GAURI DUDHMANDE : [Exam Seat No: B120208514]
:

Under the guidance of

Prof. Suchitra Pakale

DEPARTMENT OF INFORMATION TECHNOLOGY

MKSSS'S CUMMINS COLLEGE OF ENGINEERING FOR WOMEN, PUNE

AFFILIATED TO

SAVITRIBAI PHULE PUNE UNIVERSITY

MKSSS's Cummins College of Engineering for Women, Pune

CERTIFICATE

Certified that this project report titled "**CONTRIBUTION TO XAMARIN: GENERATING XAML CODE BY PROVIDING GUI**" is the bonafide work of

ANAGHA CHIPLUNKAR	**:**	**[Exam Seat No: B120208507]**
NEHA DHAKNE	**:**	**[Exam Seat No: B120208535]**
GAURI DUDHMANDE	**:**	**[Exam Seat No: B120208514]**
	:	

is a record of bonafide work carried out by them, in the partial fulfilment of the requirement for the award of Degree of Bachelor of Engineering, department of Information Technology, MKSSS's Cummins College of Engineering For Women, Pune, under the Savitribai Phule Pune University. This work is done during year 2017-18, under our guidance.

Prof. Suchitra Pakale
Guide
Dept. of Information Technology

Dr. Anagha Kulkarni
HOD
Dept. of Information Technology

Dr. Madhuri Khambete
Director

External Examiner

ABSTRACT

User Interface (UI) is the user facing side of any application. For the application to be user friendly, it goes without saying that the UI must be intuitive and visually appealing.

Developers cannot make a appealing UI if they themselves have to imagine it first. If they have a visual representation of what they are developing, chances are that it is better and closer to what user sees.

In Xamarin, developers need to code it in XAML, the UI definition language in Xamarin. The previewer then parses the code to render what it will translate to. That is cumbersome leading to people either adapting native dev-environments forgoing the benefits of cross platform app development or making a UI that is not user friendly. To solve this problem we wrote a utility that generates XAML code based on the UI.

ACKNOWLEDGEMENTS

During this project several people have provided many forms of help and support. Firstly I would like to thank Prof. Suchitra Pakale (Project Guide) for helping me to do this project and for continuous guidance throughout the project. Secondly I would like to thank the other team members who have provided ideas, been cooperative and made the team work so well. There have been no occasions where a conflict of opinion has not been resolved successfully. Finally, I would like to thank Harshad Wadkar (Project Co-ordinator) for encouraging us to take up this project. Without them it is likely I would not be on this project and even if I were the chances are I would be struggling.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Necessity of a GUI tool for developers

The trend of adopting held devices(smart phones and tablets) and wearables on the rise, owing to the ubiquity of the Internet and BYOD(Bring your own device) policies by corporates. This leads to a demand for new kind of software - that runs on these devices. Collectively these softwares are called "apps".

With multiple device vendors, there exist multiple programming interfaces. Thus a "app" needs to be written twice or more times, depending on the number of platforms that the developers want to target.

This increases the "cost-to-develop" and as well "the-time-to-market" decreasing on the revenues. Thus came in cross-platform tools with the paradigm of WORA ("Write once run anywhere").

Xamarin is one such tool. It is open-source and owned by Microsoft. Xamarin supports the Model-View-ViewModel VVM (MVVM) paradigm of development. It separates the UI and data model with a transfer path between the two. Thus for Xamarin, we use C-sharp to write the data model of the app and XAML to control and create the visual layout.

As a developer, we cannot view UI while coding it. The Android studio, XCode etc give this feature by providing "Drag-drop" UI builder. Using it the developer simply drags the elements like Layouts, Buttons etc on to a device sized Canvas and drops it to the desired location. The respective code is generated in the back-end. This "drag-drop" feature is unavailable for Xamarin. We wrote this utility and plan to contribute to Xamarin.

Due to the palette and drag-drop of controls, developer will be able view UI of the app, when it is being developed itself (WYSIWYG). Also the markup code will be generated automatically, in the background, saving the hastle and errors of coding.

1.2 Cross Platform Paradigm

The idea of cross-platform development is that a software application or product should work well in more than one platform.

Every platform has its own set of APIs for development. Thus, if a application needs to be deployed on more than one platform, we need people who know the development procedure on all target platforms. It is indeed resource and time consuming. Thus we come to cross platform tools : write once, run anywhere.

With mobile devices becoming increasingly ubiquitous, popular, powerful and diverse as well as with the high proliferation of open-source technologies like Linux, cross-platform development has gained momentum.Cross-platform software is said to be âĂIJplatform agnosticâĂİ in that it doesn't value or support one platform over another; Yet developers can use application programming interfaces (APIs) to adjust a piece of software to a specific platform.

To make a application compatible with multiple platforms, we use a common thread between all of the targets. These "common threads" can be HTML, Virtual machines or containers.The major approach is to make the program abstract at certain levels in order to accommodate different software environments.

In general, cross-platform development can make a program less efficient. For example, it can require redundant processes or file storage folders for the various systems that its supposed to support, It may also require that a program be âĂIJdumbed downâĂİ to accommodate less sophisticated software environments. However, in many cases, the limitations of cross-platform development are worth dealing with in order to offer an application or product to a wider set of users.

1.2.1 Native stack vs Cross-Platform tools

Native stacks: Each platform comes with its own development APIs. Thus when considering iOS development, most of us think about Objective-C vs Swift, and about Java while considering the Android Platform. These high level languages are compiled into interpretable codes, which are interpreted by the platform runtime like ART in Android. Native stacks offer access to the complete device like memory, camera etc. and also provide speed of execution as the build is device specific.

Turning to Cross-Platform: When one application needs to run in multiple environments, we need experts in every environment, bundling up our development costs. Thus we turn to cross platform tools like PhoneGap (derivative of Apache Cordova) or Microsoft Xamarin, Appcelerator Titanium. These tools, map a programming interface of a platform to a widely known language-framework like Java or Javascript. In the process, only common features can be abstracted, leaving out on the platform specifics.

1.2.2 Implementing Cross-Platform

Cross-Platform was a rosy dream till the advent of run-times. Runtimes were bought into limelight by Java, which even today is a sought-after skill. Virtual machines was a next step in the "WORA" idea. It was taken up and developed by Oracle. Containers are "the next".

One more approach to cross-platforms is the utilization of "the Web". Web-apps are essentially cross-platform apps. These apps cannot access the native hardware like camera, GPS, Contacts etc.

Ofcourse developers can mix and match these approaches as per requirements leading to a hybrid cross-platform app; which is widely deployed. Thus a native container is used to request a web-app. The native container acts as a bridge for data flowing to and from the device.

1.3 Xamarin

In Xamarin(tool), the business logic of the app is written in C-sharp and the UI is described in a XAML file. It allows engineers to share about 90 percent of code across major platforms. The tool is open source powered by Microsoft and already has a community of over 1.4 million developers.

The C-sharp code is shared between major platforms. Thus, along cross platform, developers can also extend this facility to write platform specific apps in C-sharp, a language that is known by many. Thus, developers can use Xamarin to write native Android, Windows and iOS apps with native user interfaces.

Unlike Phone Gap and its sister tools which rely on a HTML container, Xamarin complies the app into equivalent of native builds. This guarantees speed and accuracy of

execution, enabling developers to access native features like file and memory handling.

Figure 1.1: Xamarin Architecture

Up till its acquisition by Microsoft, Xamarin was a San Francisco -based software company founded by the engineers that created Mono - a open source runtime adapted to run in the Mac and Android environments. It is a multi-platform implementation of Common Language Specifications (often called Microsoft .NET)

Naturally it follows that Mono is installed on the target device with the Xamarin app, if not already present. Xamarin apps build code for Mono, which is the runtime environment for these apps. Thus Xamarin-Android apps bypass the Android runtime and run apps on the Mono run time.

1.4 XAML

XAML is a markup language developed by Microsoft. XAML provides an elegant definition of the user interface, and has a visual structure that mimics the tree organization of the visual elements on the page. XAML is also generally easier to maintain and modify then equivalent high-level language code. It is parallel to XML. UI in Xamarin can be declared through code (C-sharp), where in every UI element is a object or through the tree structure of XAML. A declarative UI (XAML) is a better approach than doing the UI by code, for the following reasons:

1. It's concise,
2. Its more maintainable, much more easier to read and modify than code.
3. It helps with a clear separation of concerns between the UI and logic.(MVVM approach)

CHAPTER 2

MOTIVATION

Every company wants to ride on the wave of "smart devices", by deploying consumer and/or employee-centric software to smart-devices. Thus there is need for proprietary apps, which are tailored for the organization. Thus, during our "project statement hunt mission", we came across many app making statements However, the condition was that apps that should run on both platforms.(To increase user-base).We searched for tools to make life simpler for us as developers and came across the concept of cross-platform tools. We studied each one to a certain depth and concluded Xamarin to be the better suited for our purpose (because of its native build). But the UI was not developer-friendly as Android studio (to which we were used to). Hence we took up this statement.

CHAPTER 3

LITERATURE SURVEY

Given the "off-beat-ness" of our project, we knew pre-written literature was hard to come by. Thus we relied mainly on Xanarin forums, Youtube channels, our understanding and of course the design guidance of our faculty; not to mention experimentation.

3.1 Android studio

Android studio was introduced to us via our University syllabus. Thus we were familiar with it. Thus we thought of extending the idea of a GUI designer to cross platform tool as well. Also we figured out that UI in general follows a nested hierarchy and thus,our utility could extend to web-front ends as well.In a nut shell: Though necessity is the mother of invention but Android Studio showed us the path.

3.2 Xamarin University

Xamarin, is a relatively new product. Its installation and usage had to be figured out. Hence we relied on xamarin (official site) Xamarin University, Forums and Youtube videos. Xamarin University is a place where experts "teach" Xamarin online. Also material and assignments(with solutions) are available on that site. It helps learn the platform quickly.

CHAPTER 4

EXISTNG SYSTEM

Here we explain working of Xamarin with respect to Android as the internal details of iOS are masked.

Android deploys Android Run Time. An Android app runs in the run-time environment of the ART. The .apk file contains the intermediate build of the high level logic defined by the developer. The run-time is responsible to interpret that code and execute it. Thus, compilation on android is JIT or just in time.

Xamarin apps do away with the ART completely. They rely on Mono, the open source parallel to the ART. Thus the C-sharp code is compiled into the apk by the platform with the help from Android SDKs. The apk is interpreted by Mono.

Apple also deploys a run time on iOS. However, iOS does not deploy a JIT strategy. Also the iOS SDK is available only on MAC. Thus, the C-sharp code is given to a mac to be compiled into a form the iOS can understand.

It is because of the above ingenuity, that Xamarin uses, C-sharp can be used for cross- platform app development. Such applications are often compared to native for both iOS and Android mobile development platforms in terms of performance and user experience.

4.1 Why Xamarin

1. One Technology Stack to Code for All Platforms.
2. Performance Close to Native.
3. Native User Experiences.
4. Full Hardware Support.
5. Open Source Technology with Strong Corporate Support.
6. Simplified Maintenance.
7. Xamarin.Forms: a framework for simple apps and prototypes.
8. Building apps for Macs with Xamarin.Mac Tool.
9. Xamarin University provides smooth onboarding

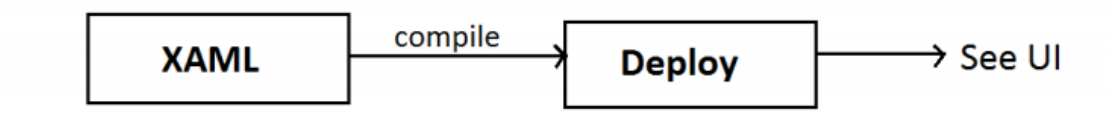
4.2 Why Not Xamarin

1. Slightly Delayed Support for the Latest Platform Updates.
2. Limited Access to Open Source Libraries.
3. Xamarin Ecosystem Problems.
4. Basic Knowledge of Native Languages Required.
5. Not Suitable for Apps with Heavy Graphics.
6. Compatibility Issues with Third-Party Libraries and Tools.

4.3 Our Contribution

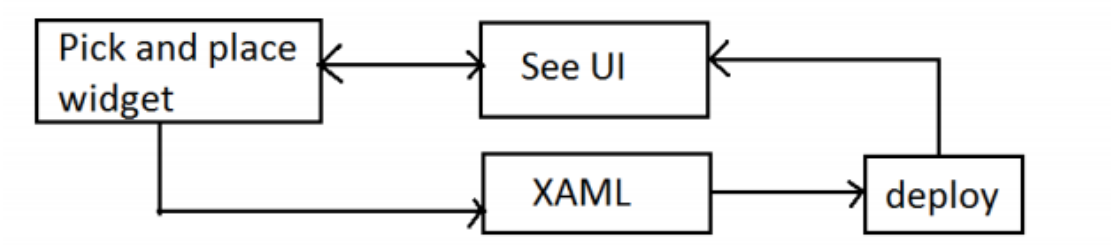
The UI in Xamarin is written in XAML. The Expression studio(now blend for visual studio) allows users to preview the UI generated by the XAML that they write. But most novice users do not know XAML. The know how that app must look. Thus they can design well but not code. A facility to convert from UI to XAML is currently, missing. It is only when the app is deployed that the user can see the implementation of the code. It is frustrating for any developer and designer.

Figure 4.1: Using Xamarin - Then



We wrote a utility for this, and have sent a pull request to the Xamarin team. It is our contribution to the open source community We drew our inspiration from the Android studio. In the studio, developers can pick-place widgets from a palette on a canvas resembling the device screen. The markup code is generated in the back-end. It saves coding hassles and time. Also the developer can see the output of the code, resulting in better UI. We provided this feature of drag and drop for Xamarin, so that the application developer can place widgets from a pallet. When application developer places a Widget in appropriate place for his UI then our system will generate an appropriate XAML code accordingly.

Figure 4.2: Using Xamarin - Now



For a app to be successful UI or presentation has a major role. we believe that our utility will find a sizeable user base

CHAPTER 5

SYSTEM DESIGN

In the following sections we describe our project in detail using UML diagrams.
UML stands for Unified modelling language; and is used as a communication medium between the designer, developer and client. It relies on the fact that a "picture speaks more than a thousand words."

5.1 Structural Diagrams

Structural diagrams represent the structure of a software. It is used to show how various components in the system interact. In this section we define the class diagram. Also we define a object diagram when the complete GUI is rendered and is ready for XAML export.

5.1.1 Class Diagram

A Class diagram enlists the major classes in the modified system and underlines their correlation. 5.1 shows the class diagram for our project.

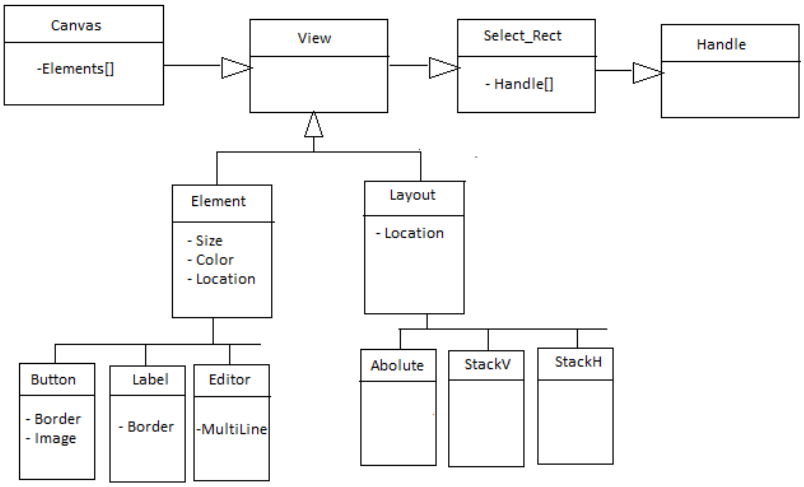
5.1.2 Object Diagram

According to UML standards, an Object diagram s a snapshot of the classes (members and methods) at a specific point. 5.2 presents a Object diagram at the instant wherein the GUI is completely designed and is ready to export to XAML.

5.2 Behavior Diagrams

Behavior diagrams depict how the system deals with data and state changes. Here we draw two Behavior diagrams to model our tool : the use case diagram and the activity digram.

Figure 5.1: Class Diagram



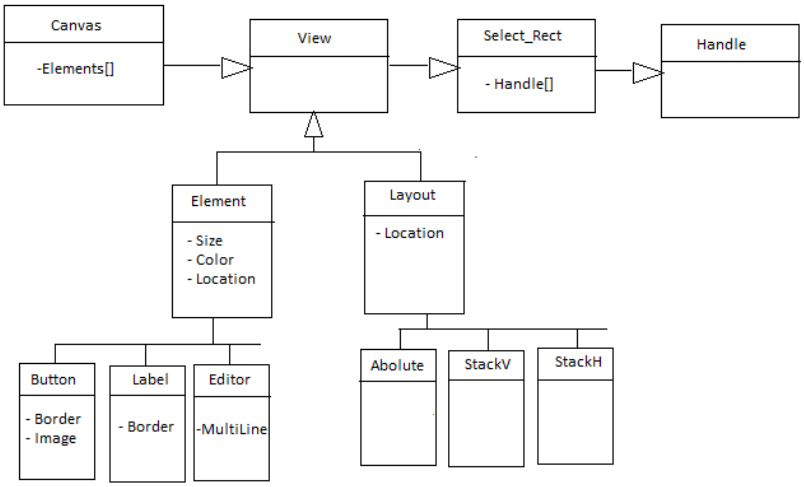
5.2.1 Use Case Diagram

It defines how different users are expected to interact with the system. Our system has only two main users namely the designer/programmer and Xamarin itself. 5.3 shows what actions each one is expected to perform on the system, before and after our contribution.

5.2.2 Activity Diagram

an Activity diagram models the different activities in the systems and their transitions. 5.5, 5.6 show activity transitions before and after our contribution.5.7 also lists the Activity diagram for the proposed scope.

Figure 5.2: Object Diagram



5.3 Interaction Diagrams

Interaction diagrams depict how system components interact with each other.

5.3.1 Sequence Diagram

Sequence Diagrams defines series of actions done by / on the software and its responses. 5.8 show sequence diagrams for Xamarin before and after our contribution.

Figure 5.3: Use Case Diagram - Then

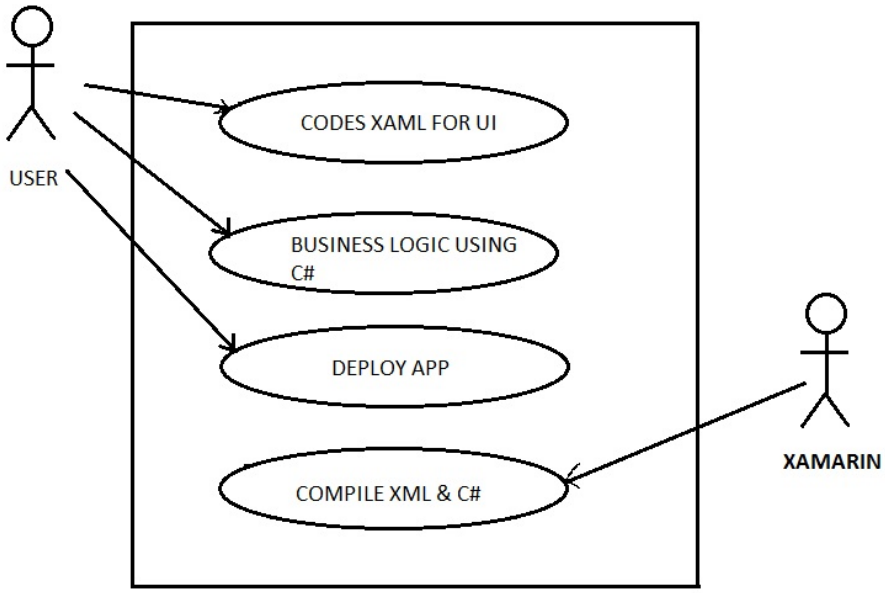


Figure 5.4: Use Case Diagram - Now

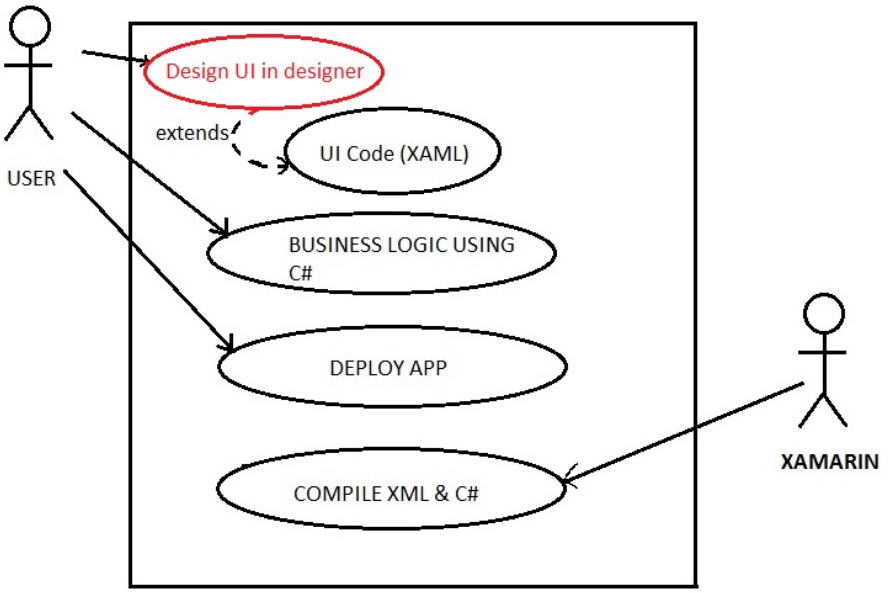


Figure 5.5: Activity Diagram - Then

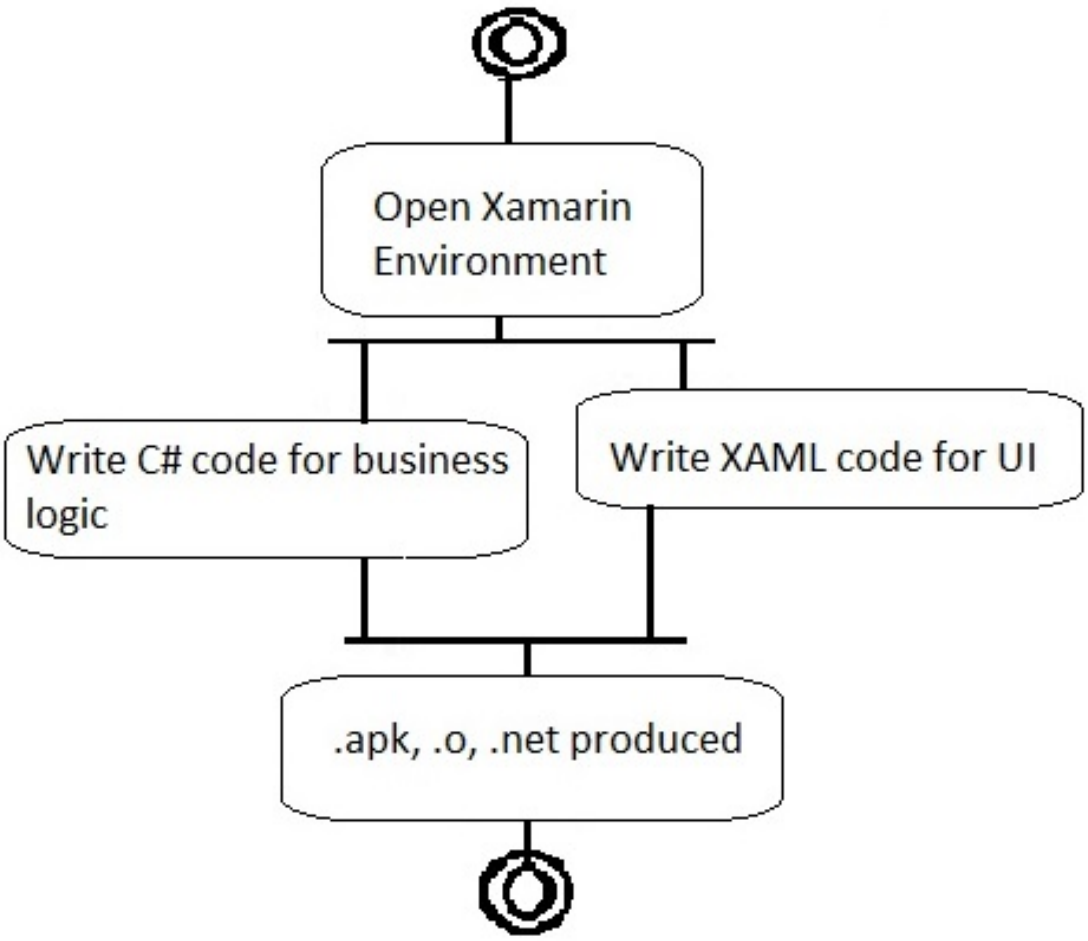


Figure 5.6: Activity Diagram - Now

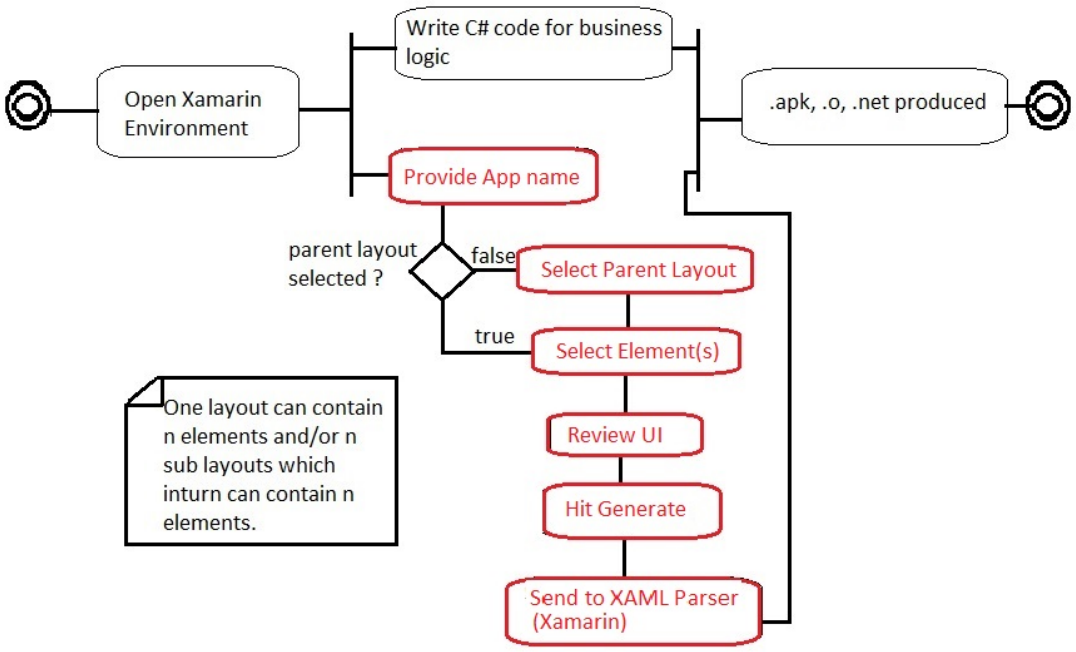


Figure 5.7: Activity Diagram - Future

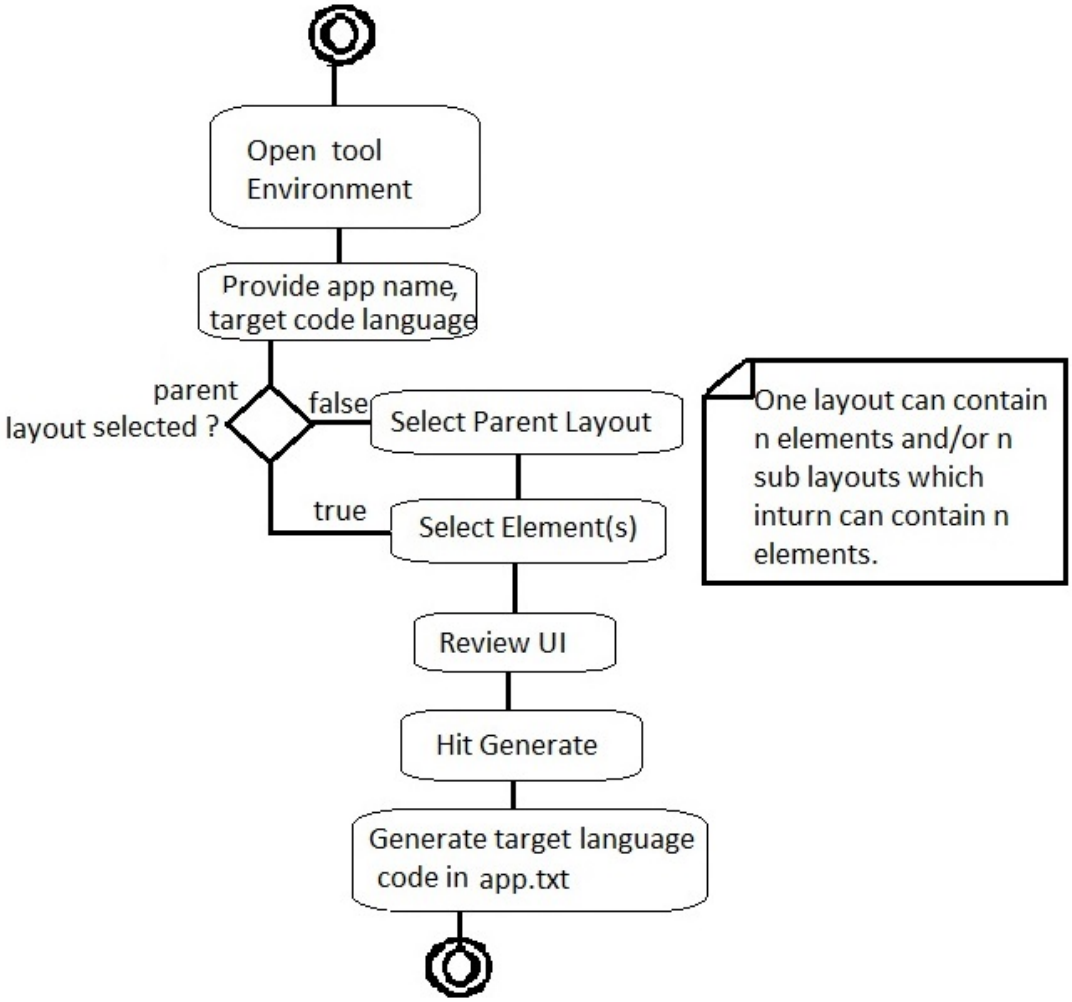


Figure 5.8: Sequence Diagram - Then

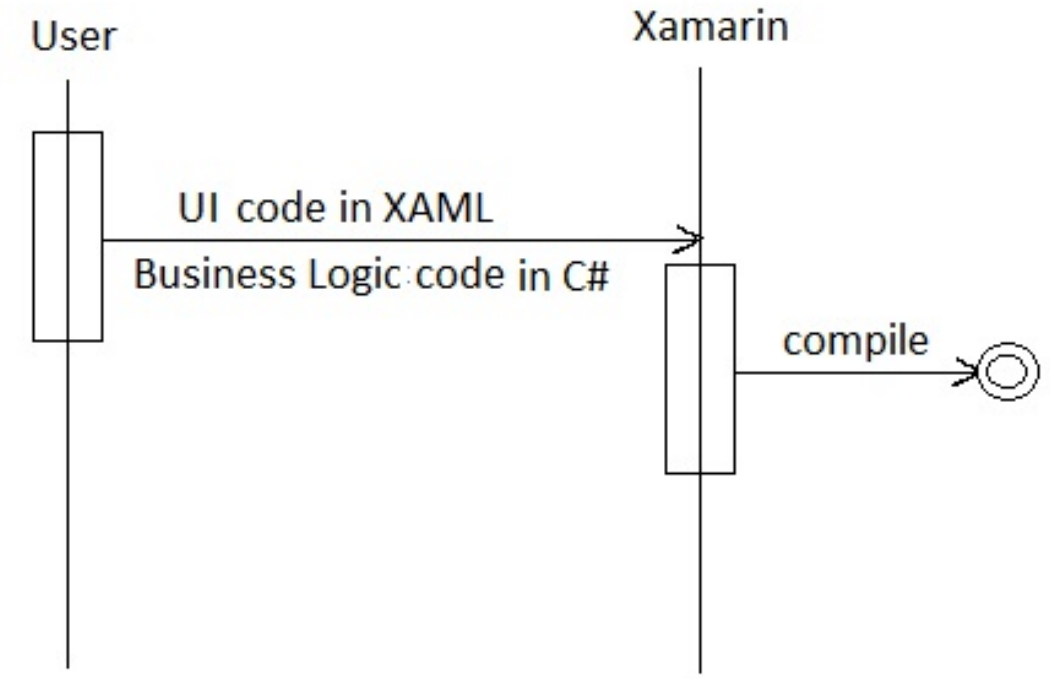
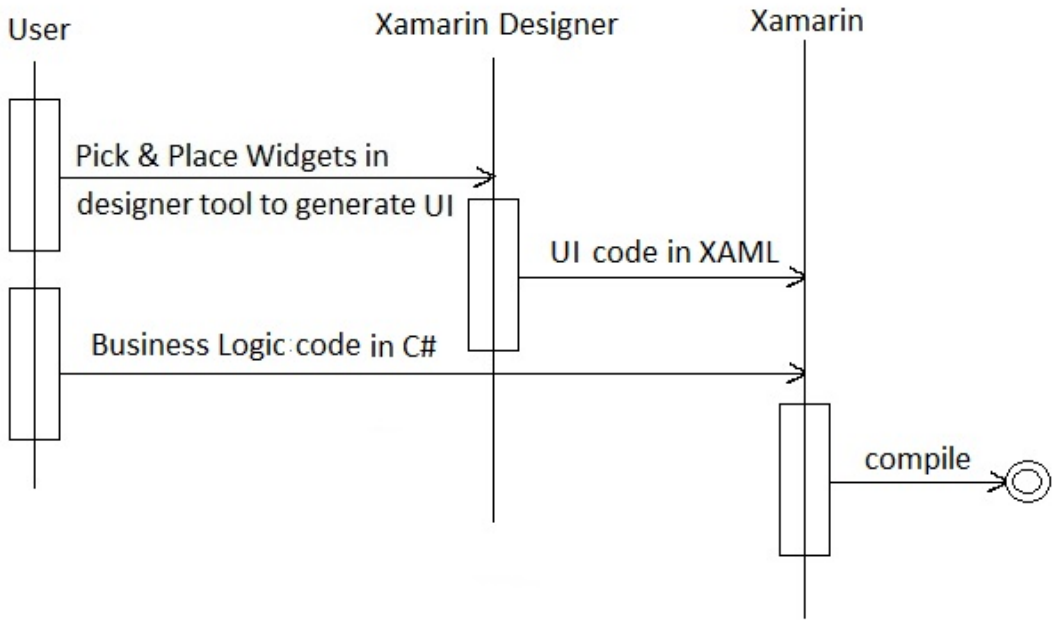


Figure 5.9: Sequence Diagram - Now



CHAPTER 6

SOFTWARE TOOLS USED FOR DEVELOPMENT

6.1 Microsoft Visual Studio

An integrated development environment (IDE) from Microsoft. It is used to develop computer programs, web sites, web apps, web services and mobile apps. Visual Studio uses Microsoft software development platforms such as Windows API, Windows Forms, Windows Presentation Foundation, Windows Store and Microsoft Silverlight. It can produce both native code and managed code.

Visual Studio includes a code editor supporting IntelliSense (a code completion component) as well as code refactoring. The integrated debugger works both as a source-level debugger and a machine-level debugger. Other built-in tools include a code profiler, forms designer for building GUI applications, web designer, class designer, and database schema designer. It accepts plug-ins that enhance the functionality at almost every level.

Visual Studio supports 36 different programming languages and allows the code editor and debugger to support (to varying degrees) nearly any programming language, provided a language-specific service exists. Built-in languages include C, C++, C++/CLI, Visual Basic .NET, C#, F#,JavaScript, TypeScript, XML, XSLT, HTML and CSS. Support for other languages such as Python, Ruby, Node.js, and M among others is available via plug-ins. We have written our contribution in C#. It is a language that uses the .net runtime. Thus the VS was an ideal choice of dev-env for us. Also the basic edition of Visual Studio, the Community edition, is available free of charge. This suited us perfectly.

6.2 MFC (Microsoft foundation Classes)

MFC is an extremely thin object-oriented C++ wrapper for the Windows API. In an MFC program, direct Windows API calls are rarely needed. Instead, programs create

objects from Microsoft Foundation Class classes and call member functions belonging to those objects.MFC is a library that wraps portions of the Windows API in C++ classes, including functionality that enables them to use a default application framework. Classes are defined for many of the handle-managed Windows objects and also for predefined windows and common controls. MFC can be used by linking a static library or by adding the MFC DLL.Since MFC is a wrapper to Windows API, MFC helps to invoke Windows graphical libraries like OpenGL etc. Thus,we prepared our prototype in MFC.

6.3 .NET frameWork and C# managed code

We did our main development in C# which is a managed language and developed mainly through Visual Studio. The C# access the .NET runtime.

6.3.1 .NET framework

.NET is a software framework that provides language interoperability (each language can use code written in other languages) across several programming languages. Programs written for .NET Framework execute in a software environment named Common Language Runtime (CLR), an application virtual machine that provides services such as security, memory management, and exception handling.

Inspite of beginning as a proprietary project, community pressures have force Microsoft to slacken the patenting, licensing, and commercial usage terms for .NET runtime and languages

.NET Framework targets mobile computing, embedded devices, alternative operating systems, and web browser plug-ins. Mono, a .NET family member is available for many operating systems and is customized into popular smartphone operating systems (Android and iOS) and game engines. .NET Core targets the Universal Windows Platform (UWP), and cross-platform and cloud computing workloads. Both Mono and .NET Core are used by Xamarin, which is why the Contribution needs to be written in a CLI compatible language. We chose C#.

6.3.2 C#

C# is a multi-paradigm programming language encompassing strong typing, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines. It was developed by Microsoft within its .NET initiative and later approved as a standard by ECMA and ISO. C# is one of the programming languages designed for the Common Language Infrastructure.

6.4 Github

Git distributed is a version control system, primarily used for source code management. It is aimed at speed, data integrity, and support for distributed, non-linear workflows.Git was created by Linus Torvalds in 2005 for development of the Linux kernel, with other kernel developers contributing to its initial development. Its current maintainer since 2005 is Junio Hamano.Git is free and open source software distributed under the terms of the GNU General Public License version 2.

GitHub is a web-based hosting service for version control using git. Along with version-ing, it provides access control, bug tracking, feature requests, task management, and wikis for every project. As of April 2017, GitHub reports having almost 20 million users and 57 million repositories,[7] making it the largest host of source code in the world.[8]

The Open-source repository of Xamarin is stored on Github. We uploaded our code repository and have requested the Xamarin team to pull it.

CHAPTER 7

IMPLEMENTATION AND TESTING

7.1 Motivation

Every company wants to ride on the wave of "smart devices", by deploying consumer and/or employee-centric software to smart-devices. Thus there is need for proprietary apps, which are tailored for the organization. Thus, during our "project statement hunt mission", we came across many app making statements. However, the condition was that apps that should run on both platforms.(To increase user-base).We searched for tools to make life simpler for us as developers and came across the concept of cross-platform tools. We studied each one to a certain depth and concluded Xamarin to be the better suited for our purpose (because of its native build). But the UI was not developer-friendly as Android studio (to which we were used to). Hence we took up this statement.

7.2 Building a Prototype

We searched on how to make Xamarin contributions and came across a wonderfully written blog post that explained the nitty-gritty details of how community-contributions work. We realised that we ultimately would have to develop in a CLI language. It had a huge learning curve. Thus we broke the task into two parts:

1. Prototyping
2. Contribution building

Proto-typing

We, as a group were well acquainted with the Object Oriented paradigm of programming and had experience with c++. Thus, we decided to do a prototype in MSVC by using MFC. Once we were satisfied with our prototype we moved towards the final implementation - migrating in C# or rewriting from scratch.

Our prototype was a simple one: It included a rectangle that substituted for a design Canvas and a menu to add elements like buttons, text boxes etc. We also focused on three elements: Label, textbox and buttons. Elements, that in short will help make a form. We studied their properties and XAML representations to arrive at a class diagram. The class diagram was the basis for coding. Also we simply considered the absolute layout- for simplicity.

7.2.1 Contribution building

Extending the prototype we added layouts. Adding layouts was a messy affair as layouts are a nested structure i.e. a layout can contain another layout or elements. Thus we had to redesign our system. Simultaneously we had also started working on C# side of the contribution which required adapting the prototyped logic to a managed environment.

7.3 Testing

7.3.1 Static and dynamic Tests

Four official reviews and periodic meets with our guide were our static tests: a test to ensure we were on track.

Deployment of the generated XAML code on the Xamarin environment was our dynamic test. We ensured that every component we developed successfully integrated into he previous code, and gave a satisfactory XAML. The "satisfaction in XAML" was based on the UI the the app showed when deployed on a device through Xamarin. During our Dynamic testing, we also had a round of Fault testing for the component added.

7.3.2 Unit Tests

For both, the prototype and contribution stage, we each did dynamic unit tests on our modules. Also we performed a visual testing, to verify if we really implemented the "WYSIWYG" idea.

7.3.3 Integration Tests

After 2-3 independently developed components, it was time to integrate. Before integration, we White box tested each others code for readability and security concerns like buffer overflows.After merging, we again ran a Functionality check.

CHAPTER 8

FUTURE ENHANCEMENT

This system is designed to generate a markup language text based on the UI designed. The structure of the Web-page is also defined by markup languages and is in essence similar to a mobile UI. Thus as an enhancement, this utility can be used to export web based front end codes like that of Bootstrap, Select etc.

The frameworks for generating mobile and web front ends are huge in number. Each has its own pros and cons. The common negative is that a designer has to learn syntax from scratch if the frame work is o be changed. This designer can be extended to all these tools, and combine their collective best - a appealing UI. We can also design custom themes which will be easy to extend as the tool will be open source.

A Web-Preview can also be given which will help users use this tool online.

CHAPTER 9

USER GUIDE : README FOR OUR PROJECT

This software tool is a GUI designer for Xamarin.forms. It enables you to drag and drop widgets onto a canvas, to generate your app layout. The XAML, representing the layout is generated in the backend, when you hot the generate button.

9.1 To create a simple form:

1. Select a layout from the layouts pane/ menu. **In this designer, every element is expected to be contained in a layout. Thus unless you select a parent layout, you will not be able to add any children.** For Example: if absolute layout is selected, all children of the parent i.e. other nested layouts and elements will be contained in this parent layout.

2. Now double click on a label from the elements pane. It will appear on the top-left of your canvas. If you double click this element, you will get a properties pop-up. From here you can control the attributes like border thickness, color, caption etc of the label. A single click on the label will select it. If you now drag he mouse, the label will be dragged to the desired location. You can resize the label from the eight handles that appear when the label is selected. Now place the label at the required place, give it a caption and set its attributes.

3. Now create a text box. The controls of the textbox, or for that matter of all elements are similar to that of the label. Thus drag it to the appropriate place. It will not have a caption.

4. Now create a button, and place it in the approximate center on the canvas.Caption it as... um.. say "Submit".

5. This creates our UI. Now hit generate. This will generate XAML in the background, in the XAML1.txt file located on your desktop.

6. Since our contribution is not yet accepted by Xamarin, we copy and paste this code into the Xamarin environment.

7. Now connect your phone and run the app. (Read Xamarin guide before this step, for additional requirements like android SDK to run Xamarin).You shold see a UI similar to the one you created on the screen.

REFERENCES