DRIBBLE Individual Report

A. Campbell

School of Electrical & Information Engineering, University of the Witwatersrand, Private Bag 3, 2050, Johannesburg, South Africa

Abstract: The design and implementation of the Dribble Android Mobile application is detailed in the report below. The requirements were to design a software system that incorporated information storage, processing, calculating and visualisation. To meet the requirements, Dribble, a geosocial networking mobile application, was implemented. Dribble mobile application was developed using the Android SDK, Java and XML in the eclipse IDE. A full description of the design and techniques are detailed as well as a short discussion on the implemented testing. The designed system is critically analysed, recommendations and trade-offs described. While the designed application was able to produce the desired functionality, the application compromised on speed for the implementation of web services, and functionality over completed error handling. The modularity priority of the system was effectively achieved and sufficient testing and integration allowed the application to fulfil the success criteria stipulated by the developers. The time management of the production was recorded and shown that the actual time was 1.5 times longer than the estimated time.

Key words: software development, android, java, xml, social networking, geosocial

1. INTRODUCTION

Social networking is a rapidly spreading and evolving technology that has infiltrated into the 'every-day life'. The chosen software development system, 'Dribble', explored a new perspective on the social networking platforms - mobile focused, geosocial networking. The generalised model of the overall system can be found in Appendix A. The client application uses a location-based, short message posting structure, implemented on Android mobile phones. The mobile-side of the application provided the Graphical User Interface (GUI) component of the software system, with minimum processing. The focus of this report is to fully describe, justify and critically analyse the chosen design and implementation for the mobile-side of the Dribble application.

This report will offer a brief background into the technologies, techniques and tools implemented on the client side of Dribble (Section 2.), followed by the project problem definition (constraints, requirements and success criteria) in Section 3. Thereafter, a short outline of the complete Dribble system is offered to put the mobile-side into perspective with the larger, complete application. The design of the application is detailed in Section 5., followed by an account of the testing and logging that was utilised. In a software development system, it is important to critically analyse the proposed solution, in order to determine strengths and weaknesses; conformance to the specifications and requirements and quality of the design. This is given in Section 8., followed by suggested improvements in the Recommendations Section (9.). Lastly a brief summary of the time management is discussed, with the full time sheet added in Appendix D.

2. BACKGROUND

The Dribble mobile application focused on employing three main technologies: Android, XML and Java. While XML and Java had been utilised before, Android was a completely new technology. The Integrated Development Environment (IDE) utilised was Eclipse, due to the powerful and readily available Android Eclipse plug-in [1]. The development platform and environment in a software development application needs to be carefully chosen to enable maximum functionality while maintaining the prescribed requirements. The IDE, tools, languages and technologies integrated within this application, were selected based on their compatibility and flexibility with one another and the efficient and well-integrated platform they created.

2.1 Android

Android is a complete software stack consisting of an operating system, middleware and key mobile applications [2]. More importantly in this context, Android has an Android Development Kit (SDK), containing the tools and APIs for developers to create a diverse range of innovative mobile applications that can utilise features within the mobile phone to maximise functionality. The choice to use Android was primarily based on the availability of the mobile phone and the extensive SDK offered. Another benefit is that Android is open-source and therefore met the stipulated requirements [2]. It is easily adapted to incorporate new technologies and is continuously evolving, with Android 2.1 just recently released [2].

Android offers a series of features that can be integrated with the phone's core functionality [2] (text messaging, camera utilisation, making calls and obtaining a real-time location of the phone). This reinforces the preference to using it within the implementation of the Dribble application. Android's key features include: application framework (reuse and replacement of components), Dalvik virtual machine, integrated browser, SQlite, GSM Telephony, Camera, GPS, compass, accelerometer and a rich development environment [3]. The Android application runs on its own instance of the Dalvik Virtual Machine (VM). Android runs each application on its own Dalvik and allows for multiple instances of the VMs. Most of the core functionality in Android, is generated using core Java libraries [3]. Due to the Linux kernel, further functionality (for instance - threading) is possible. Core services include memory management, network stack and driver model [3, 4]. A more detailed description of the Android building blocks can be seen in Appendix E.

Android is an open platform granting developers full access to all the APIs with which the native applications were created [5]. Google has distributed Google Maps API which means developers can captilise on the existing libraries to use and display information about location [5]. The extensive access to API, creates a dynamic and innovative platform to optimise the latest technologies into an application. Android has also built in an intensive security model, in which permissions are granted during installation and are thereafter fixed, which increases security by preventing unwanted information sharing (privacy) and malware from being added without user consent [5].

2.2 Java

Java is a programming language that has developed into a comprehensive framework, with different application focused SDKs or in this case Java Development Kits (JDKs) like JavaSE, JavaME, JavaEE [4] as well as a rich development community. In this application, a specific form of Java specialised for Android was utilised for the mobile-side and JavaEE was the baseline for the server-side application. While Android does have the capabilities to support a set of C/C++ libraries, Java is the primary developing language integrated with Android to provide extensive functionality.

2.3 XML

eXtensible Mark-up Language (XML) was developed by W3C to standardise the format of information for storage and transportation [6]. XML uses nodes (tags) and attributes to store information [6], but does not execute instructions or process information, it is just text and can be passed as a string. One of the benefits of XML, is the lack of a predefined syntax, which allows developers to create XML Schemas to define the protocol of the XML message. For the purposes of this application, XML was used in two ways: firstly, to convey objects between the client and the server, using serialisation (discussed in more detail in the next sections). Secondly, XML was used in the presentation layer to define the layouts and views, and their attributes to create the GUI, and utilises the other resources like the drawable items and parameters with stored values.

3. PROBLEM DESCRIPTION

A software development application needed to be developed that incorporated information storage, processing, calculation and visualisation, including the Graphical User Interface (GUI), and had to be implemented with modularity as a priority. The choice of platform was left open for the developers to decide on, and the scope of the project was to be defined by the members of the devel-

opment team, prior to development.

3.1 Constraints

The client side was limited in terms of the depth of processing that would be preformed on the mobile phone application itself. The aim of Dribble was to separate components to ensure modularity. This meant that all processing needed to be completed on the application server. In order to achieve modularity, both the server and client would be restricted to a common package (and objects), to ensure compatibility for communication objects to be sent/received. While securing an Android mobile phone would allow for practical implementation of the application, it restricts the development platform to Android and Java.

There were also several mobile specific assumptions that were made before implementation. These include: sufficient access to the internet either over 3G or EDGE as well as sufficient bandwidth and speed to run. For the location-based functionality, Google Maps needs to be available and accessible. Connectivity is a high priority; the application therefore relies on an accessible server with adequate load capabilities, as the aim of the application is to acquire mass population usage. Speed is also an important consideration for releasing technologies and applications, and needs to be considered in order to ensure the correct trade-offs are implemented. It is assumed Dribble users have Android mobile phones and are familiar with the layouts and GUI of a typical Android application.

3.2 Requirements

Some requirements for the individual components are directly linked to that of the group project. Firstly, the application needs to be developed in groups of four-five members, committing to an average of of 4.5 hours per week over the course of the allocated seven weeks. Each member is required to utilise at least one new development tool, technique or programming language. The final product needs to be released under an open-source licence agreement, and therefore requires that appropriately licensed code only (of no more than 60% of the final solution) may be used in the development of the project. The mobile side was required to provide the GUI for the application, focusing on a use-case model, and quality of use for the client. The developers of the Dribble system also enforced requirements to ensure scalability of the final design and the adherence to standard, existing technologies.

3.3 Success Criteria

The member-defined success criteria for the final designed system is to produce a product that is fully integrated across all systems (highly modular), ready to be released to market. The product would therefore need to be implemented and simulated as a real world application. Therefore the focus for the development would be to implement modular core functionality first, and build on additional features after extensive testing on the system has taken place. Vital aspects of the functionality include obtaining a GPS location, the ability to communicate with the server, XML parsing and the GUI layout and structure.

4. SYSTEM OVERVIEW

The Dribble system model is depicted in Appendix B and illustrates the main division into two components - the Mobile Application (Android) and the Remote Application Server (Glassfish). The Mobile Application is further divided into two layers: logic (Java source code) and presentation (XML). The application is executed on the HTC Dream mobile phone and provides the GUI with which the users will interact. The application uses web services to initiate a connection with the remote server, and utilises a common package to define the two main object classes ‘Drib’ and ‘DribSubject’. These classes are used to define messages and message subjects, respectively, and are sent and received over HTTP as serialised objects in XML. A variety of network technologies are available: 3G, EDGE, GPRS or HSDPA. On the remote server, the processing of requests is handled by Enterprise JavaBeans which communicate with the database to return the desired results. The mobile application then uses Java and XML to graphically display the Dribs and DribSubject (see Appendix C). There are four tabs: Maps, Topics, Dribs, New, which correspond to MapsActivity, SubjectActivity, DribActivity and CreateNewCreate Activity, in the source code.

5. DESIGN

This section will discuss key design practises and techniques that were utilised in the implementation of the mobile application. It will also highlight important features and describe the source code of the application in detail.

5.1 ICONIX

The initial steps in the design of the mobile application originate from the ICONIX process [7]. The process was initiated by defining the problem statement, followed by discussing the prototype GUI and drawing it to establish the basic layout and to determine how the core functionality would be implemented use-case model (see Appendix F) for the system. Next, a use-case model was constructed. This was an important step to ensure all core-functionality had been accounted for. The application used use-case drive (core functionality) development.

6. Development Platform

The Android SDK provided tools and APIs necessary for the Dribble development. A specific form of Java was utilised due to its compatibility with Android. Although other languages have libraries which can be used with Android, Java is still the dominant development language. Android uses the Android Development Tools (ADT) plug-in for the Eclipse IDE. The integration of ADT and Eclipse made the Eclipse IDE the obvious choice. The correct project structure for an Android application is automatically generated, maintaining and managing resources, such as XML layout files and drawable items (images, icons) and are compiled automatically into the Java source code. It also provides tools integration, custom XML editors, and a debug output pane called LogCat. LogCat provides automatic session logging, but also allows for custom-made logs embedded in the code to highlight function executions, thread and error checking and the ability to monitor warnings and override methods. The HTC Dream was used for development with Android version 1.5. Although a newer version of Android has been released, tests were not conducted on the compatibility of the application using Android 2.1.

6.1 Logic Layer

6.1.1 Activities Android uses Activities to define functionality by limiting an Activity to offering one functionality [8]. The main Activity was a class called 'DribbleTabs', which extended the 'TabsActivity' parent class. This type of activity divided the layout of the application into tabs so that fast, intuitive user interactions can be performed, optimising the use of the touch-enabled devices. This also clearly highlights the separation of application activities. Tabs can be changed by clicking on the tab icon and will automatically (functions calling other activities) using the allocated Tab Host and desired Intents (intents are used to activate components).

Two other Activity classes were implemented: 'ListActivity' and 'MapActivity'. ListActivity was used for both the Dribs and Topics tabs, as it contains automatic scrolling, dynamically created lists to be view correctly and it allowed for a set layout of each element in the array that stored either the Dribs or the DribSubjects. An 'ArrayList' was used to store and send/receive a list of objects. Although a number of other layouts were considered and even tested, it was established that ListActivity was optimal for the purposes of this application. MapActivity was used to set up and establish Google Maps (using the API built into Android SDK) and performing location detection. The initial location was found and statically stored for access within another class. Due to inexperience with Android, it was established that using static variables allowed for parameters and objects to be accessed across the Activities, which meant that location variables could be accessed in the communications class to retrieve the latest position before sending a Drib or DribSubject.

The Map display had the ability to show, pan and zoom within a map. After locking ing the user's current position the map will display the where the user is, and will overlay dribble icons onto the map at the locations where DribSubjects within the user's Dribble radius were initiated. By selecting the dribble icon, the user can identify subjects and their locations. Every time the user retrieves new subjects from the server, the overlay items are updated. Threading was implemented into this activity through MapsThread, and runs for the duration of the program. This

class implements the Runnable and LocationListener interfaces to recheck if the user has changed location (every ten seconds for testing purposes), and if they had, calls relevant methods to retrieve the new co-ordinates and store them as static variables. It is noted that the location-finding interval would need to be increased so that resources are not unnecessarily tied down every ten seconds. If GPS is unavailable, an approximation is made based on the phone's relative position to cellphone towers.

The next tab is 'Topics', which when selected obtains a list of the most highly ranked DribSubjects in the user's Dribble radius. From this tab, the user can select a topic to retrieve the Dribs corresponding to that particular topic. This will automatically change to the Dribs tab. On the Dribs tab, a list of Dribs for the selected DribSubject is displayed in a ListView. Each item also shows a timestamp of when the topic was created and a DRank which is the ranking of that particular message. The user can like/dislike the message, by selecting the corresponding button, which will send the message and the changed 'likeCount' variable for processing on the server, where the DRank will be re-calculated. Lastly, the New tab allows for the creation of new Dribs and DribSubjects.

The content of each tab is refreshed when the tab is selected. This ensures that the user always the most up-to-date information. If the user changed location after selecting a topic, when the activity content is refreshed, only the information relevant to the user's new location will populate the view.

One key concept about the activities used in this application is that they are started at the time DribbleTabs is started. Activities are only stopped when explicitly told to or when the application ends. Every time an activity moves out of focus (hidden behind other activities), it is paused and resumed when it comes back into focus. These methods 'onPause' and 'onResume', were overridden in the source code to force the refreshing of the content and to ensure that Tabs were maintained. If a new activity is started every time the tab is clicked, the application will end up with multiple instances of the same activity running simultaneously. This causes memory problems and induces lag into the application run time.

6.1.2 Packages Android Packages (APKs) are added into the Android application to add functionality that has be been bundled together within a package. This enables APKs to be modularly included or removed. There were three packages implemented in Dribble: 'common.dribble' (contains the classes common to the mobile application and the server, and wrapper classes); 'dribble.dribbleapp' (contains the classes implementing mobile functionality) and finally 'XStream'. XStream is third-party API used to serialise and de-serialise objects (in this case a ListArray of objects) into XML. The 'common.dribble' package provided a common interface between the remote server and the mobile application. The classes 'Drib' and 'DribSubject' are instantiated when a user creates a message or message subject, respectively. These objects contain processing data such as names, latitude and longitude positions, relative to the message or subject the user is using.

6.2 Communications

Communications is made up of two parts: Web Services and XSTream. Both these aspects are utilised in the 'DribCom' class. In order to send an object over HTTP it needed to be serialized into the chosen communication scripting language, XML, using XStream. Two types of requests can be made using one of the three web services available. The HTTP connections are formulated with the RESTful web services on the server-side application. Firstly, the HTTP Put request can be used when the object sent is a new Drib or when a user likes/dislikes a specific Drib. In both instances the entire Drib needs to be sent to the server for processing. Therefore, XStream serialises the object list into XML and sends it with the HTTP request. In both cases, a simple message from the server is returned in the response. This can be used for simple error checking on the client side, to confirm the processing on the server side.

The second HTTP request is the HTTP Get, which uses the URL to send parameters with values. In the case of the 'getTopics' method, the latitude, longitude and number of results is sent to the server. As no object is sent, no serialisation is required. However, the client receives a list of objects (list of subjects or a list of messages). The stream returned in the HTTP response is

converted into a string and then into XML. From the XML it can be de-serialised into the object it represents.

During testing it was noticed that the serialisation and de-serialisation API used on the server and on the mobile side were not compatible. The XML root node was different which meant that the objects were not compatible after they had been sent or received. A wrapper class for each of the object lists ('DribSubjectList' and 'DribList') was created. The purpose was to ensure the XML root nodes conformed to a standard structure, set on the server-side. Compatible XML serialised objects could now be sent and received. This also introduced a new level of modularity between the serialization package and the communications class.

6.3 Resources

The resources involved in the presentation layer of the application can be decomposed into three subcategories: layout, values, drawable. Layout consists of XML files which contain the layouts (content views) for each of the activity classes. The XML files first specify the layout for the parent frame (within the tab layout) - in the application only Linear, Relative and TabHost layouts were used. Within the layout, text boxes, labels, icons, and all defining attributes are set. While they can be modified in the code, it was found to be better practise to set all relevant variations to default values.

One specific case of layouts, is the ListView, which requires a specific layout of list items to be dynamically created during run-time. In this case, two XML files are required, one for the overall ListView and one in which all the specifications for the list item can be defined. This allows buttons, several layers of text and icons to be laid out exactly how the developers require.

The values section contains pre-defined strings that can be called from the XML layout files. The final subcategory is drawable. The drawable resource contains all the images and icons that will be called from the XML.

Resources are connected to the Java Source code through the R.java class that is automatically created. This class links the modules, and therefore connects the presentation and the logic layers.

The last item of the presentation layer to be discussed is the splash screen which is displayed on start up. It is set as the start up activity for the Dribble application and will remain on the screen for a few seconds before it initiates the DribbleTabs activity.

7. TESTING

Eclipse comes with a standard debugger which was used through the implementation and testing phase. As mentioned above, there is also a session specific logging system, called LogCat. By embedding customised-logs into the source code, LogCat will display them at run-time. Session logs can be saved to track intensive processes or methods, or for debugging purposes.

Most testing completed during implementation was achieved on an Android emulator. This could achieve the same functionality as the phone, as the latitude and longitude could be manually set. The emulator created a virtual Android mobile environment, which could be debugged to locate and fix errors. Bug finding and fixing was also incorporated in the implementation process as a pair-programming approach taken to develop the mobile side. Exception handling was incorporated into the the source code to handle unexpected data types, especially for communications with the server.

The most vital part of the testing phases was the integration testing between the server and client. As mentioned above it was during the initial communication testing that the incompatibility between the serialisation API on the client and server sides was discovered and corrected. This type of testing was first performed on the emulator over a local server. Once the remote server was set up, the mobile phone was used and finally both the phone and emulated were testing simultaneously. All use-cases and possible aspects were tested individually to ensure all cases that could be present in the final system, including performance limitations and network reliability problems.

8. CRITICAL ANALYSIS

8.1 Modularity

An important concept in software development is the integration and use of modularity in code.

This technique is implemented in the design and source code, and requires that the system be decomposed into separate components. These components need to be functionally separated so that they can be added or removed without affecting one another. In the mobile application, modularity is achieved between the presentation and logic layers, as the XML layouts can be changed without affecting the Java source code. The presentation layer is scripted in XML and links to the Java source code (logic layer), through the use of the 'R.java' class. Further, modularity was implemented through the use of packages and the web services components. The cohesion of the application is relatively strong and the coupling is low, therefore indicating a high modularity within the application.

8.2 Maintainability

The maintainability of the application is limited in this case because of the new release of Android 2.1, on which the application has not been tested. However, relative to Android 1.5, the application was well constructed, readable, adequately commented and logged effectively. Limited error prevention is implemented including the restriction of characters in the length of the DribSubject and Drib text fields. More error detection could be implemented to restrict unwanted user inputs. For instance, one of the biggest potential bugs within the system is that the user is unable to enter punctuation due to SQL Injection in the database. From the client perspective, the 'New' Drib tab could process that type of issue before submitting it to the server. The use of public static variables of latitude and longitude which all the classes can access, is undesirable and it creates a weak point in the security of the application. However due to the limited knowledge of Android, no other adequate method was found.

8.3 Trade-offs

The speed of the mobile application is slower than desired due to the delay in serialising and deserialising the objects at the client and server ends. This was unavoidable due to the implementation of web-services. More research would need to be completed to make the application and communication process faster. A possible solution to this could be to only send the unique ID for the DribSubject or message, instead of the entire object. This would decrease the overhead on the network but might require the client to keep more memory.

8.4 Version Control

For this type of application with such a new environment it was decided that pair-programming would be the most ideal technique to employee. This meant that for the client application, both developers worked together on the same source code. Storage and backup of the application was handled by Dropbox, which has limited compatibility resolution functionality, but allowed both developers access to the latest version of the code at any time. Modifications and additions to the source code are monitored and recorded on Dropbox website.

9. RECOMMENDATIONS

One of the biggest challenges was to become familiar with Android and to note how it differed from pure Java programming. This caused numerous problems during development as extensive research needed to be conducted in order to achieve the desired functionality or affect. While the outcome was achieved, the unfamiliar platform means that there is no guarantee that the chosen method of implementation was the best choice or the most optimal in terms of speed, reliability or even best practise. Therefore it is recommendation that before the product is sent to market, more research is done into captilising on optimising techniques to improve speed and reliability.

One important, yet difficult design approach is to enable Dribble to be installed on different mobile phone models and on different operating system platforms. Cross-platform compatibility is becoming more vital in ensuring the lifetime of applications. Security is an ever-increasing concern for emerging technologies, and although Dribble does not handle personal details, the application must ensure that messages (with location details) can not be intercepted. Even though it is not vital within this application, if increased security was required, the information stored within the XML could simply be encrypted.

Exploration into the compatibility with Android 2.1 (new release) would need to be explored and

rigorously tested, as well as possible restructuring of the design to accommodate for the new functionality offered.

A better implementation would be to use a thread that refreshes content automatically in the background at certain intervals which would not require user input. However, a possible disadvantage to implementing this functionality is the added processing for the application which could slow it down and run the battery down.

9.1 Licensing

As mentioned in the group Dribble report, the Dribble application will be released under the BSD licence to allow for the developers to further build on the application and take it to market.

10. TIME MANAGEMENT

The time management allocation was constructed by firstly outlining the possible tasks that would need to be completed on the client side. Each of these tasks was then assigned a percentage of the estimated total time. The estimated total time per person was taken as the suggested 4.5 hours per member per week for seven weeks. As seen in the time sheet in Appendix D, the time for each task was underestimated and required almost 1.5 times the estimated value. As pair-programming was the chosen development process, the two mobile developers shared a time sheet, with research and small segments of code individually completed. Testing was not included in the breakdown as it was a continuous process incorporated into every task.

11. CONCLUSION

The purpose of this project was to design a software system that incorporated information storage, processing, calculating and visualisation. The chosen application Dribble, a geosocial networking mobile application, met the given requirements. Dribble was developed using the Android SDK, Java and XML in the eclipse IDE. While the designed application was able to produce the desired functionality, the application traded-off speed for the implementation of web services, and functionality over completed error handling. The modularity priority of the system was effectively achieved and sufficient testing and integration allowed the application to fulfil the success criteria stipulated by the developers.

REFERENCES

[1] F. Ableson. "Develop Android applications with Eclipse.", 2008. URL `http://www.ibm.com/developerworks/edu/os-dw-os-eclipse-android.html`.

[2] "What is Android?" URL `http://www.android.com/about/`. Last Accessed: 30 April 2010.

[3] "What is Android?" Last Accessed: 30 April 2010.

[4] T. Mickelsson. "Android Overview." In Idean, editor, *Senior Technology Specialist*. 2009. URL `http://mobiledevcamp.fi/`. Last Accessed: 1 May 2010.

[5] S. Conder. "Android: A Brief Introduction.", August 2008. URL `http://www.developer.com/open/article.php/3763991/Android-A-Brief-Introduction.htm`. Last Accessed: 1 May 2010.

[6] W3Schools. "XML." URL `http://www.w3schools.com/xml/default.asp`. Last Accessed: 1 May 2010.

[7] B. Dwolatzky. "Notes on Software Design Process." In *Information Engineering Research Programme*. University of Witwatersrand, School of Electrical & Information Engineering, 2.0 ed., February 2003.

[8] Developers. "Activity Class." *Android API*. URL `http://developer.android.com/reference/android/app/Activity.html`.

Appendix A: System Overview Diagram

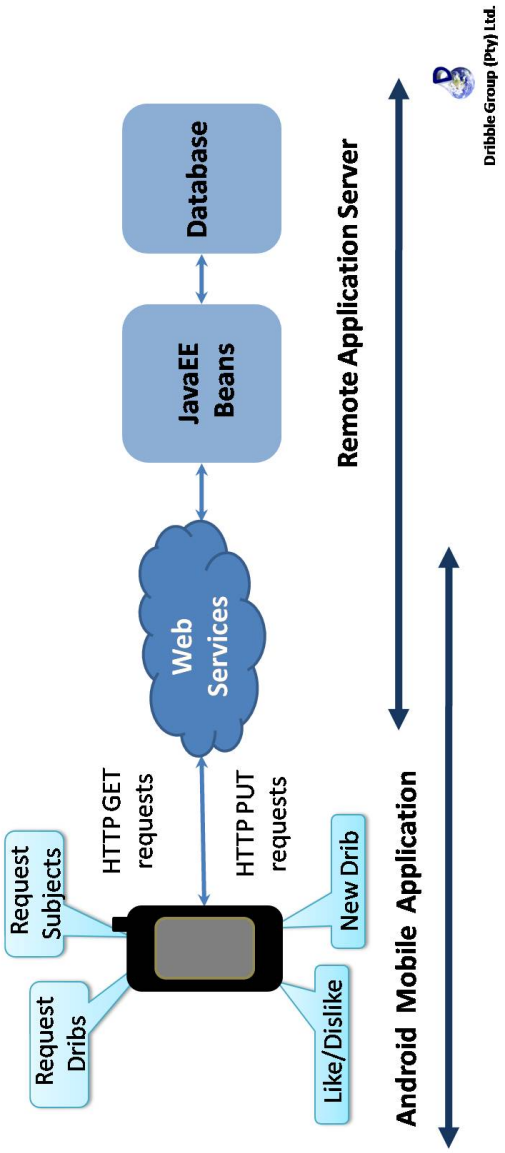

Figure 1 : Generalised System Model for Dribble

Appendix B: Mobile Application Generalised Model Diagram

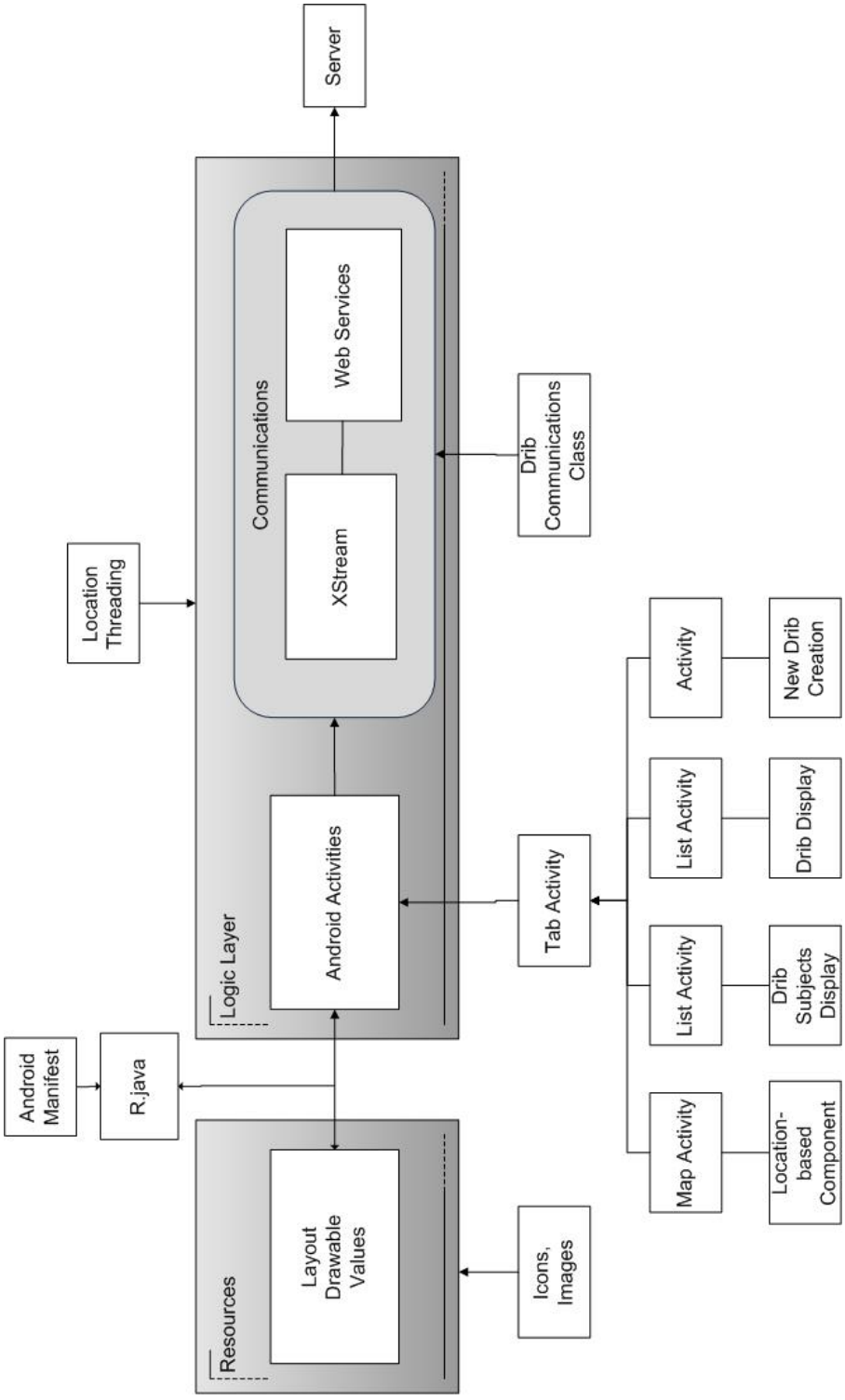

Figure 2 : Mobile Application Generalised Model Diagram

Appendix C: Screenshots

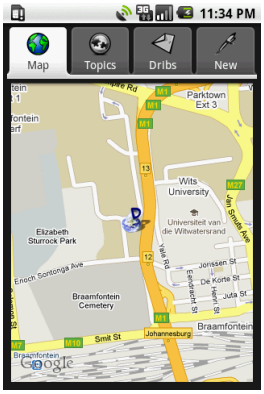

Figure 3 : Maps Tab

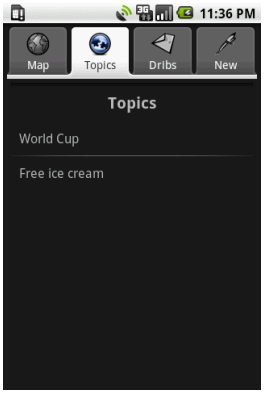

Figure 4 : Topics Tab

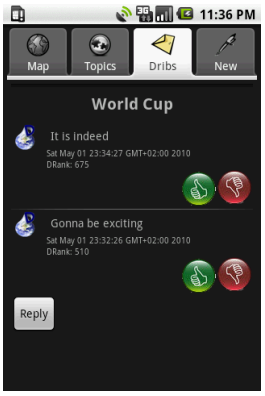

Figure 5 : Dribs Tab

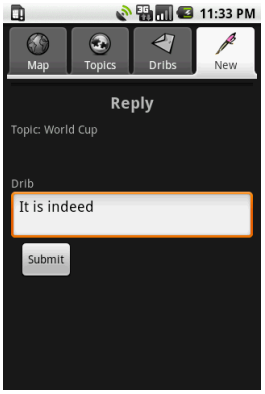

Figure 6 : Toics Tab

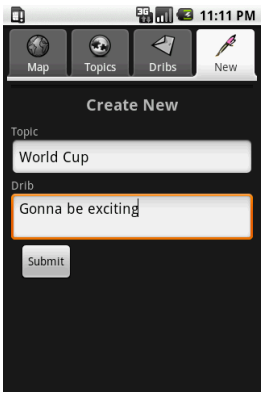

Figure 7 : Toics Tab

Appendix D: Project Timesheet

Table 1 : Showing the Time Breakdown with Tasks

Estimated Time (hours)	Actual Time (hours)
9.45	10
9.45	20
4.725	10
3.15	3
1.575	3
31.5	46

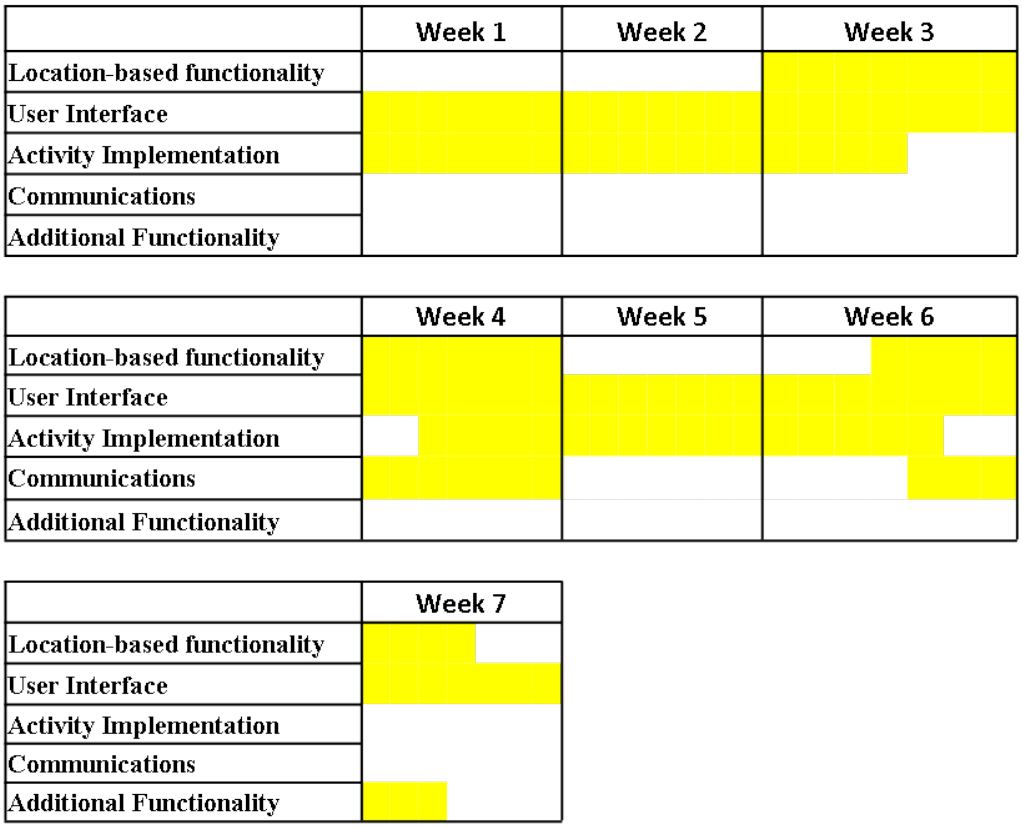

Figure 8 : Gantt Chart showing Aveagre distribution of Time to Tasks over 7 Weeks

Appendix E: Android Building Blocks

Below is a diagram depicting the major building blocks and components that are utilised in Android and Android application developement.

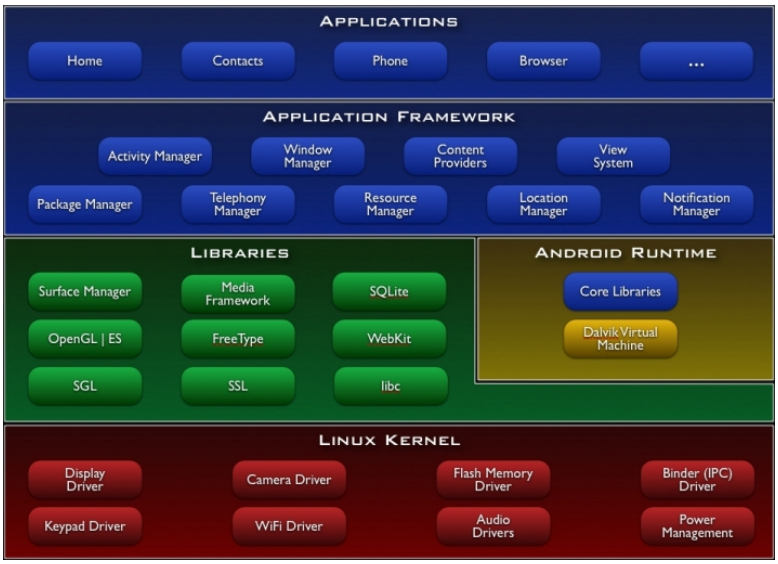

Figure 9 : Mobile Application Generalised Model Diagram

Appendix F: Use-case Model

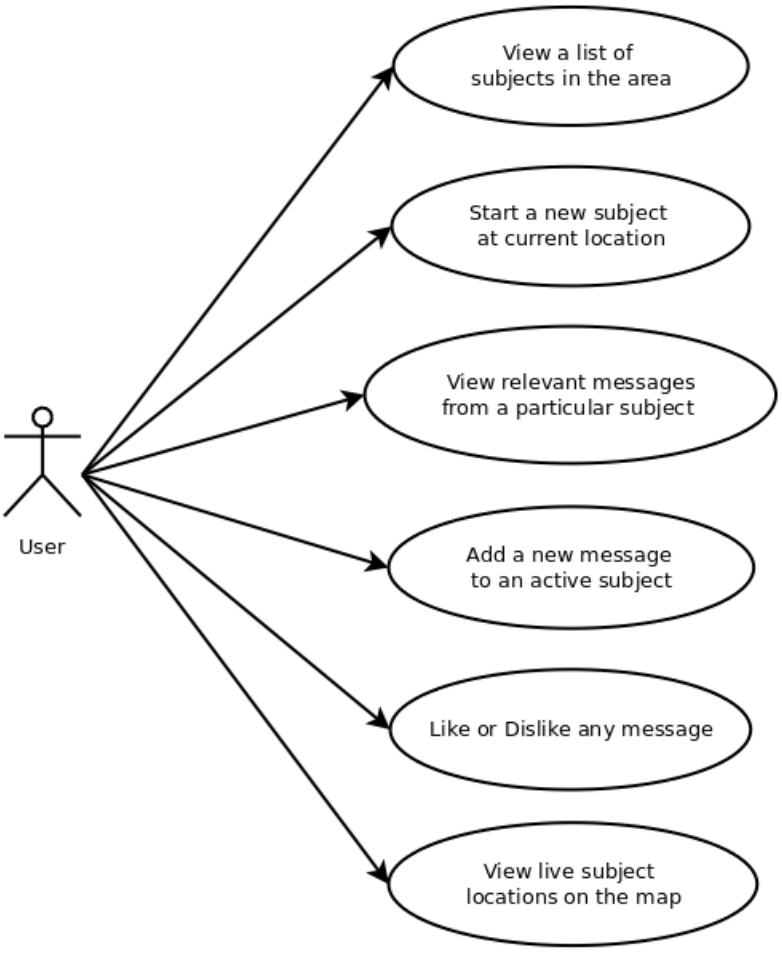

Figure 10 : Proposed Dribble System use-case Model