Android Timer Application

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: X

Synopsis:

This project is about the development of...

Preface

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

Contents

Special Words

- Time Timer - A visual clock. Used by parents and teachers as a tool to visualize time for children.
- Eclipse - An intelligent development environment for Java.
- ASD - Auto Spectrum Disorder.
- Children - when we write "children", we refers to "children with ASD".
- MVC - **M**odel **V**iew **C**ontroller [**?**].
- WOMBAT - name of the timer application, **W**ay **O**f **M**easuring **B**asic **T**ime.
- PARROT - name of the pictogram application, **P**ictogram **A**ssisting with **R**hetoric **R**easoning **O**r **T**alking

Part I

Introduction

In this part there will be an introduction to the project, including background knowledge about the target platform, and knowledge about autism. There will be an analysis of the problem followed by a system definition of the whole multi project, and the specific group project.

CHAPTER 1

Analysis

Through meetings with Mette Als Andreasen, an educator at Birken, a special kindergarten for children with autism, we have learned a lot about children with autism, and the importance of having access to well-designed communication tools.

At Birken they often use hourglasses, and other kinds of timers, in different sizes and colors to visualize the progression of time to the children. The children will then associate the color and size of an hourglass with the time it represents, and specific timers are always used when they are performing specific activities, i.e. they always spend 30 minutes on eating lunch.
Mette Als Andreasen also explained how they use pictograms to communicate with the children. They have a scheme for the day, where all their daily activities are listed in pictograms, so the children can always go to their schemes and see what they are going to do next. Also activity instructions are listed with pictograms, i.e. in the bathroom there is a scheme showing how to wash hands.
The pictograms used at Birken comes from a licensed piece of software called *Boardmaker*[**?**]. To use the pictograms, the educators have to choose and edite them on the computer, print them out, cut them out in small squares, and laminate them. After that they can be put to use either on the schemes or by showing them when needed.

In general the guardians need a lot of different tools all the time, and the tools are not very practical to transport. Therefore it would be practical to

have a digital version of the timers and pictograms, so they would only have to bring one tablet, where all the needed tools are available. This leads to the system definition of this subproject, which is found in the next section.

1.1 System Definition

The application we are developing is targeted for android tablets running Android 3.2. The use space is institutions and homes of autistic children.
The application is meant as 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 hour glass or a stop watch.
Furthermore the guardians should be able to add pictograms to the timer view, 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 run out.

Tail

Part II

Development

In this part we start with a section about project management, where we expand on the used development method and versioning. After that we describe the design, development process, implementation, and test of the WOMBAT application.

CHAPTER 2

Project Management - on a Multi-Project Level

This project is a multi-group project, and five groups of three to four people are working together to develop one combined product. For this to work out, it is necessary to have a development method which suits this kind of multi-group development. Through the course *Software Engineering*, a part of our study regulation, we learned about SCRUM, and we chose to adopt this technique.
Section 2.1 is based on a mini-project written in the course *Software Engineering*, and it explains the modifications we have applied to the SCRUM method.

2.1 Modification of SCRUM

Even if we decided that SCRUM would best suit our situation, we had to modify the method to make it fit better. The modifications are as follows:

- Daily SCRUM is substituted with weekly SCRUM.
- We have no SCRUM master. The group is together responsible for the SCRUM master's tasks.
- Extensive documentation is done, since the bachelor project next year is going to be based on the results of this project.

- Instead of a project owner we have a democracy between all groups. This is because everyone has to learn from this project and therefore everyone has to be engaged all phases of the development.
- We are limited in time, so we are probably only going to do three 2-week sprints in total.

Arguments for Chosen Method Since this project requires the entire android group to be split into smaller groups of 3-4 people, we are going to use a development method which supports small groups working together. It is important that we develop prototypes, showing the project concepts in progress, to make sure we reach a usable final product. Since everyone involved in the project is novices in the field, students being new to working with autistic children and educators being new to working with software developing, an agile method would be suitable, as changes in demands to the product might occur often.

Agile development meets that the software is developed as part of a learning process.

2.2 Versioning

For versioning the code and documentation, we use a SVN-server at *Google Code*[**?**].

There several folders on the server, which help the groups to keep track of working code, reports, etc.. The structure is as follows:

- *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.

CHAPTER 3

Design

The design has changed throughout the project. In this chapter we present the vision with the timer application, and some of the tools we have used when designing and re-designing.

3.1 Vision

Since the launcher had both a child-mode and a guardian-mode when we came with the initial design, this vision relies on the vision of the launcher project.

The initial idea of the WOMBAT system was a three-application tool to help educators illustrate time for the children they work with. One parts is the timer application as it is implemented, from which it is possible to run customized timers, such as an hourglass or a digital watch. This part should be available only from the guardian-mode.

The second part of the system is a timer overlay, which is launched when other applications, for example game 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 3.1).

Figure 3.1: Example of how we imagined how the timer overlay would look. Here there is an overlay on the Angry Birds game with about 40 minutes left.

If the launcher is in child-mode, and the child opens some 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 long time is left of the allowed time. When the time is run out, the application is automatically closed.

The settings 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" time.

3.2 Stories

Stories can help understand situations where technologi is used by one or more persons.

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, so 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. Trine tries to comfort him by making him think of something else. It does not work very well, and then she takes out the tablet from her bag. She knows that he loves to play a certain game on the tablet. 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, as she suspect that Casper would feel better by then. 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 using a drawing application on the computer (see figure 3.2).

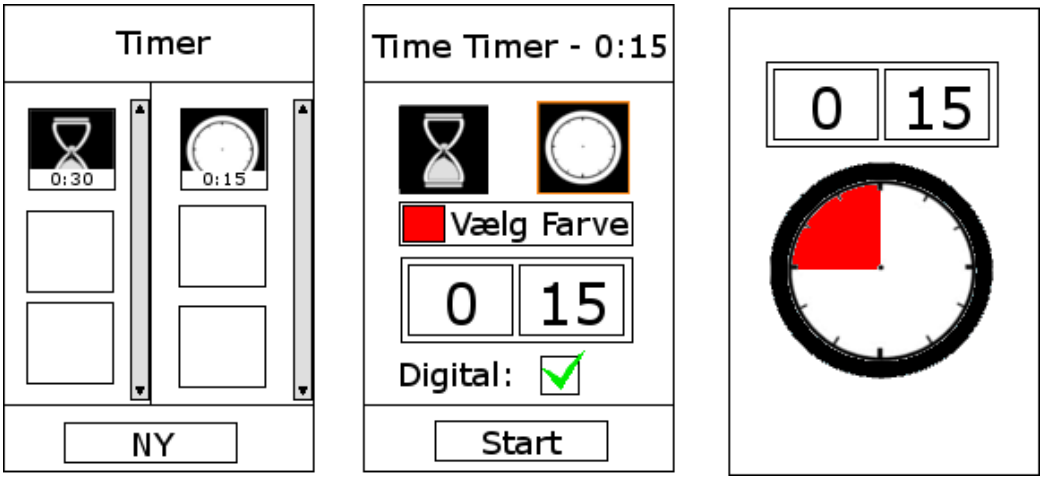


Figure 3.2: Drawing of the initial ideas of the timer application.

Paper Prototyping

Paper prototypes[?] 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. Furthermore it made it easier for us to program, when there was a specific design to work towards. In figure 3.3 is a paper prototype of the menu in the timer application, and the rest of the prototypes can be found in appendix 7.5.

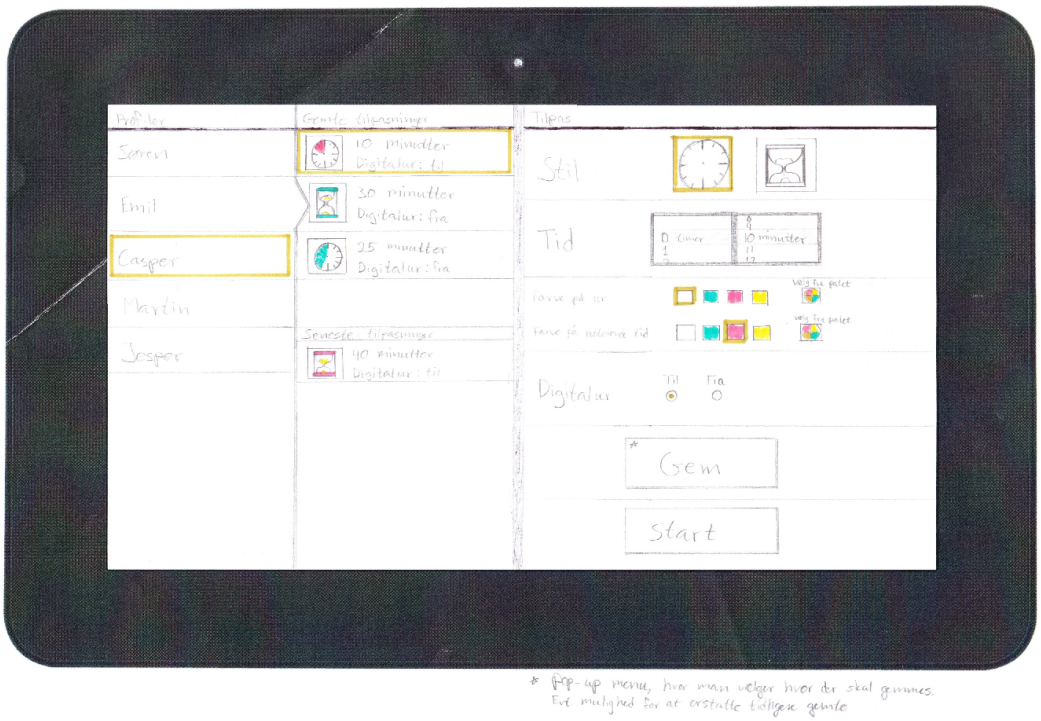


Figure 3.3: Scan of a paper prototype of the menu in the timer application.

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[?] 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 3.4 examples of the implemented metaphors are shown along with screen-shots of other systems they are implemented in.

[1]Non-free word processor developed my Microsoft

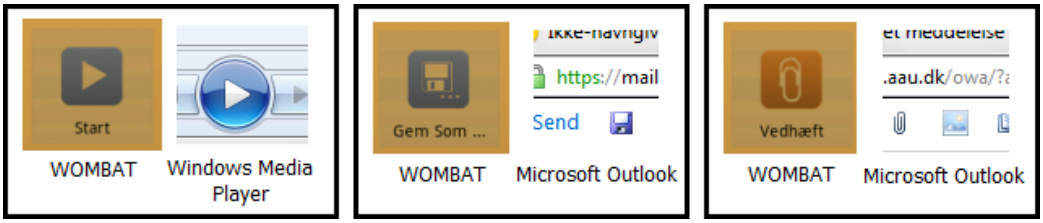


Figure 3.4: Metaphors implemented in WOMBAT vs. original implementation.

CHAPTER 4

Development Process

As stated in chapter 2, 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 7.6 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 time has run out, it would be possible to show two custom pictograms on the screen. Also she suggested 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 it was on 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

[1] Widely used 2D/3D graphics API

started making test design and test cases for the functionalities we want to test in the timer application.

Sprint 6: 14/05 - 18/05

In this sprint we implemented the last critical functionality. The only thing on the backlog we did not implement is the pictures in the profile list, because of missing functionality from the admin group. This part is only for showing the picture of the children on the profile list items, so it is not an important function.
Besides getting done with the development, we started with the acceptance test, by letting the contact person use the tablet with the timer application in everyday scenarios in the institution.

CHAPTER 5

Implementation

Nice...

5.1 Architecture

UML diagram over biblioteker
Flow chart over WOMBAT livscyclus
Arkitektur diagram over WOMBAT
Kort beskrivelse af hvad en aktivitet er

5.2 Fragments

In short a fragment in Android is functioning much like iFrames in HTML, meaning that a fragment is an embedded activity within another activity. According to Google the design philosophy of fragments 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. An example of the use of multiple fragments on a large screen compared to activities on smaller handsets, is the Gmail application from Google:

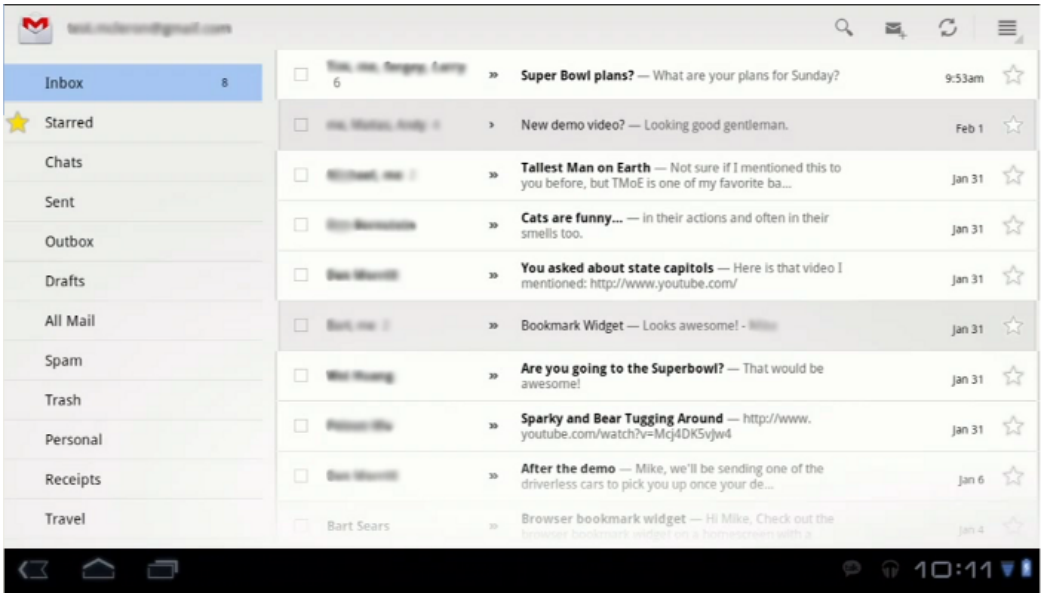


Figure 5.1: The Gmail application from Google on a tablet with fragment support.

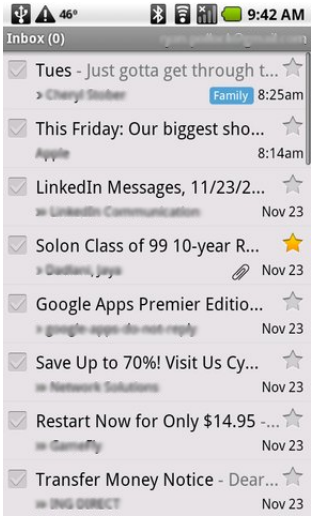


Figure 5.2: The Gmail application from Google on a handset without fragments

This and more information about fragments can be found at [**?**].

5.2.1 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 utilizes this by having an overview of the email account on the left side of the device while showing specific mail content on the right side as seen in figure 5.1.

Fragments are built to 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 has to declare which fragments it holds in the layout.

Fragments were introduced the first time with the Android 3.0 platform (API version 11). But as of 3rd March 2011 Google has made the "Android Compatibility Package", a library which let Android 1.6 devices or newer support fragments [**?**]. Mobile tuts+, a website about tutorials for Android development, has tested the "Android Compatibility Package" and they successfully used fragments with the support library on the T-Mobile G1 (the first Android device) [**?**].

5.2.2 Fragments in WOMBAT

In WOMBAT we use fragments to give the user the ability to quickly switch between multiple children and configuration templates, when developing with fragments we are able to let each part of the screen update independently based on actions in the fragments. This means that we, much like the Gmail application, can keep an overview of all children and their personal configurations, while having a detailed configuration page open at the same time. Thus avoid requiring the user to switch screen to load previous defined configurations.

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.

5.3 Lists

In Android the philosophy behind lists is that the majority of all applications at some point will have to implement a list object. Therefore does Android

ship with a build in list view, that supports a variety of list. This build in list view can be modified to match any list, that the developer wishes to create, as shown in 5.3.

Figure 5.3: Examples of how ListViews can be modified.

5.3.1 Benefits and Limitations

The advantage of a list view is, that we can modify the list to whatever we want it to look like without having to create any complicated programming to make it look good. This is done by making an adapter with the layout that the items of the list is going to look like as in figure 5.4, which is the adapter we use to show the children list.

```
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);
  }
  // FIXME: Insert pictures from admin here
  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 5.4: Example of an adapter to a list view, the actual adapter used in the children list

Creating lists in adapters and adding adapters to list views is alot easier than making custom lists. Because these built in list views are very versatile and can be configured to do exactly the same as any other view in Android.

Since lists wont be able to do anything but show a list of items without an on click listener does the list view have an interface which works exactly like any other clickable item in android. The on click listener in the list view is as easy to implement as a button or similar clickable object.

5.3.2 Lists in WOMBAT

Figure 5.4 is the actual implementation of the child list in WOMBAT, the only difference from the child list adapter and the configurations adapter is that the timer configurations require another text field with a description of the timer. Which means that the configurations list only requires another layout to be constructed, which holds the extra text view.

The on click listeners of both lists, ensures that the next fragments and Guardian object, described in **??**REF MANGLER , is updated figure 5.5 is an example of the child fragment listview.

```
1       public void onListItemClick(ListView lv, View view, int
            position, long id) {
          // Update the fragments
3         SubProfileFragment detf = (SubProfileFragment)
              getFragmentManager()
               .findFragmentById(R.id.subprofileFragment);
5         CustomizeFragment custF = (CustomizeFragment)
              getFragmentManager().findFragmentById(R.id.
              customizeFragment);
          custF.setDefaultProfile();
7
          if (detf != null) {
9           // Marks the selected profile in the guard singleton
            guard.profilePosition = position;
11          guard.publishList().get(position).select();
            guard.profileID = guard.publishList().get(position).
                getProfileId();
13          detf.loadSubProfiles();
          }
15      }
```

Figure 5.5: Example of the on click listener of the child fragment

5.4 Customize

The customize fragment is the heart of the application, this is where everything is generated and configured. The fragment can both create new and modify already existing timers, this is the key functionality of the customize fragment.

The timers that the guardians already use at the institutions comes in many different colors and timespans. Therefor is it possible to modify the look, time, and color of the individual timers in the customize fragment. Furthermore did the guardians state that a good feature would be if either pictograms or other timers were attachable to the timers.

Combined with the general idea of the application with a child and timer overview, is the requirements of the features in the customize fragment 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 a pictogram or a timer to the timer
- Change the done picture of the timer
- Save the timer
- Start the timer

Det hører næsten mere til et design afsnit

5.4.1 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 is the color pickers where the user can change the colors of the time left, the frame, and the background of the timer. The last element is the advanced element where the modifications like attachment can be found. Figure 5.6 show how the fragment has been outlined.

Figure 5.6: An outline of the elements of the customize fragment.

In the actual customize class has all buttons been initialized as global objects for quick reference. Since everyone on the WOMBAT project were new to Android programming when the structure of the class were stablished, is all functionality of the items in the customize fragment in this class. For further development would a refactoring of the entire class be adviced such that all buttons in WOMBAT is handled like the WDialog.
To be able to get an overview of the items in the customize fragment has

all items been assigned a method which initializes the item e.g. the style chooser in figure 5.7. These methods are then referenced from the onCreate method in figure 5.8.

```
1  private void initStyleChoser() {
     hourglassButton = (Button) getActivity().findViewById(
3        R.id.houglassButton);
     hourglassButton.setOnClickListener(new OnClickListener() {
5
       public void onClick(View v) {
7        selectStyle(formFactor.Hourglass);
       }
9    });

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

15     public void onClick(View v) {
         selectStyle(formFactor.TimeTimer);
17     }
     });
19
     progressbarButton = (Button) getActivity().findViewById(
21       R.id.progressbarButton);
     progressbarButton.setOnClickListener(new OnClickListener() {
23
       public void onClick(View v) {
25       selectStyle(formFactor.ProgressBar);
       }
27   });

29   digitalButton = (Button) getActivity().findViewById(R.id.
       digitalButton);
     digitalButton.setOnClickListener(new OnClickListener() {
31
       public void onClick(View v) {
33       selectStyle(formFactor.DigitalClock);
       }
35   });
}
```

Figure 5.7: The style choser initialization method, which utilizes the selectStyle that changes the style of the timer and highlights the button.

```
public void onActivityCreated(Bundle savedInstanceState) {
  super.onActivityCreated(savedInstanceState);
  currSubP = new SubProfile("", "", 0xff3D3D3D, 0xffFF0000, 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 5.8: The onCreate method, which calls the button initializers in the same order as they are shown in the layout.

For successors on the WOMBAT project does Eclipse have a keyboard shortcut (F3) which jumps to the method your cursor is at, that can be used to quickly find e.g. the color buttons in the customize class.

5.4.2 Buttons in Customize

There is roughly four kinds of buttons in the customize fragment:

- Start, Save, and style buttons
- Time picker wheels
- Color picker
- Attachment and Done picture button

Start Button

The style, save, start, and attachment buttons are ordinary Android buttons with a picture attached to the top side. The differences of these buttons is the onclick event handler which is slightly different from button to button, figure 5.9 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 5.9: 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 wheels is a wheel widget created by `yuri.kanivets@gmail.com` [?], these wheels can be customized to mimic the time pickers from the iPhone. 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

library, which is a built in feature in the Android development environment. To avoid a situation where the user would try to set the time to more than 60 minutes, did 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 5.10.

```
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 5.10: The time picker wheels, with the functionality which ensures the time is never set to more than 60 minutes

Color Picker

Just like the time picker wheels did we utilize the widget functionality in Android to implement a widget called AmbilWarna (means Take Color in Indonesian), created by Yuku Sugianto [**?**], to handle the color picker. 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 5.11.

```
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 5.11: The color picker widget implement on the second "Time Left" button

Attachment Button

The attachment buttons are implemented like the other buttons, but they make use of a custom dialog in WOMBAT. We created this dialog, because the standard dialogs in Android were very different from the rest of the WOMBAT design. Also were we unable to define exactly what kind of buttons we wanted in our dialogs.
Therefor did we implement the custom dialog which would match the rest of our WOMBAT design and give the ability to control exactly what the but-

tons on the dialog should look like and what they should do when clicked. Figure 5.12 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 arg0) {
    attachment1.cancel();
  }
});
```

Figure 5.12: 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.

5.5 Backend Library

The two libraries in WOMBAT each control a key functionality of the application, storage and drawing. These functionalities could have been implemented in the WOMBAT project, thereby avoiding situations where the project is compiled with an old library.
A problem with external libraries is that debugging can be difficult in Eclipse if the developer want to change directly in the library. This is because they require the external library to be imported in Eclipse and set as an external library in the Android properties of the project.
The benefits of using external libraries, is that they can be tested and debugged on an external test project instead of depending on a working main project. External libraries can be modelled by dummy functionalities in the main project untill they have been tested. Further, it becomes possible to change the library operation and thereby shifting entire modules in the application. Fx could the canvas drawings be changed to OpenGL drawings just by changing the canvas DrawLib in WOMBAT with a DrawLib that runs on OpenGL.

5.5.1 TimerLib

5.5.2 DrawLib

DrawLib contains the functionality that draws timers according to the settings set 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 attached, or it can be split screen if there is a timer or pictogram attached. This is done by creating the view layout programmatically instead of a static layout, figure 5.13 is an example of this functionality with a full screen timer, figure 5.14 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 5.15 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 5.13: Example of how DrawLib sets just one timer 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 5.14: Example 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 5.15: The genDrawView, which generates the draw view matching the timer

Canvas and OpenGL comparision

The timers in DrawLib are drawed with the Android canvas object, this could also 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 can only draw in two dimensions. Drawing on a canvas is like drawing on coordinates, while drawing in OpenGL varies in the way that one will be drawing in triangles. Drawing in OpenGL is therefor much different from the way one would normally draw, thereby more time consuming. Since all timers in WOMBAT can be modelled in two dimensions, the only benefits of OpenGL compared to canvas is that one would be able to move the camera around or change the lighting in OpenGL.

All timers are classes which inherits the view class, the view class draws in a method called onDraw which is called the first time it is opened and whenever the method `invalidate()` is invoked. The timers draw inside the onDraw method, as seen in figure 5.16.

```
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 5.16: The onDraw method of the progress bar.

The progress bar and the hourglass both draws according to how many percent of the total time is left, while the digital watch and the time timer draws according to the exact time left. All timers evaluates the time left

according to the number of milliseconds elapsed, which is provided by the system timer.

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

CHAPTER 6

Test

The WOMBAT application is tested through dynamic black box testing, which means that we run the application, and test the functionality through the user interface, without viewing the code. We have made test design and test cases for some of the most important functionality, so that we could outsource the test to one of the other project groups, if they had time to help us. This was not the case, so we did the testing ourselves.

6.1 Test Design

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

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 (RAM).

- 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 - *checkDoneFunc#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.

6.2 Test Cases

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*

6.3 Test Results

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

Table 6.1: Test results of the black box testing of the WOMBAT application.

6.3.1 Reflections

We did not outsource the test, so we did the testing ourselves. This is not the best way to do testing, because we wrote the program, and the test design and test cases, and thereby have a deeper knowledge about the features we want to test, and therefore we may have done things different than persons with no knowledge about the application would have done.

The only test that did not pass was known beforehand (*checkLastUsed#1*). We had trouble implementing the "Last Used" together with the features of the database developed by the admin-group. This resulted in sporadic system failure, so we decided to only use the tablet memory to save the last used timers, and save the correct implementation of this feature to future development.

During the test we stumbled upon a few vague formulations, which could have lead to misunderstandings and different testing method, if the tests had been outsourced. We learned that it is a good idea to run the test ourselves at first, so that we could correct any formulation errors we found, and then the test could be outsourced. This would require even more time, and is not likely to be done in large scale projects, but could have been done in our case.

6.4 Usability Test

6.4.1 Results and Observations

Test resultater og observationer
Tabel over hvilke funktionaliteter blev der fundet fejl ved og var det kritisk eller kosmetisk

6.5 Acceptance Test

The purpose of an acceptance test[**?**] is to test if the system fits into the context it is designed for. In this case the main objectives is to find out if it feels natural for the educators to use the timer application instead of their regular hourglasses and time timers, and to find out if the children understand the digital version of the different timers.

To test for acceptability, the system needs to be tested in the context it is developed for, and therefore the contact person borrowed the tablet with

the timer application for a few days, so that she could use the system in real life scenarios.

6.5.1 Results and Observations

The contact person wrote a diary, to keep track of her experiences with the application. The contact 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 7.7.

They wrote in the diary that the children understood the meaning of both the timers and the pictograms shown on the tablet. One of the children was more interested in watching the digital timer count down than in the activity he was meant to do, but beyond that there were no problems understanding the system.

Beside the diary, we can analyze on the timers they have used in the acceptance test by looking in the "Last Used" list (see appendix 7.7).

We can see in the list that they have used different timers with different "Done"-pictures and times, 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 anyway.

Tail

Part III

Discussion

Head

CHAPTER 7

Reflections and Evaluations

Hvordan har brugen af pair programming og refactoring fungeret
Hvordan har SCRUM fungeret som projekt styring

7.1 Conclusion

Konklusionen paa projektet

7.2 Future Work

Port to handset
OpenGL timers
Remake of design (drawer with profiles etc.)
Remake gradient Button
Implementation af vision om overlay

7.3 Conclusion

7.4 Future Work

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

7.4.1 Making a Handset Version

It is, at the moment, not possible to install and run the WOMBAT application on an Android handset, but there are several reasons for considering making a handset version of the application aswell.

7.4.2 Better Graphics

7.4.3 New Design Features

Tail

Part IV

Appendix

Appendix

7.5 Paper Prototypes

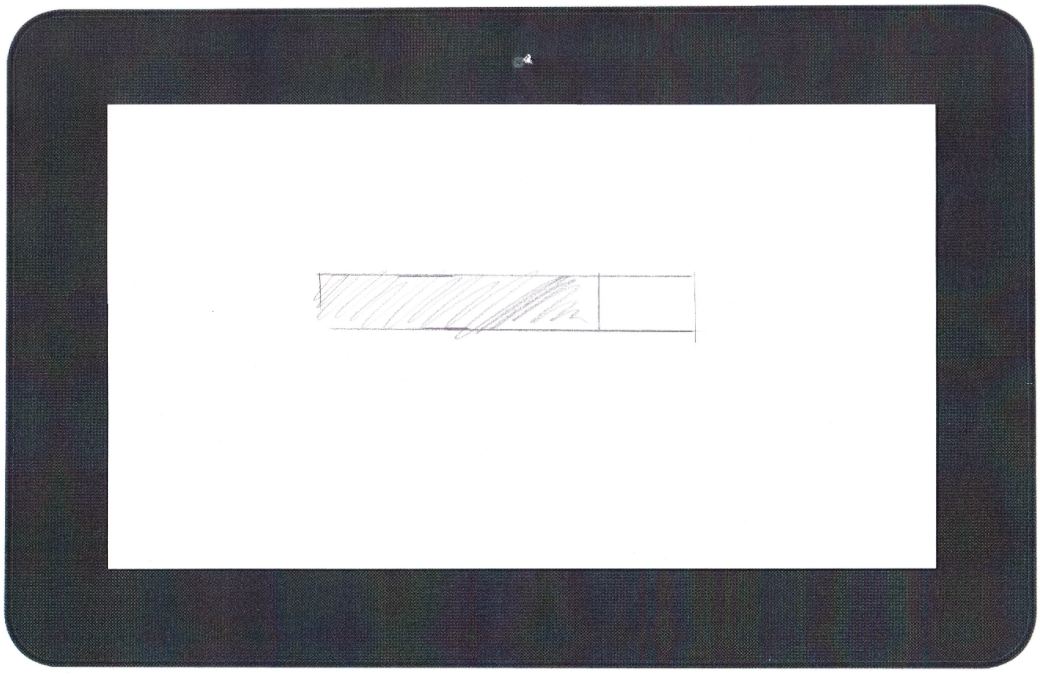


Figure 7.1: Scan of a paper prototype of a single progress bar timer.

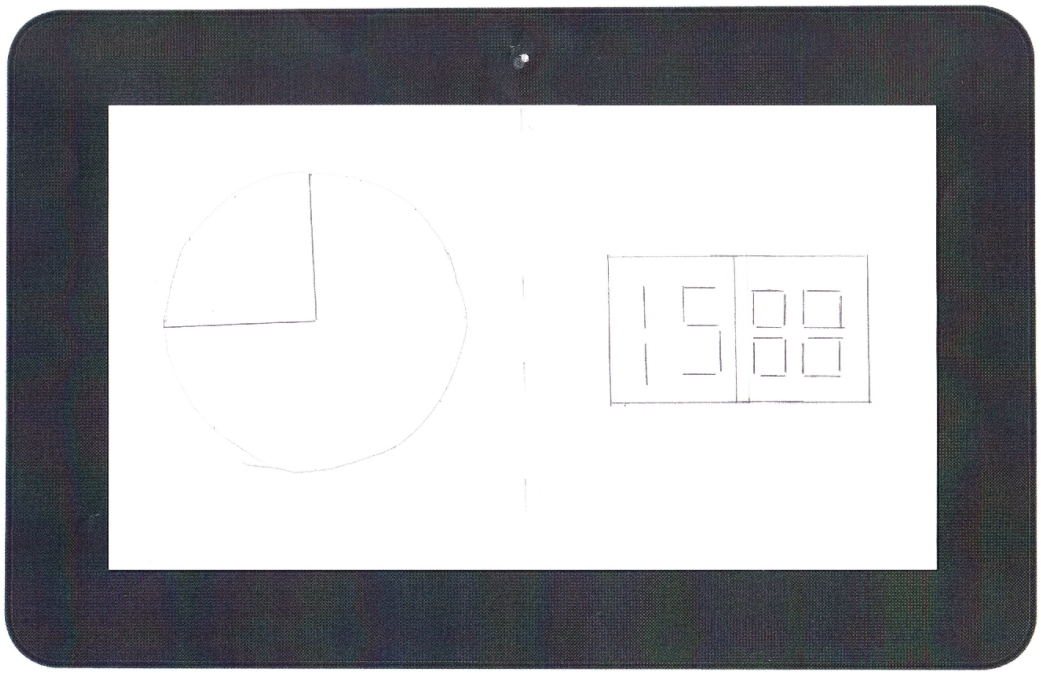


Figure 7.2: Scan of a paper prototype of a double timer.

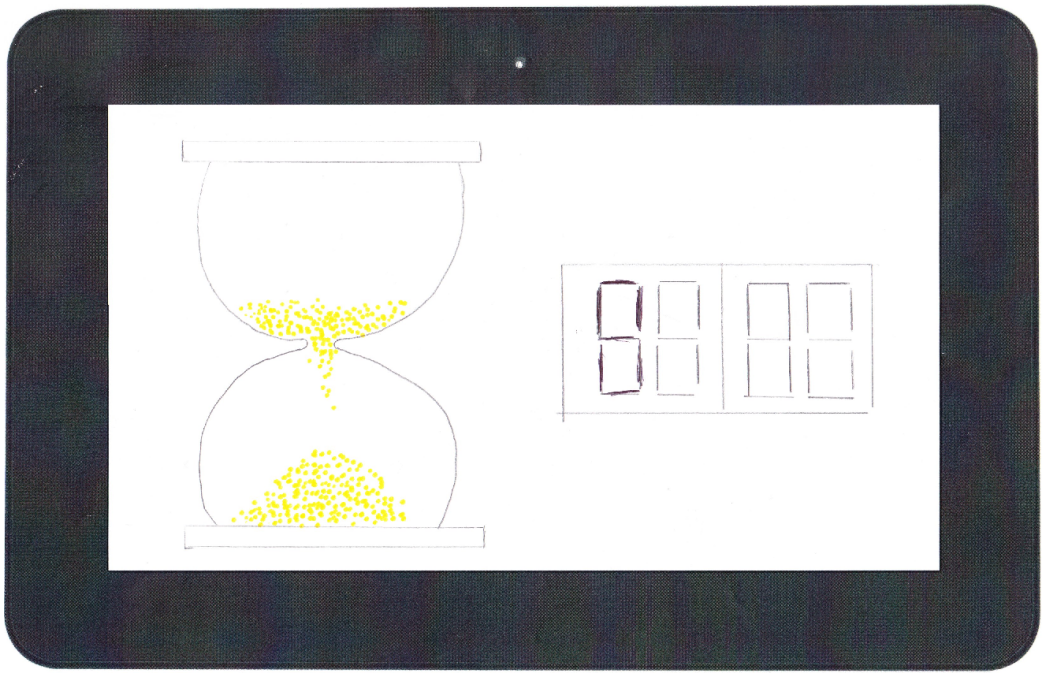


Figure 7.3: Scan of a paper prototype of another double timer.

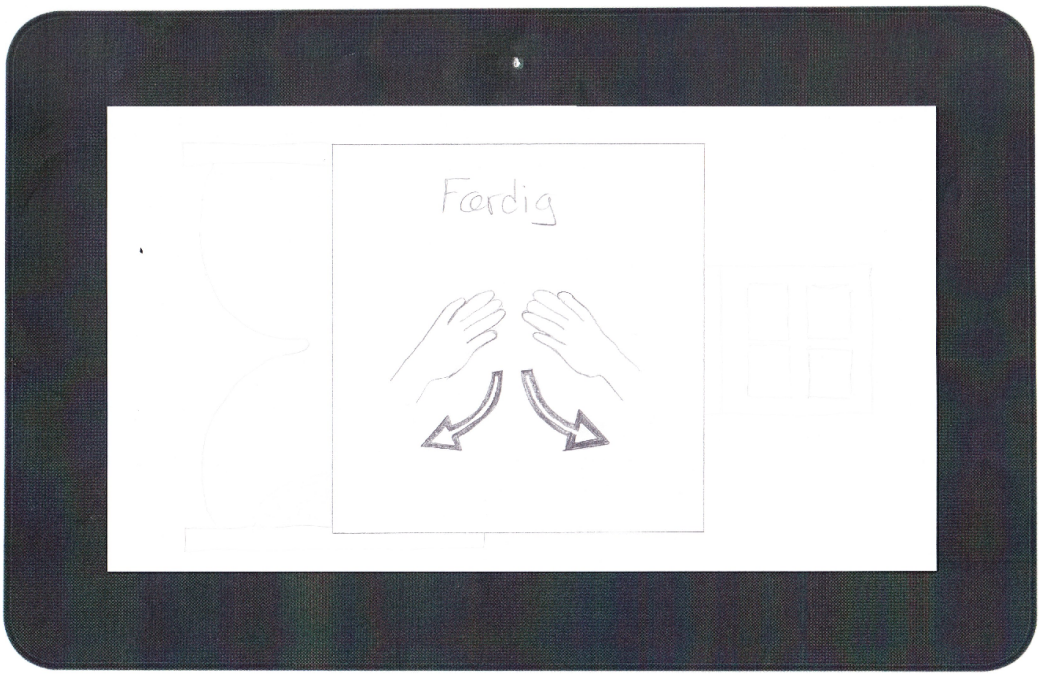
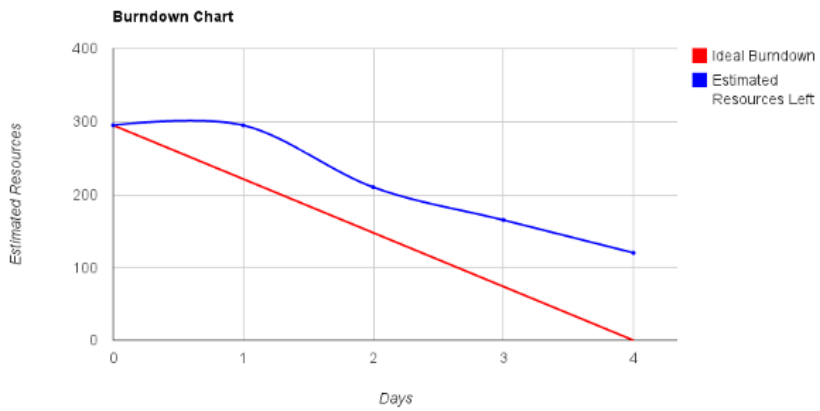


Figure 7.4: Scan of a paper prototype of the "Done" screen shown when the time has run out.

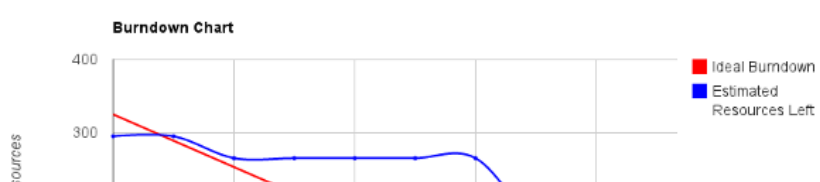
7.6 Sprint Burndown Charts and Backlogs

Timer (WOMBAT)

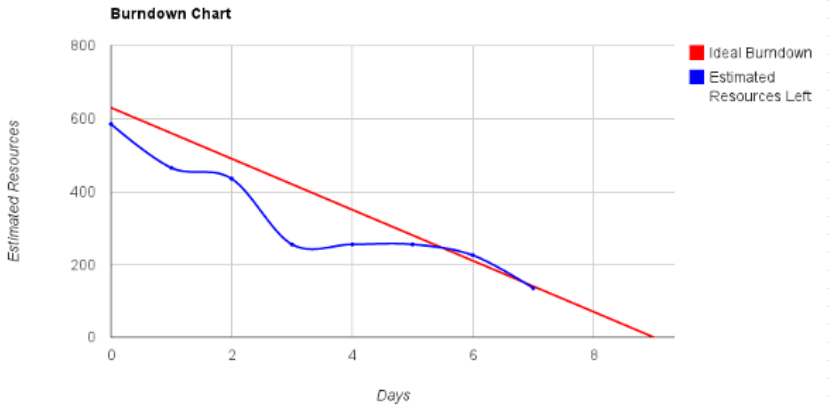


ID	Name	Priorite	Estima	How to demo	Note	Status
1	Design Start activity	14	25	Two lists, 1. = lists of list names, 2. list of profiles		*DONE*
10	Develop Start activity	13	60	According to the design [1], program an "input" for the two lists and make them clickable		*DONE*
5	Develop Load Settings	12	60	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)

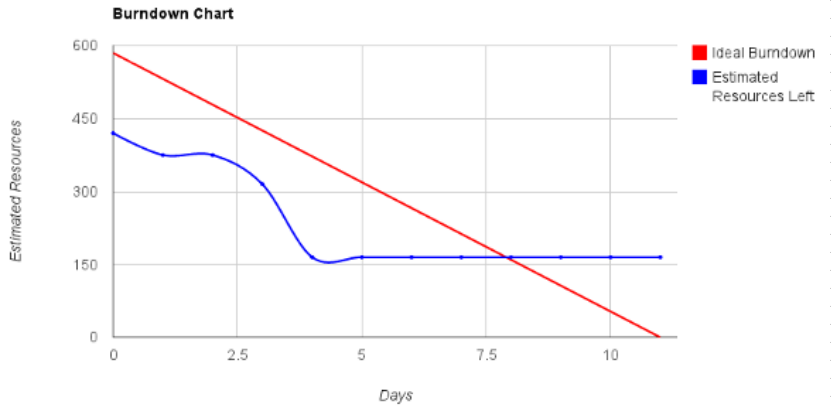


Timer (WOMBAT)



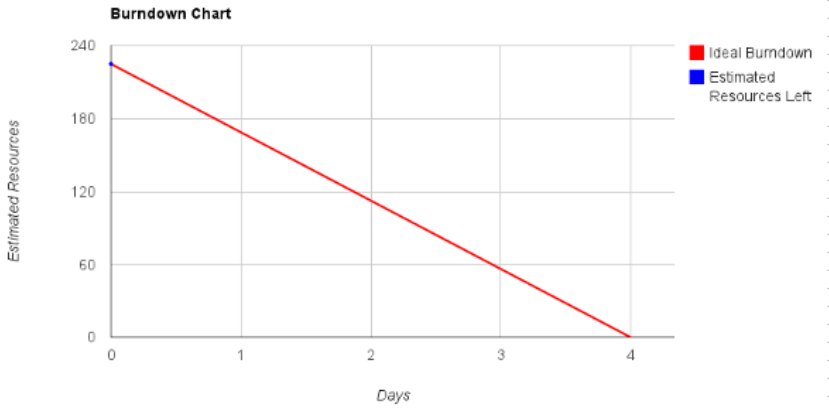
ID	Name	Priorite	Estima	How to demo	Note	Status
5	Develop Load Settings	19	60	Load relevant attributes from admin	*Function at Admin*	*In progress*
6	Develop Save Settings	21	30	Save relevant attributes to admin	*Function at Admin*	*In progress*
9	Last Used Algorithm	22	30	Design how the last used algorithm 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)



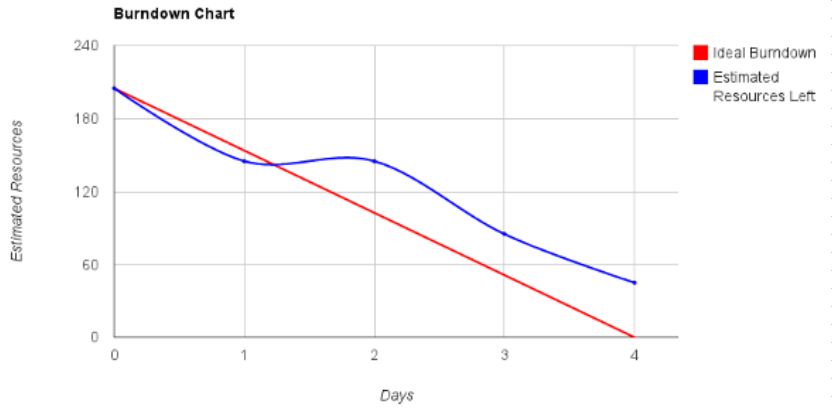
ID	Name	Priorite	Estima	How to demo	*Note*	*Status*
21	Digital Watch	44	30	According to [3]		*DONE*
38	Tekst på delete subprofile	46	15			*DONE*
39	Undim Screen	47	45			*DONE*
40	Gemte profil skal highlightes	48	30			*DONE*
18	Hour Glass		45	According to [3]		*DONE*
41	Vedhæftet skal ændre/update tid 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*

7.7 Acceptance Test Diary

Dato: 16/5 - 12

Mads

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

Dato: 18/5 · 12

Figure 7.5: Scan of a paper prototype of the "Done" screen shown when the time has run out.

7.8 WOMBAT Setup for Eclipse

Import Wombat, AmbilWarna (the color picker), and wheel projects from svn to Eclipse figure 7.6.

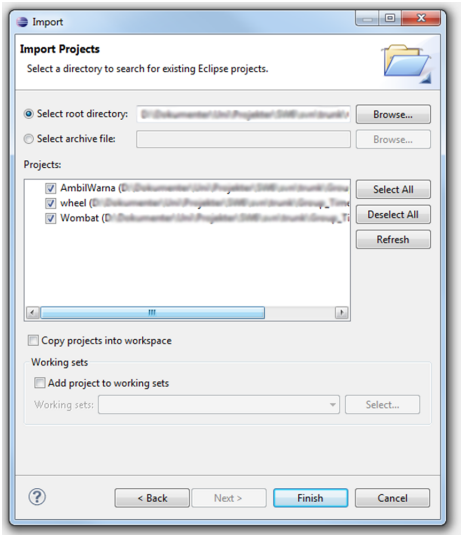


Figure 7.6: Import Wombat, AmbilWarna, and wheel.

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 7.7.

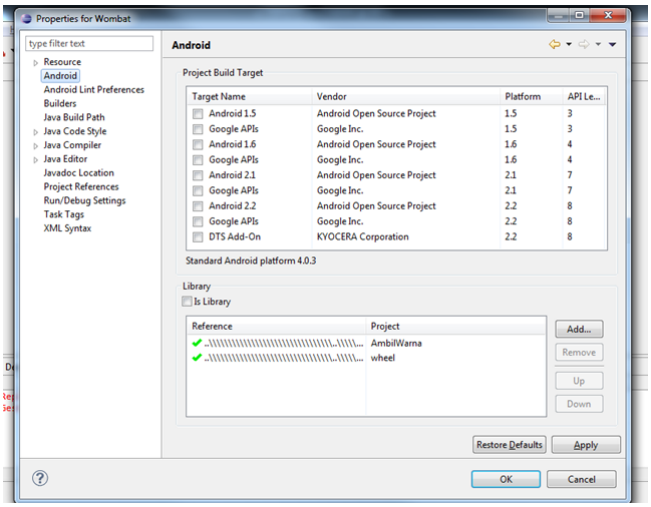


Figure 7.7: Add AmbilWarna and wheel as project libraries.

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 7.8.

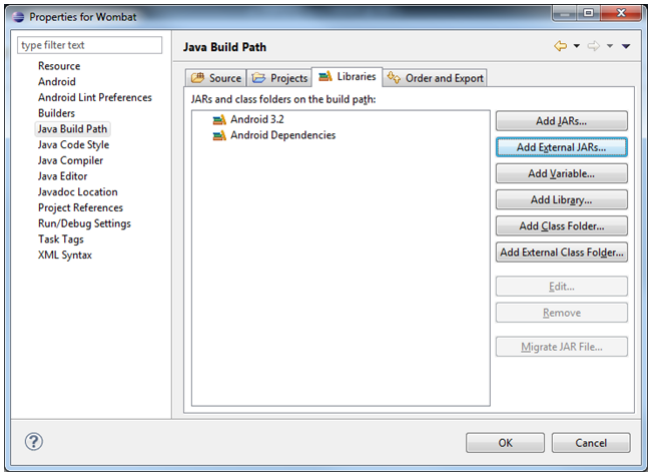


Figure 7.8: Add Oasis, TimerLib, and DrawLib as external JARs if Eclipse cant recognize them in the libs folder.

Extras

To edit the TimerLib and DrawLib the two projects has to be imported to the workspace in Eclipse, figure 7.9.

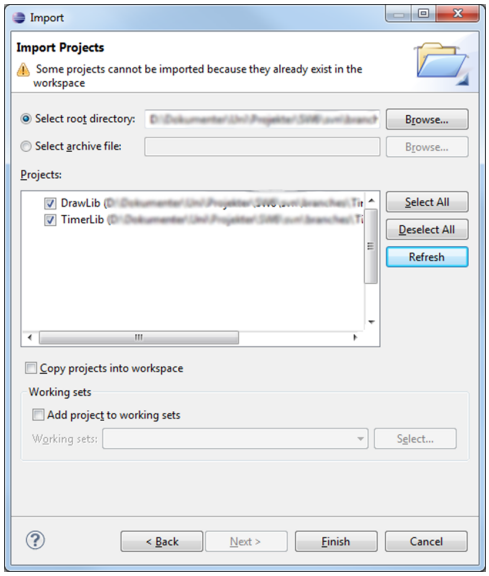


Figure 7.9: Import TimerLib and DrawLib to the workspace.

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 7.10. 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.

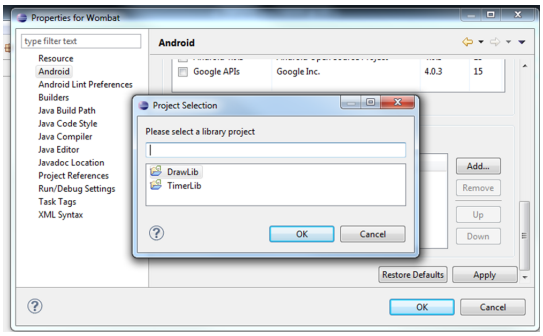


Figure 7.10: Add TimerLib and DrawLib as project libraries.