Title goes here

Developing complex software systems

Aalborg University
Software – 2015
Project group SW605F15

AALBORG UNIVERSITY
STUDENT REPORT

Department of Computer Science
Aalborg University
Selma Lagerlöfs Vej 300
9210 Aalborg East
http://cs.aau.dk/

Title:

Title name here

Theme:

Developing complex software systems

Project:

6^{th} semester project

Period:

February - May 2015

Group:

SW605E15

Authors:

Claus Nygaard Madsen
Lukas Nic Dalgaard
Martin Juul Johansen
Sean Skov Them

Supervisor:

Bin Yang

Synopsis:

Copies:
Pages:
Appendix:
Finished: 29-05-2014

Signatures

Claus Nygaard Madsen

_____

Lukas Nic Dalgaard

_____

Martin Juul Johansen

_____

Sean Skov Them

_____

Preface

Reading guide

Contents

Introduction 1

GIRAF is a series of applications(apps) for Android, intended to help citizens with autism in their everyday life. Over the past four years GIRAF have been developed by students at Aalborg University, with a new group of students taking the mantle every year. As a result of this, it was hard to get an overview of the project in its entirety. We here give an overview of the GIRAF project as it was, when we started working on it, and we also give some insight into the organization of the project.

1.1 Status of GIRAF

Overall GIRAF is in a somewhat functional state with several problems ranging from small to severe regarding stability, reliability, and usability. GIRAF is published in Google's Play Store with the individual apps being version 1.0, while the launcher-app, the main app the other apps should be executed from, is version 2.2. The version-numbering is not entirely correct, as some of the apps does not function correctly yet and as such they should not have been published as official versions but rather as alpha or beta versions. The majority of the apps does function correctly, but only a few of them are ready for public release. Google Analytics was incorporated in roughly one third of the apps, with between three and six months of data on usage and crashes being available for these apps. The data has been gathered since the newest version of the apps were published, in general since the 25th of August 2014.

1.2 Multiproject, Subprojects, and Responsibility

The overall goal for the multiproject this year was, to make a working system that is usable and stable. A multiproject in this context being an overall project that several groups, in this case all 6th semester software groups, work on in collaboration. In order to achieve a better workflow, the project was split into three subprojects: Database (DB), Graphical User Interface(GUI), and Build and Deployment(B&D). Our group were a part of B&D, along with 3 other groups. Both the DB and GUI subprojects had 5 groups each. See Figure 1.1 for an overview of the structure of this years multiproject.

1.2.1 Database

DB groups were responsible for handling all things database related, this included making current data accessible and adding new data to the database, as per request from other groups or as needed.

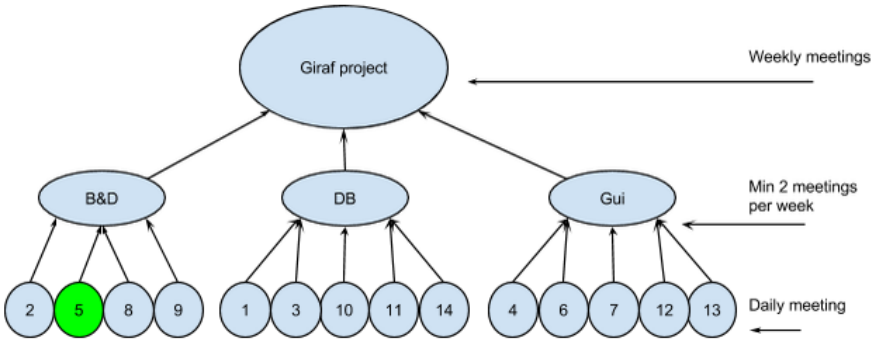

Figure 1.1. Overview of the project structure with sub-projects and the individual groups depicted, our group is marked with green.

1.2.2 Graphical User Interface

GUI groups were mainly responsible for developing the different apps, this included making general changes in the apps and bug fixing anything related to the apps.

1.2.3 Build and Deployment

B&D groups were responsible for most of the internal project matters, this included maintaining the server services, developing and maintaining the automatic build tools, handling publication of the apps, and administrating the version control.

1.3 Project organization

The project is organized using Scrum principles where the individual subprojects held Scrum of Scrum meetings twice a week. The individual groups can follow any principles that they would like, but Scrum was recommended. Our group decided to follow Scrum to have the same structure as the project. In addition an overall meeting was held every week where at least one representative from each group were present. At these meetings the overall status for each subproject were given, and any decisions that needed overall consensus were discussed.

As the overall structure is Scrum, the project was split into four sprints. The first sprint was devoted to bug fixing, user stories and tasks that were still unsolved from last year, and setting up and organizing project tools. In regards to user stories, it was decided early on at an overall meeting, to operate with two types of user stories, those from the costumers, these were called user stories; and those from the other groups, called developer stories. For consistencies sake we will just refer to both as user stories in the report, as the difference was mainly a concern when groups from different subprojects where discussing internally.

1st Sprint 2

For the first sprint our main user story was setting up and continuously support Google Play and Google Analytics. We also had a user story to write guidelines for Javadocs, and we were given the role as Product Owners for B&D.

2.1 User Stories and Tasks

This section describes the sprint planning for the first sprint.
In accordance with normal Scrum practices, we started with a sprint planning session. It was held in collaboration with the other groups for our subproject Build and Deploy (B&D), which consisted of groups 2, 5, 8, and 9, each consisting of 4 people. One of the first things the entire project team, not only B&D but the other sub-projects as well, did in the project was to scour through the reports from last year's project and construct an initial backlog to work on for the first sprint. Then we split the user stories between the relevant sub-projects.

User Stories	Tasks
Setup for Google Play	Translate descriptions to English
	Update the description of apps
	Beta upload for Google Play
Javadocs	Test javadocs
	Construct guidelines
Google Analytics Crash Reports	Create code guidelines for implementing Google Analytics
	Autoforward crash reports
	Check implemented code

Table 2.1. User stories and related tasks for sprint 1

The main goal for this sprint, as reflected by the user stories, was to acquaint ourselves with the project and the Google services.

2.2 Google Play and Analytics

This section describes our work with Google Play and Google Analytics.

In order to setup and support Google Play and Google Analytics, we were given the administrator login to both services. In the following we describe our work with the two

services.

2.2.1 Google Play

Google Play is a digital distribution platform created by Google. It is used as an appstore for android devices as well as for music and ebook distribution. Because the goal of the project is to make apps for android devices, we are using the Google Play store to distribute the different apps. [Goo, 2014]

As it was our responsibility to maintain and control Google Play for this semester, we started by familiarizing ourselves with the different features provided by Google Play. We found that Google Play provides a lot of statistics for all the apps, like how many times the apps have been installed and how many are still using them. Google Play also allowed us to change which apps should be shown on the store and to change the descriptions of the apps. One thing we noticed when we looked at some of the descriptions was that a lot of them were very short and did not provide enough information about the apps, so the user would have trouble knowing the functionalities that the apps provide. Because of this, we decided to update the descriptions, making an English version of the descriptions in the process, and to update some of the pictures showing the apps to the user.

One data that Google Play provided, which was very important, was the statistic about which version of Android the users were using. This allowed us to know that the apps have to use Android API 15, because 22% of the users were using Android version 4.0.3 - 4.0.4, which only supports API 15 and below. [API, 2014]

Alpha/Beta

At the conclusion of last year's work on the Giraf project, all the apps from the project were uploaded to Google Play. This meant that the apps became available for the users, and that data about crashes could be sent to Google Play and Google Analytics. The initial release was plagued by crash issues and other errors. Some of these issues were attributed to certain applications crashing when trying to use other apps.

To better avoid this for the current semester, it was decided to add an alpha and beta release version to Google Play. The alpha and beta versions fulfill the same role of ensuring that build is stable before release, but they are updated very differently. The alpha version will be updated whenever a new build is uploaded onto the Jenkins server and ~~it~~ passes the automatic tests to ensure it does not break the build. If it passes all of that, this new build will replace the alpha version available on Google Play. The beta version will be updated at each sprint end to the newest alpha build. There were not any plan for when to update the official release, but it follows that stable beta versions can be released officially.

For the work in this implementation of alpha and beta versions, we found a plug-in for Jenkins that could be used to make the uploading of alpha and beta versions automatic, as well as providing the information the server needs to upload to Google Play. The plug-in and information was then passed on to the group responsible for the Jenkins server, so they could implement the automated upload.

2.2.2 Google Analytics

Google Analytics is a service created by Google. The service generate statistics from website traffic. Google Analytics main function is to track visitors and display advertising, to show how people are using the products. Mobile App Analytics is a branch of Google Analytics, which focus on mobile apps. Mobile App Analytics has google play integration which allows for visitors/traffic sources tracking. this traffic can then be use to see who is installing the apps on which platform. Mobile App Analytics also provide some API's for tracking events, Crash, Exception, and user patterns, to see how the users use the app.

Mobile App Analytics and Google play have a lot of the same functionality for tracking who installed what on which platform. But Mobile App Analytics has an advantages when it comes to crash/exception rapports, because API's send a crash report directly to Mobile App Analytics, without the user has to do anything, where Google play don't include crashes that aren't reported by users. For this reason Mobile App Analytics were used as the main crash report service.

For crash reporting there were three task at hand. the first was to autoforward the crash reports for a submodule to the groups that worked on that submodule. The second and third task was to provide a guide on how to implement the API's into the apps and to provide a guide for the next years Google Analytics group on how to auto forward crash reports.

Autoforward crash reports

Mobile App Analytics has an build in Email feature, which allow the user to automatically email a crash report daily, weekly, monthly or quarterly. It is not that intuitive how to auto forward reports because the user has to first make an report that a able to filter to the time interval the user wants. Therefore an guide was made which can be found in appendix A

Mobile App Analytics API

The Crash API for Mobile App Analytics has to be implemented if crashes are to be fund. The API makes it possible to make a Tracker which can be used to report information to Mobile App Analytics, when an exception is caught. An great feature about the tracker is that it can be set to report uncaught exception.

2.3 Central Documentation

As a part of the central documentation user story, we looked into how we could streamline documentation of the code for the project by using Javadocs to provide a clearer overall form of documentation.
Javadocs is a plugin for Eclipse that automatically generates comments for coding blocks. In essence, it generates a comment stub automatically for a code block with a blank field space for all parameters.
The comments can then be automatically gathered and put in order into a html file which

partly acts as code documentation.

```
1  /**
2  * Short one line description.                    (1)
3  * <p>
4  * Longer description. If there were any, it would be [2]
5  * here.
6  * <p>
7  * And even more explanations to follow in consecutive
8  * paragraphs separated by HTML paragraph breaks.
9  *
10 * @param variable Description text text text.      (3)
11 * @return Description text text text.
12 */
13 public int methodName (...) {
14 // method body with a return statement
15 }
```

Figure 2.1. An example listing of how a javadoc should look is shown below. Note the double star at the beginning of the comment.

The motivation for using a a javadocs-like system was to improve the overall comment approach through standardization and ease of use. An added bonus is that it can provide code documentation on the project as a whole, which is a challenge under normal circumstances.

At the start of the project, Javadocs was partially fulfilling its role. It was used in some projects consistently, while in others, comments of any kind were missing.

Our responsibility was twofold, we had to check up on whether javadocs were usable across the entire code base and if it weren't, then we should make sure it got fully implemented. The other responsibility followed the first. Once it was ensured that javadocs was properly installed, a guide to Javadocs were to be distributed to the rest of the project groups and placed on the wiki. After some light testing, we could conclude that Javadocs were fully deployed and made a tutorial for the other users.

In the same sprint, another group decided to replace javadocs with another plugin called Doxygen for code documentation purposes. Overall Doxygen serves the same purposes as javadocs did. Doxygen accepts the same syntax as Javadocs, so migration was very smooth.

2.4 Product Owner

For the first sprint we took on the role as product owner B&D. This meant that we were responsible for receiving requests and requirements from the groups in GUI and DB. It was then our job to form new user stories based on the requirements. We were also the ones who handled the sprint end and the planning of the next sprint.

In the first sprint we did only got a few request and requirements but this was probably

because a lot of the time allocated to the first sprint was used to get knowledge about the project. We decided to keep the role as product owners for B&D during the next sprint.

2nd Sprint 3

For the second sprint our main user story was setting up and continuously support Google Play and Google Analytics. We also had a user story to write guidelines for Javadocs, and we were given the role as Product Owners for B&D.

3.1 User Stories and Tasks

This section describes the sprint planning for the second sprint.

After the end of the first sprint. We started sprint planning anew. As the Project Owners we were responsible for proper sprint planning for the Build and Deploy subproject. Below can be seen the user stories and associated task we found for our second sprint for our group.

User Stories	**Tasks**
Google Analytics	Provide support
Update Google Play for sprint # 1	Collect APK's and changelogs
	Sign APK's
	~~Request changelog~~
	~~Upload and update Google Play description~~
	~~Make signing guide~~
Update Google Play for sprint # 2	Collect APK's and changelogs
	Sign APK's
	~~Upload and update Google Play description~~
Stand-Alone App	Find dependencies
	Check the Local database
	Check GUI design
	Check how synchronization with database work
	Ensure consistency between original and Stand-Alone version
Core Program	Isolate dependencies
	Design and make the core package
	Publish dependency report

Table 3.1. User stories and related tasks for sprint 2

For our second sprint, the two biggest objectives for Build and Deploy was to continue cleaning up the project and start looking into making the applications work standalone.

3.2 Dependencies

In this section all the dependencies between apps and libraries are described, with the goal of finding out what is needed to get the apps to run and which dependencies would have to be removed to be able to make apps standalone.

3.2.1 Motivation

Finding the dependencies for apps and libs, would help in understanding which dependencies would have to be removed before an app is independent. It would also help the jenkins group in making test on apps, since the knowledge of which apps an app is dependent on would help them in knowing which apps would have to be installed before an app can be tested. Lastly having an overview of the dependencies would make it easier for groups working with apps and libs, by informing them about which apps and libs they can be affected by and which they have an effect on.

3.2.2 Applications and libraries

The first thing we did to find the dependencies was to make a list of all the apps and libraries that are used in the project. These apps can be found in Table 3.2 below.

Apps "Repository name"(Name of app)	**Libs**
launcher (Giraf)	giraf-component
zebra (Sekvens)	oasis-lib
wombat (Timer)	local-db
croc (Piktotegner)	pictogram-lib
cat (Kategori-værktøjet)	sequence-viewer
cars (Stemmespillet)	category-lib
train (Kategori Spillet)	TimerLib
oasis-app ("administrations værktøj")	DrawLib
tortoise (Livhistorier)	Ambilwarna
ugeplan	barcodescanner
piktosearch (Tal for mig)	
parrot (Piktooplæser)	

Table 3.2. List of apps and libraries

3.2.3 Libraries apps are dependent on

After making the list of all the apps and libraries, we found the dependencies for all the apps by opening the project for them in Android Studio, an IDE, and then look at, which libraries were included in the build gradle files.

From this information we made a list of all the libraries the apps are dependent on. This list can be seen below.

Apps	Dependencies
launcher	→ giraf-component, oasis-lib, local-db, barcodescanner
zebra	→ giraf-component, oasis-lib, local-db, pictogram-lib
wombat	→ giraf-component, oasis-lib, local-db, pictogram-lib, DrawLib, Timer-Lib, wheel
croc	→ giraf-component, oasis-lib, local-db, pictogram-lib
cat	→ giraf-component, oasis-lib, local-db, pictogram-lib, catagory-lib, Ambilwarna
cars	→ giraf-component, oasis-lib, local-db
train	→ giraf-component, oasis-lib, local-db, pictogram-lib
oasis-app	→ giraf-component, oasis-lib, local-db
tortoise	→ giraf-component, oasis-lib, local-db, pictogram-lib
ugeplan[1]	→ giraf-component, oasis-lib, local-db, pictogram-lib
piktosearch	→ giraf-component, oasis-lib, local-db
parrot	→ giraf-component, oasis-lib, local-db, pictogram-lib, catagory-lib, Ambilwarna

Table 3.3. List of library dependencies for apps

As can be seen in the list of dependencies, all the apps were dependent on 'giraf-component', 'oasis-lib', and 'local-db'. In addition to this there are a lot of the apps that are dependent on 'pictogram-lib'.

To get a better overview of the dependencies we made a graph see Figure 3.1 below. On the graph the squares represent apps and the ovals libraries. An arrow from A to B signifies that A is dependent on B. Because there are a lot of dependencies between the apps and libraries, we decided to exclude 'giraf-component', 'oasis-lib', and 'local-db' from the graph. This meant that the graph was a lot easier to read, and because all apps where dependent on those apps, we could just add a short note about the dependency.

[1] Opens through tortoise

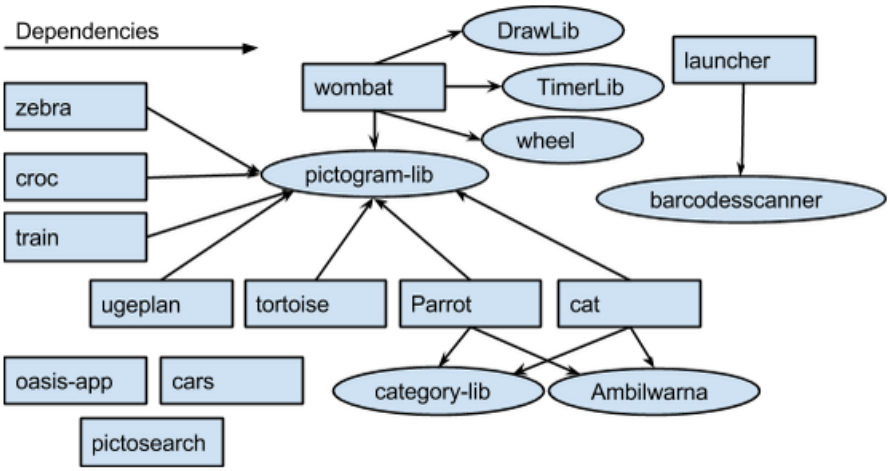

Figure 3.1. Graph of library dependencies for apps

3.2.4 Cross library dependency

After having made all the library dependencies for the apps we did the same procedure for the libraries to find out which libraries where dependent on which library. As with the apps we started by looking at the build gradle files for the libraries, and then made a list of the dependencies, which can be seen in the list below.

Libs	Dependencies
giraf-component	→ oasis-lib, local-db
oasis-lib	→ local-db
pictogram-lib	→ giraf-component, oasis-lib, local-db
sequence-viewer	→ giraf-component, oasis-lib, local-db, pictogram-lib
local-db	→ metadata
category-lib	→ giraf-component, oasis-lib, local-db, pictogram-lib
Drawlib	→ NULL
TimerLib	→ NULL
Ambilwarna	→ NULL
barcodescanner	→ NULL
wheel	→ NULL
metadata	→ NULL

Table 3.4. List of cross library dependencies

As with apps/library dependencies we also made a graph for cross dependency between the libraries shown in Figure 3.2. As in the last graph the ovals are libraries and the arrows signifies dependencies. In this graph we decided to show all dependencies since there was

no dependency that was the same for all libraries and because the number of dependencies was low enough, to be able to show it all without having to many lines crossing each other to be able to read the graph.

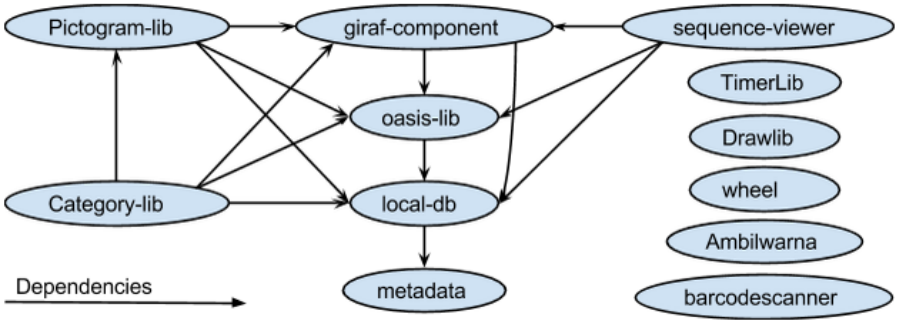

Figure 3.2. Graph of cross library dependencies

3.2.5 Cross App dependencies

As the last step in finding the different dependencies we found the dependencies between the different apps. We did this in two ways, skimming through the code for any calls to another app, and by trying to run the apps on a tablet and testing if any of the apps stopped working if another app was not installed. From the dependencies we made a graph which is shown below in Figure 3.3. Like on the other graphs the Squares are apps and the arrows signifies dependencies, but in addition this graph also have dotted and red lines. The doted lines signifies that the app is not directly dependent, but instead dependent through another app and the red lines are dependencies we are uncertain about.

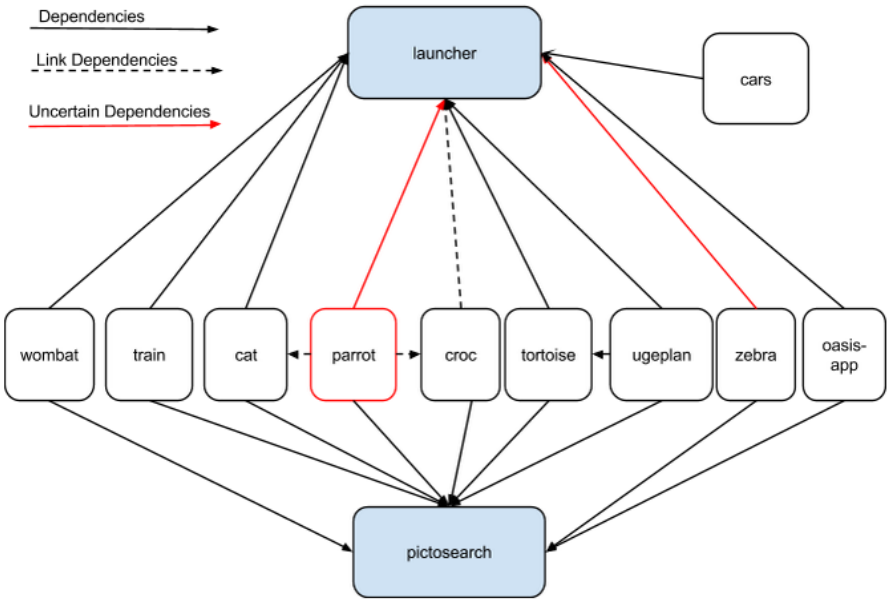

Figure 3.3. Graph of cross app dependencies

3.3 Standalone

The outside costumer showed some interest in having some apps available as standalone. This section focusses on our work in this area covering which features would be needed to make the existing apps standalone and our decision to make a core library.

3.3.1 Motivation

Making the apps standalone would increase the usability of the individual apps, as the user would no longer need to install several different apps just to ensure that they can use the app they want. As seen in 3.2 all apps in Giraf are reliant on both the launcher and 'PictoSearch'. This dependency is not only annoying for the user but is also causing trouble during the automatic tests, as one of the tests include installing the apps and testing them pseudo-randomly, but the emulation used for this test could not install more than one app at a time, which results in this test always failing on apps reliant on the launcher. The process of making the existing apps standalone will likely result in a solution that would also make it easier to make new apps standalone in the future.

3.3.2 Features

In order to make the individual existing apps standalone, we will first need to look at each app and find out which tasks the apps has to be able to perform when working standalone. For each app we will then describe the features that would enable them to become standalone apps.

Zebra

Zebra is an app that allow the user to make sequences of pictograms, which is used for communication. In order to use the app, it needs access to a database with pictograms, and if the database has not be made by the launcher, Zebra has to be able to make a database and download the pictograms to that database. In order to allow the user to use personal pictograms, the app also need a login feature, so that the user can access these pictograms as well.

Croc

Croc is an app that enable the user to make personal pictograms using the camera, existing pictograms or from a blank canvas. Like Zebra, Croc needs to be able to access pictograms from a database, and since Croc's works with personal pictograms it also need a login feature which enables the user to save the pictogram to a personal account.

Other apps

Like Zebra and Croc many of the other apps that needs to be standalone, needs the ability to access pictograms and personal pictograms. Therefore we will not go into details with these apps.

Overall necessities

So from this we have arrived to the following list of features the apps would need to work as standalone apps.

Login: This features will allow the user to access personal pictograms and save pictograms to a personal account. *Make Localdb*: This features has to check if a local database of pictogram has already been made by another app. If no local database has been found the app will then need to make it itself. *Download pictogram*: The download pictogram features has to download pictograms for the local database from the server. It also has to get personal pictograms for a user, if the user is logged in to an personal account.

3.3.3 Standalone App or library

As all the apps need the features just mentioned, it would make sense to place them in a common place, either as an app or as a library. That way multiple apps could use the same code without having to rewrite the code in the individual apps. Additionally any new common features needed could be added to this app or library. In general this will be referred to as a core app or core library. In Table **??** an overview is shown of the positive and negative aspects (pros and cons) of each solution. The two solutions were evaluated on how easy they would be for the user. This meant that the app-solution would be a bad solution because it would require the user to install another app in addition to the standalone app they wanted to use. The bigger size of the apps in the library-solution is regarded as a minor problem as currently the storage space for the pictograms used far outweighs the storage used by the installed apps.

Core standalone App	Core standalone Library
Pros	**Pros**
Smaller in size when more than 1 standalone app is installed	True standalone (no other apps needed)
Cons	**Cons**
Another app needs to be installed	Bigger app size
Other notes	**Other notes**
Fusion Core standalone and Pictosearch (only 1 'plugin' app)	Make Pictosearch part of the core library (no 'plugin' app)

Table 3.5. List of apps and libraries

3.4 AAR files

When the 2nd sprint started, the libraries used as dependencies by each app was copied into the project building the app. This resulted in many files and folders being present in several different app projects. As seen in 3.2 some libraries are used by most apps. Given that the libraries contain upwards of 4000 files this presents a potential problem if

for example some files are not copied correctly. To ensure that the libraries are the same across all apps, we used aar files that are compressed resource files not unlike zip files. The android version of Java Archive files, aar is more specifically a binary distribution of an Android Library Project, and it is a format used in Android development. **??** '.aar' files contains the following:

- /AndroidManifest.xml (mandatory)
- /classes.jar (mandatory)
- /res/ (mandatory)
- /R.txt (mandatory)
- /assets/ (optional)
- /libs/*.jar (optional)
- /jni/<abi>/*.so (optional)
- /proguard.txt (optional)
- /lint.jar (optional)

3.4.1 Motivation

By using aar files the number of files being shared could be reduced to just the one aar file, and with proper version naming, this would make it simpler to include the right library. The previous implementation also called each dependency recursively, which would then call its own dependencies, resulting in multiple calls to build the same dependency.

3.4.2 Implemention

To store and access these aar files, some of the other groups set up a maven repository. For these groups, we provided some consultation on the implementation. This way we avoid duplicates by storing the dependencies away from the application repositories, rather than each application storing it internally.

The following listing show how the implementation looks in the code for the launcher application:

```
1  dependencies {
2  compile ('com.android.support:support-v4:+')
3  compile ('fr.avianey.com.viewpagerindicator:library:2.4.1') {
4  exclude module:'support-v4'
5  }
6  compile fileTree(include: '*.jar', dir: 'libs')
7  compile project(':barcodescanner')
8  compile(group: 'dk.aau.cs.giraf', name: 'girafComponent', version: '1.2', ext:
       'aar')
9  compile(group: 'dk.aau.cs.giraf', name: 'oasisLib', version: '1.0', ext: 'aar'
       )
10 compile(group: 'dk.aau.cs.giraf', name: 'localDb', version: '1.0', ext: 'aar')
11 compile(group: 'dk.aau.cs.giraf', name: 'meta-database', version: '1.0')
12 compile files('libs/libGoogleAnalyticsServices.jar')
13 }
```

The code compiles the dependencies found in section 3.2 from the maven repository. The compile call finds the aar file specified by four parameters. 'dk.aau.cs.giraf' is the path

inside the maven repository. The name specifies the specific file. The version is the current version of the application being compiled. ext says the extension of the file called is aar.

3rd Sprint 4

Test

4.1 User Stories and Tasks

This section describes the sprint planning for the third sprint.

After the end of the second sprint. We started sprint planning anew. As the Project Owners we were responsible for proper sprint planning for the Build and Deploy subproject. Below can be seen the user stories and associated task we found for our second sprint for our group.

Our third sprint was off to a rocky start. Our initial main focus lay in renaming the google play package names and smaller administrative tasks.
However many of the tasks proved to be easier than estimated and were finished early, so we added core lib to the backlog.

4.2 Renaming of Package-names

One of our tasks for the 3rd sprint was the internal renaming of most apps and libraries. In this section we will cover the renaming process including the consequences of the method used.

4.2.1 Motivation

As there was little to no consistency in what the apps are called internally, that is on Git, Jenkins, and the package-names on Google Play, it was highly requested that some consistent names were made. A package-name is a text-string used by Google Play to uniquely identify the app. In the Giraf-project the package-names are on the form: dk.aau.cs.giraf.App, where 'App' is replaced by the internal name for the app in question. While in the process of renaming, it was also suggested that the apps got names that actually tell something about what they do, ie. not many would know what the app 'croc' does but when it is named 'pictocreator' it makes some sense. Because of the descriptive names and the consistency in naming, it would become much easier for new developers to understand what the different apps are.

User Stories	Tasks
Google Analytics	Provide support
Ensure App Name Consistency	Find internal names for apps and libs
	Inform the groups about when the change is made
	Make the changes on git (outsourcable)
	Make the changes on jenkins (outsourcable)
	Make the changes on google play
	Make the changes on google analytics
	Update pictures for all google play apps
Make PictoSearch a Library	Transform PictoSearch to a library
	Modify the code, so that it works on one of the apps
	If the process is simple, do the same for all the other apps
Code-documentation for aar files	Find out if it is doable and how
	Make a guide
Share the knowledge from Guides	Check for missing guides
	Check the content of the guides and update them
	Move guides from PDF format to the wiki page
	Inform groups about which guides exists
Release backlog	Make a release backlog for sprint 1 (if possible)
	Make a release backlog for sprint 2
	Make a release backlog for sprint 3
Backlog	Make a description of items in backlog
	Insert the backlog into the semester backlog
Core lib	Generate a lib project
	Insert the apps and libs into it
	Implement the lib in an app
	Implement the lib for the remaining apps

Table 4.1. User stories and related tasks for sprint 3

4.2.2 Initial decisions

Before we started renaming anything, we took a look at the old names, seen in Table 4.2. The names were a mixture of animal references and shorthand notations for the parts of the apps content. The animal names were originally created several years ago and while they might have made sense back then, it was decided to find new names for these. Additionally some of the shorthand notations were changed as well, to become more descriptive.

New names were suggested by us and after some discussion with members of the GUI-subproject, we settled on the names seen in the second column of Table 4.2. Everyone involved in the decisionmaking on the new names, feel that the new naming makes more sense in the way that they actually say something about the functionalities of the app.

4.2.3 Renaming process

The most important part of the project to have renamed, was the package-names used by Google Play, as due to Jenkins automatically uploading new versions of the app to Google Play, Jenkins would fail if the package-name was different from the one used on Google Play. Because package-names are unique Google have decided that they should not

Old app package-names	New app package-names
launcher	launcher
train	categorygame
pictoadmin	categorymanager
tortoise	lifestory
oasis	profilemanager
parrot	pictoreader
pictosearch	pictosearch
pictocreator	pictocreator
zebra	sequence
cars	voicegame
wombat	timer
schedulestarter	weekschedule

Table 4.2. Old and new package-names for the apps.

be changeable after uploading the app, therefore we were subsequently forced to bypass this system by uploading new apps, with the same content, to replace the apps, were we changed the package-name. Therefore we changed the names of the 'old' apps to be the name followed by 'retired' indicating that the apps would eventually be taken down. This was done to ensure that the other groups could continue working on the apps without being affected by the name change before they were ready to transition to the new name.

During this process we encountered a problem with one app not being uploadable. This was caused by some of the libraries having part of the same package-name, more specifically they were called 'dk.aau.cs.giraf.oasis.LibName', while one of the apps had previously been called 'dk.aau.cs.giraf.oasis'. Due to this problem we decided to rename the package-names of the three libraries causing the problem, in part also to get rid of the 'oasis' part of the name, as it did not tell anything about what the library was used for. Due to some unforeseen events in another group working on one of the libraries in question, we had to wait a few days before the renaming could be continued and finalized. The new and old package-names for the libraries can be seen on Table 4.3.

While uploading the 'new' apps we found that the descriptions of several apps could use an update, and we will suggest this as a user story for the next sprint.

4.2.4 Consequences

The way the renaming was done is not completely without consequences. As we had to upload essentially new apps, it was not possible to transfer the statistics from the old apps

Old library package-names	New library package-names
oasis.metadata	dbmetadata
oasis.localdb	localdb
oasis.lib	dblib

Table 4.3. Old and new package-names for the libraries.

to the new ones. In addition all current users will have to uninstall their current versions of the apps and then install the new releases in order to get updates on the apps in the future. Both of these consequences, while potentially very bad if the project had been more widespread, is not actually too bad. This is because the number of external users are low and we have ~~access to~~ direct contact with them, and the data and statistics gathered from the usage of the apps is not that useful in practise due to the rapid development on most apps. Therefore we see the renaming as an overall success.

4.3 Improvement of wiki pages

In this section is a description of how we in the 3rd sprint worked on improving the quality of the guides and information pages on the project wiki pages.

4.3.1 Motivation

Improving the quality of the information shared on the redmine wiki, like guides and summaries, would improve the workflow by shortening the time a groups uses on understanding the software configuration management tools used in the project, by providing useful guides and guidelines.

4.3.2 Process

Process As the first step we looked through the whole wiki and listed all the guides and pages that provided information. We then looked through the list and discussed in the group ~~about~~, if there were any guides that were not on the list that needed to be included. In this discussion we did not find any extra guides that we thought were needed on the wiki page. All the guides and information pages we found are shown on the list below:

- Git guide
- Android guide and guidelines
- Testing
- Issue Guidelines
- Google Analytics
- Javadocs
- Dependencies Management
- Guide for renaming Apps and libraries in android studio
- Continuous Integration Guidelines
- App and library renaming in Android Studio
- How to implement the Pictosearch library
- Process
- Repository names
- Dependencies between apps and libs
- Database architecture
- Requirements Specification and user stories for Giraf
- Requirements for Database Component
- Links
- Meetings

- Backlog sprint #4 last year projects
- Product backlog
- A list of groups and their responsibilities
- Word list, for terminology when talking/writing about the institutes.

After we had discussed if any guides were missing, we looked through all the guides again, but this time we checked if the content and the formating of the page were correct and of high enough quality. For the pages that did not provide good enough content, but where we had the knowledge ourselves to correct, we corrected ourselves, and for the one we did not have the knowledge to fix we contacted the group that had written it in the first place. For content there were only two instances of pages that did not live up to the standard, the first was "Dependencies management", where information about how to log in to the maven repository was missing, and the second was the jenkins guide which only had titles. The login information we inserted to the page ourselves. but for the jenkins page we had to contact the group responsible for jenkins in the project. It turned out that the wiki page was not supposed to have been made in the first place, so it was decided that the page got removed, instead of writing the whole guide, Because only the jenkins group would have use of the information, and they already had the knowledge themselves.

A lot of the pages did not have the correct formatting or did not have a formatting at all. For those pages we corrected the formatting so that the pages was easier to read. In addition to the formating, some of the pages also had most of their content in a PDF file. For those pages we wrote the content of the PDF file into the page and then placed a link at the top of the page where the PDF could be downloaded.

For some of the pages we also changed the title so that it was easier to understand what content was on the page without having to open the page. After having fixed all the guides and information pages on the wiki we send an email to all groups informing that we had fixed all the guided and a list of what guides was available at the moment.

4.4 Release Backlog

This section will deal with the user stories regarding the release of a backlog for sprint 1 and 2.

This is a section stub.

4th Sprint 5

Collaboration 6

7 Conclusion

Bibliography

Android 4.0.3 apis, March 2014. URL
http://developer.android.com/about/versions/android-4.0.3.html.

Google play, March 2014. URL http://en.wikipedia.org/wiki/Google_Play.

Google Analytics Crash Reports to Email Guide

A

This is a step for step guide for the setting up crash reports from Google analytics to email, for the google analytics group of 2016.

1. The first thing to do is to go to the google analytics website at http://www.google.com/analytics/
2. Then log in to the google account using the google account information provided.
3. The page should then look like shown below on figure A.1.

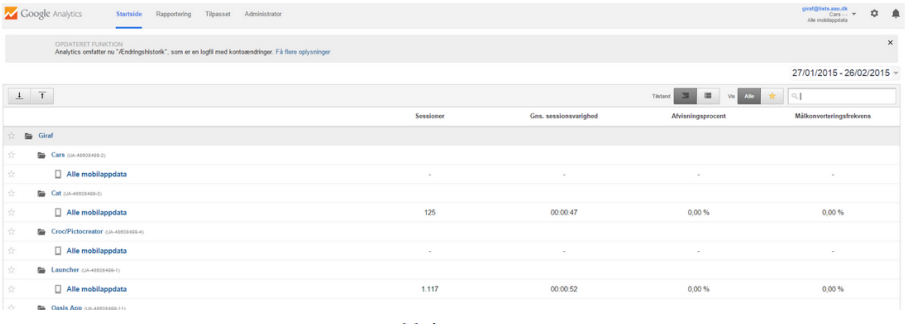

Figure A.1. Main page when logged in to Google analytics

4. Select the app you want to have a crash report from by pressing "Alle mobile appdata" Right below the app.
5. This should then lead you to a page like on figure A.2 the one below.

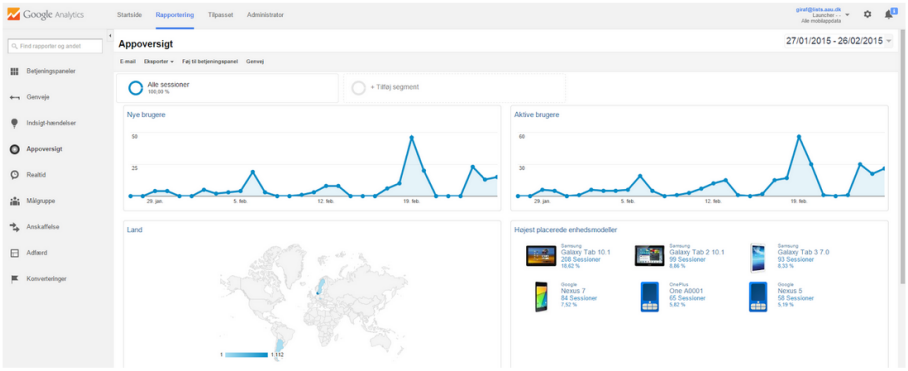

Figure A.2. App overview page

6. At the left most sidebar, press on "Adfærd"/"Behavior" and select "Nedbrud og Undtagelser"/"Crashes and Exceptions" in the dropdown menu.

7. The Page should then look like shown below on figure A.3.

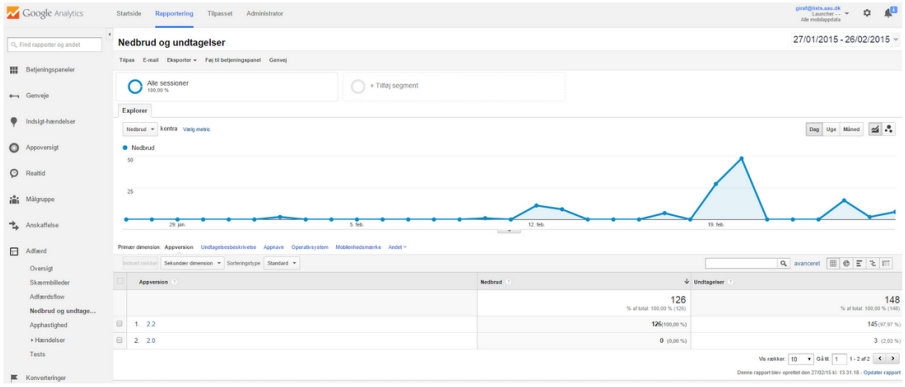

Figure A.3. Crashes and Exceptions page

8. At the bottom of the page click on the newest version of the app.
9. The page should then show all the crashes at the bottom of the screen.
10. After this go to the top right of the page and click on the date interval.
11. This should open a window like the one shown below on figure A.4.

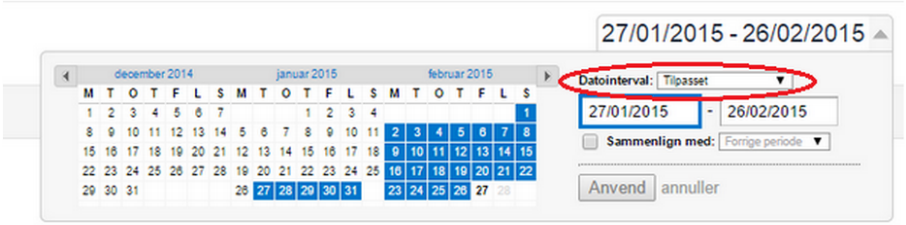

Figure A.4. Calender window

12. In the window select the time interval that you want the crash report to be sent from in the highlighted dropdown menu.
13. Press “Anvend”/”Apply” to close the window.
14. In one of the top menus there is a button named “Email”, press it.

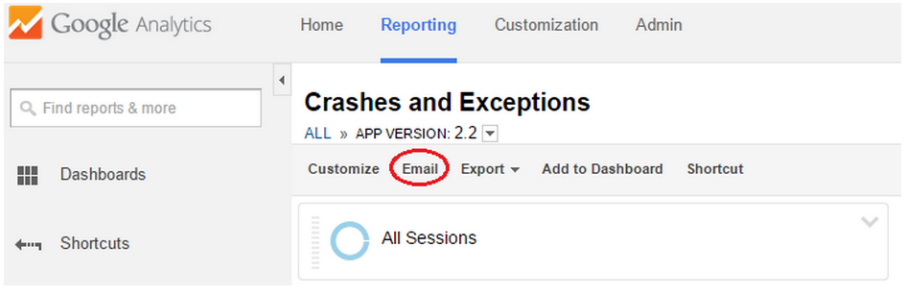

Figure A.5. Email button

15. In the window that opens select whom to send the mail to, what format the crash report should be in, and when the email should be send.

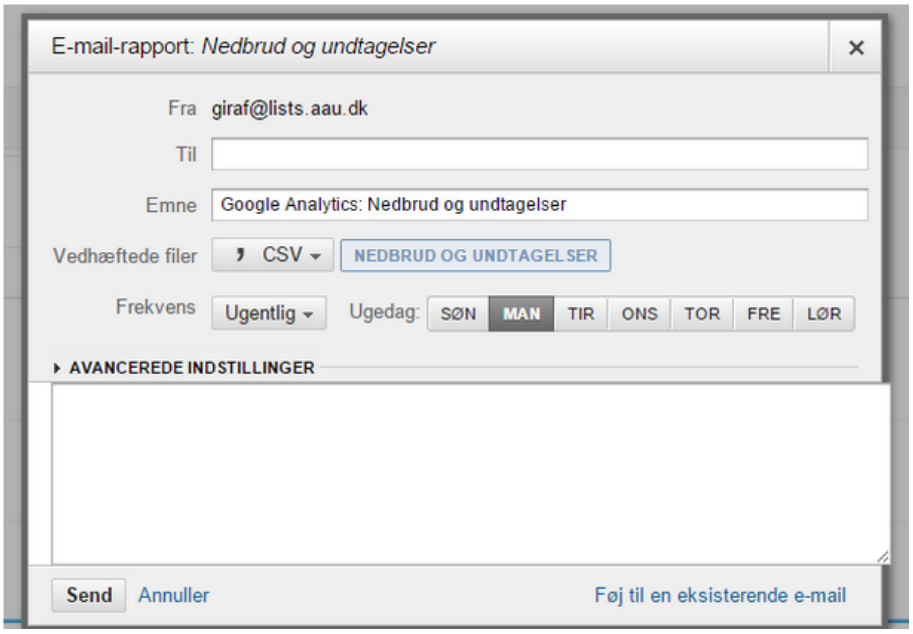

Figure A.6. Mail report window