AALBORG UNIVERSITY

STUDENTS STUDYREPORT

Group SW602F12

Bachelor Project

WOMBAT - Timer Applicaiton for GIRAF

Authors:
Kristian K. Foged-Ladefoged
Simon B. Frandsen
Rasmus H. N. Aaen

Supervisor:
Ulrik Nyman

June 1, 2012

Department of Computer Science
Aalborg University
Selma Lagerlöfs Vej 300
DK-9220 Aalborg Øst
Telephone +45 9940 9940
Telefax +45 9940 9798
http://cs.aau.dk

Title: Timer Application
Subject: Android Systems
Semester: Spring Semester 2012
Project group: sw602f12

Participants:
Kristian Kolding Foged-Ladefoged

Rasmus Hoppe Nesgaard Aaen

Simon Blaabjerg Frandsen

Supervisor:
Ulrik Mathias Nymann

Number of copies: X

Number of pages: X

Number of appendices: X Pages

Completed: 4th June – 2012

Synopsis:

This project is about the development of an Android timer application that will ease the daily routine for both children with autism spectrum disorder and their educators and parents. The educators and parents use physical timers to help the children manage their activities, these timers exists in a variety of forms, colors, and timespan. Furthermore the educators and parents can use pictograms to inform the children, of which activity is currently in progress or it can be used as a schedule to describe their daily routines. The timer application is developed with the goal to replace the physical timers by providing virtual timers. The timer application is developed such that the educators and parents can attach an extra timer or pictogram(s) next to the timer in progress.
This project is a part of the multi project system, GIRAF. Which is a system specifically developed to ease the daily routine for educators and parents by providing a collection of tools.

PREFACE

This project has been produced by group SW602f12 at Aalborg University in the spring 2012 on the sixth semester of the software engineering study.

This project is written as a part of the process of developing the timer application for the GIRAF project. The GIRAF project is designed to support guardians of children with autism spectrum disorder, among the applications in the GIRAF project is the timer application that is built to replace physical timers.

The project describes the working process used to develop the timer application, included in this is the development method, the design, the implemented features, and the tests of the timer application.

The intent of the project is to explore the possibilities of combining tools used by guardians of children with autism spectrum disorder, in a device which is portable and easy to use.

The project group would like to thank the educators whom helped developing and designing the timer application as well as our supervisor for the interest and cooperation in the project.

CONTENTS

WORD DEFINITION

- Time Timer - A visual clock. Used by parents and teachers as a tool to visualize time for children.
- Pictogram - A graphic symbol that conveys its meaning through its similarity to an object.
- Eclipse - An intelligent development environment for Java.
- ASD - Autism Spectrum Disorder.
- Children - Children refers to "Children with ASD".
- Guardians - Parents, teachers, caretakers, and educators of children with ASD.
- MVC - **M**odel **V**iew **C**ontroller [10].
- Timer application - The end product of this individual project.
- GIRAF - The name of the project from last year, which we are developing further.
- WOMBAT - Name of the timer application, **W**ay **O**f **M**easuring **Ba**sic **T**ime.
- PARROT - Name of the pictogram application, **P**ictogram **A**ssisting with **R**hetoric **R**easoning **O**r **T**alking
- We - In the introduction, it refers to the entire multi project group. After the common report, it refers to the individual project group.
- Configuration - Can refer to a saved timer.
- `SubProfile` - In code examples, `SubProfile` refers to timer configurations.

Part I

Introduction

This part is a common part of the reports in the multi project groups, and in this common part there will be an introduction to the project, including background knowledge about the target platform, and knowledge about ASD. There will be an analysis of the problem followed by a system definition of the whole multi project, and the specific group project.

Section 1.8 is the last section of the common report, which means that the rest of the report will be an individual part written only by the timer group.

1

INTRODUCTION

In order to describe the context of the system, we – as a multi project group – will in the following state the motivation of the project, the group of people we are aiming at helping, the technological platform chosen, the used development method, followed by a problem definition, a system description and architecture, and the conducted usability test.

1.1 MOTIVATION

As this is a student report written as part of a learning project, we are required to comply with the study regulation. The main areas of focus, according to the study regulation, are: multi project management and quality assurance in the form of requirements analysis, requirements management, and testing. The goal is to create a comprehensive software system, across multiple project groups, in order to enhance our competences in analysis, design, implementation, and evaluation of software applications in regards to the system requirements[15].

This project builds on top of a previous project, and is further developed, with the aim of having other students continue the development. The goal of the project, we are building on top of, is to create a touch based tablet system to support children and their guardians in everyday scenarios.

1.2 TARGET GROUP

Our target group is both children and their guardians. These guardians have certain needs for special tools and gadgets that help to ease the communication between them and the children.

Five teachers and educators, who work with children, act as customers. They will provide requirements and information about the institutions' way of working to give us an insight into their daily struggles.

1.2.1 *Working with Children with ASD*

This section is based upon the statements of a woman with ASD [6], explaining what it is like to live with ASD, and an interview with an educator at Birken, a special kindergarten for children (see appendix 10.2 for interview notes).

People with ASD are often more visual in their way of thinking. Rather than visualizing thoughts in language and text, they do it in pictures or visual demonstrations. Pictures and symbols are therefore an essential part of the daily tools used by children and the people interacting with them. Also, children can have difficulties expressing themselves by writing or talking, and can often more easily use electronic devices to either type a sentence or show pictures, to communicate with people around them. Another characteristic of children is their perception of time. Some of them simply do not understand phrases like "in a moment" or "soon", they will need some kind of visual indicator that shows how long time they will have to wait.

Different communication tools for children with autism already exist, but many of them rely on a static database of pictures, and often these has to be printed on paper in order to use them as intended. Other tools, such as hourglasses of different sizes and colors, are also essential when working with children, and these tools are either brought around with the child, or a set is kept every place the child might go, e.g. being at an institution or at home.

There exists tools today which helps the guardians in their daily life, although – as stated in Drazenko's quote – none of them are cost-effective enough to be used throughout the institutions. From the quote, it is clear that there is a need for a more cost-effective solution.

> The price of the existing solutions are not sufficiently low such that we can afford to buy and use them throughout the institution.
>
> - Drazenko Banjak, educator at Egebakken.

1.3 TARGET PLATFORM

Since we build upon last year's project, we are bound to use the platform they used, which is tablets running the Android operating system.

In this project we have been provided with five Samsung Galaxy Tab 10.1 devices[11]. The firmware on the tablets is version 3.2. This version, as of project start, is the latest stable version available for these specific tablets. [14]

1.4 DEVELOPMENT METHOD

As a part of the study regulation we have been required to use the same development method in each individual group. Two methods have been considered, *XP* (eXtreme Programming) [16], and Scrum [2].

With the knowledge of both XP and Scrum, we decided in the multi project to use Scrum of Scrums, which is the use of Scrum nested in a larger Scrum project [1].

The reason for choosing Scrum of Scrums is that everyone, at all times, will be able to know what the vision of the project is, and how close every group is to achieving their individual goals of the vision.

Another element of the Scrum method is that a close contact with the customers is maintained. This helps keep the product backlog up to date and correctly prioritized. The customers are presented with the vision of the project, as well as showing the latest release when we have meetings with our customers.

We customized Scrum to fit our project. The changes are as follows:

- The sprint length have been shortened to approximately 7 - 14 half days.
- Some degree of pair programming have been introduced.
- There is no project owner because this is a learning project.
- Everyone is attending the Scrum of Scrums meetings.
- The Scrum of Scrums meetings are only held once at sprint planning.

1.5 problem definition

The problem statement is as follows:

> How can we ease the daily life for children with ASD and their guardians, while complying with the study regulation?

This problem statement is necessarily vague to allow the individual groups some freedom in their projects, while we maintain the overall structure of the multi project, however there are limiting factors. We are limited by resources and time available, as we are only working on this project for a single semester. However, all work done in this multi project will be passed on to the next line of students, which means we can make a full system design and pass on anything we do not have the time or resources for. This also requires that our work need to be of such quality that it is understandable by students of the same educational level as ourselves.

1.6 system description

GIRAF is a collection of applications, either fully or partially interdependent, for the Android platform, designed to be used by guardians

and children. GIRAF consists of five projects with various degree of interaction. These projects are named Launcher, PARROT, WOMBAT, Oasis, and Savannah. Each of the groups have produced individual products, which are parts of a greater project, GIRAF.

Launcher handles execution of GIRAF apps, and at the same time it provides safety features to ensure that a user that is not authorized to interact with the rest of the system will not be able to do so. When the launcher executes an app, it will provide it with profile information, specifying which child is currently using the app, as well as which guardian is signed in.

PARROT is an app which provides access to pictograms – pictures with associated information such as sound and text – which can be used for communication. PARROT also gives guardians functionality for adding additional pictograms, as well as organizing the pictograms into categories for ease of access, based on the needs of the individual child.

WOMBAT is an app which purpose is to help the children to understand the aspect of time, by visualizing it. WOMBAT provides different ways of displaying time, as well as the possibility to configure the app for the needs of individual children.

Oasis locally stores the data and configuration of the GIRAF platform, and provides an API to access it. The stored data and configurations are synchronized to the Savannah server, if available. In addition, an app is provided for the guardian to access the stored data and configurations.

Savannah provides Oasis with a way to synchronize tablets running GIRAF. Furthermore, a website is provided to ease administration of the synchronized data.

1.7 ARCHITECTURE

Our System architecture – shown in Figure 1 has been designed with simplicity in mind and was greatly inspired by the MVC pattern. This means that the architecture is divided into three layers. The lowest layer is the database where the information is stored. Above this layer is the controller layer which, in the GIRAF platform, is known as Oasis. The controller is responsible for querying the database for information needed in an app and the controller is also responsible for storing information in the database. The last layer is the apps. This division of layers give the GIRAF platform a low cohesion which makes it easier to work with individual parts of the platform independently.

We have chosen to redesign last year's architecture [7] to make it easier to work with. We have simplified the architecture because we feel it is unnecessarily complex.

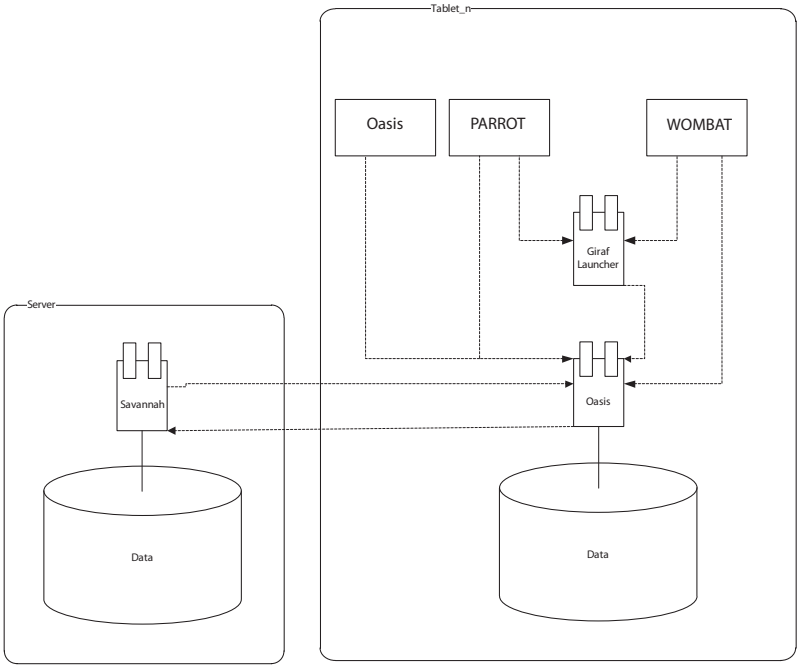


Figure 1: The GIRAF architecture

1.8 USABILITY TEST

As stated in the motivation, quality assurance through testing of the system is required. Therefore a usability test was conducted in order to measure the current usability of the GIRAF platform as a whole, as well as of the individual parts of the platform. Furthermore, the next wave of developers will immediately be able to start correcting the found usability issues.

1.8.1 *Approach*

The test group consists of the five contact persons. We assess that they, as a test group, are representative. We base this on them being a mix of educators and teachers, with varying computer skills.

They have prior knowledge about the overall idea of the GIRAF platform, and although some of the contact persons had previously informally used some aspects or parts of the system, they had not been exposed to the platform as a whole, and therefore still are of value.

The invitation sent to the test persons can be found in Figure 45.

The Instant Data Analysis (IDA) method for usability is chosen. A traditional video analysis method could be used, but since IDA is designed for small test groups, this approach is used. [8]

Setup

The usability test is divided into two tests: A test of three user applications, and a test of two administrative applications. The user applications are: The launcher, PARROT, and WOMBAT. The administrative applications are: The Oasis app and the Savannah web application. Each test is assigned a team to accommodate the need to run two tests simultaneously. The teams are made with respect to the criteria of the Instant Data Analysis process.
Each team consisted of:

- 1 x Test Coordinator
- 1 x Test Monitor
- 1 x Data Logger
- 2 x Observers

The usability lab at Aalborg University is designed with two rooms for usability testing and a control room to observe and record the tests. The two test chambers are assigned a test each and the control room is used to observe both tests as seen in figure 2.

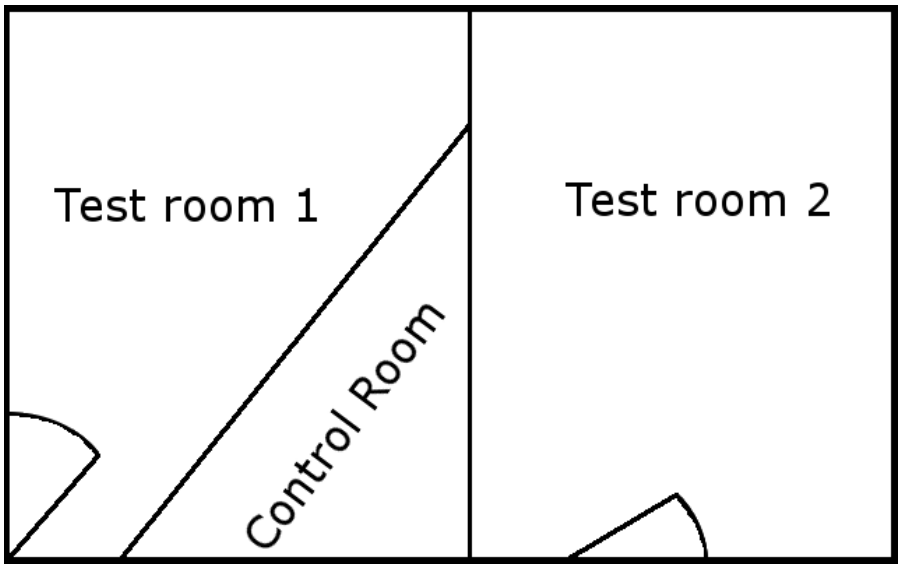


Figure 2: An overview of the usability lab at Cassiopeia, Department of Computer Science, Aalborg University.

Execution

Figure 3: The schedule of the usability test.

The tests are conducted according to the schedule in Figure 3.

Briefing, debriefing, and questionnaire documents can be found in **??**, and the results of the test can be found in Section 7.4.

2

ANALYSIS

Through meetings with an educator at Birken, a kindergarten for children with ASD, we have learned about the importance of having access to well-designed communication tools when working with children with ASD.

At Birken they often use hourglasses and time timers, in different sizes and colors to visualize the progression of time to the children. The children can then associate the color and size of an hourglass with the time it represents.

Our contact person explained how the educators at the institution use pictograms to communicate with the children. The children have a schedule of the day, where all their daily activities are listed in pictograms, such that the children can always go to their schedules and find out what they are going to do next. Also activity instructions are listed with pictograms, i.e. in the bathroom there is a scheme showing what to do when going to the bathroom.

The pictograms used at Birken is from the software *Boardmaker*[9]. To use the pictograms from *Boardmaker*, the educators have to print, cut, and laminate them.

The educators have to take timers and pictograms with them, everywhere they go. Since there is supposed to be more than one specific timespan available, the timers can take up much space. Furthermore the pictograms are lightweight, which can make it difficult to use them outside when it is bad weather or windy.

2.1 SYSTEM DEFINITION

The timer application is targeted for Android tablets running Android 3.2. The use space is institutions and homes of children with ASD.
The application is a tool, such that parents and educators can visualize time in a way customized for each child by changing color schemes, symbols, forms, and save this information in profiles stored on a server. The visualization is formed as a full-screen timer, which can be customized to be shown as an hourglass, stopwatch, progressbar, or digital watch.
Furthermore the guardians should be able to add pictograms to the timer, to show them what they are going to do while the time is running, and what they are going to do when the time has expired.

The introduction consisted of two parts: a common part were general subjects for the whole multi project was presented, and an analysis of the problem area with a focus on the timer group as an individual.

We have learned that children with ASD are visual in they way of communication, and existing electronic communicational aiding systems are expensive. Therefore we want to build a low cost system, with Android tablets as target platform, to aid the children and their guardians in their everyday.

We, the timer group, focus on building a timer application, to help the guardians visualize time progressing for the children. This application is a part of a collection of software components, which together forms the GIRAF system.

Since there are 18 students, divided into five project groups, working on the same software system, and there is no concrete system specification to develop out from, we have chosen to use an agile development method, Scrum of Scrums, because we expect changes in system requirements during the development process.

Part II

Development

In this part we describe the whole development process from design to implementation and testing. There will be an outline of the various tools we have used during the different parts of the development process, and we will present a diary describing the sprint periods.

In chapter 6 we will describe the timer application architecture and the implementation of the most essential functionality.

In chapter 7 there will be documentation of the three tests performed on the WOMBAT system: black-box testing, acceptance test, and usability test.

3

DESIGN

The design of WOMBAT has changed throughout the project. The design chapter presents the vision of the timer application and the tools used to design and develop the product.

3.1 VISION

Since the launcher had both a child-mode and a guardian-mode at the first period of the project, this vision relies on the initial design of the launcher project.

The idea of the WOMBAT application was, that is should be a tool for educators to illustrate time for the children.

One part is the timer application as it is implemented, from which it is possible to run timers, such as an hourglass or a digital watch. This part should be available only from the guardian-mode.

The second part of the application is a timer overlay, which is launched when other applications are launched through the GIRAF launcher. When such applications are launched through guardian mode, the user should be prompted to select if there is a time limit on the application, and what the limit should be. If a time limit is chosen, the given application, i.e. a game, is run with the timer overlay showing a custom timer with the time left (see figure 4).

Figure 4: Example of how the timer overlay would look. Here there is an overlay on a game with about 40 minutes left.

If the launcher is in child-mode, and the child opens an application, the timer overlay is run according to settings chosen in the third and last part of the WOMBAT system; the settings application. The overlay can be used if a child is only allowed to play a game for 30 minutes a day, then the overlay can be customized to show the child how much time is left of the allowed time. When the time has expired, the application is automatically closed.

The third part of the application is only available from the guardian-mode in the launcher, and from this application it is possible to customize which applications should be run with the timer overlay. Also it is possible to define how the overlay for every child should look like, and what should happen when the time limit has been reached. Also it is also possible to set constrains for the applications, such as time constrains, i.e. certain applications can only be opened in a different timespan on the day, or when a specific application has been run for 30 minutes straight, the application is closed and cannot be opened before a certain "cooldown" has ended.

3.2 STORIES

These stories are fictive, and are based on the vision of the timer system, together with an interview with our contact person.
We use a fictive person in the use cases, Trine, who acts as an educator in a special kindergarden for children with ASD.

Timer Application

Trine got an Android tablet from the kindergarden yesterday, and some of the other educators suggested that she should try out the WOMBAT timer application. Trine has planned a playing session with a few of the children, and decides to use the WOMBAT application to time the session, such that the children can see when they are done playing. They have about 30 minutes to play in, so Trine opens WOMBAT on the tablet and selects a predefined hourglass set to 30 minutes. When the children are in place and ready to play, she press the start button, and the hourglass starts. When the time runs out, a "Done" screen appear, and the children can see that they are done playing.

Timer Overlay

Trine walks in the playground while the children are playing. She sees one of the boys sitting on the ground in the corner of the playground. She walks over to him. It is Casper, he had tripped over his own feet, and hurt his knee. To get Caspers mind of his knee Trine lets Casper play his favorite computer game for the next 10 minutes. She starts the game, and gets prompted to choose a profile and the amount of time the game should allow to run. She selects Casper's profile and 10 minutes. The game starts with the timer overlay showing 10 minutes left of play time, with Casper's favorite green digital clock. When the time has run out, the game closes, and Casper is again ready to play with his friends.

3.3 PROTOTYPING

Before the actual development started, we made a few drawings of our ideas (see figure 5).

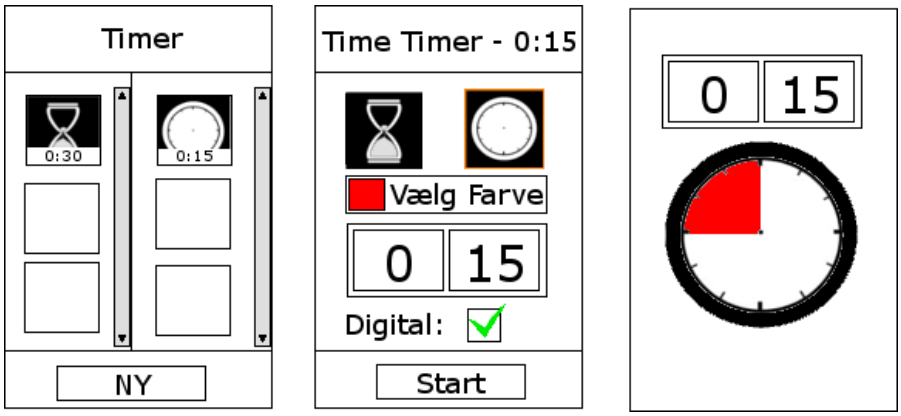


Figure 5: Drawing of the initial ideas of the timer application.

Paper Prototyping

Paper prototypes[3] has been used in the first iterations in the development process. The initial idea of the system design was drawn on paper, so that our contact person could give us some feedback on the design, before we started programming. In figure Figure 6 is a paper prototype of the menu in the timer application, and the rest of the prototypes can be found in appendix Section 10.3.

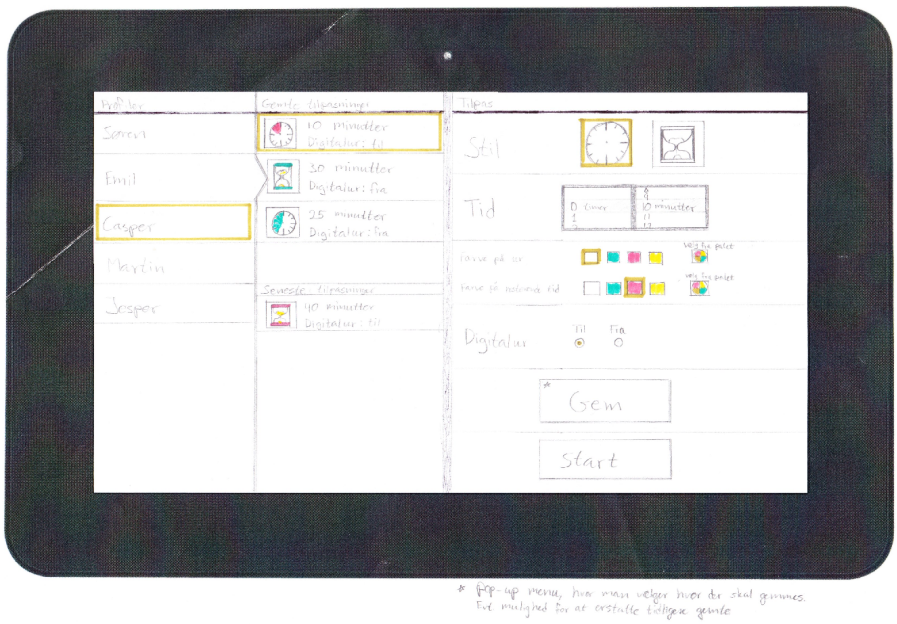


Figure 6: Scan of a paper prototype of the menu in the timer application.

Paper prototypes are produced quickly, and they capture early design ideas.

3.3.1 *Metaphors*

To enhance usability and learn-ability, we have used metaphors[3] on the buttons in the application. On the "Attach"-button, used to attach a second timer or one or two pictograms to the main timer, we have placed a paper-clip, which is known from the attach function in other programs, for example Microsoft Outlook Express. Furthermore we have used metaphors on the "Start Timer"-button, which looks like the "Play"-button known from various media players, and the "Save" and "Save As" buttons have a floppy disc icon, which is known from the save button in various word processing programs, for example Microsoft Office Word[1]. In figure 7 examples of the implemented metaphors are shown along with screenshots of other systems they are implemented in.

1 Non-free word processor developed my Microsoft

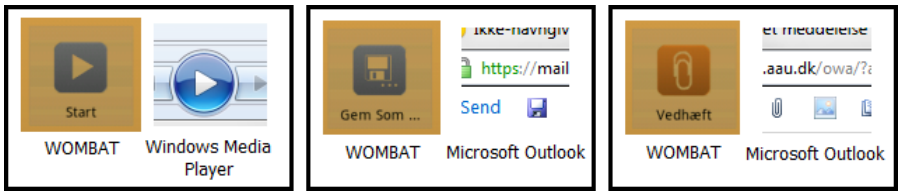


Figure 7: Metaphors implemented in WOMBAT and the original implementation.

4 DEVELOPMENT PROCESS

As stated in chapter 1.4, the development method used in this project is a modification of Scrum, which means that the development evolve through sprints. Here is a description of the six development sprints we had, and in appendix 10.4 all sprint backlogs can be found.

4.1 SPRINT WALK-THROUGH

Because it took some time for all groups to agree on a development method, we started making some design and talking to our contact person before the actual sprints started, so when the first sprint was started, we already had an idea of the functionalities we wanted to end up with.

Sprint 1

19/03 - 23/03: In the first sprint, the main objective was to set up the android project in development environment, and get the development started. At this point we were expecting to end up implementing the whole vision (see section 3.1), and a natural place to start would be the timer. We had an idea of how the design should be, and we began to implement lists for holding children and configurations.

Sprint 2

26/03 - 04/04: In the middle of this sprint the launcher group stated that they would only be able to make the guardian mode in the launcher in this semester, so we decided in the timer group, that the two last parts of the timer application, the overlay and the settings, would not be developed in this semester either, since there will be no need for them if there is no child mode in the launcher.
We continued designing and developing, and in the end of this sprint, we had the design ready for implementation. We also started to make some OpenGL development[1], as we thought that OpenGL was the best way to implement the timers.
We had a meeting with the contact person, and she suggested that when the timer in the application has exceeded, it would be possible to show two custom pictograms on the screen. Also she suggested

1 Widely used 2D/3D graphics API

that two different timers could run in parallel on the screen, if a child was used to one specific timer, but was learning how to use another type of timer. We added these to suggestions to the product backlog.

Sprint 3

10/04 - 19/04: We found out during this sprint, that it was more convenient to implement the timers using canvas and 2D drawings. The progress bar had been implemented in both OpenGL and on canvas, and the canvas version was the easiest to implement, and required fewer lines of code.

Sprint 4

23/04 - 04/05: In this sprint we finished drawing the timers, and a lot of the most vital functionalities, such as loading and saving configurations, highlight on list items, and the "Done" screen, were implemented.
During this sprint we had a lot of contact with the admin group, because we had difficulties implementing functions to load and save data to the database. We also integrated the timer with the launcher, and did some testing on the interaction between them.

Sprint 5

07/05 - 11/05: This sprint was used to do some refactoring of the code, to make it more readable and understandable. We also implemented the last things from the backlog, and did some polishing to the overall design. Furthermore we started making test design and test cases for the functionalities we wanted to test in the timer application.

Sprint 6

14/05 - 18/05: In this sprint we implemented the last critical functionality.

Besides getting done with the development, we started with the acceptance test, by letting our contact person use the timer application at the institution over a period of three days.

Sprint 7

21/05 - 25/05: This sprint was used to perform and evaluate on usability test, acceptance test, and black-box test.

5 DEVELOPMENT TOOLS

As part of a mini project in the course *Software Engineering* we analyzed the project situation, when we had started the development process, by listing strengths, weaknesses, threats, and opportunities of the project, and discussed if some of the threats and weaknesses could be eliminated, or some strengths or opportunities could be exploited at a low cost. We here list the development tools we decided to use, and the reason for choosing them, the analysis of the project can be found in Section 10.5.

PAIR PROGRAMMING

We implemented this technique because we had done some single person programming in the start of the development process, and there was a growing need for explanation when using methods the other project members had written. Also we wanted to enhance the quality of the code we wrote, and thereby minimize the time we had to spend on debugging.

The cost of this technique was, in this case, moderate, since the project group is very small, and it makes a difference when one person is not writing code. On the other hand, the code written in pairs is not likely to be re-written, because of the higher quality, which makes the benefit high.

REFACTORING

The reason refactoring was a usable technique for our project, is that we were suffering from code cluttering and code generally being difficult to understand. When debugging the system it could sometimes be difficult to know where the bug had evolved from. Since refactoring can give a better overview it would make debugging an easier task. Also this project is going to be continued by another project group next year, which requires our code to be easy to understand for the next project group to be able to smoothly proceed the development.

The cost of this technique is low, and the benefit is, for ourselves, moderate. The benefit is much higher for the project groups next year.

6 IMPLEMENTATION

In this chapter there is a description of the implementation of WOMBAT, starting with the architecture of the application. After that, we go in detail with the different parts of the WOMBAT system, from the user interface components to the back end library.

6.1 ARCHITECTURE

6.1.1 *Architecture model*

The WOMBAT application depends on the Launcher project and Oasis project, but WOMBAT is designed to function as an independent application with minor modifications to cooperate with the GIRAF launcher and Oasis. We choose to design WOMBAT as an independent application, because it was easier to modify features in the application when they were ready, instead of waiting for the features to be implemented in the GIRAF Launcher or Oasis. The dependency diagram of the WOMBAT architecture can be found in Figure 8. The WOMBAT architecture is a five layer architecture which enhances how you can perform testing and do collaborative work. The five layers consists of:

Main
: This layer is the main activity of WOMBAT which means that it initiate the whole WOMBAT application. The main layer is dependent on the Tools layer since it contains the initiating tools.

Layout
: This layer consists of the three fragments; profile fragment, configuration fragment, and customize fragment, and the custom `ArrayAdapters` which WOMBAT uses. The layout layer is depended on the Tools layer since it contains the objects and methods that is required for the layout to work. The Layout layer is also depended on the Main layer, without the Main layer the Layout layer would never be initiated. The Layout layer is depended on the Draw layer, the Draw layer delivers the methods that generates the `Views` for the timers and pictograms.

Draw
: The Draw layer contains the methods that generate the `Views` of the timers and pictograms. Draw layer is depended on the Tools layer since the Tools layers contain all the different types of objects that the Draw layer implements. The Draw layer is

also depended on the Layout layer, the Layout layer initiate the Draw layer.

Tools
: This layer consists of all the types of objects and methods that WOMBAT uses. The Tools layer is depended on the Main layer to initiate the proper objects, you can read more about this in section: 6.5.1. The Tools layer is also depended on the CRUD layer which contains the connection to the `OasisLocalDatabase`.

CRUD
: This layer is responsible for saving and loading from the `OasisLocalDatabase`, it is depended on the Tools layer because it uses the objects and methods that the Tools layer provides.

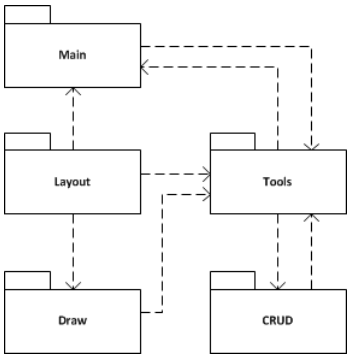


Figure 8: Dependecy diagram of WOMBAT architecture

Library

The original idea was to split the layers into two projects, that way it would be able to conduct tests outside the main layer. The first project would be an Android project with a main activity, WOMBAT. The second project would be an Android library which should contain all the back-end functionality, `TimerLib`. This is how the two projects would look like:

WOMBAT

- Main layer
- Layout layer

TIMERLIB

- Tools layer
- Draw layer
- CRUD layer

We later decided to redesign, and make three projects instead of two projects. We choose the redesign because that we are three developers in our project whom all want to conduct independent testing. The first project, WOMBAT, will stay as it originally was. The Second project is split into two projects, `TimerLib` and `DrawLib`. `TimerLib` contains the Tools layer and the CRUD layer. The `DrawLib` contains the draw layer.

WOMBAT

- Main layer
- Layout layer

TIMERLIB

- Tools layer
- CRUD layer

DRAWLIB

- Draw layer

The dependency diagram of the WOMBAT projects can be found in figure 9.

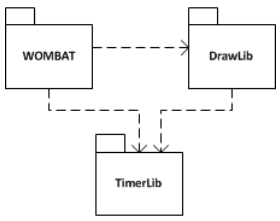


Figure 9: Dependecy diagram of WOMBAT projects

Wombat lifecycle

You can see a detailed flowchart over the WOMBAT life cycle on figure 10. The flowchart describes whatever that can occur while the WOMBAT application is running. The flowchart can help debug and understand the application if somebody chooses to develop further on WOMBAT.

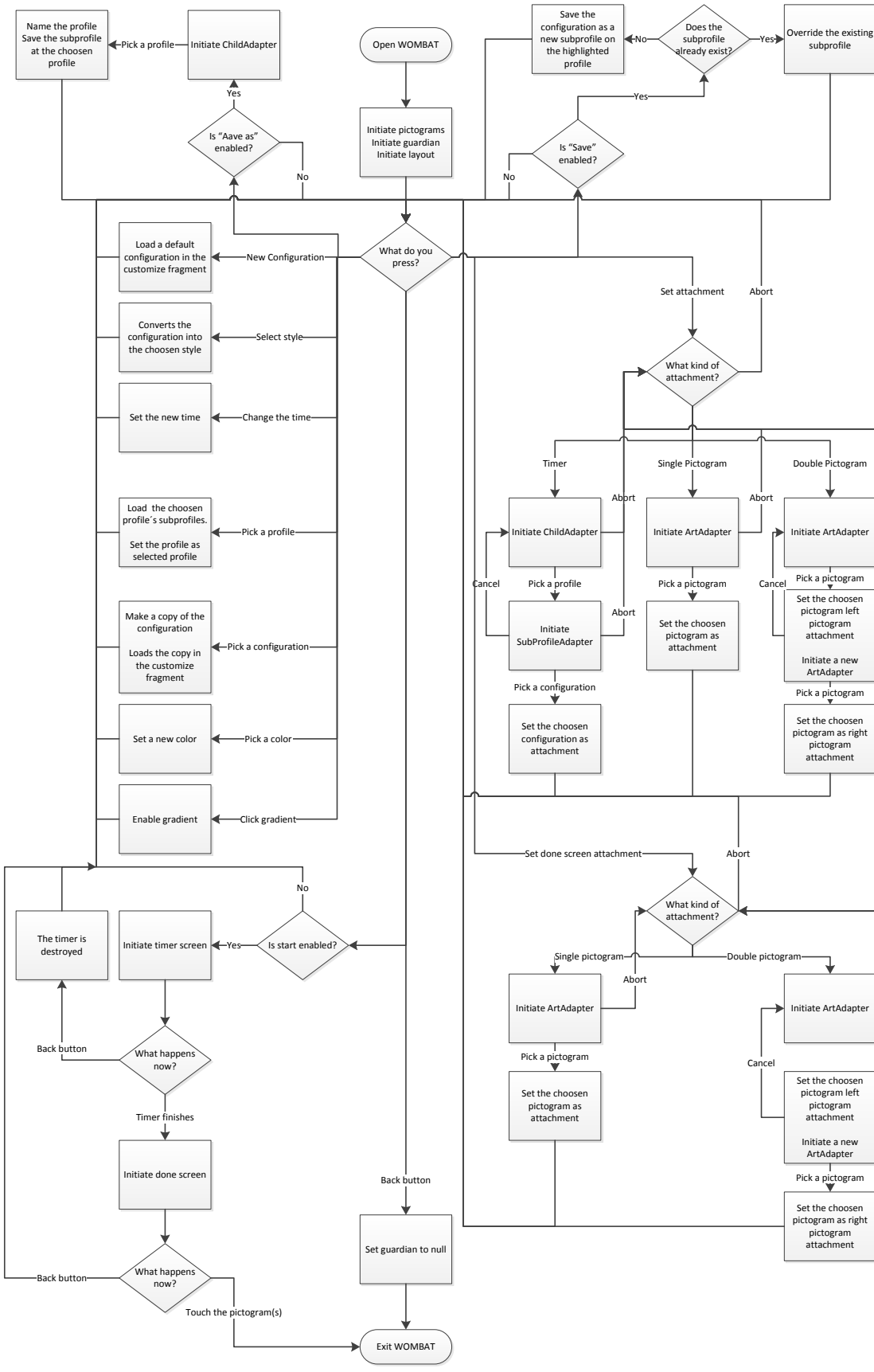


Figure 10: Flowchart of WOMBAT lifecycle

To better understand fragments, a short introduction to activities is useful.

An activity[13] is a key component for android application, and it provides a screen which the user can interact with. When initiating an Android application, a main activity is used, from where the rest of the application is initiated. An Android application can use multiple activities, each activity provides a window in which the developer can draw a user interface.

A fragment in Android is an embedded activity within another activity. According to Google, the design philosophy of fragments[5] are that they support a more dynamic and versatile design of applications on devices with larger screens, such as tablets. Google states that on these kind of devices there are more room to combine and interchange interface components, compared to a handset[5]. An example of the use of multiple fragments on a large screen compared to activities on smaller handsets, is the Gmail application from Google, seen in Figure 11 and Figure 12:

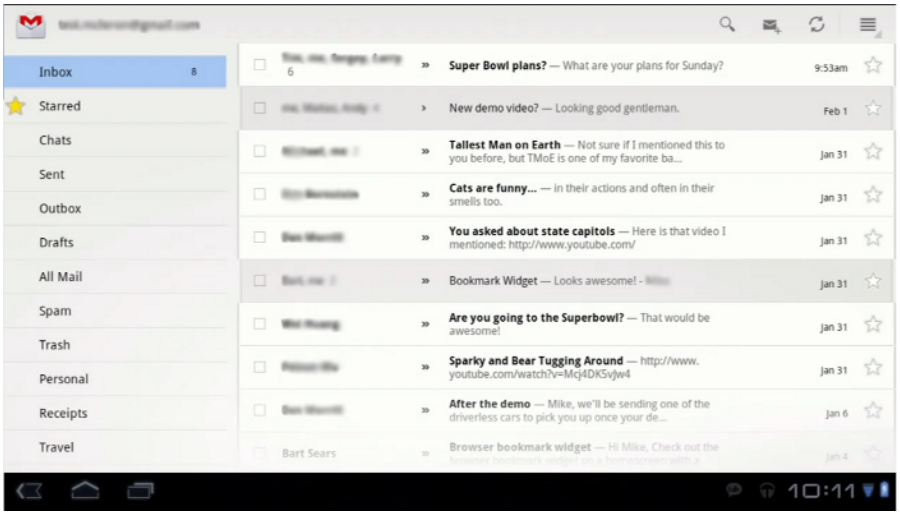


Figure 11: The Gmail application from Google on a tablet with fragment support.

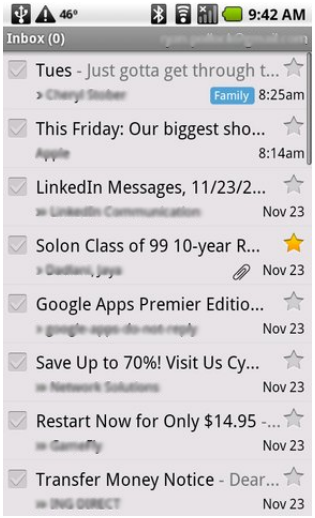


Figure 12: The Gmail application from Google on a handset without fragments

6.2.1 *Benefits and Limitations*

Fragments alone are not more powerful than activities, and fragments can not exist without activities. When combining multiple fragments in one activity, the fragments are able to create dynamic interfaces which does not require the entire activity to be reloaded when changing an object on the screen.

The Gmail application utilize this by having an overview of the email account on the left side of the device while showing specific email content on the right side, as seen in Figure 11.

Fragments can be reused in multiple activities, which means that a single fragment can be used in several ways depending on the platform. The main activity only have to declare which fragments it holds in the layout.

Fragments were introduced with the Android 3.0 platform (API version 11), and as of 3rd March 2011 Google has released the "Android Compatibility Package", a library which let Android 1.6 devices or newer support fragments [4].

6.2.2 *Fragments in WOMBAT*

In WOMBAT fragments are designed to quickly switch between multiple children and configuration templates. When developing using fragments the screen can be divided into several parts, that can update independently based on actions in the fragments. This means that, much like the Gmail application, WOMBAT can keep an overview of all children and their personal configurations, while having a detailed configuration page open at the same time, thereby avoiding having to

switch screen to load previous defined configurations. The layout of fragments in WOMBAT can be found in Figure 13.

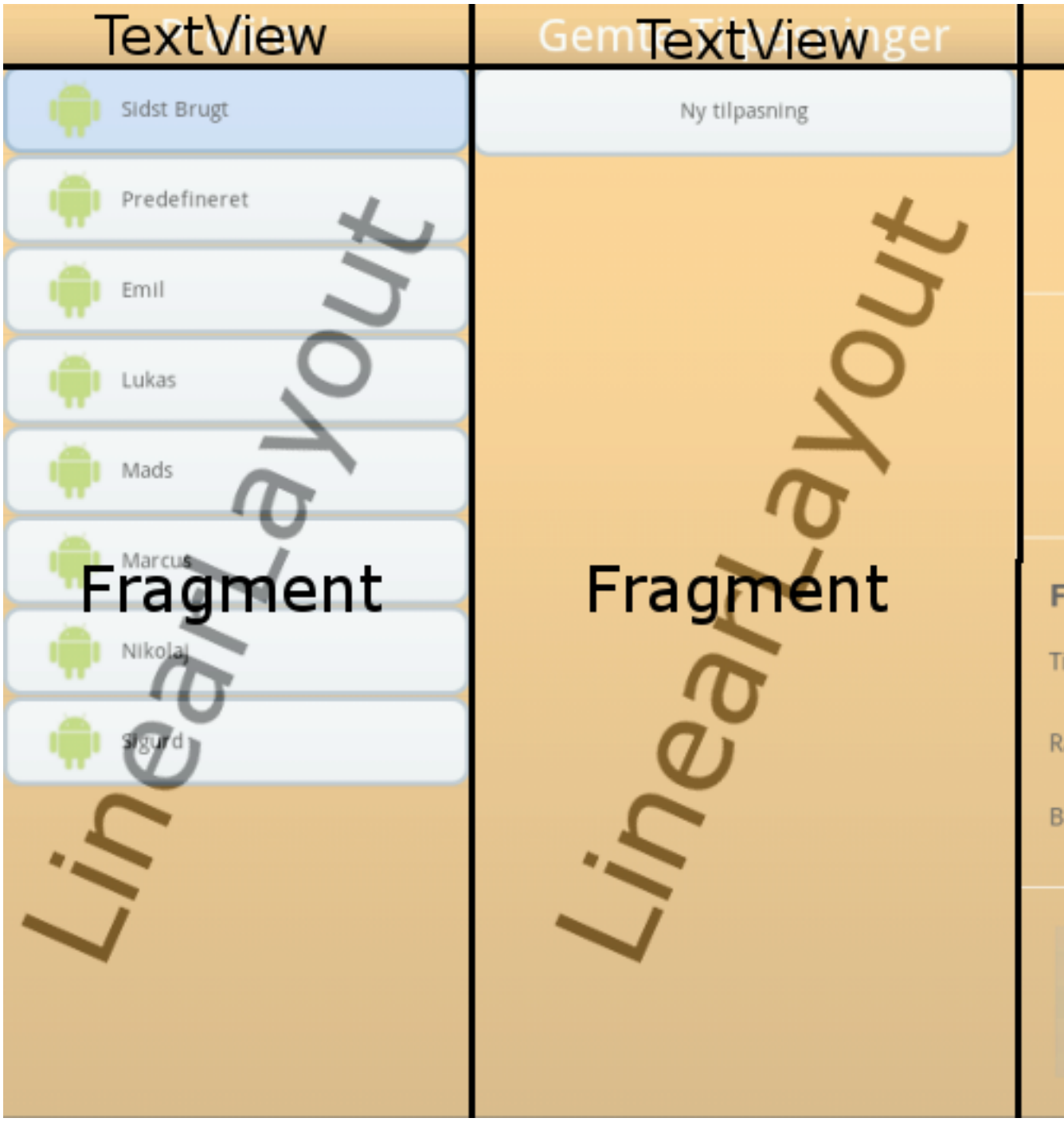


Figure 13: The layout of fragments in WOMBAT.

Having developed WOMBAT with fragments means that it is possible to create a handset version with few modifications, since it only

requires the "Android Compatibility Package" library and some new activities to handle the fragments on a smaller screen.

6.3 LISTS

Android provide a built-in `ListView`, which can be modified to match any list, that the developer wish to create, as shown in Figure 14.

Figure 14: Three examples of how `ListView` can be modified.

6.3.1 *Benefits and Limitations*

The advantage of a `ListView` is, that it can be modified to whatever the developer want it to look like. This is done by making an adapter with the changes the developer want to make to the design. An example of how to use this adapter can be seen in Figure 15.

```
View v = list_item_view;
if (v == null) {
  LayoutInflater li = (LayoutInflater) getContext().
      getSystemService(
      Context.LAYOUT_INFLATER_SERVICE);
  v = li.inflate(R.layout.profile_list, null);
}
Child c = items.get(position);
if (c != null) {
ImageView iv = (ImageView) v.findViewById(R.id.profilePic);
TextView tv = (TextView) v.findViewById(R.id.profileName);

if (iv != null) {
  iv.setImageResource(R.drawable.default_profile);
}
if (tv != null) {
  if (c.name == "Last Used") {
    tv.setText(R.string.last_used);
  } else if (c.name == "Predefined Profiles") {
    tv.setText(R.string.predefined);
  } else {
    tv.setText(c.name);
  }
}
}
```

Figure 15: The WOMBAT implementation of a `ListView` `ArrayAdapter` modified to use a list of profiles.

The `ListView` have an interface, `OnItemClickListener`, which works like any other clickable item in Android.

6.3.2 *Lists in WOMBAT*

Figure 15 is the actual implementation of the child list in WOMBAT. The difference from the `ArrayAdapter` used in the profile list and the configuration list, is that the configurations is designed to use a text field with a description of the timer. The configurations list only requires another layout to be constructed, this layout contains the extra `TextView`.

The `OnItemClickListener` of the child fragment ensures that the configuration and customize fragments are updated. Figure 16 is an example of the `OnItemClickListener` of the child fragment.

```
public void onListItemClick(ListView lv, View view, int
    position, long id) {
  // Update the fragments
  SubProfileFragment detf = (SubProfileFragment)
      getFragmentManager()
      .findFragmentById(R.id.subprofileFragment);
  CustomizeFragment custF = (CustomizeFragment)
      getFragmentManager().findFragmentById(R.id.
      customizeFragment);
  custF.setDefaultProfile();

  if (detf != null) {
    // Marks the selected profile in the guard singleton
    guard.profilePosition = position;
    guard.publishList().get(position).select();
    guard.profileID = guard.publishList().get(position).
        getProfileId();
    detf.loadSubProfiles();
  }
}
```

Figure 16: Example of the `OnClickListener` of the child fragment.

6.4 CUSTOMIZE

The customize fragment is where the configurations are generated and altered.

As stated in Chapter 2 the timers our contact person use has different sizes and colors, these customization options are essential for the timer application. The list of customization features are as follows:

- Change style of the timer.
- Change the timespan of the timer.
- Change the color of the timer and background.
- Change the color of the timer to be changing gradiently.
- Attach one or two pictograms, or a timer.
- Change the default done pictogram to one or two pictograms.
- Save the timer.
- Start the timer.

6.4.1 *Architecture of Customize*

The customize fragment consists of four main elements, the first element is the style picker where the user can change the style of

the timer. The second element is the time wheels where the user can change the timespan of the timer. The third element is the color pickers where the user can change the colors of the time left, the frame, and the background of the timer. The fourth element is the menu where the modifications like attachment is found. Figure 17 show architecture of the fragment.

Figure 17: An outline of the architecture of the customize fragment.

All functionality of the items in the customize fragment in placed in the customize class. For further development a refactoring of the entire class is advised such that all buttons in WOMBAT is handled like the `WDialog`, which can be found in Figure 6.4.1.

To be able to get an overview of the items in the customize fragment all items are assigned a method, these methods are then referenced from the `onCreate` method in Figure 18. An example of one of these methods is the style chooser in Figure 19.

```
public void onActivityCreated(Bundle savedInstanceState) {
  super.onActivityCreated(savedInstanceState);
  currSubP = new SubProfile("", "", 0xff3D3D3D, 0xffFFf0000, 0
      xffB8B8B8,
      0xff000000, 600, false);
  currSubP.save = false;
  currSubP.saveAs = false;

  /********* TIME CHOSER *********/
  initStyleChoser();

  /********* TIMEPICKER *********/
  initTimePicker();

  /********* COLORPICKER *********/
  initColorButtons();

  /********* ATTACHMENT PICKER *********/
  initAttachmentButton();

  /******** BOTTOM MENU **********/
  initBottomMenu();
}
```

Figure 18: The `onCreate` method, which calls the button initializers in the same order as they are shown in the layout.

```
private void initStyleChoser() {
  hourglassButton = (Button) getActivity().findViewById(
      R.id.houglassButton);
  hourglassButton.setOnClickListener(new OnClickListener() {

    public void onClick(View v) {
      selectStyle(formFactor.Hourglass);
    }
  });

  timetimerButton = (Button) getActivity().findViewById(
      R.id.timetimerButton);
  timetimerButton.setOnClickListener(new OnClickListener() {

    public void onClick(View v) {
      selectStyle(formFactor.TimeTimer);
    }
  });

  progressbarButton = (Button) getActivity().findViewById(
      R.id.progressbarButton);
  progressbarButton.setOnClickListener(new OnClickListener() {

    public void onClick(View v) {
      selectStyle(formFactor.ProgressBar);
    }
  });

  digitalButton = (Button) getActivity().findViewById(R.id.
      digitalButton);
  digitalButton.setOnClickListener(new OnClickListener() {

    public void onClick(View v) {
      selectStyle(formFactor.DigitalClock);
    }
  });
}
```

Figure 19: The style chooser initialization method, which utilize the `selectStyle` that changes the style of the timer and highlights the button.

Buttons in Customize

There is four kinds of buttons in the customize fragment:

- Start, save, and style buttons.
- Time picker wheels.
- Color picker.
- Attachment and done pictogram button.

Start Button

The style, save, start, and attachment buttons are default Android buttons with a picture attached on top. The differences of these buttons is the `onClick` event handler, figure 20 is the source code of the start button.

```
private void initStartButton() {
 startButton = (Button) getActivity().findViewById(
     R.id.customize_start_button);
 Drawable d;
 if (currSubP.saveAs) {
   d = getResources().getDrawable(R.drawable.thumbnail_start);
   startButton.setOnClickListener(new OnClickListener() {

     public void onClick(View v) {
       currSubP.addLastUsed(preSubP);
       guard.saveGuardian(currSubP);
       currSubP.select();
       Intent i = new Intent(
           getActivity().getApplicationContext(),
           DrawLibActivity.class);
       startActivity(i);
     }
   });
 } else {
         ...
     }
   });
 }

 startButton
 .setCompoundDrawablesWithIntrinsicBounds(null, d, null, null)
     ;
}
```

Figure 20: The source code of the start button, which sets the top image of the button to the drawable `thumbnail_start_gray`

Time Picker Wheels

The time picker is a widget[17]. All functionality of the wheel widget is handled by the widget itself and only requires the widget to be imported to Eclipse and added as a project library, which is a built-in feature in the Android development environment.

WOMBAT is limited to timers on 60 minutes, therefore we implement a functionality which sets the seconds wheel to zero whenever the minutes wheel is set to 60, this can be seen in figure 21.

```
private int previousMins;
private int previousSecs;
private void initTimePicker() {
  /* Create minute Wheel */
  mins = (WheelView) getActivity().findViewById(R.id.minPicker
      );
  mins.setViewAdapter(new NumericWheelAdapter(getActivity()
      .getApplicationContext(), 0, 60));
  mins.setCyclic(true);

  /* Add on change listeners for both wheels */
  mins.addChangingListener(new OnWheelChangedListener() {
    public void onChanged(WheelView wheel, int oldValue, int
        newValue) {
      updateTime(mins.getCurrentItem(), secs.getCurrentItem())
          ;

      if (mins.getCurrentItem() == 60) {
        previousMins = 60;
        previousSecs = secs.getCurrentItem();

        secs.setCurrentItem(0);
        secs.setViewAdapter(new NumericWheelAdapter(
            getActivity()
            .getApplicationContext(), 0, 0));
        secs.setCyclic(false);
      } else if (previousMins == 60) {
        secs.setViewAdapter(new NumericWheelAdapter(
            getActivity()
            .getApplicationContext(), 0, 60));

        secs.setCurrentItem(previousSecs);
        secs.setCyclic(true);
        previousMins = 0;
      }
    }
  });
}
```

Figure 21: The time picker wheels.

Color Picker

Just like the time picker wheels we utilize the widget functionality in Android to implement a widget called AmbilWarna[1], to handle the color picker[12]. The color picker widget is a custom dialog which returns the color picked by the user in the widget, the implementation of the color picker widget can be found in Figure 22.

1 AmbilWarna means "Take Color" in Indonesian

```
colorGradientButton2 = (Button) getActivity().findViewById(
    R.id.gradientButton_2);
setColor(colorGradientButton2.getBackground(), currSubP.
    timeSpentColor);
colorGradientButton2.setOnClickListener(new OnClickListener()
    {
  public void onClick(View v) {
    AmbilWarnaDialog dialog = new AmbilWarnaDialog(getActivity
        (),
        currSubP.timeSpentColor, new OnAmbilWarnaListener() {
      public void onCancel(AmbilWarnaDialog dialog) {
      }

      public void onOk(AmbilWarnaDialog dialog, int color) {
        currSubP.timeSpentColor = color;
        setColor(colorGradientButton2.getBackground(),
            currSubP.timeSpentColor);
      }
    });
    dialog.show();
  }
});
```

Figure 22: The color picker widget implemented on the second "Time Left" color button

Attachment Button

The attachment buttons are implemented like the start button, the difference is that they make use of a custom dialog in WOMBAT. This custom dialog, `WDialog`, is created because the standard dialogs in Android are very different from the rest of the WOMBAT design. Also the standard dialogs limit the programmer to use exactly the buttons defined in the dialog API.
The custom dialog is implemented because the standard dialogs are inconsistent with the WOMBAT design and the custom dialogs are much more versatile when adding buttons. Figure 23 is an example of how a dialog could be specified.

```
final WDialog attachment1 = new WDialog(getActivity(),
  R.string.attachment_dialog_description);

ModeAdapter adapter = new ModeAdapter(getActivity(),
  android.R.layout.simple_list_item_1, mode);

attachment1.setAdapter(adapter);

attachment1.addButton(R.string.cancel, 1,
  new OnClickListener() {

  public void onClick(View argo) {
    attachment1.cancel();
  }
});
```

Figure 23: Initialization of the dialog, that appears when the attachment button is clicked, the `OnItemClickListener` is not shown here because of lack of space on the page.

6.5 BACK-END LIBRARY

The libraries in WOMBAT each control the storage and drawing functionality of the application. These functionalities have been imported as Android project libraries, because this gives the developer the ability to create an external project and test the library, without being dependent on a working main project.

Furthermore is a benefit that the libraries works like modules which can be changed to handle e.g. OpenGL instead of canvas in the `DrawLib`.

6.5.1 *TimerLib*

`TimerLib` contains the functionality that store and manage all the different objects that WOMBAT uses. The main goal of `TimerLib` is to provide a set of simple methods and objects which the rest of WOMBAT can use without knowledge of how `TimerLib` works. This goal is achieved by being strict about visibility of attributes and methods.

Classes

The `TimerLib` classes are described in the following.

GUARDIAN

The `Guardian` class is an object that represent the guardian in our system. The idea of the GIRAF system is that only one guardian can be logged in at a given time, that is why the `Guardian` class

use a singleton pattern to enforce WOMBAT to never have more than one guardian logged into the application. There are two `getInstance` methods in the `Guardian` class, and when initiating the `Guardian` class for the first time, it is important to use the method seen in code snippet 24. The method will always return the same guardian due to the singleton pattern.

```
1  private static Guardian _instance = null;

3  public static Guardian getInstance(long m_childId, long
       m_guardianId, Context c, ArrayList<Art> artList){
     if(_instance == null){
5      _instance = new Guardian();
       _instance.ArtList = artList;
7      TimerHelper help = new TimerHelper();
       _instance.profileID = m_childId;
9      _instance.guardianId = m_guardianId;
       _instance.m_context = c;
11     _instance.oHelp = new Helper(c);
       long appId = _instance.findAppId();
13     _instance.guardianId = _instance.findGuardianId();
       _instance.createChildren();
15     _instance.crud = new CRUD(appId, c);
       _instance.crud.loadGuardian(_instance.guardianId);
17     help.loadPredef();
       //_instance.crud.initLastUsed(_instance.m_oGuard.getId()
           );
19     _instance.publishList();
       //crud.retrieveLastUsed(m_guardianId);
21   }
       return _instance;
23 }
```

Figure 24: Code snippet to initiate the `Guardian` object.

Whenever the back button is pressed while WOMBAT is on the main screen, WOMBAT calls the reset method which resets the `Guardian` and exits the application. The reset method can be seen in code snippet 25. The reset method sets the `Guardian` instance to null, such that next time the `getInstance` method is called it will create a new `Guardian` object, we are doing so because Java has a garbage collector and therefore we can not manually destroy an object.

```
public void reset(){
    _instance = null;
}
```

Figure 25: Code snippet to reset the `Guardian` object.

The `Guardian` class creates a `Guardian` if WOMBAT does not receive a valid `Guardian` id, this is to ensure that WOMBAT can run outside the GIRAF launcher. In code snippet 26 you can see the method which checks and creates a new `Guardian`.

```
private long findGuardianId() {
  if(guardianId != -1){
    // Does the original guard exist
    m_oGuard = oHelp.profilesHelper.getProfileById(
        guardianId);
  } else {
    m_oGuard = null;
  }
  // If not, try the default guard
  if(m_oGuard == null){
    for (Profile p : oHelp.profilesHelper.getProfiles()) {
      if(p.getFirstname().equals("John") && p.getSurname().
          equals("Doe")){
        m_oGuard = p;
        break;
      }
    }
    // If thats not valid either, make the default guard
    if(m_oGuard == null){
      m_oGuard = new Profile("John", "Doe", null, 1,
          88888888, null, null);
      m_oGuard.setId(oHelp.profilesHelper.insertProfile(
          m_oGuard));
      oHelp.appsHelper.attachAppToProfile(m_app, m_oGuard);
      oHelp.profilesHelper.setCertificate("jkkxla ... hfwsh"
          , m_oGuard);
    }
  }
  return m_oGuard.getId();
}
```

Figure 26: Code snippet to create a default `Guardian`.

The `Guardian` class also creates a list of profiles if the current `Guardian` has no relation to any profiles. The code snippet **??** shows how this is done.

```
1  private void createChildren() {
     if(oHelp.profilesHelper.getChildrenByGuardian(m_oGuard).
         isEmpty()){
3      List<String> names = new ArrayList<String>();
       names.add("Sigurd");
5      names.add("Marcus");
       names.add("Emil");
7      names.add("Lukas");
       names.add("Mads");
9      names.add("Nikolaj");

11     for (String s : names) {
         Profile newProf = new Profile(s, " ", null, 3,
             99999999, null, null);
13       newProf.setId(oHelp.profilesHelper.insertProfile(
             newProf));
         oHelp.profilesHelper.attachChildToGuardian(newProf,
             m_oGuard);
15       oHelp.appsHelper.attachAppToProfile(m_app, newProf);
       }
17   }
   }
```

Figure 27: Code snippet to create default profiles.

The `Guardian` class generates an list, of the object type `Child`, from three lists; last used, predefined, and the profiles related to the `Guardian`. In code snippet 28 you can see how this list is generated. It is important to know that any changes made on this list will not be saved. Last used is always in the top of the list and right after is the predefined and after that the profiles are listed alphabetically.

```
public ArrayList<Child> publishList(){
if(_sortedList == null){
  _sortedList = new ArrayList<Child>();
}
_sortedList.clear();
Child lastUsedChild = new Child("Last Used");
lastUsedChild.setProfileId(-3);
lastUsedChild.SubProfiles().addAll(reverse(lastUsed()));
lastUsedChild.setLock();
lastUsedChild.lockDelete();
_sortedList.add(lastUsedChild);

Child predefChild = new Child("Predefined Profiles");
predefChild.setProfileId(-2);
Collections.sort(predefined());
predefChild.SubProfiles().addAll(predefined());
predefChild.setLock();
predefChild.lockDelete();
_sortedList.add(predefChild);
if(_guard != null){
  Collections.sort(_guard);
  for(Child p : _guard){
    Collections.sort(p.SubProfiles());
  }
  _sortedList.addAll(_guard);
}
return _sortedList;
}
```

Figure 28: Code snippet to generate the list containing last used, predefined, and the profiles related to the `Guardian`.

CHILD

The `Child` class is an object that represent either last used, predefined, or a profile. The `Child` object, no matter what it represents, has a collection of `SubProfiles`, which is a list of configurations. The user is not allowed to save or delete from the last used and predefined list, the `Child` class therefore provides two kinds of locks, a save lock and a delete lock. When the save lock is true, you cannot save a configuration on the `Child` object. When the delete lock is false, you cannot delete configurations from the `Child` object. Both delete and save lock is read only for projects outside `TimerLib`, this is because the WOMBAT should never lock any `Child` objects that are loaded from the `OasisLocalDatabase`. The lock methods can be seen in code snippet 29.

```
/**
 * Used to check if you can save on a specific Child
 * @return boolean, if true, you cannot save on the child.
     if false you can save.
 */
public boolean getLock(){
  return _lock;
}
/**
 * Enables the lock
 */
void setLock(){
  this._lock = true;
}
/**
 * Enable delete lock, so you cannot delete from a certain
     child.
 * This will always be used on lastUsed and predefined.
 */
void lockDelete(){
  _deleteCheck = false;
}
/**
 * Used to check if you can delete a subprofiles on a
     specific child
 * @return boolean, true = you can delete: false = you
     cannot delete.
 */
public boolean deleteCheck(){
  return _deleteCheck;
}
```

Figure 29: Code snippet of how the locks work.

The `Child` class contains a select method that selects a certain profile so it can be reused later. The `select` method is called whenever the user highlights a profile in the WOMBAT application. To use a selected profile, a method, `getChild`, is called from the `Guardian` object. The idea of this method is to make it easier to work with the same `Child` object across multiple projects.

The `Child` class provides methods for saving and removing. The `save` method saves a certain `SubProfile` either as a new `SubProfile` or overrides a old `SubProfile`, the `save` method calls `saveChild` from `Guardian` to make sure it is saved in the `OasisLocalDatabase`. The `remove` method removes a certain `SubProfile` from the `Child` and calls `removeSubprofileFromProfileId` from `Guardian` to delete the `SubProfile` from the `OasisLocalDatabase`. You can see `save` and `remove` method in code snippet 30.

```
  public SubProfile save(SubProfile p, boolean override){

    if(!override){
      p.setDB_id(getNewId());
      p.setId(guard.getId());
    }
      this.SubProfiles().add(p);
      guard.saveChild(this, p);

    return p;
  }

  public void remove(SubProfile p) {
      _profileList.remove(p);
      guard.crud.removeSubprofileFromProfileId(p, this.
          getProfileId());
}
```

Figure 30: Code snippet of how to `save` and `remove` `SubProfile`.

SUBPROFILE

The `SubProfile` class is a super class for all the different kinds of timers which WOMBAT supports. The idea of having a super class that represent the timers is to make it possible to have a collection of all types of timers. Originally the `SubProfile`, and the classes which inherits from it, should contain methods to draw the timers, find more in Section 6.1. See code snippet 31 for an example of how `SubProfile`, and classes which inherits from it, was meant to work.

```
Hourglass hourglass = new Hourglass("Timeglas - 30 sek", "
    Timeglas - (0:30)", 0xff3D3D3D, 0xffffffff, 0xffB8B8B8, 0
    xff000000, 30, false);
    ProgressBar progressbar = new ProgressBar("ProgressBar -
        30 sek", "ProgressBar - (0:30)", 0xff3D3D3D, 0
        xffffffff, 0xffB8B8B8, 0xff000000, 30, false);
    DigitalClock digitalclock = new DigitalClock("DigitalClock
        - 30 sek", "DigitalClock - (0:30)", 0xff3D3D3D, 0
        xffffffff, 0xffB8B8B8, 0xff000000, 30, false);
    TimeTimer timetimer = new TimeTimer("Ur - 30 sek", "Ur -
        (0:30)", 0xff3D3D3D, 0xffffffff, 0xffB8B8B8, 0
        xff000000, 30, false);
    ArrayList<SubProfile> timers = new ArrayList<SubProfile>()
        ;
    timers.add(hourglass);
    timers.add(progressbar);
    timers.add(digitalclock);
    timers.add(timetimer);
    for(SubProfile sp : timers){
      View v = sp.draw();
    }
```

Figure 31: Code snippet of how `SubProfile` was meant to work.

All `SubProfiles` got two different ids. The first id is used internal in the `TimerLib`, this id is used to check if the `SubProfile` already exists in the last used list and check if the `SubProfile` already exist on a certain `Child` object and thereby overriding that `SubProfile`. The internal id is provided by the `Guardian` class. The other id matches the `SubProfile`'s id in the `OasisLocalDatabase`, the id is used for deleting a `SubProfile` in the `OasisLocalDatabase`.

WOMBAT loads a copy of a configuration into the customize fragment. The idea of using a copy is because, we want to allow the user to change options on a configuration and start the timer without saving. If the user saves the modified configuration, this will replace the original configuration.

It is possible to change the type of a `SubProfile` by using one of the four object convert methods. You can see these four methods in code snippet 32.

```
public SubProfile toHourglass() {
  Hourglass form = new Hourglass(this.name, this.desc, this.
      bgcolor, this.timeLeftColor, this.timeSpentColor, this
      .frameColor, this._totalTime, this.gradient);
  form.setId(this.getId());
    form.setAttachment(this._attachment);
    form.setDoneArt(this._doneArt);
  return form;
}

public SubProfile toProgressBar() {
  ProgressBar form = new ProgressBar(this.name, this.desc,
      this.bgcolor, this.timeLeftColor, this.timeSpentColor,
       this.frameColor, this._totalTime, this.gradient);
  form.setId(this.getId());
    form.setAttachment(this._attachment);
    form.setDoneArt(this._doneArt);
  return form;
}

public SubProfile toTimeTimer() {
  TimeTimer form = new TimeTimer(this.name, this.desc, this.
      bgcolor, this.timeLeftColor, this.timeSpentColor, this
      .frameColor, this._totalTime, this.gradient);
  form.setId(this.getId());
    form.setAttachment(this._attachment);
    form.setDoneArt(this._doneArt);
  return form;
}

public SubProfile toDigitalClock() {
  DigitalClock form = new DigitalClock(this.name, this.desc,
       this.bgcolor, this.timeLeftColor, this.timeSpentColor
      , this.frameColor, this._totalTime, this.gradient);
  form.setId(this.getId());
    form.setAttachment(this._attachment);
    form.setDoneArt(this._doneArt);
  return form;
}
```

Figure 32: Code snippet of the four object converters.

Whenever a configuration is started it will be added to the last used list, if the configuration already exist in the last used list, it will be removed and added again, that way we ensure that the configuration in the last used list is the newest version of the configuration, and it is placed in the top of the last used list. You can see the method for adding a configuration to the last used list in code snippet 33.

```
public void addLastUsed(SubProfile oldProfile){
  if(oldProfile == null){
    guard.addLastUsed(this);
  } else {
  this._id = oldProfile._id;
  this.refPro = oldProfile.DB_id;
  long ref = 0;
  for(Child c : guard.Children()){
    for(SubProfile p : c.SubProfiles()){
      if(p.getId() == this.getId()){
        ref = c.getProfileId();
      }
    }
  }
  for(SubProfile p : guard.predefined()){
    if(p.getId() == this.getId()){
      ref = -2;
    }
  }
  this.refChild = ref;
  guard.addLastUsed(this);
  }
}
```

Figure 33: Code snippet of how to add a configuration to the last used list.

A configuration may contain an attachments which will be placed next to the active timer. Furthermore it is possible to change the default done screen pictogram, which is also treated as an attachment. You can read more about the `Attachment` class later in this section.

`OasisLocalDatabase` saves the configuration as a hashmap, for that `SubProfile` provides a method for generating a hashmap with all the needed attributes. You can see this method in code snippet 34.

```
public HashMap<String, String> getHashMap(){
  HashMap<String, String> map = new HashMap<String, String
      >();
  map.put("db_id", String.valueOf(this.getDB_id()));
  map.put("type", this.formType().toString());
  map.put("Attachment", String.valueOf(this._AttaBool));
  map.put("Name", this.name);
  map.put("desc", this.desc);
  map.put("bgcolor", String.valueOf(this.bgcolor));
  map.put("timeLeftColor", String.valueOf(this.timeLeftColor
      ));
  map.put("timeSpentColor", String.valueOf(this.
      timeSpentColor));
  map.put("frameColor", String.valueOf(this.frameColor));
  map.put("totalTime", String.valueOf(this.get_totalTime()))
      ;
  map.put("gradient", String.valueOf(this.gradient));
  map.put("save", String.valueOf(this.save));
  map.put("saveAs", String.valueOf(this.saveAs));
  map.put("refChild", String.valueOf(this.refChild));
  map.put("refPro", String.valueOf(this.refPro));
  map.put("timeKey", String.valueOf(this.timeKey));
  if(this._doneArt != null){
    map.put("doneArtType", String.valueOf(this._doneArt.
        getForm()));
    switch(this._doneArt.getForm()){
    case SingleImg:
      map.put("doneArtPic", String.valueOf(this._doneArt.
          getImg().getId()));
      map.put("doneArtLeftPic", String.valueOf(-1));
      map.put("doneArtRightPic", String.valueOf(-1));
      break;
    case SplitImg:
      map.put("doneArtPic", String.valueOf(-1));
      map.put("doneArtLeftPic", String.valueOf(this._doneArt
          .getLeftImg().getId()));
      map.put("doneArtRightPic", String.valueOf(this.
          _doneArt.getRightImg().getId()));
      break;
    }
  } else {
    map.put("doneArtType", String.valueOf(formFactor.
        undefined));
    map.put("doneArtPic", String.valueOf(-1));
    map.put("doneArtLeftPic", String.valueOf(-1));
    map.put("doneArtRightPic", String.valueOf(-1));
  }
  // TODO: Attachment this needs some testing and remember
      to add it to the crud!!!!
  if(this._AttaBool){
    map = this._attachment.getHashMap(map);
  } else {
    Attachment tempAttachment = new Attachment();
    map = tempAttachment.getHashMap(map);
  }
  return map;
}
```

Figure 34: Code snippet for generating a hashmap with the needed attributes

TIMETIMER - HOURGLASS - PROGRESSBAR - DIGITALCLOCK

These four classes inherit from `SubProfile`, and it currently the only type of timers WOMBAT supports.

ATTACHMENT

The `Attachment` class is a super class and does only contain the base methods of the inherited classes. The idea of having a super class is for making `TimerLib` more dynamic.

TIMER

The `Timer` class inherits from the `Attachment` class. The `Timer` class represent a `SubProfile` as an attachment. The `Timer` class were made to simplify the attachment methods and only display the methods other projects might need. The `Timer` class supports all types of `SubProfile` objects. The `Timer` class contains a method which returns a hashmap, that is used when saving a configuration with an attachment into the `OasisLocalDatabase`. You can see the method in code snippet 35.

```
1  public HashMap getHashMap(HashMap map){
     //Defines what kind of attachment it is
3    map.put("AttachmentForm", String.valueOf(this.getForm()));

5    //Timer
     map.put("timerForm", String.valueOf(this._form));
7    map.put("_bgColor", String.valueOf(this._bgColor));
     map.put("_frameColor", String.valueOf(this._frameColor));
9    map.put("_timeLeftColor", String.valueOf(this.
         _timeLeftColor));
     map.put("_timeSpentColor", String.valueOf(this.
         _timeSpentColor));
11   map.put("_gradient", String.valueOf(this._gradient));

13   //SingleImg
     map.put("singleImgId", String.valueOf(-1));
15
     //SplitImg
17   map.put("leftImgId", String.valueOf(-1));
     map.put("rightImgId", String.valueOf(-1));
19
     return map;
21 }
```

Figure 35: Code snippet of how to generate a hashmap of an attachment.

SINGLEIMG

The `SingleImg` class inherit from `Attachment`. This class represents one object of the type `Art`, you can read more about the `Art` class later in this section. The `SingleImg` object can be set either as an attachment next to a timer, or as a pictogram for the

done screen. SingleImg contains a hashmap method with same structure as the one in the Timer class.

SPLITIMG

The SplitImg class inherit from Attachment. This class represents two objects of the type Art. The SplitImg object can be set either as an attachment next to a timer or as pictograms for the done screen. SplitImg contains a hashmap method with same structure as the one in the Timer class.

ART

The Art class is used for generating pictograms as objects in TimerLib. The Art class constructor takes three arguments; The first argument is the path to the pictogram, the path is an integer that is the id from the resource class in Android. The second argument is a caption which is a string, that is used to add text to a pictogram. The last argument is a unique identifier as an integer. The id need to be the position it got in the ArtList from Guardian. In future development, this id should be set by the TimerLib and not by the WOMBAT project. If one was to implement sounds into WOMBAT, the Art class would be the place to do it.

FORMFACTOR

The formFactor class is a enum class which is used to manage object types. It contains nine enums which are; undefined, SubProfile, Hourglass, TimeTimer, ProgressBar, DigitalClock, Timer, SingleImg, and SplitImg. They all represent their own kind of object.

TIMEHELPER

The TimeHelper class contains two functions; the first function is its main function, to generate the predefined configurations. The second function is a method which generates test data.

CRUD

The CRUD class responsibility is to create, retrieve, update, and delete Guardian, Child, and SubProfile objects in the OasisLocalDatabase. The CRUD class is only called from within the Guardian class. This is done to make it easy to change the way WOMBAT create, retrieve, update, and delete objects, if WOMBAT was ever to be released as a fully independent application.

Class diagram

You can see a simplified class diagram on figure: 36.

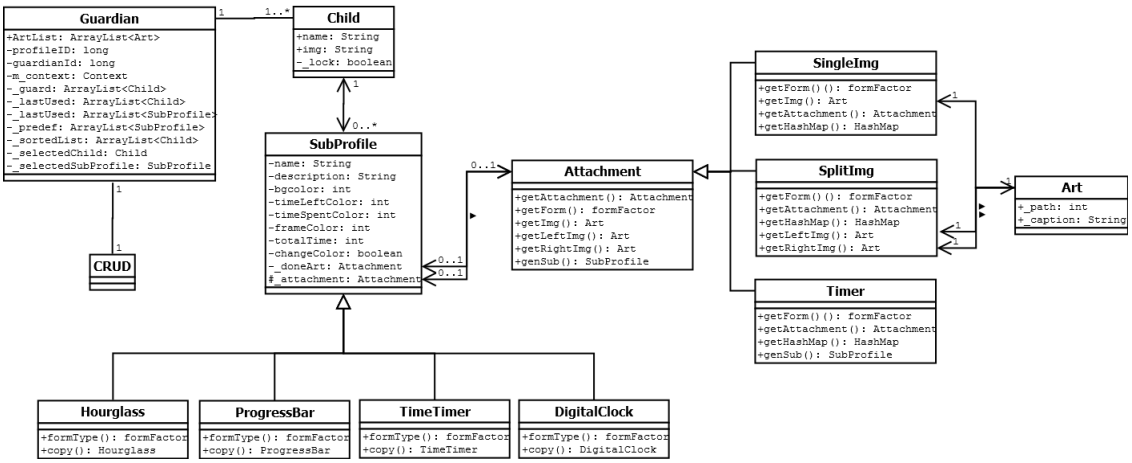


Figure 36: Dependecy diagram of WOMBAT projects

6.5.2 *DrawLib*

`DrawLib` contains the functionality that draws timers according to the configuration selected in the main activity. The main functionality in the `DrawLib` is that it can be either full screen, if there is no timer or pictogram(s) attached, or it can be split screen if there is a timer or pictogram(s) attached. This is done by creating the view layout programmatically instead of a static layout, figure 37 is an example of this functionality with a full screen timer, figure 38 is an example of the functionality with a split screen timer. In both figures the method `genDrawView` is a method which returns the draw view, that matches the timer specified in customization, figure 39 is a code example of `genDrawView`.

```
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  requestWindowFeature(Window.FEATURE_NO_TITLE);
  View main_layout = findViewById(android.R.id.content).
      getRootView();
  main_layout.setSystemUiVisibility(View.STATUS_BAR_HIDDEN);
  Guardian guard = Guardian.getInstance();
  SubProfile sub = guard.getSubProfile();

  // Get display size and store it in static variables
  WindowManager wm = (WindowManager) getSystemService(Context.
      WINDOW_SERVICE);
  Display disp = wm.getDefaultDisplay();
  frameHeight = disp.getHeight();
  frameWidth = disp.getWidth();

  if (sub.getAttachment() == null) {
    /* Set the drawing class (which extends View) as the
        content view */
    View v = genDrawView(sub,frameWidth);
    v.setKeepScreenOn(true);
    setContentView(v);
  }
...
}
```

Figure 37: Code snippet of how `DrawLib` sets just one time `View`.

```
LinearLayout frame = new LinearLayout(this);
frame.setKeepScreenOn(true);
GradientDrawable gd = new GradientDrawable(GradientDrawable.
    Orientation.TOP_BOTTOM, new int[] {sub.bgcolor, 0xFF000000
    });

...

switch(sub.getAttachment().getForm()){
case Timer:
  frameWidth = frameWidth/2;

  firstView = genDrawView(sub, frameWidth);
  frame.addView(firstView, frameWidth, frameHeight);

  secondView = genDrawView(sub.getAttachment().genSub(),
      frameWidth);
  frame.addView(secondView, frameWidth, frameHeight);
  break;
case SingleImg:
  ...
case SplitImg:
  ...
}

setContentView(frame);
```

Figure 38: Code snippet of how `DrawLib` sets two timers in one `View`

```
private View genDrawView(SubProfile sub, int frameWidth) {
  switch (sub.formType()) {
  case ProgressBar:
    return new DrawProgressBar(getApplicationContext(), sub,
        frameWidth);
  case Hourglass:
    return new DrawHourglass(getApplicationContext(), sub,
        frameWidth);
  case DigitalClock:
    return new DrawDigital(getApplicationContext(), sub,
        frameWidth);
  case TimeTimer:
    return new DrawWatch(getApplicationContext(), sub,
        frameWidth);
  default:
    return null;
  }
}
```

Figure 39: Code snippet of the `genDrawView`, which generates the draw `View` matching the timer

The timers in `DrawLib` are drawed with the Android `Canvas` object, this could have been done by creating and modelling an object in OpenGL. The major difference between the two approaches is that OpenGL is used to draw three dimensional objects and Android `Canvas` draws in two dimensions. Drawing on a canvas is like drawing on coordinates, while drawing in OpenGL varies in the way that it will be drawn in triangles. Since all timers in WOMBAT can be modelled in two dimensions, the benefits of OpenGL compared to `Canvas` is that one would be able to move the camera around, change the zoom, or change the lighting in OpenGL.

All timers are classes which inherits the `View` class, this class draw in a method called `onDraw`, `onDraw` is called when the class is initialized and afterwords when the method `invalidate()` is invoked. The timers draw inside the `onDraw` method, as seen in figure 40.

```
protected void onDraw(Canvas c) {
  super.onDraw(c);

  ... // Initialization of time + Draw Background and frame

  /* Draw the backgroundcolor inside the frame */
  paint.setColor(background);
  r.set(r.left + 2, r.top + 2, r.right - 2, r.bottom - 2);
  c.drawRect(r, paint);

  /* Draw the timespent color (on the right) on top of the
      timeleft */
  paint.setColor(timespent);
  r.set(left + 3, top + 3, left + width - 3, top + height - 3)
      ;
  c.drawRect(r, paint);

  if (endTime >= System.currentTimeMillis()) {
    timenow = endTime - System.currentTimeMillis();
    double percent = (timenow) / totalTime;

    paint.setColor(timeleft2);
    r.set((int) ((left + 3) + ((width - 5) * (1-percent))),
        top + 3, (left + 3) + width - 5, top
        + height - 3);
    c.drawRect(r, paint);

    /* Draw the gradient color */
    ...

    /*************** IMPORTANT **************/
    /* Recalls Draw! */
    invalidate();
  } else {
    paint.setColor(timespent);
    r.set(left + 3, top + 3, left + width - 3, top + height -
        3);
    c.drawRect(r, paint);
  }
}
```

Figure 40: Code snippet of the `onDraw` method of the progress bar.

Drawing can be time consuming, to optimize the timers, the background and frame is painted and stored as a bitmap file at the initialization of the drawing, seen in **??**. The stored bitmap can then be redrawed on the canvas without having to be recalculated, this is specially useful when drawing the digital watch since all numbers has to be redrawn every time the view is evaluated. With the bitmap it is possible to draw the numbers once and then blank out the lines not needed when evaluating.

```
/* Initialize a bitmap with the "standard" drawings */
bitmap = Bitmap.createBitmap(frameWidth, frameHeight, Bitmap.
    Config.ARGB_8888);
Canvas c = new Canvas(bitmap);

/* Fill the canvas with the background color */
...

y = (frameHeight - numHeight) / 2;

/* Draw first number */
x = numWidth/2 + numWidth;
c.drawPath(drawNumberPath(8, x, y), paint);

/* Draw second number */
x = numWidth/2 + numWidth * 2 + (numSpace);
c.drawPath(drawNumberPath(8, x, y), paint);

...
```

Figure 41: Code snippet of how the background and frame is stored as a bitmap.

7

TEST

The WOMBAT application is tested through dynamic black box testing, which means that the application is executed and the functionality is tested through the user interface, without knowledge of the code. The test design is conducted on the multi projects plans to outsource test to each other. Since the multi project groups agreed on not having the spare time to familiarize themselves with unknown tests, did we execute the tests ourselves.

7.1 TEST DESIGN

The test designs are split into four schemes, which are listed in this section:

- Save and Load
- Last used is updated correct
- Highlight is working
- Deviation in time on done activity

IDENTIFIER Save and load.

FEATURE Saving and loading configurations.

APPROACH NB! Check if the configuration has changed in the database by closing the application and resetting the memory.

- Create a configuration in three ways:
  - Click "New Template" and click "Save As".
  - Edit a current configuration and click "Save As".
  - Check if it is possible to "Save As" a configuration in "Predefined" or "Last Used".
  - Check if loaded settings are the same as the when they were saved.
- Edit and save (by clicking "Save") each of the configuration:
  - Check if it is possible to save configurations in "Predefined" or "Last Used".
  - Check if they have the settings they were saved with.
  - Check if there is any duplicates of any of the configurations.
  - Check if any other configuration was changed while editing.

TEST CASE ID

1. Check save as functionality - *saveAs#1*
2. Check save functionality - *save#1*

PASS/FAIL CRITERIA Pass

- It is possible to create a new configuration by clicking "New Template" and "Save As".
- It is possible to make a new configuration by clicking "Save As" on any configuration.
- No profiles in "Last Used" or "Predefined" is editable.
- It is not possible to save new profiles to "Last Used" or "Predefined".

Fail

- If any of the above does not hold.
- When saving with "Save" the configuration is being duplicated.
- When saving another configuration is being altered.

IDENTIFIER Last used is updated correct

FEATURE When a timer has been used, it should lie in the top of the list of last used configurations.

APPROACH

1. Start any timer
    - Check if the timer has been added to the last used list
2. Repeat step [1] 7 times with different timers
3. Start any of the timers in "Last Used" and check if it is being moved to the top of the list.

TEST CASE ID

1. Check "last used" functionality - *checkLastUsed#1*

PASS/FAIL CRITERIA Pass

- Every time a timer used, it is saved on the top of the "Last Used" list.

Fail

- If, in any case, a timer is not saved in the "Last Used" list after it has been used.

IDENTIFIER Highlight is working

FEATURE When choosing a child or a configuration this should be highlighted, and when Wombat is started from the GIRAF launcher a child is chosen, this should be highlighted.

APPROACH

1. Test if children and configurations is being highlighted when they are chosen.
2. Test if the child chosen in the GIRAF launcher is being highlighted in Wombat.
3. Test if the child chosen is still highlighted after saving.

NOTE! An element in the child list is chosen if there is configurations in the configurations list, and an element in the configurations list is chosen if there is loaded a configuration in the edit screen.

TEST CASE ID

1. Check if list elements is highlighted when clicked - *hOnClick#1*
2. Check if child is still highlighted after save - *stillHAfterSave#1*
3. Check if the right child is highlighted after launch through the GIRAF launcher - *hChildOnLaunch#1*

PASS/FAIL CRITERIA Pass

- If list elements are always highlighted when selected.

Fail

- If a child is selected and it is not highlighted.
- If a configuration is selected and it is not highlighted.
- If nothing is highlighted when starting Wombat from the GIRAF Launcher

IDENTIFIER Deviation in time on done activity

FEATURE When the timer has run out, the "Done" screen will appear.

APPROACH

1. Run a timer with any timespan, and wait for the "Done" screen to appear
   - Use an independent stopwatch to verify the time it takes to show the "Done" screen.
   - Verify that there is no more than a 2 second deviation in time, from the time has run out until the "Done" screen appears.
2. Repeat step [1] 3 times with different timespans.
3. Do step [2] again with any timer with a "Digital Watch" attached.
4. Start a timer with any timespan, click the "back" button, and verify that the "Done" screen do not show up randomly.

TEST CASE ID

1. Check if timer matches real-life time - *checkTimerTime#1*
2. Check if the done screen appears when it should - *checkDone-Func#1*

PASS/FAIL CRITERIA Pass

- If the "Done" screen appears no more than two seconds after the time has run out.
- If any timer is not deviating more than two seconds from real-world time.

Fail

- The "Done" screen appears even though the timer is not running anymore or the timer is not finished.

IDENTIFIER *saveAs#1*

TEST ITEM Functionality to save customized timers in specific lists.

INPUT SPEC.

1. Click "New Template", choose a timer type, click "Save As", and choose name and location. Check if the profile was saved in the chosen location and with the chosen name.
2. Choose any configuration, edit the settings, click "Save As", and choose name and location. Check if the profile was saved in the chosen location and with the chosen name.
3. Create a new configuration with random settings and use "Save As" to save it. Go to the saved configuration, and check if the settings has changed since the save.
4. Choose an existing configuration or create a new one, and do step 1. with "Predefined" and "Last Used" as save locations.
5. Do step 1-3 again, but clear the tablet memory before the correctness is checked.

OUTPUT SPEC.

1. Configurations is saved in the chosen locations, unless the chosen locations is "Predefined" or "Last Used".
2. Configurations is saved with the chosen name.
3. Configurations is saved with the chosen settings.

ENVIRONMENTAL NEEDS

- Tablet running Android 3.2.
- Timer application installed.
- OasisLocalDatabase installed.
- One staff member to perform the test.

IDENTIFIER *save#1*

TEST ITEM Functionality to save customized or predefined timers in the highlighted child, without having to choose name and location.

INPUT SPEC.

1. Check if it is possible to save configurations in "Predefined" or "Last Used" by choosing one of the configurations in each list, edit some settings, and press "Save".
2. Select a child, edit the settings, and press "Save". Check if the chosen settings were saved.
3. Select a child, select a configuration among the child's configurations, edit the settings, and press "Save". Check if the chosen settings were saved in the same configuration.
4. Select a child, edit the settings, and press "Save" two times and see if two identical configurations are saved on the given child.
5. Highlight another configuration than the one you have just saved, then highlight the one you just saved and press "Save". Check if there is now saved a duplicate of the first saved configuration.
6. Do step 2-3 again and check if any other configuration was changed during the saving process.

OUTPUT SPEC.

1. Configurations is saved in the highlighted child.
2. When "Predefined" or "Last Used" is highlighted, nothing is saved when the "Save" is pressed.
3. New configurations are saved with the chosen settings.
4. When selecting and saving existing configurations, they are updated with the edited settings.

ENVIRONMENTAL NEEDS

- Tablet running android 3.2.
- Timer application installed.
- OasisLocalDatabase installed.
- One staff member to perform the test.

IDENTIFIER *checkLastUsed#1*

TEST ITEM Functionality to save timers into the "Last Used" list every any timer has been run.

INPUT SPEC.

1. Run 3 different timers, and see if they were saved on top if the "Last Used" list.

2. Repeat step 1, but clear the tablet memory before "Last Used" is inspected.

OUTPUT SPEC.

1. Whenever a timer has been used, it is saved on top of the "Last Used" list.

ENVIRONMENTAL NEEDS

- Tablet running android 3.2.

- Timer application installed.

- OasisLocalDatabase installed.

- One staff member to perform the test.

IDENTIFIER *hOnClick#1*

TEST ITEM Functionality to highlight list items when they are clicked.

INPUT SPEC.

1. Select three different list items in both the child list and the configuration list and see if they stay highlighted.

OUTPUT SPEC.

1. When a list item is selected, it is highlighted, and it stays highlighted until another list item is selected.

ENVIRONMENTAL NEEDS

- Tablet running android 3.2.
- Timer application installed.
- One staff member to perform the test.

SPECIAL PROCEDURAL REQUIREMENTS

- The configurations on every child will always be visible when a child list has been selected. Therefore, make sure that the highlighted child has at least one configuration before testing.
- There is no element in the configuration list if no element in the child list has been selected.

IDENTIFIER *stillHAfterSave#1*

TEST ITEM Functionality to highlight list items after a save procedure.

INPUT SPEC.

1. Select a child and a configuration, edit the settings for the configuration, and click "Save". See if the selected list items stay highlighted after it has been updated.

OUTPUT SPEC.

1. When a child and configuration is selected, and the settings for that configuration is changed and saved, the child list item and configuration list item is still highlighted.

ENVIRONMENTAL NEEDS

- Tablet running android 3.2.
- Timer application installed.
- One staff member to perform the test.

SPECIAL PROCEDURAL REQUIREMENTS

- The configurations on every child will always be visible when a child list has been selected. Therefore, make sure that the highlighted child has at least one configuration in the list before testing.
- There is no element in the configuration list if no element in the child list has been selected.
- When a configuration is selected, the settings for that configuration is always shown set in the "Customize" menu.

INTERCASE DEPENDENCIES *save#1*

IDENTIFIER *hChildOnLaunch#1*

TEST ITEM Functionality to highlight list items according to the chosen child when the application is launched through the GIRAF launcher.

INPUT SPEC.

1. Start the GIRAF launcher and open the timer application.

2. Select a child and note the name of the child.

OUTPUT SPEC.

1. The child selected in the GIRAF launcher is highlighted and the configurations belonging to this child is loaded.

ENVIRONMENTAL NEEDS

- Tablet running android 3.2.

- Timer application installed.

- GIRAF launcher installed.

- One staff member to perform the test.

SPECIAL PROCEDURAL REQUIREMENTS

- The configurations on every child will always be visible when a child list has been selected. Therefore, make sure that the highlighted child has at least one configuration before testing.

- There is no element in the configuration list if no element in the child list has been selected.

IDENTIFIER *checkTimerTime#1*

TEST ITEM Functionality which draws and updates the timer according to the time left and ensures the timer ends when the time is up.

INPUT SPEC.

1. Run four different timer styles with a static timespan (fx 20 minutes).

2. Each time a timer is started, start a precise independent stopwatch.

3. When the timer reaches zero stop the independent stopwatch.

OUTPUT SPEC.

1. The stopwatch must deviate no more than two seconds from the time selected in **input spec.** step [1].

ENVIRONMENTAL NEEDS

- Tablet running android 3.2.

- Timer application installed.

- Stopwatch.

- One staff member to perform the test.

IDENTIFIER *checkDoneFunc#1*

TEST ITEM The "Done" screen appearing when the time has run out.

INPUT SPEC.

1. Start a timer at any timespan and let the time run out. See if the "Done" screen appears within two seconds after the time has run out.
2. Start a timer at any timespan and click the "back" button, and wait at least the amount of time the timer would have run, to verify that the "Done" screen do not show up anyways, if the timer has been interrupted.

OUTPUT SPEC.

1. When a timer has been run, and not interrupted, the "Done" screen appears about two seconds after the time has run out.
2. The "Done" screen is only shown if the timer is not interrupted, and the time has run out.

ENVIRONMENTAL NEEDS

- Tablet running android 3.2.
- Timer application installed.
- Stopwatch.
- One staff member to perform the test.

INTERCASE DEPENDENCIES Test case: *checkTimerTime#1*

7.3 TEST RESULTS

7.3.1 *Reflections*

We did not outsource the tests, therefore did we conduct the tests ourselves, this is not the proper way to perform black box testing. This means that the testers also developed the application, the test design, and the test cases, and thereby have knowledge of how the features are implemented. Therefore may a test person without knowledge of the code have another approach to the tests than what we have.

Test Case ID	Pass	Fail	Notes
saveAs#1	✓		
save#1	✓		
checkLastUsed#1		✓	When tablet memory is cleared, the last used timers disappears.
hOnClick#1	✓		
stillHAfterSave#1	✓		
hChildOnLaunch#1	✓		
checkTimerTime#1	✓		
checkDoneFunc#1	✓		

Table 1: Test results of the black box testing of the WOMBAT application.

We already knew that *checkLastUsed#1* would not pass before conducting the tests. We had trouble implementing the "Last Used" feature with the OasisLocalDatabase, this resulted in sporadic system failure. We decided to use the tablet memory to save the last used configurations.

7.4 USABILITY TEST

In this section we explore the results of the usability test, which was described in Section 1.8. We focus only on the results for the Timer application, since we are not going to need the results for the other parts of the GIRAF system.

7.4.1 *Results and Observations*

Through the IDA method, we found six problems with the application, and they are categorized in table 2.

The problems found, besides the color palette, are design related, and can be corrected without changing any of the implemented functionality. They could be solved by slightly changing the design of buttons and adding more confirmation messages.

7.5 ACCEPTANCE TEST

The purpose of the acceptance test[3] is to test if the system fits into the context it is designed for. The main objectives is to know if the interface feels natural for the guardians to use the timer application instead of their regular physical timers, and to know if the children understand the digital version of the different timers.

Category	Notes
Cosmetic	1: the users were not sure whether the attachments were selected, when they had chosen the attachments. 2: the user had difficulties finding the "Start"-button.
Serious	1: the user had difficulties using the imported Android color palette. 2: deletion by long-click is not intuitive. 3: the "Last Used" list was difficult to find.
Critical	1: when the background of the Timer application had a certain color, the "Gradient"-button was invisible.

Table 2: Problems found from the usability test of the WOMBAT Timer application.

To test for acceptability, the application needs to be tested in the context it is developed for, therefore did an educator, at the kinder garden Birken, borrow the tablet with the timer application for a few days, such that she could use the system in real life scenarios.

7.5.1 *Results and Observations*

The test person wrote a diary, to keep track of her experiences with the application. The test person used the timer application herself, and she let some of the other educators try it as well. The diary can be found in appendix 10.6.

The diary explains, that the children understood the meaning of both the timers and the pictograms. One of the children was more interested in watching the digital timer countdown than in the activity he was meant to do, but beyond that there were no problems with the application.

We can tell by the last used list, found in Section 10.6, that the test person used different timers with different pictograms and timespans, only one out of eight timers had a slightly different color. This could indicate that changing the colors on the timers is not very intuitive, or they do not need the functionality.

Through the development process we have used a variety of tools. When designing we have made use of stories an prototypes to communicate our ideas to our contact person and the other project groups. We made use of metaphors in our system design, to enhance the learnability of the system, by using symbols and design which may be recognized by the users.

During the development we used pair programming and refactoring to ensure higher quality of the code, and thereby enhance the readability for the sake of the students of next year, who are probably going to continue the development.

In the implementation of WOMBAT fragments and lists are essential for the main screen, lists because they hold the user data, and fragments because they ensure smooth and customizable user interface features.

The testing has also been an essential part of the development of the WOMBAT system. During the integration of all the different GIRAF system applications there has been a lot of undocumented testing, to get the applications to work together.

Formal black-box test and acceptance test has proven that WOMBAT functions correctly, and is ready to be used in the institutions, but the usability test revealed a number of design issues, which could be corrected in the future.

Part III

Evaluation

In this part we discuss and evaluate the use of development method and development tools. Furthermore we will conclude on the project, and in perspective give ideas for further development on the WOMBAT system.

8 DISCUSSION

8.1 EVALUATION OF DEVELOPMENT METHOD

In this section, we evaluate on the development method described in Section 1.4.

AGILE DEVELOPMENT In the beginning of the project we worked towards the vision described in Section 3.1. Because other parts of the multi project changed, we had to alter the focus of our project. These changes occurred when we had begun developing, and since we used an agile development method, it was possible for us to change direction instead of starting all over.

MEETINGS The meetings have been useful, since the individual projects changed when the development begun, and these changes was presented at the meetings. This helped the groups adapt their project to fit in the multi project.

SPRINT LENGTH Since meetings were only planned in the beginning and end of each sprint, and these meetings were used to present backlogs and discuss overlapping concerns, it was important that the sprint length was short (7-14 days). This was an appropriate length, because it was possible to stay updated on the progress of the multi project, and adapt the individual project to it.

PRODUCT OWNER Since we had no product owner, we did not have one person to manage the multi project, and therefore all decisions were made democratically by the multi project groups. This lead to several time consuming discussions of minor details regarding the project.

8.2 DEVELOPMENT TOOLS

As described in Chapter 5, we have used pair programming and refactoring. We evaluated on our use of these methods as part of a mini project in the course *Software Engineering*, this evaluation is included here.

8.2.1 *Pair programming*

We did some pair programming, but we were unable to do all programming in pairs. Since we are three persons in the group, we would either have to work three around one computer, or exclude one person from the pair programming. We chose to exclude one person from the pair programming, also because it would be a waste of resources to work in groups of three since the efficiency does not match the cost, when adding the third person.

One of the downsides of the pair programming technique is that we have been unable to work in pairs outside the university, i.e. when two group members are doing pair programming on a specific part of the code, and one of them got sick, the programming would be continued by only one group member, the third group member would join the programming and thereby pause the tasks he was doing, or the programming of the specific part of code would be paused until both group members would be gathered again.

The code written in pairs, is of higher quality than the code written before we implemented the pair programming technique, and the higher quality code is more readable, and thereby has no need for refactoring. We therefore believe that the use of the pair programming technique has been beneficial. This technique could be useful in our future careers.

8.2.2 *Refactoring*

There are two main reasons for the need for the code to have high readability and understandability. One reason is that the project will be handed over to the students of next year for further development, so to help them we strive to write understandable code. The other reason is that we are three persons in the group, and we have not all been included in developing every part of the program, so with understandable code it is easier for the other group members to understand the parts they did not participate in writing.

Some of the refactoring we have done is renaming variables and objects to match the content of them. Also all private variables and objects has been refactored to always start with either "m" or "_".

The use of this technique has helped ourselves to better understand the code, and we believe that when the project is passed on to the next project groups, it will be easier to familiarize themselves with it. We have learned that refactoring is very useful when several people are developing on the same product, and since we are very likely to work in groups later in our careers, we will probably use this technique again.

9 CONCLUSION

The problem definition in Section 1.5 states:

> How can we ease the daily life for children with ASD and their guardians, while complying with the study regulation?

To fulfill with the study regulations the five project groups created one unified system. The project groups have been using the same development method to ensure that all project groups have had the possibility to keep abreast of the multi project, Section 1.4.

Together did the groups develop the system GIRAF, a collection of the projects of the semester, of which this project has focused on the timer application, WOMBAT. The timer application is designed to replace the physical timers that the guardians uses, Section 2.1 and Chapter 3.

The usability test proves that the application is user friendly for the educators in the test group, Section 7.4. Furthermore does the acceptance test prove that the children, understand the meaning of the timers and attached pictograms, Section 7.5. The pictograms is a new feature that the educators did not have available previously with the physical timers, **??** Der hvor mette foreslår det!.

The timer application is configurable to match any timer the guardian use, Section 6.4.1, further can the guardians attach pictograms or timers to the timers in progress as an which is an improvement compared to the physical timers, Figure 6.4.1.

The timer application is easy to use, as proven by the tests, as well as improved in functionality compared to the physical timers, we therefore conclude that the timer application can ease the daily life for children with ASD and their guardians.

10

PERSPECTIVE

10.1 FUTURE WORK

In this section we will list some ideas for further development on the WOMBAT application.

10.1.1 *Ideas based on the usability and acceptance test*

In this subsection we will briefly describe changes we would like to implement based on the usability test and the acceptance test.

Color picking

WOMBAT is currently using a color palette that the guardians uses to pick colors, we would like to change this into a list of predefined colors and an advanced button where they are able to define new colors with a color palette. This change would ease the color picking.

Gradient button

The gradient button can be difficult to spot, it is therefore necessary to improve the layout. It might help if the button layout looks the same as all the other buttons in WOMBAT.

Improve customize fragment buttons

The buttons in WOMBAT need to be redesigned to appear more like buttons. The attachment buttons visual effect after attaching an attachment needs to be improved such that it is more clear for the user, when an attachment has been attached.

Delete

It is not intuitive how to delete a configuration. It might help to remove the long-click and make a delete button on each configuration. Figure: 42 illustrates how such a button might look like.

Figure 42: Illustration of possible delete button

Last used and Predefined layout changes

The usability test showed that the last used and predefined category were hard to find, these two categories need a design change to have a different layout than the children in the list. This could be achieved by changing the background color of these two categories.

Done screen lock

The acceptance test showed that it is necessary to improve the lock on the done screen picture. The done screen unlocks by a single touch on the screen, which is issue, would be changed to e.g. a long-click instead.

10.1.2 *Ideas*

In this subsection we will briefly describe ideas which can be used for future development of WOMBAT.

Handset Version

It is, at the moment, not possible to install and run the WOMBAT application on an Android handset, but it might be more convenient to bring a handset than a tablet, when the guardians and children is on a trip.

Improved Graphics

WOMBAT is currently using canvas for drawing all the timers, which is simple 2D graphic. It is possible to improve the graphic by using OpenGL instead. OpenGL is used for drawing detailed 3D objects. The benefit of improving the graphic in WOMBAT is to make the timers more realistic and look like the timers which the children already uses. There have been conducted a comparison between canvas and OpenGL which can been read about in subsection: 6.5.2.

Overlay timers

This idea is based on the second part of our original idea, that can be found in Section 3.1. When you exit the timer screen with a timer in progress, the timer will be destroyed. Instead of destroying the timer, it should enable an overlay timer which should continue from where the timer left off. The overlay timer should always appear on top of all the other application. This would make it possible for the guardian to conduct multiple tasks without stopping a timer. When the guardian enters WOMBAT again, it should remove the overlay and continue the timer normally.

Code Architecture

The draw feature which is a part of the `DrawLib`, were original designed to be a part of the `TimerLib`, the decision of this can be read in Section 6.1. The ideal architecture would be to import the draw methods into each timer class, that way you could call the method on a certain timer object and return a `View`. By doing so, you would not have to check on which object you are handling, see Figure 43 and Figure 44.

```
...
        switch(sub.getAttachment().getForm()){
        case Timer:
          ...
          break;
        case SingleImg:
          ...
          break;
        case SplitImg:
          frameWidth = frameWidth/2;
          v = genDrawView(sub, frameWidth);
          frame.addView(v, frameWidth, frameHeight);
          frameWidth = frameWidth/2;
          frameWidth = frameWidth - 15;
          i = new ImageView(this);
          i.setImageResource(sub.getAttachment().getLeftImg().
              getPath());
          i.setBackgroundDrawable(gd);
          frame.addView(i, frameWidth, frameHeight);
          i2 = new ImageView(this);
          i2.setImageResource(sub.getAttachment().getRightImg().
              getPath());
          i2.setBackgroundDrawable(gd);
          frame.addView(i2, frameWidth, frameHeight);
          i3 = new ImageView(this);
          i3.setBackgroundDrawable(gd);
          frame.addView(i3, 30, frameHeight);
          break;
        }
        setContentView(frame);
        ...
```

Figure 43: Code snippet of how `DrawLib` checks for object form to generate the corresponding `View`.

```
LinearLayout frame = new LinearLayout(this);
Guardian guard = Guardian.getInstance();
frame.addView(guard.getSubProfile().draw());
setContentView(frame);
```

Figure 44: Example of how WOMBAT would generate timer `View`, if `DrawLib` was apart of `TimerLib`.

Last used

WOMBAT do not save the configurations in the last used list, this is because it is not implemented to work together with `OasisLocalDatabase`. When the WOMBAT activity is destroyed or the memory of the device is cleared, the last used list will be empty. In the future

this function should be reimplemented to work together with the `OasisLocalDatabase`.

Delete and save

There is a bug which occurs when you create three or more new timers on the same profile. When you try to delete these timers it will delete them randomly and might even force WOMBAT to crash due to it is trying to delete a non existing setting in the `OasisLocalDatabase`. This is a bug in the application that needs to be future developed on.

Part IV

Appendix

APPENDIX

10.2 NOTES FROM INTERVIEW

This is notes from an interview with Mette Als Andreasen, an educator at Birken in Langholt, Denmark.

Naar tiden loeber ud (kristian har tage et billede):
Faerdig - symbol
Gaa til skema - symbol
Taget fra boardmaker

Kunne vaere godt hvis man kunne saette egne billeder ind som start/stop symboler.

Roed farve = nej, stop, aflyst.

De har saadan et ur paa 60 minutter hvor tid tilbage er markeret med roed, og saa bipper den lige kort naar den er faerdig.
Det ville vaere fint hvis de kunne bruge sort/hvid til dem der ikke kan haandtere farver, men ogsaa kan vaelge farver.

Stop-ur:
en fast timer paa 60 minutter + en customizable som ikke ser helt magen til ud, som f.eks, kan vaere paa 5, 10 eller 15 minutter for en hel cirkel.

timeglas:
skift farve paa timeglassene, men ikke noedvendigvis goere dem stoerre. Kombinere med mere/mindre sand. Eventuelt kombinere med et lille digitalt ur, til dem der har brug for det, skal kunne slaaes til og fra.

Dags-plan:
ikke saerlig relevant til de helt smaa og ikke saerligt velfungerende boern. Men kunne vaere rigtig godt til de lidt aeldre.
En plan gaar oppefra og ned, og hvis der saa skal specificeres noget ud til aktiviteterne, saa er det fra venstre mod hoejre ud fra det nedadgaaende skema.

Til parrot:
Godt med rigtige billeder af tingene, som paedagogerne selv kan tage, eventuelt ogsaa af aktiviteter, saa pedagogerne kan have billeder af aktiviter som de kan liste efter skeamet.

Invitation til deltagelse i test af GIRAF

- En Android applikation bygget til børn med autisme

Kære ekspert

Vi vil gerne inviterer dig til at deltage i den første brugervenligheds test af GIRAF, en Android applikation bygget til børn med autisme. Formålet med denne test vil være at undersøge hvor brugervenlig applikationen er og hvor nemt eller svært det er at bruge den. Derfor er det helt fint hvis du aldrig har set eller hørt om denne applikation før nu, da vi gerne vil observerer, hvordan første gangs brugere så vel som brugere med kendskab til applikationen, har det med denne applikationen.

Bemærk venligst at vi er ikke tester din kendskab til applikationen eller evner med en tablet, men derimod om GIRAF applikationen er nem at bruge, vi har kun interesse i at kende til de svagheder der ville være i applikationen. Dette betyder også at du ikke kan give nogle forkerte svar, da du er eksperten.

Derfor vil vi gerne inviterer dig ud i vores brugervenligheds laboratorie, hvor vi kan studere din brug af applikationen. Under brugervenligheds testen vil du blive givet en række opgaver, som skal udføres. Yderligere vil du blive bedt om at tænke højt og fortælle alle tanker, indtryk og valg du tager ved brug af applikationen under testen. Under testen af applikationen vil der blive optaget både video og lyd, til at studere testen senere.

Dagen kommer til at bestå af:

Vi vil meget gerne høre fra dig hvis du har lyst og tid til at deltage i denne brugervenligheds test, den 22/5 - 2012, på Aalborg Universitet.
For at vide hvornår på dagen du kan komme vil vi gerne, at du går ind på denne side (http://www.doodle.com/d2h6swgbtsdf6z2b) skriver dit navn og vælger det tidspunkt på dagen du helst vil komme, dette er svar nok for at vi ved du gerne vil komme.

Kommentarer og spørgsmål kan sendes retur til den mail invitationen kom fra.

På forhånd tak,
Android projektet
Software 6. semester
Aalborg Universitet
Selma Lagerlöfs vej 300, 9220 Aalborg

AALBORG UNIVERSITY

Figure 45: Invitation sent to the test persons of the usability test.

Der var mange skemaer rundt omkring, og der henviser det sidste billede i raekken til naeste skema, som haenger f.eks. paa badevaerelset eller i garderoben.

10.3 PAPER PROTOTYPES

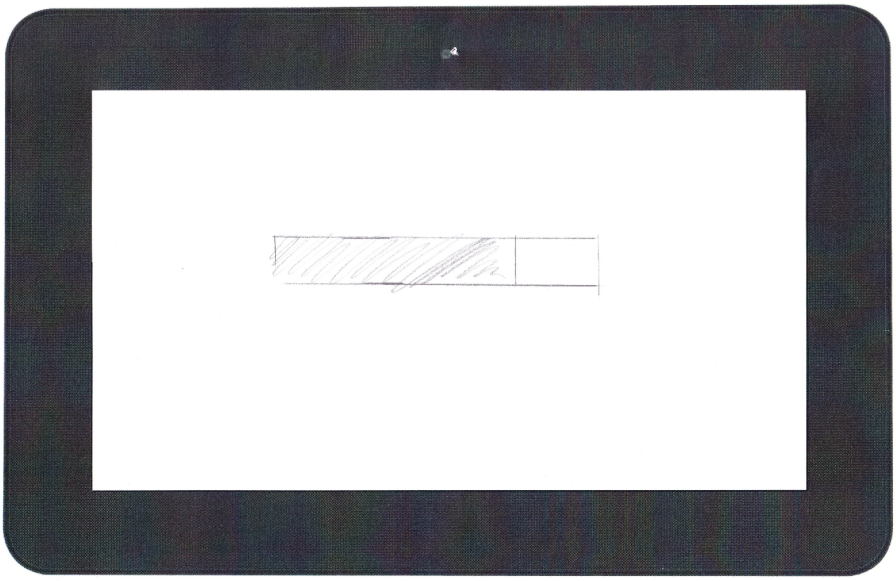


Figure 46: Scan of a paper prototype of a single progress bar timer.

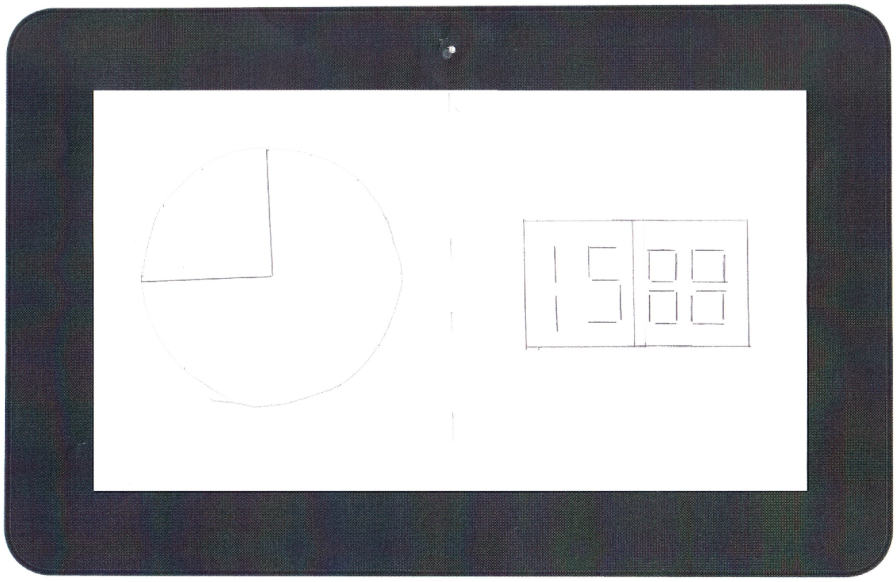


Figure 47: Scan of a paper prototype of a double timer.

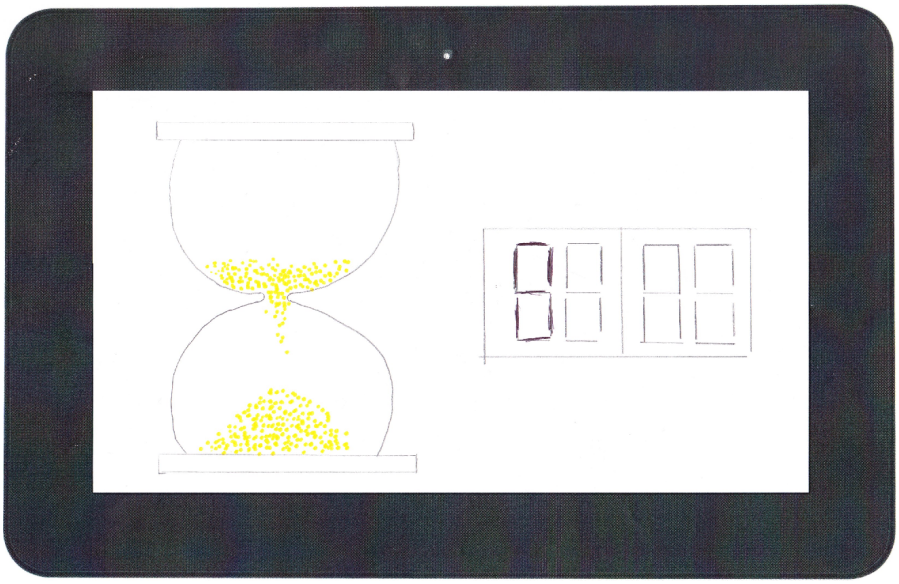


Figure 48: Scan of a paper prototype of another double timer.

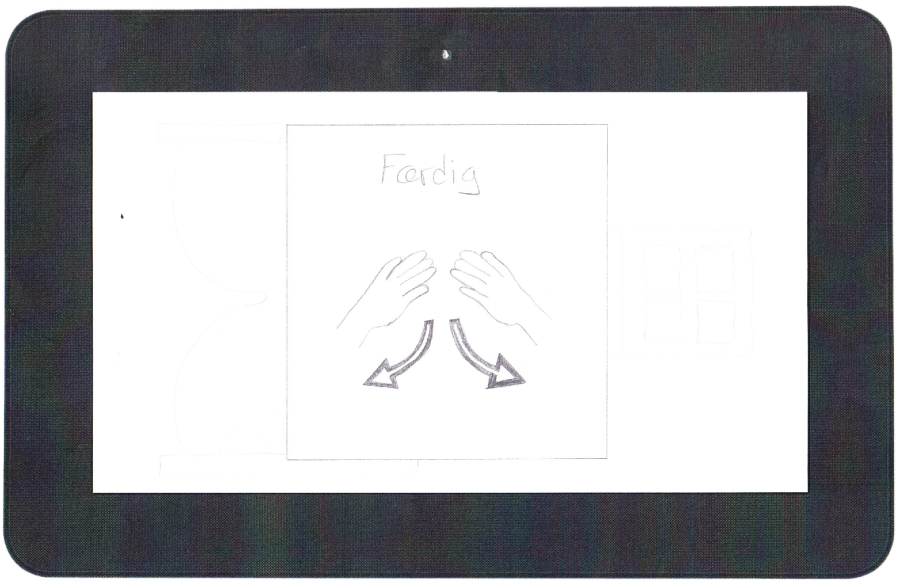
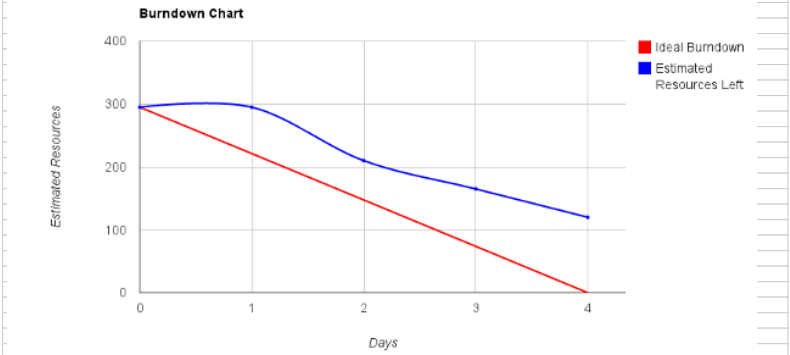


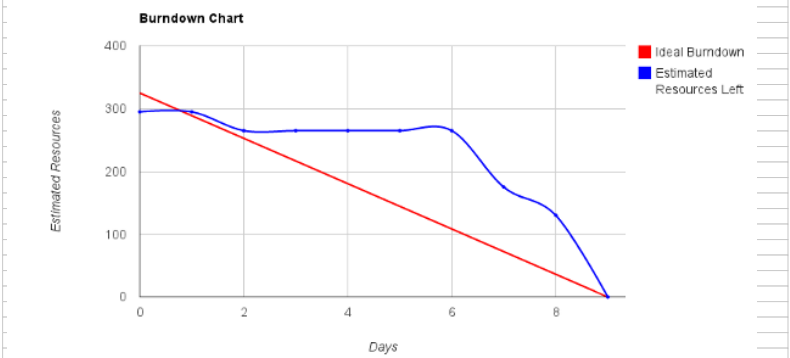
Figure 49: Scan of a paper prototype of the "Done" screen shown when the time has run out.

Timer (WOMBAT)



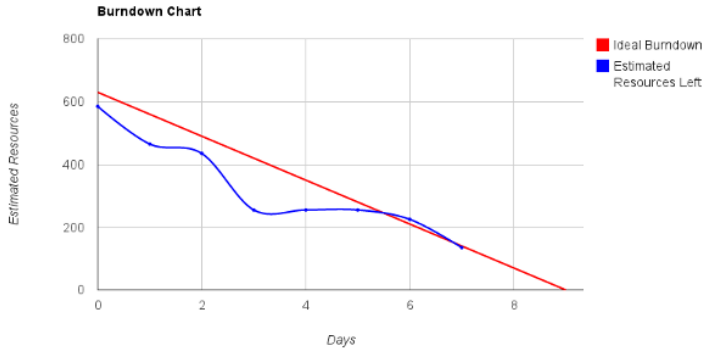
ID	Name	Priorite	Estima	How to demo	Note	Status
1	Design Start activity	14	25	Two lists, 1. = lists of list names, 2. list of profiles		DONE
10	Develop Start activity	13	60	According to the design [1], program an "input" for the two lists and make them clickable		DONE
5	Develop Load Settings	12	60	Load relevant attributes from admin	*Function at Admin*	*In progress*
14	Retrieve list of list profiles	11	45	Retrieve list of list of profiles from admin group	*Get from admin*	*Admin function not working*
11	Show lists	10	30	Get lists of list profiles from [14], function which returns to start activity [10]	*Missing show pictures*	*In progress*
12	Show profiles	9	30	Get list of profiles from [11], according to click from start activity, returns to start activity [10]	*Missing show pictures*	*In progress*
8	Select user profile	8	30	Open "Tilpas" activity when a profile is selected in [12]		DONE
2	Design Tilpas activity	7	15	Window with all possible settings, "Gem" prompts a profile chooser		DONE

Timer (WOMBAT)



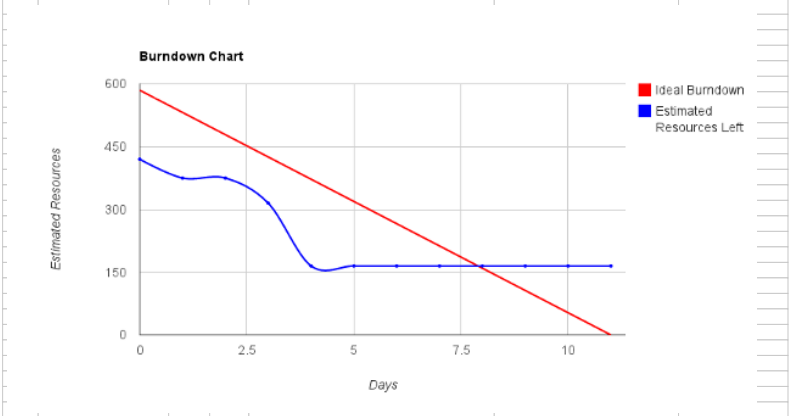
ID	Name	Priorite	Estima	How to demo	Note	Status
3	Design Vis activity	12	15	Digital Watch + the Timer according to the settings		DONE
4	Design "Færdig" activity	13	15	According to the "done" picture from the institution		DONE
7	Design Predefined Profiles	14	30	Design the predefined profiles	*Pictures Aquired*	DONE
23	Implement load/save	15	45	Implement load/save in the profile and template lists		DONE
15	Develop "tilpas" activity	16	90	According to [2], implement save settings		DONE
17	Develop "vis" activity	17	30	According to [3], link til OpenGL		DONE
				General openGL settings e.g. camera		

Timer (WOMBAT)



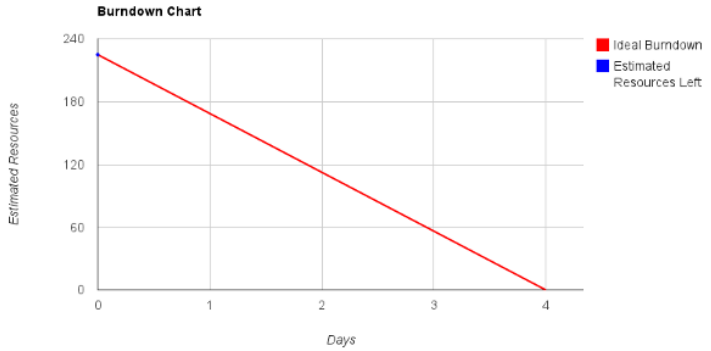
ID	Name	Priorite	Estima	How to demo	*Note*	*Status*
5	Develop Load Settings	19	60	Load relevant attributes from admin	*Function at Admin*	*In progress*
6	Develop Save Settings	21	30	Save relevant attributes to admin	*Function at Admin*	*In progress*
9	Last Used Algorithm	22	30	Design how the last used algorithm works		*DONE*
14	Retrieve list of list profiles	20	45	Retrieve list of list of profiles from admin group	*Function at admin*	*DONE*
22	Pictures in Lists	41	45	Implement pictures in the profile and template lists	*Missing how to pic from admin*	
26	Impelment Last Used	24	45	Implement last used algorithm in the profile list [11]		*DONE*
27	Attachment Functionality	40	90	Complete the attachment functionality according to the design	*SubProfile, digiPecs, missing pecs from admin*	*DONE*
28	Save/Save As error handling	26	60	Dont save if something is missing	*Hide "gem" and "gem som" and set lock on "ny subprofile" and "last used" and "predefined"*	*DONE*
30	Implement Predef profiles	29	30			*DONE*
31	1 min bug	28	15	Bug when opening and saving timer with set to 1 min		*DONE*
32	Highlight after save	27	60	Highlight the chosen subprofile after saving		*DONE*
33	Select subprofile functionality	25	30			*DONE*
34	Design icons for Last Used / Predef	30	30		*Not inplemented yet*	*DONE*
35	Long press delete	31	30		*Missing delete function*	*DONE*
36	"New" button	32	30			*DONE*

Timer (WOMBAT)



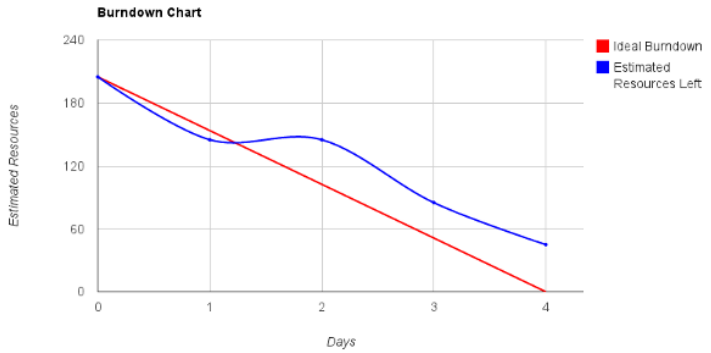
ID	Name	Priorite	Estima	How to demo	*Note*	*Status*
21	Digital Watch	44	30	According to [3]		*DONE*
38	Tekst på delete subprofile	46	15			*DONE*
39	Undim Screen	47	45			*DONE*
40	Gemte profil skal highlightes	48	30			*DONE*
18	Hour Glass		45	According to [3]		*DONE*
41	Vedhæftet skal ændre/update tid ved tryk på start/gem	49	45		*Bliver gemt når man opdaterer tiden på timeren*	*DONE*
5	Develop Load Settings	19	60	Load relevant attributes from admin	*Function at Admin*	*DONE*
6	Develop Save Settings	21	30	Save relevant attributes to admin	*Function at Admin*	*DONE*
16	Develop "færdig" activity	51	60	According to the "færdig" sign used in the institution		*DONE*
43	Develop "Vælg færdig billede"	52	60	Show a custom picture when the time has run out. Default is "Færdig billede"	*Mangler at kunne sætte færdig billede på subprofil*	
22	Pictures in Lists	43	45	Implement pictures in the profile and template lists	*Missing how to pic from admin*	
37	Vedhæftet ikon skal ændrer baggrund	45	60			*DONE*
42	Lås/Authenticate	50	60			

Timer (WOMBAT)



ID	Name	Priorite	Estima	How to demo	Note	Status
22	Pictures in Lists	43	45	Implement pictures in the profile and template lists	*Missing how to pic from admin*	*In progress*
43	Develop "vælg færdig-billede"	52	60	Show a custom picture when the time has run out. Default is "færdig-billedet" from Mette		*In progress*
44	Develop vedhæft pictogrammer	53	60		*Missing a part from admin*	*In progress*
45	Re-factoring	54	60	CustomizeFragment etc.		*DONE*

Timer (WOMBAT)



ID	Name	Priorite	Estima	How to demo	Note	Status
22	Pictures in Lists	43	45	Implement pictures in the profile and template lists	*Missing how to pic from admin*	*In progress*
42	Lås/Authenticate	50	60			*DONE*
43	Develop "vælg færdig-billede"	52	60	Show a custom picture when the time has run out. Default is "færdig-billedet" from Mette		*DONE*
44	Develop vedhæft pictogrammer	53	20		*Missing a part from admin*	*DONE*
45	Update færdig aktivitet	54	20	Update according to [43]		*DONE*
45	Update færdig aktivitet	54	20	Update according to [43]		*DONE*

Figure 50: Picture of our burndownchart and list of assignments. Used to manage the total project backlog.

Analysis of Project - SWOT

Strengths

1. Specific target group
2. Versatile Platform
3. The majority of the multi project groups are using scrum
4. Version control (revision control)
5. Previous work experience within the group
6. Good communication with the customer
7. Agile development method

Weaknesses

1. Limited time
2. Inconsistent use of development method among groups
3. Lack of experience with development method and target platform
4. No project owner (Scrum)
5. Clutter source code
6. Undocumented source code

Opportunities

1. The target platform is growing in popularity
2. The platform is getting more widespread throughout the world, and might therefore get cheaper
3. Our target group is present in the entire world, we could therefore expand international

Threats

1. Project management
2. Miscommunication between the multi project groups
3. Testing
4. Multi project integration
5. Our customer change opinion and leave the project

Technique Matching

To eliminate the weaknesses we can use the following tools:

1. Planning to better use the time available.
2. Meeting with other groups to agree on development method.
3. Pair programming to achieve better understanding of the target platform.
4. Refactoring to prevent code clutter and make code more readable.
5. White box testing can point out code which is not documented well enough.

Technique Research

Pair Programming

"All production code is created by two programmers at one computer; they rotate using the input device periodically. Paris may change frequently, for different tasks. The observer is doing a Real-time code review, and perhaps thinking more broadly than the coding developer typically does, considering tests and so forth.

Certainly, team productivity is not simply a function of the number of hands typing - it is more nuanced. The XP claim is that the combination of cross learning, the peer pressure of more disciplined practice observance and more hours actually programming than procrastinating, defect reduction due to real-time code review, and the stamina and insight to carry on when one programmer is stuck, all add up to an overall team improvement."

source: Craig Larman, Agile & Iterative Development - A Manager's Guide, p. 149

Refactoring

"Refactoring is the continual effort to simplify the fine-grained code and larger design elements, while still ensuring all tests pass. That is, cleaning the code and design, without changing functionality.

The goal is minimal, simple, comprehensive code. It is achieved by small change steps, verifying tests after each, and ideally the use of refactoring tools, now available in some IDEs."

source: Craig Larman, Agile and Iterative Development - A Manager's Guide. p. 149

Dato: 16/3 - 12

Mads

Virkede godt	Virkede knapt så godt/slet ikke
Var klart genkendelig for barnet (Saudur)	Slutbilledet var for hurtigt væk. Måske kunne det blive indtil jeg bestemmer andet.

Dato: 18/5 - 12

Virkede godt	Virkede knapt så godt/slet ikke
Drengene forstod straks mad mulighen var. De gik til skema da tiden var udløbet. Saudur Lukas. Emil	"SAMSUNG" (Visuel stej)

Figure 51: Scan of a paper prototype of the "Done" screen shown when the time has run out.

To fetch the released code from SVN, make a checkout to `https://sw6-2012.googlecode.com/svn/tags/wombat/`, previous versions of the project, reports, and the wiki can be found by making a checkout to `https://sw6-2012.googlecode.com/svn/`. The folders on SVN contains the following:

- *trunk* - contains iteration releases of the code.
- *branches* - contains code under development.
- *tags* - contains the newest final release of the code.
- *common_report* - contains the common part of the report.
- *Reports* - contains all group reports.
- *wiki* - contains summaries from all group- and supervisor meetings, and guides/agreements.

When SVN checkout has finished downloading content from the SVN, import Wombat, AmbilWarna (the color picker), and wheel projects from svn to Eclipse, figure 52.

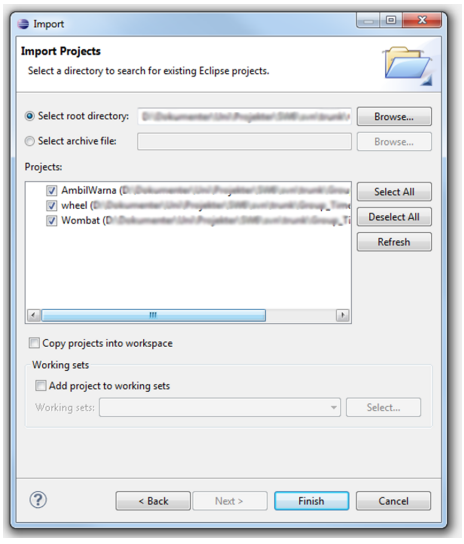


Figure 52: Import Wombat, AmbilWarna, and wheel.

Right click the Wombat project in Eclipse and click "Properties". Under the "Android" pane, ensure that AmbilWarna and wheel are added as project libraries, figure 53.

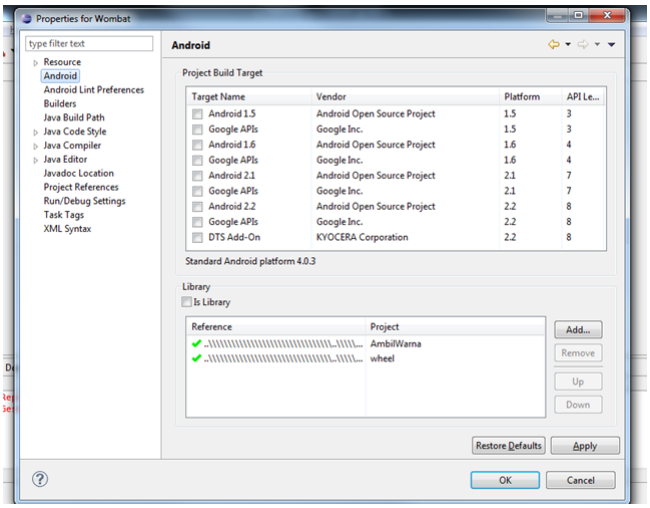


Figure 53: Add AmbilWarna and wheel as project libraries.

In case Eclipse fails to recognize the libraries in the libs folder, manually add the three JAR files in Wombat/libs on svn as external JARs, in Wombat - Properties, figure 54.

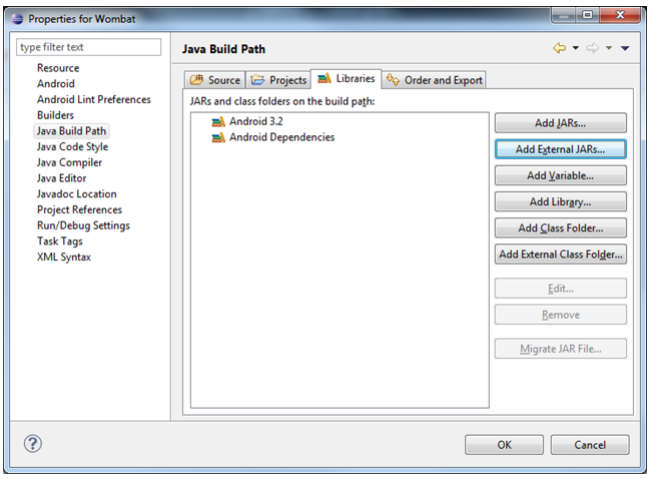


Figure 54: Add Oasis, TimerLib, and DrawLib as external JARs if Eclipse cant recognize them in the libs folder.

Extras

To edit the TimerLib and DrawLib the two projects has to be imported to the workspace in Eclipse, figure 55.

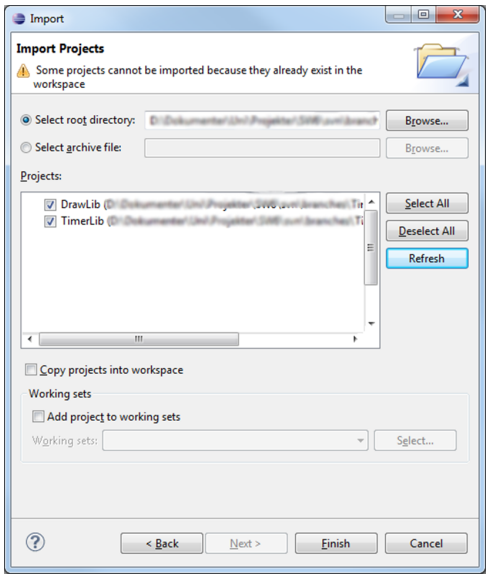


Figure 55: Import TimerLib and DrawLib to the workspace.

To debug the TimerLib and DrawLib in WOMBAT the projects has to be added as libraries in the Wombat properties, with AmbilWarna and wheel, figure 56. Then the TimerLib and DrawLib JARs has to be deleted from the libs folder or the Java Build Path library, to avoid a compile error.

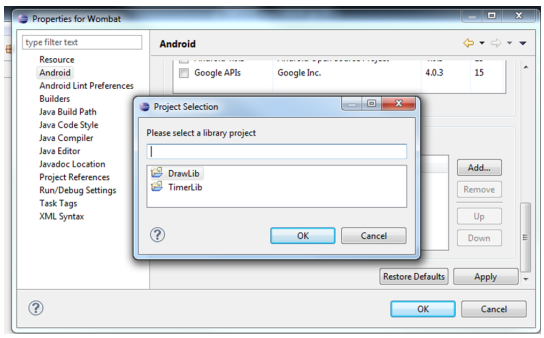


Figure 56: Add TimerLib and DrawLib as project libraries.

BIBLIOGRAPHY

[1] Scrum Alliance. Advice on conducting the scrum of scrums meeting, 2011.

[2] Scrum Alliance. Scrum alliance, 2011.

[3] David Benyon. Designing ineractive systems. Book, ? 2010. Pages: 88, 187, 203-207.

[4] Tim Bray. Fragments for all. http://android-developers.blogspot.com/2011/03/fragments-for-all.html, 2011. Last visit: 15-05-2012.

[5] Google. Fragments | android developers. http://developer.android.com/guide/topics/fundamentals/fragments.html, 2012. Last visit: 15-05-2012.

[6] Temple Grandin. Teaching tips for children and adults with autism, December 2002.

[7] Steffan Bo Pallesen Jacob Bang, Lasse Linnerup Christiansen. Administration module for giraf, May 2011.

[8] Jan Stage Jesper Kjeldskov, Mikael B. Skov. *Instant Data Analysis: Conducting Usability Evaluations in a Day*. Last viewed: 2012-05-24.

[9] DynaVox Mayer-Johnson. Boardmaker software. http://www.mayer-johnson.com/boardmaker-software/, 2011. Last visit: 03-05-2012.

[10] Microsoft. Model-view-controller, May 2012.

[11] Samsung. Samsung tablet.

[12] Yuku Sugianto. Android color picker. http://code.google.com/p/android-color-picker/, 2011. Last visit: 23-05-2012.

[13] Android Development Team. Activity | android developer, May 2012.

[14] Android Development Team. Android 4.0.3 platform, 2012.

[15] Aalborg University. Studieordning for bacheloruddannelsen i software. *URL: http://www.sict.aau.dk/digitalAssets/3/3331_softwbach_sept2009.pdf*. Last viewed: 2012-03-15.

[16] Don Wells. extreme programming, 2009.

[17] yuri.kanivets. The wheel widget for android. http://code.google.com/p/android-wheel/, 2011. Last visit: 23-05-2012.