

SlideNoter

- Developing a notepad application for Android tablets

Kandidatarbete inom Data- och informationsteknik

Patrik Aldenvik

Robert Moberg

Axel Pelling

Jonas Scholander

Institutionen för Data- och informationsteknik

CHALMERS TEKNISKA HÖGSKOLA

Göteborg, Sverige 2012

Kandidatarbete/rapport nr 2012:024

Abstract

This report presents the process and thoughts behind the development of an open source Android application; an application for reading PDF-files and adding notes upon them. The main purpose is to provide an alternative for students, enabling them to take notes during lectures without pen and paper.

The report covers the basic building blocks of the Android graphical user interface, blocks that the application use, as well as the integrated PDF-reader, VuDroid. As a software engineering process called Spiral Model was applied to the project, a chapter brings up how this affected work. Information regarding the implementation choices and a few problems that arose during the coding process as well as the solutions to them can also be found within.

The result is an application that can read PDF-files, present them for the user and have the ability to add a layer with text and drawings. The resulting application and work process is evaluated at the end of the project report.

Sammanfattning

Den här rapporten presenterar processen och tankegångarna kring utvecklingen av en Androidapplikation med öppen källkod; en applikation för att läsa PDF-filer och lägga till anteckningar på dem. Huvudsyftet är att ge studenter ett alternativ till penna och papper.

Rapporten omfattar de grundläggande byggstenarna i Androids grafiska användargränssnitt, byggstenar som applikationen använder, såväl som den integrerade PDF-läsaren, VuDroid. Eftersom en programutvecklingsmetodik som kallas Spiralmetoden tillämpades i projektet, tilldelas detta ett kapitel i rapporten. Information om implementationens val och ett par problem som uppstod under kodningsprocessen, såväl som lösningar till dessa problem finns också att hitta i rapporten.

Resultatet är en applikation som kan läsa PDF-filer, visa dem för användaren och ge möjligheten att lägga till ett lager med anteckningar. Den resulterande applikationen och arbetsprocessen evaluerades i slutet av projektrapporten.

Table of Contents

Glossary.....	1
1. Introduction	3
1.1. Background	3
1.2. Purpose of the report.....	4
1.3. Limitations	4
2. Theory - Android, licenses and the Spiral Model	5
2.1. Android Building Blocks	5
2.1.1. Activity	5
2.1.2. Intent.....	6
2.2. Graphical components within Android.....	6
2.2.1. Bitmap.....	6
2.2.2. Canvas.....	6
2.2.3. Drawing Primitives	7
2.2.4. Paint.....	7
2.3. The View Package.....	7
2.3.1. ViewGroup	7
2.3.2. How Android Draws Views	8
2.3.3. SurfaceView.....	8
2.4. Open Source and Licenses.....	8
2.4.1. Copyright and Copyleft.....	8
2.4.2. GNU GPLv3	9
2.5. The Spiral Model	10
2.5.1. The Four Phases.....	11
2.6. Version Control System	11
3. Development methods and processes	12
3.1. Choice of Software Engineering Process.....	12
3.2. Choice of Platform and OS	12
3.3. Choice of Programming Language	13
3.4. Specification	13
3.5. PDF-reader.....	13
3.6. Version Control System	14
3.7. Testing	14

3.8. How to satisfy the requirements.....	14
3.8.1. User Testing.....	15
4. Implementation	16
4.1. The Program Structure	16
4.1.1. VuDroid.....	16
4.1.2. The Whiteboard.....	16
4.1.3. The SlideNoter Application.....	16
4.2. Design of GUI.....	17
4.3. Scroller	17
4.3.1. Scrolling the Whiteboard	18
4.3.2. Scrolling both Views at the same time.....	18
4.4. Implementing textboxes.....	18
4.4.1. EditText.....	18
4.4.2. Canvas.drawText	19
4.4.3. Text input	19
4.4.4. Editing text	20
4.5. Other issues	20
4.5.1. Showing both Views at once	20
4.5.2. Layout design.....	20
4.5.3. Changing between modes.....	21
5. Results.....	22
5.1. The Application.....	22
5.1.1. The Main Menu	22
5.1.2. The Browser.....	23
5.1.3. The PDF-Viewer.....	24
5.1.4. The Whiteboard.....	25
5.2. Fulfillment of the functional requirements.....	26
5.3. Fulfillment of the non-functional requirements.....	26
5.4. User tests	26
6. Discussion	27
6.1. Evaluation of the Specification.....	27
6.2. Evaluation of major decisions	27
6.2.1. Working with the Spiral Model.....	27

6.2.2. Android as a Development Platform	27
6.2.3. SurfaceView as Drawing Area.....	28
6.2.4. VuDroid as the PDF-reader	28
6.2.5. The Usage of a Version Control System	28
6.3. Possible other uses of the application.....	29
7. Conclusion	30
7.1. Critical discussion.....	30
7.2. Future work	30
7.2.1. Fixing memory leak	30
7.2.2. Support for more file extensions.....	31
7.2.3. The ability to search for notes	31
7.2.4. Cloud based storage and multiple user editing.....	31
7.2.5. Word lookup.....	31
Bibliography	32
Appendixes	34
Appendix A.....	34
Iteration 1 - PDF-reader	34
Iteration 2 - Whiteboard.....	34
Iteration 3 - SlideNoter	35
Iteration 4 - Continued Developments	35
Appendix B - UML.....	37
Appendix C - List of Contributions	39
C.1. Fields of responsibility.....	39
C.2. Presentations.....	39

Glossary

Abstract function - Unimplemented function, any implementation of the class containing an abstract method must implement it.

Alpha Testing - A very early test of the product with basic functionality designed to locate and help flatten out the most apparent bugs. It also give the users a chance to make developers aware of certain functions that should be changed or integrated.

Android - An Operating System developed by Google for mobile devices.

Android SDK - Android Software Development Kit. Tools to develop Android applications.

API - Application Programming Interface. A set of instructions how to use a class or a library.

Beta Testing - A test later in the process where more people are invited to help locate the more odd bugs that can be overlooked in the alpha testing.

Class - A collection of methods with a specific purpose. For example provide a User Interface.

DMCA - Digital Millennium Copyright Act, a law in the US that make DRM breaking illegal.

Drawn Objects - All the lines, text boxes and highlighting the users has done.

DRM - Digital Rights Management, a combination of software protection designed to control distribution and spreading of copies.

GUI - Graphical User Interface. A graphical UI. Allows interaction between humans and machines on a graphical level, with images for example.

I/O - Input / Output. Refers to the inputs into the system a user makes and the systems response/output.

iOS - formerly known as iPhone Operating System, it is now the name of the operating system that apple use in their products.

Java - An object oriented programming language developed by Sun and is maintained by Oracle.

MotionEvent - Object used to report movement. Including mouse, pen, finger and trackball events.

MyHighlight - Help class to save coordinates and paint colour.

MyText - Help class to save coordinates, text string and paint colour.

MyPoint - Help class to save coordinates and paint colour.

OOP - Object Oriented Programming. A programming style where you treat parts of your codes like different objects.

OS - Operating System. The system that directs the operations of a computer and take user input to do so. For example Android, Windows, OS X & Linux distributions.

Parent - A parent node in a tree structure refers to a node in the structure that has a child.

Superclass - A class that provides its properties and methods to its subclasses.

PDF - Portable Document Format. A file format that is widely used for documents.

PPT - PowerPoint. A file format made by Microsoft that is used for creating slide shows.

Revision - A collection of files on SVN connected to a specific number.

Subclass - A class that inherits properties and methods from its parent class.

SVN - Subversion. A Version Control System.

Top-down traversal - Traverse through a tree structure starting with the root and work down towards the children.

Tree Structure - A tree structure is a way of representing the hierarchical nature of a structure in graphical form.

Throwaway prototyping - A development method based on fast creating a prototype, find flaws and see if it meets the requirements. Create a new prototype without the found flaws and repeat.

UI - User Interface. The system where interaction between humans and machines occurs.

VCS - Version Control System, also known as reversion control. A way for multiple programmers to code on a single project without the risk of overwriting the work of other programmers.

View - A class used in Android OS for UI structures and components.

XML - eXtensible Markup Language. A markup language used to save data in a text based format.

Z-ordered - Orders two-dimensional objects by the Z axis. Allows a programmer to choose which components in a UI should appear on top of another.

1. Introduction

When the first Tablets was released everyone wanted one. If you asked them the question “What are you going to use it for?” the answer was almost always the same. It was along the lines of “I do not know, but I want one”. Since the first release the answer has not really changed much; a tablet offers just about the same functionality as a smart phone with the exception of a bigger screen. This report will document the development of an application that will be a step in changing the answer to “I want one because...”.

1.1. Background

One answer to the aforementioned question could be to aid you in your education. Within higher education, many lecturers use projectors and PowerPoint as a tool in their presentations. Slides are often shown picturing what is discussed during the lecture. These slides are usually uploaded to the Internet before the class, thus providing students the ability to view and study them beforehand. The student can then choose to print the slides and bring them to the lecture and write additional notes on them. At the moment there is a very limited availability of electronic options to the pen and paper when it comes to writing notes on these slides.

To tackle this problem an idea was formulated to develop an application for Android. An application that could be used by the attendants at a lecture. With this application they would be able to open PDF-files and scribble notes on them. The users would be able to draw lines and add simple figures as well as type additional text and highlight existing text. When the edits have been made the application should have the ability to save the changes, enabling viewing at a later time. In order to be appealing the application had to focus on a friendly user interface and a wide variety of useful functions.

To ensure that everyone who would like to use the application can use it and everyone who wants to participate in the development can participate, the application will be licensed under a open source licence. As of today there are similar applications (Google Play, 2012a,b,c) that already tackles this problem, yet none which is to licensed under an open source licence.

1.2. Purpose of the report

This report serves as documentation for future developers, documentation which they can read to familiarize themselves with the project; a project to develop an application for Android tablets where you can write notes on PDF-files. The information which is covered includes why some decisions were made, how they were executed, the results and discussion about the results and certain choices.

1.3. Limitations

Tablets were chosen as the platform for development. The reason for this was the benefit of them having larger screens than smartphones. Due to the relatively small screen of smartphones they are not optimal for reading large pages of texts, nor for writing notes or drawing pictures.

Due to the limited timeframe some of the possible features of the application will not be implemented during the course of this project. Hence, one limitation is that PDF-files and not PowerPoint files should be usable for this project. Another reason for this decision was that PDF-format is an open standard (Adobe Systems Incorporated, 2006) whereas PPT is a Microsoft license (Microsoft, 2012a). The implementation of support for PPT files among other features may however be implemented in future releases of the application.

2. Theory - Android, licenses and the Spiral Model

This chapter describes Android packages and classes that are important building blocks in the application. The packages and classes are native to Android and the understanding of them is vital for the system design. In addition to the information about Android classes, there is sections about other tools used in the project: free software and their licences, the Spiral Model, and Version Control Systems.

2.1. Android Building Blocks

In the following sections basic objects of the Android SDK will be described. An Android developer needs to be familiar with these building blocks to understand and develop applications. The first of these building blocks is the Activity.

2.1.1. Activity

Every time a user interacts with something on an Android device there is an Activity to take care of the interaction. With few exceptions an Activity is always interacting with a user in one way or another (Android, 2012b).

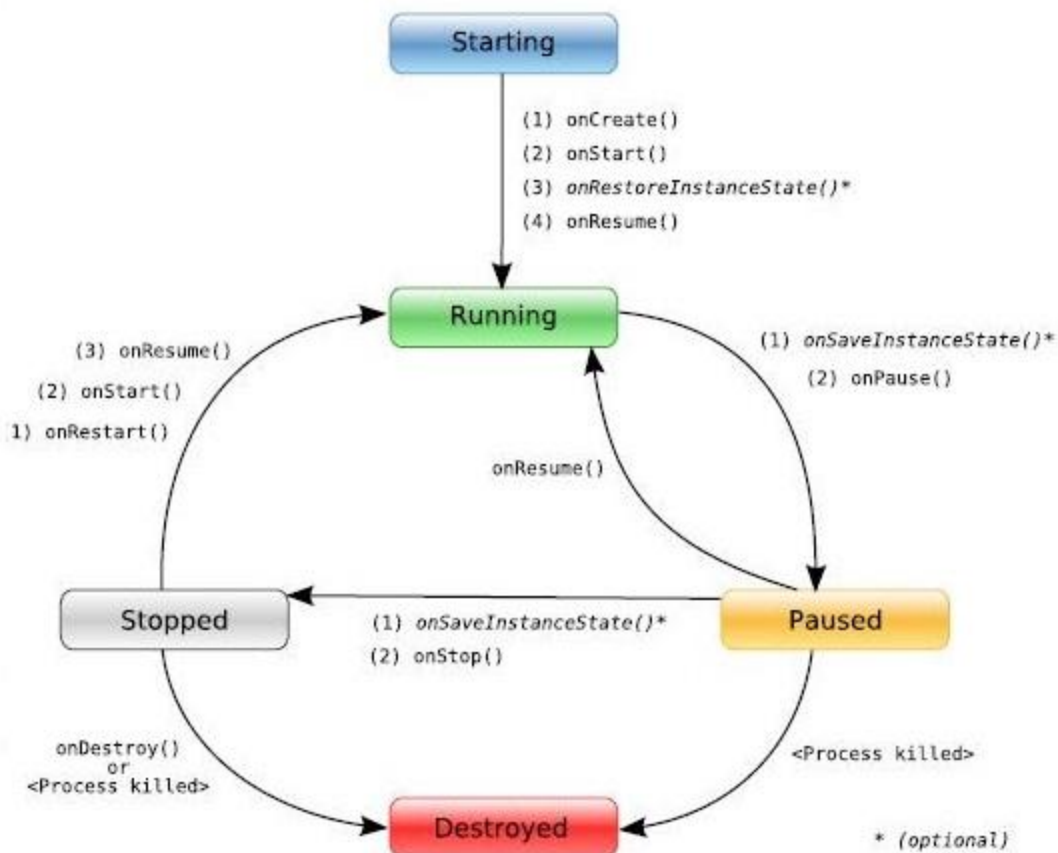


Figure 1. Lifecycle of an Android activity (AndroidTics, 2012)

When a user starts an application in Android, Android starts it and brings it to the foreground so that the user is able to interact with it (Burnette, 2010). An application usually has multiple Activities plus a Process which acts like a container for them. Each Activity has its own life cycle, illustrated in figure 1.

Beside the methods an Activity has to perform its purpose, it should also include the methods described in figure 1. These methods are in charge of what will happen when an application starts, is brought to the background, when it is resumed, and when the program is closed or when Android kills it to reuse the resources the application uses.

2.1.2. Intent

Before the start of a new activity, there is an Intent which starts it. Brunette describes an intent as “*a mechanism for describing a specific action*” (Brunette, 2010). Some examples of an intent could be to: “send an email”, “take a photo” or “make a call”. The Intent is the part of an application that stores information to be forwarded when a new activity should be started. The intent also tell the application to start the activity.

2.2. Graphical components within Android

Within Android graphics there are four basic components for drawing objects, a Bitmap, a Canvas, a drawing primitive and a Paint. All four have to be present to draw something on the screen (Android, 2012c). These can all be found within the android.graphics package. The section is based on information from the Android Developers documentation (Android, 2012c-f).

2.2.1. Bitmap

To enable an application to draw something on the screen, a Bitmap is required. A Bitmap holds information on all the pixels that is rendered on the screen. The Bitmap has no native drawing functions. Due to this, some other class is required to tell the Bitmap what pixels to store. A Canvas is such a class.

2.2.2. Canvas

When an application's Canvas is first constructed, a Bitmap has to be specified before anything can be done. When the Bitmap is set the Canvas can begin its work. The work it performs is managing all calls to draw something. When a function should draw something on the screen all calls go via the Canvas. The Canvas in turn forwards the information to the Bitmap that store the pixels.

2.2.3. Drawing Primitives

The Canvas cannot simply draw anything, what it can draw must contain a drawing primitive. Android supplies a range of predefined classes of drawing primitives. Two examples of drawing primitives are a previously saved Bitmap and a Path. Unlike the Bitmap described in 2.2.1. a Path only contains information about the position of the lines and padding.

2.2.4. Paint

The drawing primitive does not contain colour information, styling, size and similar properties. A fourth class is thus required to keep track of this information, which is called Paint. If an application should have multiple drawable objects in multiple colours and size, one Paint object must be created per drawable object, else all objects share the same properties.

2.3. The View Package

In order to combine the graphical components into a User Interface the Android.view package is required. This package includes methods to create and handle the layout of any Android Application (Android, 2012g). The package contains dozens of classes which can be used to create many types of UI's. This sub-chapter will explain a few classes of the package which are vital for this project. The section is based on information from the Android Developers documentation (Android, 2012e, g - k).

2.3.1. ViewGroup

Within the View package there is a class called ViewGroup. A ViewGroup is exactly what the name implies, a group of Views. A ViewGroup is Z-ordered, the Z value of the View is used to determine what Views should be on top. The Views in a ViewGroup are called children. It is the parent class for layout and View containers. It also defines the ViewGroup.LayoutParams class which is used by Views to tell their parents how they want to be laid out.

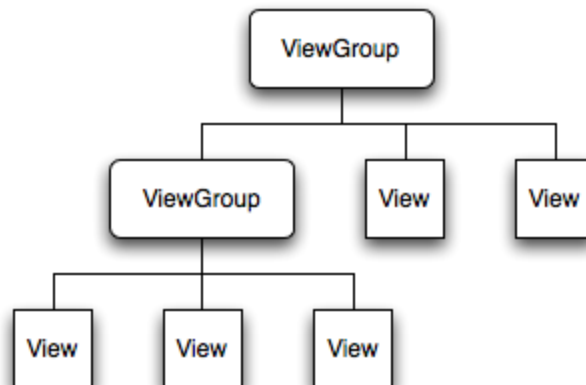


Figure 2. A ViewGroup tree (Android, 2012j)

2.3.2. How Android Draws Views

When an Activity receives focus from an Application, it will receive a request to draw its layout. Every Activity has its own layout hierarchy, usually defined in an XML layout file, which is used by the Android framework that handles the drawing procedure. The Activity provides the root of its layout hierarchy where the drawing begins. The tree is then traversed to render each View that intersects the region in which to draw. Each ViewGroup is responsible for requesting its children to be drawn and each View is responsible for drawing itself.

It is a process consisting of two cycles, where both of these are top-down traversals. The first cycle is to collect measurements for the Views. During the second cycle the parents are responsible for positioning their children according to the sizes received from the first cycle

2.3.3. SurfaceView

SurfaceView is a subclass to View, it provides a dedicated drawing surface within an existing View hierarchy. The idea is that the surface will be “behind” the existing window and when something is drawn on it, the SurfaceView will “punch a hole” in the window to display its content. Interaction with a SurfaceView is not direct, instead the class uses an interface to handle the surface, called SurfaceHolder which can be acquired by using the getHolder() method. The class has implementable methods to detect when the surface is created and destroyed.

2.4. Open Source and Licenses

The following section describes the difference between Copyright and Copyleft and the GNU GPL v3 license.

2.4.1. Copyright and Copyleft

The term copyright is defined at Dictionary.com as: “*the exclusive right to make copies, license, and otherwise exploit a literary, musical, or artistic work, whether printed, audio, video, etc.*” (Dictionary.com, 2012). Copyright is given to a person by default on anything the person produces, be it text, music or video. As soon as text has been written by a person, that person becomes the author and receives copyright for it. The text now belongs to the author and he or she has the sole rights and may do whatever with it.

Copyleft is a general method for making a copyrighted work free, and requiring all modified and extended versions of the program to be free as well (GNU, 2012b). What this means is that copyleft is a way for the author to retain the copyright, whilst still releasing the rights to make alterations, copies and re-distribute the work. This release can be achieved in one of

two ways: the work can be released into the “Public Domain” directly. However, in some countries it is not possible to release the rights to a work until after death of the author. In these countries the work must be licensed using a copyleft license. In Sweden for example, an author holds the right to his or her work until 70 years post mortem. In order to release the copyrights prior to 70 years post death in Sweden the work therefore has to be copylefted (Lagen.nu, 2011).

A copyleft license is based on the four freedoms of free software (GNU, 2012a). The user is free to run the program for whatever purpose, and to modify the code to suit a specific need and to re-distribute the code for others with similar needs. Furthermore, the user is free to base completely different software that utilizes parts of the code from the first. The only requirement of a copyleft license, is for the re-distributions to be under the same license (GNU, 2012b).

2.4.2. GNU GPLv3

There are several different copyleft licenses that a developer can use. Among them, the most widely used one is called GNU GPLv3, which is written by GNU and The Free Software Foundation. As mentioned it is the most commonly used licenses to distribute free software under. For example around 70% of the projects released on sourceforge.net are licensed with one of the different GPL versions. (Sourceforge, 2012)

GNU was founded in 1984. Their goal was to release an operating system with similar functions to UNIX, with the exception that it was to be based solely on free software. The organization coded much of the functions themselves. They saw the need for a unique license that everything would be released under so that the work would stay free in the future. In February of 1989 the first version of GPL was released (GNU, 2012c), and unlike for example BSD, GPL required that any further releases would also have to include the same license, and thus the four freedoms.

2.5. The Spiral Model

The Spiral Model is a software development process defined by Barry Boehm in 1986 (Sommerville, 2011). The Spiral Model consists of several iterations, each divided in four phases with incremental releases of the product, as seen in figure 3. The model was originally intended for large, expensive and complicated projects.

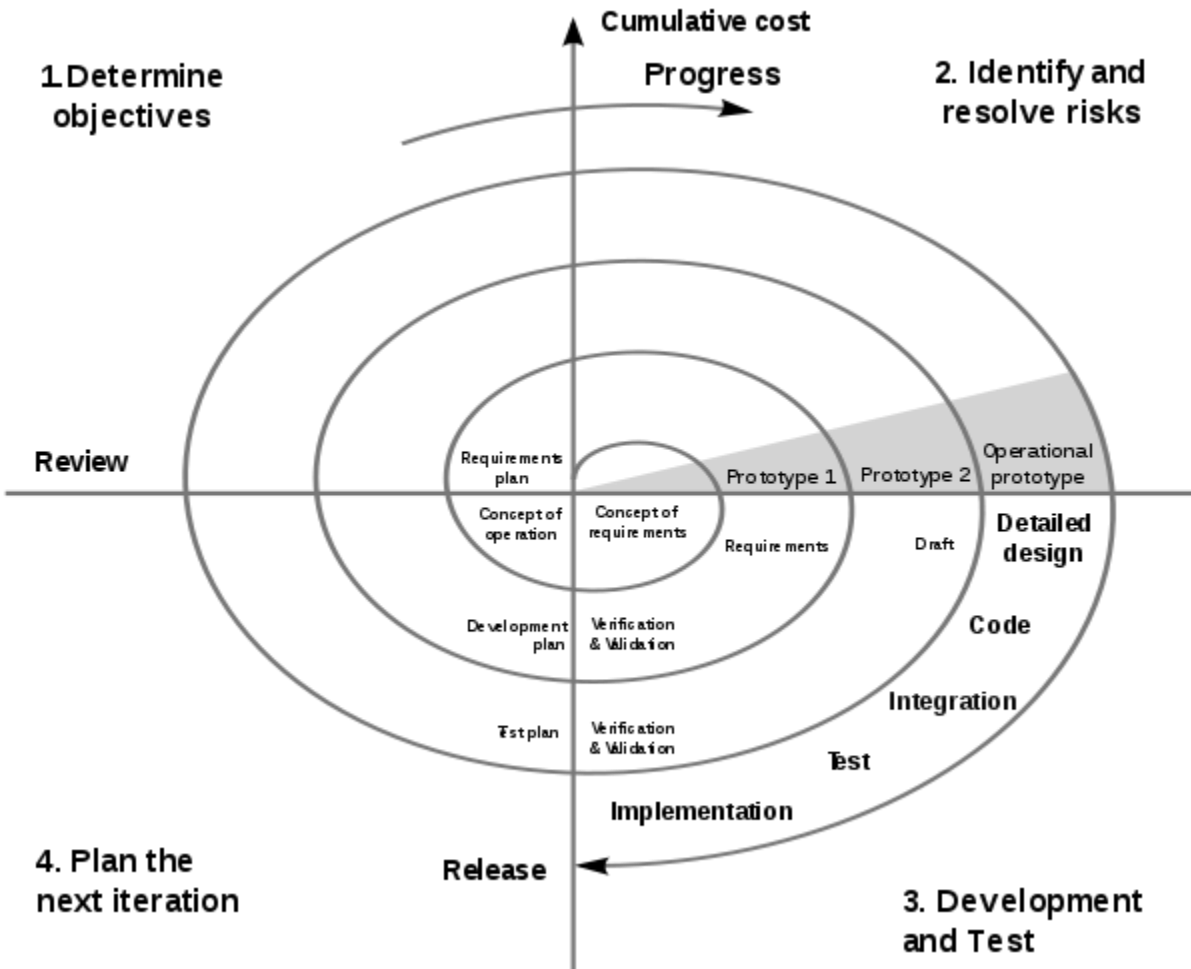


Figure 3. An illustration of the Spiral Model (Wikimedia, 2008)

2.5.1. The Four Phases

Sommerville describes that during the first phase the objectives of the current iteration is decided.(Sommerville, 2011) The objective constraints on product and process are also identified, thus enabling a detailed management plan to be outlined. When the plan is created the last part of this phase is conducted. A risk analysis identifies problems that may occur and depending on the nature of these, alternative strategies are planned.

When the second phase starts, the risks identified in the first phase are analyzed in detail. After the analyzing stage, steps are taken to reduce the risks. The third phase is when the actual development and validation is done. However, before any development is done a development model for the current iteration is chosen. The approach for the development model varies depending on the nature of the project.

When the final phase is reached the project is reviewed. This requires a decision to be made; whether to continue with another iteration or not. If another iteration is to be made, plans are drawn up and the project re-enters the first phase once again (Sommerville, 2011).

2.6. Version Control System

A VCS is a tool which enables multiple users to work on the same project without overwriting another user's code contribution. The users commit the files they are working on while also being able to receive updated versions of parts of the project that others in the group are working on. If the users have edited the same file then Subversion will try to merge the files. If conflicts in the file occur on a text row based level, meaning the users have made two different changes to the row, then the user will be prompted to decide if the line of code located on the server, or the one he is trying to commit is correct (Chalmers Insidan, 2012).

3. Development methods and processes

This chapter describes how the project was executed. It explains what software engineering process was used during the project, how it works and why it was decided to use this process. This chapter will also present a few of the challenges of the project and describe how they were dealt with.

3.1. Choice of Software Engineering Process

Four major development phases were identified during the planning phase of the project. In the first phase a PDF-reader application was to be found and implemented. The second phase would consist of creating a basic paint application. The third phase would be to integrate and combine the results of the previous two phases. The fourth and final phase would be to implement more functionality and improve the GUI.

Each phase would require the completion of the previous phase before the new phase could start. A development process which could deal with incremental steps was required. Based on this information a decision was made to use the Spiral Model for the project. There were many advantages using the Spiral Model for the project. Due to the linearity of the project the phases needed to be executed in a specific order. As the Spiral Model is an iteration through four phases, it was a suitable choice for the project. Even if the Spiral Model initially was intended for projects larger than this, it provided the suitable tools for the organisation of the project. These tools do not have the disadvantages of elaborate documentation throughout the process, as for example the *Waterfall Model* (Waterfall Model, 2012).

During the third phase, the development phase, a development approach called *throwaway prototyping* was used. This approach was used since it is suited for projects where user interface risks are dominant (Sommerville, 2011).

3.2. Choice of Platform and OS

When designing an application for a tablet there are three larger platforms to choose between. These platforms are: Android, iOS, Windows. All platforms have their pros and cons. One good thing about Android is that is a more open platform compared to the others. In order to release products for iOS or Windows the applications needs to be approved by Apple or Microsoft before they are published. Applications for Android does not need to be approved in this way. Android is also supported by an increasing number of brands, while Apple is the sole producer of products with iOS. Windows phones are not as widespread as the other two at the moment.

One disadvantage with Android is that it is developed by a lot of different companies. This makes most Android tablets and phones look slightly different and have different hardware, which can impact the performance of some applications.

The programming language that Android applications are written in is Java, one of the world's most used programming languages (Langpop, 2012). Due to this, a lot of documentation and solutions to simple problems exist on the internet.

Taking these two pros into consideration, along with the fact that several project members already had some experience developing applications for Android, the decision was taken to develop for Android.

3.3. Choice of Programming Language

It is worth mentioning that there are other languages than Java that can be used on Android. The support for these relies on third party software and require additional software to either be included in the release or already present on the device. However, since the group was used to Java and the fact that it is the native language of Android, no other language was considered.

3.4. Specification

At the start of each iteration of the project a specification was created. If there were ideas which could not be implemented in the current iteration or if new ideas were formulated, they would be added to the draft of the next iterations specification. The first two iterations through the Spiral Model were sub-projects with their own requirements. the third an integration of the two sub-projects and the fourth was additional features. During the merge all previous requirements had to be met, while also following the specification for the requirements of the combined application. The specification with a complete list of functional and non-functional goals can be found in Appendix A.

3.5. PDF-reader

After the initial research, a decision was made that building a PDF-reader from scratch would not be possible, having the timeframe of the project in mind, as it would be a large project in itself. Instead of writing the PDF-reader it was decided to make use of one of the advantages of *Free Software*, the freedom to modify existing code for use in other projects.

Research was conducted about using an existing PDF-reader and the possibility of porting it for use on Android. As Android applications are written in Java, the focus was on finding PDF-reader projects written in Java. One of the projects found was PDF-renderer (PDF-renderer 2012). PDF-renderer was a simple PDF-reader written in Java.

Another project that was considered was JPedal (JPedal 2012). This project was interesting because it allowed you to convert PDF to images, which could potentially be a useful feature. After looking through the code however, it was decided that porting a pure javaproject to Android would nearly be as time consuming as writing a reader from scratch.

A new decision was made to find a project that was already available on Android and further build upon that. The first one was a port of the PDF-renderer project, which had been investigated earlier. Unfortunately this implementation was last updated in 2009 (Ferenc Hechler 2012). It basically did what was needed, however it was too slow and the user interface was not very appealing.

Other projects were investigated and it was decided that VuDroid (VuDroid 2012) met the requirements. The renderer was relatively fast and UI was slick and easy to use. First and foremost it worked well with tablets, which was vital. Secondly VuDroid was licensed under GNU GPLv3, which means it fell under the category of *free software* mentioned in section 2.4.

3.6. Version Control System

To be able to manage using several work-stations when writing the code for the program a VCS called Subversion was used. Subversion is a useful tool which enables programmers to commit revisions of the project to a remote server. The user can then log onto another work-station, connect to the server and download an updated revision of the project.

There are multiple choices of VCS's, the system used in this project was SVN. To this, a shell extension for windows called TortoiseSVN was used. This was used for a simple reason: Chalmers supplied a repository, and the guide on "*Chalmers Insidan*" suggested TortoiseSVN for Windows.

3.7. Testing

When developing the application, the code was continuously tested for errors and bugs. Finding the bugs at an early stage would consequently reduce the risk of them cascading and causing more errors further down the line. At the start and middle of the project the functions were tested individually while they were coded.

3.8. How to satisfy the requirements

At the end of each cycle of the project process the current implementation of the application was put through a Quality Assurance process. This process consisted of testing the application and going through all the specifications to verify that the specification had been followed. Was it to be found that some part of the application did not follow the

specification a decision had to be made; whether the application should have been changed so that it would have followed the specification, or if a change in the specification would produce a better end result.

3.8.1. User Testing

Project plans were made to conduct user testing. The user testing would consist of sending out a form to selected test users, along with a public link on facebook. A basic form with information on how to download and install the application was created to aid the user testing. Over the course of the project this form was consistently updated with instructions and questions concerning the implemented functions.

4. Implementation

This chapter describes the most important parts of the application and why certain choices were made.

4.1. The Program Structure

The program consists of two integrated parts. The already existing VuDroid project and the WhiteBoard which is specifically developed for this project. A visual representation of the class diagrams can be found in Appendix C.

4.1.1. VuDroid

The VuDroid project contained features which would not be used in SlideNoter. To minimize the code and remove the risk of errors by unimplemented features VuDroid was scaled down to fit the use of the application. The smaller version consists of eight packages. The seven core packages contain the main components required to run the program. The last package consists of subclasses of the core's codec package. Its purpose is to create a class for the PDF, a class that has all the necessary information and variables, for example how many pages a PDF-file contains. This information is then used to decode and render a PDF document.

The classes that do the actual decoding had to be modified in order work together with the WhiteBoard parts of the application. These modified classes were moved into the WhiteBoard packages.

4.1.2. The Whiteboard

The slidenoter.whiteboard package contains the classes used for creating, rendering and editing the notes. The main class of the package is WhiteboardView, which is a subclass to SurfaceView. It uses the help classes MyText, MyHighlight and MyPoint to create and maintain the notes that the user has written. When implemented, the WhiteBoard is a transparent layer that is put on top of another application.

4.1.3. The SlideNoter Application

The SlideNoter application consists of a core package that handle UI, a package that contains the Whiteboard area, a package that holds the modified VuDroid packages and the remaining VuDroid packages that are left unchanged.

The slidenoter.core package contains the classes to start and operate the application. These classes contain the main menu, the file browser and the PDF-viewer with the ability to scribble notes. To be able to decode and render a PDF-file the slidenoter.vudroid

package is used. It is in this package that the classes which originate from the VuDroid project are located. The third package is called `slidenoter.whiteboard`, which is described in 4.1.2.

4.2. Design of GUI

A minimalistic GUI design was chosen for the application. The UI is a straightforward approach, with the idea that the user should not be able to missclick. Since the point of the application is to create notes on a PDF document most of the screen should be used to show the PDF. An in depth look at the UI can be found in chapter 5.1.

4.3. Scroller

In order to be able to navigate files that are large enough to not fit on one page, a scroll function is required. This was implemented by enabling a scroll mode when none of the drawing modes are activated. This scrolling function moves the drawings only along the y-axis. Scrolling is accomplished with help of a method that is implemented in the `WhiteboardView` class called `onTouchEvent()`. This method is inherited from `SurfaceView`.

Each time the user touches the screen on the device and it is in a region of the `WhiteboardView`, a `MotionEvent` is generated and sent to the `onTouchEvent` method belonging to the `WhiteboardView`. From a `MotionEvent` it is possible to get the information about where the event occurred, what kind of action was made, among others. To get the scrolling working, the two aforementioned parts of information are essential. There are three different `MotionEvent`s used in the scroller `ACTION_DOWN`, `ACTION_MOVE` and `ACTION_UP`.

- `ACTION_DOWN` is when the user makes the first contact with the screen.
- `ACTION_MOVE` is when the finger is dragged along the screen.
- `ACTION_UP` is when the user removes the finger from the screen.

To get the scroll to work, certain methods had to be called upon depending on the action. When an `ACTION_DOWN` action occurred, the coordinates of that event was saved to a variable. An `ACTION_MOVE` then continuously calculates the offset from the last known position and redraws the `Whiteboard` with this new offset. The move action also overwrites last known position. When the user lifts the finger from the screen, the `ACTION_UP` is invoked, and a final redraw of the screen is made. This is done to ensure that the application has the very last touch event as a saved state.

4.3.1. Scrolling the Whiteboard

During the implementation of the scrolling function two related problems occurred. Firstly, when a user scrolled the drawn objects, the position of the objects changed, yet they were never rendered on the canvas. The solution was to send the drawn objects to the canvas to be rendered every time they changed position. Secondly; an artifact originating from the solution to the first problem. While scrolling the View, all the drawn objects left a path of *copies* after itself. What had happened was that the scroll function painted the drawn objects on the canvas every time they were moved, without removing the existing objects. To solve this problem the canvas was cleared before rendering the moved object.

4.3.2. Scrolling both Views at the same time

The two parts of the application, drawing and viewing PDF, were separate from each other. When the time came for putting the Views on top of each other a problem arose. An Android application only allow the top view of a layout to be manipulated at any given time. Thus a way to manipulate both Views using only the top one had to be devised.

Due to how the Views were implemented, the Whiteboard viewer is always on “top”. This made for a simple solution. A variable was added to the Whiteboard in which a reference to the VuDroid PDF-viewer was stored. Whenever a scroll is called for in the Whiteboard, the Whiteboard scrolls first, and then invokes a call to the referenced viewer. The viewer is then turned active and is scrolled.

Since both of the Views are scrolled individually, there was a need to limit how long it should be possible to scroll. VuDroid already had these limit values stored. The VuDroid reference was used again, in order to get the limit values used in VuDroid and make sure that the Whiteboard stops its scrolling at these values.

4.4. Implementing textboxes

This section discuss the choice and implementation of textboxes.

4.4.1. EditText

There were several options for an application to display text. The application had the requirement that the text should be editable. Therefore the View EditText appeared as a good choice. It had many desirable properties, for example movability, background transparency, changing text colour, et cetera. An attempt was made to implement a method that created and then added a EditText to the ViewGroup. The main problem with this implementation was focus. As mentioned earlier the ViewGroup is Z-ordered. So the latest EditText were on “top” of the previous Views added. If a user was to write a whole

paragraph of text inside the EditText, it covered a lot of space and the user was not able to access anything underneath it.

4.4.2. Canvas.drawText

The focus problem was solved by using a method that the Canvas class provided, drawText(). Instead of writing some text into an EditText this method is given a string of text and then makes sure it will be drawn. The method could however not handle the newline sign, '\n'. All text was drawn on a single line. Instead of a newline a symbol, '[]', was drawn. To draw the right amount of lines, a split at every '\n' symbol was implemented splitting the string into several smaller strings. Then draw them one by one with a y-offset of the text size.

How the Canvas should know what text to draw was solved by creating objects of the MyText class and adding them to a list. Then the onDraw method of the canvas iterates through this list and draws them. MyText holds a String, coordinates and a TextPaint object that contains attributes as size, colour et cetera.

4.4.3. Text input

The next step was to find a way to get a string from the user. Again, EditText had a lot of the desired properties. A relevant question arose: how would a user want this to work? Of the different designs tried, the simplest one was to make a transparent EditText that the user could move around and write text in. When the user was done writing, a string containing the content and the coordinates of the EditText was sent to the drawing method. The text was then drawn in the exact same place on the Canvas as the EditText.

Unfortunately this was not as easy as anticipated, especially to fit the text in the transparent EditText with the drawn text on the Canvas. At first the font of the two were different. Fortunately there was a subclass to Paint named TextPaint that solved this problem. Another problem which still has to be resolved is the offset between lines. There is a small difference between the EditText and the text that is drawn. This is made more obvious when a string of text uses a large text size and is a couple of lines long. A further issue was the difference in how text was placed, something that had a rather simple solution. EditText always placed itself with its upper left corner at the coordinate where the user pressed. Whilst the drawText() method placed the lower left corner of its first line at the coordinate. To compensate for this difference, the y coordinate is decreased by 1x of the text size. With this compensation the text ends up in the correct area.

4.4.4. Editing text

Once the text had been drawn on the Canvas it is not editable. A fix was to add some kind of point on the screen that corresponds to the text. When pressed the text is loaded into the EditText and removed from the list of MyTexts. The text could then be edited and when the user is done it is added to the MyText list again and then redrawn into the Canvas.

4.5. Other issues

This section describes a few issues the project group encountered during the implementation of SlideNoter.

4.5.1. Showing both Views at once

As mentioned the application has been developed in two different parts. A part that shows a PDF document and a drawing part. Both of them were developed as Views in order to be able to put them in the same ViewGroup. A suitable layout had to be found making it possible to view both views at the same time. Multiple layouts were tested a, however most of them presented an issue when it came to scrolling. In the end a FrameLayout was chosen, a layout that Z-order the Views added to it. Because the DocumentView, the PDF part, could have many different layouts. For exapmle it could consist of images, it is hard to make it transparent therefore it was placed as the first child of the FrameLayout. WhiteboardView, the drawing part, was then added as a second child to the FrameLayout. Since WhiteboardView was placed “on top” of the DocumentView, it was made transparent so both of them were shown.

4.5.2. Layout design

Android has a system using XML files for creating the layout of an application. While in theory the usage of the building blocks that was provided is simple, problems occurred in strange ways. The layout included a bar on the right side of the screen where tools are located. During the implementation of the layout the code `android:layout_gravity="right"` was used without success. To solve the problem a simple workaround was found. When the layout is rendered it goes through the layout in a sequential manner. To solve the problem a reordering of the objects was made in the XML which resulted in the fields being shown in the correct location.

4.5.3. Changing between modes

Changing between drawing and scrolling was easily solved by using a so called Enum type that was simply named mode. When the user touched the screen different method calls were made depending on what mode was active. Modes could easily be added on as development proceeded: just add the mode to the Enum list, then add methods for a touch event. For example, Text mode was added this way.

In order to clarify what mode the user is in, when a user clicks a button to change mode, the text on it changes and an indicator on it is lit. Some extra buttons appear below the active mode indicating there are other options available in the new mode compared to the previous one. For example, the button for changing text size only appear while in Text Mode.

5. Results

In this chapter a prototype of the application is presented. The prototype is a working copy of the application, yet not the finished product. This chapter is dedicated to present this prototype, its functions and if it fulfills the requirements. This will be presented with text and pictures of the running prototype.

5.1. The Application

The user experience of the application consists of four different sections. These four sections are described in their respective section.

5.1.1. The Main Menu

The main menu is the first view a user sees when starting the program. In figure 4 you can see three buttons. The first one let the users browse through the hard drive to locate PDF-files. The second button opens up a blank Whiteboard without an underlying PDF. The last button is simply an exit button that shuts the program down.

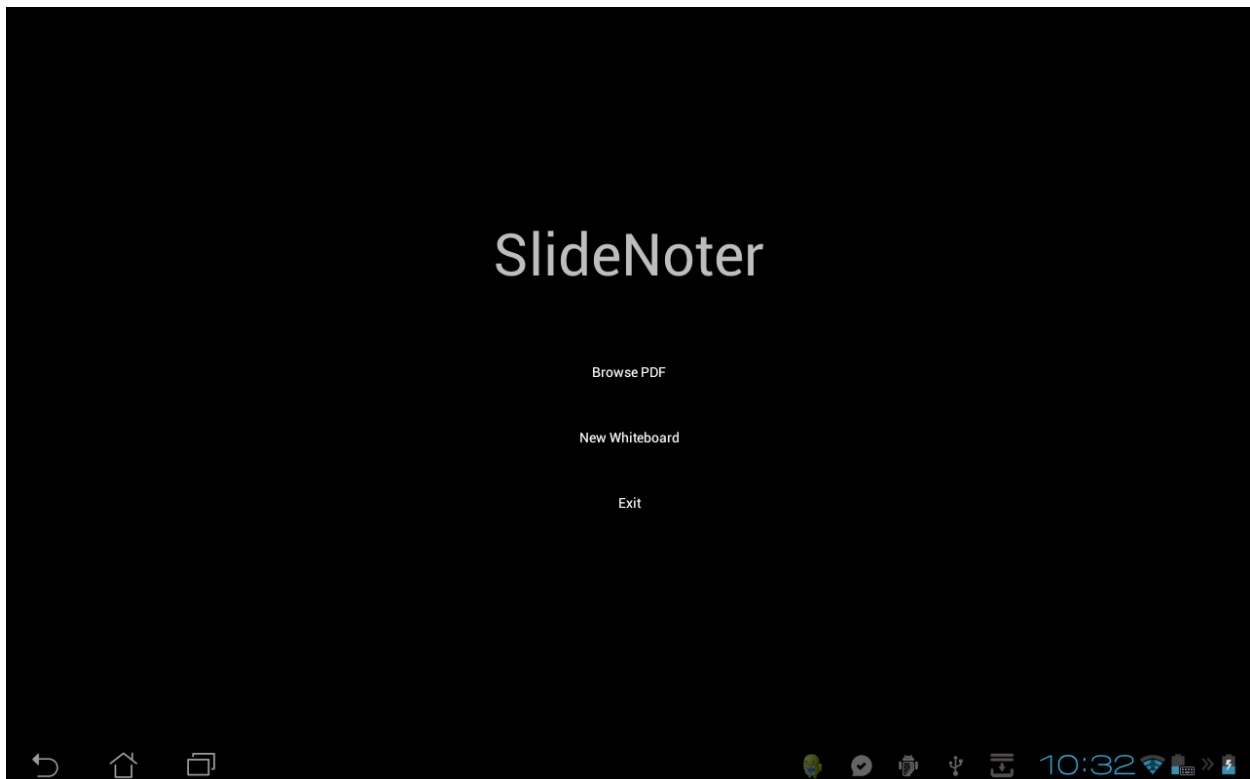


Figure 4. The Main Menu

5.1.2. The Browser

When a user presses the browse button he or she is taken to the browser view. This is a view used to find the PDF-file the user would like to open. As shown in figure 5, the browser contains two tabs. The first tab is called *Browse*. It is in the browse tab the entire folder hierarchy of the filesystem is shown; the user can browse through the folders until the desired PDF-file is found. When a PDF-file is selected the browser redirects the user to the PDF-Viewer and Whiteboard.

The second tab is called *Recent*. As the name implies it shows the PDF-files which the user recently have opened.

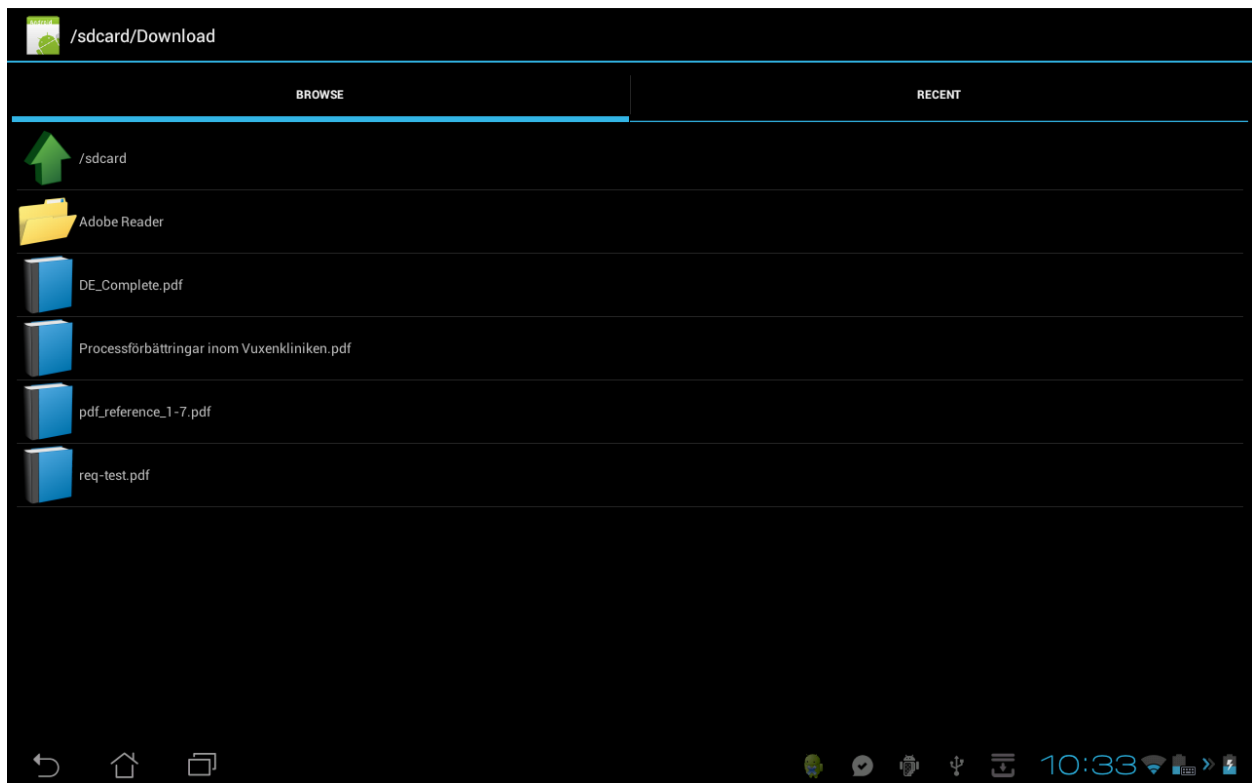


Figure 5. The Browser

5.1.3. The PDF-Viewer

When a User has opened a PDF-file via the browser, the application will change view to the PDF-view. In this view the PDF-file will be rendered and the user is able to scribble notes upon the PDF. The Whiteboard currently has four modes. The modes are selected through the use of three buttons located in a menu bar on the right hand side. Scroll mode is the default mode and it is automatically selected if none of the buttons are pressed.

The Draw mode lets the user draw freely by hand. The pen size and colour can be changed with buttons that appear on the left when this mode is selected. An erase tool is to be found here as well. In figure 6, lines are drawn to point out where the text comments are pointed to.

The Text mode lets the user add text comments by pressing where he wants them on the screen. The font size and colour can be changed by buttons that are only visible when in Text mode. Once the text is placed, a little square appear in the top left corner. When clicking the square the user can edit the text string, move the text box around on the screen or remove the comment all together. As seen in figure 6, two text comments have been made.

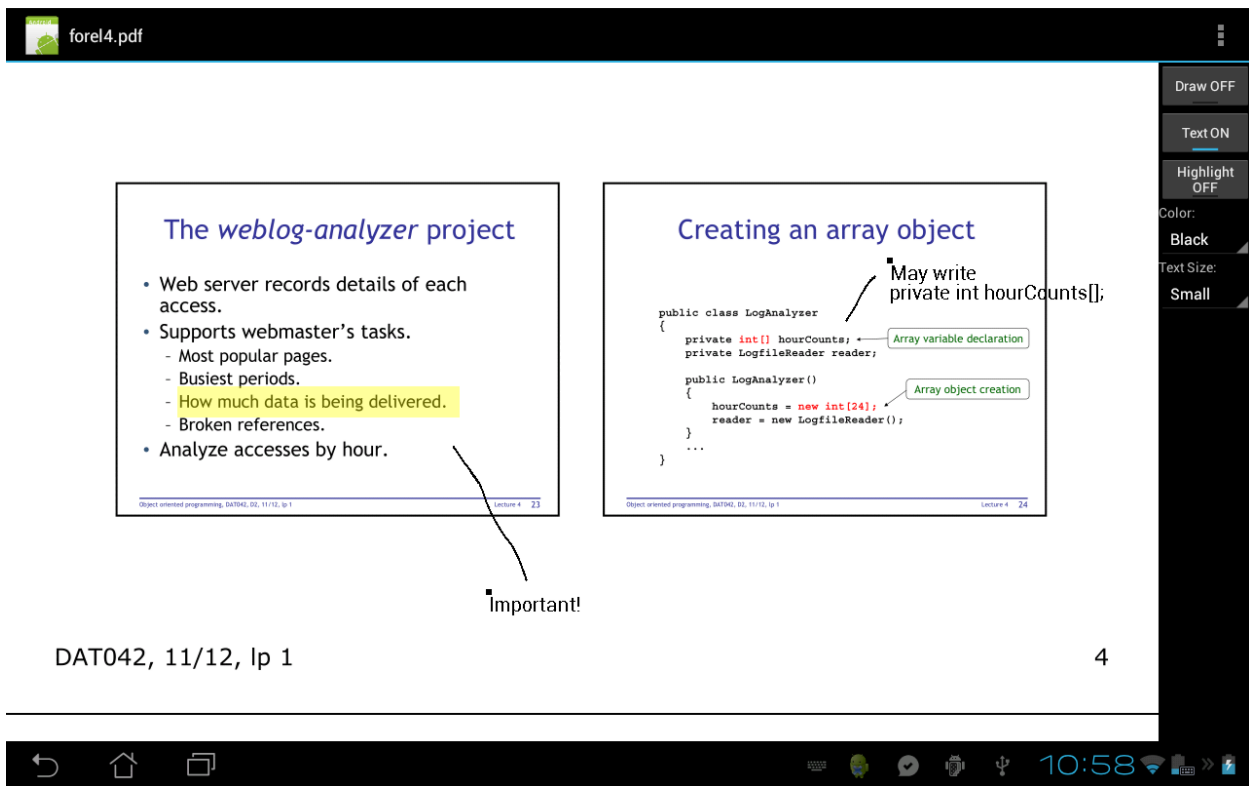


Figure 6. A PDF-file with a few notes on it

In the Highlight mode, the user is presented with the possibility to highlight areas. In order for areas to be highlighted the user must press and hold down a finger, then drag it diagonally from corner to corner of the area to be highlighted. The current implementation adds an opaque layer of yellow on the area selected. It is used in figure 6 to highlight an important note.

5.1.4. The Whiteboard

If the user has pressed the New Whiteboard button of the main menu page, the application renders a blank view with the mode buttons on the right hand side. This view is the very same as described in section 5.1.3 above, with the exception that this has no underlying PDF-file. All of the aforementioned functions are shown in figure 7.



Figure 7. A demonstration of the Whiteboard

5.2. Fulfillment of the functional requirements

The functional requirements of the first three iterations have all been met. At the end of the specified time for the project, work on implementing and fulfilling the requirements of the fourth iteration had begun. These implementations were however cut out from the release of the prototype as they were not yet finished. As a consequence none of the functional requirements from the fourth iteration have been fulfilled.

5.3. Fulfillment of the non-functional requirements

Unlike the functional requirements, the non-functional ones cannot be measured. As an example, the definition of “User-friendly UI” is in the eyes of the beholder. While certain users may think a UI is easy to navigate, others might experience the opposite. Hence, it is hard to decide if a requirement is fulfilled or not. A good way to decide the fulfillment of non-functional requirements is to let the customer or end-users test and evaluate the application. However, since grand scale testing was left out of the project, the non-functional requirements have been evaluated by the project members instead. Due to this, the evaluations are not unbiased and before a release of the application an unbiased evaluation would be desirable

At the start of the third iteration the non-functional requirements of the previous iterations were fulfilled. However, with the integration of the two parts together of the application a bug occurred causing a memory leak. This caused the PDF to stop rendering after extensive scrolling. This problem left the requirement *“it should be stable, even after long uses”* unfulfilled. Of the requirements from the third iteration the *“Fast and responsive UI”* was met. As the work from iteration four was cut out, no work on fulfilling those requirements has been done.

5.4. User tests

Due to the development process requiring more time than planned, the project group agreed that it would be more beneficial to conduct public user tests when the application is closer to its final release. Therefore the focus lay upon implementing more functionality and solving the current bugs instead of conducting the user testing.

6. Discussion

This chapter will discuss the pros and cons of the specification and major decisions that were made throughout the project process, as well as discuss possible uses outside of education.

6.1. Evaluation of the Specification

Before the project started a basic specification was formulated. What the finished product would contain, and what would be ideas for future development. From the above mentioned specification, some of the functions were selected for each iteration. After considering the time required for each part, the decision was taken to add some extra time for each component to compensate for bugs and problems that might occur.

At the end of the project time, iteration three was near complete. The application was not yet complete according to the initial specification. The application fulfilled all the specifications for what a finished product would contain, except for saving and loading.

6.2. Evaluation of major decisions

This section describes and evaluates the major decisions of the project in their respective sub-chapter.

6.2.1. Working with the Spiral Model

Working with the Spiral Model was causing work to only progress on the functions included in each iteration. It provided a solid structure for the development of the application, without requiring a lot of time for formalities. There were always clear goals on what to be finished and when. The time schedule held rather well throughout the whole process. Some of the problems described in chapters 4.3-4.5 required a bit more time than was expected, therefore saving and loading had to be cut from the third iteration and put into the fourth.

The fact that the Spiral Model was developed for larger projects was not an issue for the project. By dividing the project into four small parts and applying the Spiral Model on them, a more agile approach was created which gave good results for the project.

6.2.2. Android as a Development Platform

As stated in 3.3. Android application programming is based on one of the world's most used programming languages, Java, and a lot of documentation and solutions to problems exist. Google also provides excellent documentation on everything that they themselves have added to the Java Language via their developer web page. All this in unison made Android an easy platform to adapt to and develop for.

As the layout is kept separate from the actual code, this allowed for code and layout to be developed simultaneously. While one person focused on code, the other could focus on the UI, without the need for the functions to be fully coded yet. As code for layout and UI are kept separate from the java code, the one working on UI does not have to know how to code functions, and vice versa. This feature proved to be beneficial since two project members could work on the layout while the other who could work on the contents. Quite a few problems occurred when making the UI as no one in the group really was well oriented when it came to UI, and layout.

6.2.3. SurfaceView as Drawing Area

During the research phase the SurfaceView class was found. The advantages of the class were many and it was decided to use it. However, later on in the project the choice of SurfaceView proved to be less than ideal. During the implementation it was discovered that SurfaceView is used ideally only when a smaller drawing surface is required. Since the drawing surface in SlideNoter may span over dozens of pages this proved to be an issue, an issue which may be the cause of a memory leak. Beside this, it turned out that all of the features used in SlideNoter could be achieved with some tricks with the regular View class.

6.2.4. VuDroid as the PDF-reader

VuDroid was chosen based on primarily two factors. Firstly, it had a nice UI and was easy to use. Secondly, it was the fastest of the projects that the project group looked at. The project lacked in one key area though, documentation, which proved to be a problem during the process. In order to be able to use VuDroid as a base, all the different parts had to be understood, which was fairly time consuming. In hindsight, if any of the other projects had been better documented, they would probably have been a better choice in the end, at least regarding time.

Altogether though, VuDroid was a good choice to build the application on. The browser was already in place and it was well structured to allow other file extensions in the future. After the initial understanding of VuDroid, it did not require too much work to integrate it into the application and have it work together with the Whiteboard.

6.2.5. The Usage of a Version Control System

The use of some kind of Version Control System is required for a project of larger size with multiple developers. As section 3.6. brings up VCS are a way for multiple users to effectively code together. The usage of TortoiseSVN and the repository supplied by Chalmers were at the beginning confusing. However, when all the project members had

learnt to use tool efficiently it proved to be a valuable tool for the project, especially when a need to revert changes to a previous revision emerged.

Working with SVN instead of cloud based storage areas such as Dropbox, or simply mailing code back and forth proved to be positive. This is mainly because SVN provided a relatively easy way to synchronize the project files and allows for a revision history. Thus providing the team with the means to revert changes to an earlier state if something went wrong. While this could have been achieved with mail as well, SVN was much more structured and easy to navigate.

6.3. Possible other uses of the application

During the project the focus has been to develop an application for scribbling notes on lecture slides. However, with added functionality there are other possible fields of application for the application. One example could be when reading a document and the user encounters a word it does not understand. If there is a function which enables word lookup or translation, then the reader could use it to grasp the meaning of the word. Building on this feature the application could add the explanation to the document, so that the next time the user encounters the word there is already an explanation for it.

7. Conclusion

The following section describes experiences learnt during the project and how it could have been done differently. It also contains a section on further improvements that could be made in future by the project members or other developers.

7.1. Critical discussion

One initial criticism is that the software engineering process could have been smoother. Looking back at it, more emphasis should have been put on the research phase in the beginning of the project. If a more extensive research phase was conducted, with documentation of which parts of the Android SDK that were suitable for the project's goal, this could potentially have avoided some of the problems that occurred. For example the SurfaceView as discussed in 6.2.3.

One of the toughest problems of the project was handling both writing code and the report simultaneously. When working on the project, the time was usually spent working several days on the code, followed by several days on the report. A better solution would have been to manage our time differently; to work a few hours with coding followed by a few hours of writing the report every day or week. This would have led to the project progressing in a more even pace where it would have been easier to keep to a time schedule.

One of the biggest flaws of the application is the fact that the save function is not implemented properly. Without it, there is little reason to actually use the application, since the comments you add will not be saved for the next time you open the PDF. The main reason for it being left out was because it was overshadowed by the other functions. Since it was fairly low priority in the requirements specification, time did not allow for it to be implemented. This was a planning failure; the project group underestimated the time some parts of the application would take, for example getting the PDF-reader to work and integrating it with our application.

7.2. Future work

There are improvements to the application that can be considered for future development, following is a short selection with brief descriptions.

7.2.1. Fixing memory leak

When a user has opened a PDF-file and has scrolled down a sufficient amount, the document stops rendering. According to Log Cat, this is due to insufficient memory. VuDroid alone can scroll much farther than with an added layer of Whiteboard.

7.2.2. Support for more file extensions

Taking the allocated time frame for this project into consideration, it was decided that an open source solution VuDroid was to be used. The choice of VuDroid currently limits the application to reading PDF-files. Thus future work on this project is adding support for other file extensions, for example PPT.

7.2.3. The ability to search for notes

As the application works right now the user cannot search for specific text written in notes or on the PDF. Further development on either the reader or on the Whiteboard would result in the ability to search for text on the slides, or within the text boxes. The best solution for the latter would be integrating the Whiteboard with a SQLite database which Android has built in support for.

7.2.4. Cloud based storage and multiple user editing

Storage of files is slowly being moved from the user to a cloud based server. Further development of this application could allow users to connect to a web server and load files from there. Along with a cloud based solution multiple users have the ability to read the same file. With further work this could enable multiple users to edit simultaneously in real time.

7.2.5. Word lookup

When the user is reading a large document in a foreign language, for example an E-book. The user may want to lookup a word in a dictionary. A dictionary plugin is therefore a possible thing to implement.

Bibliography

- Adobe Systems Incorporated. (2006) *PDF Reference*. Sixth edition.
http://www.adobe.com/content/dam/Adobe/en/devnet/acrobat/PDFs/PDF_reference_1-7.PDF (2012-03-16).
- Android (2012a) Android 3.0 Platform Highlights. *Android Developers*.
<http://developer.android.com/sdk/android-3.0-highlights.html> (2012-03-19).
- Android (2012b) Activity. *Android Developers*.
<http://developer.android.com/reference/android/app/Activity.html> (2012-05-05).
- Android (2012c) Canvas. *Android developers*.
<http://developer.android.com/reference/android/graphics/Canvas.html> (2012-05-05).
- Android (2012d) Paint. *Android developers*.
<http://developer.android.com/reference/android/graphics/Paint.html> (2012-05-05).
- Android (2012e) Android 2d graphics. *Android developers*.
<http://developer.android.com/guide/topics/graphics/2d-graphics.htm> (2012-05-05).
- Android (2012f) Drawable. *Android developers*.
<http://developer.android.com/reference/android/graphics/drawable/package-summary.html> (2012-05-05).
- Android (2012g) Packages. *Android Developers*.
<http://developer.android.com/reference/android/view/package-summary.html> (2012-05-05).
- Android (2012h) ViewGroup. *Android Developers*.
<http://developer.android.com/reference/android/view/ViewGroup.html> (2012-03-20).
- Android (2012i) How Android Draws Views. *Android Developers*.
<http://developer.android.com/guide/topics/ui/how-android-draws.html> (2012-03-19).
- Android (2012j) User Interface. *Android Developers*.
<http://developer.android.com/guide/topics/ui/index.html> (2012-03-19).
- Android (2012k) SurfaceView. *Android Developers*.
<http://developer.android.com/reference/android/view/SurfaceView.html> (2012-03-14).
- AndroidTics (2012) Android Activity and Intent.
<http://androidtics.com/ActivityIntent.php> (2012-05-14).
- Burnette, E (2010). *Hello, Android. Introducing Google's Mobile Development Platform*. Third Edition. Raleigh, North Carolina. The Pragmatic Bookshelf
- Chalmers Insidan (2012) Versionshantering.
<http://www.chalmers.se/insidan/SV/arbetsredskap/it/bastjanster/versionshantering> (2012-05-05).
- Copyright Directive (2001) *European Union law*, European Union
<http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2001:167:0010:0019:EN:PDF> (2012-05-05).
- Dictionary.com (2012) Unabridged. *Copyright*.
<http://dictionary.reference.com/browse/copyright> (2012-05-05).
- DMCA (1998) *U.S. Copyright Office*. US Government
<http://www.copyright.gov/legislation/dmca.PDF> (2012-05-05).
- Ferenc Hechler (2012) *Android PDF-Viewer*. <http://andPDF.sourceforge.net/> (2012-03-16).
- GNU (2012a) *What is free software?* <http://www.gnu.org/philosophy/free-sw.html> (2012-03-19).
- GNU (2012b) *GNU Project*, Free software Foundation.

<http://www.gnu.org/copyleft/copyleft.html> (2012-05-05).
GNU (2012c) *GNU Project*, Free software Foundation
<http://www.gnu.org/licenses/gpl-1.0.html> (2012-05-05).
GNU (2012d) *GNU Project*, Free software Foundation
<http://www.gnu.org/licenses/gpl-3.0.html> (2012-05-05).
Google Play (2012a) *qPDF Notes*, Qoppa Software.
<https://play.google.com/store/apps/details?id=com.qoppa.activities.noteskey> (2012-05-05).
Google Play (2012b) *ezPDF Reader*, Unidocs Inc.
<https://play.google.com/store/apps/details?id=udk.android.reader> (2012-05-05).
Google Play (2012c) *RepliGo Reader*, Cerience Corporation.
<https://play.google.com/store/apps/details?id=com.cerience.reader.app> (2012-05-05).
JPedal (2012) <http://www.ipedal.org/> (2012-03-16).
Lagen.nu (2011) *Lag (1960:729) om upphovsrätt till litterära och konstnärliga verk*.
<https://lagen.nu/1960:729#P43> (2012-05-05).
Langpop.com (2012) *Programming Language Popularity*. <http://langpop.com/> (2012-05-06).
Microsoft (2012a) Microsoft Trademarks. *Legal and Corporate Affairs*.
<http://www.microsoft.com/about/legal/en/us/IntellectualProperty/Trademarks/EN-US.aspx>
(2012-03-19).
Microsoft (2012b) *Save as PDF*.
<http://office.microsoft.com/en-us/powerpoint-help/save-as-PDF-HA010064992.aspx>
(2012-03-19).
PDF-renderer (2012) <http://java.net/projects/PDF-renderer/> (2012-03-16).
Sommerville, I (2011). *Software engineering*. Ninth Edition, p.48-49. Boston. Pearson.
Sourceforge (2012). <http://sourceforge.net/directory/license:osi-approved-open-source/>
(2012-05-05)
VuDroid (2012) <http://code.google.com/p/vudroid/> (2012-03-16).
Waterfall Model (2012) All about the Waterfall Model. <http://www.waterfall-model.com/>
(2012-05-05)
Wikimedia (2008) Spiral model (Boehm, 1988).
http://commons.wikimedia.org/wiki/File:Spiral_model_%28Boehm,_1988%29.png
(2012-05-14)

Appendixes

Appendix A

When specifying the specification requirements the following form was used.

ID:	###
Use case / scenario:	The use case
Trigger:	What triggers this case
Precondition:	A precondition for the case to trigger
Basic path:	
A basic path that the user follows in order to trigger	
Exception path:	
What cases should trigger an exception and what kind of exception should be triggered	
Post condition:	
What happens after a successful case	
Author:	Author name

Compact list of the specification requirements, divided on the iterations where they were set.

Iteration 1 - PDF-reader

Functional Goals

Usable on Android OS.

Read and draw a PDF file.

A basic interface containing open/options/exit etc.

Non-Functional Goals

Should open and show a PDF file fast.

It should be stable, even after long uses.

Iteration 2 - Whiteboard

Functional Goals

Usable on Android OS.

Draw basic figures.

Scroll the figures (so they can follow the PDF-page later).

UI for all current functions with room for new functions.

Non-Functional Goals

Good structured object oriented code for easy maintenance

Iteration 3 - SlideNoter

Functional Goals

Scroll the PDF-file along with the notes.

Draw basic figures on a PDF-file.

Highlight

Text boxes

Draw figures/lines/arrows

Non-Functional Goals

If the application crashes then the notes should not be lost.

Fast and responsive UI.

Iteration 4 - Continued Developments

Functional Goals

Prio 1.

Save your notes in a stand-alone file.

Load a PDF file along with the file containing the notes.

Text formatting boxes (bold etc)

Prio 2.

Search among the notes

Symbol recognition

Search among symbols

Voice recording

Bind voice recording playback to specific pages

Support for PPT

Server-based storage

Add notes to a database

Spellchecker

Tabbed browsing

Non-Functional Goals

A very user-friendly UI.

Should be able to handle PDF files with 1000 pages

Appendix B - UML

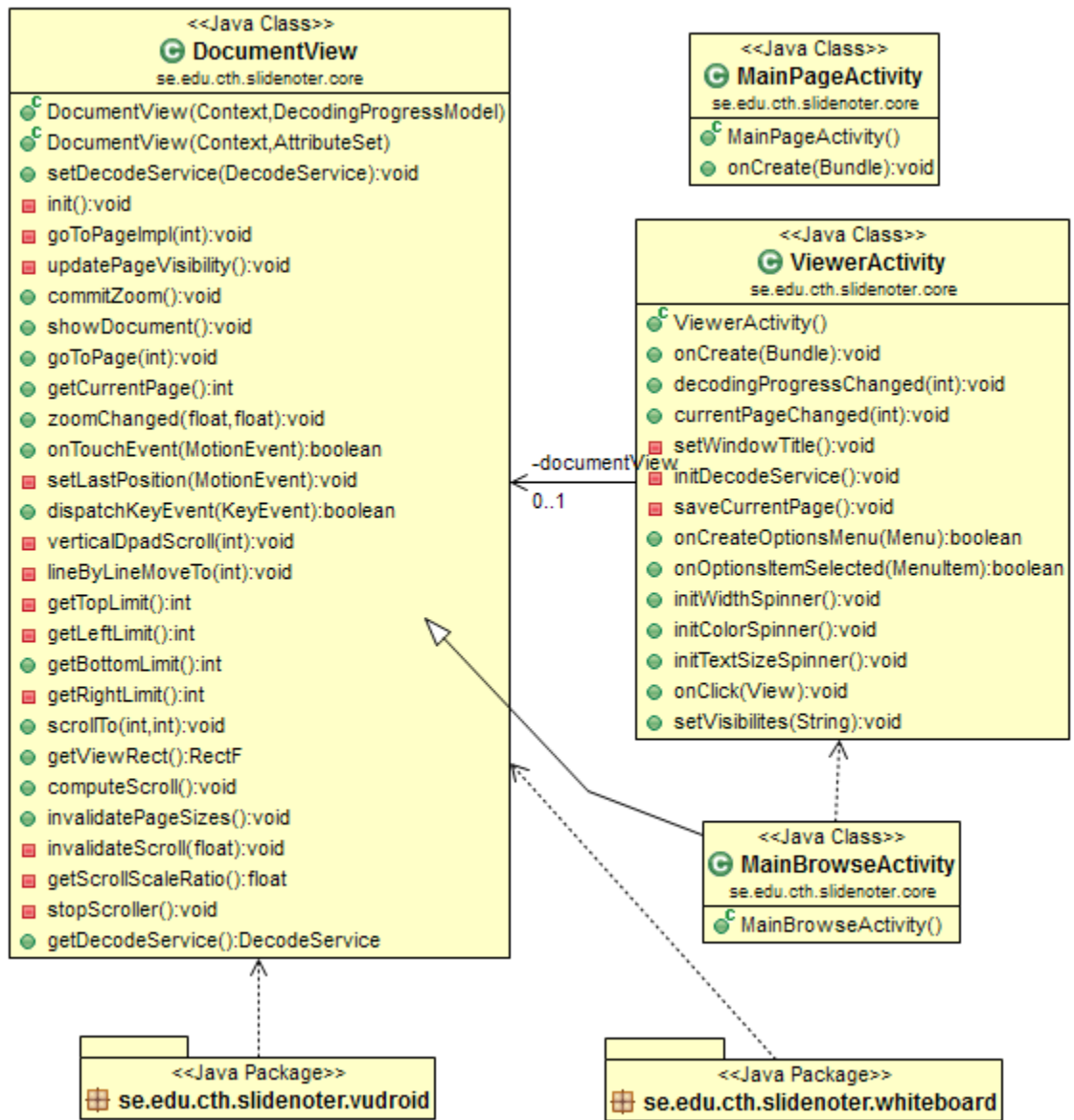


Figure 8. A UML diagram of the SlideNoter package

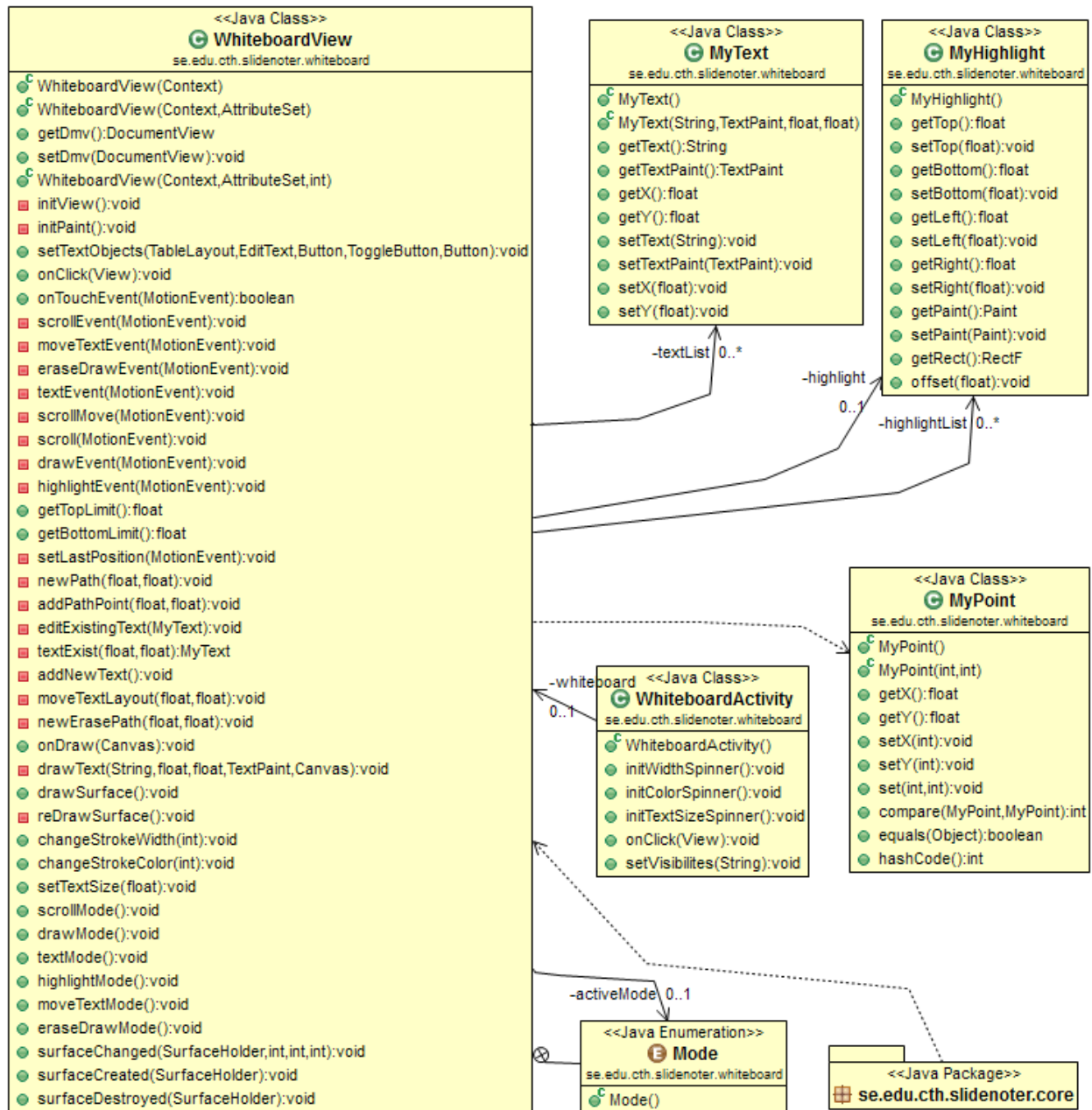


Figure 9. A UML diagram of the Whiteboard package

Appendix C - List of Contributions

C.1. Fields of responsibility

Background research - Everyone

Project Description - Everyone

Choice of methods - Robert Moberg

Design of poster - Axel Pelling

Graphical User Interface - Everyone

Integration of VuDroid - Everyone

Interaction between VuDroid and the Whiteboard - Patrik Aldenvik, Jonas Scholander

Lead Editor - Robert Moberg

Lead Programmer - Patrik Aldenvik

Planning - Robert Moberg

Project Log - Everyone

Whiteboard - Everyone

Project Report - Everyone

C.2. Presentations

Midterm presentation: Axel Pelling, Jonas Scholander

Final presentation: Robert Moberg, Patrik Aldenvik

Verbal opposition: Axel Pelling, Jonas Scholander