

CHALMERS



Extending web applications to mobile platforms: a software engineering view

Master of Science Thesis in the Programme Software Engineering and Technology

**CAN PESKERSOY
MALEEKANYA TANINTARA-ART**

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden, January 2013

The Authors grant to Chalmers University of Technology and University of Gothenburg the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet.

The Authors warrant that they are the authors of the Work, and warrant that the Work does not contain text, pictures or other material that violates copyright law.

The Authors shall, when transferring the rights of the Work to a third party (for example a publisher or a company), inform the third party about this agreement. If the Authors have signed a copyright agreement with a third party regarding the Work, the Authors warrant hereby that they have obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

Customer Relationship Management application for mobile platforms

CAN PESKERSOY
MALEEKANYA TANINTARA-ART

© CAN PESKERSOY, 2012.
© MALEEKANYA TANINTARA-ART, 2012.

Supervisor: Sally A. McKee

Examiner: Sven-Arne Andréasson

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

Cover:
QBIS CRM application logo
©QLogic AB, 2012.

Department of Computer Science and Engineering
Göteborg, Sweden

Acknowledgments

First and foremost, we would like to thank Andrew Johnston for providing continuous support and supervision for our thesis. Also we would like to express our deepest gratitude to our supervisor, Sally A. McKee, for her valuable advice and suggestions which have provided guidance throughout the writing of thesis.

We are indebted to our families for their endless support and encouragement during all academic life

Last, but not least, we thank all the people who helped us to succeed with our thesis, especially employees of QLogic AB for being hospitable and kindly.

Abstract

The purpose of this master thesis is to investigate the problems and solutions for a company who wants to expand their applications onto mobiles. It is important for the company if they can find the faster way to develop the same system onto mobile apps, what artifacts can be used in common and what can be reused. Even smartphones have ability to work almost like PC but they also have many limitations such as screen size, limited resources like RAM, disk, and battery. How can we compromise the full data context with those limitations? In this master thesis, we have studied all perspectives in the software engineering area and come up with problems and solutions through the real implementation of mobile apps. We have learned that a project plan and organization involvement at the beginning is crucial because the decision on the target market, software architecture, offline/ online necessity affects a lot later on in the implementation phase. We have found out that a large extent on the reuse is based on the software architecture and design pattern. The tradeoff between using the same design pattern and different mobile frameworks' best fits design patterns is a one interesting future study topic. A testing coverage is also one topic that we have found out needed because a mobile app needs more considerations than a traditional application.

Glossary

CRM: Customer Relation Management

SDK: Software Development Kit

API: Application Programming Interface

RUP: Rational Unified Process

UML: Unified Modeling Language

ADT Plugin: Android Development Tools Plugin

SOA: Service Oriented Architecture

WSDL: Web Services Descriptive Languages

SOAP: Simple Object Access Protocol

REST: REpresentational State Transfer

MVC: Model-View-Controller

GPS: Global Positioning System

OS: Operation System

UI: User Interface

HTTP: Hypertext Transfer Protocol

XML: Extensible Markup Language

GUI: Graphic User Interface

IDE: Integrated Drive Electronics

Table of Contents

ACKNOWLEDGMENTS	3
ABSTRACT.....	4
GLOSSARY	5
TABLE OF CONTENTS.....	6
1. INTRODUCTION	8
2. BACKGROUND.....	10
2.1. CUSTOMER RELATIONSHIP MANAGEMENT (CRM).....	10
2.2. QBIS CRM	11
2.3. MOBILE PLATFORMS.....	11
2.3.1. IOS	11
2.3.1.1. Architecture.....	12
2.3.1.2. Software Development Kit	13
2.3.1.3. Gestures	13
2.3.2. ANDROID	15
2.3.2.1. Architecture.....	15
2.3.2.2. Gestures	16
2.3.2.3. Platform, Tools.....	18
2.3.2.4. Version History	18
3. METHODS & ANALYSIS	20
3.1. DEVELOPMENT PROCESS.....	20
3.2. REQUIREMENT ELICITATION TECHNIQUES.....	21
3.3. TESTING METHODS	22
3.4.1 UNIT TESTING	22
3.4.2 SYSTEM TESTING	22
3.4.3 ACCEPTANCE TESTING	23
3.4. ARCHITECTURE	23
3.5.1 SERVICE ORIENTED ARCHITECTURE	23
3.5.2 SOAP VS. REST	24
3.5.3 DESIGN PATTERNS.....	25
4. DESIGN & IMPLEMENTATION	27
4.1. ITERATION 1: INCEPTION PHASE.....	28
4.1.1. WORLD MARKET RESEARCH.....	28
4.1.2. SWEDEN MARKET RESEARCH	29
4.1.3. ANDROID VERISON SELECTION FOR THE BEST MARKET RESULT	30
4.1.4. IPHONE VERSION SELECTION FOR THE BEST MARKET RESULT	31
4.2. ITERATION 2: ELABORATION PHASE.....	32
4.2.1. REUSABILITY	32

4.2.1.1. Project Overview.....	33
4.2.1.2. UML Reuse.....	34
4.2.1.3. Database Reuse.....	35
4.2.2. SYSTEM ARCHITECTURE AND DESIGN PATTERNS.....	37
4.2.2.1. QBIS Architecture and Communications.....	37
4.2.2.2. Web Service Reuse.....	39
4.2.2.3. Best fits Design Patterns in iPhone and Android.....	39
4.2.2.4. The Model-View-Controller Design Pattern (MVC).....	40
4.2.2.5. The Model-View-Presenter Design Pattern (MVP).....	41
4.2.3. USER EXPERIENCE (GUI).....	42
4.2.3.1. Comparing Attributes with Competitors.....	42
4.2.3.2. Interviewing.....	43
4.2.3.3. Prototyping.....	43
4.2.4. FINAL PRODUCTS.....	46
4.3. ITERATION 3: CONSTRUCTION PHASE.....	49
4.3.1. SYNCHRONIZATION.....	49
4.3.2. UNIT TESTING.....	51
4.3.2.1. SenTestingKit (OCUnit).....	51
4.3.2.2. Android Testing Framework (JUnit).....	52
4.4. ITERATION 4: TRANSITION PHASE.....	53
4.4.1. USER VALIDATION (APPLYING ACCEPTANCE TESTING).....	54
4.4.2. DISTRIBUTE THE APPLICATIONS.....	55
4.4.2.1. App Store.....	55
4.4.2.2. Android Markets.....	56
4.4.3. ONLINE HELP.....	56
<u>5. DISCUSSION.....</u>	<u>57</u>
<u>6. CONCLUSION.....</u>	<u>60</u>
6.1. MOBILE APPLICATION DEVELOPMENT.....	60
6.2. FUTURE WORK.....	61
<u>7. REFERENCES.....</u>	<u>62</u>

1. Introduction

In order to increase sales, many companies are choosing to extend their desktop applications to the Internet. Also, the increasing popularity of cloud computing services in recent years has encouraged the software companies to provide all computations, software applications, data access, and data management to customers such that the customers have to pay only for the services they use. We call this kind of service web-hosted software solutions, which are often referred to as Software as a Service (SaaS). [Pragyaan_IT_June2011]

QLogic AB, the company we are doing master thesis with, has also seen this opportunity. They have marketed QBIS, a suite of intelligent web-based business modules online for many modules, for example Time Reporting, Project Management, CRM (Customer Relationship Management) and Service Desk since 2001.

Along with growing cloud computing and web-hosted software, the introduction of smartphones platforms like Apple iPhone or Google's Android has also caught attention of SaaS companies, including QLogic AB, since the smartphones are capable of general-purpose computing (we can call them the next PCs), so they are able to provide solutions like web-hosting does. [Pro-Android] So, QLogic AB decided to take a step toward this chance as well. Considering the smartphones' market share and the usage of QBIS target customers, QLogic AB chose to start extending their QBIS web suite onto smartphones with a CRM module.

There is quite a big opportunity for many companies who are SaaS providers (or becoming SaaS providers) to gain more profits by extending their application to smartphones. We as software engineer students have dedicated this master thesis to finding out solutions to guide the companies in deciding what kind of considerations should be taken into account and also what possible solutions exist before they start to extend their applications onto mobiles. We address the following questions:

Q1) What kind of problems should be considered when companies want to extend their existing application to mobile client applications (from a software engineering perspective)?

Q2) Considering the whole software development process, what significant artifacts can be used in common when it comes to developing one system on two platforms (in order to reduce cost of implementation)?

Q3) What constraints should be used to decide the graphic user interface context needed for light weight mobile application?

Q4) What kind of methods and what considerations need to be taken into account for synchronization to serve the needs for data correctness and efficiency?

Q5) How much test coverage should be considered to fulfill all user requirements and sufficient in target platforms?

In this thesis, we have limited our research to the CRM system, and also to only two largest mobile platforms (the iPhone and Android platforms).

2. Background

In this section, we explain what is CRM and what kind of advantages can be taken using the system as a mobile application. Then we talk about the QBIS CRM which is a web-based solution developed by QLogic AB. Finally we provide technical information about two popular mobile platforms, Apple iOS and Google Android, with explaining their specifications, architectures and gestures.

2.1. Customer Relationship Management (CRM)

Customer relationship management or CRM is a marketing strategy for organizations or individual customers who plan business operations to understand their needs and provide rapid solutions. [Top CRM] To succeed in competitive market, it is really important to eliminate obstacles as fast as possible. Instead of building a new customer relationship each time, CRM mostly aim to retain current ones to have long-term business success. There is a set of stages that need to be applied when company put the CRM strategy on practice:

- **Customer Selection:** Targeting high value and low attrition-risk prospects.
- **Customer Acquisition:** Determining the most profitable customers
- **Customer Retention:** Ensuring customer loyalty with increasing satisfaction of customer.
- **Customer Enhancements:** Understanding behaviors of existing customer and providing the most profitable opportunities depending on their needs.

Information technologies play a fundamental role in the retention and development process of customer relations. Especially computer-aided software applications assist to make more profitable growth for companies by supporting flexibility. With technological innovations most of software companies offer different kinds of CRM tools with extensive features. However most of the users incur different kinds of problems that software providers have to deal with. Several complaints are caused by lack of usability such that customers do not feel comfortable while using such tools. They were unnecessarily complex and sophisticated. Consequently, software companies aim to overcome these challenges in smart way. One of the solutions could be keeping all kinds of customer-related information in a complex back-end database while presenting significant information within a simple user-friendly interface.

Mobile applications are rapid, supportive, and especially light solutions that allow users --- salesmen, service staff, and support teams --- to access and modify their customer records while they are away from computers (even when there is no Internet connection). Using different kinds of mobile devices, users can retrieve customer-specific information from central servers and can manage their activities, opportunities or tasks as much as they like. [Mobile CRM]

The advantage of having CRM application on mobile device is an improvement in productivity and simplicity. User-friendly mobile applications significantly decrease the time within which the user is able to perform operations. Moreover, an internet connection is not always required. Users can access and modify customer data locally and then synchronize the application later.

2.2. QBIS CRM

QBIS CRM is a web-based hosted CRM solution that helps users in managing their customer information and sales activities in a single system. Because of being web-based hosted users can use the web application from any location, which means that it needs neither to be installed on their computers nor that they need to buy additional hardware. This approach is compatible with most of the leading web browsers, including Internet Explorer, Google Chrome, Mozilla Firefox so that it can be used by almost any internet user.

All functionalities in the suite are easy to use for different business roles so that salesmen can manage their accounts, tasks, and prospects while executives trace their progress. But the most important benefits of the QBIS CRM solution lie in management layer of business structure. It will give enhanced support to each user for evaluating their activities, which helps to improve productivity and effectiveness. It also provides a robust security solution in that all critical customer or account information is stored confidentially.

2.3. Mobile Platforms

The market of mobile devices grows very fast. Manufactures have to deal with various demands from customers. Most of them provide different features such as GPS navigations, high resolutions, enhanced network connectivity, or speech recognition. However each platform has also different kinds of limitations, such as battery consumption, low-power CPUs and device memory. [Comparison Mobile] These pros and cons lead users to choose best practices according to their preferences.

Mobile platforms have different operating systems that support various application frameworks and programming languages. [Comparison Mobile] Android and iOS are the two most common operating systems, each having their own native languages and integrated development environments. Although each platform has unique and different features that are supported by their own framework, there are some other third-party engines for multi-platform application development today. But they are not perfect solutions in most cases, since they have limited capabilities because of lack of support. Therefore, we focus on these two operating systems with their own development environment in our thesis.

2.3.1. iOS

Apple released the first version of iOS in 2007 (initially named iPhone OS). iPhone and iPod Touch are the first compatible devices for the operating system. Unlike

other manufacturers, Apple does not offer any open-source license for their developers to modify the source code for their own customization or install it on other mobile devices. [Apple MobileHIG]

2.3.1.1. Architecture

The iOS architecture consists of four software layers that provide different frameworks to be used in the development phase of applications. Each layer has an abstract interface that makes communication easy and consistent among them.

Figure 1 shows the representation level of each layers and how the application use hardware components of the device.

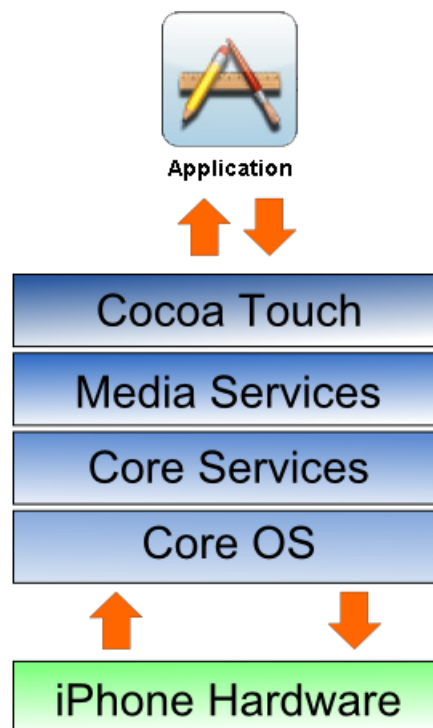


Figure 1 – Layers of iOS stack

The Cocoa Touch layer is located on top of the other layers comprising the development frameworks to be used by developers. Multimedia services (such as audio, video, animation and graphics) are provided by the Media Service Layer. Core Services provides system services to upper layers either directly or indirectly. iCloud is a good example for this layer introduced with iOS 5. Users are able to store personal data on the central server and access them from other Apple devices. [Apple Technology] On the bottom of the iOS stack Core OS layer sits with low-level features. The user has to deal with the layer if an application will communicate with device built-in hardware or accessories.

2.3.1.2. Software Development Kit



The iOS SDK (Software Development Kit) has been mainly written in Objective-C, which is also the native development language for iPhone development. Since Objective-C is one of the C family languages and most of the parts are derived from C language, some C libraries are still available in the SDK. Objective-C is also an object-oriented language that supports message passing model. Objects are able to send or receive a message to/from other objects.

There are also other key tools included in SDK. XCode is the one of the main development tools that enables user to design and develop their applications. Interface Builder is a built-in interface design tool, developers able to create graphical interface of the application. Instruments is another tool to analyze performance of the application in terms of memory management, CPU usage and data storage. Users can also track memory leaks, file I/O operations, and network activity and traffic.

2.3.1.3. Gestures

The communication of the interface is accomplished through several interactions: touching (single-multi touch) the screen directly using interface control elements (buttons, sliders, switches) and also pressing physical buttons. The iOS also supports multi-touching gestures to perform various actions. There are four main gestures on interface layer which are also show in **Table 1**:

Table 1 – Main gestures of iOS devices.

	
Tap: Users touch the screen to open an application or interact with any controller such as button, editable fields.	Double Tap: Users touch the screen twice to copy some text or make a zoom in/out.



Pinch: Users place the two fingers (commonly thumb and a finger) on the screen and move them apart without lifting them from the screen. Pinch is mostly used for zoom in.

Reverse Pinch: Users place the two fingers on the screen and move them near without lifting them from the screen. Reverse pinch mostly used for zoom out.



Rotate: Users place the two fingers (commonly thumb and a finger) on the screen and move their fingers in clockwise or anti-clockwise direction to rotate the images.

Tap and Hold: Users touch the screen for a while without moving finger. It is commonly used to arrange the order of menu items or paste clipboard text into tapped field.

2.3.2. Android

Google acquired the startup company Android Inc. in 2005 to start the development of the Android Platform, and in late 2007, a group of industry leaders (including Motorola, Samsung, Sony Ericsson, Vodafone, and Google) came together to form the Open Handset Alliance, the goal of which is to innovate rapidly and respond better to consumer needs, and the first key outcome of which was the Android Platform. [Pro-Android]

2.3.2.1. Architecture

The Android SDK was issued first in November 2007 and updated in October 2008, Google has made the source code of the Android Platform available under Apache's open source license. [Pro-Android]

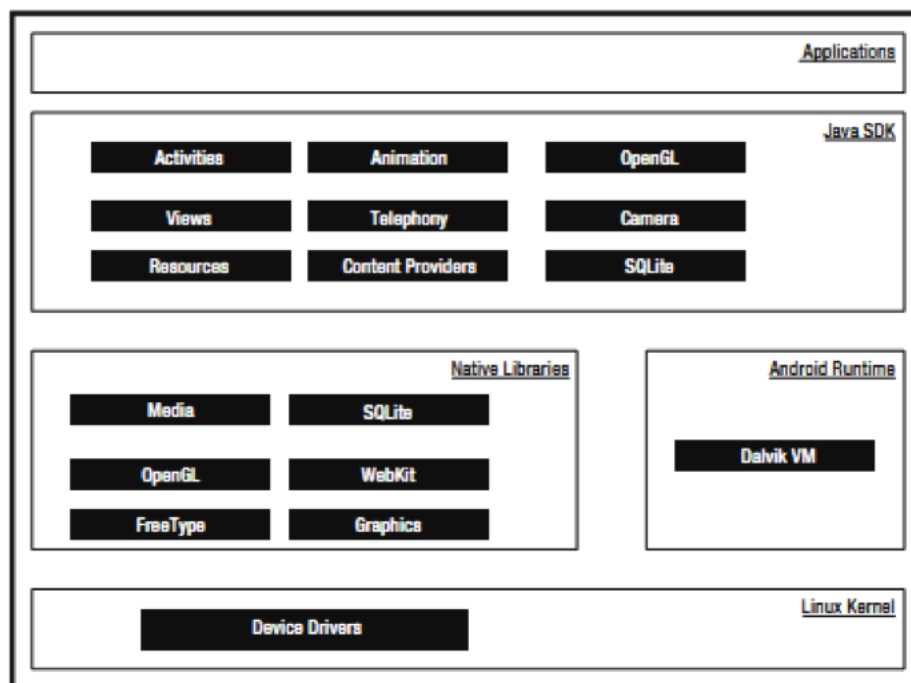


Figure 2 – Architecture of Android platforms.

At the core of the Android Platform is Linux kernel version 2.6, which is responsible for device drivers, resource access, power management, etc. [Pro-Android] On top of the kernel, there are C/C++ libraries such as WebKit, OpenGL, FreeType, Secure Sockets Layer (SSL), the C runtime library (libc), Media, and SQLite. [Pro-Android]

The WebKit library is used for browser support, and this is actually the same library that supports Apple Safari and Google Chrome. The FreeType library is for supporting fonts. SQLite is a relational database available either pre-installed on devices or as an independent open source which can be downloaded and used. [Pro-Android] The gateway to access Android Platform is Dalvik VM, which most application frameworks use for accessing the core libraries.

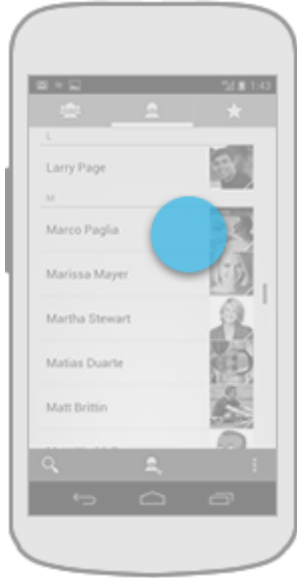

Developers create end-user applications on mobile device (for example Contacts, Phone, Home, and Browser) on top of the Java API which consists of a main set of Java libraries that address telephony, resources, UI, locations, content providers, and package managers. [Pro-Android]

In addition to the general APIs, Android depends on hardware APIs to support wireless infrastructures such as WiFi, 3G, EDGE, Bluetooth, and Global System for Mobile Communication (GSM) telephony. [Pro-Android]

2.3.2.2. Gestures

Table 2 illustrates the core gesture set that Android supports.

Table 2 – Seven different gestures for Android based devices. [Andriod Gestures]

	
<p>Touch: User presses and lift will trigger the selected item to show the default functionality.</p>	<p>Long press: User presses wait and lift will allow the user to select one or more items in a view.</p>



Swipe: User presses then move and lift will scroll over content or navigate between views in the same page's hierarchy.




Drag: User long presses then move and then lift will rearrange data within a view, or moves data into a container such as folders.



Double touch: User two touches quickly will zoom into content and can also be used in a text selection.



Pinch open: User two finger presses then move outwards and then lift will zoom into content.

	
<p>Pinch close: User two finger presses then move outwards and then lift will zoom out the content.</p>	

2.3.2.3. Platform, Tools

The Android SDK (Software Development Kit) includes several tools and utilities to help develop; create, test, and debug mobile applications for the Android platform. The tools are categorized into two groups: SDK tools and platform tools. SDK tools are platform-independent, but platform tools are customized to support features for only the latest Android platform.

The most important and frequently used SDK tools include the Android SDK Manager (Android SDK), the AVD Manager (Android ABD), the emulator (Emulator), the Dalvik Debug Monitor Server (DDMS), and sqlite3. [Android SDK Tools]

2.3.2.4. Version History

There are a lot of updates in Android version since its first release. The updates are typically about new features adding and bugs fixing. and has seen a number of updates since its original release. [Android Versions]

The released versions of Android are:

- **2.0/2.1 (Eclair)**, the user interface modified, HTML5 introduced and Exchange ActiveSync 2.5 supported.
- **2.2 (Froyo)**, speed improvements introduced with JIT(Just-In-Time) optimization and the Chrome V8 JavaScript engine, Wi-Fi hotspot tethering and Adobe Flash support added.

- **2.3 (Gingerbread)**, the user interface refined, the soft keyboard improved and the feature copy/paste improved, and the feature to support for Near Field Communication added.
- **3.0 (Honeycomb)**, a tablet-oriented and many new user interface features released, multicore processors supported, hardware acceleration for graphics increased.
- **4.0 (Ice-cream sandwich)**, a combination of Gingerbread and Honeycomb into a "cohesive whole,"

3. Methods & Analysis

In this section, we introduce our development model RUP that we use as a guideline of our project. We explain the reasons why we prefer the model among the others. Then we present the techniques that we used to elucidate the requirements. After that, we explain how can we ensure the quality of the mobile applications, especially what kind of testing methods we apply to them. We also share the experiences that help to developers to understand restrictions of mobile application in a better way. Finally, we explain the overall architecture of the system with its sub-components and design patterns.

3.1. Development Process

There are several models that we can follow during the application development process. Each model has its own specification among which developers have to choose for their models according to several constraints. An iterative and incremental development process is the most adaptable discipline, and so we decided to follow one such process in our work.

The Rational Unified Process (RUP) is a comprehensive process development framework that has development blocks that can be adaptable and includes some the elements that can be selectable by developers. As seen in **Figure 3**, the development process is shown in a two-dimensional graph that has several development disciplines, where each discipline is divided into four consecutive phases. Disciplines can be varied based on comprehension of the project. Milestones sit at the end of each phase for which critical decisions have been made regarding key objectives that have or have not been accomplished.

- **Inception Phase:** Develops the business case and delimiters of the project are determined in this phase. Also success factors, market potential and risk assessments are made in this phase. [Rational]
- **Elaboration Phase:** The goal of this phase is to analyze the domain of the problem and eliminate higher risk elements of the project. Also development plan of the overall project is scheduled and most of the use-case models are developed in this step. [Rational] In addition, a software requirements specification document is written.
- **Construction Phase:** The main development of the software system occurs in this phase. In comprehensive projects there could be several consecutive iterations depending on requirements. All components of the system are verified with test cases. The most important outcome is the beta release of the system.

- **Transition Phase:** The developed system is converted into a software product, which should be understandable and verifiable. All other project components are summed up into final artifacts.

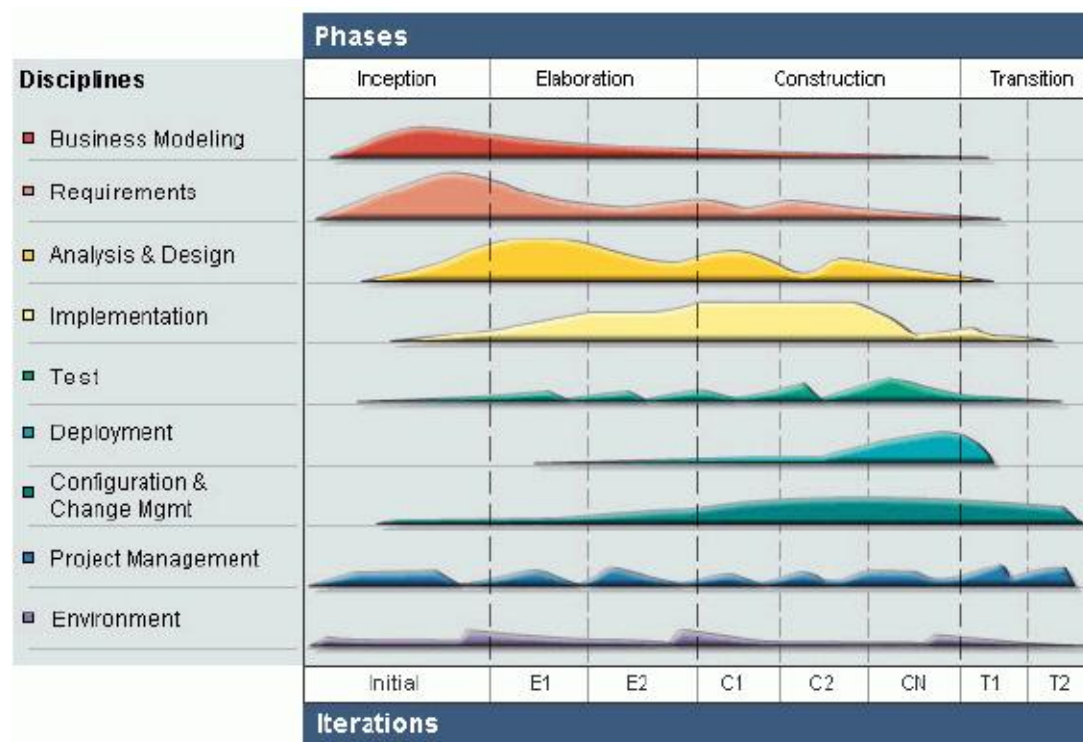


Figure 3 – RUP hump chart indicates the effort of phases over specific discipline.

One of the best advantages of the RUP is iterative development that developer can re-adapt their system according to additional requirements. Given complicated and uncertain requirements sometimes it is not possible to provide the entire solution. [Rational] With iterative approach, developer can have a good understanding and provide reliable design decisions that are considered in successive iterations without effecting project structure.

The model also presents best practices to use UML model effectively in development process. UML model allows developer to form the business model, combine requirements in common structure, analyze and design the system behaviors with visual models.

3.2. Requirement Elicitation Techniques

Two types of requirement elicitation techniques have been selected for this thesis. In mobile applications, an obvious limitation which we all can see is the smaller screen size than desktop applications, and also the limited memory size. The latter means that having all data stored in the same way as in the QBIS web application might not be desirable. In order to have just enough information from the CRM system, the requirement elicitation has to be used as a tool to select the right information for CRM users given the limited resources of mobile devices.

At the beginning of the project, the study of the existing system QBIS-CRM was performed, comparing it to other products in the market. In addition to interviewing business analysts, we also solicited feedback from a salesperson that is familiar with the system and user's requirements. Based on these two activities, we selected all information needed for the application. We also exploited prototyping techniques because of the unfamiliar screen size and actions on mobile screens. Such early feedback is needed in order to adjust the captions, information, and screen styles for the users' look and feel before starting the implementation. [RE_NESEIBEH]

3.3. Testing Methods

Mobile software system development has many aspects differ from traditional software development, as mobile software has special requirement and constraints such as wireless communication issues, special privacy and customizability needs, so software produced for mobile environments should be at a high level of quality in order to operate properly on different kind of mobile devices. [Designing Agile for Mobile]

Testing is one of the key components used to ensure the quality of software applications. Testing increases the reliability of the software and prevents potential failure. Mobile applications bring new challenges, such as connectivity, data integrity, resolution, performance, and security. All these factors should be considered when the testers describe their test strategies to evaluate functionalities. In our thesis we applied both white-box and black-box testing from unit level to acceptance level.

3.4.1 Unit Testing

Unit testing is a crucial method for examining the system at the source code level. Small pieces or building blocks are tested with unit testing before being integrated into large modules. Detecting and removing defects with unit testing is easy and more cost effective if it is compared with other level.

In quality perspective, Test-driven Development (TDD) is one of the highest importance characteristics of iterative-incremental development. [Designing Agile for Mobile] TDD concept is to use test cases to drive the design and implementation by increment the simple small pieces generalized to complex software system. [TDD] Test cases consist of several unit tests that automate to execution and benchmark predicted outcomes and actual outcomes. Although TDD seems to be suitable to the project, our inexperience in field and time constraints force us to keep TDD out of the project scope.

3.4.2 System Testing

System Testing is an integration testing of separate modules or system components to ensure communication as intended. A test plan is prepared with requirement specifications that it covers assessment of all related components in particular sections. In our thesis, the plan is included in acceptance testing process which are held by other QBIS developers. Since they are already familiar with the system, various scenarios have been applied depending on the test plan.

3.4.3 Acceptance Testing

Acceptance Testing helps to ensure the requirement specifications are completely implemented into software product. Generally, test procedures are applied by end users which mostly interact with the system in future. Since it is a black box testing, they are only responsible for system inputs and their expected results.

The difficulty in developing mobile applications can be a major impact from the different style of user interaction and smaller display. While the traditional interface style that we are familiar with like Microsoft Windows and Apple's iOS is WIMP (Windows, Icons, Menu, Pointer), mobile paradigm is based on touch, widgets, physical motion, and keyboards (physical and virtual). [Software Engineering Issues for Mobile App]

One common thing from users that user interface designer should be concerned about is in each particular native applications like Android or iPhone, they, especially on touch-screen devices users, expect to use the platform's standard set of gestures. So, it would be interest for the designer to design common "look and feel" in one platform which each of those platforms has also provided their own UI libraries and guidelines. [Software Engineering Issues for Mobile App]

With the limited screen space, the challenge is to make the best usage out of it. We might think that we want to take full range of functionalities like in a traditional web application, but actually users are seeking only to quickly complete a simple task on their mobile. So, the design must be completed by including most used functions, attributes while the screen must be used effectively and follow mobile user interface paradigm. [Software Engineering Issues for Mobile App] So, the question is how we do that and also how to measure it?

3.4. Architecture

The system's software architecture is a set of structure in which explain and reason about the system. The software architecture is comprised of software elements, their relations and their properties. [Doc Software Arch]

The term software architecture also means documentation used to communicate between stakeholders. The early decision making on software architecture in high-level design is crucial for software project. It will allow and ease the further development and changing requirements, such as reuse of the components and patterns in the same project or between projects. [Sof Arch In Practice]

To identify elements, their relation and their properties systematically, knowing design pattern fits first is probably the fine and elegant step.

3.5.1 Service Oriented Architecture

Service oriented architecture (SOA) is widely used web standard with providing collection of protocols for constructing communication bridge between two different software architectures. The implementation of SOA is done with web services which

isolate the infrastructure of server and build up a new flexible interface. [Mobile Web] With the strength of web services, different functionalities are grouped into several parts that clients can consume one or more of them at the same time. Since they are located at the end point of the server communication, they should be self-describing and discoverable that clients need to know how to consume them and what kind of response will be returned. All these functionalities, protocols, bindings and message encodings are described with WSDL. It provides XML-based file which clients read the file for retrieving operations and call one of them.

There are also two traditional web service communication models which are: Simple Object Access Protocol (SOAP) and REpresentational State Transfer (REST). SOAP is a XML based protocol for exchanging information via HTTP. It provides messaging framework that it can be implemented by web services. REST is an architectural style, which is a lightweight alternative of the SOAP. The information message is also transferred via HTTP.

3.5.2 SOAP vs. REST

Although both services use HTTP standard as a communication protocol, they have different kind philosophy inside. SOAP uses XML-based structures such as XML Namespaces and XML Schemas whereas REST deals pre-defined nouns and verbs to supports some set of operations. For example, HTTP GET method is used for retrieving a resource from server. The resource could be representation of any meaningful concept. Response of the server is mostly in either XML or JSON format but sometimes it can be HTML page as well. The other supported HTTP operations are PUT, POST and DELETE.

On the other hand SOAP message has more complex format that is combined in a specific namespace and named SOAP-Envelop. It consists of two parts, header and body, like other any message format. Although SOAP-Header is an optional part of the message, it can contain significant information such as authentication or encoding style. SOAP-Body contains the actual message that is required and also the name of the requested method, which is already defined in WSDL file with its parameters. As you see in **Table 3**, messaging structure of REST is much simpler then SOAP. REST request can be sent within a few lines.

Table 3 – Request message formats for both web services.

<p>SOAP Example</p>	<pre>POST /WSDomain.wsName.asmx HTTP/1.1 Host: address.com Content-Type: text/xml; charset=utf-8 Content-Length: length SoapAction: "https://address.com/WSDomain/wsName/GetExample" <?xml version="1.0" encoding="utf-8"?> <soap:Envelope</pre>
--------------------------------	--

	<pre> xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelop/"> <soap:Body xmlns:m="http://www.example.org/stock"> <GetExample xmlns="https://address.com/WSDomain/wsName"> <Paramater1>Value1</Parameter1> </GetExample> </soap:Body> </soap:Envelope> </pre>
REST Example	<pre> GET /GetExample/IBM HTTP/1.1 Host: address.com Accept: text/xml Accept-Charset: utf-8 </pre>

Although REST is much simpler approach to implement and consume, SOAP is most commonly used among the software companies. Because REST has not been standardized yet. Also the company had already implemented the SOAP services into their systems. Besides we had limited time to introduce a new web services. So we decided to keep the web service implementation out of the scope of the project and continue with existing web services.

3.5.3 Design Patterns

A design pattern has become a useful tool for architecture, software engineering and development. It is an abstraction template for a design to solve a general and recurring problem in software's particular context. [Apple MVC]

The design pattern depends on themes, principles or rule of thumb for constructing object-oriented systems that influences design patterns. The examples of design principles are such as encapsulate concept, interface, and implementation. [Apple iOS Dev]

If a system needs to have a flexibility property, it is necessary to design the system as loosely coupling and eliminate dependencies as much as possible. The design especially with interfaces that vary a lot should be encapsulated, also not tied to any other parts. So when we want to alter or extend those variable parts, it will be a lot easier to implement. Also, the change will not affect another parts and the whole system. Another benefit of the design patterns is that it will make a program more elegant, more efficiency and having fewer lines of code compared to a program that is not designed based on the design pattern concept. [Apple iOS Dev]

One of the research questions of this master thesis is about how we can reuse artifacts which including components/ source code among three platforms: web

application, iPhone and Android. If the design pattern has been fully considered at the beginning of the project, we would be able to save a lot of implementation time then also a future implementation. Unfortunately we could not because of our implementation time was limited. However, we believe that the research on the design pattern for reusability will be useful for companies or people who come to the same situation. So we have dedicated our time on it after the implementation and it will be explained in details in later sections.

4. Design & Implementation

In this section, we provide detailed information about design and implementation process. We explain how we decide to choose suitable versions for mobile platforms based on markets research. We also mention about reusable artifacts that we used in our thesis. Then we discuss requirement gathering methods to provide best user experience. After that, we have a section about design patterns which help us to design the structure of the applications. We clarify the communication between client and server sides. Finally we introduce validation and verification methods to meet user expectation.

The **Figure 4** shows a roadmap of how we implemented the project. We have implemented the project according to four phases of RUP. As we have mentioned in the previous section and from **Figure 3**; it shows that RUP is a disciplined approach which assigns and manages tasks and responsibilities into phases in development organization. [The RUP]

We have chosen this process because RUP elaborates each discipline which we can examine all possible reusable artifacts in every phase in the project. Moreover, the extending mobile apps from web applications is a project which is not quite traditional software project and since RUP is an iterative and incremental development process which is quite adaptable discipline; so it is rather been used for our project rather than other methods.

RUP is divided into four consecutive phases. Disciplines can be varied based on comprehension of the project. As well as our project, one discipline have been done iteratively more than one phase but we have put the discipline artifacts into the phase that they have been mainly working on.

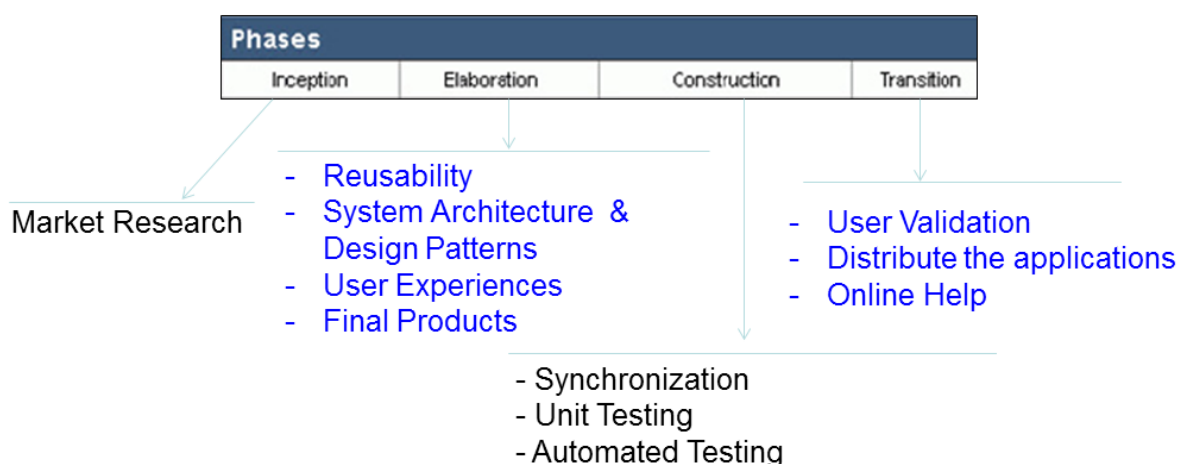


Figure 4 – The roadmap of the project implementation

- **Inception Phase:** The objective of this phase is to scope costing and business case. We have done a market study based on QLogic AB customers in this phase.
- **Elaboration Phase:** The problem domain analysis is made in this phase; use case diagram, UML diagram, software user interface design, and architecture design.
- **Construction Phase:** In this phase, we built the project and show the interfaces example in the elaboration phase. In this section of the report, we show how we implemented regarding synchronization for the sake of having offline/online apps. Also, we explain on how we implement testing, how to ensure the test is coverage for the whole project.
- **Transition Phase:** In order to transit the system from development to production, there is a way that we checked the quality before submitting it. We describe it in this section of the report.

4.1. Iteration 1: Inception Phase

We did a market study in this inception phase to meet the phase's objective; to scope costing and business case. We broke down the market study into 2 parts; international market and Swedish market. The results from the market study show why the QLogic AB has chosen to develop iPhone and Android applications instead of Windows Mobile or other mobile platforms.

It is important for SaaS companies to study on their target customers before starting a software implementation. The more they know about the market, the more they can gain profits from it. After the market study, we did a study on Android and iPhone platforms to see which version has the highest number of users. The versions that the company selected would affect the number of users using the application and certainly their profits. Once the versions selected, tools for the implementation were chosen accordingly.

4.1.1. World Market Research

According to Gartner sales research [Gartner], there were about 110 million smartphones in use in the world in second quarter of 2011. Android got the first place with 43.4%, Symbian follows in the second place with 22.1% and iOS 18.2% in the third place. The total number of smartphones' market shares is presented in **Table 4**. The table provides a comparison between the market share in a second quarter of year 2010 and 2011.

Table 4 – The market share of the smartphones operating system in the second quarter of 2010 and 2011.

Operating System	2Q11 Units	2Q11 Market Share (%)	2Q10 Units	2Q10 Market Share (%)
Android	46,775.9	43.4	10,652.7	17.2
Symbian	23,853.2	22.1	25,386.8	40.9
iOS	19,628.8	18.2	8,743.0	14.1
Research In Motion	12,652.3	11.7	11,628.8	18.7
Bada	2,055.8	1.9	577.0	0.9
Microsoft	1,723.8	1.6	3,058.8	4.9
Others	1,050.6	1.0	2,010.9	3.2
Total	107,740.4	100.0	62,058.1	100.0

4.1.2. Sweden Market Research

According to the market survey from Metro Business (2011), there is a huge debate between Android and iOS in this country. Each operation system has an equal market share which is 41%, leaving only 18% to Symbian. [Metro]

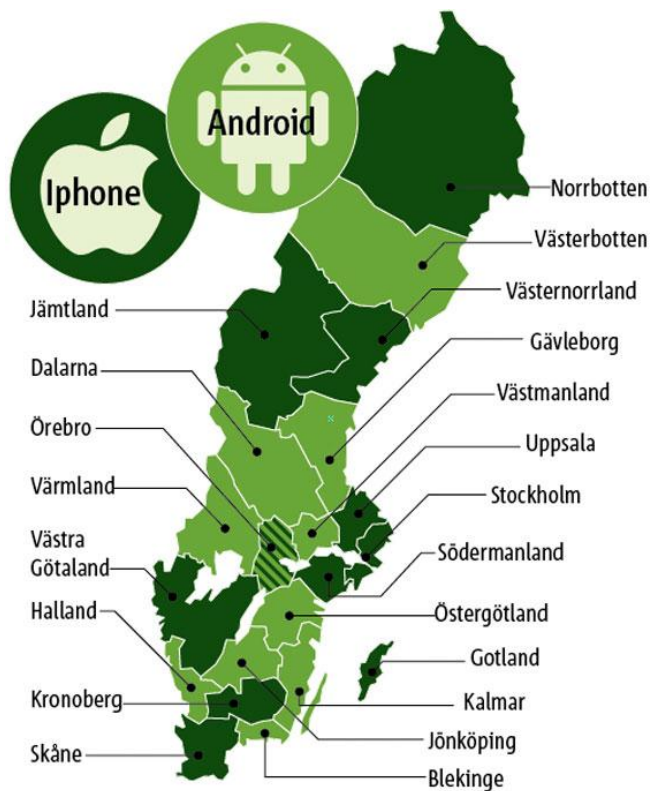


Figure 5 – Android and iOS smartphone market in Sweden [Metro].

The survey also shows that iOS has a dominance in three major cities of Sweden: Norrbotten, Jämtland, Västra Götaland while Android is more popular in the rest of the country. **Figure 5** shows the preference of two major smartphone's operating systems across the country.

4.1.3. Android Version Selection for the Best Market Result

The research regarding the usage share of Android platform, which was conducted in March 2011, is presented in **Figure 6** and **Table 5**. The highest usage percentage was the platform 2.2 which was 61.3% and the second highest percentage was Android 2.1; 29.0%.

It could be a good choice for companies to pick a platform which has the highest users' distribution; for example in this case, Android platform 2.2. One advantage of picking the platform 2.2 instead of others is that developers can use the highest level of the platform's API (Application Program Interface) which might not yet exist in platform 2.1.

It is important to consider about the API since API is a set of routines, protocols, and tools for building software applications. A good API can make the development easier and save a lot of development time.



Figure 6 – The usage share among Android platform on March 15, 2011

Table 5 – The percentage of usage share among Android platform on March 15, 2011

Platform	API Level	Distribution
Android 1.5	3	3.0%
Android 1.6	4	4.8%
Android 2.1	7	29.0%
Android 2.2	8	61.3%
Android 2.3	9	0.7%
Android 2.3.3	10	1.0%
Android 3.0	11	0.2%

However, companies should not base their decision on the development perspective only. As seen in examples, it can be realized that if the companies choose to use the API platform 2.2, they will lose totally 36.8% of customers who currently use older version of API platform. The 36.8% comes from 29.0% customers who are using Android 2.1, Android 1.6 = 4.8%, Android 1.5 = 3.0%. Thus, the customer usage is another important perspective companies must take into consideration as well.

QLogic AB has considered in both perspectives; development and customers, and decided to select Android platform 2.1 for the development.

Nonetheless, despite the fact that technology grows fast, a mobile new platform releases quite often. Companies should foresee on their new developing product's release date and a new platform version release date.

To give an example, if a company selects to support platform version 2.1 but the developers need 6 months to release an app, by that time no customers would use the 2.1 version any longer. So it is better for the company to take an advantage by selecting version 2.2 to save time in development instead.

4.1.4. iPhone Version Selection for the Best Market Result

After Apple announced to developers to stop supporting to 3.x version of iOS, most of the iPhone users had to upgrade their system up to 4.x version at the beginning of 2010. The new version 4.0 was not compatible with first generation of iPhone that developers had to design the applications to be aware of this separation. Developer faced up same problem with 4.3.1 version for second generation of iPhone. One of the reporter companies announced the separation of usage of 4.x version as 95% by March 2011. In **Figure 7**, usage distribution between iOS versions is shown on the left side and iOS version 4.x on the right side [CocoaNetics]

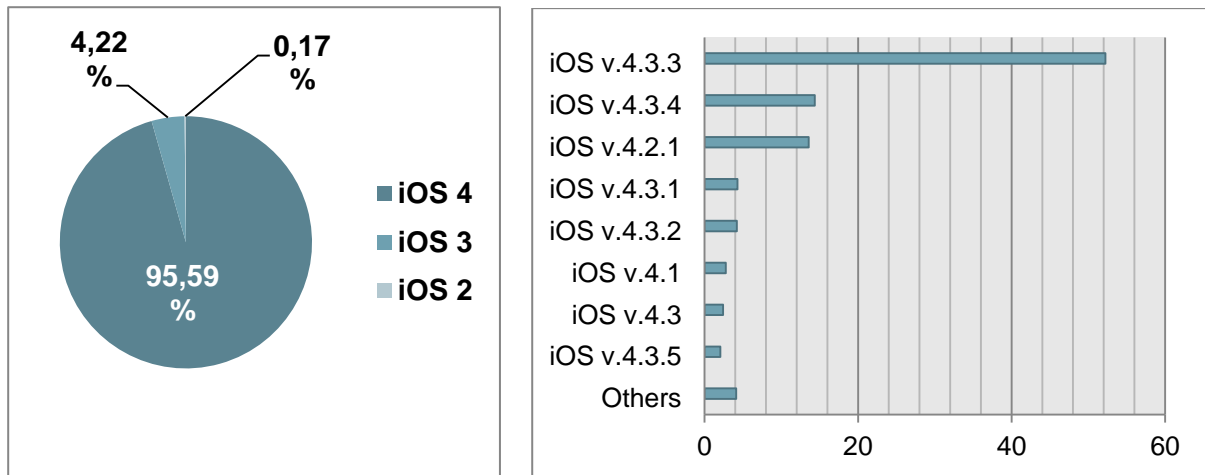


Figure 7 – Usage distribution between iOS versions.

Depending on the research above, we decided to set the lowest required version (deployment target) to 4.0 which is the most common version from all around the world.

4.2. Iteration 2: Elaboration Phase

We have analyzed problem domain in this phase also produced use case diagram, UML diagram, database, software user interface design, architecture design. To answer the research questions about usability and user experience, from those designed we have created, we answer them in this report section. As well as at the final part of the section, we have sample screen shots of mobile apps that we have implemented.

In this section, we discuss four main things: reusability which describes importance of it, reuse/non-reuse artifacts and how the project was implementing regarding the reuse. The next one is about system architecture & design patterns, we explain QBIS architecture, the importance of architecture and how we implement in the project. After that about user experience; importance of it, method on the implementation and then we show the prototype example. The last thing we illustrate in this phase is our example apps' screenshots.

4.2.1. Reusability

One of the research questions of this master thesis is "Consider in the whole software development process, what significant artifacts that can be used in common when it comes to develop one system in two platforms (in order to reduce cost of implementation)?"

Reusability is an essential to the project in term of minimizing development time, maintenance cost, decrease testing time. Especially this type of project which expanding new systems from the existing one, we know that the core business logic is the same, only the details and user interface layer design in client side is different. Web service methods which being used by current system and can be also used in the two new mobile applications is an easy example for the reusability.

In this section, as illustrate, we explain about our study and out implementation regarding usability. At the beginning of this section we briefly explain about the project overview, then later about artifacts that can/ cannot be reused and why. At the second section we explain about the system architecture, design pattern, web service reuse and the important role which affects the reusability. The third section talks about the user experience where we have done three ways in order to get the most clarify requirements. The last one we have shown the sample of one screenshots of our final products.

After QLogic AB's decision to implement this project which is called CRM mobile apps project, we began the implementation by designing the software. To analyze requirements and design software systems to communicate among developers and customers in this project, we used UML as a tool. Since UML has been a well-known tool in software projects for a very long time [UML Forum], we chose to analyze what artifacts we can reuse and how we can reuse them from the UML.

The UML Forum and Smart Draw are good websites which offers good explanations general idea of UML diagrams, tool suggestions and training [UML Forum][Smart Draw]. There are eight types of diagrams in UML that are generally used in software projects: Use Case Diagram, Class Diagram, State Chart Diagram, Activity Diagram, Sequence Diagram, Collaboration Diagram, Component Diagram, and Deployment Diagram. [Using UML] The definitions, explanations and how to write those UML diagrams can be found easily from the internet. But we can also recommend a couple of websites as well:

In this section we present how the project uses UML, see what UML is, and what artifacts in UML can be reused in order to reduce the cost of implementation.

In order to understand how we design the project architecture, GUI and UML in this elaboration phase, it is necessary for us to first describe the project requirements.

4.2.1.1. Project Overview

The project is to develop client phone applications that will work together with QBIS CRM server software provided by QLogic AB. The phone applications will be a way that users can conveniently connect to QBIS CRM and access key features and data. The phone applications should be developed for the iPhone and/or Android current platforms.

Introduction to QBIS CRM

QBIS CRM is a hosted web based CRM software tool that helps automate sales support, CRM and customer management. The software is a cloud based solution enabling users to use the system from any location and at any time. QBIS CRM is easy to use and there is no software to be installed on your computer.

Functional Requirements for CRM Phone Application

1. Companies and Contacts – search / add / edit / delete / view
2. Can call or email a contact directly from contact call card (one touch)
3. My tasks – add / edit / delete / view (phone calls, appointments, tasks) – all booked entries should also appear in the phone's inbuilt calendar functionality.
4. Opportunities – add / edit / view / delete
5. Products, product groups & prices (including filters) – search & viewing
6. Sales Pipeline – show the current sales pipeline.

General Application Requirements

1. Secure Login – Should be able to securely use the login credentials stored at server side for the user.
2. Language supported (initially two languages will be include - Swedish and English)
3. Work offline when there no reception – the application should work even when outside cellular network coverage. Be able to work with locally cached data and remote server data.
4. Highlight unsynchronized data. The user should be able to see data that is unsynchronized with the central server. The user should see when the last successful synch was made.
5. Extend the current QBIS XML Web Services gateway to facilitate the functionality by the client phone application and central qbis server.
6. Handle user privileges in the same manner as QBIS CRM.

4.2.1.2. UML Reuse

The only UML artifact which can be fully reused in this project is a use case diagram. The diagram can be used in both platforms because of the design is in a high level (as a matter of fact, it must be the same in all platforms). We give an example by one of the functional requirement, Companies as depicted in **Figure 8**. User *actor* has to be able to use all functions: Search Company, Add Company, Edit Company, View Company, Delete Company regardless platforms.

A use case specification can also be used in both platforms if a designer design in common view in both platform or not to mention in to details like user interface. For instance, not to specifically indicate where the menu or button is and where to return after error occurred. For example do not put the sentence such as “User clicks on add contact button **on menu bar**” instead of “User clicks on add contact button”. The first sentence makes it too specific for iPhone screen because

Android can use an option menu for the Add contact function instead of using a menu bar.

For reuse purposes, designers have to realize that two platforms can have different ways of working in the particular details. The general idea for designers is if they want to fully reuse analyze or design artifacts is that try not to design too specific into details.

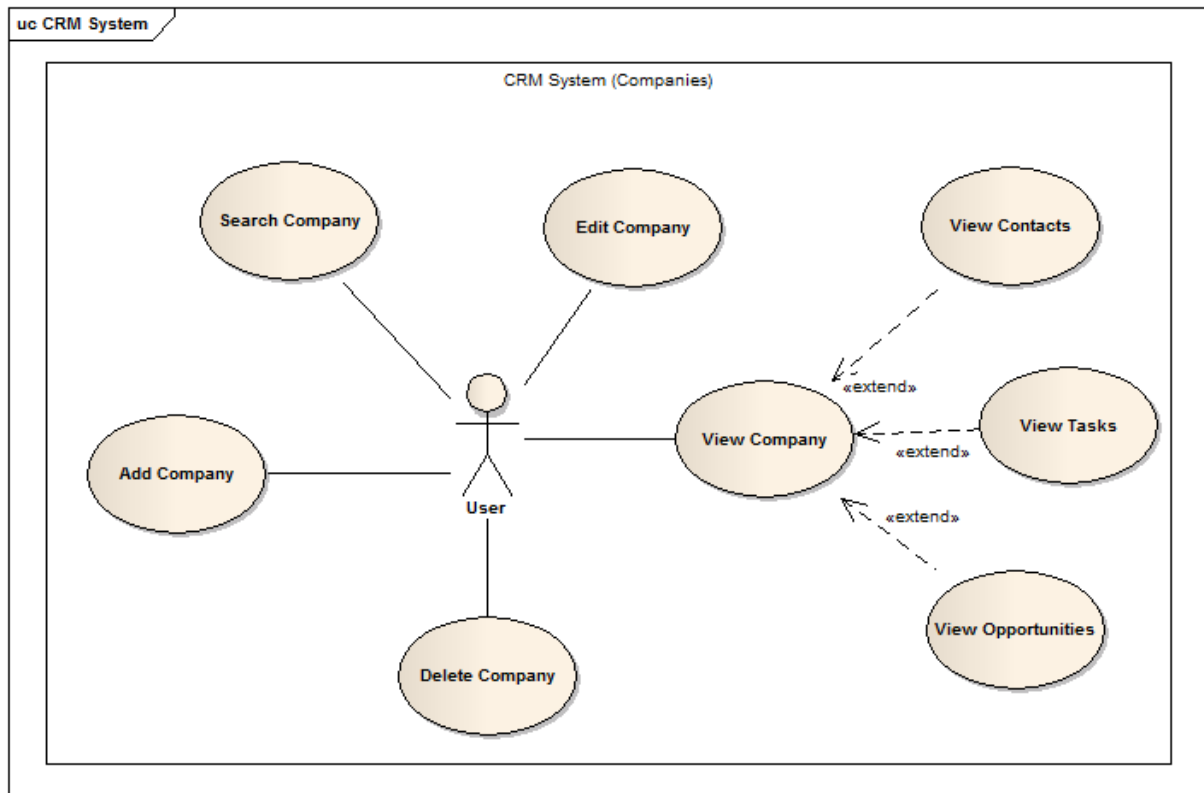


Figure 8 – Use case diagram of the Companies

The other UML diagrams besides the use case cannot be reused among both platforms in this project because they have different architecture and design pattern. Since the other UML diagrams we have mentioned such as a class diagram, component diagram are the diagram which contains such low level design or deeper into class, methods, sequence, so it is difficult to make a reuse out of it without a common design of the design pattern which we will explain further in this report.

4.2.1.3. Database Reuse

Core Data is a framework which Apple provides to developers or 'schema-driven object graph management and persistence framework' or the framework which is used to manage where/how data is stored, data catching and memory management. The reason why using Core Data with SQLite to store data instead of direct SQLite, XML or property lists is because the Core Data API allows developers to use SQL-less conditions on use a relational database, validate records or query data. It is

beneficial for developers who are familiar with object-relational mapping (ORM) technologies for example Ruby on Rails, CakePHP which abstract access to the database and another benefit using this approach is that when interact with SQLite in Objective-C, developers can ignore about managing the connection and database schema. [Apple iOS Dev]

We cannot say that there is existing same concept with core data of Apple because Android has the only simple store for keeping data called Shared Preferences. Shared Preferences class provides a general class used for save and retrieve a pair of key and value of primitive data types. However, even Android does not provide such core data like iOS does, there is also third party who provides such similar framework for example greenDAO which will be a layer between the objects and SQLite data storage.

From an experience of the implementation on this project, we have found out that there can be one artifact that can be fully reused. In order to explain easier, we have divided database artifacts into two levels below:

- Core Data Level.
According to the design of core data's framework regardless third party provider, the design of core data will not be fully used by both platforms because Android OS still has not been designed to support such core data like iOS does.
- Data Storage Level.
In this data storage level, both platforms use the same database storage which is SQLite. So we can and we have used exactly the same relational database design as shown in **Figure 9**.

In conclusion for this database reuse, we conclude that the reuse utilization depends on how the whole architecture designs which companies want to have. If the design is prepared for both platforms to have the core data level, for example if companies invest for a third party the core data's component for the Android platform, the core data level's artifact will also be able to reuse as well as the data storage level.

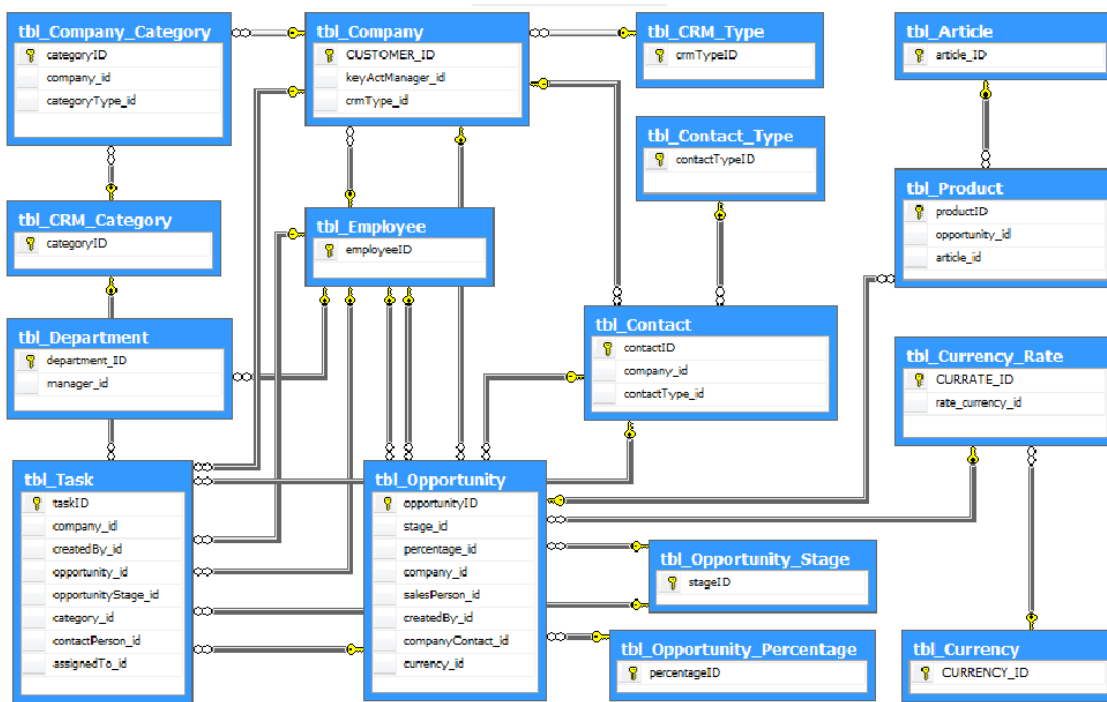


Figure 9 – The CRM relational database diagram

4.2.2. System Architecture and Design Patterns

We have mentioned in previous sections how a system architecture and design patterns play important role in reusability; in this section we explain why. The analyzed and design artifacts in UML which require details such as class, methods: Class Diagram, State Chart Diagram, Activity Diagram, Sequence Diagram, Collaboration Diagram, Component Diagram, and Deployment Diagram are difficult to be reused if the systems were designed with different design patterns. However, different software platforms have different architectures which will lead to the fact that we cannot just use the same design patterns in all platforms or at least use it after agree on pros and cons.

So in this project, as we mentioned, the implementation time was limited: we had to develop a project before we could fully research the architecture and design patterns. Nevertheless, we have done some research after the implementation which is describing here in this section. We begin by explaining about QBIS suite's architecture first because it is a core of the systems that will be extended by iPhone and Android. Then we explain the mobile platform's best fit design pattern and the possibility of using the same design pattern to gain more benefit from reusability.

4.2.2.1. QBIS Architecture and Communications

Current QBIS suite's architecture is logically divided into three tiers as shown in figure 10; user interface tier, application tier and data tier. All communication services are provided within cloud computing manner. Clients can request specific information with sending a web services message to QBIS server. Server processes the

message, fetches the requested information from database and returns back to the client.

- **User Interface Tier:** All available operations and message format to provide basic interactions are defined in this layer as a service contract. Also it includes policy of the communication which defines message exchange pattern such as request/response or one-way. It is also responsible to convert received message format to proper data types, which will be used in application tier.
- **Application Tier:** All the business logic of the web service is located in this layer. Service calls, accessing data layer or invocation of all kind of operations are performed. Also authentication and validation are verified in here as well.
- **Data Tier:** Database adapters and all other recourses to access database are defined in here. It contains an agent to establish communication between business logic and database, which is located in another server. Also authentication for database access is verified in this layer.
- **Cloud Service:** End users access cloud-based applications which in this project QBIS-CRM through their mobile app or web browser.

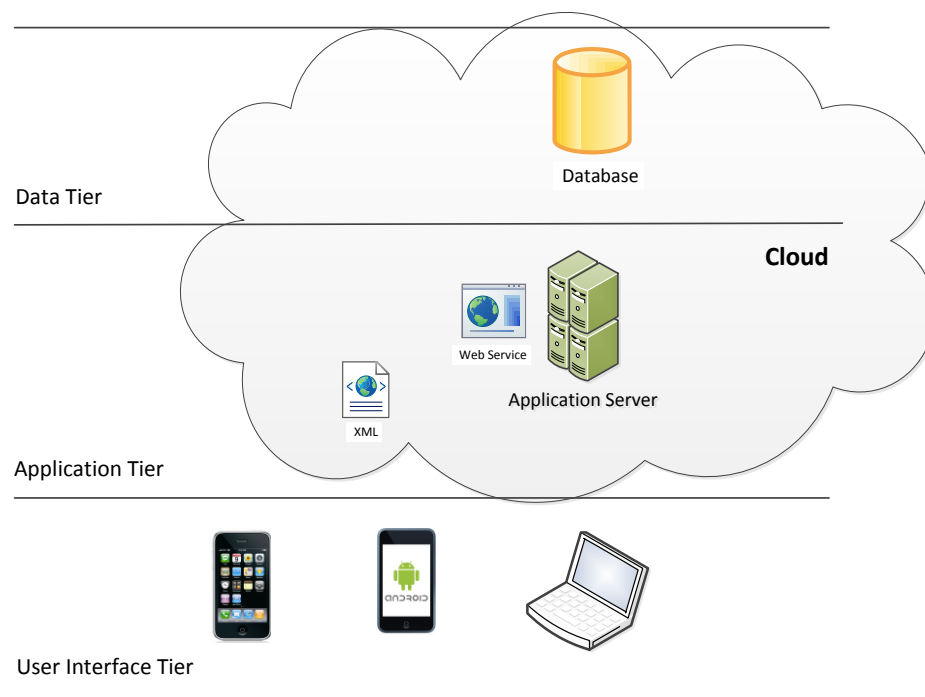


Figure 10 – QBIS general architecture

All communications to QBIS application server is provided through web services via the protocol SOAP (Simple Object Access Protocol). The request is retrieved from client in SOAP message format. As previously mentioned, the message is converted serial commands and data types, which are used to call related operations. Then the operations communicate the stored procedures via database adapters. Procedures

are performed and return related result to business logic layer again. Sometimes these results can be simple data types like integers as well as data tables. Depending on the results, the response message is encoded and sent back to client. Sometimes an error can occur in any levels of the process. In such cases, the service prepares SOAP fault message to inform client. Also QBIS administrator receives a descriptive mail about the situation.

4.2.2.2. Web Service Reuse

Despite the fact that in this project we had three different clients; web application, Android app, iPhone app, so we applied SOA concept in order to bridge those three different software architectures together. The QLogic AB provides those .NET web services for the project; those web services are flexible isolated interfaces which all three systems can call the exact same one web service, no need three different functions provided. As shown in **Sample Code 1**, the QBIS architecture provides XML-based file which described with WSDL (Web Services Description Language). The code shows the WSDL example of one of the web services; GetCRMCompanies which all clients commonly use.

```
GetCRMCompanies
Get all CRM companies from QBIS. The companies are returned as a .NET untyped
DataSet. For more information about the return type see QBIS Help. In case of any error the
method returns a NULL value.

POST /ws/qbis_mobileapp.asmx HTTP/1.1
Host: bin.qbis.se
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "https://bin.qbis.se/WS/QBIS_MOBILEAPP/GetCRMCompanies"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetCRMCompanies xmlns="https://bin.qbis.se/WS/QBIS_MOBILEAPP">
      <Company>string</Company>
      <Password>string</Password>
      <SyncDate>dateTime</SyncDate>
    </GetCRMCompanies>
  </soap:Body>
</soap:Envelope>
```

Sample Code 1 – The WSDL example of GetCompanies

4.2.2.3. Best fits Design Patterns in iPhone and Android

Design patterns tell how classes, objects should be created, their relationships how they are interacting with each other. The UML reuse artifacts and coding components rely so much on design pattern, the more similar design, the more reusable components can be utilized. However, each mobile environment has probably been designed to support different design patterns which can be one issue that can stand in a way of usage reusability.

The Cocoa environment on the iPhone has many of architectures and mechanisms to support different design patterns. Android is also flexible to design in any kind of patterns but the question is which pattern we can make the most use from. Patterns supported by Cocoa are for example Abstract Factory, Adapter, Chain of Responsibility, Facade, Singleton etc. but MVC (Model-View-Controller) and object modeling seems to be the most important and widely-used in Cocoa and both patterns are largely interrelated. The **Figure 11** is illustrated the Model-View-Controller (MVC) Design Pattern.

In Android, it is as well possible to design the application based on MVC pattern. However, Android OS does not provide the whole architecture between layers isolated enough to implement MVC easily, for example the ambiguous of Activity class; we are not really certain whether we can call it a view or a controller. One of the patterns we have found it might be the better selection for Android is Model-View-

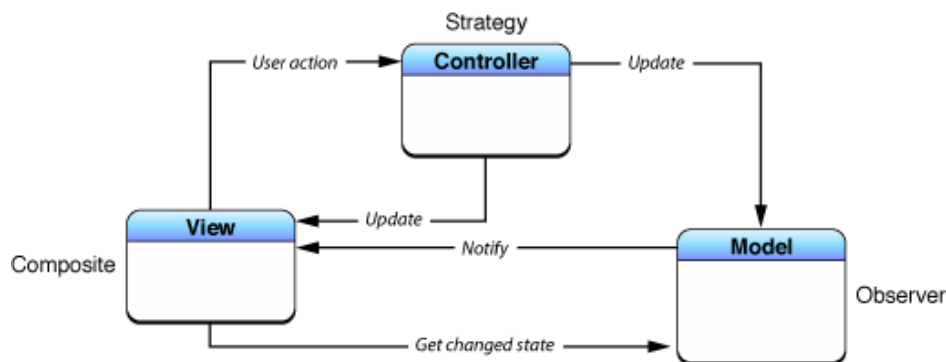


Figure 11 – Model-View-Controller (MVC) Design Pattern is shown with its relationships.

Presenter or MVP as illustrated in **Figure 12**.

However, different patterns have different ways to use and have different limitations. For example, pattern Abstract Factory who provides an interface for creating families of dependent objects by not specifying concrete classes, so it is easier to learn and to use the class. However, the tradeoff is a simplified super class cluster will make more difficult to create custom subclasses. So, in the sakes of commonly used of two mobile platforms and OS structure design's compatibility, two design recommendation patterns will be explained here in this report which are MVC and MVP.

4.2.2.4. The Model-View-Controller Design Pattern (MVC)

Object-oriented programs that based their design on MVC are easily extensible than those which are not. When the programs adapt MVC design pattern, interfaces can be define better and tend to be more reusable and adaptable when there are some requirement changes. In addition to Cocoa, some of their new technologies such as bindings, the document architecture and scriptability are based on MVC; custom objects must play one role that defined by MVC.

Roles and Relationships of Model-View-Controller (MVC) Objects

- For Model objects, they are used to encapsulate important data and their expertise behavior. The best practice of model objects design is to have no explicit connection to view or user interface layer.
- For View objects, they are used to display data from the application's model.
- For Controller objects, they are the middle person, responsible to ensure that the view objects have access to the model objects and make the view learned if the model has some changes.

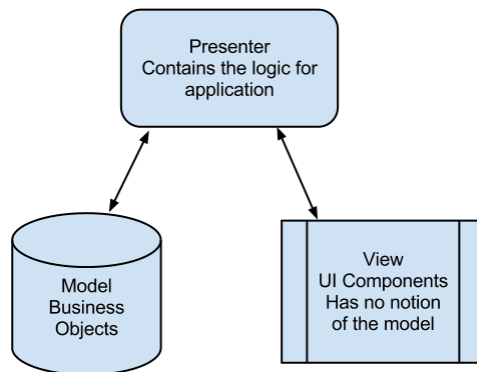


Figure 12 – Model-View-Presenter (MVP) Design Pattern

4.2.2.5. The Model-View-Presenter Design Pattern (MVP)

Model-View-Presenter is a derivative of the Model View Controller Pattern. The MVP pattern provides a cleaner implementation of the Observer connection between Application Model and view.

Roles and Relationships of Model-View-Controller (MVC) Objects

- For View objects, contains UI components, and handling their events
- For Presenter objects, this will handle communication between the model and view layer. It can be seen as a gateway of the model. It retrieves data from repositories (the model), format and display to the view.
- For model objects, is an interface used for defining data to be displayed or acted upon the user interface.

In conclusion, we have found from our research and experience as a software engineers that the best-fits design pattern for iPhone is MVC and for Android is MVP. So to make a reuse of artifacts such as UML design and coding components by two different design patterns is quite difficult because of their classes, method, interfaces will be designed in different ways. However, it is an interesting idea if companies want to trade the best fits with a usage of reuse as ignoring about the best fits and use the same design pattern in both platforms. The tradeoffs, the possibility of implementing it can be put to the further research in our future work.

4.2.3. User Experience (GUI)

Another research question of this master thesis is “What constraints used to decide the GUI context needed on light weight mobile application?”. It is clearly not an easy job for GUI designer to compress all information from screen size 19 inches to 4 inches as illustrated in **Figure 13**. In this section we explain three methods we have used in the project to produce the best GUI design: comparing attributes with competitors, interview and prototyping.



Figure 13 – Web application and Mobile application size comparison

4.2.3.1. Comparing Attributes with Competitors

At the beginning of the project, we have studied what CRM system is, what the main CRM functions in the market are. Then we listed all screens, attributes, and functions from the existing QBIS-CRM web application. We do the same with main CRM mobile apps providers; both Android and iPhone platforms. Then we compared field-by-field, function-by-function among current QBIS application and other competitors as shown in **Table 5**.

As illustrated in the table, column “A” is the column that combines all fields and functions that exist in potential providers’ applications; column “C” shows what QBIS-CRM application has, four columns as shown as “D” are columns that identify what items each competitor has in their mobile application and “B” column indicates the final decision what fields should be included in our mobile apps. The decision of the selected functions have been made by the business analyst who know best about the system, also is familiar with users most. The business analyst’s decision can be a good based conclusion for what is needed for CRM mobile application users. However, we did not based decision only on the decision from the comparison, but we also have done some interviews and prototype which will be discussed further as well.

Table 5 – Comparison between QBIS web application and other CRM mobile apps provider

No.	Competitors/ Item	Decision	QBIS Web Application	D			
				Mobile App A	Mobile App B	Mobile App C	Mobile App D
3.3	Company Detail						
	o Name	Y	/	/		/	/
	o Customer Code	Y	/				
	o Abbreviation	Y	/				
	o Organization Number	Y	/				
	o Postal Address	Y	/	/ (Address1)			/
	- Address	Y	/	/			
	- Postcode	Y	/	/			
	- City	Y	/	/			
	- Country	Y	/	/			
	o Invoice Address	Y	/	/ (Address2)		/	/
	- Address	Y	/	/			
	- Postcode	Y	/	/			
	- City	Y	/	/			
	- Country	Y	/	/			
	o Visiting Address	Y	/				
	- Address	Y	/				
	- Postcode	Y	/				
	- City	Y	/				
	- Country	Y	/				
	o Phone	Y	/	/		/	/
	o Phone2	Y		/			
	o Phone3			/			
	o Fax	Y	/	/			/
	o E-mail Address	Y	/	/		/	
	o Web page	Y	/	/		/	/
	o Key Account Manager	Y	/				
	o Active	Only show	/				

4.2.3.2. Interviewing

Even though, the decision from the business analyst has been made based on the comparison among competitors, but we believed that interviewing is still needed. We thought that what does not contain in competitors' apps does not mean that QBIS-CRM users do not need.

We have managed to get interview sessions from the business analyst and also from sales persons who are close with users and customers. There are a lot of times that they got the application's problems, new requirements, changes from the users which can bring some new idea for the mobile applications.

4.2.3.3. Prototyping

After all information needed had been decided, it was rather difficult to imagine what it would be like in the mobile small screens, difficult to realize that can all information fit in the screens and how it will be operated as screen scenario. From what has been said in [RE-Good Practice Chapter 4] that if there are vague or poorly understood requirements, it is better to demonstrate their behaviors or what system can provide into a prototype to users and system stakeholders. Then those users can experience with the prototype in order to refine their ideas about the system requirements. So, we took all attributes, functions from the result of the decision from comparison and

the interview to make the prototype applications to get an early feedback from business analyst, and then we can adjust information, captions, and screen styles in order to fit the best for users' experiences. The Companies module prototypes of Android and iPhone apps have been displayed in **Figure 14 - 15**.

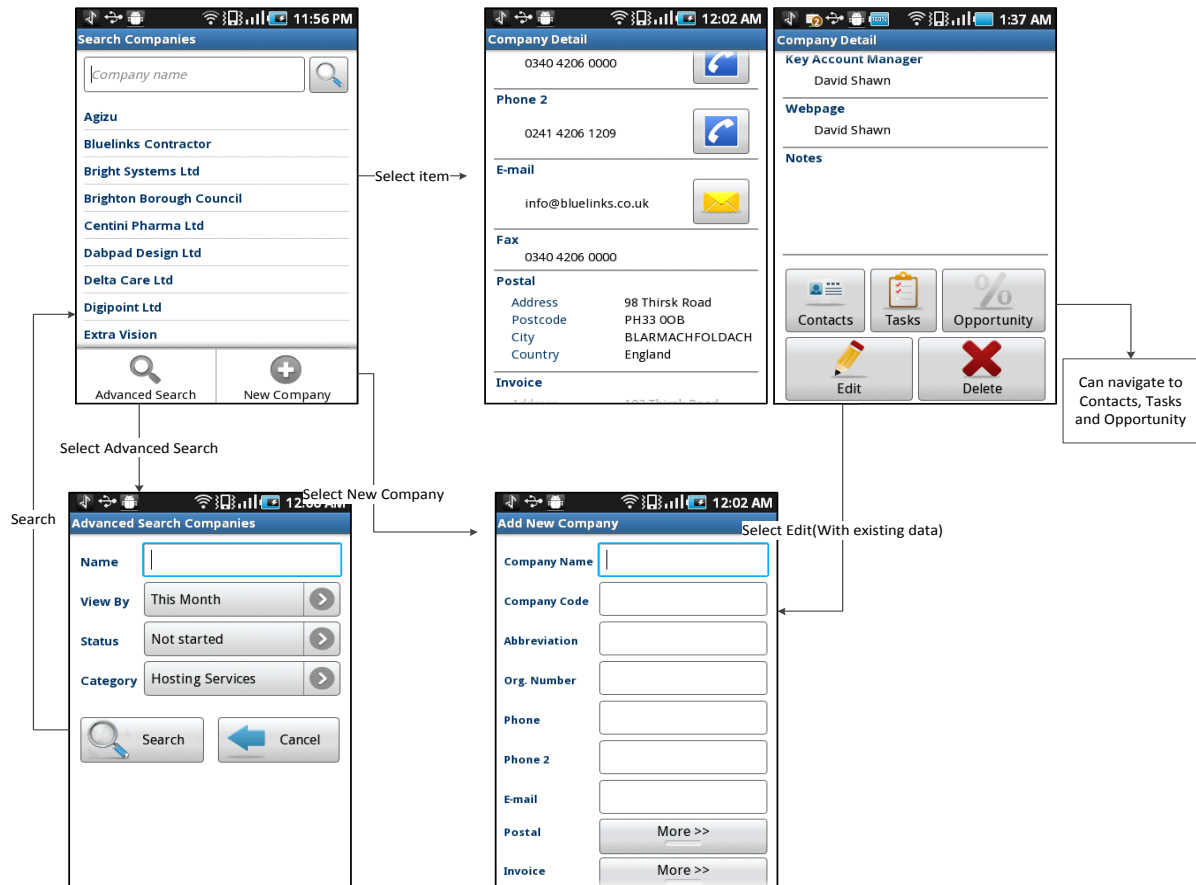


Figure 14 – A “Companies” prototype from Android app.

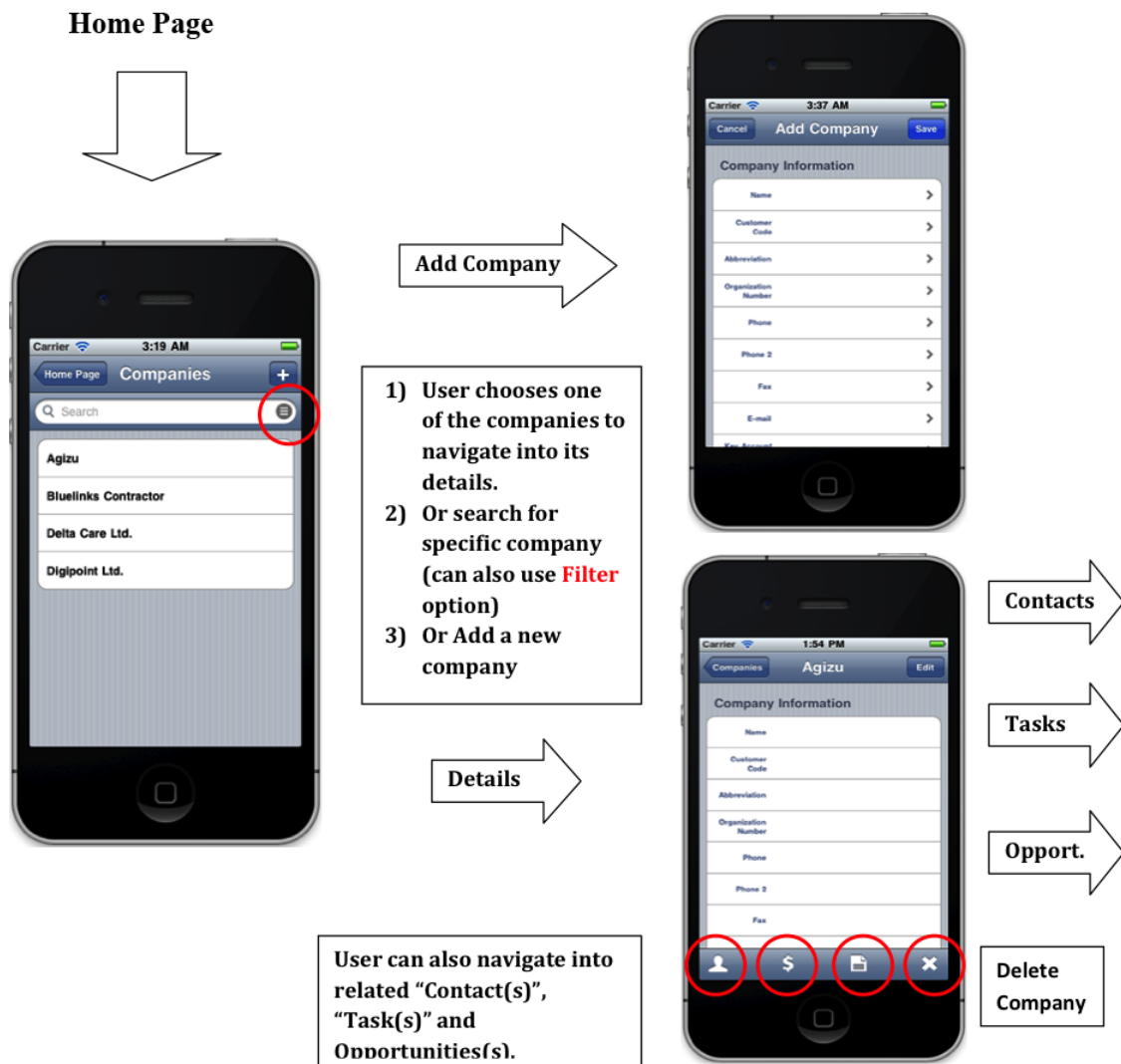


Figure 15 – A “Companies” prototype from iPhone app.

4.2.4. Final Products

The main menus of iPhone and Android apps are shown in the **Figure 16**. All functional requirements which is described in the section 4.2.1 have been implemented; Companies, Contacts, Tasks, Opportunities, Products, Pipelines. We illustrate only one module which is Companies module on the **Figure 17** for the Android app and for the iPhone app on the **Figure 18**.



Figure 16 – A screen capture of the main menu of mobile apps

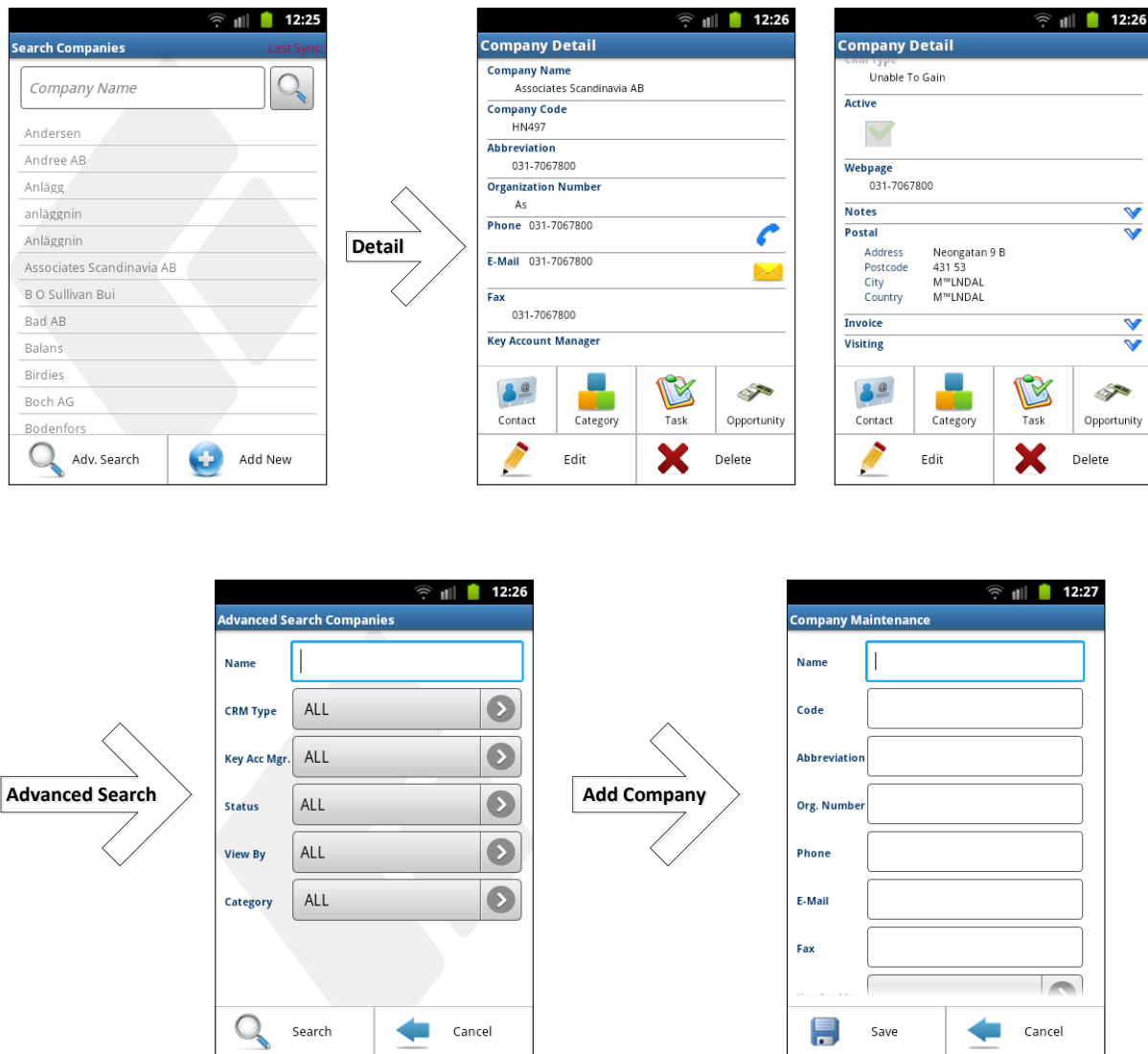


Figure 17 – Screen captures of Companies screens on Android app

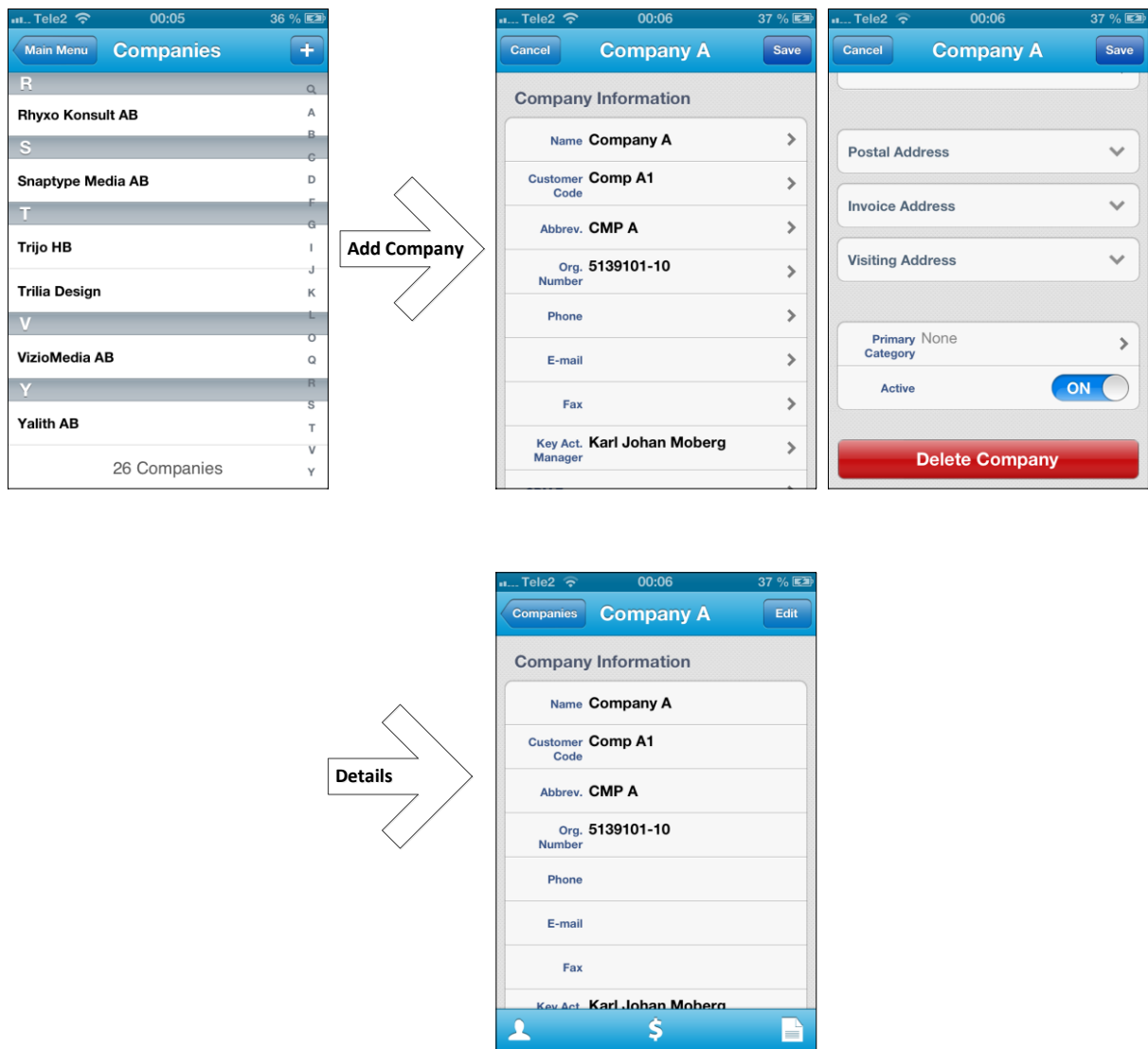


Figure 18 – Screen captures of Companies screens on iPhone app

4.3. Iteration 3: Construction Phase

In this phase, we explain how the synchronization process works in both server and client side. We introduce four steps that the two-way (bi-directional) synchronization occurs in a quick and smart way. Also we present low-level testing methodologies to validate the behavior of the source code and functionalities. Unit testing help us to be sure of the quality of the mobile applications.

4.3.1. Synchronization

It is important to have multiple copies of server data in all QBIS CRM applications without corrupting data integrity. The applications can able to work offline and keep the both server and mobile side in sync. But the synchronization process should be applied in a smart way. Both sides only exchange recently modified data since downloading whole data is not a good solution indeed. Because it takes long time to be done and also consume large bandwidths that means extra costs for the users.

There are some synchronization strategies that had been already applied in some projects before. In one solution, the data is pulled down from backend application servers and synchronized back to central server when it is needed, the solutions called polling and push synchronization. [Rhomobile] Or another solution could be third party API that named Microsoft Sync Framework 4.0 CTP. All business logic is moved into server that enables to keep client (mobile application) out the mandatory installments. The sync mechanism detects conflictions automatically in both sides and eliminates them. [Sync Framework]

Although each solution is well defined and had given successful results before, they are so comprehensive that require a lot of time to implement. We had to prefer much simpler but strength solution that it could be easily adapted to existing environment of the company. So we decided to build up our own solution. In this way we were able to use current structure while adding some extra features as well.

As you see in **Figure 19**, we decided to divide synchronization phase into 4 steps:

1. **Get Deleted Records:** Retrieve all deleted records from server and remove permanently if any of them is still in the mobile application.
2. **Send Deleted Records:** Send all deleted records to server and remove permanently if any of them is still in the server database.
3. **Send Modified Records:** Send all modified records to server and if a record does not exist in server, it creates a new one otherwise modifies the existed one.
4. **Send Modified Records:** Retrieve all modified record from server and if a record does not exist in mobile application, it creates a new one otherwise modifies the existed one.

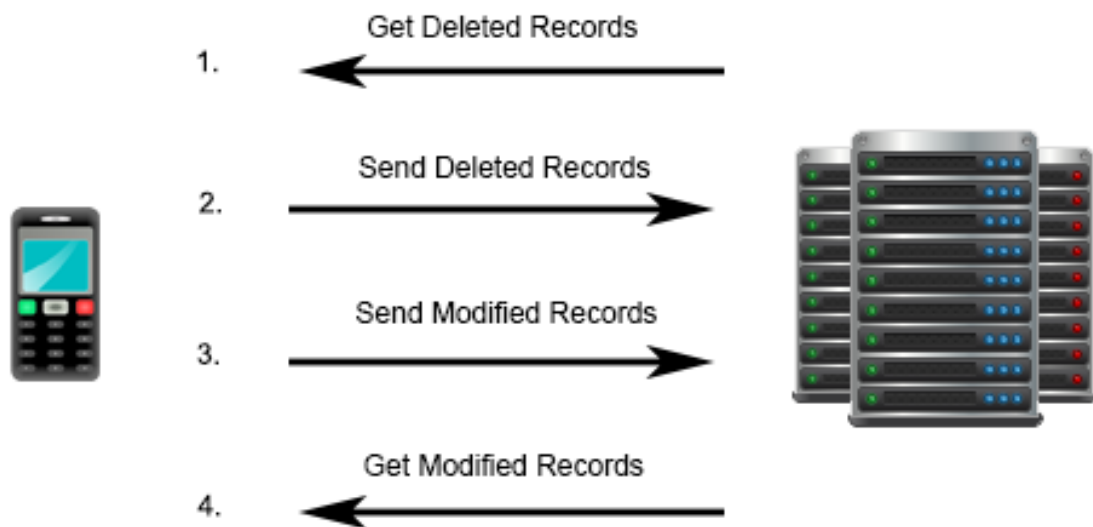


Figure 19 – Synchronization process in four steps.

Since QBIS Web application is connected to SQL server on the background, we need to know which data had been modified and should be retrieved into mobile device. It is the same for mobile application that recently modified data must be identified and sent to the server. Also deleted records should be kept in another table to be forwarded in next synchronization.

First, we introduced “last synchronization time” parameter that the time is saved in when the application is synchronized. Then we define a timestamp parameter for each record that holds time value. The value is automatically updated with current time if the record is modified. Then we define a sync flag for each record in application that indicates the record had been modified or not. Finally, we create duplicate tables to keep deleted records with deleted time stamp.

So when application request to retrieve all deleted records in first step, it also adds the last sync time in the request. Then server fetches the records whose deleted time is later than last sync time of the app. In next step, application fetches all deleted records from its tables and sends to server to be deleted. When the deleting process is done successfully, application removes all deleted records from the tables. In third step, application fetches all modified data with checking their sync flags and sends to server to be modified. If last updated time of the sent record is earlier than last updated time of the server record, the updating process is skipped. And if there is not related record at all, the new one is created directly. In final step, application retrieves modified records from server depending on last sync time. As in the previous step, the records are either modified or newly created.

4.3.2. Unit Testing

Some of the mobile platforms offer different framework to create test cases while others support with third party plug-ins. The iOS supports SenTestingKit (also known as OCUnit) which is fully integrated with Xcode. [Unit Test] On the other hand Android SDK comes with a testing framework within ADT Plugin (Android Development Tools Plugin). It is based on JUnit that developers can also use JUnit classes separately without calling Android SDK. [Android Testing]

4.3.2.1. SenTestingKit (OCUnit)

OCUnit is a testing framework which is designed for Objective-C based applications. It consists of a set of classes and command-line tools that developer design test cases and run them with the targeted application on simulator or device. There are two types of tests in this framework [Apple Testing]:

- **Logic Tests:** are intended to check for correct functionality of the code. It is only applicable in simulator.
- **Application Tests:** covers full application environment with involving connection of user interface and controller objects. It is only applicable in device which is more realistic environment.

Test case is a regular instance method of test suite class. [Apple Testing] It invokes the methods which will be tested in unit. There two optional methods, one is called before to setup the testing environment and the other one is called after the case to remove any residual variables, structures or objects. An example code is shown in **Sample Code 2:**

All the results of the test suites are shown as system logs in console. Developers can also create new schemes to separate logic and application test suites or apply only specific test cases from both of them.

```

#import "QBISCRMSampleTestSuite.h"
#import "QBISHomeViewController.h"
#import "Employee.h"

@implementation

- (void)setUp {

    [super setUp];
    app_delegate = [[UIApplication sharedApplication] delegate];
    employee = [[[Employee alloc] initWithName:@"Jack" andSurname:@"Sparrow"]
retain];
}

- (void)tearDown {

    [super tearDown];
    [app_delegate release];
    [employee release];
}

/* Logic Tests */
- (void)testAppDelegate{
    STAssertNotNil(app_delegate, @"Could not find application delegate.");
}

- (void)testEmployeeObject{
    STAssertNotNil(employee, @"Employee object has a null reference.");
}

- (void)employeeCodeTest{
    STAssertTrue([[employee code] isEqualToString:@"JaSp"], @"Employee code test
is failed");
}

/* Application Test */
- (void)testLogoutButton{

    QBISHomeViewController *homeViewController = app_delegate.rootViewController;
    UIView *homeView = homeViewController.view;

    [homeViewController buttonClicked:[homeView viewWithTag: 8]];
    STAssertTrue([homeViewController.loginViewController, @"Login view was not able
to created"]);
}

@end

```

Sample Code 2 – Test suite for checking functionality of employee code and logout button behavior.

4.3.2.2. Android Testing Framework (JUnit)

Android test structure consists of test project that includes test packages and test-case classes which are derived from standard JUnit classes. Test project is a directory to keep all created source-codes and other related files in the same place. Eclipse offers a wizard for creating test projects. It automatically sets up and creates

the test package which will be used by test runner. Test-case classes are abstract java classes that could have multiple test methods inside. An example code is shown in **Sample Code 3**:

```
package com.qlogic.qbiscrm.test;

import com.qlogic.qbiscrm.QBIS_CRM;
import android.test.QBIS_CRMTTestCase;
import android.widget.TextView;

public class QBIS_CRMInfoBoxTest extends QBIS_CRMTTestCase <QBIS_CRM> {
    private QBIS_CRM activity; // the activity
    private TextView infoView; // the textview
    private String testString;

    public QBIS_CRMInfoBoxTest() {
        super("com.qlogic.qbiscrm",QBIS_CRM.class);
    }
    @Override
    protected void setUp() throws Exception {
        super.setUp();
        activity = this.getActivity();
        infoView = (TextView) activity.findViewById(com.qlogic.qbiscrm.R.id.textview);
        testString = activity.getString(com.qlogic.qbiscrm.R.string.infoString);
    }
    public void testPreconditions() {
        assertNotNull(infoView); // Test pre-condition
    }
    public void testText() {
        assertEquals(resourceString,(String)infoView.getText()); //Test case
    }
}
```

Sample Code 3 – Test case class for comparing two strings.

After start to run test case with JUnit framework, the application under testing is launched on target device and the result is shown with different colors which red indicates the failed while green indicates passed. Developers are also able to see overall view such as number of test cases, number of failures and errors.

4.4. Iteration 4: Transition Phase

In this phase, we explain how the applications were validated. Since we had limited time to launch the applications, we suggested that the company apply one of the agile practices, user stories. After that, we mention distribution process of applications for both platforms. We provide brief information about application markets and submitting steps. Finally, we talk about the online help that gives technical information about the applications.

4.4.1. User Validation (Applying Acceptance Testing)

In software projects, validation criteria are generally created by business customer, which mostly uses the software product. But sometimes validator can be owner of the product or project manager if the project is developed internally.

In our project, we suggested to apply acceptance testing in agile way. Agile development primarily focuses on iterative development processes that a new requirements or modification can be received in any phase of the development. And since we applied the one of the iterative developments, keeping these requests in a single structure would be best option. So our product owner summarizes all requirements in user stories. User stories are used in most of the agile development process. It helps to determine significant points for being sure about functionality of the program regarding to its requirements, which are specified before. The main purpose of the user stories is reducing the testing time with getting respond much faster and applying last retouches rapidly. Most of the stories are written in daily languages, which consist of 1-2 short but clear sentences.

Accordingly, our product owner wrote the all user stories that few of them are shown below. We group them as functional and non-functional requirements. Since QBIS CRM has three different role levels (user, manager and administrator), stories for functional requirements were written regarding to these roles. For non-functional requirements, we use “the application” key word.

Functional User Stories:

- As a user, I am able to see CRM companies with their contacts, tasks and opportunities.
- As a user, I am able to download latest modified records from server.
- As a user, I am able to call-mail companies as well as contacts.
- As a user, I am able to list all tasks, which are assigned to me. The list can be exportable if its “Show in Calendar” option has been activated. As a user, I am able to see all “Opportunities” which are created by or assigned to me.
- As a user, the system must warn me if there is a new version of the app. Also it blocks the all synchronization functionalities.
- As a user, I am able to use the application in both English and Swedish.
- As a manager, I am able to see all “Opportunities” in “My Team” which are created by or assigned to my teammate or me. To be a part of “My Team”, person should work with same department.
- As an administrator, I am able to see whole “Opportunities” in the system.
- As an administrator, I am able to delete the companies which is not assigned to any tasks or opportunities.

Non-functional User Stories:

- The application needs to provide secure lightweight connection with server.

- The application needs to download and import all initial data within 4 minutes.
- The application needs to launch each page less than 3 seconds.
- The application needs to have “delete all data” functionality that application data can be removed securely.

Before submitting the applications, we ensured that all scenarios in stories have been successfully applied.

4.4.2. Distribute the Applications

After all test procedures have been done successfully, it is time to publish the application in stores. There are few things need to be done to prepare the application files before upload them to stores. We will briefly describe these steps for both platforms separately.

4.4.2.1. App Store

The App Store is an online application market that allows to developer to distribute their applications, which are developed in iOS SDK. It is accessible from mobile devices directly that user can download the applications on their mobile device by purchasing them. Charging depends on the application that it can be free or at a cost. Also users have a chance to download the application onto their personal computers to sync mobile device later.

Distribution process of the iOS applications consists of 5 steps:

- 1. Sign up for Developer Account and Program:** The developer account is required for distributing an application in App Store. After signing up for an account, developers need to select one of the developer programs that identify the user and allow distributing the apps in store. For commercial releases we chose “Developer Program – Company”.
- 2. Create and Install Certificate:** All applications must be signed with using developer’s key chain. Using the Keychain Access application, which is default application for all Mac OS, a new key pair can be generated. After uploading the key chain to developer program web site, a certificate can be downloadable.
- 3. Generate Application ID and Provision Profile:** Each application must have a unique identifier that should be associated with the application and be located into the Provisioning Profile.
- 4. Prepare the Application for Distribution:** Provisioning profile needs to be imported into Xcode Development IDE. Deployment target must be chosen which is required to set minimum SDK. And also icon file and version number must be set. Finally the project can be built with distribution settings, which includes the signing application with certificate.

- 5. Distribute the Application:** The iTunes Connect is the web site allows the user to manage all related tools and materials such as sales, contracts, reports, user accounts, and apps. All the information about application and screenshots must be provided. Finally the archived version of the application needs to be uploaded using Application Loader utility, which is also integrated into Xcode IDE (Integrated Drive Electronics).

After submitting the app to the App Store, Apple developers review the app and publish it. This process can take couple days and sometimes they can ask for more information about the app or directly reject the submission in some reasons.

4.4.2.2. Android Markets

Unlike App Store, Android developers can distribute their application in different markets. Even they can publish the applications in their web servers with providing “.apk” file. But still most of the developers prefer to share in Google Play.

Google Play, previously named Android Market, is the official application store of the Google that users can download various Android applications. Also developers can publish their own applications to the market.

There are few steps to submit the Android app to market:

- 1. Register for publisher account:** A developer needs to register Google Publisher Account to submit the Android App. After the agreement is confirmed and registration fee is paid, the account will be ready for uploading process.
- 2. Check publishing list:** Before submitting the app to store there are few things need to be checked by developer. Google Play provides a list to developers with highlighting some significant points that developer can easily prepare the application for successful release. All points in the document need to be done one by one.
- 3. Compile and submit the app:** After all points are applied carefully, it is time to compile the Android project for submission. First of all developers must sign the application with cryptographic private key whose validity period ends after October 22th, 2033. Then the app will be ready for uploading process and after developer uploads the app to store, it will be available in a few hours.

4.4.3. Online Help

Online help documents of QBIS mobile applications give brief information about application features and functionalities to provide best user experience. Documents are web-based files that can be reachable online from both web and mobile applications. Both English and Swedish languages are supported.

5. Discussion

In this section, we discuss the research questions that we have strived to address in this thesis and that we used to guide the engineering of our mobile applications. First, we discuss the different problems that guided a better and faster migration solution when the existing application was extended to the mobile versions. Then, we explain the significant artifacts for both platforms that reduced cost of implementation. Subsequently we clarify the constraints to have a lightweight context for mobile applications. After that, we explain the synchronization methods to show how data correctness and efficiency is provided. Finally we present our amount of test coverage explaining the test methods which we applied.

What kind of problems should be considered when companies want to extend their existing application to mobile client applications (from a software engineering perspective)?

Companies encounter different kind of problems when they want to extend their existing systems to mobile applications. Especially performance was an important issue when the company wants to migrate the web application into mobile one. Although most of the mobile devices are multi-functional, they have restricted hardware resources. We designed the QBIS mobile applications in a smart way that they fetch and present the user data efficiently. Also they only download the recently modified data to reduce the synchronization process time significantly. On the other hand confidentiality was another problem for the company. Most of the security violations occurred in server side and can be eliminated with providing some extra security policies such as IP-blocking or firewall protection. On the contrary, mobile devices are exposed to various threats. But in mobile applications, we added additional authentication procedures, which are apart from default behavior of the device. Users have to login to the system to reach the application data and also they can logout to leave the app more securely. We also provide secure communication with server that protects the application against security attacks. It eliminates all interception attempts with providing default encryption mechanism.

Considering the whole software development process, what significant artifacts can be used in common when it comes to developing one system on two platforms (in order to reduce cost of implementation)?

In this project we developed a couple of significant artifacts which are commonly used by both platforms. When we started the project, we create the time plan that we can follow the same process rather than focusing on independent strategies. Most of the research were done together and software documents were prepared for both applications. Especially, all specifications were written in a single document. By this way, we saved lots working time. At some point we had to separate the project in two similar branches. But still we continued to implement web services for both iOS and

Android applications that they could consume same methods. It reduces the development time and costs to setup different services for each platforms (e.g. Bonjour or JBoss) which are more friendlier individually.

What constraints should be used to decide the graphic user interface context needed for lightweight mobile application?

We knew that there would be constraints to have lightweight mobile applications on the other hand QBIS web solution consists of several modules and dozen functionalities. It would be heavy to implement all of them into applications. Also each user database has thousands of CRM information that would be impossible to download into mobile device. Most of the mobile applications were designed to assist its users when they are not available to reach web solution. Only significant information should be kept the device. Consequently, we decided to cover basic functionalities and download essential information that would be enough to meet user expectations. First of all, we specified the features, which are generally used in web solution. We determined them with a small internal questionnaire. Then we talked with marketing department to have their opinion since they are the one who knows the customer requirements. They also communicated with the customers to acquire additional feedbacks. And finally, we gathered all results in a document and selected the important ones, which should be cover in our project.

What kind of methods and what considerations need to be taken into account for synchronization to serve the needs for data correctness and efficiency?

Synchronization methods should provide the data correctly and efficiently. We had to design the synchronization phase smartly since it directly affects the application performance. Waiting a long time for synchronization process does not please the customers. Besides weak implementation can consume additional bandwidths, which equals to extra costs. But in our applications operate the upload and download requests according to last synchronized time. The records in server have timestamp that when the mobile application sends a download request, server checks for timestamps and returns the records whose last updated time later then the last synchronized time of the application. Also when the mobile application sends an upload request for saving a new record, server assigns a unique id for that record, create a timestamp to notify other application and return the id back. If the server has already same record or even any problem occurs in this process, server returns an error code notify the application that the record is marked as a fault one. In this way duplications are avoided and date correctness is provided.

How much test coverage should be considered to fulfill all user requirements and get more sufficient in target platforms?

Another interesting point is determining the amount of the testing coverage that we include within the project. It is not easy to estimate how much testing is required or how many test cases must be written for the completeness. However there are

several coverage criteria that could be considered. Especially Requirement Coverage is one of the high level metric that verifies the all required functionalities in user perspective. We created a couple of test cases for each requirement. Results of the test cases are quantitative information that we aimed to reach 100 percentage of coverage with fulfilling all requirements. On the other hand, we had to remove or change some requirements during the project that we also considered applying the modifications to regarding test cases. After all, we believe that we provide full test coverage for all requirements.

6. Conclusion

As intent of the thesis, we explained why CRM application is so important for salespersons and should also be developed mobile versions to assists them when they are outside the office. With the strength of the mobile applications, they can be more productive. Besides we focused the attention into two most dominant mobile platforms, iOS and Android, in order to provide more diversity and flexibility.

Based on our research, we focus to obtain optimal solutions for each research questions. We also verified the report with taking consideration of several quality assurance criteria such as usability, security and performance. The performance of the applications was also significant that we optimized the communication process to avoid long synchronization time. Meanwhile we followed one of the iterative development processes to handle requirements in every stage of the development. In some cases could not apply best practices since we were restricted by marketing strategy and time limit. However we reported our findings to show advantages and drawbacks of each solution.

6.1. Mobile Application Development

Mobile application development was a new topic, which we have never got any experience before. We encountered different kind of problems, which are specifically related to mobile development. But the curiosity to the field motivates us in a good way so we almost overcame all of the problems that we stated before. Unfortunately there were few of them that needed more investigation but we had to skip because of time constraint.

Especially the variety of the operating system versions on mobile devices affects our thesis plan. We had to find the proper versions for both iOS and Android apps that will cover majority. Sometimes because of the version differences we were not able use new functionalities that are not supported in previous versions and in some situation the existing functionalities, which are deprecated with new version, do not perform at all. We had to apply different test cases for each versions to be sure of functionality. But because of the limited time we were able to test the main version of the platforms.

Another problem appeared almost at the end of the project. Initially we had designed our systems with small-scaled databases, which keep 500-600 records at most. Also our synchronization system was adapted to this amount. As always the requirement changed and suddenly we found ourselves in a big trouble. There were lots of memory overflow exceptions that we had to deal with. Then we gave a drastic decision, which caused to 30-40% of modification in code. Eventually we solved the problem in a smart way which also satisfied our supervisor.

6.2. Future Work

Nowadays, CRM applications are extremely huge projects that QBIS CRM is also one of the comprehensive web solutions which has several sub divisions and functionalities. Our requirements consisted of the basic functionalities with extending some mobile specific ones as well. Most of the sub divisions or less important functionalities had to be skipped because of the time limitation. Especially quotations and leads are essential parts of CRM, which were not implemented, in QBIS CRM mobile apps. Also we had to restrict some creation and deletion functionalities, which are not necessary for mobile apps. Of course they could be implemented in future depending on the user demand.

As we explained in first section, QBIS web solution consists of several sub modules that CRM is just one of these modules. Also our project was the first remarkable initiation among the mobile applications that they have been developed before. With our project the base domain structure has been constructed that if the company decide to develop mobile versions of other modules, the new applications can be easily adopted into system. Almost all web service methods have already been implemented and since we design the applications in a reusable manner, most parts can use in next developments. This was the one of the achievements when we start to design the system concerning software product line engineering.

7. References

- [Android Gestures]** Android Developers (2012). *Gestures*. [online]
<http://developer.android.com/design/patterns/gestures.html> (accessed June 20, 2012).
- [Android SDK Tools]** Android Developers (2012). *SDK Tools*. [online]
<http://developer.android.com/tools/help/index.html#tools-sdk> (accessed June 20, 2012).
- [Android Testing]** Android Developers (2012). *Testing Fundamentals*. [online]
http://developer.android.com/tools/testing/testing_android.html (accessed June 21, 2012).
- [Android Versions]** Android Developers (2012). *Platform Versions*. [online]
<http://developer.android.com/about/dashboards/index.html#Platform> (accessed June 20, 2012).
- [Apple iOS Dev]** Apple Developers Documentation (2011). *About iOS Development*. [online]
<http://developer.apple.com/library/ios/#documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/iPhoneOSOverview/iPhoneOSOverview.html> (accessed July 17, 2012).
- [Apple MobileHIG]** Apple Developers Documentation (2012). *iOS Human Interface Guideline*. [online]
<http://developer.apple.com/library/ios/DOCUMENTATION/UserExperience/Conceptual/MobileHIG/MobileHIG.pdf> (accessed March 05, 2012).
- [Apple MVC]** Apple Developers Documentation (2010). *The Model-View-Controller Design Pattern*. [online] <https://developer.apple.com/library/ios/-documentation/Cocoa/Conceptual/CocoaFundamentals/CocoaDesignPatterns/CocoaDesignPatterns.html> (accessed July 17, 2012).
- [Apple Technology]** Apple Developers Documentation (2011). *iOS Technology View*. [online]
<http://developer.apple.com/library/ios/DOCUMENTATION/Miscellaneous/Conceptual/iPhoneOSTechOverview/iPhoneOSTechOverview.pdf> (accessed March 05, 2012).
- [Apple Testing]** Apple Developers Documentation (2012). *Xcode Unit Testing Guide*. [online]
http://developer.apple.com/library/ios/#documentation/DeveloperTools/Conceptual/UnitTesting/00-About_Unit_Testing/about.html (accessed June 21, 2012).
- [Applied SPLE]** Kang, Kyo C., Vijayan Sugumaran, and Sooyong Park (2010). *Chapter 1 - Software Product Line Engineering: Overview and Future Directions*. *Applied Software Product-Line Engineering*. Auerbach Publications.
- [CocoaNetics]** Drobnik KG (2011). *iOS Versions in the Wild*. [online]
<http://www.cocoanetics.com/2011/08/ios-versions-in-the-wild/> (accessed June 02, 2012).

[Comparison Mobile] Hee-Yeon Cho, Choon-Sung Nam, Dong-Ryeol Shin (2010). *A Compariosn of Open and Closed Mobile Platforms*. IEEE Publication; ICEIE 2010, pg. V2-141 V2-143.

[Designing Agile for Mobile] Vahid Rahimian, Raman Ramsin (2008). *Designing an Agile Methodology for Mobile, Software Development: A Hybrid Method Engineering Approach*. IEEE Publication; RCIS 2008, pg. 337-342.

[Doc Software Arch] Clements, Paul; Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Paulo Merson, Robert Nord, Judith Stafford (2010). *Documenting Software Architectures: Views and Beyond, Second Edition*. Boston: Addison-Wesley. ISBN 0-321-55268-7.

[Frakes and Kang] Frakes, W. B., and K. C. Kang. 2005. *Software reuse research: Status and future*. *IEEE Transactions on Software Engineering* 31 (7): 529–536.

[Gartner] Gartner (2011). *Gartner Says Sales of Mobile Devices in Second Quarter of 2011 Grew 16.5 Percent Year-on-Year; Smartphone Sales Grew 74 Percent*. [online] <http://www.gartner.com/it/page.jsp?id=1764714> (accessed June 02, 2012).

[Hallsteinsen et al. 2008]. Hallsteinsen, S., M. Hinchey, S. Park, and K. Schmid. 2008. Dynamic software product lines. *IEEE Computer* 41 (4): 93–95.

[Krueger] Krueger, C. 2002. Eliminating the adoption barrier. *IEEE Software* 19 (4): 29–31.

[Metro] Metro Business (2011). *Din bostadsort styr valet av mobiltelefon*. [online] <http://www.metro.se/nyheter/din-bostadsort-styr-valet-av-mobiltelefon/EVHkee!jdl5r8aLCHJOw/> (accessed June 02, 2012).

[Mobile App Challenges] Mobile Application Software Engineering: Challenges and Research Directions, Josh Dehlinger and Jeremy Dixon, Department of Computer and Information Sciences Towson University

[Mobile CRM] Giovanni Camponovo , Yves Pigneur , Andrea Rangone , Filippo Renga (2005). *Mobile Customer Relationship Management: An Explorative Investigation of the Italian Consumer Market*. IEEE Publication; ICMB 2005, pg. 42-48.

[Mobile Doc] Marta Rauch (2011). *Mobile documentation: Usability guidelines, and considerations for providing documentation on Kindle, tablets, and smartphones*. IEEE Publication; IPCC 2011, pg. 1-13.

[Mobile Web] Fahad Aijaz, Syed Zahid Ali, Muzzamil Aziz Chaudhary, Bernhard Walke (2010). *The Resource-Oriented Mobile Web Server for Long-Lived Services*. IEEE Publication; WiMob 2012, pg. 7.

[MobileTest] MobileTest: A Tool Supporting Automatic Black Box Test for Software on Smart Mobile Devices, Jiang Bo, Long Xiang, Gao Xiaopeng, Second International Workshop on Automation of Software Test (AST'07) 0-7695-2971-2/07 \$20.00 © 2007, IEEE

[OMG] Object Management Group (2003). *Introduction to OMG's Unified Modeling Language*. [online] <http://www.omg.org> (accessed July 17, 2012).

[Pragyaan_IT_June2011] Pragyaan : Journal of Information Technology, Volume 9 : Issue 1. June 2011 , Institute of Management Studies Dehradun, Sanjay Singh, Rudra Pratap Singh Chauhan

[Polling Wiki] http://en.wikipedia.org/wiki/Polling_%28computer_science%29, accessed on 30/Mar/2012

[Pro-Android] Sayed Y. Hashimi and Satya Komatineni, 2009, Apress, Inc.

[Product Line Use Cases] Software Product Lines, Springer Berlin Heidelberg, 2006, Chapter Product Line Use Cases: Scenario-Based Specification and Testing of Requirements, Antonia Bertolino, Alessandro Fantechi, Stefania Gnesi and Giuseppe Lami

[Pushing Wiki] [http://en.wikipedia.org/wiki/Push_technology] 30/Mar/2012, accessed on 30/Mar/2012

[Rational] Rational: the software development company (1998). *Rational Unified Process: Best Practices for Software development Teams*. Rational Software White Paper TP026B, Rev 11/01

[RE-Good Practice] Sommerville, Ian; Sawyer, Pete (1997). Requirements Engineering - A Good Practice Guide.. John Wiley & Sons.

[RE_NESEIBEH] Requirements Engineering: A Roadmap, Bashar Nuseibeh & Steve Easterbrook, ICSE '00 Proceedings of the Conference on The Future of Software Engineering, Pages 35 - 46, ACM New York, NY, USA ©2000

[SmartDraw] SmartDraw Software, LLC
<http://www.smartdraw.com/resources/tutorials/> (accessed 2012-11-03).

[Soft Arch In Practice] Bass, Len; Paul Clements, Rick Kazman (2003). *Software Architecture In Practice, Second Edition*. Boston: Addison-Wesley. pg. 21–24. ISBN 0-321-15495-9.

[Software Engineering Issues for Mobile App] Software Engineering Issues for Mobile Application Development, Anthony I. Wasserman Carnegie Mellon Silicon Valley, FoSER '10 Proceedings of the FSE/SDP workshop on Future of software engineering research Pages 397-400, ACM New York, NY, USA ©2010

[Software Portability] Ken Garen (2007). *Software Portability: Weighing Options, Making Choices*. The CPA Journal; Nov 2007; 77, 11; ABI/INFORM Global, pg. 10

[SPLE] Klaus Pohl, Günter Böckle and Frank van der Linden (2005). *Software Product Line Engineering, Foundations, Principles, and Techniques*. Springer.

[TDD] Russell Gold, Thomas Hammell and Tom Snyder (2005). *Test Driven Development: A J2EE Example*. Apress 2005 (296 pages) Citation, ISBN 9781590593271.

[TDD in large projects] Raghvinder S. Sangwan and Phillip A. Laplante (2006). *Test-Driven Development in Large Projects*. IT Pro.

[TDD J2EE] Russell Gold, Thomas Hammell and Tom Snyder (2005). *Getting Started, Test Driven Development: A J2EE Example*. Apress © 2005 Citation

[The RUP] The Rational Unified Process: An Introduction By Philippe Kruchten, 3rd Edition, Library of Congress Cataloging-in-Publication Data, Addison-Wesley.

[Top CRM] Business Software (2011). Top 40 CRM Software Vendors are revealed 2011.

[UML Forum] UML Forum. "<http://www.umlforum.com/faq/>". (accessed 2012-11-03).

[Unit Test] Paul Solt (2010). *iPhone Unit Testing Explained - Part 1*. [online] <http://paulsolt.com/2010/11/iphone-unit-testing-explained-part-1/> (accessed June 21, 2012).

[Using UML] Using UML and Agile Development Methodologies to Teach Object-Oriented Analysis & Design Tools and Techniques, Jeffrey Brewer, Leslie Lorenz, CITC4 '03 Proceedings of the 4th conference on Information technology curriculum, Pages 54 - 57, ACM New York, NY, USA ©2003

[Yves] A relationship between sequence and statechart diagrams. Yves Dumond, Didier Girardet , Flavio Oquendo, LLP/CESALP Laboratory – University of Savoy