

**PHONE CALLS AND TEXT MESSAGES CONTROL
FOR ANDROID PHONES**

BY

CHAN JIA HUI

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfilment of the requirements

for the degree of

BACHELOR OF INFORMATION SYSTEMS (HONS)

INFORMATION SYSTEMS ENGINEERING

Faculty of Information and Communication Technology
(Perak Campus)

January 2013

DECLARATION OF ORIGINALITY

I declare that this report entitled “**PHONE CALLS AND TEXT MESSAGES CONTROL FOR ANDROID PHONES**” is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature : _____

Name : Chan Jia Hui

ID No. : 10 ACB 06386

Date : 5th April, 2013

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and great appreciation to my supervisor, Mr. Liew Shiuh Deh, who has given me this bright opportunity to engage in an Android application development project. Thanks for his guiding through the project, stimulating suggestions and correcting various documents of mine with much attention and care. It is my first step to establish a career in the mobile application field. A million thanks to you.

My special thanks to a very special person in my life, Yi-Hoong Lau, for her patience, unconditional support and love, and for standing by my side during my hard times. Finally, I must say thanks for my parents and my family for their concern, support and continuous encouragement throughout the course.

I am making this project done not only for academic marks but to also proliferate my knowledge in this field area. Thanks again to all who helped me laterally in this project.

ABSTRACTS

This project is an Android mobile application development project for academic purpose. In telephony, call forwarding is a feature provided by mobile network operators to enable the mobile phone users to transfer or redirect the incoming calls to another phone under several conditions. The primary aim of this project is to allow the users of Android-powered mobile phone to have the ability to control the incoming calls and text messages according to their needs. A detailed comparison of the existing solutions and the mobile network operators was performed. It is shown that in some countries, e.g. Malaysia, most of the mobile network operators do requiring their users additionally pay for a subscription fee by monthly for using this service. They might allow us to forward any calls that call-in to our mobile phone to another mobile phone, but unfortunately we might not be able to select which call to forward, and which one should be dropped. Besides, the majority of the mobile network operators only forward calls but do not forward text messages. It is important where sometimes we do receive a few unwanted calls or messages because our mobile phone number has been disclosed to unwanted people or some other reasons. This project is to develop an application to transform an existing Android-powered mobile phone to become an intermediate point that allows the users to control every incoming calls and text messages, using it to screen and filter the incoming calls and messages before the phone notify the user. Moreover, it will help the users to pick-up the incoming calls automatically for driving safely. It will also allow the users to activate the forwarding function or changing the other settings remotely by sending a text message to the mobile phone with some available commands. The output material of the project would be the Phone Calls and Text Messages Control application for Android-powered mobile phones.

TABLE OF CONTENTS

TITLE	i
DECLARATION OF ORIGINALITY	ii
ACKNOWLEDGEMENTS	iii
ABSTRACTS	iv
TABLE OF CONTENTS	v
LIST OF FIGURES	viii
LIST OF TABLES	x
LIST OF ABBREVIATIONS	xi
CHAPTER 1 INTRODUCTION	1
1.1 Motivation	1
1.2 Problem Statement	3
1.3 Project Scope	4
1.4 Project Objectives	5
1.5 Impact, Significance and Contribution	7
1.6 Background Information	8
CHAPTER 2 LITERATURE REVIEW	10
2.1 Android Architecture	10
2.2 Java and Dalvik Virtual Machine	12
2.3 Stack-based VM vs. Register-based VM	16

2.4	Processes and Threads	17
2.5	Reviewing the Existing Solution	19
2.6	Comparison with the Mobile Network Operators	21
CHAPTER 3	METHODOLOGY	24
3.1	Methodology and Tools	24
3.2	Requirement Specifications	26
3.3	Implementation and Testing	27
CHAPTER 4	PHONE CALLS AND TEXT MESSAGES CONTROL APPLICATION	28
4.1	Overall Application Architecture	28
4.2	Application Icon	30
4.3	Version Release History	32
4.4	User Interface Design	34
4.5	Service: Call Manager and SMS Manager	37
4.6	Function: Call Forwarding	39
4.7	Function: Call Blocking	43
4.8	Function: Auto Answer	49
4.9	Function: SMS Forwarding	51
4.10	Function: SMS Blocking	57
4.11	Function: Remote Control	60
CHAPTER 5	TESTING AND RESULTS	65
5.1	Test Cases for Call Manager & SMS Manager	65
5.2	Test Cases for Call Blocking	66
5.3	Test Cases for Auto Answer	71
5.4	Test Cases for SMS Forwarding	73

5.5	Test Cases for SMS Blocking	76
5.6	Test Cases for Remote Control	79
5.7	Summary	82
CHAPTER 6	CONCLUSION AND DISCUSSION	83
6.1	Project Review	83
6.2	Problems Encountered	85
6.3	Future Work	86
6.4	Conclusion	87
REFERENCE		88
APPENDIX A	GLOSSARY	A-1
APPENDIX B	ANDROID PLATFORM VERSIONS	B-1
APPENDIX C	BIWEEKLY REPORTS AND OTHER ATTACHED DOCUMENTS	C-1

LIST OF FIGURES

Figure Number & Title	Page
Figure 2.1.1: Android system architecture.	10
Figure 2.2.1: Application Virtual Machine for platform independence.	12
Figure 2.2.2: Comparison between Java Class File Format and Dalvik Executable File Format.	15
Figure 2.4.1: The Dalvik VM is the intermediate layer of the Linux process and the Android application.	17
Figure 2.6.1: Illustrating how call charges apply when a call is forwarded to another number.	22
Figure 3.1.1: Incremental Life Cycle Model.	24
Figure 4.1.1: Overall application architecture for Phone Calls and Text Messages Control Application.	28
Figure 4.4.1: User interface displayed on Android version 2.3.	34
Figure 4.4.2: User interface displayed on Android version 3.2.	35
Figure 4.4.3: User interface displayed on Android version 4.0.	35
Figure 4.5.1: Notification for each individual service is shown to indicate the particular service is turned on.	37
Figure 4.6.1: The overview of the process of Call Forwarding function.	39
Figure 4.6.2: MODIFY_PHONE_STATE permission has been reserved for system apps in newer SDK.	40
Figure 4.7.1: The user interface of Call Blocking.	44
Figure 4.7.2: Users can choose to allow only receiving calls from contacts, favourites or block all the calls.	45
Figure 4.7.3: An individual screen to let the users to manage the blacklist and display the list of blocked phone numbers.	46
Figure 4.7.4: A confirmation dialog when a user trying to clear the blacklist.	48
Figure 4.8.1: The user interface of Auto Answer.	49

Figure 4.8.2: Delay Timer in list selection.	50
Figure 4.9.1: The overview of the process of SMS Forwarding function.	51
Figure 4.9.2: SMS content in different mobile phones for SMS Forwarding.	52
Figure 4.9.3: The user interface of SMS Forwarding	53
Figure 4.9.4: Tutorial generated for the users to know how to use the SMS Forwarding.	56
Figure 4.10.1: The user interface of SMS Blocking.	58
Figure 4.10.2: Several actions towards blocked SMS for users to choose.	59
Figure 4.11.1: A scenario's illustration for Remote Control.	60
Figure 4.11.2: The user interface of Remote Control.	63
Figure 4.11.3: List of commands with respective usage for Remote Control.	64
Figure B.1: Distribution of the Android Platform Versions.	B-1

LIST OF TABLES

Table Number & Title	Page
Table 2.2.1: Minimum Device Requirements for Android Devices.	14
Table 2.4.1: Process life cycle in Android. The five levels in the importance hierarchy.	18
Table 2.6.1: Comparing the “call forwarding” service from different mobile network operators.	21
Table 2.6.2: Comparing the family plan from different mobile network operators.	23
Table 4.2.1: The application icon demonstrated in two different background colours.	30
Table 4.2.2: Brief explanation on each object used in the application icon.	31
Table 4.7.1: Consequences for the blocked callers and allowed callers when Call Blocking is enabled.	43
Table 4.10.1: Consequences for the blocked callers and allowed callers when SMS Blocking is enabled.	57
Table 4.11.1: Comparison of different verification methods for Remote Control.	61
Table 5.1.1: Black Box Test Result (Part 1).	65
Table 5.2.1: Black Box Test Result (Part 2).	66
Table 5.3.1: Black Box Test Result (Part 3).	71
Table 5.4.1: Black Box Test Result (Part 4).	73
Table 5.5.1: Black Box Test Result (Part 5).	76
Table 5.6.1: Black Box Test Result (Part 6).	79
Table B.1: Distribution of Android devices that have accessed Google Play, ending on 4 March, 2013.	B-2

LIST OF ABBREVIATIONS

<i>ADT</i>	Android Development Tools
<i>API</i>	Application Programming Interface
<i>AVD</i>	Android Virtual Devices
<i>CPU</i>	Central Processing Unit
<i>GSM</i>	Global System for Mobile Communications
<i>HP-UX</i>	Hewlett Packard UniX
<i>I/O</i>	Input/output
<i>JVM</i>	Java Virtual Machine
<i>OS</i>	Operating System
<i>QVGA</i>	Quarter Video Graphics Array
<i>SDK</i>	Software Development Kit
<i>SMS</i>	Short Message Service
<i>UI</i>	User interface
<i>USB</i>	Universal Serial Bus
<i>VM</i>	Virtual Machine

CHAPTER 1 INTRODUCTION

1.1 Motivation

In the modern days, getting a new mobile phone and a new cell phone line (either prepaid or postpaid call plan) are much easier and cheaper compared to the past few years ago. It is getting more common that people are having two or more mobile phones for themselves, each of the mobile phones with an individual cell phone line or number.

The reasons of people having multiple phone numbers might vary, some might want to use different phone numbers for different group of contacts; some might want to have one phone number for each of the mobile network operators (also known as carriers), to communicate with those contacts which are using the same carrier; some might want to use one of them as the public contactable number, whereas keeping another one privately for security and privacy concerns.

With the rising of Apple iPhone and various kinds of Android-powered devices, people have started to move forward from the traditional mobile phone into the Smartphone era in a very fast pace. Because of the trend is changing, Android phones are getting more and more popular nowadays. Android is an open-source operating system for mobile devices and it has many features, including all the essential telephony features.

However, some of the telephony features are requiring the services offered by the users' mobile network operators or carrier, those services are now commonly known as Value Added Services. Without relying on the carrier, we are not able to forward the incoming calls and SMS (Short Message Service) when necessary.

In addition, we are required to contact the customer service in order to request for blocking certain numbers. This is very inconvenient to us as a mobile phone user, especially if we just want it to be temporary, because sometimes we might only want to allow calls or text messages from those people who are in the saved contacts list, or only those grouped in the favourite list are allowed during the meeting.

For privacy concerns, we can choose to disclose one of our phone number to the public, then using the call forwarding and SMS forwarding features to help us to conceal our private phone number, and then make use of the filtering feature to control the unwanted incoming calls and text messages, all these need to be done using one Android mobile phone.

With the deliberation of all these issues, I am become more inspired and enthused to take this project as a challenge.

1.2 Problem Statement

The usage of mobile phone is kept rising (mobiThinking, 2012), sooner the security and privacy issues also arise where we often receive phone calls or text messages (also known as SMS) from unwanted people; one of the reasons is because of our friends had disclosed our phone number to them, either intentionally or unintentionally.

Generally, “call forwarding” or “call diverting” is a feature or additional service provided by the mobile network operators or carriers, a subscriber can active this feature to temporarily transfer all incoming calls from his/her current mobile phone number to another phone number. The subscriber can choose to activate the call forwarding only if the line is busy, or if there is no answer, or even only for incoming calls from selected numbers (Wikipedia contributors, 2003).

In fact, this feature is depending on the carriers and the type of plan subscribed by the user, so it may not be available for every phone. Besides that, not all mobile operators do support forwarding, as well as forwarding with filtering enabled. So sometimes we might wish there is a tool that can help us to filter the incoming calls or messages before forwarding it to our private phone number without relying on the carriers.

In another scenario, most of us are aware that using mobile phone while driving is illegal and we are supposed to use hands-free devices, i.e. Bluetooth headphones. In most of the countries, hands-free are allowed, but in Singapore, hands-free is allowed provided that the hands are not used to answer the call, i.e. voice control/auto-pickup (Singapore Police Force, 2012). So it will be convenient if there is a tool that able to answer the incoming calls automatically without the user’s reaction.

As a result, there is a need for such an application that can perform screening or filtering on the phone calls and text messages in order to resolve these issues. After screening, this application can help the users either forwarding incoming calls or text messages without relying on the mobile network operators, and bundled together with some other useful features.

1.3 Project Scope

This project is to deliver an application for Android phones to control the phone calls and text messages in the final stage, which allows all incoming phone calls or text messages to be filtered and forwarded to a user's preferred phone number when the selected function has been activated.

In this application, a foreground program or user interface will be produced for settings and configuration. Every instruction should be easily understandable by the users. Next, there will be two service programs that run in the background on the Android phones when certain functionality has been enabled by the users, one is for monitoring all the incoming calls, whereas another one is for monitoring all the SMS.

Besides that, this project is also useful to car drivers whereby it can pick-up the incoming calls automatically without any user reaction. Therefore when the users are driving the car, they can also pick-up and answer to the incoming calls in a safe manner, because they do not need to leave their hands away from the steering wheel.

For secrecy purpose, this application is suggested to build-in with a blacklist/whitelist feature for the users, so that it can help the users to block or allow certain phone numbers in the user-defined list, especially the way before forwarding the incoming calls or text messages.

In addition, users will be able to send SMS from any other mobile phone to the mobile phone that installed with this application, to change the settings remotely, provided with the matched security pin. For instance, a user leaves his/her mobile phone at home, he/she may want to turn on the SMS Forwarding feature so that he/she will not miss out any incoming SMSs later. This application will make use of text recognition technique to check the command in the SMS's content.

All the individual features in this application are independent, but they are compatible to use with one another at the same time, so that the users can choose to activate only those required ones.

1.4 Project Objectives

The objective of developing the Phone Calls and Text Messages Control application for Android phones is to allow forwarding not only phone calls, it also forward the text messages to a user-predefined phone number, with some additional features such as filtering and remote controlling to make it more multipurpose.

As mentioned earlier, call forwarding is a feature that depends on the mobile network operators, not every kind of subscription plans do provide it. This project aims to simplify the users into controlling the received phone calls and the text messages, and to reduce the users' dependency on their mobile phone operator. For text messages, this project aims to achieve a two-ways forwarding, so that a user not only can receive but also can reply to the forwarded SMS by going through another mobile phone.

This project will be focused on developing an Android application that can used to perform call and SMS forwarding using just one mobile phone, notwithstanding the carriers do support this functionality or not. This project requires a mobile phone with Android OS (Operating System) to act as an intermediate point to receive, and forward in an automated manner.

Besides that, the phone numbers filtering feature will be developed with both blacklist and whitelist types of checking. This feature can be used as an additional protection layer together with the other bundled features as well as using it individually. It makes use of the database service in the Android platform to keep the records of phone numbers and retrievable for filtering purposes, the incoming calls or text messages will be dropped if it is sent from one of the blacklisted phone numbers.

“Each day, more than 15 people are killed and more than 1,200 people are injured because of distracted driving”, as mentioned in the analysis (Centers for Disease Control and Prevention, 2012). Hence, this project also aims to reduce the needs of car drivers divert their attention and to use their hands to pick-up the incoming calls while driving on the road, this act will actually augment the rate of accidents happen. This consequence can be reduced by developing an automated call answering feature to this application, so that the users do not need to manually answer the phone calls by lifting their hands away from the steering wheel.

Furthermore, a remote controlling feature by using SMS will be developed to enhance the usability of the whole application, it uses the text recognition technique to recognise the command given in the received SMS and perform the designated action. The users can make use of this functionality to change some settings of this application, such as turning on the forwarding feature, changing the destination number, and etc., by using any mobile phone to send a text message to the mobile phone that is running with this application, provided with the predefined security pin and command, only the person with correct security pin has the right to change those settings.

Lastly, in order to make all these working, this application should contain two service programs that run in the background of the Android system. The service programs are used to receive the broadcast from the system whenever there is an incoming calls or text messages.

1.5 Impact, Significance and Contribution

By having the Phone Calls and Text Messages Control application, the Android mobile phone users are able to control the incoming calls and text messages according to their needs and situations. This application is aimed to help its users to forward the incoming calls and messages from one mobile phone number to another, which can support most of the Android phones; the most important thing is to eliminate the dependency on the mobile network operators. Refers to Appendix B for the trend of Android platform versions.

For instance, some of the users would prefer not wanted to show their personal mobile phone number to the public, this application will be able to forward the incoming calls and/or text messages from his/her public mobile phone number to the personal mobile phone. In addition, they can also choose to make use of the filtering feature to block the unwanted calls or messages before the application starts forwarding it.

Moreover, this application can help the car drivers to automatically pick-up the incoming calls without the users reacting to it while driving. This can be significantly helped to decrease the high chances of initiating an accident with the cause of distracted driving. Users can choose to make use of the filtering feature to select only automatically pick-up calls from a certain number of people.

1.6 Background Information

Android is an operating system for mobile devices initially developed for smart phones and now it comes to the tablet computers. It is developed by a group of companies, known as the Open Handset Alliance, led by Google (Google, n.d.).

In order to extend the functionality of the devices, Android has the ability of letting the developers writing applications (known as “apps”) for it, primarily in Java and Google-based Java libraries (Resco Developer Tools, 2011). Google launched an application marketplace for its Android users, formerly named Android Market, now renamed Google Play (<https://play.google.com/>), with more than 638 thousand of Android apps available to the users in 18th of February, 2013 (AppBrain, 2013).

This is a development-based project that requires the basic knowledge of the Java programming language and how to make use of the libraries that provided in the Android SDK (Software Development Kit). A SDK is typically a set of software development tools that allows for the creation of applications for a certain software package, software framework, hardware platform, computer system, video game console, operating system, or similar platform (Wikipedia contributors, 2012).

As described in the Android Developers website (Google, n.d.), the Android SDK provides the tools and necessary APIs to allow developers to start developing applications on the Android platform using the Java programming language. Inside this website, we can also learn more in-depth on how to develop an Android application.

The reason of why Android is using Java as its primary language is because of there is a large community of developers and it is relatively easy to use compared to other mobile programming languages such as C and C++ (Maker, 2011).

In order to start to develop an Android application, we first need to install the Eclipse IDE (Integrated Development Environment), a tool that ease the developing process for Java Developers, it can be downloaded at <http://www.eclipse.org/downloads/>. The Java Development Kit (JDK) should be already installed before installing the Eclipse IDE, the latest version of JDK can be downloaded at <http://java.sun.com/javase/downloads/index.jsp>.

After installing the Eclipse IDE, then continue with downloading and installing the Android SDK (<http://developer.android.com/sdk/index.html>), together with the Android Development Tools (ADT) plugin (<http://developer.android.com/sdk/eclipse-adt.html>) into the Eclipse IDE. For more details on how to install the Android SDK, we can refer to <http://developer.android.com/sdk/installing.html>.

CHAPTER 2 LITERATURE REVIEW

2.1 Android Architecture

Android operating system (OS) is basically known as a software stack. Each layer of the stack groups together several programs that support specific OS functionalities (Strickland, 2007).

Figure 2.1.1 shows the major components of the Android OS, it has been subdivided into five layers for categorising each components.

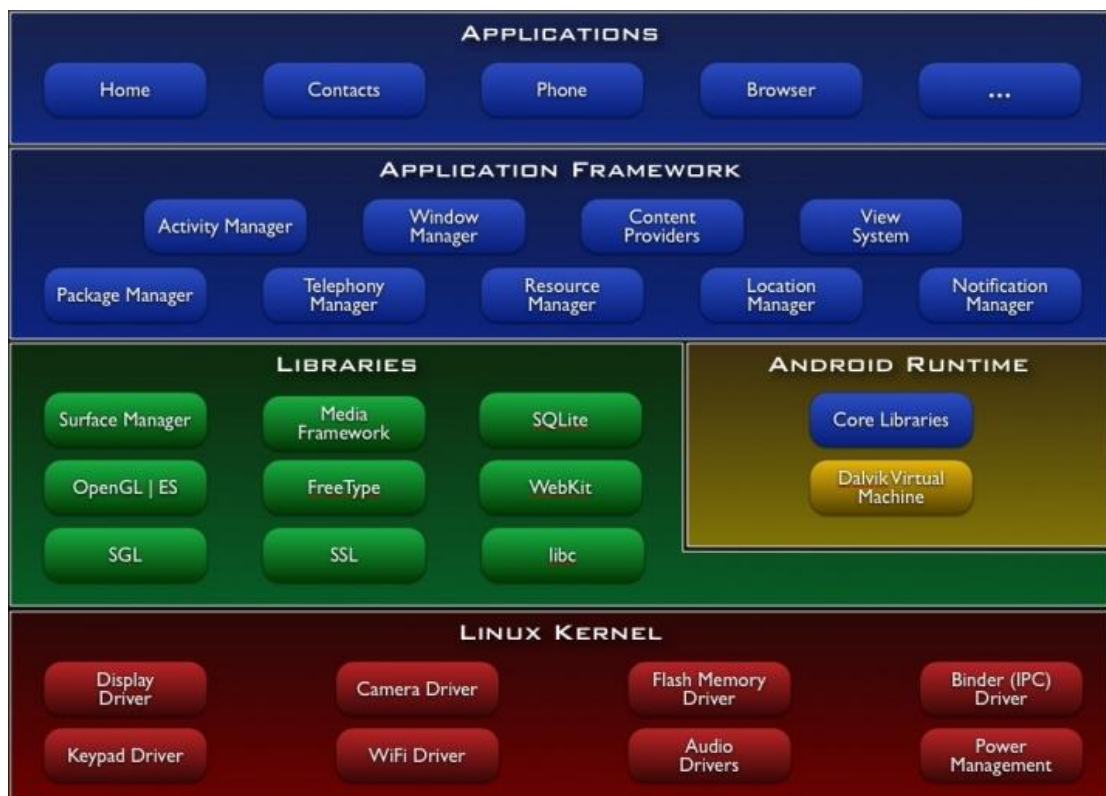


Figure 2.1.1: Android system architecture.

(Source: <http://developer.android.com/about/versions/index.html>)

The layer in green colour, which contains libraries, is mainly written in C and C++, the native libraries that contain daemons and services (Bird, 2009), including the graphics libraries, they are encapsulated in a higher-level Java API (Hashimi, et al., 2010), thus they can be called through the Java interface, which are the application

layer and application framework layer that in blue colour. These two layers are written in Java and run in the Dalvik Virtual Machine (Dalvik VM) (Brahler, 2010).

The Android runtime layer that placed on the same level as the libraries layer includes a set of core Java libraries, including the Dalvik VM (Strickland, 2007). Android uses virtual machines to run each of the application as its own process (Strickland, 2007), so that each application can run independently and it simplifies memory management process.

In order to make an OS completely works together and perform actual data processing with the physical hardware, it needs the lowest level of easily replaceable software that can interfaces with the hardware (Garrison, 2010), which is the kernel. Android relies on Linux version 2.6 series kernel, modified for core system services such as security, memory management, process management, networking, and driver model (Google, n.d.).

The Linux kernel does have some advantages that make many OS using it. Linux kernel is one of the largest open source projects in the world (Garrison, 2010), its advantages including portability, better security, small install footprint and memory footprint.

2.2 Java and Dalvik Virtual Machine

Java is a high-level object-oriented programming language that first developed by the Sun Microsystems that wasn't tied to any particular operating system or microprocessor (Rial & Rodríguez Silva, n.d.).

In the early stage, it was only developed keeping in mind the consumer electronics and communication equipments. In the 1990s, it came into existence as a part of web application, web services and a platform independent programming language. In the year 1991, it was released with the name called Oak, but later in the year 1995, it was renamed as Java and then Java 1.0 was officially released to the world (Rose India Technologies Pvt. Ltd., 2007).

Java is influenced by C, C++, Smalltalk and it also borrowed some other programming languages; it solves the limitation of C++ where it is not platform independent and need to be recompiled for each different CPUs (Central Processing Unit) (Rose India Technologies Pvt. Ltd., 2007).

Android applications are usually written in the Java programming language, similar with the normal Java applications, they need to run within a virtual machine (VM). In the application virtualisation space, VMs are used to provide a platform independent environment for the execution of application (Jones, 2011). Figure 2.2.1 shows a clear illustration on the differences of application execution process in VM environment and non-VM environment.

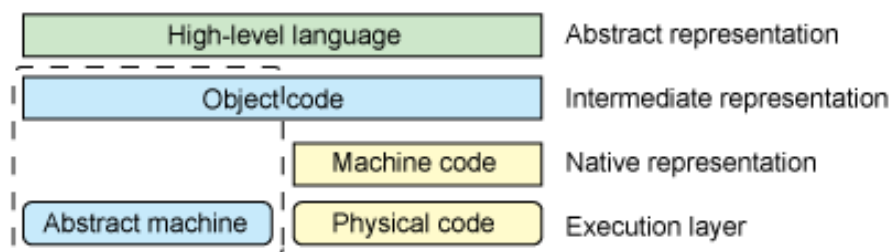


Figure 2.2.1: Application Virtual Machine for platform independence.

(Source: <http://www.ibm.com/developerworks/linux/library/l-virtual-machine-architectures/>)

Developers using the simpler high-level programming language to construct applications, compared to low-level languages like C, which is a lot more complicated. After the compilation process, this high-level code is compiled into an intermediate representation, called object code (Jones, 2011). In a non-virtualised environment, this machine independent's object code is compiled into the native machine code for execution on the physical hardware platform (Jones, 2011). But in an application virtualisation environment, the object code is interpreted within an abstract machine to provide the execution. With the advantages of application VM, the same object code can be executed on any hardware platform that supports the abstract machine (or the interpreter) (Jones, 2011).

The advantage of Java is that it can be developed and run on any device that installed with Java Virtual Machine (JVM). This programming language is applicable in all kinds of operating systems including Windows, Linux, Solaris, HP-UX, and etc. (Rose India Technologies Pvt. Ltd., 2007)

Unlike Java, Android does not use JVM, but is an alternative deployment target called the Dalvik VM. It is a non-standard VM developed by Google for the Android OS, an interpreter for byte code that has been transformed from Java byte code into Dalvik byte code. Dalvik itself is compiled into native code whereas the core libraries in the Android Runtime are written in Java (as shown in Figure 2.1.1), thus interpreted by Dalvik (Brahler, 2010).

After an Android application is developed using Java's syntax, the Dalvik VM first takes the generated Java class files and then combines them into one or more Dalvik Executable (.dex) files. It reuses duplicate information from multiple class files, effectively reducing the space requirement by half from a traditional Java Archive (.jar) file without compressing it (Sayed, et al., 2010).

One of the main purposes of why Google chooses to use Dalvik VM is because of the Dalvik Executable (.dex) format is optimised for minimal memory footprint, in order to overcome the minimum device requirements (refers to Table 2.2.1) set for all the Android devices.

Table 2.2.1: Minimum Device Requirements for Android Devices.

Feature	Minimum Requirement
Chipset	ARM-based
Memory	128 MB RAM; 256 MB Flash External
Storage (optional)	Mini or Micro SD
Primary Display	QVGA TFT LCD or larger, 16-bit colour or better
Navigation Keys	5-way navigation with 5 application keys, power, camera and volume controls
Camera (optional)	2 Megapixel/2MP CMOS
USB	Standard mini-B USB interface
Bluetooth (optional)	1.2 or 2.0

(Source: http://www.netmite.com/android/mydroid/development/pdk/docs/system_requirements.html)

The Android platform was created for devices with limited processing power, memory and storage, even though nowadays most of the smart phones have already exceeded these minimum recommendations and getting more powerful.

In order to achieve minimal memory footprint for the Android applications, Dalvik provides the “dx” tool to combines all the compiled Java Class (.class) files into a Dalvik Executable (.dex) file, which means a Dalvik Executable (.dex) file contains multiple classes (Chengalva, 2012).

Besides of memory usage optimisation, the design of Dalvik Executable (.dex) file is primarily driven by sharing of data (Chengalva, 2012) among multiple classes. The Figure 2.2.2 below contrasts the Java Class (.class) file format used by the JVM with the Dalvik Executable (.dex) file format used by the Dalvik VM.

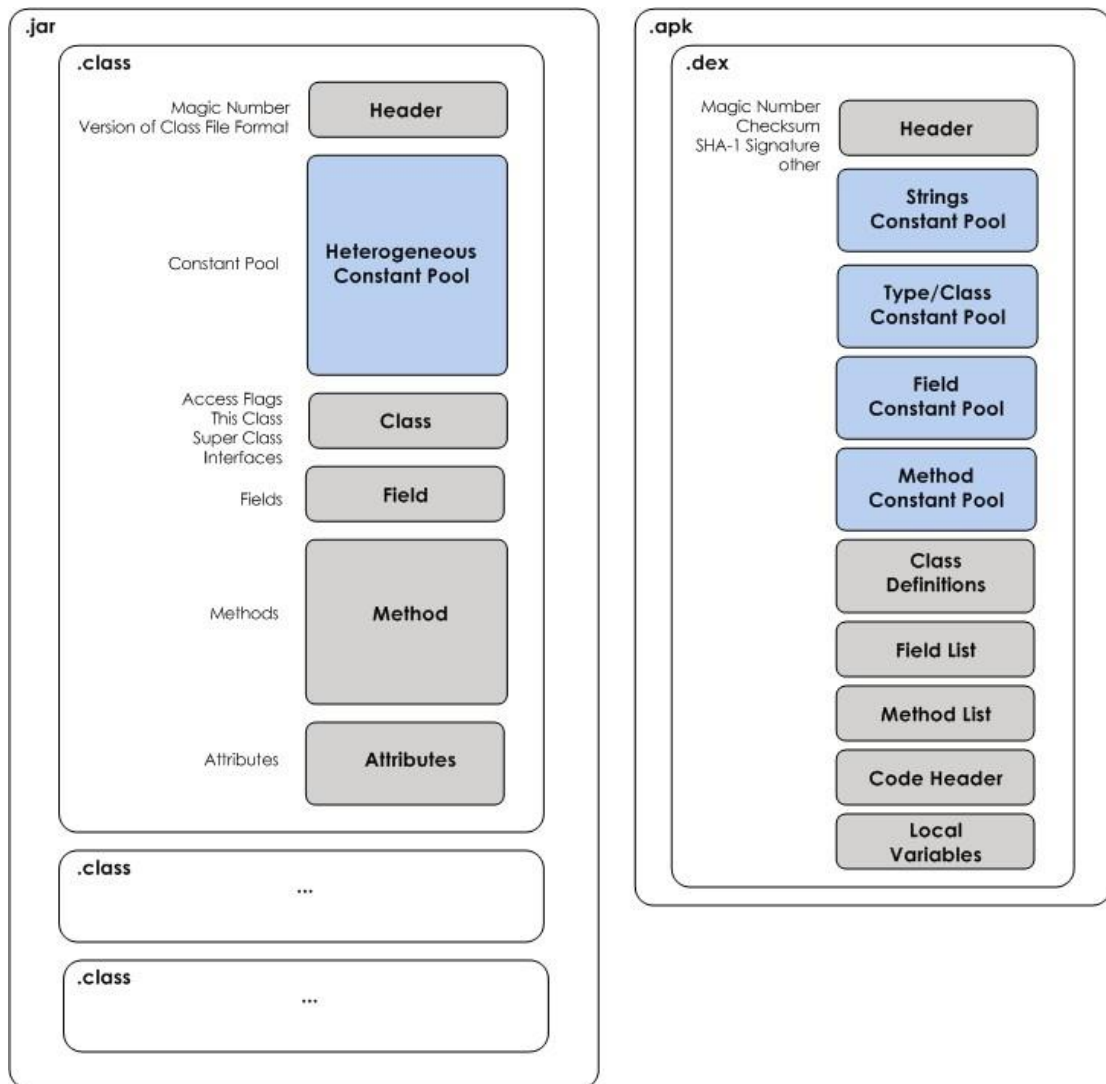


Figure 2.2.2: Comparison between Java Class File Format and Dalvik Executable File Format.

(Source: <http://chengalva.com/2012/10/29/android-dalvik-virtual-machine-dvm-android-runtime/>)

Based on Figure 2.2.2 above, we can clearly know that the Dalvik Executable (.dex) file format uses shared and type-specific constant pools as its primary mechanism for conserving memory. The constant pools in the Java Class and Dalvik Executable files are highlighted in blue colour above. In the Java Class file, each class has its own private, heterogeneous constant pool. Duplication of constants across Java Class files are eliminated in the Dalvik Executable file format (Chengalva, 2012).

2.3 Stack-based VM vs. Register-based VM

There are plenty of differences among the JVM and Dalvik VM; the JVM is stack-based VM, whereas the Dalvik VM is a register-based VM (Google, n.d.; Jones, 2011). Dalvik VM was designed and written by Dan Bornstein with contributions from other Google engineers. It is optimized for low memory requirements, and is designed to allow multiple VM instances to run at once, relying on the underlying OS for process isolation, memory management and threading support (Kumar, 2011).

In tradition, most of the VMs are implemented using stack-based architectures rather than register-based architectures, due to the simplicity and produce smaller executable size.

Studies have been done in (Shi, et al., 2005) and significantly proven that the execute time for register code is faster than stack code, although the register machine byte code (or object code) size could be larger than that of the corresponding stack byte code in some circumstances.

The register-based Dalvik VM uses a different kind of assembly-code generation, where it uses registers as the primary units of data storage, instead of using the stack. Google is hoping to accomplish 30 percent fewer instructions as a result (Hashimi, et al., 2010).

2.4 Processes and Threads

As mentioned earlier, each Android application runs in its own Linux process, with its own instance of the Dalvik VM (Google, n.d.), as illustrated in the Figure 2.4.1 below.

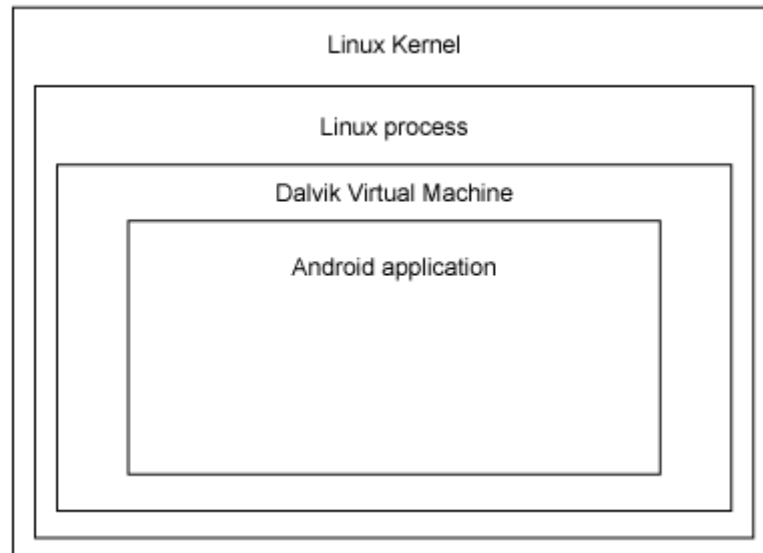


Figure 2.4.1: The Dalvik VM is the intermediate layer of the Linux process and the Android application.

(Source: <http://www.ibm.com/developerworks/library/os-android-devel/>)

Most of the time Android does needs to remove old processes to reclaim memory for new or more important processes. To determine which processes to keep and which to kill, the system places each process into an "importance hierarchy" (refers to Table 2.4.1) based on the components running in the process and the state of those components. Processes with the lowest importance will be eliminated first, following by those with the next lowest importance, as necessary to recover system resources (Google, n.d.).

Table 2.4.1: Process life cycle in Android. The five levels in the importance hierarchy.

Level of Importance	Types of Processes	Description
Highest	Foreground	A process that is running an <i>Activity</i> , a <i>Service</i> providing the <i>Activity</i> , a starting or stopping <i>Service</i> or a currently receiving <i>BroadcastReceiver</i> .
	Visible	If a process holds a paused but still visible <i>Activity</i> or a <i>Service</i> bound to a visible <i>Activity</i> and no foreground components, it is classified a visible process.
	Service	A process that executes an already started <i>Service</i> .
	Background	An <i>Activity</i> that is no longer visible is hold by a background process.
Lowest	Empty	These processes contain no active application components and exist only for caching purposes.

(Source: http://os.ibds.kit.edu/downloads/sa_2010_braehler-stefan_android-architecture.pdf)

Processes in Android can contain multiple threads, as usual on Linux based systems. Most Android applications consist of multiple threads in order to separate the UI (user interface) from input handling and I/O operations, or other time consuming calculations, hence the processes are multi-threaded. The threads used on application level are standard Java threads running in the Dalvik VM (Brahler, 2010).

2.5 Reviewing the Existing Solution

With the limitation of mobile network operators, there are more and more call forwarding or call management applications have been developed because there is a demand in the market.

There are some well-known applications like Skype (Skype, n.d.), Google Voice for mobile (Google, 2011; Google Inc., n.d.), Advanced Call Manager for Symbian (WebGate, n.d.), SMS Forwarder (Wireless Labs Technologies, n.d.; Shen, 2011; Pixelchef, 2011), and others. Most of the applications that created for call forwarding are only available for certain countries and major mobile network operators only.

There are various ways of forwarding for the existing solution or applications, some of them forwarding missed calls and SMSs using emails, some setting up call forwarding using certain code or prefix (GeckoBeach, n.d.), e.g. *21*[*phone number*]*#* to activate call forwarding for GSM (Global System for Mobile Communications) carriers, #21# for deactivation.

There are also some applications that are created for forwarding SMSs or just forwarding incoming calls. Some advanced features provided in the *Call Forward* (Techvision Systems, n.d.), *Call Forward +* (Techvision Systems, n.d.) and *SMS Forwarding* (Shen, 2011), they are including remotely enable or disable forwarding feature by sending a specific SMS to the mobile phone. The users just need to send a certain command with the predefined PIN number to enable/disable certain features.

However, one of the major problems in these SMS forwarding applications is they are only used to forward the SMS in one-way, which means the application that installed on the intermediate mobile phone will only forward the SMS to the user-defined recipient's number, and the users cannot reply to the sender from another mobile phone by going through the intermediate mobile phone.

Besides, another similar application designed for Symbian OS, *Advanced Call Manager* (WebGate, n.d.), it does have a scheduler for the users to switch on and off the different features automatically as scheduled. Advanced applications like Skype and Google Voice do allow the users to perform additional features like participating

in conference calls, instant messaging or sending free text messages, video calls, etc., but the most serious drawback would be a lot of the functions including call forwarding feature are only supported certain countries like the United States.

Eventually, most of the existing applications have included the filtering functionality that can used to blacklist or whitelist a list of phone numbers, therefore we can conclude that this feature is a necessity in a phone call's managing type of application.

2.6 Comparison with the Mobile Network Operators

Nowadays in most of the mobile network operators, the “call forward” or “call divert” feature is known as Value Added Services, which means it is an additional service in subscription form whereby the users are required to pay for a one-time installation fee and a monthly fee in order to use the service.

Table 2.6.1: Comparing the “call forwarding” service from different mobile network operators.

Network Operator	DiGi	Maxis	Maxis	Celcom	TM
Type	Business	Consumer	Business	Business	Business
Contract	Postpaid	Postpaid	Postpaid	Postpaid	Postpaid
Minimum Monthly Fee for Phone Line	RM 50	RM 28	RM 30	RM 28	RM 45
Monthly Fee for Call Forwarding	RM 0	RM 3	RM 3	RM 3	RM 1.50
Remarks	Cannot divert to Voicemail			RM10 of installation fee	

Table 2.6.1 above is a comparison between the major local carriers with the call forwarding service that provided by each of them, all the data are based on the information published on the official website of each carrier.

Based on the research, in most circumstances, there are only postpaid users have an option to activate the call forwarding feature (only if the service is offered by the carriers), whereas the prepaid users couldn't.

The following illustration shows you a scenario of forwarding an incoming call to the user's office phone number.



Figure 2.6.1: Illustrating how call charges apply when a call is forwarded to another number.

(Source: <http://www.maxis.com.my/faq/default.asp?strWebsite=maxiscom&catID=25>)

In the Figure 2.6.1, we can see that on top of the monthly fee, the user will be additionally charged by the carrier while using the call forwarding service to forward an incoming call, just like manually dialling another call to the office phone number. This is not much different with the conference call method that proposed in this project.

In contrast, the user can choose to subscribe to the family plan offered by the carriers in which they are bundled with free calls and free SMSs between its supplementary lines, as summarised in the table below.

Table 2.6.2: Comparing the family plan from different mobile network operators.

Network Operator	DiGi	Maxis	Celcom
Contract	Postpaid	Postpaid	Postpaid
Monthly Commitment Fee (each line)	RM 28	RM 30	RM 15
Maximum Number of Supplementary Line	6	10	5
Free Calls Between Supplementary Line	100 hours (each line)	15 hours (shared)	40 hours (shared)
Free SMSs Between Supplementary Line	3,800 units (each line)	1500 units (shared)	2,000 units (shared)
Call Charges (per minute)	RM 0.15	RM 0.15	RM 0.20
Free Calls' Value	100 x 60 x 0.15 = RM 900	15 x 60 x 0.15 = RM 135	40 x 60 x 0.20 = RM 480

So we can conclude that users can choose to use two mobile phone lines with the family plan, they can make use of the free calls given to perform call forwarding with free of charge and with no worried about the capability of their carrier. In addition, the users can perform filtering on their own mobile phone instead of calling to the respective customer service to request for it from time to time.

CHAPTER 3 METHODOLOGY

3.1 Methodology and Tools

A software development methodology or system development methodology in software engineering is a kind of framework that used to structure, plan, and control the process of developing an information system (Association of Modern Technologies Professionals, n.d.).

There are a broad variety of frameworks evolved over the years; each of them has its own recognized strengths and weaknesses. Some of the commonly used methodologies are Waterfall development, Incremental development, and Prototyping. This project is going to practice the principles and utilise the strengths of the Incremental development methodology until the upshot or outcome has been produced.

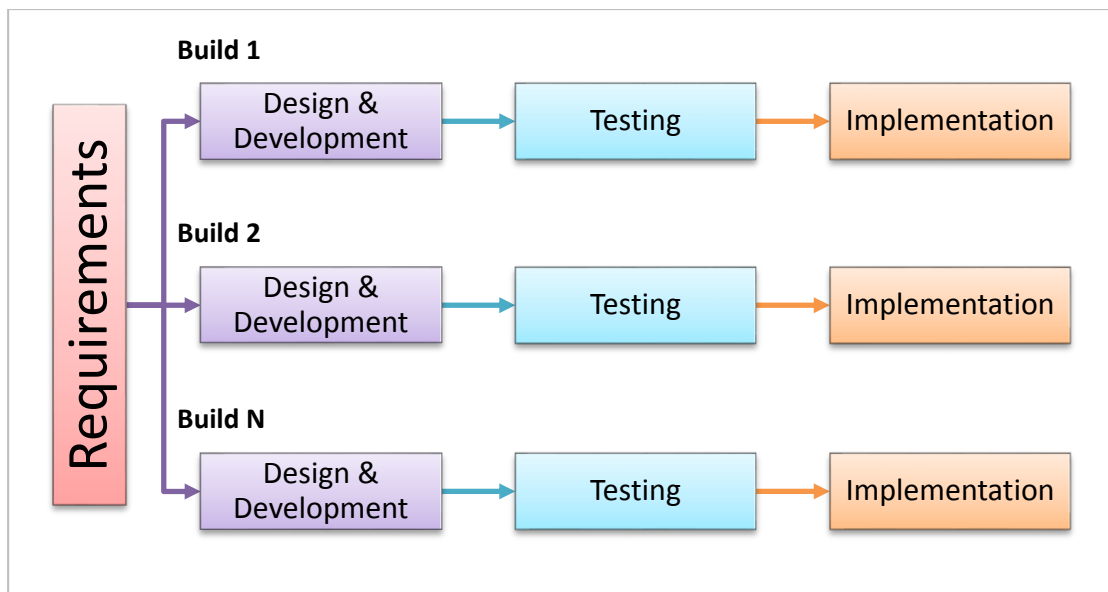


Figure 3.1.1: Incremental Life Cycle Model.

(Adapted from: <http://www.qualitytesting.info/profiles/blogs/sdlc-incremental-model>)

The primary objective of the Incremental development methodology is to reduce inherent project risk by breaking a project into smaller segments and providing

more ease-of-change during the development process (Centers for Medicare & Medicaid Services, 2008).

In the early stage of development, each of the individual features in the application will be broken into smaller segments, so that it can accept some future changes and mitigate the risks earlier in the project. Using this methodology allows delivery of a series of implementations that are gradually more complete and can go into production more quickly as incremental releases.

In this methodology, it allows the developers incrementally release multiple builds/versions to production, gradually to a complete version. Thus there will be several versions released for this application in the whole project life cycle.

Android recommends and supports the Java programming language primarily for its applications, so Java programming language will be used to develop this project. In order to develop the application, the Eclipse IDE and at least one Android mobile phone will be used for the testing purpose.

3.2 Requirement Specifications

By referring to the latest statistic reviewed in Appendix B, there are about 81.4% of the Android mobile phones are using the version in between version 2.2 (Froyo) and version 4.0.4 (Ice Cream Sandwich). To make this application least complicated, this range of Android versions will be able to install this application and enjoy the benefits of it.

Next, the user's mobile network operator should enable the user to start or create a conference call with at least three mobile phones, else the user will only able to use the SMS forwarding feature in this application.

Lastly, the user's public mobile phone line should have a sufficient amount of credits in order to allow the application forwarding calls and messages properly.

3.3 Implementation and Testing

During the testing phase, I need to perform multiple tests with different major versions of Android OS, to clarify the supported Android versions. This application will be first installed in the emulators or Android Virtual Devices (AVD), testing in several versions of Android; most of the functionality can properly tested using the emulator.

In the real environment, the application will then be installed into the actual Android-powered mobile phones (with Ice Cream Sandwich and Gingerbread) and execute it. The Android mobile phone will be used as the intermediate point or the public mobile phone number, another two mobile phones will also involve for testing the forwarding features.

For more in-depth testing, this application will be tested using Black Box Testing technique to test and ensure it does what it is supposed to do based on its functional requirements.

CHAPTER 4 PHONE CALLS AND TEXT MESSAGES CONTROL APPLICATION

4.1 Overall Application Architecture

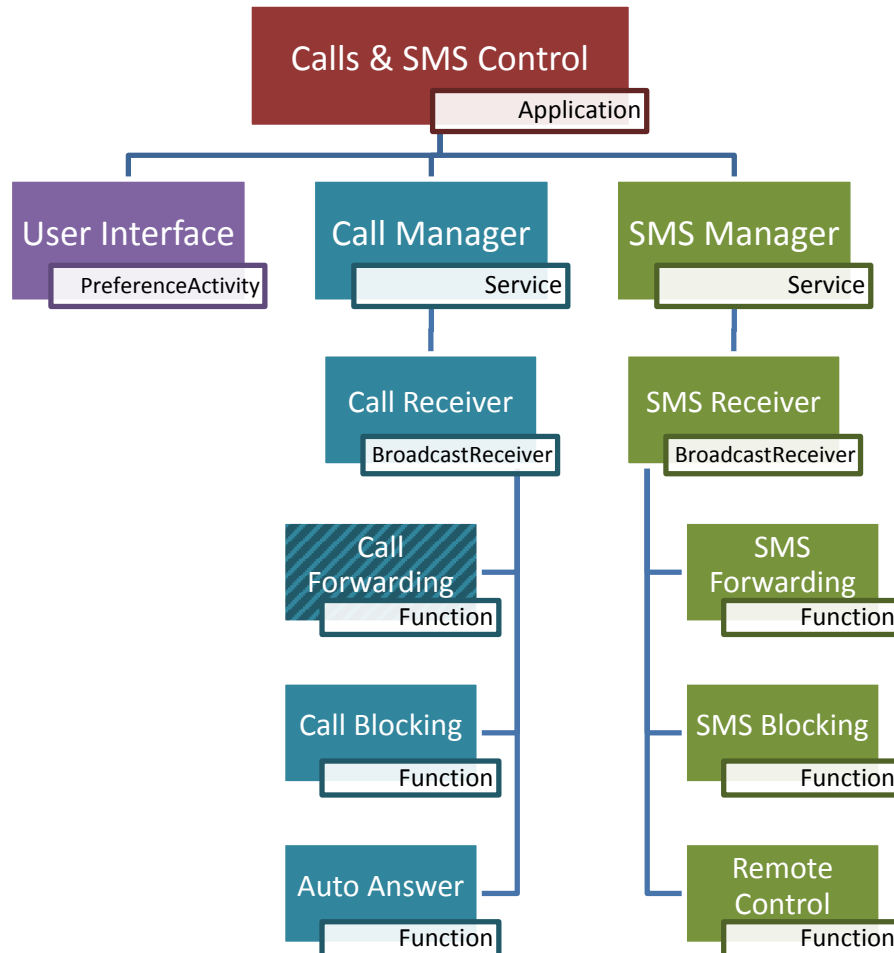


Figure 4.1.1: Overall application architecture for Phone Calls and Text Messages Control Application.

The overall application architecture for Phone Calls and Text Messages Control application was shown in Figure 4.1.1. The application name has to be shorter for better branding, easy to remember and reflects what it can be performed, thus I simplified it into “Calls & SMS Control”. All of the proposed functions in this project are used to control or intercept the incoming calls and text messages, the list below describe the usages of each function name:

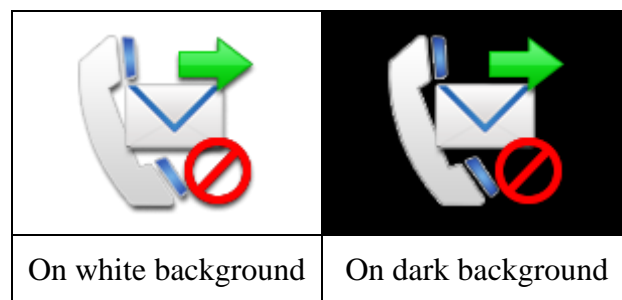
- **Call Forwarding:** This function is to allow the users to redirect/transfer incoming phone calls from one phone to another, to protect the users' privacy. (This function has been discarded)
- **Call Blocking:** This function is to allow the users to screen/filter the incoming phone calls, block unwanted calls based on the user-defined whitelist/blacklist.
- **Auto Answer:** This function is to ease the users while driving car, it automatically pick-up the incoming calls without users reaction on it.
- **SMS Forwarding:** This function is to allow the users to redirect/transfer incoming text messages (SMS) from one phone to another, to protect the users' privacy.
- **SMS Blocking:** This function is to allow the users to screen/filter the incoming text messages (SMS), block unwanted text messages based on the user-defined whitelist/blacklist.
- **Remote Control:** This function is to allow the users to change the settings of this application remotely by using text messages (SMS).

However, due to some restrictions on the Android SDK, the Call Forwarding feature could not succeed. Refers to Chapter 4.6 for more detailed explanation.

4.2 Application Icon

Every application should have an icon to represent itself, it is a graphic symbol that denotes an application. A good icon became part of a brand, and often helped to make the program or application stand out in an app store (Infragistics, 2013). It helps to express what this application can accomplish and helps the users to remember it. The Android Developers site (http://developer.android.com/guide/practices/ui_guidelines/icon_design_launcher.html) does provide a proper guideline for the application developers to follow. Table 4.2.1 shows the application icon that has been designed for this project.

Table 4.2.1: The application icon demonstrated in two different background colours.



In most circumstances, the application icon will appear on different types of background, it can be in the Home screen, or at least in the All Apps screen. Demonstrating the application icon in two contradict background colours can help to ensure the application icon has minimal possible differences on different user interfaces.

Every object used in the application icon design do carry a specific purpose or reason, Table 4.2.2 below shows the explanation for each object.

Table 4.2.2: Brief explanation on each object used in the application icon.

Object	Explanation
Phone	<ul style="list-style-type: none"> • Represent incoming voice calls.
Envelope	<ul style="list-style-type: none"> • Represent incoming text messages (SMS).
Green Right-Arrow	<ul style="list-style-type: none"> • Allowed to proceed if not blocked, can continue with forwarding or auto answering. • Green colour represents safety or approved.
Red Blocked Sign	<ul style="list-style-type: none"> • Not allowed to proceed if blocked, either drop the call or remove the text message. • Red colour represents danger or malicious.

4.3 Version Release History

Version 0.1:

- Initial release
- Initial function: SMS Forwarding

Version 0.2:

- New function: Call Blocking
- New function: Auto Answer
- Bug fixes and enhancements

Version: 0.3:

- New function: SMS Blocking
- Update: Call Blocking
 - User can now choose not to automatically end the call
 - User can now choose not to automatically remove missed-call notification
- Known issue: Missed-call notification cannot be removed automatically in Android 4.0 onwards.
- Bug fixes and enhancements

Version 0.4:

- New function: Remote Control
 - 4 commands available
 - CALL MANAGER ON
 - CALL MANAGER OFF
 - SMS MANAGER ON
 - SMS MANAGER OFF

Version 1.0:

- Finalised release
- Update: Remote Control
 - All commands are properly categorised

- 6 new commands added
 - CALL BLOCK ON
 - CALL BLOCK OFF
 - SMS FORWARD ON
 - SMS FORWARD OFF
 - SMS BLOCK ON
 - SMS BLOCK OFF
- Update: Call Blocking
 - Added the ability to remove all blocked numbers from the blacklist
- Update: SMS Blocking
 - Added the ability to remove all blocked numbers from the blacklist

4.4 User Interface Design

The user interface (UI) is necessary for the normal users to view and customise the settings according to their needs and different situations. For more personalisation on the UI, we can make use of the different layout types in common layout objects that provided by Android SDK to declare a layout.

For this project, users are only required to interact with multiple settings for the app, thus using the Android's *Preference* APIs can build an interface that's consistent with the user experience in other Android apps; the *Preference* APIs are commonly using in the system settings (Google, n.d.).

Figure 4.4.1 to Figure 4.4.3 demonstrating the UI in three different versions of the Android platform – 2.3, 3.2, and 4.0.

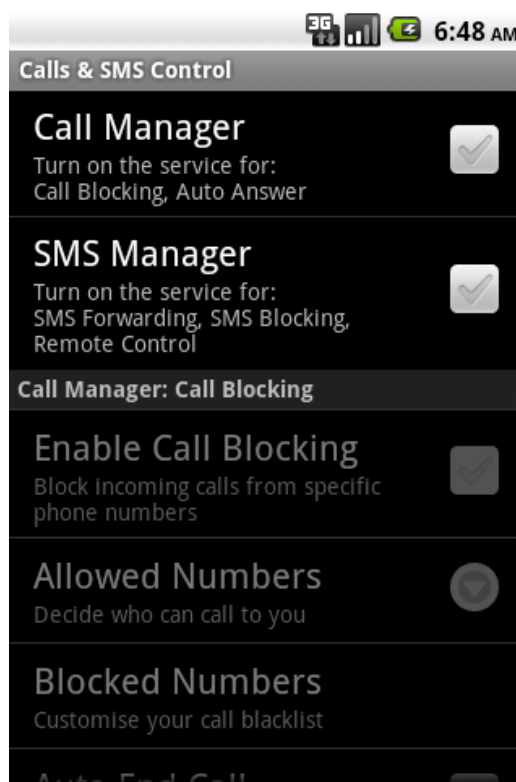


Figure 4.4.1: User interface displayed on Android version 2.3.

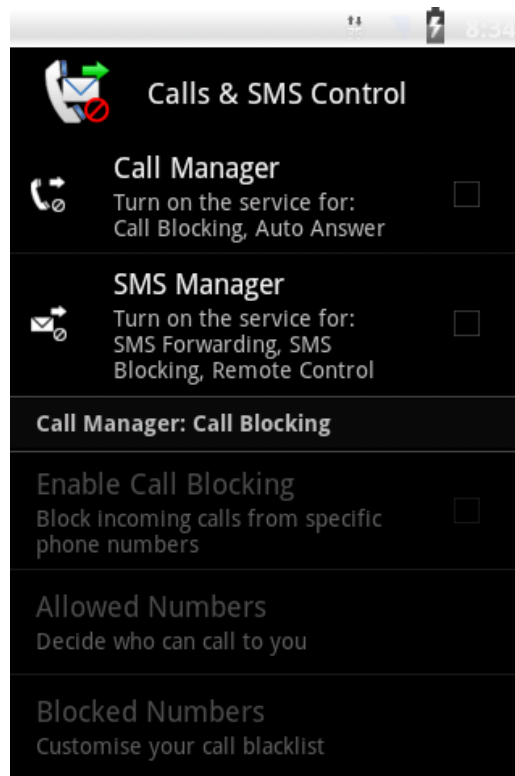


Figure 4.4.2: User interface displayed on Android version 3.2.

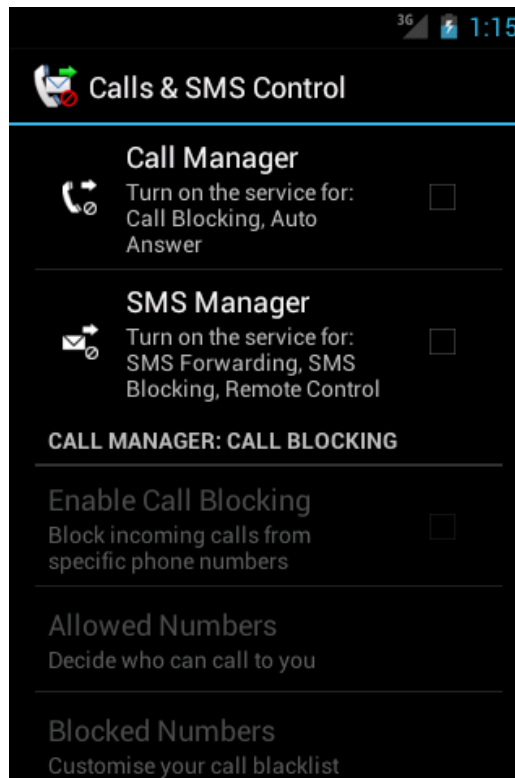


Figure 4.4.3: User interface displayed on Android version 4.0.

Each *Preference* has a corresponding key-value pair that the system uses to save the setting in a default *SharedPreferences* file, whenever the user changes a setting, the system will update the corresponding value in the *SharedPreferences* file automatically.

4.5 Service: Call Manager and SMS Manager

The “Call Manager” and “SMS Manager” services are used to monitor the incoming calls and text messages on the mobile phone. A started service is used run in the background continuously even if the user switches to another application, so that all the incoming calls or text messages can be intercepted by this app and perform what the user requests.

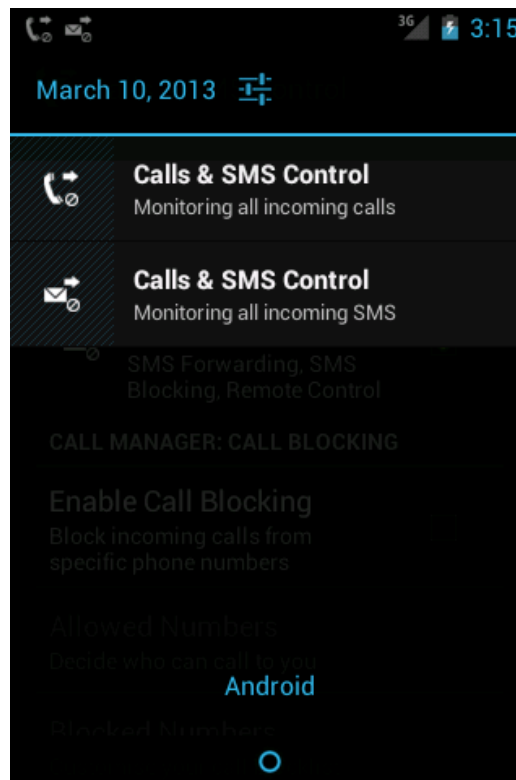


Figure 4.5.1: Notification for each individual service is shown to indicate the particular service is turned on.

Figure 4.5.1 shows the notification drawer of an Android 4.0 device while both services are turned on. Each of the services has its own status bar icon that can clearly recognised by the users.

An intent is an abstract description of a process to be executed. Each of the services will register a *BroadcastReceiver* class to enable this application to receive

intents that are broadcast by the system, the Call Manager will register the Call Receiver to receive the intent from the system when the phone state changed, whereas the SMS Manager will register the SMS Receiver to receive the intent from the system when SMS received.

4.6 Function: Call Forwarding

The aim of this function is to “forward” the incoming calls from one phone to another phone. The whole process is illustrated in the following figure for better understanding.

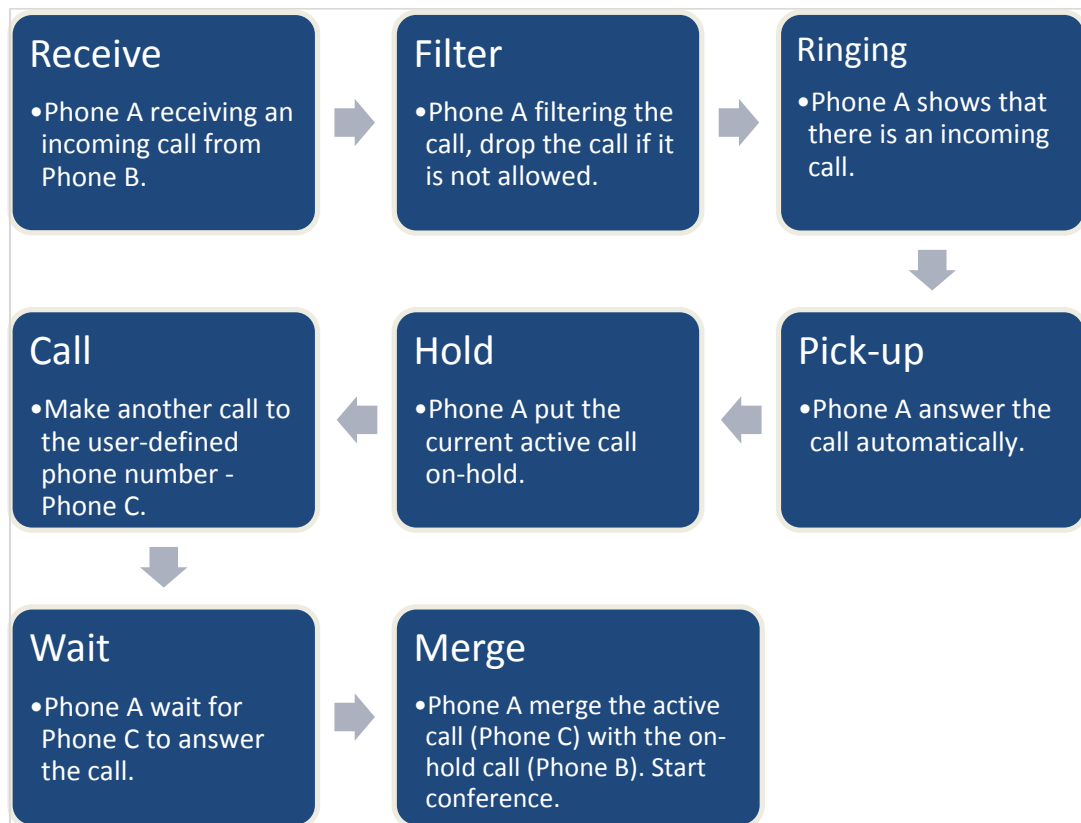
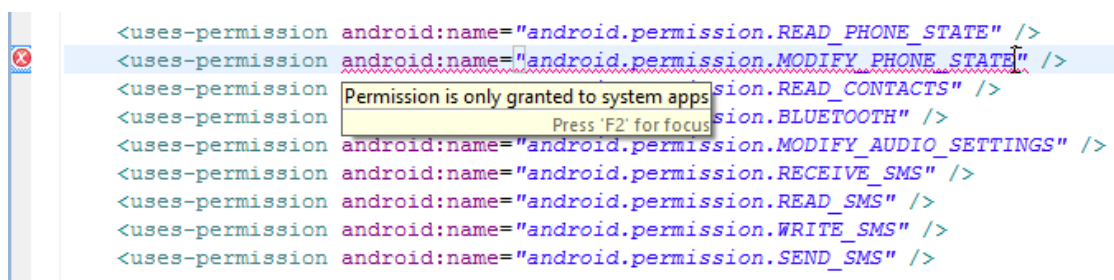


Figure 4.6.1: The overview of the process of Call Forwarding function.

Let’s take Phone A as the mobile phone with this application installed, which is the user’s public mobile phone; Phone B as the mobile phone that make a call to Phone A; and Phone C is the user-defined phone number, which is the user’s private mobile phone. The process begins when Phone B makes a phone call to Phone A, Phone A perform screening and filtering on the call if user requested, if the call is blocked, it will be dropped by the application, or continue to let the phone ring. Then this application will pick-up the incoming call automatically, put the call on-hold and make a new phone call to Phone C. If the user pick-up the call on Phone C, Phone A

will merge the two calls together to create a conference call, Phone B can now communicate with Phone C without knowing the phone number of Phone C.

By going through the process above, we can make use of the conference call method to omit the user-dependent on their carriers. Unfortunately, this function is failing to implement due to the limitation in the Android SDK. Start from API Level 9 – Android 2.3 (Janecek, 2011), the `MODIFY_PHONE_STATE` permission has been granted only to system apps (refers to Figure 4.6.2). This permission is important for user apps to alternate the phone state, e.g. answering the call, drop the call. A number of the APIs/classes (e.g. *ITelephony*, *CallManager*) have become hidden and restricted from using in user apps, caused this function failed to merge the two calls together in its development stage.



```

<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.MODIFY_PHONE_STATE" />
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.RECEIVE_SMS" />
<uses-permission android:name="android.permission.READ_SMS" />
<uses-permission android:name="android.permission.WRITE_SMS" />
<uses-permission android:name="android.permission.SEND_SMS" />

```

Figure 4.6.2: `MODIFY_PHONE_STATE` permission has been reserved for system apps in newer SDK.

Before deciding to discard this function from the application, several solutions have been applied to solve the problems faced. This function requiring the internal library – *ITelephony* to pick-up the call programmatically, one of the alternative method is to use Java Reflection to gain access to *TelephonyManager*'s *ITelephony* getter, but the `MODIFY_PHONE_STATE` permission is required. In that case, this method is not working.

In order to pick-up the call programmatically, I am using a Bluetooth headset hook to trigger a “pick-up the call” button event to answer the incoming call, below is the code that simulate the event:

```

private void answerPhoneHeadsethook(Context context) {
    // Simulate a headset plugged event
    Intent headSetPluggedintent = new Intent(
        Intent.ACTION_HEADSET_PLUG);
    headSetPluggedintent
        .addFlags(Intent.FLAG_RECEIVER_REGISTERED_ONLY);
    // state = 1 for plugged
    headSetPluggedintent.putExtra("state", 1);
    headSetPluggedintent.putExtra("name", "Headset");
    sendOrderedBroadcast(headSetPluggedintent,
        "android.permission.CALL_PRIVILEGED");

    // Simulate a press of the headset button to pick up the call
    Intent buttonDown = new Intent(
        Intent.ACTION_MEDIA_BUTTON);
    buttonDown.putExtra(Intent.EXTRA_KEY_EVENT,
        new KeyEvent(KeyEvent.ACTION_DOWN,
            KeyEvent.KEYCODE_HEADSETHOOK));
    context.sendOrderedBroadcast(buttonDown,
        "android.permission.CALL_PRIVILEGED");

    // Froyo (2.2) and beyond trigger on buttonUp instead of
    buttonDown
    Intent buttonUp = new Intent(Intent.ACTION_MEDIA_BUTTON);
    buttonUp.putExtra(Intent.EXTRA_KEY_EVENT,
        new KeyEvent(KeyEvent.ACTION_UP,
            KeyEvent.KEYCODE_HEADSETHOOK));
    context.sendOrderedBroadcast(buttonUp,
        "android.permission.CALL_PRIVILEGED");

    // Unplug headset
    // state = 0 for unplugged
    headSetPluggedintent.putExtra("state", 0);
    sendOrderedBroadcast(headSetPluggedintent,
        "android.permission.CALL_PRIVILEGED");
}

```

This method above can successfully “trick” the system that the user has plugged in a Bluetooth headset and pressed on the “pick-up” button, so the system will answer the ringing call.

While reaching the stage whereby needing to merge the active call with the on-hold call together, several methods have been tried to complete it but failed. There are several internal libraries that have the method that is called internally to merge two calls together to start the conference, but none of them can be accessed without the sharing the same user ID with the system (“android.uid.system”), these libraries including:

- com.android.internal.telephony.CallManager,
- com.android.internal.telephony.Phone, and

- `com.android.internal.telephony.PhoneProxy`.

To share the same user ID with the system, the application need to be signed with the same private key of the system and this application will become a system app, whereby it requires the users to install into the system directory in the internal storage.

Besides of this method, using Java Reflection to grant access to those internal library are also facing the same error. The last method that I have tried is to replace the public Android 4.2.2's SDK with a complete set with all the internal class files, so that I can import those internal libraries without using Java Reflection. However, this method is still return no result because of the same error.

4.7 Function: Call Blocking

“Filtering” is one of the main functions that need to be done by this application, and it is getting more important to exist in mobile phones nowadays. As illustrated in Figure 4.6.1, this is the stage that will occur before the phone is ringing or vibrating to alert the user that there is a caller calling in.

The purpose of this function is to check whether the caller’s phone number is in the user’s blacklist or not, the call will be blocked and hung up automatically if it is blacklisted, else the mobile phone will proceed to tell the user about the incoming call. The consequences of these two types of callers are shown in the Table 4.7.1 below.

Table 4.7.1: Consequences for the blocked callers and allowed callers when Call Blocking is enabled.

<div data-bbox="422 1070 742 1384" data-label="Image"> </div> <p>Blocked Callers</p> <ul style="list-style-type: none"> • Turn off the ringer’s volume and disable the vibration until the call is ended. • Drop the call (optional). 	<div data-bbox="901 1093 1332 1361" data-label="Image"> </div> <p>Allowed Callers</p> <ul style="list-style-type: none"> • Continue with other remaining processes, e.g. Auto Answer (optional). • Let the phone ringing and/or vibrating.
--	---

This application will help the users to turn off the ringing’s volume and disable the vibration when the call is in the blacklist, and then the call will be hung up by this application (together with removing the missed-call notification) if the users

choose to end the call immediately. On the other hand, the mobile phone will ring or vibrate as usual if the call is from the allowed callers.

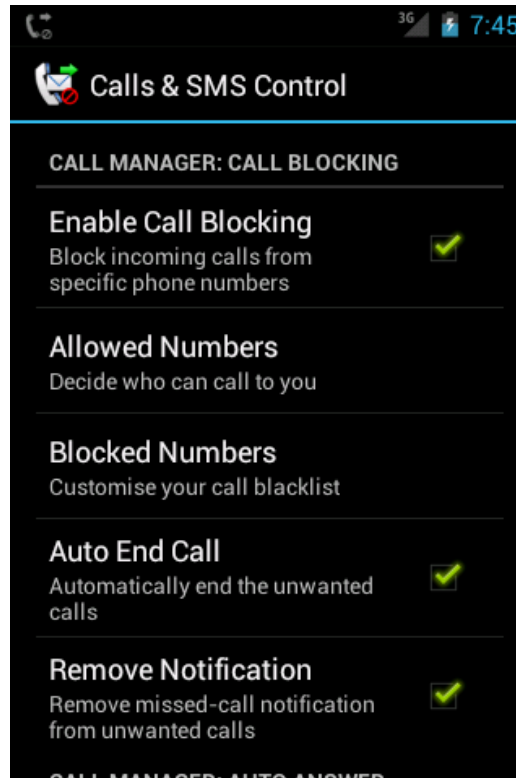


Figure 4.7.1: The user interface of Call Blocking.

On the UI, the users will be giving some basic settings to customise, as shown in Figure 4.7.1. By clicking on the “Allowed Numbers”, users are given to choose the allowed callers (also known as whitelist) from their current contacts list that stored in their mobile phone, as shown in Figure 4.7.2. This can save the users’ time to key in all the numbers again.

Besides of using contact list as the whitelist, the users also can choose their favourite/starred list (provided by Android) to limit the allowed numbers to a smaller group of people. This application also can help the users to block every incoming calls if they choose “No Calls Allowed” in the setting.

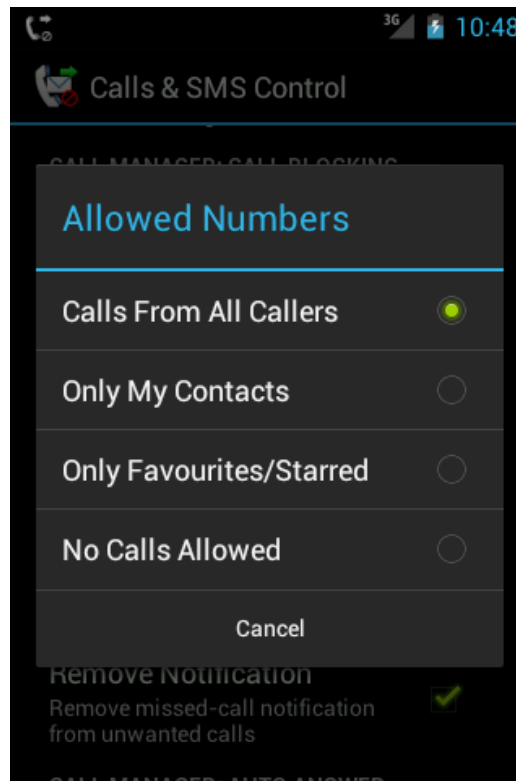


Figure 4.7.2: Users can choose to allow only receiving calls from contacts, favourites or block all the calls.

However, to customise the blacklist, the users can no longer choosing from a list of options but they are provided with the ability to add unwelcomed phone numbers into a custom made list. The blacklist for Call Blocking and SMS Blocking are isolated, each of them has its own list, so that the users can choose to block a particular phone number from calling in or sending in text messages.

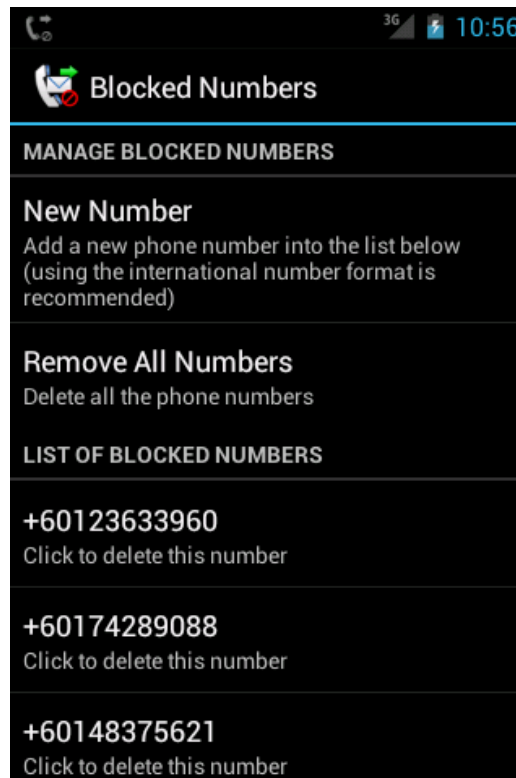


Figure 4.7.3: An individual screen to let the users to manage the blacklist and display the list of blocked phone numbers.

I am using (import) the *LinkedHashMap* in Java to store the list of numbers, the *LinkedHashMap* will then save into the into the mobile phone's memory with the *SharedPreferences* object. By using *LinkedHashMap* instead of *HashMap*, the list can be sorted accordingly the same as the insertion. The *LinkedHashMap* provides constant-time performance like *HashMap* (with $O(1)$ in Big 'O' Notation), for the basic operations like insertion, contains and removal, it overcomes the limitation whereby the *HashMap* do not sort the list entries in insertion.

Every list item will have a unique key to represent each phone number, this key can be used in the *Preference*'s key attribute. There is no any method that can directly saving a *Map* object into the *SharedPreferences* object, here is the code segment for saving the blacklist (*LinkedHashMap*) into the mobile phone (*SharedPreferences*):

```
private final String call_bl_arrayName = "call_bl_item_";
private static Map<String, String> sCallBlacklist = new
```

```

LinkedHashMap<String, String>();

public boolean saveCallBlacklist() {
    SharedPreferences prefs = mContext.getSharedPreferences(
        "call_blacklist", Context.MODE_PRIVATE);
    Editor editor = prefs.edit();
    // Remove old blacklist
    editor.clear().commit();

    long i = 1;
    for (String key : sCallBlacklist.keySet()) {
        editor.putString(call_bl_arrayName + i++,
            sCallBlacklist.get(key));
    }

    return editor.commit();
}

```

Code segment for loading the blacklist (*LinkedHashMap*) from the mobile phone (*SharedPreferences*):

```

private Object loadBlacklist(String listName) {
    SharedPreferences prefs = mContext.getSharedPreferences(
        listName, Context.MODE_PRIVATE);

    Object temp = prefs.getAll();

    if (temp instanceof Map<?, ?>) {
        return temp;
    } else {
        return null;
    }
}

public void loadCallBlacklist() {
    // Retrieve the call blacklist
    Object temp = loadBlacklist("call_blacklist");

    if (temp != null) {
        // List is not empty
        sCallBlacklist = (Map<String, String>) temp;
        sCallBlacklistSize = sCallBlacklist.size();
    } else {
        // List is empty
        sCallBlacklist = null;
        sCallBlacklistSize = 0;
    }
}

```

As demonstrated in the code segments above, we will be able to store and retrieve the blacklist records from the mobile phone.

To delete a phone number from the blacklist, the users can just click on that particular phone number to delete a single record, or clicking on the “Remove All Numbers” to clear the entire list. For best practices, a confirmation dialog (see Figure 4.7.4) will be prompted to the users before it continue deleting those phone numbers, to preclude the users accidentally removed them.

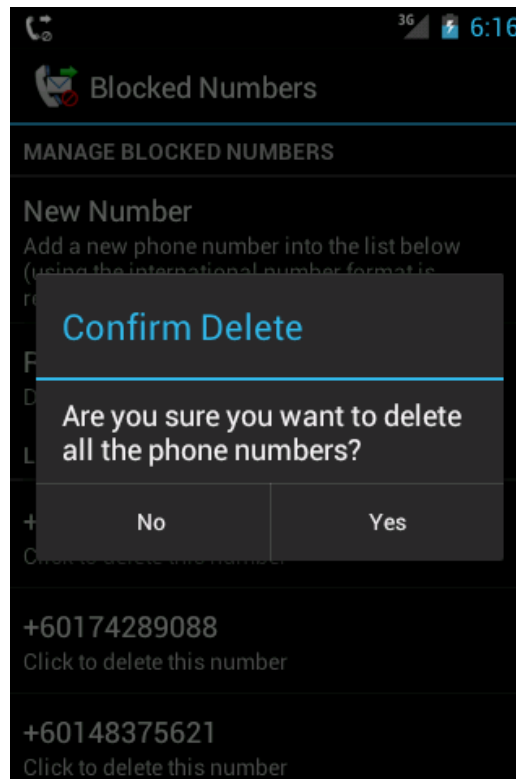


Figure 4.7.4: A confirmation dialog when a user trying to clear the blacklist.

In future development, this function can be further improved by adding a search feature to let the users to find for a particular phone number, and a list sorting feature to allow the users to view the list in different sorting.

4.8 Function: Auto Answer

This function is used to pick-up the incoming calls for the users automatically without any user reaction. Therefore when the users are driving the car, they can also pick-up and answer to the incoming calls in a safe manner, because they do not need to leave their hands away from the steering wheel.

This function is derived from the discarded function – Call Forwarding, by making use of what I have done for that function and then further enhance on it. Refers to Chapter 4.6 for the implementation of this function.

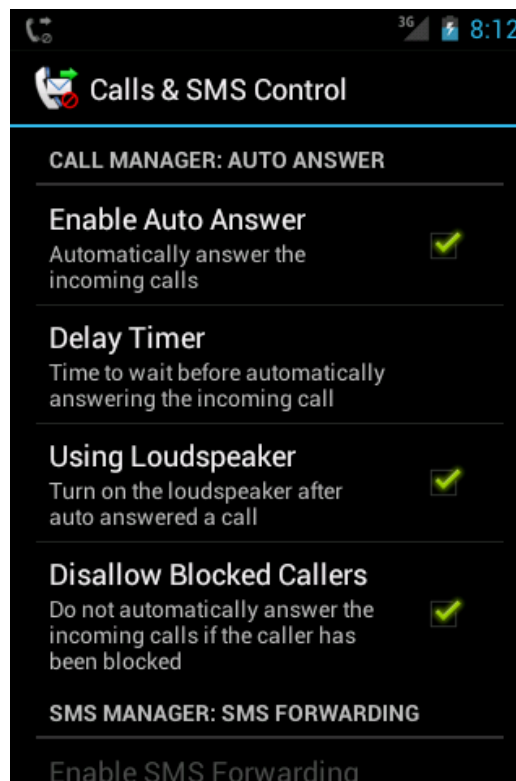


Figure 4.8.1: The user interface of Auto Answer.

“Auto Answer” allows the users to set a delay in seconds (see Figure 4.8.2), from 0 (zero) to 8 seconds, before this application starts to pick-up the call, by default it is set to be 3 seconds. It will turn on the loudspeaker immediately after the call has been picked-up, the user can directly listen and respond the caller.

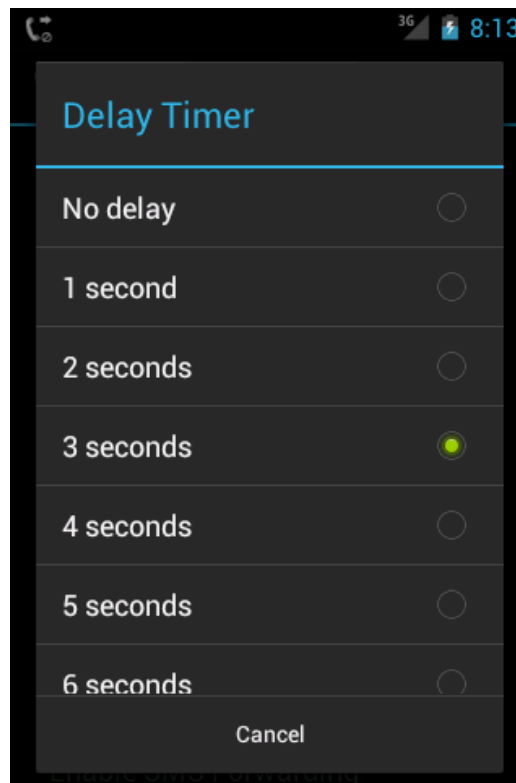


Figure 4.8.2: Delay Timer in list selection.

This function can be used together with Call Blocking to instruct this application only pick-up the allowed calls automatically, instead of all incoming calls.

4.9 Function: SMS Forwarding

The aim of this function is to “forward” the incoming text messages (SMS) from one phone to another phone. The whole process is illustrated in the following figure for better understanding.

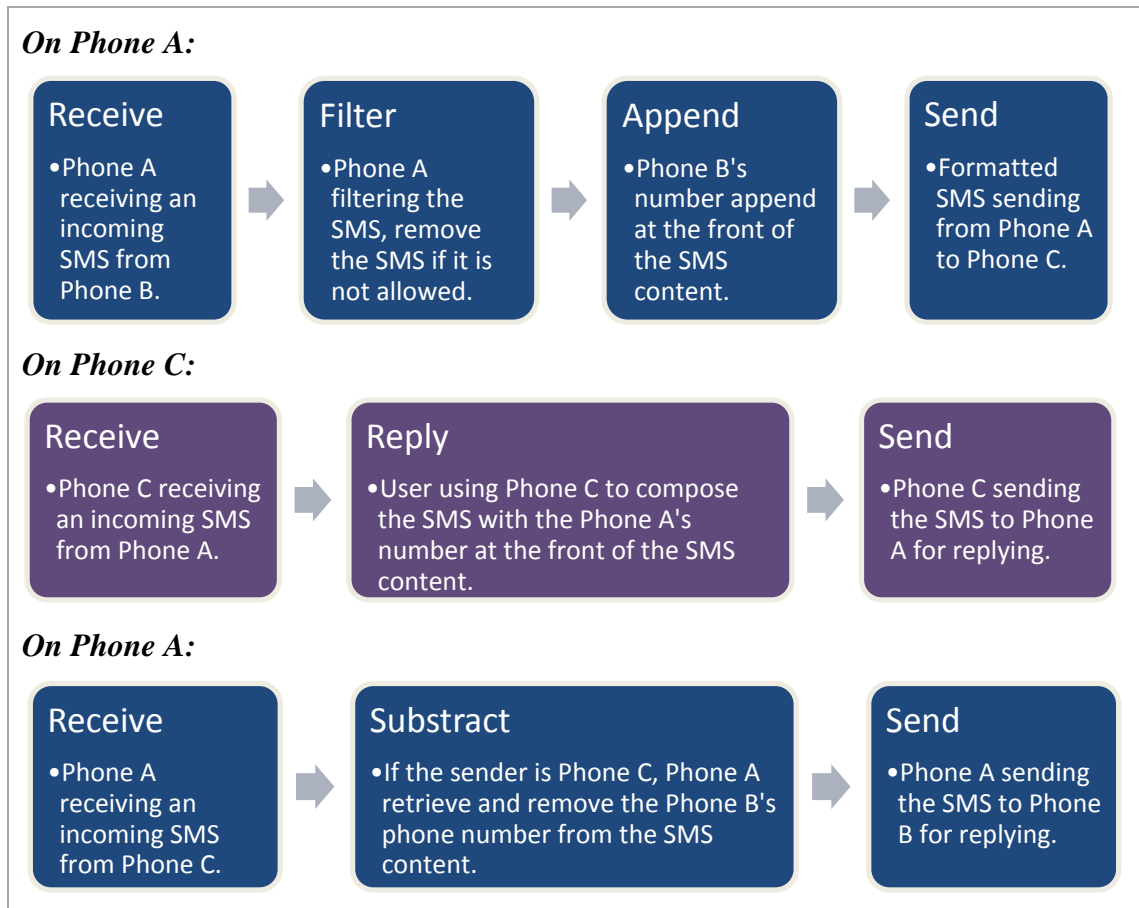


Figure 4.9.1: The overview of the process of SMS Forwarding function.

Let's take Phone A as the mobile phone with this application installed, which is the user's public mobile phone; Phone B as the mobile phone that make a call to Phone A; and Phone C is the user-defined phone number, which is the user's private mobile phone. The process begins when Phone B sent an SMS to Phone A, Phone A perform screening and filtering on the SMS if user requested, if the SMS is blocked, it will be removed by the application, or append Phone B's number in front of the SMS and send it to Phone C. To reply from Phone C to Phone B, the user just need to put

the Phone B's number in front of the SMS and send it to Phone A, when Phone A receive the SMS, it will remove the Phone B's phone number from the SMS and then send it to Phone B.

By going through these processes, the user can use Phone C to receive and reply to text messages with Phone B without disclosing the Phone C's phone number. Figure 4.9.2 is used to show the process of formatting the SMS content and retrieving the phone number from it, assuming the phone number of Phone A is 11111, Phone B is 22222, and Phone C is 33333.



Figure 4.9.2: SMS content in different mobile phones for SMS Forwarding.

In order to let the users know who the sender of that forwarded SMS is, the sender ID (sender's phone number) will append to the front of the message. Whereas to allow the owner (the mobile phone that has the same number with the user-defined recipient's number) to reply to the sender, I use the same technique but just to retrieve the sender ID from the message that receive from the owner, this makes the users able to receive and reply to SMS without revealing the private phone number.

Google does not restrict the Android developers from accessing the APIs for SMS, so this function can be succeeded but not the Call Forwarding. Figure 4.9.3 shows some of the settings of this function.

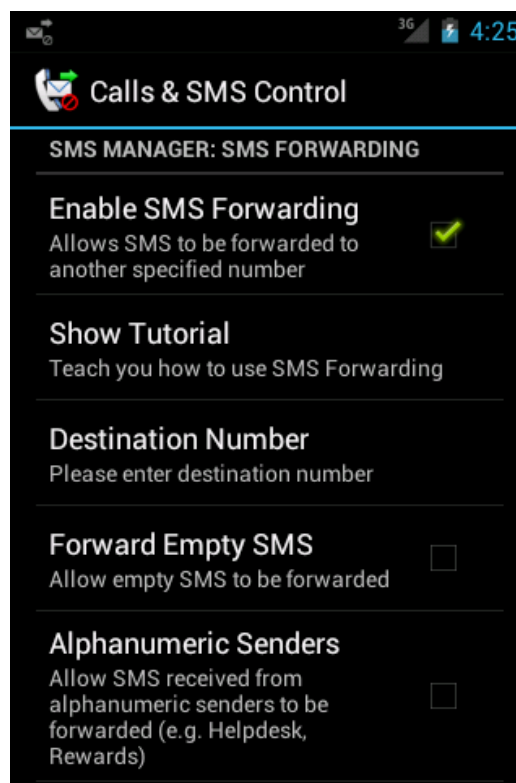


Figure 4.9.3: The user interface of SMS Forwarding

When the mobile phone receiving a new text message, it will proceed to the filter stage if the user has enabled the SMS Blocking function. After the filtering (refers to Chapter 4.10) is done, the application will append the sender's phone

number (or sender ID) following by a delimiter at the front of the SMS content before sending/forwarding it to the user-defined phone number. The default delimiter used in this application is a ':' (colon), using a single character (or punctuation) is more than enough to separate the sender ID with the message.

A concatenated SMS can contain more than one SMS, where each SMS normally allows only 160 number of characters. To send the concatenated SMS, the SMS content need to be divided into multipart before send, as shown in the code segment below:

```
/**
 * @param recipient
 *         Recipient number
 * @param message
 *         Message body
 */
public boolean sendSms(String recipient, String message) {
    try {
        SmsManager smsMgr = SmsManager.getDefault();
        // Divide the message into parts
        ArrayList<String> parts = smsMgr
            .divideMessage(message);
        // Send the multipart messages
        smsMgr.sendMultipartTextMessage(recipient, null,
            parts, null, null);
    } catch (Exception e) {
        return false;
    }

    return true;
}
```

The application will perform a comparison check on the sender's phone number, if it is identical with the user-defined phone number, the application will change the recipient's phone number to the original sender, which is appended in the SMS content. In order to retrieve the recipient's phone number from the SMS content, I make use of the *StringTokenizer* in Java to retrieve and subtract the appended phone number from the SMS content. Here is part of the code:

```
String mDelimiter = ":";

StringTokenizer tokens = new StringTokenizer(message,
    mDelimiter);
int tokensCount = tokens.countTokens();

if (tokensCount == 2) {
```

```
// Get destination number from SMS
destination_number = tokens.nextToken();

// Get actual message body
fullMessage = tokens.nextToken();
}
```

Some of the SMS's senders can be displayed to the users in alphabets, they are known as alphanumeric senders (e.g. Helpdesk, Rewards, or the name of mobile network operators). The users are not be able to reply to these type of senders. This application need to perform a checking on the recipient's address before sending the SMS. There is also a setting for the users (refers to Figure 4.9.3) to decide whether they would like this application to forward those text messages from the alphanumeric senders.

To let the users get to know about how it works, a tutorial feature has been created to provide guidelines on how to write the SMS. The users can click on the "Show Tutorial" to get started. The Messaging app will be displayed with the tutorial generated, as shown in Figure 4.9.4. The tutorial is in scenario-based for easy understanding.

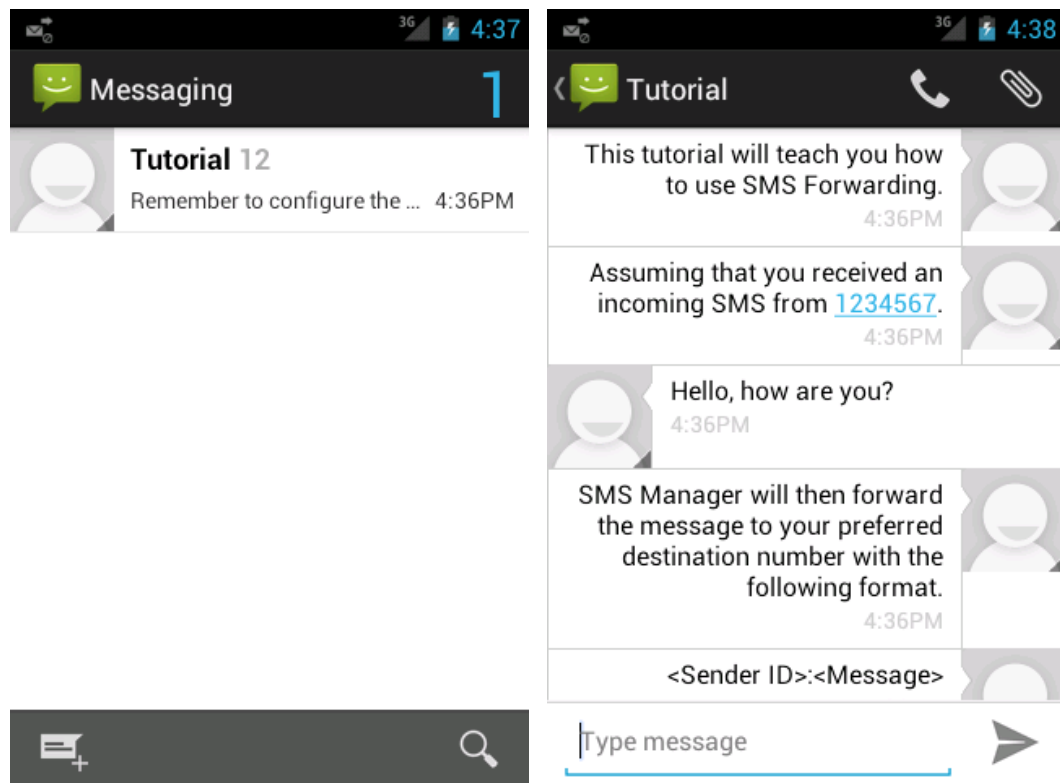


Figure 4.9.4: Tutorial generated for the users to know how to use the SMS Forwarding.

In future development, this feature can be improved by letting the users to choose the user-preferred delimiter to use on separating the sender ID with the message.

4.10 Function: SMS Blocking

The purpose of this function is similar with the Call Blocking function, to check if the sender's phone number is in the user's blacklist or not, the SMS will be blocked and deleted automatically if it is blacklisted, else the mobile phone will proceed to tell the user about the incoming SMS. The consequences of these two types of senders are shown in the Table 4.10.1 below.

Table 4.10.1: Consequences for the blocked callers and allowed callers when SMS Blocking is enabled.

<div data-bbox="422 824 742 1137" data-label="Image"> </div> <p data-bbox="327 1169 550 1205">Blocked Callers</p> <ul data-bbox="375 1227 774 1263" style="list-style-type: none"> • Delete the SMS (optional). 	<div data-bbox="901 846 1332 1120" data-label="Image"> </div> <p data-bbox="853 1169 1077 1205">Allowed Callers</p> <ul data-bbox="901 1227 1364 1370" style="list-style-type: none"> • Continue with other remaining processes, e.g. SMS Forwarding (optional).
---	--

The UI of this function is similar with the UI of Call Blocking, with slightly lesser settings. Users are also able to customise the blacklist and whitelist for this function, detailed explanation can referring to Chapter 4.7. Figure 4.10.1 shows the UI for this function.

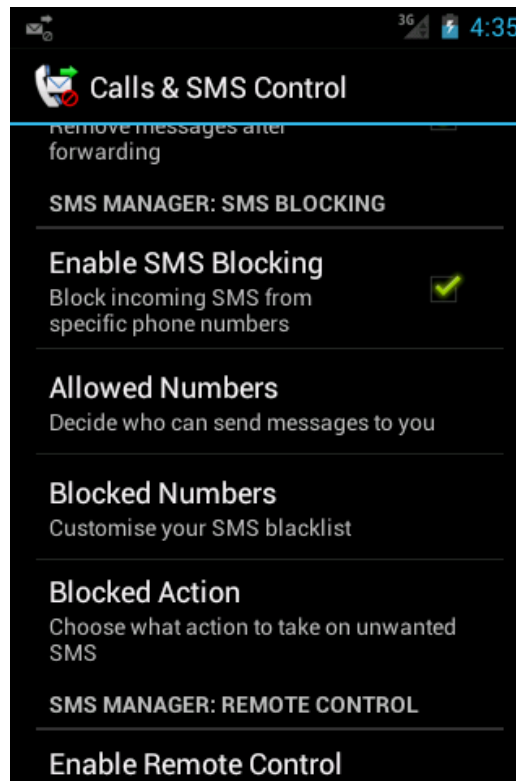


Figure 4.10.1: The user interface of SMS Blocking.

This application will help the users to delete the newly received SMS from the mobile phone when the sender is in the blacklist. With the integration in between functions, the users can use the SMS Blocking function with SMS Forwarding. In the “Blocked Action”, the users can choose to preclude the SMS to forward if it is from the unwanted sender but keep the SMS in the mobile phone, and 2 other options, as shown in the Figure 4.10.2.

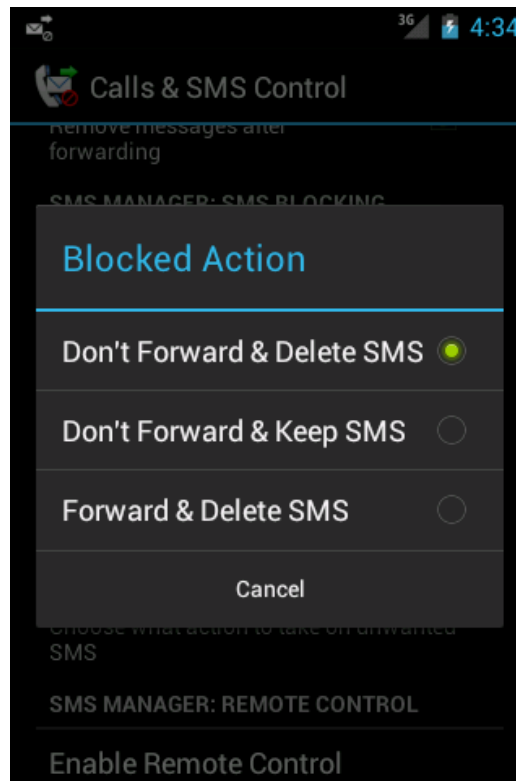


Figure 4.10.2: Several actions towards blocked SMS for users to choose.

The blacklist and whitelist for SMS Blocking are based on the same concept and the same implementation like the Call Blocking. This application also can help the users to block every incoming SMS if they choose “No SMS Allowed” in the “Allowed Numbers” setting.

4.11 Function: Remote Control

This function is to enhance the usability of the whole application by allowing the users to change some settings in this application by using SMS, which means the users can control part of this application from another mobile phone. For instance, when a user notice that he left his mobile phone in the office, he worried that he will miss out any text messages from his clients. He can use another mobile phone to send an SMS to the mobile phone that left in the office to enable the SMS Forwarding function in this application, so he will no longer missed any new incoming SMS until he gets back that mobile phone. This scenario can be illustrated using the picture below.



Figure 4.11.1: A scenario's illustration for Remote Control.

There are two types of verification being deliberated in the early stage of development, verification done using either the sender's phone number or a security pin. A security pin is a secret numeric password that shared between a user and a system, it used to authenticate the user to the system. Both of these verification method do have their own advantages and disadvantages (see Table 4.11.1).

Table 4.11.1: Comparison of different verification methods for Remote Control.

Type	Using Phone Number	Using Security Pin
Description	The application will detect if the sender's phone number is matching with the phone number predefined in the setting.	The application will detect if the security pin given in the SMS content is matching with the security pin predefined in the setting.
Advantages	<ul style="list-style-type: none"> • Faster in processing, do not need to retrieve the security pin from the SMS content before proceeding. 	<ul style="list-style-type: none"> • Accept SMS from multiple phone numbers. Users can use any mobile phones to send the SMS.
Disadvantages	<ul style="list-style-type: none"> • Inconvenient, cannot use other mobile phones to send the SMS. 	<ul style="list-style-type: none"> • Can be easily accessed by unauthorised people if they know the security pin.

Using the first method might not be that convenient to the users, due to the users can only use that particular mobile phone to control the application. Hence, using the security pin method is more preferable to be used in this application. In order to let the application to distinguish the received SMS is for remote controlling from normal SMS, there should be a keyword for it to identify it. To make it simple and memorable, the determined keyword is "RC" (case insensitive), the initials of this function's name – "Remote Control".

To standardise the user experience, this function is using the same delimiter (colon) to separate the "RC" keyword, the security pin, and following with the command. The format of the text message should be like this:

RC:<SECURITY PIN>:<COMMAND>

Recall back to the scenario given just now, the user wants to enable SMS Forwarding from another mobile phone, the security pin that he had set is "7821", and he can just compose the message like the following:

RC:7821:SMS FORWARD ON

To detect the "RC" keyword from the beginning of the text message, it can be implemented by using `String.startsWith("RC" + delimiter)`, but if the

users added one or more white space in between the keyword and the delimiter, the application will not be able to recognise that SMS as SMS for Remote Control. To overcome this problem, I make use of the *StringTokenizer* to for the implementation, below is the code segment.

```
private final String mDelimiter = ":";

private boolean isRemoteControlSms(String message) {
    StringTokenizer tokens = new StringTokenizer(message,
        mDelimiter);
    int tokensCount = tokens.countTokens();

    if (tokensCount > 2) {
        // Get first part from SMS
        String firstToken = tokens.nextToken().toUpperCase(
            Locale.ENGLISH);

        // Remove the white space from the beginning and
        // end of string
        firstToken = firstToken.trim();

        // If the first part is "RC", go to Remote Control
        if (firstToken.equals("RC")) {
            RemoteManager rmgr = new RemoteManager(mContext);
            rmgr.detectCommand(tokens);

            return true;
        }
    }

    return false;
}
```

By splitting the message using *StringTokenizer*, we can now using `String.trim()` to remove all the surrounding white space from the keyword.

The security pin used in this application is set to be in the range in between 4 to 5 numbers, instead of just 4 numbers in general usage (e.g. automated teller machines). This range of numbers is suitable and not over complicated for the users to memorise it, the users can change it from the UI. The figure below shows the UI of this Remote Control function.

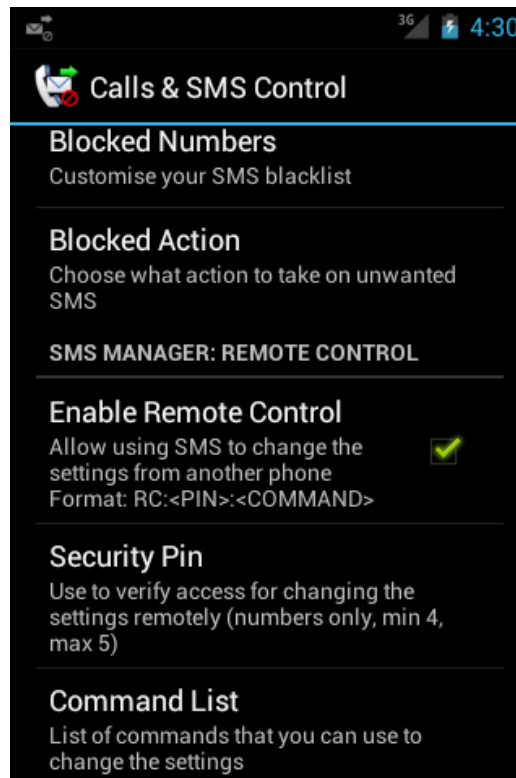


Figure 4.11.2: The user interface of Remote Control.

In Figure 4.11.2, we can see that the format for composing the message for Remote Control has been given to let the users always aware about it when they enabling the function. By default, the security pin was set to be “1234”, the users are suggested to change the default pin to own preferences.

Besides, this application should show the list of commands and including with their respective usage on a separated screen, so that those commands will not cluttering the whole UI. Another screen with a list of commands will be shown to the users when they clicking on the “Command List”, as shown in Figure 4.11.3.

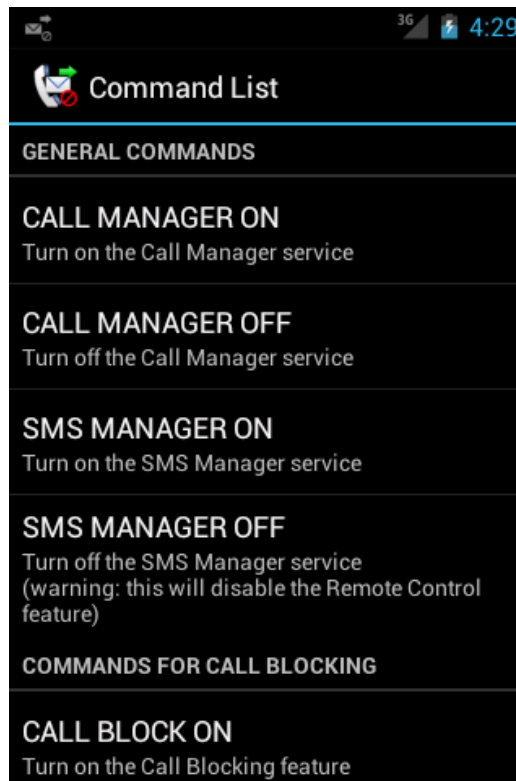


Figure 4.11.3: List of commands with respective usage for Remote Control.

I am trying to keep those commands clear and as short as possible but not using abbreviation, each of the commands will accompany by its usage explanation. Up till now, there are a total of 10 commands available, including commands for Call Manager, SMS Manager, Call Blocking, SMS Forwarding, and SMS Blocking.

In future development, this function can be improved by allowing the users to change the destination number for SMS Forwarding, and can be considered to extend the range of settings, for instance, the system settings, or volume settings.

CHAPTER 5 TESTING AND RESULTS**5.1 Test Cases for Call Manager & SMS Manager****Table 5.1.1: Black Box Test Result (Part 1).**

Test ID	Description	Expected Results	Actual Results
1	Precondition: Call Manager is not enabled. Call Manager: Enable	Call Manager enabled and a notification appears. Call Receiver registered. Other settings that are dependency on Call Manager will be activated.	Identical with expected results.
2	Precondition: Test case 1 has successfully completed. Call Manager: Disable	Call Manager disabled and the notification removed. Call Receiver unregistered. Other settings that are dependency on Call Manager will be deactivated.	Identical with expected results.
3	Precondition: SMS Manager is not enabled. SMS Manager: Enable	SMS Manager enabled and a notification appears. SMS Receiver registered. Other settings that are dependency on SMS Manager will be activated.	Identical with expected results.
4	Precondition: Test case 3 has successfully completed. SMS Manager: Disable	SMS Manager disabled and the notification removed. SMS Receiver unregistered. Other settings that are dependency on SMS Manager will be deactivated.	Identical with expected results.

5.2 Test Cases for Call Blocking

Table 5.2.1: Black Box Test Result (Part 2).

Test ID	Description	Expected Results	Actual Results
5	Precondition: Test case 1 has successfully completed. Precondition: Call Blocking is not enabled. Call Blocking: Enable	Call Blocking enabled. Other settings that are dependency on Call Blocking will be activated.	Identical with expected results.
6	Precondition: Test case 5 has successfully completed. Call Blocking: Disable	Call Blocking disabled. Other settings that are dependency on Call Blocking will be deactivated.	Identical with expected results.
7	Precondition: Test case 5 has successfully completed. Allowed Numbers: Set to "Calls From All Callers"	Allowed Numbers has been set to "Calls From All Callers".	Identical with expected results.
8	Precondition: Test case 5 has successfully completed. Allowed Numbers: Set to "Only My Contacts"	Allowed Numbers has been set to "Only My Contacts".	Identical with expected results.
9	Precondition: Test case 5 has successfully completed. Allowed Numbers: Set to "Only Favourites/Starred"	Allowed Numbers has been set to "Only Favourites/Starred".	Identical with expected results.
10	Precondition: Test case 5 has successfully completed. Allowed Numbers: Set to "No Calls Allowed"	Allowed Numbers has been set to "No Calls Allowed".	Identical with expected results.
11	Precondition: Test case 5 has successfully completed. Blocked Numbers: Add 0164005688 into the list	0164005688 added into the list.	Identical with expected results.
12	Precondition: Test case 11 has successfully completed.	0164005688 removed from the list.	Identical with expected

	Blocked Numbers: Remove 0164005688 from the list		results.
13	Precondition: Test case 5 has successfully completed. Auto End Call: Enable	Auto End Call enabled.	Identical with expected results.
14	Precondition: Test case 13 has successfully completed. Auto End Call: Disable	Auto End Call disabled.	Identical with expected results.
15	Precondition: Test case 5 has successfully completed. Remove Notification: Enable	Remove Notification enabled.	Identical with expected results.
16	Precondition: Test case 15 has successfully completed. Remove Notification: Disable	Remove Notification disabled.	Identical with expected results.
17	Precondition: Test case 7 has successfully completed. Precondition: Test case 11 has successfully completed. Precondition: Test case 13 has successfully completed. Precondition: Test case 15 has successfully completed. Precondition: 0164005688 is not in the contact list but in the blacklist. Calling in from: 0164005688	The incoming call did get blocked. The incoming call is ended. The mobile phone is not ringing. No missed-call notification appears.	Identical with expected results.
18	Precondition: Test case 7 has successfully completed. Precondition: Test case 12 has successfully completed. Precondition: Test case 13 has successfully completed. Precondition: Test case 15 has successfully completed. Precondition: 0164005688 is not in the contact list and blacklist.	The incoming call did not get blocked. The mobile phone is ringing as usual.	Identical with expected results.

	Calling in from: 0164005688		
19	Precondition: Test case 7 has successfully completed. Precondition: Test case 11 has successfully completed. Precondition: Test case 14 has successfully completed. Precondition: 0164005688 is not in the contact list but in the blacklist. Calling in from: 0164005688	The incoming call did get blocked. The mobile phone is not ringing.	Identical with expected results.
20	Precondition: Test case 7 has successfully completed. Precondition: Test case 12 has successfully completed. Precondition: Test case 14 has successfully completed. Precondition: 0164005688 is not in the contact list and blacklist. Calling in from: 0164005688	The incoming call did not get blocked. The mobile phone is ringing as usual.	Identical with expected results.
21	Precondition: Test case 8 has successfully completed. Precondition: Test case 12 has successfully completed. Precondition: Test case 13 has successfully completed. Precondition: Test case 15 has successfully completed. Precondition: 0164005688 is not in the contact list. Calling in from: 0164005688	The incoming call did get blocked. The incoming call is ended. The mobile phone is not ringing. No missed-call notification appears.	Identical with expected results.
22	Precondition: Test case 8 has successfully completed. Precondition: Test case 12 has successfully completed. Precondition: Test case 13 has successfully completed. Precondition: Test case 15 has successfully completed.	The incoming call did not get blocked. The mobile phone is ringing as usual.	Identical with expected results.

	<p>Precondition: 0164005688 is in the contact list.</p> <p>Calling in from: 0164005688</p>		
23	<p>Precondition: Test case 9 has successfully completed. Precondition: Test case 12 has successfully completed. Precondition: Test case 13 has successfully completed. Precondition: Test case 15 has successfully completed. Precondition: 0164005688 is not in the favourite list.</p> <p>Calling in from: 0164005688</p>	<p>The incoming call did get blocked. The incoming call is ended. The mobile phone is not ringing. No missed-call notification appears.</p>	<p>Identical with expected results.</p>
24	<p>Precondition: Test case 9 has successfully completed. Precondition: Test case 12 has successfully completed. Precondition: Test case 13 has successfully completed. Precondition: Test case 15 has successfully completed. Precondition: 0164005688 is in the favourite list.</p> <p>Calling in from: 0164005688</p>	<p>The incoming call did not get blocked. The mobile phone is ringing as usual.</p>	<p>Identical with expected results.</p>
25	<p>Precondition: Test case 10 has successfully completed. Precondition: Test case 12 has successfully completed. Precondition: Test case 13 has successfully completed. Precondition: Test case 15 has successfully completed. Precondition: 0164005688 is not in the favourite list.</p> <p>Calling in from: 0164005688</p>	<p>The incoming call did get blocked. The incoming call is ended. The mobile phone is not ringing. No missed-call notification appears.</p>	<p>Identical with expected results.</p>
26	<p>Precondition: Test case 10 has successfully completed. Precondition: Test case 12</p>	<p>The incoming call did get blocked. The incoming call is ended. The mobile phone is not ringing.</p>	<p>Identical with expected</p>

	<p>has successfully completed. Precondition: Test case 13 has successfully completed. Precondition: Test case 15 has successfully completed. Precondition: 0164005688 is in the favourite list.</p> <p>Calling in from: 0164005688</p>	<p>No missed-call notification appears.</p>	<p>results.</p>
--	---	---	-----------------

5.3 Test Cases for Auto Answer

Table 5.3.1: Black Box Test Result (Part 3).

Test ID	Description	Expected Results	Actual Results
27	Precondition: Test case 1 has successfully completed. Precondition: Auto Answer is not enabled. Auto Answer: Enable	Auto Answer enabled. Other settings that are dependency on Auto Answer will be activated.	Identical with expected results.
28	Precondition: Test case 27 has successfully completed. Auto Answer: Disable	Auto Answer disabled. Other settings that are dependency on Auto Answer will be deactivated.	Identical with expected results.
29	Precondition: Test case 27 has successfully completed. Precondition: Using Loudspeaker is not enabled. Using Loudspeaker: Enable	Using Loudspeaker enabled.	Identical with expected results.
30	Precondition: Test case 29 has successfully completed. Using Loudspeaker: Disable	Using Loudspeaker disabled.	Identical with expected results.
31	Precondition: Test case 27 has successfully completed. Precondition: Disallow Blocked Callers is not enabled. Using Loudspeaker: Enable	Disallow Blocked Callers enabled.	Identical with expected results.
32	Precondition: Test case 31 has successfully completed. Using Loudspeaker: Disable	Disallow Blocked Callers disabled.	Identical with expected results.
33	Precondition: Test case 11 has successfully completed. Precondition: Test case 13 has successfully completed. Precondition: Test case 15 has successfully completed. Precondition: Test case 29 has successfully completed.	The incoming call did get blocked. The incoming call is ended. The mobile phone is not ringing. No missed-call notification appears.	Identical with expected results.

	<p>Precondition: Test case 31 has successfully completed. Precondition: 0164005688 is in the blacklist.</p> <p>Calling in from: 0164005688</p>		
34	<p>Precondition: Test case 12 has successfully completed. Precondition: Test case 13 has successfully completed. Precondition: Test case 15 has successfully completed. Precondition: Test case 29 has successfully completed. Precondition: Test case 31 has successfully completed. Precondition: 0164005688 is not in the blacklist.</p> <p>Calling in from: 0164005688</p>	<p>The incoming call did not get blocked. The mobile phone is ringing as usual. The incoming call got picked-up by automatically. Loudspeaker turned on.</p>	<p>Identical with expected results.</p>
35	<p>Precondition: Test case 11 has successfully completed. Precondition: Test case 14 has successfully completed. Precondition: Test case 29 has successfully completed. Precondition: Test case 32 has successfully completed. Precondition: 0164005688 is in the blacklist.</p> <p>Calling in from: 0164005688</p>	<p>The incoming call did get blocked. The mobile phone is ringing as usual. The incoming call got picked-up by automatically. Loudspeaker turned on.</p>	<p>Identical with expected results.</p>

5.4 Test Cases for SMS Forwarding

Table 5.4.1: Black Box Test Result (Part 4).

Test ID	Description	Expected Results	Actual Results
36	Precondition: Test case 3 has successfully completed. Precondition: SMS Forwarding is not enabled. SMS Forwarding: Enable	SMS Forwarding enabled. Other settings that are dependency on SMS Forwarding will be activated.	Identical with expected results.
37	Precondition: Test case 36 has successfully completed. SMS Forwarding: Disable	SMS Forwarding disabled. Other settings that are dependency on SMS Forwarding will be deactivated.	Identical with expected results.
38	Precondition: Test case 36 has successfully completed. Precondition: Forward Empty SMS is not enabled. Forward Empty SMS: Enable	Forward Empty SMS enabled.	Identical with expected results.
39	Precondition: Test case 38 has successfully completed. Forward Empty SMS: Disable	Forward Empty SMS disabled.	Identical with expected results.
40	Precondition: Test case 36 has successfully completed. Precondition: Alphanumeric Senders is not enabled. Alphanumeric Senders: Enable	Alphanumeric Senders enabled.	Identical with expected results.
41	Precondition: Test case 40 has successfully completed. Alphanumeric Senders: Disable	Alphanumeric Senders disabled.	Identical with expected results.
42	Precondition: Test case 36 has successfully completed. Precondition: Delete Forwarded SMS is not enabled.	Delete Forwarded SMS enabled.	Identical with expected results.

	Delete Forwarded SMS: Enable		
43	Precondition: Test case 42 has successfully completed. Delete Forwarded SMS: Disable	Delete Forwarded SMS disabled.	Identical with expected results.
44	Precondition: Test case 39 has successfully completed. Precondition: Test case 42 has successfully completed. Precondition: SMS contains 2 white space Receiving SMS from: 0164005688	SMS did not forward. SMS keeps in the phone storage.	Identical with expected results.
45	Precondition: Test case 38 has successfully completed. Precondition: Test case 42 has successfully completed. Precondition: SMS contains 2 white space Receiving SMS from: 0164005688	SMS did forwarded to the destination number. SMS does not appear in the phone storage.	Identical with expected results.
46	Precondition: Test case 38 has successfully completed. Precondition: Test case 42 has successfully completed. Precondition: SMS contains 10 characters. Receiving SMS from: 0164005688	SMS did forwarded to the destination number. SMS does not appear in the phone storage.	Identical with expected results.
47	Precondition: Test case 38 has successfully completed. Precondition: Test case 43 has successfully completed. Precondition: SMS contains 10 characters. Receiving SMS from: 0164005688	SMS did forwarded to the destination number. SMS keeps in the phone storage.	Identical with expected results.
47	Precondition: Test case 40 has successfully completed. Precondition: Test case 42 has successfully completed. Precondition: SMS contains	SMS did forwarded to the destination number. SMS does not appear in the phone storage.	Identical with expected results.

	100 characters. Receiving SMS from: DiGiRewards		
48	Precondition: Test case 41 has successfully completed. Precondition: Test case 42 has successfully completed. Precondition: SMS contains 100 characters. Receiving SMS from: DiGiRewards	SMS did not forward. SMS keeps in the phone storage.	Identical with expected results.

5.5 Test Cases for SMS Blocking

Table 5.5.1: Black Box Test Result (Part 5).

Test ID	Description	Expected Results	Actual Results
49	Precondition: Test case 3 has successfully completed. Precondition: SMS Blocking is not enabled. SMS Blocking: Enable	SMS Blocking enabled. Other settings that are dependency on SMS Blocking will be activated.	Identical with expected results.
50	Precondition: Test case 49 has successfully completed. SMS Blocking: Disable	SMS Blocking disabled. Other settings that are dependency on SMS Blocking will be deactivated.	Identical with expected results.
51	Precondition: Test case 49 has successfully completed. Allowed Numbers: Set to "SMS From All Senders"	Allowed Numbers has been set to "SMS From All Senders".	Identical with expected results.
52	Precondition: Test case 49 has successfully completed. Allowed Numbers: Set to "Only My Contacts"	Allowed Numbers has been set to "Only My Contacts".	Identical with expected results.
53	Precondition: Test case 49 has successfully completed. Allowed Numbers: Set to "Only Favourites/Starred"	Allowed Numbers has been set to "Only Favourites/Starred".	Identical with expected results.
54	Precondition: Test case 49 has successfully completed. Allowed Numbers: Set to "No SMS Allowed"	Allowed Numbers has been set to "No SMS Allowed".	Identical with expected results.
55	Precondition: Test case 49 has successfully completed. Blocked Numbers: Add 0164005688 into the list	0164005688 added into the list.	Identical with expected results.
56	Precondition: Test case 55 has successfully completed. Blocked Numbers: Remove 0164005688 from the list	0164005688 removed from the list.	Identical with expected results.
57	Precondition: Test case 5	Blocked Action has been set to	Identical

	has successfully completed. Blocked Action: Set to “Don’t Forward & Delete SMS”	“Don’t Forward & Delete SMS”.	with expected results.
58	Precondition: Test case 5 has successfully completed. Blocked Action: Set to “Don’t Forward & Keep SMS”	Blocked Action has been set to “Don’t Forward & Keep SMS”.	Identical with expected results.
59	Precondition: Test case 5 has successfully completed. Blocked Action: Set to “Forward & Delete SMS”	Blocked Action has been set to “Forward & Delete SMS”.	Identical with expected results.
60	Precondition: Test case 51 has successfully completed. Precondition: Test case 55 has successfully completed. Precondition: Test case 57 has successfully completed. Precondition: 0164005688 is not in the contact list but in the blacklist. Receiving SMS from: 0164005688	The incoming SMS did get blocked. The SMS does not stored in the phone storage.	Identical with expected results.
61	Precondition: Test case 51 has successfully completed. Precondition: Test case 56 has successfully completed. Precondition: Test case 57 has successfully completed. Precondition: 0164005688 is not in the contact list and the blacklist. Blocked Numbers: Remove 0164005688 from the list	The incoming SMS did not get blocked. The SMS does stored in the phone storage.	Identical with expected results.
62	Precondition: Test case 36 has successfully completed. Precondition: Test case 51 has successfully completed. Precondition: Test case 55 has successfully completed. Precondition: Test case 57 has successfully completed. Precondition: 0164005688 is not in the contact list but	The incoming SMS did get blocked. The SMS does not forward. The SMS does not stored in the phone storage.	Identical with expected results.

	<p>in the blacklist.</p> <p>Receiving SMS from: 0164005688</p>		
63	<p>Precondition: Test case 36 has successfully completed. Precondition: Test case 51 has successfully completed. Precondition: Test case 55 has successfully completed. Precondition: Test case 58 has successfully completed. Precondition: 0164005688 is not in the contact list but in the blacklist.</p> <p>Receiving SMS from: 0164005688</p>	<p>The incoming SMS did get blocked. The SMS does not forward. The SMS does stored in the phone storage.</p>	<p>Identical with expected results.</p>
64	<p>Precondition: Test case 36 has successfully completed. Precondition: Test case 51 has successfully completed. Precondition: Test case 55 has successfully completed. Precondition: Test case 59 has successfully completed. Precondition: 0164005688 is not in the contact list but in the blacklist.</p> <p>Receiving SMS from: 0164005688</p>	<p>The incoming SMS did get blocked. The SMS does forward. The SMS does not stored in the phone storage.</p>	<p>Identical with expected results.</p>

5.6 Test Cases for Remote Control

Table 5.6.1: Black Box Test Result (Part 6).

Test ID	Description	Expected Results	Actual Results
65	Precondition: Test case 3 has successfully completed. Precondition: Remote Control is not enabled. Remote Control: Enable	Remote Control enabled. Other settings that are dependency on Remote Control will be activated.	Identical with expected results.
66	Precondition: Test case 65 has successfully completed. Remote Control: Disable	Remote Control disabled. Other settings that are dependency on Remote Control will be deactivated.	Identical with expected results.
67	Precondition: Test case 65 has successfully completed. Security Pin: Change to 101	A dialog prompted to notify that the security pin cannot be less than 4 numbers.	Identical with expected results.
68	Precondition: Test case 65 has successfully completed. Security Pin: Change to empty	A dialog prompted to notify that the security pin cannot be less than 4 numbers.	Identical with expected results.
69	Precondition: Test case 65 has successfully completed. Security Pin: Change to 74283	The new security pin is saved.	Identical with expected results.
70	Precondition: Test case 65 has successfully completed. Precondition: Test case 69 has successfully completed. Receiving SMS from: 0164005688 SMS content: RC:74283:CALL block On	Call Blocking is enabled by using SMS. The SMS does not stored in the phone storage.	Identical with expected results.
71	Precondition: Test case 65 has successfully completed. Precondition: Test case 69 has successfully completed.	SMS Blocking remaining unchanged. The SMS does not stored in the phone storage.	Identical with expected results.

	<p>Receiving SMS from: 0164005688 SMS content: RC:74283:sms Blocking On</p>		
72	<p>Precondition: Test case 65 has successfully completed. Precondition: Test case 69 has successfully completed.</p> <p>Receiving SMS from: 0164005688 SMS content: RC:7428:sms Block On</p>	<p>Security pin is invalid. SMS Blocking remaining unchanged. The SMS does not stored in the phone storage.</p>	Identical with expected results.
73	<p>Precondition: Test case 5 has successfully completed. Precondition: Test case 65 has successfully completed. Precondition: Test case 69 has successfully completed.</p> <p>Receiving SMS from: 0164005688 SMS content: rc:74283: CALL block off</p>	<p>Call Blocking is disabled by using SMS. The SMS does not stored in the phone storage.</p>	Identical with expected results.
74	<p>Precondition: Test case 4 has successfully completed. Precondition: Test case 65 has successfully completed. Precondition: Test case 69 has successfully completed.</p> <p>Receiving SMS from: 0164005688 SMS content: Rc : 74283 : SMS manager on</p>	<p>SMS Manager is enabled by using SMS. A new notification appeared. SMS Receiver registered. Other settings that are dependency on SMS Manager will be activated. The SMS does not stored in the phone storage.</p>	Identical with expected results.
75	<p>Precondition: Test case 3 has successfully completed. Precondition: Test case 65 has successfully completed. Precondition: Test case 69 has successfully completed.</p> <p>Receiving SMS from:</p>	<p>SMS Manager is disabled by using SMS. The notification is removed. SMS Receiver unregistered. Other settings that are dependency on SMS Manager will be deactivated. The SMS does not stored in the</p>	Identical with expected results.

	0164005688 SMS content: Rc: 74283 : SmS manager off	phone storage.	
--	--	----------------	--

5.7 Summary

After completing the test cases above, we can determine that this application does perform as stated in specifications and requirements, by using the Black Box testing technique. All of the functions are performing as it defined.

CHAPTER 6 CONCLUSION AND DISCUSSION

6.1 Project Review

The main purpose of this project is to allow the users who are using the Android-powered mobile phone to experience the ability of taking control over the incoming calls and text messages that they are receiving from time to time. For instance, forwarding calls and SMS (without relying on the mobile network operators), filtering the mobile phone numbers with whitelist and blacklist technique, and auto answering the phone calls.

As a result, this project has been successfully completed within the time frame, and achieved most of the project objectives that had been stated in the proposal.

In this development project, I have been getting on the way to experience the Android development environment and as well as getting to know the possibilities and limitations in Android programming. This is absolutely useful for my future career.

In the end of this project, the Phone Calls and Text Messages Control application is able to perform call filtering, call auto answering, SMS forwarding, SMS filtering and remote control the application by using SMS. The whole application has met its objectives to help the users to control the phone calls and SMSs, need not to worry about the capability of their carriers.

There have been a total of 5 incremental versions released, including the final version (refers to Chapter 4.3). Each version usually includes one or more new and workable function added, other bug fixes and improvements, each of the components are added in part by part until the final upshot has been produced. Here is the list of achievements of this project:

- The application is able to support multiple Android platforms, from version 2.2 up to 4.0.4, devices with other versions might not be fully supporting all the functions that available.
- The application is able to help the users to forward and reply to SMS without revealing the users' private mobile phone number.

- The application is able to filter the incoming calls and SMS by using customisable whitelist and blacklist.
- The application is able to pick-up the incoming call automatically to assist the users while driving.
- The application is able to let the authorised users to change the application settings remotely by using SMS.

6.2 Problems Encountered

Despite the fact that this project is completed, but unfortunately there is one function not able to deliver to the end users, which is the Call Forwarding function. There are some constraints and restrictions faced while developing this function, as a result this useful function has to be discarded due to the telephony-related APIs and permission have been reserved only to system apps, the user apps are not able to do so.

Besides that, there is also another problem encountered where the developers are not able to request the apps to pick-up the incoming calls programmatically, but this can be solved by triggering a Bluetooth headset hook, and this method is only works for Android versions that below 4.1.

6.3 Future Work

Nothing is perfect. This application does have a lot more space for improvement to provide more flexibility and controllability. In future development, this application can add in a search feature into the blacklist of Call Blocking and SMS Blocking to ease the users while searching through a long list.

Next, the Remote Control feature can be further enhanced by providing more handy commands for the users to use, and to extend the range of settings to system-wide settings can be considered. Allowing the users to choose their own preferable delimiter in Remote Control and SMS Forwarding can make this application more customisable.

Besides that, another new feature is also suitable to implement it in this application where it can help the user to send a user-defined message to the caller if the user is busy and failed to answer the incoming call.

Lastly, further research can be done in the Call Forwarding function, continue to find out an alternative way to achieve it, instead of attempting the conference call method.

6.4 Conclusion

Sincere thanks for the opportunity given to challenge this project, after going through all the phases in this project, I have been improving my report writing skill, learning more about Android OS and its programming nature by practically experiencing on it.

This aim of this application is to let the Android-powered mobile phone users to have more controls over the incoming calls and SMS that they receive, without much relying on the mobile network operators or subscribing to the value added services provided, whereby the users usually needs to pay for monthly subscription fees for those services.

Regrettably, the call forwarding function is failing to achieve with the conference call technique, due to lack of support from the Android SDK. However, with the suggestions and advices given by my project supervisor, Mr Liew, this project can be going on successfully with other useful functions that met with the scopes and objectives of this project.

In a nutshell, this project is a very good challenge to me and I believe that the upshot of this project will bring convenience and benefits to many of the Android-powered mobile phone users.

REFERENCE

Ableson, F., 2009. *Introduction to Android development*. [Online]

Available at: <http://www.ibm.com/developerworks/library/os-android-devel/>

[Accessed 9 March 2012].

AppBrain, 2013. *Android Statistics > Number of available Android applications*. [Online]

Available at: <http://www.appbrain.com/stats/number-of-android-apps>

[Accessed 19 February 2013].

Association of Modern Technologies Professionals, n.d. *Software Development Methodologies*. [Online]

Available at: <http://www.itinfo.am/eng/software-development-methodologies/>

[Accessed 28 February 2013].

Barra, H., 2012. *500 million*. [Online]

Available at:

<https://plus.google.com/app/basic/stream/z12mjfgpozyie3y0j04cczrooo3uj5zi1jc>

[Accessed 5 March 2013].

Bird, T., 2009. *Android Architecture*. [Online]

Available at: http://elinux.org/Android_Architecture

[Accessed 7 March 2012].

Bozz, B., 2013. *VoiceMail & Call Forwarding Deactivation & Activation*. [Online]

Available at:

https://getsatisfaction.com/celcom/topics/voicemail_call_forwarding_deactivation_activation

[Accessed 3 March 2013].

Brahler, S., 2010. *Analysis of the Android Architecture*. [Online]

Available at: http://os.ibds.kit.edu/downloads/sa_2010_braehler-stefan_android-architecture.pdf

[Accessed 7 March 2012].

Briden, P., 2012. *Android 4.2 Jelly Bean lands on Nexus 7*. [Online]

Available at:

http://www.knowyourmobile.com/blog/1678521/android_42_jelly_bean_lands_on_nexus_7.html

[Accessed 5 March 2013].

Celcom Axiata Berhad, n.d. *Business 1+5*. [Online]

Available at: http://www.celcom.com.my/biz/products.php?page=familyplan_biz1plus5

[Accessed 3 March 2013].

Celcom Axiata Berhad, n.d. *P28*. [Online]

Available at: http://www.celcom.com.my/biz/products.php?page=voiceplan_p28

[Accessed 3 March 2013].

Centers for Disease Control and Prevention, 2012. *Distracted Driving*. [Online]
Available at: http://www.cdc.gov/Motorvehiclesafety/Distracted_Driving/index.html
[Accessed 28 February 2013].

Centers for Medicare & Medicaid Services, 2008. *Selecting a development approach*. [Online]
Available at: <http://www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information-Technology/XLC/Downloads/SelectingDevelopmentApproach.pdf>
[Accessed 28 February 2013].

Chengalva, 2012. *Android – Dalvik Virtual Machine (DVM) – Android Runtime. Differences to a normal Java VMs? What is .dex format?*. [Online]
Available at: <http://chengalva.com/2012/10/29/android-dalvik-virtual-machine-dvm-android-runtime/>
[Accessed 6 March 2013].

DiGi Corporate, n.d. *DG Family™ Postpaid*. [Online]
Available at: <http://www.digi.com.my/postpaid/dgfamily.do>
[Accessed 3 March 2013].

DiGi Corporate, n.d. *DiGi SmartBizPlan*. [Online]
Available at: <http://www.digi.com.my/business/productsandservices/callplans.do>
[Accessed 3 March 2013].

Garrison, J., 2010. *What is the Linux Kernel and What Does It Do?*. [Online]
Available at: <http://www.howtogeek.com/howto/31632/what-is-the-linux-kernel-and-what-does-it-do/>
[Accessed 7 March 2012].

GeckoBeach, n.d. *GSM feature codes*. [Online]
Available at: <http://www.geckobeach.com/cellular/secrets/gsmcodes.php>
[Accessed 15 February 2013].

Google Inc., n.d. *Google Voice*. [Online]
Available at:
<https://play.google.com/store/apps/details?id=com.google.android.apps.googlevoice&hl=en>
[Accessed 31 March 2012].

Google, 2008. *Device Requirements*. [Online]
Available at:
http://www.netmite.com/android/mydroid/development/pdk/docs/system_requirements.html
[Accessed 9 March 2012].

Google, 2011. *Google Voice for mobile*. [Online]
Available at: <http://www.google.com/mobile/voice/>
[Accessed 31 March 2012].

Google, 2013. *Dashboards | Android Developers*. [Online]
Available at: <http://developer.android.com/about/dashboards/index.html>
[Accessed 5 March 2013].

Google, n.d. *App Framework*. [Online]
Available at: <http://developer.android.com/about/versions/index.html>
[Accessed 18 February 2013].

Google, n.d. *Philosophy and Goals*. [Online]
Available at: <http://source.android.com/about/philosophy.html>
[Accessed 1 March 2012].

Google, n.d. *Processes and Threads*. [Online]
Available at: <http://developer.android.com/guide/components/processes-and-threads.html>
[Accessed 16 February 2013].

Google, n.d. *Settings | Android Developers*. [Online]
Available at: <http://developer.android.com/guide/topics/ui/settings.html>
[Accessed 10 March 2013].

Hashimi, S., Komatineni, S. & MacLean, D., 2010. *How the Dalvik Virtual Machine Works on Google Android*. [Online]
Available at: <http://www.ctoedge.com/content/how-dalvik-virtual-machine-works-google-android>
[Accessed 8 March 2012].

Infragistics, 2013. *The Importance Of A Good Icon*. [Online]
Available at:
<http://www.infragistics.com/community/blogs/marketing/archive/2013/01/14/the-importance-of-a-good-icon.aspx>
[Accessed 8 March 2013].

Janecek, J., 2011. *Android 2.3 cant get MODIFY_PHONE_STATE..* [Online]
Available at: <https://code.google.com/p/android/issues/detail?id=14789>
[Accessed 13 March 2013].

Jones, M. T., 2011. *Application virtualization, past and future*. [Online]
Available at: <http://www.ibm.com/developerworks/linux/library/l-virtual-machine-architectures/>
[Accessed 9 March 2012].

Kumar, S., 2011. *Dalvik Virtual Machine insights*. [Online]
Available at: <http://santhosh0705.wordpress.com/2011/08/25/vms-and-dalvik-vm/>
[Accessed 15 February 2013].

Maker, F., 2011. *Android Dalvik Virtual Machine*. [Online]
Available at: <http://jaxenter.com/android-dalvik-virtual-machine-35498.html>
[Accessed 22 February 2012].

Maxis Bhd, n.d. *FAQ - Maxis Consumer*. [Online]

Available at: <http://www.maxis.com.my/faq/default.asp?strWebsite=maxiscom&catID=25>
[Accessed 3 March 2013].

Maxis Bhd, n.d. *Maxis Postpaid Plans*. [Online]

Available at: <http://www.maxis.com.my/postpaid/main.asp>
[Accessed 3 March 2013].

Maxis Bhd, n.d. *Mobile & Plans - Rate Plans & Charges*. [Online]

Available at:
http://www.maxis.com.my/personal/mobile/rate/family_plus_30.asp?iStruct=0:0:1:0
[Accessed 3 March 2013].

Maxis Bhd, n.d. *Mobile Solutions - Business Value Share*. [Online]

Available at: http://www.maxis.com.my/business/business_value_share.asp?iStruct=0:0
[Accessed 3 March 2013].

Maxis Bhd, n.d. *SME - Voice - Mobile Services - VAS Mobile - Call Forwarding*. [Online]

Available at:
http://www.maxis.com.my/business/sme/voice/mobile/vas/call_manag/forwarding.asp
[Accessed 3 March 2013].

mobiThinking, 2012. *Global mobile statistics 2012 Part A: Mobile subscribers; handset market share; mobile operators*. [Online]

Available at: <http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats/a>
[Accessed 23 February 2013].

Pixelchef, 2011. *PassItForward*. [Online]

Available at: <https://play.google.com/store/apps/details?id=pixelkitchen.passitforward>
[Accessed 31 March 2012].

Resco Developer Tools, 2011. *Developing Android Applications in .NET*. [Online]

Available at: <http://www.codeproject.com/Articles/195450/Developing-Android-Applications-in-NET>
[Accessed 1 March 2012].

Rial, J. B. & Rodríguez Silva, D. A., n.d. *Java history*. [Online]

Available at: http://papa.det.uvigo.es/~theiere/cursos/Curso_Java/history.html
[Accessed 9 March 2012].

Rose India Technologies Pvt. Ltd., 2007. *What is Java, it's history?*. [Online]

Available at: <http://www.roseindia.net/java/beginners/what-is-java.shtml>
[Accessed 14 February 2012].

Sayed, H., Satya, K. & Dave, M., 2010. *How the Dalvik Virtual Machine Works on Google Android*. [Online]

Available at: <http://www.ctoedge.com/content/how-dalvik-virtual-machine-works-google->

android

[Accessed 15 February 2012].

Shen, T., 2011. *SMS Forwarding*. [Online]

Available at: <https://play.google.com/store/apps/details?id=com.intensoft.smstransfer>

[Accessed 31 March 2012].

Shi, Y., Gregg, D., Beatty, A. & Ertl, M. A., 2005. *Virtual Machine Showdown: Stack Versus Registers*. [Online]

Available at: http://static.usenix.org/events/vee05/full_papers/p153-yunhe.pdf

[Accessed 20 March 2013].

Singapore Police Force, 2012. *Singapore Police Force FAQs*. [Online]

Available at: http://www.ifaq.gov.sg/spf/apps/fcd_faqlmain.aspx#TOPIC_6125

[Accessed 23 February 2013].

Skype, n.d. *Skype on your Mobile*. [Online]

Available at: <http://www.skype.com/intl/en-us/get-skype/on-your-mobile>

[Accessed 31 March 2012].

Strickland, J., 2007. *How the Google Phone Works*. [Online]

Available at: <http://electronics.howstuffworks.com/google-phone.htm>

[Accessed 7 March 2012].

Techvision Systems, n.d. *Call Forward*. [Online]

Available at: http://www.androidzoom.com/android_applications/communication/call-forward_pwnt.html

[Accessed 31 March 2012].

Techvision Systems, n.d. *Call Forward +*. [Online]

Available at: http://www.androidzoom.com/android_applications/productivity/call-forward_bkxja.html

[Accessed 31 March 2012].

Telekom Malaysia Berhad, n.d. *Businessline Wireless*. [Online]

Available at: <http://www.tm.com.my/sme/products/voice/Pages/BusinesslineWireless.aspx>

[Accessed 3 March 2013].

Telekom Malaysia Berhad, n.d. *Call Transfer*. [Online]

Available at:

<http://www.tm.com.my/sme/products/ComplementaryServices/Pages/CallTransfer.aspx>

[Accessed 3 March 2013].

WebGate, n.d. *Advanced Call Manager by WebGate*. [Online]

Available at: <http://my-symbian.com/s60v3/software/applications.php?faq=1&fldAuto=73>

[Accessed 31 March 2012].

Wikipedia contributors, 2003. *Call forwarding*. [Online]
Available at: http://en.wikipedia.org/wiki/Call_forwarding
[Accessed 5 March 2012].

Wikipedia contributors, 2012. *Software development kit*. [Online]
Available at: http://en.wikipedia.org/wiki/Software_development_kit
[Accessed 23 February 2012].

Wireless Labs Technologies, n.d. *SMS Forwarder for Symbian 1.20*. [Online]
Available at: http://www.downloadplex.com/Mobile/Symbian/Application/sms-forwarder-for-symbian_225508.html
[Accessed 31 March 2012].

APPENDIX A GLOSSARY

This glossary has been included to assist readers who may be unfamiliar with some of the technical terms used.

Application Programming Interface (API) Including is a set of routines, protocols, and tools for building software applications.

Blacklist A list of items that are denied access to a certain system or protocol. It uses to allow access from all items, except those included the list. (*antonym: whitelist*)

Byte code A term that used to represent various forms of instruction sets designed for efficient execution by a software interpreter, also suitable for further compilation into machine code.

Call forwarding / Call diverting A telephone service feature whereby, when a customer chooses, all calls coming into one number are automatically transferred to another, designated number.

Mobile Network Operator / Carrier Also known as mobile phone operator, wireless service provider, wireless carrier, or cellular company, is a telephone company that provides phone services for mobile phone subscribers.

Open-source Source code that is available to the public without charge. Open-source code is often enhanced, improved, and adapted for specific purposes by interested programmers, with the revised versions of the code are made available to the public.

Operating system (OS) The collection of software that directs a computer's operations, controlling and scheduling the execution of

other programs, and managing storage, input/output, and communication resources.

Prepaid	Mobile phone plan for which credit is purchased in advance of service use.
Process	In computing, it is an instance of a computer program that is being executed, containing the program code and its current activity.
Postpaid	Mobile phone plan for which the user is billed after the fact according to their use of services at the end of each month.
Quarter Video Graphics Array (QVGA)	A display standard used in the mobile device market, it refers to a screen resolution of 320 pixels x 240 pixels, and the aspect ratio is 4:3.
Thread	In computing, a thread is the smallest unit of processing that can be scheduled by an operating system. A program or process can have two or more concurrently running threads at the same time.
Whitelist	A list of items that are granted access to a certain system or protocol. It uses to deny access to all items, except those included in the list. (<i>antonym: blacklist</i>)

APPENDIX B ANDROID PLATFORM VERSIONS

Android keeps releasing new version updates to its consumers from time to time, as of 2013, there are over 500 million devices actively using the Android OS in the worldwide (Barra, 2012). The most recent release was the Jelly Bean (4.2), which was released in November 2012 (Briden, 2012).

List of Android versions:

1. Android 1.0
2. Android 1.1
3. Android 1.5 (Cupcake)
4. Android 1.6 (Donut)
5. Android 2.1 (Éclair)
6. Android 2.2 (Froyo)
7. Android 2.3 – 2.3.7 (Gingerbread)
8. Android 3.1 – 3.2 (Honeycomb)
9. Android 4.0.3 – 4.0.4 (Ice Cream Sandwich)
10. Android 4.1 – 4.2 (Jelly Bean)

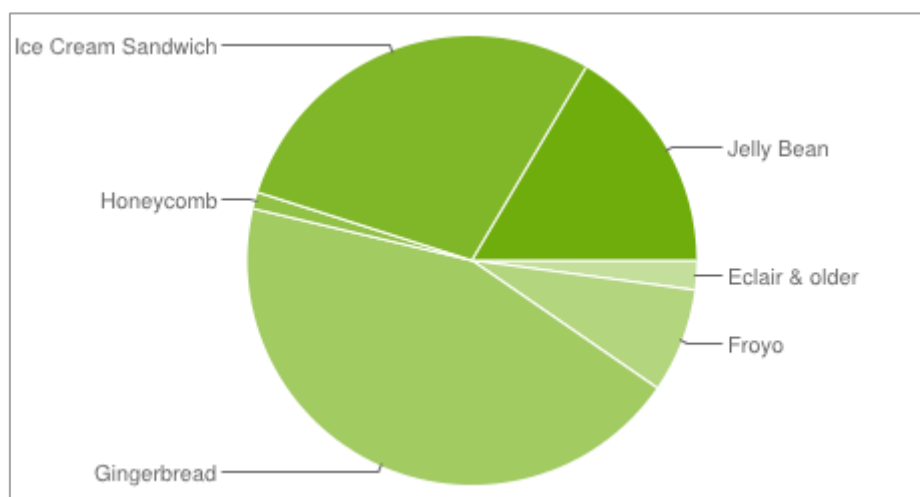


Figure B.1: Distribution of the Android Platform Versions.

(Source: <http://developer.android.com/about/dashboards/index.html>)

According to the current distribution statistic shown in Android Developers site (Google, 2013), the two major versions that distributed the most are the Gingerbread (Android 2.3 - 2.3.7) with 44.1%, following by the Ice Cream Sandwich (Android 4.0.3 - 4.0.4) with 28.6%. Table below shows the detailed distribution for each version.

Table B.1: Distribution of Android devices that have accessed Google Play, ending on 4 March, 2013.

Version	Codename	API	Distribution
1.6	Donut	4	0.2%
2.1	Eclair	7	1.9%
2.2	Froyo	8	7.5%
2.3 - 2.3.2	Gingerbread	9	0.2%
2.3.3 - 2.3.7		10	43.9%
3.1	Honeycomb	12	0.3%
3.2		13	0.9%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	28.6%
4.1	Jelly Bean	16	14.9%
4.2		17	1.6%

(Source: <http://developer.android.com/about/dashboards/index.html>)

By getting to know this statistic, we can develop the applications targeting to the major group of Android users.

APPENDIX C BIWEEKLY REPORTS AND OTHER ATTACHED DOCUMENTS

The attachments below are including my biweekly reports and other documents.