Anafi Babatunde

ANDROIDSOFTWARE DEVELOPMENT PROCESS. Case study: ChronometerX

Thesis

CENTRAL OSTROBOTHNIA UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Information Technology

May 2015

Centria
UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

Unit	Date	Author/s
Kokkola- Pietarsaari	May 2015	Babatunde Anafi

Degree programme
Information Technology

Name of thesis
Android software development process: Case study of ChronometerX

Instructor	Pages
Kauko Kolehmainen	[71]

Supervisor
Kauko Kolehmainen

Artificial intelligence is advancing rapidly, Integrated Circuitry technology is progressing swiftly and computer processor speed is increasing. Thus, the future of mobile computers seems increasingly thrilling. Compact and small mobile computers are gradually gaining ground in the computer world. The future of Computers may be handheld, wearable or even smaller.

The human personal lifestyle and working lifestyle have changed dramatically; our mobile devices continue to influence our lifestyle. The number of active mobile devices and human-beings crossed over somewhere around the 7.19 billion mark, and Android phones account for about 80% of them. Thus, this thesis is based on Android software development process using ChronometerX (the Android application developed during the project) as a case study.

The aim of this thesis is to report the six phases of the development life cycle of ChronometerX: requirement gathering and analysis, design, implementation or coding, testing, deployment and maintenance. Requirement gathering and analysis is the first stage in the development of an application. It involves collecting and analysing the functions and services a proposed system should perform. The design and implementation stage is the point where a functional software is developed. The software developed is tested with respect to the requirements collected during requirement gathering and analysis. The verified software is published to users at the deployment phase. The last phase in the life cycle of a software is the maintenance process. Software maintenance involves providing a cost effective support to an application and retiring the application if necessary. Application development is a never-ending story; the story ends when the application is retired.

Key words
Android application, Mobile software development, software design, software implementation, software requirement and analysis, software testing, software maintenance.

CONTENTS

1 INTRODUCTION

The advancement in technology has increased the capacity and the capability of mobile devices drastically. Smartphones unlike classic mobile phones can now perform computer-like function. Thus, they are computers with limited features. The smartphone is now more powerful than it has ever been, a smartphone can now be used for various purposes aside from just making calls and sending SMSs. These devices can now be utilized for multiple tasks that will normally require different gadgets like digital camera and music player.

The Android Play Store is a large and lucrative market for mobile software developers. Android devices accounting for more than 80% of activated mobile devices. With this number of activated Android device users; the Play Store seems like a gold mine to software developers. The fast increasing number of Android mobile phone users has attracted many software developers. Thus, the rate at which the number of applications in Android Play Store grows is enormous; there is always more than one application for the same purpose. This situation has challenged developers to become more innovative in their development.

The goal of this project is to recreate the mobile clock application and document the six stages of the development life cycle of the application; requirement gathering and analysis, design, implementation or coding, testing, deployment and maintenance. This project's goal was achieved by integrating five different applications into one application package: Alarm, Calendar, Stopwatch, Timer and To-do list and also deliver the summary of upcoming events or tasks to users on a separate screen (Activity). The major innovation in the project is the summary screen. The ChronometerX is an alpha level application and is yet to be published in the Android Play Store. This thesis only describes the development cycle and does not include the actual implementation process (the coding phase).

The application (ChronometerX) reported in this thesis was built using Eclipse Android Development Environment. The programming language used is Java. The GUI (Graphical user interface) was developed using XML (Extensible Markup Language), but controlled

with Java programming language and the images used in the application were edited with Adobe Photoshop CS6. The feasibility study conducted during ChronometerX's development was mainly downloading, using and analyzing related applications from the Android Play Store. The Minimum Required SDK version for the ChronometerX is API 17 (Android 4.2, JELLY_BEAN_MR1) and the Target SDK version is API 21 (Android 5.0, LOLLIPOP). The Application prototype was tested on various real Android devices like: Samsung Galaxy Trend, OnePluse1, and Samsung Galaxy Note 2. The application was also tested on virtual devices: Samsung Galaxy S5 and Google Nexus6. Genymotion was the Android emulator used during application testing.

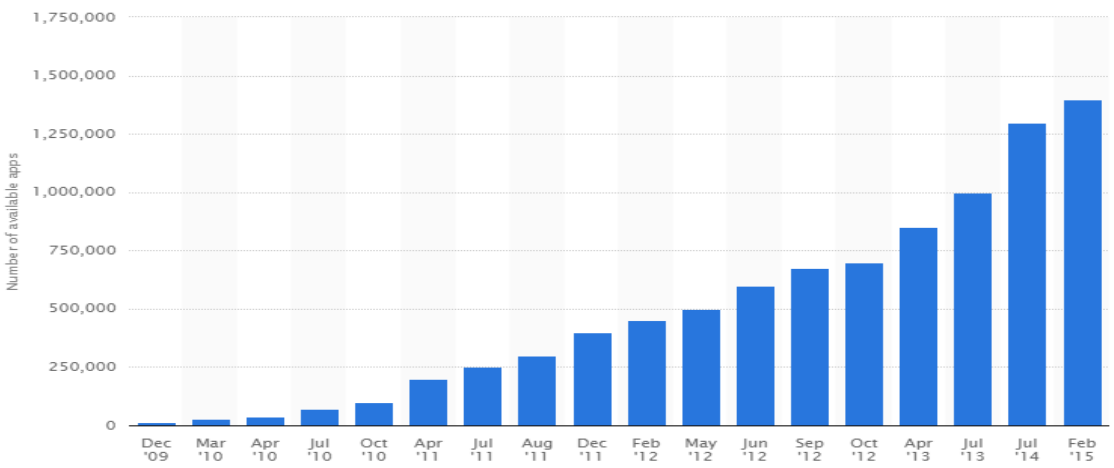
The contents of this thesis work starts by describing the Android operating system (OS) and Android software development process, followed by and illustration of what ChronometerX is. These descriptions are followed by a consecutive explanation of ChronometerX's Application life cycle: requirement gathering and analysis, design, implementation or coding, testing, deployment, and maintenance. Requirement gathering and analysis is the first stage in the development of an application. It is where the software's requirements are gathered. The designing phase involves identification and conceptualization of software components, including their relationships and connections, based on requirements collected during the requirement gathering and analysis phase.

The implementation or coding stage is where the actual programming occurs to produce a working application. The application is tested against the acquired requirements in the testing stage and published in the deployment phase. Application development is a never-ending story. Thus, the maintenance process is the last stage in the software development life cycle. Software maintenance involves providing a cost effective support to an application and retiring the application if necessary.

2 ANDROID OPERATING SYSTEM (OS)

The Android operating system powers more than a billion smartphones and tablets. The operating system (OS) based on the Linux kernel has a user interface entrenched on direct manipulation. Android was fundamentally designed for touchscreen mobile devices such as smartphones and tablet computers. Android has been adapted with specialized user interfaces for televisions (Android TV), cars (Android Auto), game consoles, digital cameras, regular PCs and even wrist watches (Android Wear). Since Android is basically designed for touchscreen mobile, it employs the uses of touch inputs that mimic everyday human actions like tapping, pinching, swiping, and reverse pinching to control objects on the screen. It also includes a virtual keyboard; some Android devices also provide a physical keyboard for text manipulation. Google currently develops Android; each version of Android operating system is named after a dessert. For example, Android 5.0 is called Lollipop. (Android Open Source Project 2015c; Sheusi 2013.)

Google Play, formally know as Android Market (released in October 2008) is Google's official store and portal for Android applications and games including other digital media contents such as movies, music, eBooks magazines, films and TV. Applications that require no fee to download are available all over the world and paid applications are obtainable only in certain countries. Physical devices and accessories, such as phones, tablets, chromecast (TV and Video), device cases and cords are also available on Play Store (Nickinson 2015). As of February 2015 the number of applications on Play Store was estimated to be around 1.4 million and the applications downloads from Play Store is approximately 50 billion. The Graph below shows a statistic of the estimated figures of applications available on the Google Play Store from December 2009 to February 2015. (The Statistics Portal 2015.)



GRAPH 1. The number of available applications in the Google Play Store from December 2009 to February 2015(The Statistics Portal 2015.)

2.1 Android Interface

The Android input subsystem enables input from various device classes; touchscreen, keyboard, joystick, mouse, and trackball. For mobile devices running the Android operating system the default device class is the touch screen. The touch screen employs direct manipulation that simulates everyday human actions. For example, tapping, pinching, swiping, and reverse pinching to control objects on the screen, including a virtual keyboard. The screen is designed to provide fluid touch interfaces that respond to user input immediately. Devices may also give haptic feedbacks utilizing the vibration capabilities of an Android device. Android applications may also utilize internal hardware such as accelerometers, gyroscopes and proximity sensors to respond to additional user actions. Such respond is established when a user changes the screen orientation from portrait to landscape or vice versa (depending on how the device is oriented) by rotating the device. (Android Open Source Project 2015d; Android Open Source Project 2015e.)

When an Android device is switched on, it boots to the home-screen (if it is not locked with a passcode). The Android home screen is similar to the PC desktop, the Android home-screen primarily consists of application icons and widgets. When an application icon is clicked or taped it launches the associated application. On the other hand, the

Android widgets display current or auto-updating content; for example, time, a news ticker, weather forecast, and similar updates directly on the home-screen. A typical Android home-screen may contain numerous pages; which user can swipe trough. However, an Android device home-screen is profoundly customisable. it processes functionalities, which allow users to adjust the look and feel to their taste. Users may also download third party home applications from the Play Store. (Android Open Source Project 2015f.)

2.2 Android Applications

Android native applications (“apps”) are Android software applications that extends the functionality of an Android device. Android applications typically consists of Activities. An Activity is a single thing an Android user can do at a given time; it is the focus of the screen. Virtually all Activities are interactive. Thus, the Activity class is charged with creating a window for an Android developer to place his/her User Interface. (Android Open Source Project 2015i.)

Android native applications are written in the Java programming language using the Android software development kit. These applications work only on Android devices or Android emulators; Android Application is an exception to the "write once, run anywhere" claim of the Java platform. The Android software development kit (SDK) is a collection of software tools used in the development of an Android application. The Android SDK comprises of an emulator based on QEMU, required software libraries, relevant documentation for the Android application program interfaces (APIs), debugger, sample source codes, and tutorials for the Android OS. (Janssen 2015a.)

The official Android SDK is Android Studio, which is based on IntelliJ IDEA, other development tools are available for Android software development. These tools include, Native Development Kit, Google App Inventor and various cross platform mobile web application frameworks such as Masons, PhoneGap and Sencha. Native Development Kit (NDK) is a set of development tools used to write the critical parts of an Android application in C or C++ programming language. Android advises Android application

developers to use NDK only if it is essential to the app being developed— not because the developers is more comfortable writing programs in C or C++ Programming Language (Janssen 2015b; Android Open Source Project 2015g). Google App Inventor was created to enable people with no programming experience to develop an application for Android devices. (Hardesty 2010.)

The Google Play Store contains a fast growing number of third-party applications. Android applications can be downloaded from Google Play Store (the official Android store) or third-party stores (such as Galaxy App, Amazon Appstore, GetJar, F-Droid and SlideMe) using the store's application program that allows users to install, update, and remove applications from their devices. An Android user may also download and install Android applications from other sources such as websites, but such a user will have to enable installation of applications from unknown sources from the Android device's security settings. Android application package, APK (. apk) is the installable file format of an Android application. As of February 2015, the number of applications available on Play Store was around 1.4 million. (Soomro 2014; The Statistics Portal et al. 2015.)

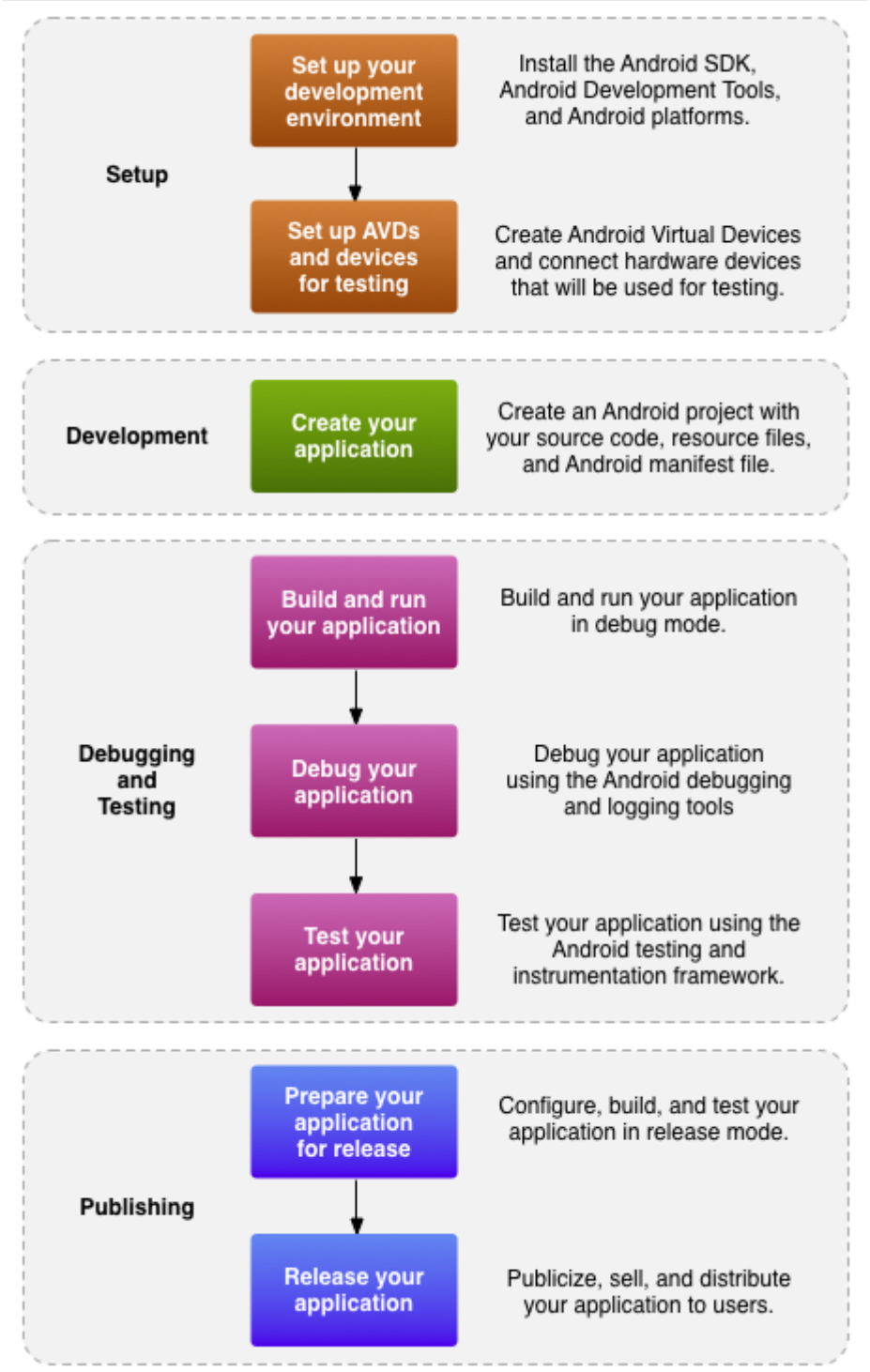
2.3 Android Software Development

Android software development is the process of creating or writing Application software for the Android operating systems. Android application is typically developed using the Android Software Development Kit (SDK). The Android Software Development Kit (SDK) enables a developer to separate the programming logic from the presentation layout (Graphical User Interface). The control codes are written in the Java programming language, while the presentation layout is coded in Extensible Markup Language (Leiva 2014). Android application development is similar to other software development. It consists of several phases. The basic steps involved in Android software development; include, Environment Setup, Project Setup and Development, Building, Debugging and testing and Publishing.

Environment Setup phase involves setting up the software development environment to be used during the application development. It also involves preparing basic testing tools

to test the progress of the application being developed. Such tools might be an Android Virtual Device (AVD) or real Android devices. Project Setup and Development is the stage where a developer sets-up and develops the Android application project and application modules. The application modules hold all the application's source codes and resource files (for example XML files, and pictures). As the application development progresses, there will come a phase where the application developer would want to see what the application actually looks like and also check if the implemented functionalities are working properly. (Android Open Source Project 2015h.)

During the Building, Debugging and Testing phase the Android project is first built into a debuggable .apk package(s), which can install and run on and Android emulator or an Android-powered device. After building and installing the Application on an Android-powered device, the developer can check for and debug errors manually or using the SDK debugging and logging tools. After debugging the application, a developer should test the efficiency of the developed application and various other aspects using the Android SDK testing tools. The publishing phase involves releasing and distributing the finished application to the users. Graph 2 shows the Android workflow in chronological order. (Android Open Source Project 2015h.)



GRAPH 2. The Android App Workflow (Android Open Source Project2015h).

3 CHRONOMETERX

The ChronometerX is an Android application. it consists of five different utilities: Alarm, Calendar, Stopwatch, Timer and To-do list utility. The aim of ChronometerX is to compile five basic and essential applications into one application package and give the user a summary of the upcoming events in the application main activity (screen).

3.1 Requirement gathering and Analysis

Software requirements are the descriptions of what a system or software should do, that is, the services it should provide and the limitations on its operations. These requirements reveal the needs of customers or users for system or software that serves a certain purpose or solves a certain technology problem such as controlling a device, editing text or placing an order. Requirements engineering (RE) is the process of finding out, analyzing, documenting and checking software or system's services and constraints. (Sommerville 2010, 83.)

The general requirement of ChronometerX is that it should contain simple, but vital time saving or time monitoring applications. The requirement for a system or software may be categorized into two related, but distinct parts: User requirements and System requirements. User requirements are statements written in natural language, including diagrams showing the purpose and services a system or software is expected to provide to users and the constraints under which it must operate. (Sommerville 2010, 83.) ChronometerX has two User requirements. The application shall contain an Alarm utility, Calendar utility, Stopwatch utility, Timer utility and To-do list utility and also, the application shall be able to show the summary of upcoming events. Table 1 shows the list of ChronometerX's User requirements.

TABLE 1. ChronometerX's User requirements

	ChronometerX's User requirements
1.	The application shall contain Alarm, Calendar, Stopwatch, Timer and To-do listTo-do list utility.
2.	The application shall be able to show the summary of upcoming events.

System requirements are detailed descriptions of a software system's functions, services, and operating restrictions. The software requirements document or functional specification of software should define precisely the functions to be realized by the system. Sometimes it is a part of the contract between the system developer and the system or software buyer. (Sommerville 2010, 83.)

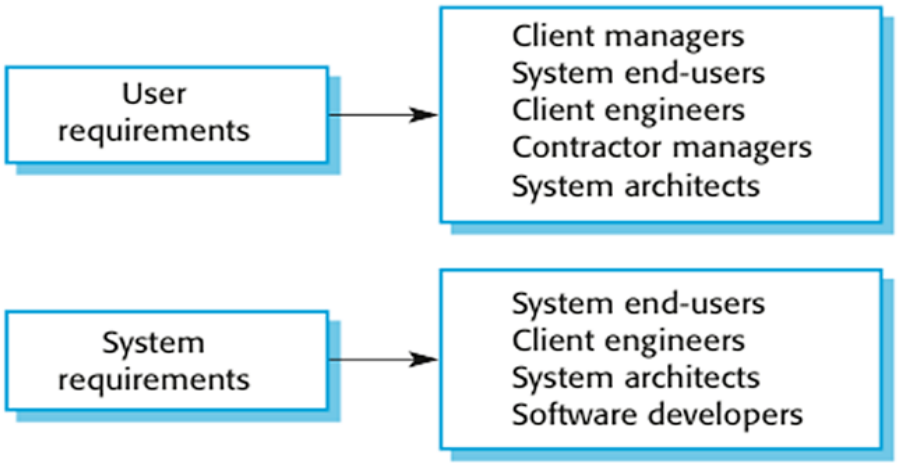
ChronometerX has six System requirements. These requirements are the basis for ChronometerX software function. The Alarm utility of application shall allow its users to set multiple alarms and users shall be able to set an alarm to repeat daily or on weekdays. The user shall be able to add an event to the Calendar application and check current date. The user shall be able to measure range of time using the Stopwatch. This paragraph only contains three of the six ChronometerX's System requirements, see table 3 for the full list. Table 2 shows the list of ChronometerX's System requirements.

TABLE 2. ChronometerX's System requirements

	ChronometerX's System requirements
1.	The Alarm Utility of application shall allow user to set multiple Alarms and users shall be able to set an Alarm to repeat daily or on weekdays.
2.	The user shall be able to add an event to the Calendar application and check current date.
3.	The user shall be able to measure range of time using the Stopwatch.
4.	The Timer shall allow user to input a specific time (hour: minute: second), the time decrease until the time given is up and then the phone starts to ring (a sound start to play).
5.	The To-do list shall allow user input tasks and each task may have subtasks
6.	The application shall show a summary of upcoming events.

3.2 Functional and non-functional requirements

Software System requirements are mainly categorized into functional requirements or non- functional requirements. Functional requirements are descriptions of the services a software system should render. It includes what the software should do, how the software should handle certain inputs, how the software should present it outputs and more importantly how the software should act in certain scenarios. Sometimes functional requirements may include the proscription of the software system. (Sommerville 2010, 84-85.) ChronometerX Software requirements is the same as the functional requirement. Graph 3 illustrates subdivisions of the Functional requirements.



GRAPH 3. Readers of different types of requirements specification (Pearson, 2011).

A non-functional requirement outlines the limits of a software system services and functions. This requirement focuses more on the performance characteristics like, interoperability, maintainability, and quality. These requirements usually concern the totality of the system and not a specific system or software function (Sommerville 2010, 84-85). ChronometerX has three Non functional Requirement. The application should load in less than four seconds and should not crash without giving a crash report. Application should be as light as possible. Application should not occupy more than 10gb of memory. Application shall be easy to maintain. Application GUI shall be separated from

control codes as much as possible. Table 3 explicates the ChronometerX's Nonfunctional requirements.

TABLE 3. ChronometerX's Nonfunctional Requirements

	ChronometerX's Nonfunctional Requirements	
	Requirement	**Explanation**
1.	Response times	Application must be highly responsive, ChronometerX is a mobile application, user experience really matters. The application should load in less than 4 seconds and should not crash without giving a crash report (feedback).
2.	Storage	Application should be as light as possible; application should not occupy more than 10gb of memory
3.	Maintainability	The application shall be easy to maintain, GUI shall be separated from control codes as much as possible.

3.3 Requirements Specification

Requirements specification is the process of documenting user and System requirements in a requirement document. For easy system development and good communication between system developer and stakeholders, the Requirements specification should be well defined, explicit, comprehensive, and coherent. This is difficult to achieve in practice because stakeholders translate the requirements in various ways and sometimes there are inherent conflicts and discrepancies in the requirements. (Sommerville 2010, 94.)

The User requirements target a system's user. Thus, the User requirements are descriptions of the functional and non- functional requirements in a language a user without detailed technical knowledge can comprehend. In simpler terms, they feature what a user expects of a system. User requirements are normally written in natural language in conjunction with simple visual descriptions such as tables and charts. These visuals should be as simple as possible. (Sommerville 2010, 94.)

The System requirements of a software system are more elaborate description of the functional and non- functional requirements. It is unlike the User requirements, which

contain only simple descriptions of what a user should expect of a system software, the System requirements serve as the basis for the system design. Thus, they contain in-depth descriptions of the internal and external behaviour of the system, including the architectural details to help the developer get a comprehensive picture of the system as a whole. (Sommerville 2010, 94.)

3.4 Software specification

Software specification or requirements engineering is the procedure of understanding and documenting the breakdown of the functions and services of a system, including the system's operational and developmental limits and restrictions. Requirements engineering is a peculiar and critical phase in a software process. As a result, errors at this point are critical and must be firmly eschewed, because these lapses may disrupt the system design and implementation. (Sommerville 2010, 37-38.)

The requirements engineering practice is intended to yield a concurred requirements document that describes a system fulfilling stakeholder requirements. These requirements are usually produced at two levels of detail. End-users and customers need a high-level statement of the requirements, while system developers need a more detailed system specification. There are four fundamental exercises in the requirements engineering procedure: Feasibility study, Requirements elicitation and analysis, Requirements specification and Requirements validation. Sommerville 2010, 37-38.)

The feasibility study is an assessment and investigation of the capability of a proposed software system. This evaluation is done based on whether the perceived client needs may be fulfilled utilizing current technological tools (software and hardware technology). The study considers whether the proposed project will be visible from a business perspective and can be realized within existing budgetary limitations. A feasibility study ought to be generally cheap and considerably fast. The result should bolster the strategy of decision-making. (Sommerville 2010, 37-38.)

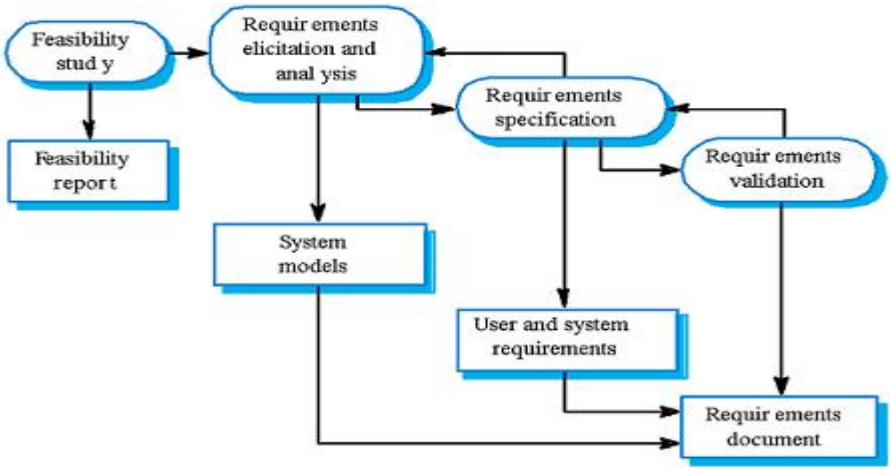
During the course of this thesis a survey was conducted in the Android Play Store and

Apple APP Store and the result shows that although there is mobile application such as Apple clock application and Android clock applications that contains some of the applications (Alarm, Calendar, Stopwatch, Timer and To-do list) to be built in to ChronometerX, but none has a screen to show the summary of events in the same application. ChronometerX will satisfy the user's need to see the summary of current event and upcoming events.

Requirements elicitation and analysis, also known as requirement gathering involves close perception of existing systems, discussing with potential users and procurers and task analysis to derive the proposed system's requirements. One or more system models and prototypes might be developed in the process, in order to comprehend the proposed system better. Requirement gathering might be carried out by the means of an interview, workshops, questionnaires, user observation, brainstorming and having users test software prototypes. (Sommerville 2010, 37-38.)

Requirements specification is the process of translating and documenting the information collected during requirements elicitation and analysis. The requirement specification consists of two kinds of requirements: User requirements and System requirements. User requirements are abstract descriptions of the functional and non- functional requirements to suit the customer and the end-user of the system. System requirements are a comprehensive description of the functional and non- functional requirements. They are translated to help the system developer. Thus, System requirements serve as the basis for the system design. (Sommerville 2010, 37-38.)

Requirements validation involves checking whether or not the documented requirements are realistic and fulfil predefined quality and quantity criteria. Discussing with pertinent stakeholders, and involving requisite sources such as laws and standards is important. Errors in the requirements are ineluctably discovered during the phase and must be rectified. Requirements validation should make sure the System requirements are complete, traceable, veridical, comprehensive, consistent, up-to-date and confirmed by the stakeholders. (Sommerville 2010, 37-38.) Graph 4 describes the Requirements engineering process.



GRAPH 4. The requirements engineering process (Sommerville 2010).

3.5 Software Modelling

System modelling is the methodology of creating abstract models of a system and every model introduces a disparate perspective or viewpoint of that system. System modeling generally involves representing the system with graphical notations, which is virtually always based on the notations in the Unified Modelling Language (UML). (Sommerville 2010, 119.)

Different models are developed to represent the system from different perspectives. An external perspective involves modelling the context or environment of the system. An interaction perspective is concerned with modelling the interactions between a system and its environment or interactions between the components of a system. A structural perspective entails modelling the organization of a system or the structure of the data that is processed by the system, while a behavioural perspective involves modeling the dynamic behaviour of the system and how it responds to events. (Sommerville 2010, 119.)

4 INTERACTION MODELS

All systems involve interaction. In a software system context, these interactions might be user interaction, which has to do with user inputs and outputs, or interaction between different components of the system and even interaction between different systems. Modelling user interaction uncovers and describes the User requirements. Modelling the system component interaction helps to comprehend and validate System requirements based on performance and dependability. Modelling system-to-system interaction sheds light on the communication problems that may arise between the proposed system and system it may have to communicate with. (Sommerville 2010, 124.)

4.1 ChronometerX's Use case documentation

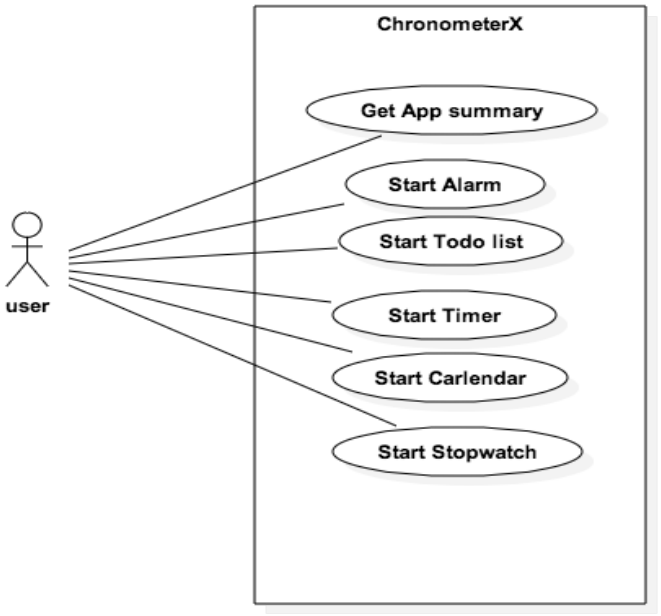
This section covers the ChronometerX use case document and comprises of use case diagrams and use case tables. The use case is a UML diagram; it is very effective for modelling a software system. It bridges the intellectual gap between the software system's developer, stakeholder and the user. It shows a comprehensive illustration of a system or part of a system in one diagram. A use case diagram identifies all the actors interacting with a system and their roles including external systems communicating with the proposed system. A use case table highlights important characteristics of a use case it shows the name of the use case, the actors interacting with the use case, use case description, preconditions, post-conditions and use case exceptions. Use case preconditions are conditions which must be met before a use case can be fired while post-condition indicates what will happen after a use case is fired. Use case exceptions are ways a use case may fail to complete. When a use case exception occurs the benefit of the use case is not delivered to the user. (UML diagrams 2015.)

A UML use case diagram typically comprises of an actor, subject (system) use cases and interaction lines (arrows). Each use case signifies a unit of useful functionality that a subject (system) provides to actors. An association (interaction lines) between an actor and a use case implies that the actor and the use case interact or communicate with each

other. In UML use case diagrams actors are behaviour classifiers. They specify a role played by an external entity that interacts with a system. For example, a human user using a mobile application. In this example the human is the external entity that is interacts with the system (mobile application). The actors are usually represented with a stick man in UML use case diagrams. Use cases are presented in the form of an ellipse enclosing the use case's name and the subject is represented by a rectangle, which engulfs the use cases. (UML diagrams 2015.)

GRAPH 5 is a use case diagram. It shows an interaction between the user and ChronometerX's Main screen and its fundamental Activities. The Start Alarm, Start To-do list, Start Timer, Start Calendar and Start Stopwatch. The tables describe each of the fundamental activities in details. Table 4 explains the Get App summary use case

The table consists of 6 rows. The first row contains the use case name (Get App summary). The second row shows the actor (application user) interacting with the Get App summary Use case. Row3 features the Get App summary use case precondition. To fire the Get App summary use case, a user must have ChronometerX installed on his or her Android phone. Row4 contains a brief description of the Get App summary use case. To fire the Get App summary use case a user clicks on the ChronometerX's application Icon either from the Android system environment or from ChronometerX's application environment. Row 5 holds the GetApp summary use case Post-condition. When the Get App Summary use case is fired, the summary of upcoming events is shown. Row 6 highlights the use case exception. It indicates that the Get App summary use case has no exception. Tables 5, 6, 7, 8 and 9 give details of Start Alarm use case, Start To-do list use case, Start Timer use case, Start Calendar use case and Start Stopwatch use case respectively.



GRAPH 5. ChronometerX fundamental Activities Use case diagram.

TABLE 4. Get App Summary Use case.

Name of the Use case: Get App Summary
Actors: User
Preconditions **ChronometerX must be installed on an Android devices and must be open.**
Description of the Use case **User clicks on the ChronometerX's application Icon either from the Android system environment or from the ChronometerX's application environment.**
Post-conditions **Summary of upcoming events is shown.**
Exceptions descriptions **None.**

TABLE 5. Start Alarm Use case.

Name of the Use case: Start Alarm
Actors: Users
Preconditions **ChronometerX must be open.**
Description of the Use case **User clicks on the Alarm action Icon on the Get App Summary screen.**
Post-conditions **Alarm Application is open.**
Exceptions descriptions **None.**

TABLE 6. Start To-do List Use case.

Name of the Use case: Start To-do list
Actors: Users
Preconditions **ChronometerX must be open**
Description of the Use case **User clicks on the To-do list Action Icon on the Get App Summary screen.**
Post-conditions **To-do list application is open.**
Exceptions descriptions **None.**

TABLE 7. Start Timer Use case.

Name of the Use case: Start Timer
Actors: Users
Preconditions **ChronometerX must be open**
Description of the Use case **User clicks on the Timer Action Icon on the Get App Summary screen.**
Post-conditions **Timer Application is open.**
Exceptions descriptions **None.**

TABLE 8. Start Calendar Use case.

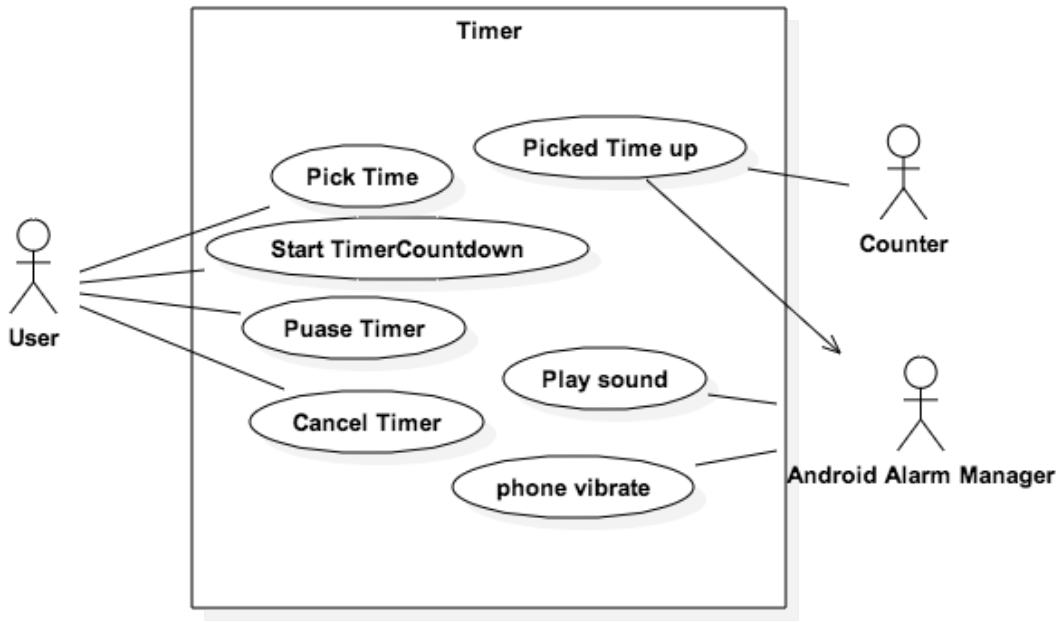
Name of the Use case: Start Calendar
Actors: Users
Preconditions **ChronometerX must be open**
Description of the Use case **User clicks on the Calendar Action Icon on the Get App Summary screen.**
Post-conditions **Calendar Application is open.**
Exceptions descriptions **None.**

TABLE 9. Start Stopwatch Use case.

Name of the Use case: Start Stopwatch
Actors: Users
Preconditions **ChronometerX must be open**
Description of the Use case **User clicks on the Stopwatch Action Icon on the Get App Summary screen.**
Post-conditions **Stopwatch Application is open.**
Exceptions descriptions **None.**

4.2 Timer Use case documentation

The Timer use case documentation in Graph 6 describes the ChronometerX's Timer application. Graph 6 shows a pictorial illustration of the ChronometerX's Timer application. It features actors and use cases, and also shows the relationship between the actors and the Use cases. Tables 7, 8, 9, 10, 11, 12 and 13 describe each of the Timer application use cases in details. Table 7 explains the Pick Time use case and Tables 8, 9, 10, 11, 12 and 13 give details of use cases: StartTimerCountdown, PauseTimer, ResetTimer, PickedTimeUp, PlaySound and PhoneVibrate respectively.



GRAPH 6. ChronometerX Timer Use case diagram

The Timer use case features seven use cases and three actors. Actor one is the application user who interacts with four of the seven use cases: Pick Time, StartTimerCoundown, Pause Timer and Cancel Timer. These four use cases represent onscreen objects which the use interact with whist using the Timer utility. The Pick Time use case enables the user to set Timer's time; the Start Timercoundown enables the user to start the Timer. The user pauses the Timer by firing the Pause Timer use case and the Timer is cancelled when Cancle Timer use case is fired.

The four use cases the application user is interacting with are all front end use cases. That is, the user interacts with them directly. The last three use cases: PickedTimeUp, PlaySound and Phone Vibrate are back end use cases. These use cases operate in the background. The actors Counter and AndroidAlarm Manager are the actors interacting with the back end use cases. After a user starts a Timer and the Timer's countdown is done, the Counter fires the Picked Time up use case and the use case sends a message to the AndroidAlarm Manager. The AndroidAlarm Manager in turn fires the play sound and phone vibrate use case. The play sound use case and phone vibrate use case are intended to play sound and vibrate the phone respectively.

TABLE 10. Pick Time Use case.

Name of the Use case: Pick Time
Actors: Users
Preconditions **The Timer application must be open.**
Description of the Use case **The user sets the time on the Timer picker.**
Post-conditions **The user has picked a time.**
Exceptions descriptions **None.**

TABLE 11. StartTimerCountdown Use case.

Name of the Use case: StartTimerCountdown
Actors: Users
Preconditions **The Timer application must be open and user has picked a time.**
Description of the Use case **User clicks on the start button to start the Timer and the picker disappear and a Text view is shown counting down from the chosen time.**
Post-conditions **The Timer starts to count down from the picked time.**
Exceptions descriptions **None.**

TABLE 12. PauseTimer Use case.

Name of the Use case: PauseTimer
Actors: Users
Preconditions **The Timer application must be open and the Timer must still be counting down.**
Description of the Use case **User clicks on the pause button to pause Timer count down.**
Post-conditions **Timer count down is paused.**
Exceptions descriptions **None.**

TABLE 13. ResetTimer Use case.

Name of the Use case: ResetTimer
Actors: Users
Preconditions **The Timer application must be open, and the Timer must still be counting or paused**
Description of the Use case **User clicks on the Reset button to reset Timer**
Post-conditions **Time picker is shown.**
Exceptions descriptions **None.**

TABLE 14. PickedTimeUp Use case.

Name of the Use case: PickedTimeUp
Actors: Counter
Preconditions **The Timer application must be open, and countdown must be up (have ended).**
Description of the Use case **When the countdown is up the count fires the PickedTimeUp to notify the AndroidAlarm manager.**
Post-conditions **The AndroidAlarm manager has received the PickedTimeUp notification.**
Exceptions descriptions **None.**

TABLE 15. PlaySound Use case.

Name of the Use case: PlaySound
Actors: AndroidAlarm Manager
Preconditions **The Timer application must be open, countdown must be up, and the counter has sent notification.**
Description of the Use case **When the AndroidAlarm manager received the notification it fires the PlaySound use case.**
Post-conditions **Phone is ringing.**
Exceptions descriptions **None.**

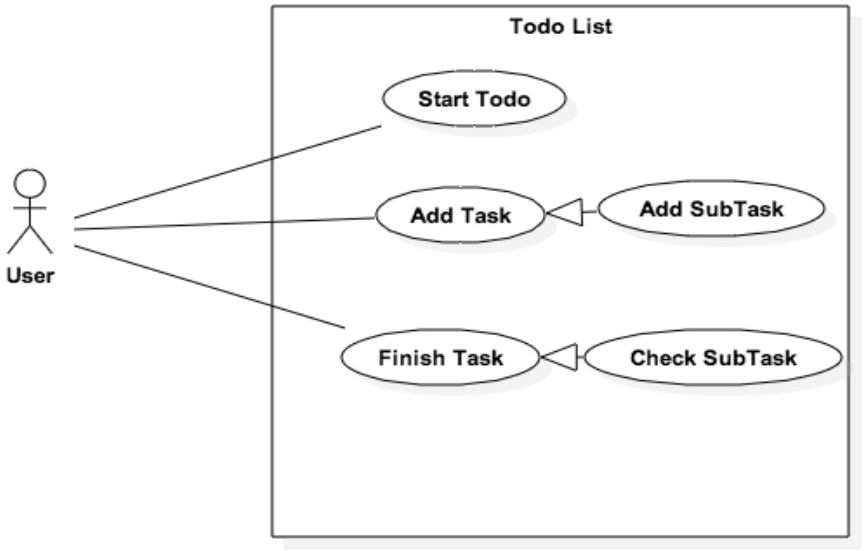
TABLE 16. PhoneVibrate Use case.

Name of the Use case: PhoneVibrate
Actors: Users
Preconditions **The Timer application must be open, countdown must be up, and the counter has sent notification to the AndroidAlarm manager.**
Description of the Use case **When the AndroidAlarm manager received the notification it fires the PhoneVibrate Use case.**
Post-conditions **Phone Vibrates.**
Exceptions descriptions **None.**

4.3 To-do list Use case documentation

This section consists of the breakdown and the documentation of the functions of the ChronometerX's To-do list application. Graph 7 describes the ChronometerX's To-do list application, and the tables give detailed explanations of each use case in the To-do list application use case diagram. Tables 17, 18, 19, 20, and 21 give details of use cases: StartTodo, AddTask, Add SubTask, FinishTask and CheckSubTask respectively.

The To-do list use case diagram includes an Actor and five use cases. The Actor in this use case diagram is ChronometerX's user. The StartTodo use case opens the To-do list application when fired. The user adds a task and task's subtask to the To-do list by firing the Add Task use case and the Add subtask use case respectively. When the use is done with a subtask he/she will fire the check subtask use case and once the main task is done the user fires the Finish task use case to indicate to the application that the task is done.



GRAPH 7. ChronometerX To-do List Use case diagram.

TABLE 17. StartTodo Use case.

Name of the Use case: StartTodo
Actors: Users.
Preconditions **ChronometerX must be open.**
Description of the Use case **User clicks on the To-do list Action Icon in Launch Activity.**
Post-conditions **To-do list Application is open.**
Exceptions descriptions **None.**

TABLE 18. AddTask Use case.

Name of the Use case: AddTask
Actors: Users
Preconditions **To-do list Application must be open, the name of the task must be entered**
Description of the Use case **User clicks on the AddTask button**
Post-conditions **The task is added and displayed.**
Exceptions descriptions **None.**

TABLE 19. Add SubTask Use case.

Name of the Use case: Add SubTask
Actors: Users
Preconditions **To-do list Application must be open, the user must have added a Task and the name of the sub-task must be entered.**
Description of the Use case **User clicks on the Add subTask button to add a sub-task to a Task**
Post-conditions **The subtask is added to the task.**
Exceptions descriptions **None.**

TABLE 20. Finish Task Use case.

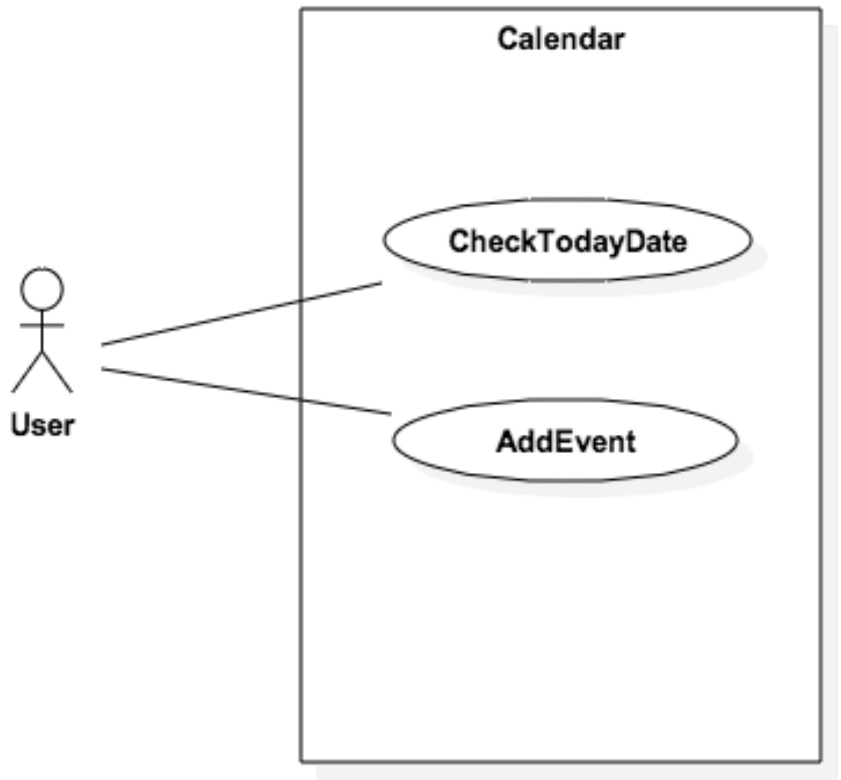
Name of the Use case: Finish Task
Actors: Users
Preconditions **To-do list Application must be open and user clicks the Finish Task button to finish a given task.**
Description of the Use case **This Use case is used to finish a given task and delete it from the phone memory.**
Post-conditions **Finished task has been deleted from memory.**
Exceptions descriptions **None.**

TABLE 21. CheckSubtask Use case.

Name of the Use case: CheckSubtask
Actors: Users
Preconditions **A sub-task has been added.**
Description of the Use case **User clicks on the sub-task check box to check the sub-task**
Post-conditions **Sub-task check box is checked.**
Exceptions descriptions **None.**

4.4 Calendar Use case documentation

The Calendar use case documentation below describes the ChronometerX's Calendar Application based on Requirement gathered during Requirement gathering and analysis phase of the ChronometerX's Application life cycle. Graph 6 shows a graphic illustration of the ChronometerX's Calendar application. it features an actor and two use cases, and also shows the relationship between the actors and the use cases. Table 22 and table 23 include detailed descriptions of the Calendar application's use case. Table 22 and table 23 explain the CheckTodayDate use case and the AddEvent use case respectively. The application user is the actor interacting with the Calendar use case. The user fires the CheckTodayDate use case to check the current date and adds an event to the Calendar by firing the Add Event use case.



GRAPH 8. ChronometerX AlarmUse case diagram

TABLE 22. CheckTodayDate Use case.

Name of the Use case: CheckTodayDate
Actors: Users
Preconditions **Calendar Application is opened.**
Description of the Use case **This use case is use to show the current date.**
Post-conditions **The current date is shown.**
Exceptions descriptions **None.**

TABLE 23. Add Event Use case.

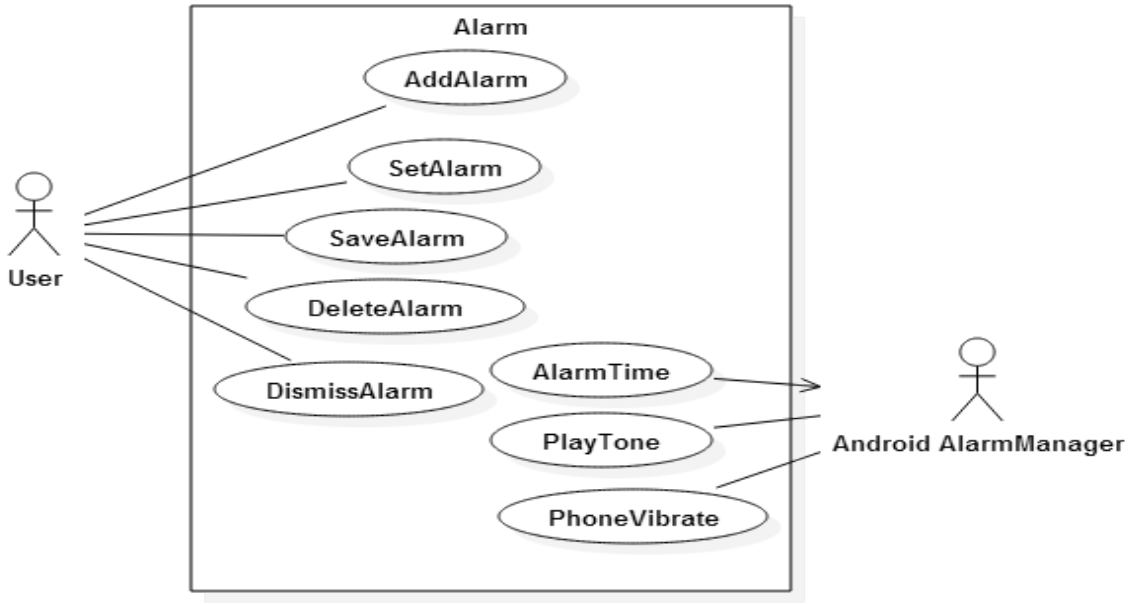
Name of the Use case: Add Event
Actors: Users
Preconditions **Calendar Application is opened, the name and location of the event is given.**
Description of the Use case **The user clicks on the sub-task check box to check the sub-task**
Post-conditions **The event is added to the Calendar.**
Exceptions descriptions **None.**

4.5 Alarm Use case documentation

This segment documents and explains ChronometerX's Alarm application requirements. Graph 9 describes the ChronometerX's Alarm application, and Tables 24, 25, 26, 27, 28, 29, 30, and 31 give detailed explanations of each use case in the Alarm application use case diagram. Tables 24, 25, 26, 27, 28, 29, 30, and 31 explain the following use cases: Add Alarm, Set Alarm, SaveAlarm, DeleteAlarm, DismissAlarm, AlarmTime, PlayTone and PhoneVibrate respectively.

Graph 9 consists of two actors and 8 use cases. The actor on the left hand side of the use case is the application user, the application user interacts directly with 5 use cases: Add Alarm, SetAlarm, SaveAlarm, DeleteAlarm, DismissAlarm, AlarmTime. The user fires the AddAlarm use case to add an Alarm. The user sets the Alarm details by firing the SetAlarm use case. After the user have entered the Alarm details he or she can save the Alarm by firing the SaveAlarm use case. An Alarm is deleted by firing the DeleteAlarm use case. When an Alarm rings the user can dismiss the Alarm by firing the DismissAlarm use case.

The second actor and the actor on the right hand side of graph 9 is the AndroidAlarm Manager. The AndroidAlarm Manager is responsible for firing the Playtone use case and PhoneVibrate use case used by ChronometerX to play a tone and vibrate the phone. When an Alarm time is reached AlarmTime (the counter) notifies the AndroidAlarm Manager, then the AndroidAlarm Manager fires the Playtone and PhoneVibrate use cases.



GRAPH 9. ChronometerX Alarm Use case diagram.

TABLE 24. AddAlarm Use case.

Name of the Use case: AddAlarm
Actors: Users
Preconditions **Alarm Application is opened.**
Description of the Use case **User clicks on the Add Alarm action button to add an Alarm**
Post-conditions **The setAlarm screen opens and the user is able to set Alarm values.**
Exceptions descriptions **None.**

TABLE 25. SetAlarm Use case.

Name of the Use case: SetAlarm
Actors: Users
Preconditions **Alarm Application is open, the user has fired the AddAlarm Use case.**
Description of the Use case **This enables the user to set Alarm values like Alarm time and Alarm name.**
Post-conditions **The Alarm is ready to be saved.**
Exceptions descriptions **None.**

TABLE 26**.** SaveAlarm Use case.

Name of the Use case: SaveAlarm
Actors: Users
Preconditions **Alarm Application is opened; user has fired the AddAlarm Use case, and use has setAlarm values.**
Description of the Use case **This enables the user to save Alarm information to the database.**
Post-conditions **The Alarm is saved to the database, and Alarm is shown in the Alarm application main screen.**
Exceptions descriptions **None.**

TABLE 27. DeleteAlarm Use case.

Name of the Use case: DeleteAlarm
Actors: Users
Preconditions **Alarm Application is opened, there is an Alarm to be deleted in the Alarm main screen.**
Description of the Use case **This enables user to delete a given Alarm.**
Post-conditions **The Alarm is deleted from the database and the main screen.**
Exceptions descriptions **None.**

TABLE 28. AlarmTime Use case.

Name of the Use case: AlarmTime
Actors: Users
Preconditions **Alarm has been Added.**
Description of the Use case **This use case alerts the AndroidAlarm manager when the Alarm time is reached.**
Post-conditions **The AndroidAlarm manger is aware that the Alarm time has been reached.**
Exceptions descriptions **None.**

TABLE 29. PlayTone Use case.

Name of the Use case: PlayTone
Actors: AndroidAlarm Manager
Preconditions **Alarm Time is reached.**
Description of the Use case **A tone is played when this Use case is fired**
Post-conditions **Timer Application is open.**
Exceptions descriptions **None.**

TABLE 30. PhoneVibrate Use case.

Name of the Use case: PhoneVibrate
Actors: AndroidAlarm manager
Preconditions **Alarm Time is reached.**
Description of the Use case **When the Alarm time is reached the AndroidAlarm manager fires the PhoneVibrate Use case.**
Post-conditions **The phone vibrates.**
Exceptions descriptions **None.**

TABLE 31. Dismiss Alarm Use case.

Name of the Use case: Dismiss Alarm
Actors: Users
Preconditions **Alarm Time is reached, Play Tone or Phone Vibrate is fired.**
Description of the Use case **This helps to dismiss Alarm temporarily. I.e. Stop playing tone and stop the phone from vibrating.**
Post-conditions **The Alarm is dismissed temporarily. I.e. phone stop playing and phone stop vibrating.**
Exceptions descriptions **None.**

5. DESIGN AND IMPLEMENTATION

The software design and implementation stage is the phase in the life cycle of a software system where the proposed functional software system is developed. For simple systems, software design and implementation is synonymous to software engineering. All other activities are merged with the software design and implementation. However, for large systems, software design and implementation is only one part of the process (requirements engineering, verification and validation) involved in software engineering. (Sommerville 2010, 174.)

Software design and implementation activities are always interwoven. Software design is an innovative activity by which software service or components are identified and conceptualized including their relationships and connections based on a stakeholder's requirements. Implementation is a systematic and structural approach to transforming the designing concept into a working program (Sommerville 2010, 174). ChronometerX contains 6 basic activities. The first activity (summary screen) is used to show a summary of the other activities. It also includes Action buttons used to launch the other five activities. The summary of the Calendaractivity, To-do list activity, Stopwatch activity, Timer activity and the Alarm activity is shown on the summary screen. Table 32 explains the 6 basic activities contained in ChronometerX.

Table 32. ChronometerX's 6 Basic Activities.

	ChronometerX 6 Basic Activities	
	Activity	**Explanation**
1.	Main activity	The main activity or summary screen is the First screen the user sees when the application is launched. It contains the action icon (button) to start the other application. It also shows the current date and time.
2.	Calendar	The Calendar application is launched using the Calendar action button in the main activity. It is a simple, but useful application that shows the Calendar months, allows user to add an event to the Calendar by clicking on the "addevent" action button and also highlights current date when the user clicks the today action button.
3.	To-do-List	The to-do activity is the most complex of all the utilities. It consists of and custom Expandable list view which displays to-do tasks, an action button to add a task.
4.	Stopwatch	This is a one-activity application that counts upward from zero. There are three buttons in the application. Start button that starts the counting, the Stop button that stops the counting and the Reset button that resets the values of the counter and a chronometer view that displays the count.
5.	Timer	The Timer activity lets a user set a time, counts down from the given time in seconds and then plays a sound when the time elapses.
6.	Alarm	The Alarm application is a simple application, which allows the user to set multiple Alarms.

5.1 Main activity or summary screen

The Main Activity is ChronometerX's launch Activity. It shows the summaries of upcoming events. For example, the next Alarm and the next Calendar event. The Main Activity consists of five action buttons that is used to launch the applications within ChronometerX. The Main Activity is launched when the application is opened, because it is designated as the launch activity in the application manifest file, it is also the parent activity of the other five activities. This makes navigation easy between the Main Activity and the other five activities. The code in Graph 10 below shows the designation of the Main Activity as the launcher activity in the application manifest file.

```
<!-- The main/home activity (it has no parent activity) -->
<activity
    android:name=".MainActivity"
    android:label="@string/app_name"
    android:uiOptions="splitActionBarWhenNarrow"
    android:screenOrientation="portrait">
    <meta-data
        android:name="android.support.UI_OPTIONS"
        android:value="splitActionBarWhenNarrow" />

    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

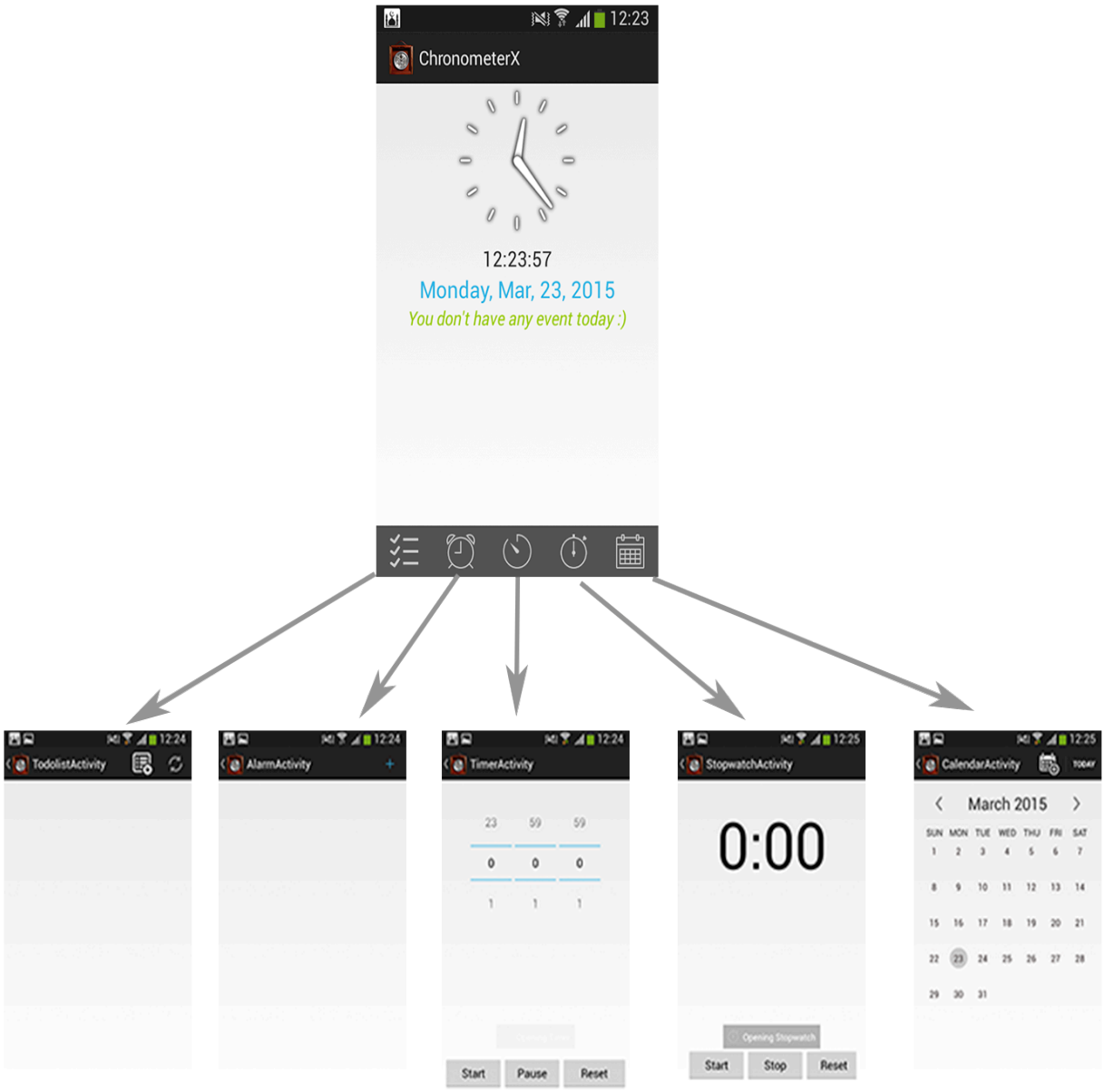
GRAPH 10. The Main Activity Portion of the ChronometerX's Manifest file.

The five applications within the ChronometerX are launched using application intent. When the user clicks the action button assigned to the application in the Main Activity the application opens. Graph 11 shows the switch case (Java source code) used to open each of the applications. Graph 12 shows the relationship between ChronometerX's Main Activity and the main screen of the five utilities contained in ChronometerX. This relationship is established by a pointing an arrow from a utility's launch Action button to the utility's main screen. For example, an arrow points from the Alarm Action button to the Alarm main screen.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();

    switch (id) {
        case R.id.action_todolist:
            startTodolist();
            return true;
        case R.id.action_stopwatch:
            startStopwatch();
            return true;
        case R.id. action_alarm :
            startAlarm();
              return true;
        case R.id.action_timer:
            startTimer();
              return true;
        case R.id.action_calendar:
            startCalendar();
              return true;
        default:
            return super.onOptionsItemSelected(item);
}
}
```

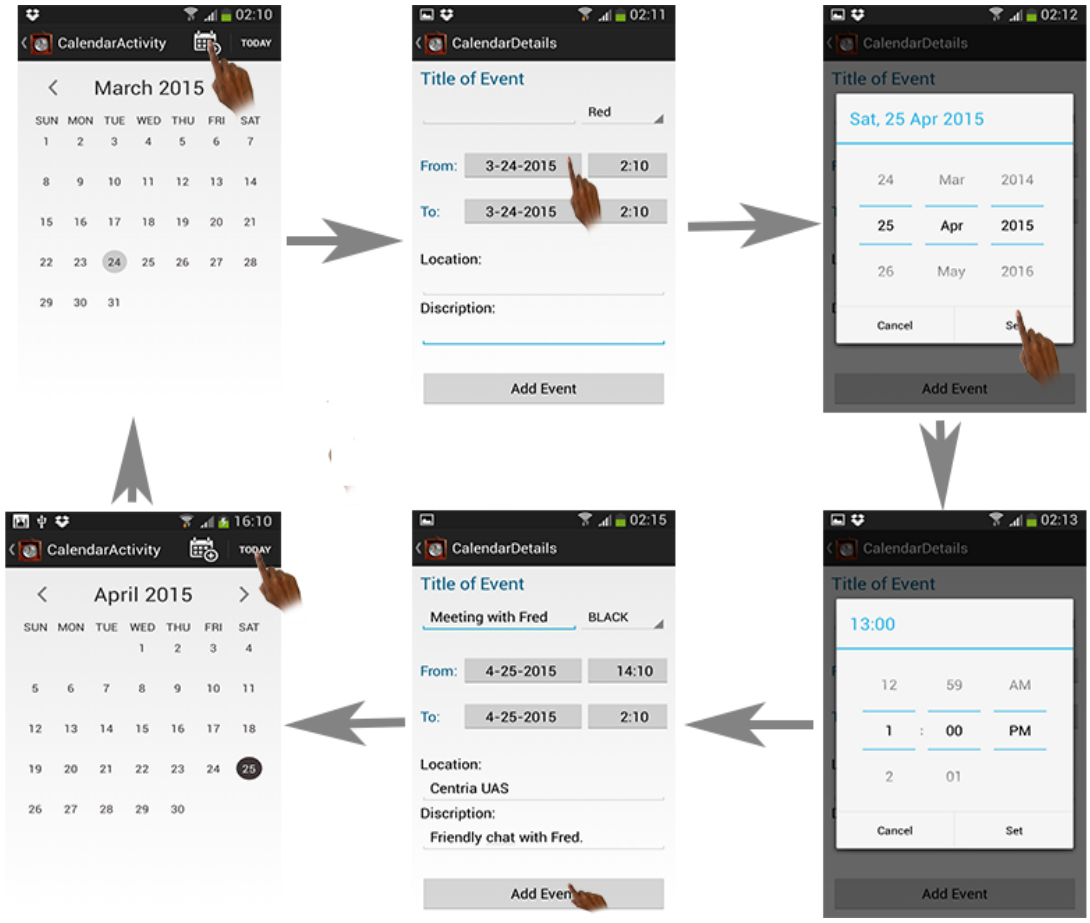
Graph 11. Source code (Switch case) used to open each of the applications in ChronometerX.



GRAPH 12. Views of ChronometerX fundamental Activities.

5.2 Calendar

The Calendar Application consists of two activities. The first activity shows the Calendar month, and features two Action buttons and the Application icon. The Today action button show the current date, while the AddCalendarEvent button launches the second activity as shown in Graph 13 below. The Second activity is used to collect the information of an event to be added to the Calendar application. Graph 13 demonstrates how to add an event to the Calendar Application. First, the user clicks on the AddCalendarEvent button to launch the Calendar Details activity, the user fills the event details and saves it by clicking on the Add event button as shown in the Graph 13.

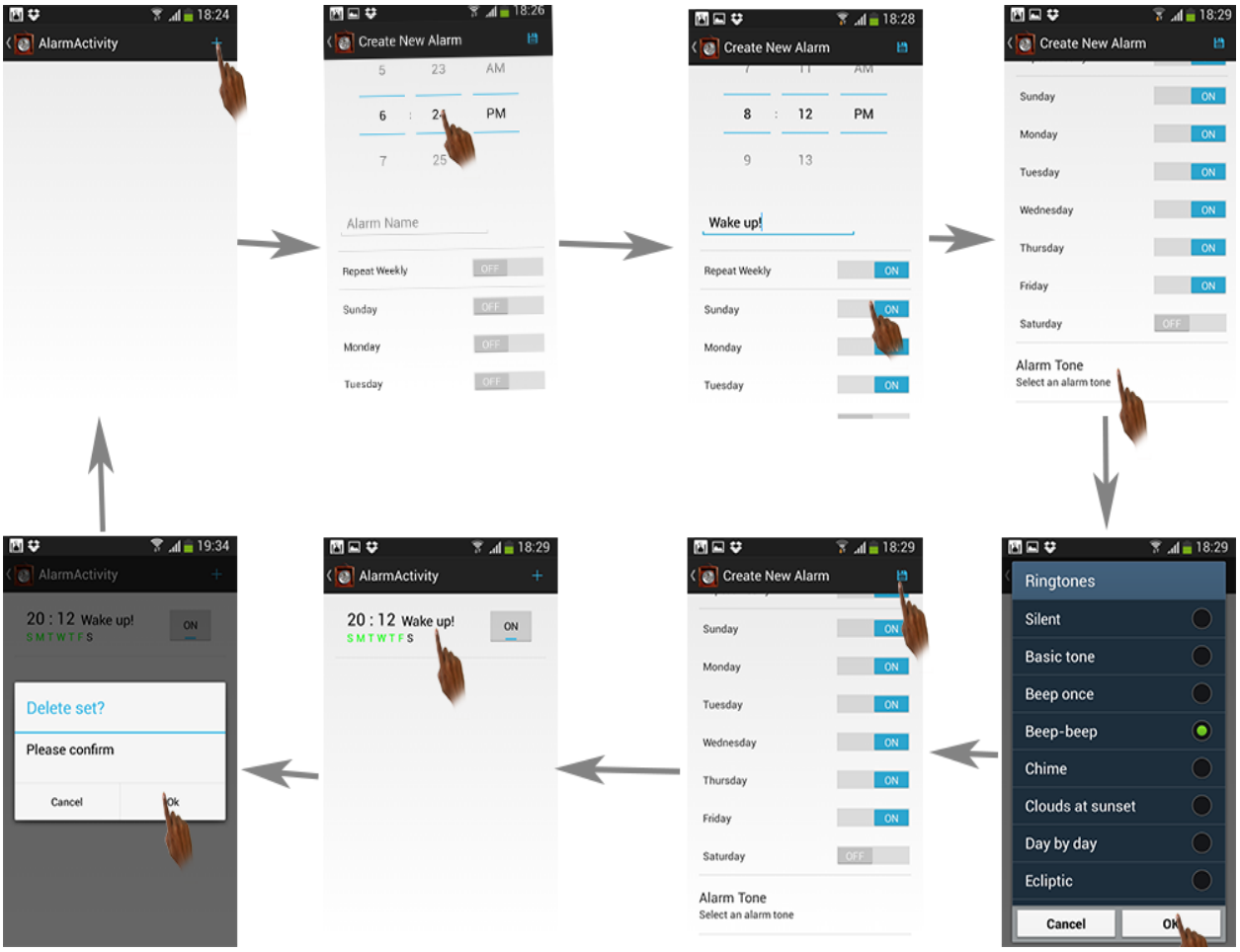


GRAPH 13. Calendar Application.

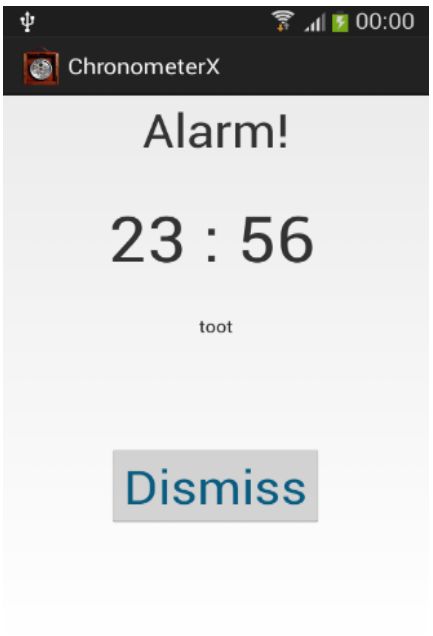
5.3 Alarm

The Alarm Application consists of three activities. The first activity shows the summary of available Alarm(s), and features an Action button to add Alarm. When the Add Alarm action button is clicked, the second activity is launched. The second activity consists of Android views used to collect alarm information and add a ringtone to the Alarm. After the user has set the Alarm information the user has to click on the saveAlarm action button in the Activity to save the Alarm to the database.

Graph 14 shows the process involved in setting up an alarm in Chronometer Alarm application. The user clicks on the AddAlarm button at the top right corner of the screen. After the click, the user is presented with the alarm detail screen to set the alarm details and Alarm ring tone. The user saves the Alarm by clicking on the save Action button at the top right corner of the screen. Then, the Alarm will appear in the Alarm application main screen as seen in Graph 14. Graph 15 shows the alarm ringing. The activity shown in Graph 15 appears when the alarm starts to ring. It contains the alarm name, alarm time and a button to dismiss the alarm.



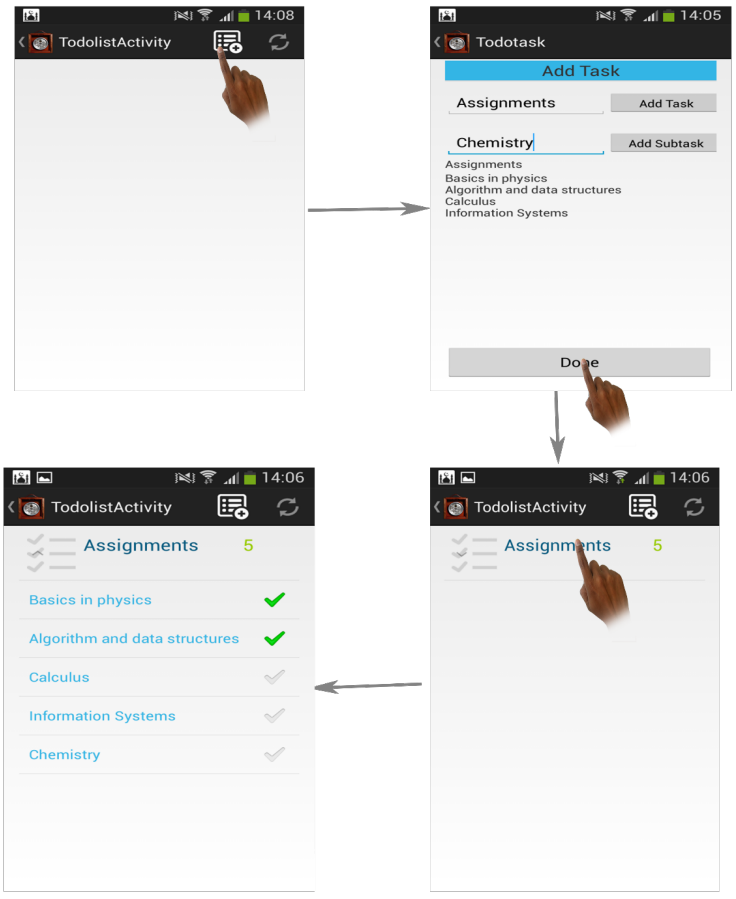
GRAPH 14. Setting up Alarm with the ChronometerX Alarm Application.



GRAPH 15. Alarm Ringing.

5.4 To-do list

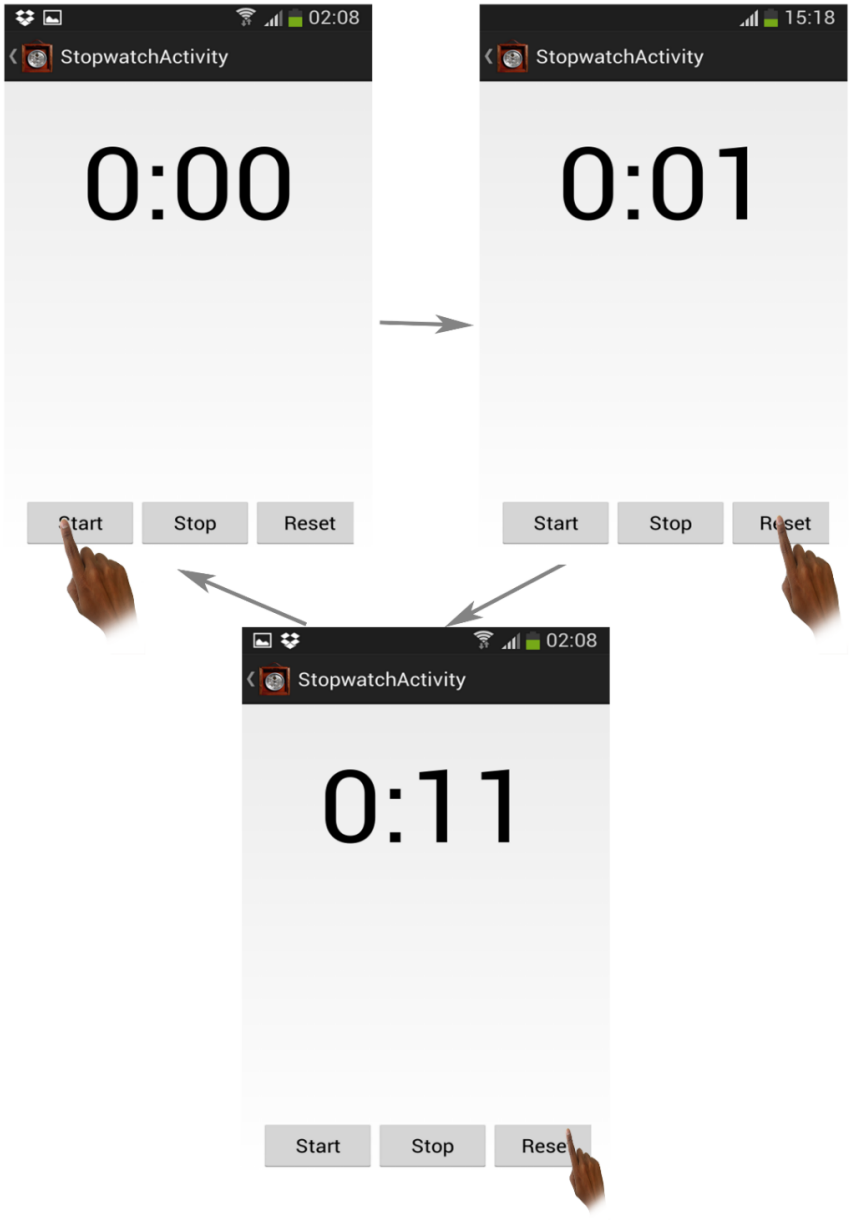
The To-do list application consists of two activities. The first activity displays the summary of the To-do list. The activity has an AddTask action button use to add tasks to the To-do list application. When the user clicks the AddTask action button the second activity launches. This is where the user inputs the To-do task information. The user can also add sub-tasks to a particular task. Graph 16 shows the process of adding a task and sub-tasks to the To-do list application. To add a task to the To-do list application, the user simply clicks on the Add Task button at the upper right corner of the screen. After the user had clicked the button, he/she is presented with a screen to add a task and subtasks. After Adding the task and subtasks, the user clicks on the Done button to save the data and then the task and it subtasks are shown in the To-do list application main screen as seen in Graph 16.



GRAPH 16. The To-do List Application.

5.5 Stopwatch

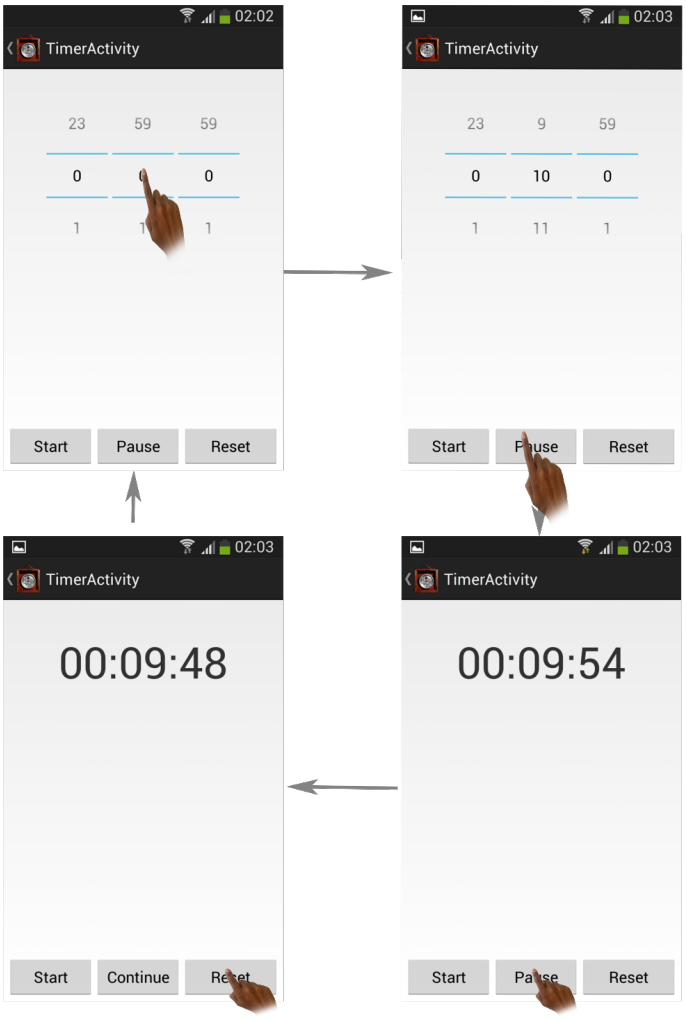
The Chronometer Stopwatch has a single activity with features a start button to start the Stopwatch, stop button to stop the Stopwatch, a reset button to reset the Stopwatch and an Android chronometer view to display the count. The Graph 17 demonstrates the use of the ChronometerX Stopwatch application. The user starts the Stopwatch by clicking on the Start button. After the user has triggered the Stopwatch, he or she can pause of reset the Stopwatch using the Pause button or Reset button respectively as shown in Graph 17.



GRAPH 17. Stopwatch Application.

5.6 Timer

The Chronometer Timer has a single activity with features a start button to start the Timer, pause button to pause the Timer, a reset button to reset the Timer, three number pickers used to set the Timer's hours, minutes and seconds respectively and lastly a text view to display the countdown. Graph 18 below is a pictorial illustration that explains how to use ChronometerX Timer application. First the user sets the desired time using the onscreen time pickers and starts the countdown by clicking on the Start button. After the countdown has begun, the user may pause or reset the Timer using the pause button or the reset button respectively.



GRAPH 18. ChronometerX Timer Application.

6 TESTING

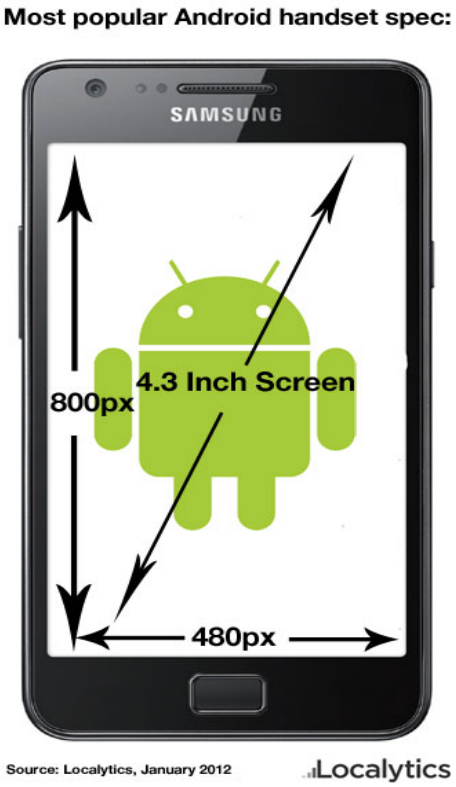
Android Mobile Application testing exercise involves testing a given Android application for it functionality, usability and consistency. However, there are critical factors to consider when testing an Android application. The Android application developer has to deal with problems that arise with multiplicity of screen sizes, operating system versions, device variation, mobile testing tool Availability, core hardware in Android devices and also sensor fragmentation. This thesis has taken into account only two factors: device variation and mobile testing tool Availability. (Android Open Source Project 2015; Chauhan & Kumar et al. 2013.)

6.1 Device Variation and Mobile Testing Tool Availability

Android powers hundreds of mobile device types with several different screen sizes. Therefore, it is important to design and test an application so that it is compatible with all screen sizes so as to be available to as many users as possible. Application compatibility with different screens and screen sizes is not enough. Each screen size presents different possibilities and challenges for user interaction. In order to truly satisfy and impress users, this project must go beyond merely supporting multiple screens and screen sizes. It must optimize the user experience for each screen configuration. (Android Open Source Project 2015b.)

By default, the Android system automatically scales and resizes an Android application graphical user interface (GUI) to make the application work on a variety of screens and screen sizes. (Android Open Source Project 2015b). The Android default scaling and resizing is not enough to optimize ChronometerX's user experience, and convince ChronometerX users that ChronometerX was actually designed for their devices. To achieve an optimized user experience, this project carefully considered screen size, screen density, screen orientation, resolution and platform versions while building and testing ChronometerX.

Screen size is the term used to describe the actual physical dimensions of a phone display screen, measured from corner to corner. Android (2015b) categorizes all phone display screens into four fundamental sizes: small, normal, large, and extra large. The ChronometerX is only optimized for small and normal, but it also works on large screens. Graph 17 shows the physical dimensions of a typical Android device.



GRAPH 19. A Picture of Android device screen size showing it screen dimensions (Localytics 2012.)

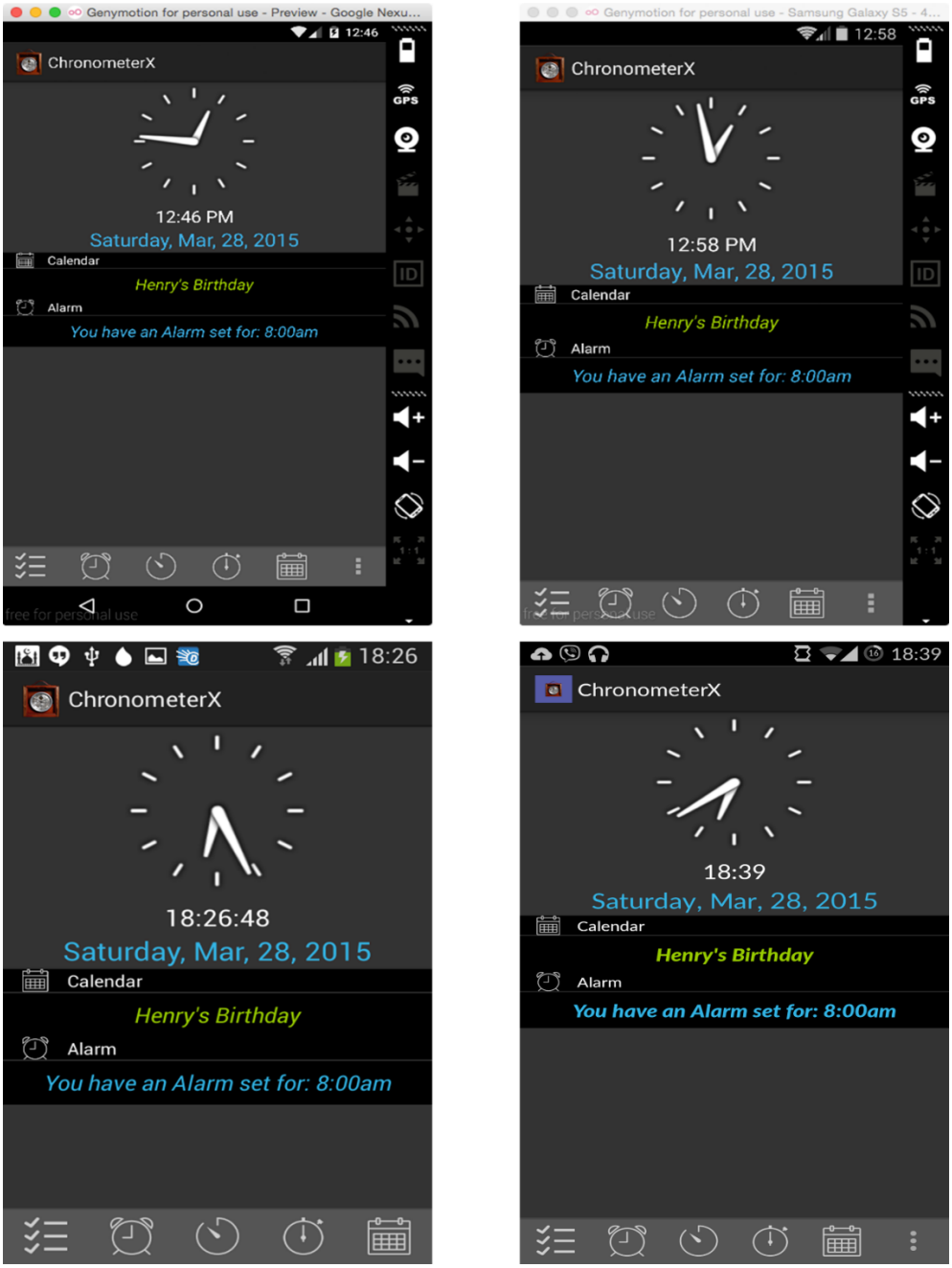
Screen density is defined in terms of Pixels. Webopedia (2015) defines pixels (Short for picture element) as a single point in a graphic image. Android (2015b) defines screen density as the number of pixels within the physical area of the screen, popularly known as dpi (dots per inch). Android (2015b) grouped screen densities into six density categories: low (ldpi, ~120dpi), medium (mdpi, ~160dpi), high (hdpi, ~240dpi), extra-high (xxhdpi, ~320dpi), extra-extra-high (xxhdpi, ~480dpi), and extra-extra-extra-high (xxxhdpi, ~640d). (Android Open Source Project 2015b.)

Low-density screens are screens with less pixels within the physical area of the screen, while “high” density screen contains more pixels within a given physical area, compared

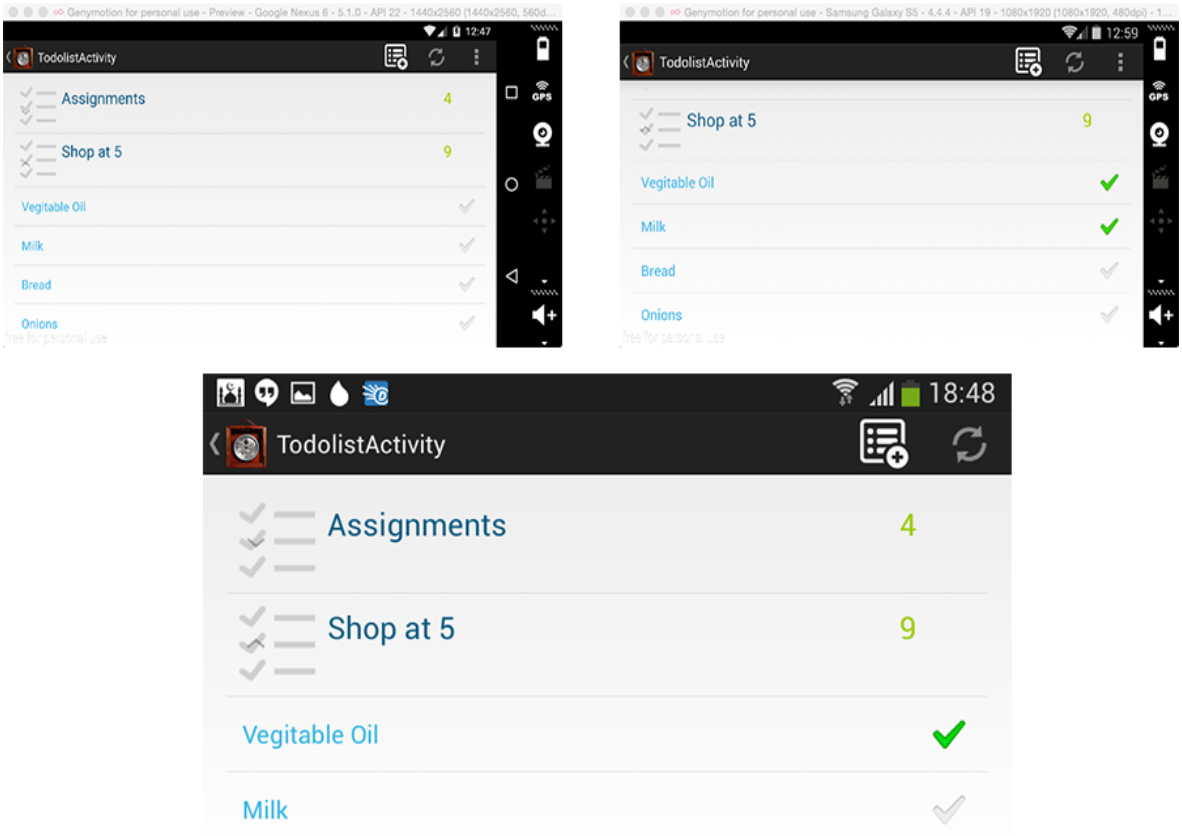
to "low" and "normal" respectively. Extra-extra-high has the most number pixels within a given physical area. In order to maximize user experience, layout sizes and positions of ChronometerX was expressed in density-independent pixels. This project also provided images of five different densities: low, high, extra-high, extra-extra-high, and extra-extra-extra-high. (Android Open Source Project 2015b.)

The term screen orientation has everything to do with the user's perspective. It is the position at which the user is using an Android device either straight (portrait) or slant (landscape). As a result, the screen's viewpoint proportion is either tall or wide, respectively. This project considered the fact that an Android device may run in either landscape or portrait orientation by default. This orientation might change from portrait to landscape of vice versa, if the user rotates the device. This rotation may result in loss of unsaved data. To avoid such contingency this project has configured some of ChronometerX's activities to work only in portrait orientation. (Android Open Source Project 2015b.)

The main activity, the Calendar activity, the Timer activity and the Stopwatch activity only work in portrait orientation to optimize user experience, while the rest of ChronometerX's activities can run in landscape or portrait orientation. Graph 20 shows ChronometerX's summary screen on two Android emulators and two real devices. Graph 21 shows the To-do Activity in landscape mode. Graphs20 and 21 show the result of testing ChronometerX across different screen size and orientation. The main goal of the Graphs 20 and 21 is to give a reader a glimpse of what ChronometerX looks like on real and virtual devices and to show that the application works as desired on different screen sizes and screen orientations.



GRAPH 20. The ChronometerX Summary screen on two emulators and on two real devices. Google Nexus 6, Samsung S5, Galaxy Trend and OnePlus respectively.



GRAPH 21. The ChronometerX To-do list screen on three different screens in Landscape orientation.

Screen resolution is the aggregate of the amount of physical pixels across the entire width and height of a device's screen. Android applications do not work directly with screen resolution. Thus, the generalized size and density groups advise Android developers to focus only on screen size and density when adding support for multiple screens. When defining GUI layers, a developer should express layout dimensions or position in terms of a virtual pixel unit called Density-independent pixel (dp). (Android Open Source Project 2015b.)

The density-independent pixel is the standard implied by the Android system for a "medium-density" screen. It is equal to a physical pixel on a 160 dpi screen. During device operation, the system automatically scales any Density-independent pixel units based on the density of the screen being used. The mathematical formula for converting DP units

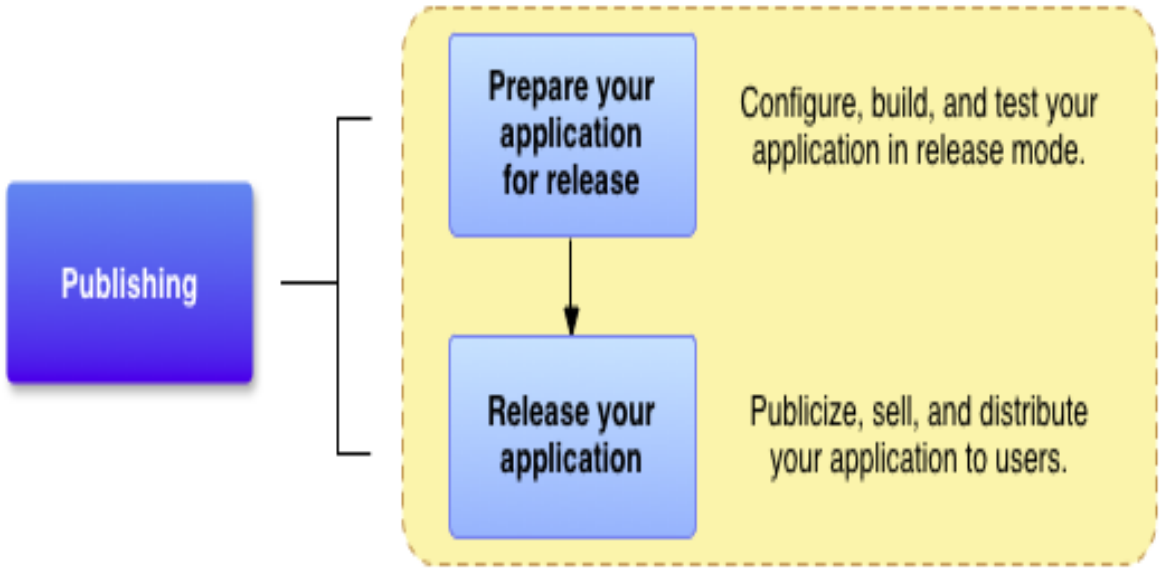
to screen pixels is expressed as: px = dp * (dpi / 160). Thus a screen of 240 dpi and 1 dp is equivalent to 1.5 physical pixels. While defining the ChronometerX's GUI, this project utilized dp units to guarantee efficient display of graphics on screens of different densities. (Android Open Source Project 2015b.)

Platform versions refers to the version of operating system an Android device is running. Consequently, this thesis sets the minimum SDK (software development kit) version to 17 and the target SDK version to 21 in order to maximize user experience, and serve as many users as possible. This thesis tested ChronometerX on different screen sizes and orientations using emulator and real Android devices. The ChronometerX test results are as designed and as desired.

7 DEPLOYMENT

After rigorous testing, troubleshooting and fixing bugs, the application is ready to be released to the users. The release process is the process of preparing and packaging an application for publication. The release process includes preparing an Android application for release. It encompasses configuring the application, building the application, testing the application in release mode and releasing the application to the users. The release process may also include advertising, selling and distributing the application. The release process is illustrated in Graph 22.

The final build is expected to be delivered to users is called the release candidate build. The release candidate build must be thoroughly tested before publishing. After the release candidate build is tested and verified, it becomes the release build that is the official build to be published. ChronometerX is functionally complete, but I need to take one final step before publishing it. I must package the application so that it can be deployed to users. (Darcey & Conder 2011, 383-385.)

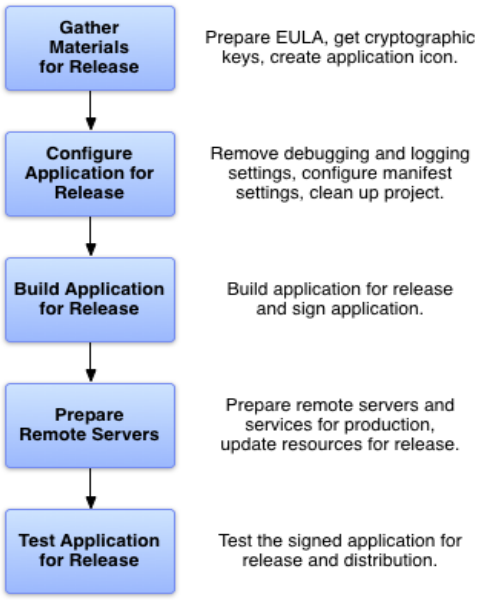


GRAPH 22. Preparing for release is a required developmental task and is the first step in the publishing process (Android Open Source Project 2015a.)

7.1 Preparing the Release Candidate Build

While preparing the release candidate build, it is important to resolve any open or outstanding problems or issues with the application that may hinder the release. First, all application features need to implement and test. Secondly, the application is troubleshooted and all bugs should be fixed. Finally, the programmer should delete all unnecessary diagnostic code from the application, which could affect application performance and verify that the application configuration settings in the Android manifest file are appropriate for release. (Android Open Source Project 2015a.)

To prepare an application for release, five main tasks should be performed (see GRAPH 23). Each main task may include one or smaller tasks. For example, if an application is to be released through Google Play special filtering rules should be added to the application manifest while configuring the application for release. Similarly, to meet Google Play publishing guidelines, screenshots and promotional text should be made while gathering materials for release. (Android Open Source Project 2015a.)



GRAPH 23. You perform five main tasks to prepare your application for release (Android Open Source Project 2015a).

7.1.1 Gathering Materials and Resources

Numerous supporting items are needed whist preparing an Android application for release. The official Android website encourages developers to acquire a cryptographic key for signing the application and an application icon. Including an end-user license agreement might be of great advantage too. The next two paragraphs further explain the supporting items needed while preparing an Android application for release as described on Android official website.

Every application must be digitally signed with a certificate that is owned by the application's developer before it can be installed on an Android system. The developer holds the private key to the certificate. The key referred to in this paragraph is a cryptographic key. The Android system makes use of the certificate as a means of identifying the author of an application in order to establish trust relationships between Android applications. A developer may sign an application using the tool provided in the Android SDK.

A cryptographic key is a private piece of information or parameter, usually a string of bits used by a cryptographic algorithm to determine a functional output when given an input. Without this key, the algorithm result will be garbage. It is typically used to transform plain test into the cipher and vice versa. These certificates do not need the intervention of any certificate authority. The Android system allows developers to sign their applications with a self-signed certificate. (Android Open Source Project 2015a.)

An application icon is a graphic that occupy a small portion of the screen to provide a quick, intuitive representation of an action, a status, or an Android app. The application Icon must follow recommended icon guidelines. An application may be installed on a variety of devices that offer a range of pixel densities. To make an application icons look great on all devices a developer must provided multiple sizes of the application icons. Graph 24 shows the application icon dimensions in DP units.

GRAPH 24. Recommended Android application Icon dimensions in dp units (Android Open Source Project 2015a).

Application's icon helps users identify an application on a device's Home screen and in the Launcher window. It also appears in Manage Applications, My Downloads, and elsewhere. In addition, publishing services such as Google Play display application icon to users. Thus a developer should be mindful of their application icon and make sure it is in accordance with the Android recommended icon specifications. To publish an application in Google Play, a developer will need to create a high-resolution version of the application icon. The explanation given in this report is limited. For more details on Android application icons and graphics see “Graphic Assets for your Application” on Android official website. (Android Open Source Project 2015a.)

7.1.2 Configuring Your Application for Release

The next thing to do after gathering all supporting material is to configure an application for release. This section provides a summary of the configuration changes Android (2015a) recommends that developers make to their source code, resource files, and application manifest before releasing their application.

The first and one of the most important step involved while configuring an application for release is to choose a good package name. The Android application package name cannot be changed once it has been published, so it is recommended to choose a package name that is suitable over the life of an application. The application package name can be set in the application's manifest file. (Android Open Source Project 2015a.)

The second step is to turn off logging and debugging. Developers should deactivate all logging and disable the debugging option before building the application for release. Deactivate logging by removing calls to Log methods in the application source files and disabling debugging by removing the Android debuggable attribute from the <application> tag in the application manifest file, or by setting the Android debuggable attribute to false in the manifest file. Also, remove any log files or static test files that were created in the project. After turning off logging and debugging, the project directories should be cleaned. Developers should clean their project and make sure it agrees with the directory structure described in Android Projects. Leaving stray files in an Android project can prevent an application from compiling and cause the application to behave unpredictably. (Android Open Source Project 2015a.)

Review and update the manifest and gradle build settings. Developers should verify that <uses-permission> element, Android icon and Androidlabel attributes, Androidversion code and Androidversion name attributes are set correctly. The developer should specify only those permissions that are relevant and required for his or her application, specify values for these attributes that are located in the <application> element and lastly, specify values for these attributes that are located in the <manifest> element. (Android Open Source Project 2015a.)

After Reviewing and updating the application manifest and gradle build settings, a developer should address compatibility issues. To make an application available to the largest number of users, a developer should add support for multiple screen configurations and optimize his/her application for Android tablet devices. Android developers should consider using the Support Library while designing an application for devices running Android 3.x or older. The Support Library provides static support libraries that can be added to an Android application, which enables developers to use APIs that are either not available on older platform versions or use utility APIs that are not part of

the framework APIs. If an Android application uses remote servers or services, a developer should use an updated URLs for servers and services. The production URL or path for the server or service should be used and not a test URL or path. (Android Open Source Project 2015a.)

The last step is to Implement Licensing (if you are releasing on Google Play). If a developer is releasing a paid application through Google Play, Android(2015a) advices the developer to consider adding support for Google Play Licensing. The Licensing gives developers control access to their application. The developer does not have to use Google Play Licensing even if they are publishing an Application through Google Play (it is simply optional). ChronometerX is an alpha level application and is not ready for the Play Store.

7.1.3 Building Your Application for Release

When an application is completely configured it is time to build it into a release-ready .apk file that is signed and optimized. The JDK includes the tools for signing the .apk file (Jarsigner and Key tool); the Android SDK also contains software tools capable of compiling and improving the .apk file (Android Open Source Project 2015a). ChronometerX was developed using the Eclipse Android development environment. Eclipse is able to automate the entire build process.

7.1.4 Preparing External Servers and Resources

If an application relies on a remote server, the developer should make sure the server is secure and that it is configured for production use. This is particularly important if an application implementing in-app billing and signature verification step is performed on a remote server. Also, if an application fetches content from a remote server or a real-time service (such as content feed); the developer should be sure the contents being provided is up to date and production-ready. (Android Open Source Project 2015a.)

7.1.5 Testing Your Application for Release

Testing the release version of an Android application helps to ensure that the application runs properly on realistic device and network conditions. Ideally, a developer should test his or her application on at least a handset size device and a tablet size device to ascertain that the user interface elements are sized appropriately and that the application's performance and battery efficiency are satisfactory. (Android Open Source Project 2015a.)

This report contains only a few Android situations that should be consider when testing an application. After an Android application is tested rigorously and the release version of the application behaves correctly. The developer can then release the application to users. For detailed information on Android release process, check “Releasing Your Application to Users” on Android official website.

8 MAINTENANCE

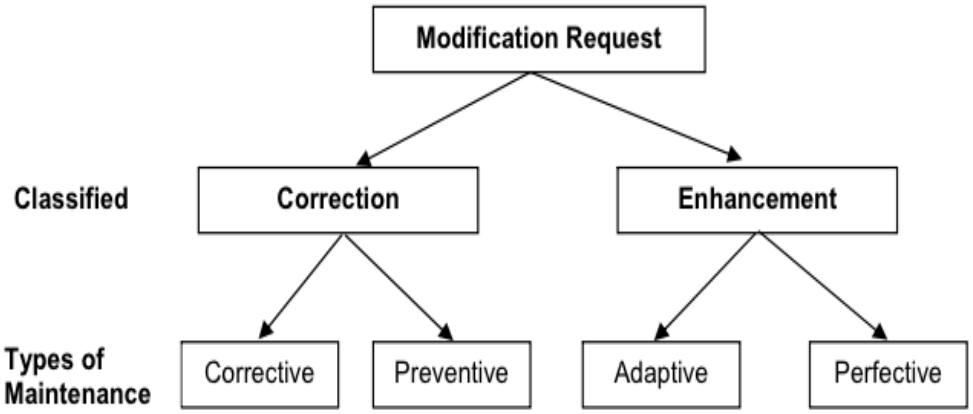
The need for maintenance in all human endeavours cannot be over emphasized. Oxford Advanced Learner's Dictionary (2015) defines the maintenance of something as “the act of keeping something in good condition by checking or repairing it regularly”. This thesis is more concerned with software maintenance, which has a narrower meaning. Software maintenance is defined as the aggregate of all activity (either compulsory or obligatory) required to provide a cost effective support to a software or software system. These activities may be performed during the pre-delivery stage or at the post-delivery stage. The support activities in this context may involve correcting errors, improving the software design, making enhancements, creating Interfaces with other systems or software retirement. (The International Standard 2006, 4.)

8.1 Types of Maintenance

Software or system maintenance may be grouped under four categories: adaptive maintenance, corrective maintenance, perfective maintenance and preventive maintenance. Graph 25 shows the types of software maintenance. Adaptive maintenance involves the modification of a software product performed after the software has been delivered to the software buyer. Adaptive maintenance is provided to adjust the software so that it is able to adapt to change in its operating environment. The phrase operational environment in this context denotes external conditions and the influences that act on the system. For example, accommodation change in the hardware of an operating system. (The International Standard 2006, 3.)

Corrective maintenance is performed in reaction to malfunctions, faults and defects discovered in software or system after delivery. This fault might be as a result of designing error or logical or coding error. Design errors occur when changes made to the software are flawed, deficient, wrongly transmitted, or the change request is

misunderstood. Logical errors are made during the coding phase and may result in incorrect and unexpected behaviour. These behaviours might lead to software crash, invalid tests and conclusions. All these errors are referred to as residual errors. Residual errors derail the software behaviour from the established specifications. Note that the need for corrective maintenance is usually initiated by bug reports from the user. (The International Standard 2006, 3.)



GRAPH 25. Modification Request chart (International Standard 2006).

Perfective maintenance involves updating a software to improve functionality by implanting new services and correcting latent defects before they escalate to a major malfunction. Perfective maintenance might include improvement of graphical user interface, improving code to raise efficiency and reduce responds time. (The International Standard, 2006, 4.)

Preventive maintenance encompasses the act of anticipating tackling imminent error and the modification of a software product to correct these errors before they occur. Preventive maintenance might include updating documentation, code restructuring and code optimization. Preventive maintenance also includes updating the software document so that it reflects the current state of the software. (The International Standard 2006, 4.)

ChronometerX will be maintained by uploading the source code to GitHub in order to keep it alive and enabling other programmers from around the world contribute to its development. Babatunde Anafi will do other maintenance such as corrective maintenance. GitHub is a website that let users share code with friends, colleagues, classmates, and even complete strangers. It had over eight million users at the time this thesis was written. (GitHub 2015.)

9 EVALUATION AND DISCUSSION

This section explains the evaluation process. The evaluation process may be described as a method of determining the advantages, the importance and significance of an entity. Thus, the results of such a process are called evaluations (Scriven 2009). The “entity” being evaluated in this thesis is ChronometerX (Software Application). Software evaluation is a process that assesses a given software application to determine its advantages, importance and significance to a user, in order to ascertain whether or not it is well suited for the specific needs of a client. Application software is evaluated by scrutinizing the tools and resources provided by the software base on a number of evaluation criteria and practical experiment to determine if the product is appropriate or another application software will serve the user better. The evaluation criteria are usually based on the Software requirements. (Baumgartner & Payr 1997, 2; Tatum et al.2015.)

9.1 Goals and results of evaluation

Success is defined as the favourable execution and the end of an attempt or endeavour; the achievement of one's goals (Dictionary.com 2015). For any process to be successful, it must have a pragmatic goal. In the context of the software evaluation, Baumgartner & Payr (1997) claim one or more of three simple questions can be used to express the goal: “Which one is better? How good is it? Why is it bad?”

Which one is better? This kind of evaluation is achieved by comparing alternative software, software prototypes or software versions to determine the one that will best serve the needs of a client for the specific purpose. How good is it? The goal of this question or the process it represents is to determine the degree of desired qualities of a finished software. Why is it bad? This scrutinizes to spot the failures in specific software; The result is often used to improve software functionalities. The evaluation question of this thesis is: Which one is better? The goal of this thesis evaluation process is to

determine if ChronometerX is useful to the Android user by comparing it to the native Android clock application. The criteria for comparison is stated in section 9.2.

9.2 Evaluation criteria

Software usability is the measure of the efficiency and effectiveness of a software product with respect to the satisfaction a customer derives from a software product (Stewart 2000). To successfully evaluate a software application, the evaluation criteria should be based on general guidelines and standards.

The guidelines and standards should take into account the application's context of use. The context of use is defined based on the user, the task to be performed, the equipment to be used and the application environment. The user in this case is the Android phone user and the equipment is the Android phone. These evaluation criteria for ChronometerX is based on the ChronometerX's user requirements. ChronometerX and the Android clock application will be scrutinized to determine which one meets the user requirements.

9.3 Evaluation techniques

Evaluation technique is the method employed during a software evaluation process, which may be defined in terms of organisation or behaviour. Evaluation techniques are categorised into two categories: the descriptive evaluation techniques and the predictive evaluation techniques. Both descriptive evaluation techniques and the predictive evaluation techniques should be used in every evaluation. (Baumgartner & Payr 1997, 7; Gediga & Hamborg 2001.)

The descriptive evaluation technique is based on the descriptions. It is used to objectively describe the main problem of the software. The descriptive evaluation technique is user-oriented and is divided into two approaches: behaviour-based evaluation techniques and opinion-based evaluation methods. The behaviour-based evaluation techniques are based

on the user's reaction to a software application. This technique observes and records user behaviour. Opinion-based evaluation methods are also based on user reaction to a software application, but the method collects data by confronting the user. Opinion base evaluation methods may come in the form of interviews, surveys or questionnaires. (Baumgartner & Payr 1997, 7; Gediga & Hamborget al. 2001.)

Usability testing employs both behaviour and opinion based techniques, in addition to a number of experimental controls. For example, a hypothesis about a given software product provided by an expert. Descriptive evaluation techniques always involve a software prototype and a user or users to test the prototype. The data collected during Usability testing may require additional analysis and explanations in order to draw reasonable conclusion. (Baumgartner & Payr 1997; Gediga & Hamborget al. 2001.)

The predictive evaluation technique is an expertise-based technique unlike the descriptive evaluation technique which is user-based. The goal of predictive evaluation technique is to collect information to help improve future software development and reduce usability errors. The predictive evaluation technique is expertise-based; users may partake in certain cases. A predictive evaluation technique relies on data. This data is almost always a simulation of real users by the expert in charge. (Baumgartner & Payr 1997.)

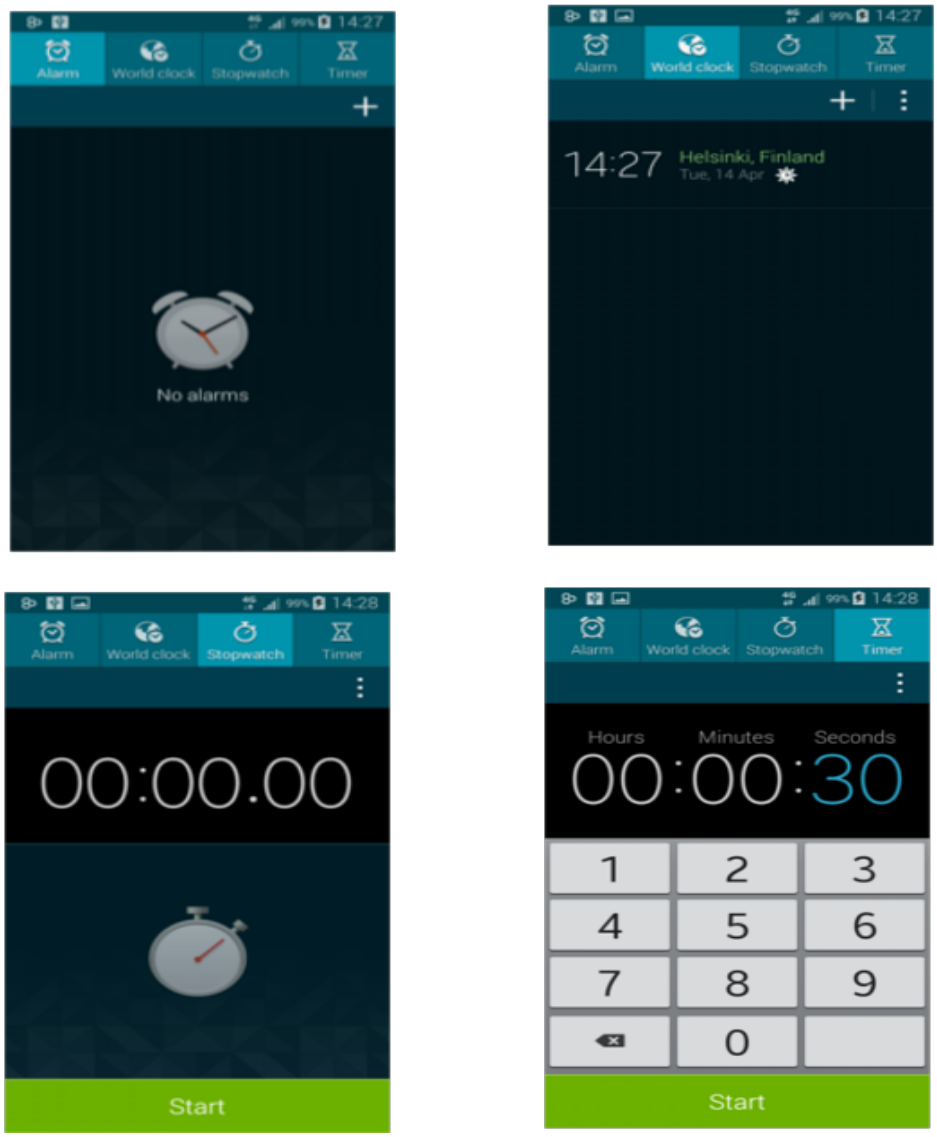
9.4 ChronometerX Evaluation

The ChronometerX is evaluated based on the descriptive evaluation (usability testing) technique, by comparing the ChronometerX application with the Android clock application based on the evaluation criteria (User requirement). See table 1 on page 10 of this thesis for ChronometerX's User requirements.

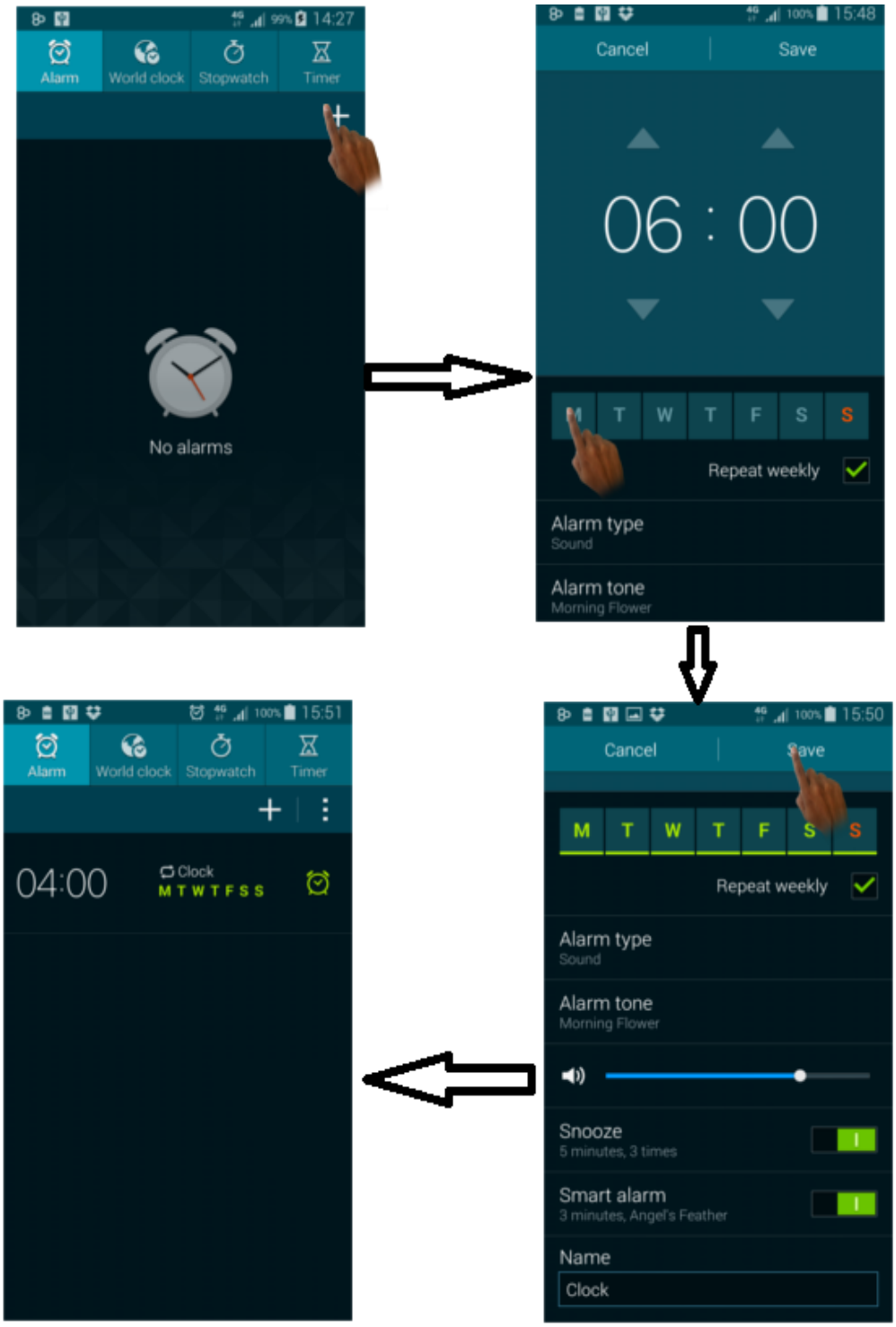
The Android clock application consists of four different utilities: alarm, world clock, stopwatch and timer. Only three of the utilities present in the Android clock application are listed in the evaluation criteria, but the ChronometerX contains all the five utilities listed in the criteria. Android clock application's world clock is a utility that shows the

current time in a particular place or country at a particular time. Graph 26 shows the main activities of the utilities present in the Android clock application. The first picture to the left shows the alarm, the second screenshot displays the world clock application, below first picture is the stopwatch activity screenshot and to its right is Android clock's timer activity.

A user sets an alarm in Android clock by clicking on the Add action button on the upper right corner of the screen below the Timer action button. The Add action button is represented by a plus sign. After the user clicks on the plus sign, the user is presented a screen where he/she can set the Alarm details. When the user is done setting the details, he/she can either save the Alarm by clicking on the Save button or cancel the operation using the cancel button. Graph 27 describes Android clock application’s alarm utility. It uses hand clicks to illustrate the process involved in setting an alarm in the Android clock application’s alarm utility.



Graph 26. The Android clock application.



Graph 27. The Android clock application's Alarm Utility.

9.5 Evaluation Conclusion

The Android clock application showcases better graphical user interface than the ChronometerX's GUI. It also includes world clock utility which is not present in the ChronometerX. The Alarm utility of the Android clock application presents additional features such as, the snooze and smart alarm, which are not present in ChronometerX. The smart Alarm feature of the Android clock application let an Alarm begin to ring at a low volume some minutes earlier and slowly increase in volume. The Android clock application is a good application, but it does not meet the evaluation criteria (The ChronometerX user requirements) while the ChronometerX meets the user requirements. Chronometers contain five utilities: Alarm, Calendar, Stopwatch, Timer and To-do list utility while Android clock application has only four and only three are listed in the user requirements. The Android clock application only shows all available alarms, but ChronometerX has a separate screen to show upcoming events such as next Alarm and next Calendar event. Based on the Evaluation criteria, ChronometerX is better suited for the user than the Android clock application.

10 CONCLUSION

The advancement in technology has increased the capacity and the capability of mobile devices significantly. Smartphone can now perform computer-like functions. Android operating system powers 80% of activated mobile devices, which includes smartphones. Thus, the Android Play Store is a large and highly lucrative market for mobile software developers. The fast increasing number of Android mobile devices users has attracted a large number of software developers. Consequently, the number of applications in Android Play Store is increase at a very fast rate, there is always more than one application for the same purpose. This situation has challenged developers to become more innovative in their development.

The goal of this thesis project was to recreate the mobile clock application and document the six stages of the development life cycle of the application; requirement gathering and analysis, design, implementation or coding, testing, deployment and maintenance. The recreation was achieved by assembling five different applications into one application package: Alarm, Calendar, Stopwatch, Timer and To-do list and also deliver the summary of upcoming events or tasks to users on a separate screen (Activity).

The summary screen is the major innovation in the project. After ChronometerX was developed during this thesis work, it was evaluated and compared to the Android clock application based on the ChronometerX's user requirements (see page 11 table 1) and was found to be better suited for the user than the Android clock application. ChronometerX is an alpha level application and is yet to be published in the Android Play Store. This thesis only describes the development cycle and does not include the actual implementation process (the coding phase).

This thesis encourages software developers to take advantage of the advancement in mobile technology and the lucrativeness of the Android Play Store to become more innovative. This thesis also tries to give software developers who want to go into Android software development a glimpse of all the processes involved in Android software development and also to encourage recreation and innovation.

REFERENCES

Android Open Source Project. 2015a. Launch Checklist. Available: http://developer.Android.com/distribute/tools/launch-checklist.html. Accessed 25 March 2015.

Android Open Source Project. 2015b. Supporting Multiple Screens. Available: http://developer.Android.com/guide/practices/screens_support.html. Accessed 21 March 2015.

Android Open Source Project. 2015c. The Android Story. Available: http://www.Android.com/history/. Accessed 15 April 2015.

Android Open Source Project. 2015d. Input Technical Information. Available: http://source.Android.com/devices/input/. Accessed 15 April 2015.

Android Open Source Project. 2015e. Sensors Overview. Available: http://developer.Android.com/guide/topics/sensors/sensors_overview.html. Accessed 16 April 2015.

Android Open Source Project. 2015f. Widgets. Available: http:// http://developer.Android.com/design/patterns/widgets.html. Accessed 18 April 2015.

Android Open Source Project. 2015g. Android NDK. Available: https://developer.Android.com/tools/sdk/ndk/index.html. Accessed 18 April 2015.

Android Open Source Project. 2015h. Developer Workflow. Available: http://developer.Android.com/tools/workflow/index.html. Accessed 18 April 2015.

Android Open Source Project. 2015i. Android Activity. Available: http://developer.Android.com/reference/Android/app/Activity.html. Accessed 19 May 2015.

Baumgartner, P. & Payr, S. 1997. Methods and practice of software evaluation: The case of the European Academic Software Award (EASA). In: Proceedings of ED-MEDIA 97 - World Conference on Educational Multimedia and Hypermedia. Available: http://peter.baumgartner.name/wp-

content/uploads/2013/08/Baumgartner_Payr_1997_Methods-and-Practice-of-Software-Evaluation_EASA.pdf. Accessed 11 April 2015.

Chauhan, M. & Kumar, M. 2013. Best Practices in Mobile Application Testing (White Paper). Available: http://www.infosys.com/flypp/resources/Documents/mobile-application-testing.pdf

Darcey, L & Conder, S. 2010. Sams Teach Yourself Android Application Development in 24. Indianapolis, USA: Sams publishing.

Dictionary.com. 2015. Success. Available: http://dictionary.reference.com/browse/success. Accessed 13 April 2015.

Gediga, G. & Hamborg, K. 2001. Encyclopedia of Computer Science and Technology: Evaluation of Software Systems. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.87.8362&rep=rep1&type=pdf. Accessed 11 April 2015.

GitHub 2015. About GitHub. Available: https://github.com/about. Accessed 1 April 2015.

Janssen, C. 2015a. Android SDK. Available: http:// http://www.techopedia.com/definition/4220/Android-sdk. Accessed 18 April 2015.

Janssen, C. 2015b. Native Mobile App. Available: http:// http://www.techopedia.com/definition/4220/Android-sdk. Accessed 18 April 2015.

Hardesty, L. 2010. The MIT roots of Google's new software. Android. Available: http://newsoffice.mit.edu/2010/Android-abelson-0819. Accessed 19 April 2015.

Leger, B. 2012. Android market not as fragmented as many think. Available: http://info.localytics.com/blog/Android-not-as-fragmented-as-many-think. Accessed 2 April 2015.

Leiva, A. 2014. MVP for Android: how to organize the presentation layer. Available: http://antonioleiva.com/mvp-Android/. Accessed 19 April 2015.

Nickinson, P. Google Play Store. Available: http://www.Androidcentral.com/google-play-store. Accessed 13 April 2015.

Scriven, M. 1991. The Nature of Evaluation. Available: http://www.rismes.it/pdf/Scriven_domain_evaluation.pdf. Accessed 11 April 2015.

Sheusi. J. 2013. Android™ Application Development for Java® Programmers. Boston, USA: Course Technology PTR.

Soomro, D. 2014. How to Install APK Files: Sideloading on Android Available: http://www.ubergizmo.com/how-to/how-to-install-apk-files-sideloading-on-Android/. Accessed 19 April 2015.

Stewart, T. 2000. The International Standard. Ergonomics user interface standards: are they more trouble than they are worth? Ergonomics. Geneva, Switzerland: International Organization for Standardization (ISO).

Tatum, M. 2015. What Is Software Evaluation? Available: http://www.wisegeek.com/what-is-software-evaluation.htm. Accessed 13 April 2015.

Techopedia 2015. Cryptographic Key Available: http://www.techopedia.com/definition/24749/cryptographic-key. Accessed 17 May 2015.

The International Standard 2006. Software Engineering — Software Life Cycle Processes — Maintenance. Available: http://dis.unal.edu.co/~icasta/ggs/Documentos/Normas/14764-2006.pdf. Accessed 1 April 2015.

The Oxford Advanced Learner's Dictionary. 2015. Maintenance. Available: http://www.oxfordlearnersdictionaries.com/definition/english/maintenance. Accessed 23 March 2015.

The Statistics Portal 2015. Number of available applications in the Google Play Store from December 2009 to February 2015. Available: http://www.statista.com/topics/1001/google/. Accessed 17 April 2015.

Sommerville, I. 2010. Software engineering. 9th Edition. USA: Pearson.

UML diagrams 2015. UML Use Case Diagrams. Available: http://www.uml-diagrams.org/use-case-diagrams.html. Accessed 20 May 2015.

Webopedia 2015. Pixel. Available: http://www.webopedia.com/TERM/P/pixel.html. Accessed 5 April 2015.