eyeCam

Android “Augmented Reality App” zur Unterstützung der Dichromatopsie

Dominik Spengler, Patrice Müller

IFS INSTITUTE FOR SOFTWARE

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL

FHO Fachhochschule Ostschweiz

Erklärung der Selbstständigkeit

Hiermit versicheren wir, die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die Zitate deutlich kenntlich gemacht zu haben.

Rapperswil, den 3. Juni 2011 Dominik Spengler, Patrice Müller

Impressum

Kontakt

Dominik Spengler
Patrice Müller

Homepage

https://github.com/tom-and-jerry/eyeCam

Abstract

Die Farbenblindheit ist eine seltene Farbsinnstörung, bei der teilweise Farben nicht erkannt werden können, sondern nur Kontraste. Die häufigste Art der Farbenblindheit ist die Rot-Grün Sehschwäche. Durchschnittlich 8 Prozent aller Männer sind Farbenblind. Die Farbenblindheit lässt sich in zwei wesentlich Problemstellungen unterteilen:

Unterscheidung von verschiedenen Farben in einem Bereich mit gleichem Kontrast.

Erkennung um welche Farbe es sich handelt.

Bei eyeCam handelt es sich um eine Augmented Reality Android Applikation welche zum Ziel hat, Farbenblinde in diesen beiden Problemen zu unterstützen. Diese Funktionalität wird durch die folgenden Features erreicht:

Farbfilter welche das Vorschaubild der Kamera soweit verändert dass Farbunterschiede erkennbar werden.

Farberkennung bei Berührung des Bereiches der Vorschau.

Durch die oben genannte Funktionalitäten ist es nun möglich für Farbenblinde Personen Farben zu erkennen und erleichtert das Lesen von Farbecodierten Bildern.

Inhaltsverzeichnis

Teil I

Technischer Bericht

1 Einführung

Tabelle 1.1: Dokumenthistory - Einführung

Rev.	Datum	Wer	Änderung
0.1	05.03.2011	Patrice Müller	Dokument erstellt
0.2	05.03.2011	Patrice Müller	Einführung in die Problemstellung
0.3	07.03.2011	Patrice Müller	Umfeld der Arbeit erstellt
0.4	08.03.2011	Patrice Müller	Umfeld der Arbeit fertiggestellt
0.5	07.04.2011	Dominik Spengler	Vorgehen, Aufbau der Arbeit erstellt

1.1 Problemstellung

Bei der Farbenblindheit handelt es sich um eine Erbkrankheit welche durch einen Gendefekt im X-Chromosom hervorgerufen wird an welcher wischen 7 und 10 % aller Männer leiden[1]. Es besteht allerdings auch die Möglichkeit aufgrund eines Gehirn- oder Augenschadens beispielsweise durch das Schaken baby syndrom[2] an Farbenblindheit zu erkranken.

Die Farbenblindheit kann je nach Fähigkeit Farben zu erkennen in drei Typen unterteilt werden[3]:

- Rot / Grün Farbenblindheit
- Blau / Gelb Farbenblindheit
- Totale Farbenblindheit

Zwischen den einzelnen Sehstörungen gibt es viele unterschiedliche Stärken der Sehschwäche. Gemeinsam ist Ihnen jedoch, die Schwierigkeit kontrastähnliche Farben zu unterscheiden. Dies führt zu einigen Problemen im Alltag, wie zum beispiel bei der Flaschentrennung, oder der Erkennung ob eine Toilette besetzt ist.

Das Problem der Farbenblindheit kann man in zwei wesentliche Probleme aufteilen:

1 http://www.colour-blindness.com/general/prevalence/
2 http://de.wikipedia.org/wiki/Shaken_baby_syndrom
3 http://www.colour-blindness.com/variations/

- Die Unterscheidung von Farben d.h. überhaupt zu erkennen ob es sich bei einer Fläche um zwei verschiedene Farben handelt.
- Die Kategorisierung von Farben d.h. die Fähigkeit zu erkennen um welche Farbe es sich handelt.

1.2 Vision

Unser Applikation soll genau zwei oben genannte Probleme für Farbenblinde lösen.

1.3 Aufgabenstellung, Ziele

Die Aufgabenstellung kann man grob in vier Teile unterteilen, Bedürfnis- und Konkurrenzanalyse, “Development environmentünd die Applikation selbst.

Development environment: Es existiert ein Build-Server der nach selbst definierten zyklischen Abständen jeweils von definierten Branches Builds generiert. Als Build-Server dient hier Jenkins. Ob es nun Jenkins oder ein anderer äquivalenter Server ist, ist eigentlich egal, solange er die Möglichkeit bietet das “Ctoss-Device Engineeringßu realisieren.

Bedürfnisanalyse: Die Studenten sollen in einem ersten Schritt in die Lage sein sich in eine Person die an Dichromatopsie leidet zu versetzen und sich somit ein Mindset erarbeiten das ihnen bei der Entwicklung der App helfen soll. Die App selbst sollte später mit mindestens zwei Probanden getestet bzw. entwickelt werden. Die Bedienbarkeit und der Nutzen soll an Hand von mindestens drei Aufgaben, im Rahmen eines Usability-Tests gezeigt werden.

Konkurrenzanalyse: Bevor die Studenten mit der eigentlichen Entwicklung der Applikation beginnen, sollten Sie eine Analyse der bestehenden App’s durchführen. Dabei ist es zu empfehlen, das man die zwei Probanden des Usability-Test einbezieht.

Applikation: Es soll eine “Augmented Reality Appërstellt werden, welche die Probleme der Konkurrenz aufgreift und im idealen Fall verbessert. Das Ziel ist es eine App zu entwickeln welche am ende des Projektes im Android-Market zu finden ist.

1.4 Umfeld der Arbeit

1.4.1 Rahmenbedingungen

Aufwand Mit dieser Semesterarbeit erwerben die beiden Studenten Dominik Spengler und Patrice Müller jeweils 8 ECTS-Punkte. Aus der allg. Definition der ECTS-Punkte ergibt sich folgende Rechnung: 8 ECTS-Punkte à 30h = 240h pro Student über 14 Wochen. Somit entspricht das einem Arbeitsaufwand von ca. 17.2h pro Woche.

Betreuung Von der Seite der HSR wird die Arbeit von Herrn Prof. Dr. Markus Stolze betreut.

1.4.2 Infrastruktur

Tabelle 1.2: benutzte Hardware

Gerät	Beschreibung
1x T410	Privater Laptop von Patrice Müller
1x T60	Privater Laptop von Dominik Spengler
1x Server	Ubuntu 64bit Kernel-img 2.6.32-28-server
1x HTC Desire	Android 2.2 , Zusammen mit der Gruppe von EcoHelper

Tabelle 1.3: Entwicklungsumgebung

Komponente	Beschreibung	Version
IDE	Eclipse 3.6 (Helios)	3.6 Helios
SDK	Java Platform (64bit)	1.6.0_24
SDK	Android-SDK	2.2 API 8, Revision 2
Eclipse-Plugin	Android Development Toolkit	10
Tool	Git	1.7.1
Tool	Enterpreise Archtect	7.1
Dokumentation	Wiki-Redmine	0.9.3
Dokumentation	Redmine DocPu plugin	0.0.2
Dokumentation	TexMaker	2.2.2

Tabelle 1.4: Organisation

Tool	Beschreibung
Redmine	Ein Web-Basiertes Projektmanagement-Tool
Redmine Gitrevision Download plugin	Redmine-Plugin zum verwalten des Git-Repos

1.4.3 Abgrenzungen

Die nachfolgende Bereiche bzw. Funktionalitäten werden von unseren Arbeit nicht abgedeckt oder Stark eingeschränkt. Die Begründung werden unter dem jeweiligen Abschnitt erörtert.

Ausrichtung: Das Produkt das am Ende dieser Arbeit hervorkommt soll als Prototyp verstanden werden, welcher nur die Grundfunktionalität, die vom Projektteam durch Interviews mit Farbenblinden eruiert wurden, implementiert.

Internationalisierung: Auf die Internationalisierung wird bewusst verzichtet. Da sich das Team entschieden hat sich auf die Kernfunktionalitäten zu konzentrieren. Um es später einfach implementieren zu können wir von Anfang an so Designed das eine spätere Internationalisierung kein Problem darstellen sollte.

Device Kompatibilität: Die Applikation wird nur für das HTC-Desire konfiguriert und getestet, da es sich dabei um das offizielle Dev-Device handelt.

1.5 Vorgehen Aufbau der Arbeit

Als Vorgehensmodell der Semesterarbeit wurde eine Variation von Scrum[1] gewählt. Diese Wahl wurde wegen der ausgesprochenen Agilität und der Fokusierung auf funktionierende Programme getroffen. Nach dem in der ersten Woche das Thema der Arbeit ausgewählt war, wurde die Arbeit dementsprechend in Sprints.

Die ersten zwei Wochen wurden genutzt um sowohl die Infrastruktur aufzusetzen als auch um die ersten Anforderungen an die Software zu erfassen. In der Konkurrenzanalyse wurde die Geschwindigkeit als eine absolute Muss-Anforderung identifiziert. Aufgrund der Erkenntnisse des ersten Sprints wurden in einem zweiten Schritt verschiedene Prototypen auf Geschwindigkeit getestet. Als Resultat dieses Sprints konnte das Risiko der Performance minimiert werden. Um die Grundfunktionalitäten der Applikation implementieren zu können, musste als nächstes verschieden Algorithmen evaluiert, und die Architektur der C-Bibliothek erstellt werden.

Damit in einem nächsten Schritt mit dem Usability Testing begonnen werden konnte, musste die Applikation in den Grundzügen den UI Mocks entsprechen und die Grundfunktionalitäten vorhanden sein. Deshalb wurde zur in der Mitte des Projektes mit der Entwicklung des User Interfaces begonnen. Da wir die Komplexität des User Interfaces komplett unterschätzt haben, kam das Projekt etwas ins Stocken. Da es sich beim User Interface aber um den wichtigsten Aspekt einer Smartphone Applikation handelt, wurde sehr viel Zeit in die Entwicklung investiert. Dies machte sich vor allem dadurch bemerkbar, dass viele Pendenzen wie die Reviews,

1 www.scrumalliance.org

Dokumentation, Usability Testing nach hinten verschoben wurden und erst gegen Ende des Projektes aufgeholt werden konnten.

2 Stand der Technik

Tabelle 2.1: Dokumenthistory - Stand der Technik

Rev.	Datum	Wer	Änderung
0.1	05.03.2011	Patrice Müller	Konkurrenzprodukte evaluiert und bewertet
0.2	16.03.2011	Dominik Spengler	Bestehende Arbeiten evaluiert

2.1 Bestehende Arbeiten

Leider lässt sich kaum auf bestehende Arbeiten in der Videobearbeitung für Farbenblinde auf mobilen Geräten zurückgreifen. Es existieren allerdings einige interessante Studien welche sich mit der Bildbearbeitung mit dem Ziel Farbenblinde in der Farbunterscheidungsproblematik zu unterstützen. Eine hervorragende Einführung und Vergleichsstudie ist der IEEE Artikel “Color Transformation Methods for Dichromats[1]". Im folgenden wird kurz auf die unterschiedlichen Techniken eingegangen.

2.1.1 Grundlegende Idee

Obwohl verschiedene Methoden zur Erleichterung der Farbunterscheidung für Farbenblinde existieren, ist die grundlegende Idee hinter den gängigsten Algorithmen dieselbe:

Simulation der Farbsehschwäche: Um die Problematischen Bereiche eingrenzen zu können wird in einem ersten Schritt die Farbenblindheit simuliert.

Korrektur der Unterschiede: Die mittels der Simulation der Farbsehschwäche identifizierten Unterschiede werden angepasst.

Die Simulation der Farbsehschwäche geschieht üblicherweise durch die Transformation des RGB-Farbraums in den LMS-Farbraum. Die Repräsentation der Farben im LMS-Farbraum entspricht dabei der Art und Weise wie Farben durch die Zapfen im menschlichen Auge wahrgenommen werden. Daher eignet sich dieser Farbraum ausgezeichnet zur Simulation von fehlerhaften Zapfen.

1 http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5514503

In einem zweiten Schritt wird dann der identifizierte Problembereich bearbeitet. Dabei lässt sich grob zwischen zwei unterschiedliche Vorgehensweisen unterscheiden:

- **Farbersetzung**
- **Farbtransformation**

2.1.2 Farbersetzung

In dieser Methodik werden problematische Farben durch andere Farben ersetzt. Der Vorteil dieses Lösungsansatzes ist die Einfachheit der Implementation. Jedoch führt die Einführung von falschen Farben zu einer verzerrten Wahrnehmung der Realität und somit zu Fehlinterpretationen der Farben.

2.1.3 Farbtransformation

Bei der Farbtransformation wird versucht den Kontrast im problematischen Bereich soweit anzupassen, dass es für Farbenblinde möglich wird die Farben zu unterscheiden. Hauptvorteil dieser Methode ist, dass das Transformierte Bild stärker der Realität entspricht und daher nicht so stark befremdend wirkt wie Bilder bei welchen Farben ersetzt wurden. Generell wurde dieser Ansatz von Farbenblinden als angenehmer empfunden. Der bekanntere Daltonize[1] Algorithmus beispielsweise arbeitet mit einer (eher starken) Farbtransformation.

2.1.4 Alternative zu Farbtransformation

Eine Weiter Möglichkeit wäre noch, die Farben in unterschiedlichen Schraffierungen zu unterteilen. D.h. das jede Farbe ihre eigne Schraffierung bekommt. Aber auch hier hat man das Problem, wenn die Farbnuancen nahe beieinander liegen das der Unterschied der Schraffierung zu gering ist. Wie schon im Paper[2] erwähnt eignen sich die verwendeten Algorithmen ehere für diskrete Farbräume und nicht für fliesende Farbverläufe wie in unserem Fall.

1 http://www.vischeck.com/daltonize/

2 http://152.96.56.18/redmine/attachments/download/27/Paper_Mensch_Computer.pdf

2.2 Konkurrenzprodukte

Im Rahmen unserer Recherche haben wir eine App entdeckt die, die gleich Problemdomäne behandelt wie die unsere. Die App heisst “DanKam A2.0+ünd wurde von einem Amerikaner geschrieben welcher einen Farbenblinden Freund hat. Die App wurde ursprünglich für das iPhone geschrieben und dann in einem zweiten Schritt auf das Android portiert. Was zu gewissen Unschönheiten geführt hat, genaueres im Usability teil.

Unsere Analyse haben wir in eine Technische-Analyse und in eine Usability-Analyse unterteilt.

2.2.1 Technische Analyse

Hier kurz die Technischendaten der Konkurenz-App

Tabelle 2.2: Technische Daten – DanKam

Tech-Daten der “DanKam A2.0+A	
OS Anforderung	
Android	>= 2.1
Hardware Anforderung	
Eine Photokamera	

2.2.2 Technische Daten unseres Test-Devices

Tabelle 2.3: Technische Daten - Samsung Galasy S

Samsung I9000 Galaxy S	
Hardware / OS	
Marktstart	2.Quartal 2010
Android	2.2 mit Touchwiz UI
RAM	256 MB
SD	8GB (Intern)
CPU	ARM Cortex A8 1GHz
GPU	PowerVR SGX540
Akku	
Akku	Li-Ionen 1500 mAh
Standby	31,3 Tage
Sprechzeit	13,0 Stunden
Display	
Auflösung	480×800 pixels–(WVGA)
Grösse	4.0Zoll
Type	Capacitive, SUPER-AMOLED, Touch-sensitive display
Kamera	
Video	HD 720p @ 30fps
Blitz	-
Auto-Fokus	Ja
Megapixles	5MP

Nun man könnte nun auf die Idee kommen das es ja nicht gerade viel ist was die App braucht, so war auch unser erster Gedanke. Nun mussten wir leider feststellen das die App “DanKam A2.0+ëxtrem viel Power braucht. Das schlug sich nicht nur im Akku-Verbrauch nieder sondern auch im Verhalten der des UI’s. D.h. konkret das Bild das die Kamera uns lieferte war nicht mehr “Realtime". Das soll heissen wenn man das Smartphone z.B. von Rechts nach Links bewegt hat und dabei ca. einen Meter Distanz hinter sich gebracht hat, reagierte das Kamerabild erst nach gut drei Sekunden.

Zur Verteidigung der Applikation, bei diesem Test haben wir den Daltonize Algorithmus verwendet welcher durchaus aufwendig ist aber gemäss unseres Probanden die besten Ergebnis liefert.

Wir haben bei unserm Interview mit unserm ersten Probanden ca. 45min mit der App gearbeitet und mussten mit Erschrecken feststellen das der Akku von 45% auf 15% runter gefallen ist. Neben dem wurde das Device extrem warm.

2.2.3 Usability-Test

Der erste Eindruck der App ist, das sie langsam ist. Und zwar so langsam das sie bis zu unbrauchbar von unserem Probanden bezeichnet wurde. Obwohl auf der Homepage von [1] davon gesprochen wird das es unter dem Galaxy S nun auch laufen sollte. Ok wir müssen einräumen sie läuft zwar schon aber nicht gerade so das unser Probanden daran freunde hatten.

Was man dem Entwickler von “DanKam A2.0+ßu Gute kommen lassen muss ist das er die Diversen Modi sprich unterschiedliche Filter Optionen hat. Was aber leider unsern Probanden überfordert hatte und somit für ihn total unbrauchbar war.

Was uns gleich zu der Menüführung bring. Hier merkt man das die App vom iPhone portiert wurde, denn die Guidelines von Android wurden in keinster Weise berücksichtigt. Was einem das Gefühl gibt mit einem Fremdkörper im System zu interagieren.

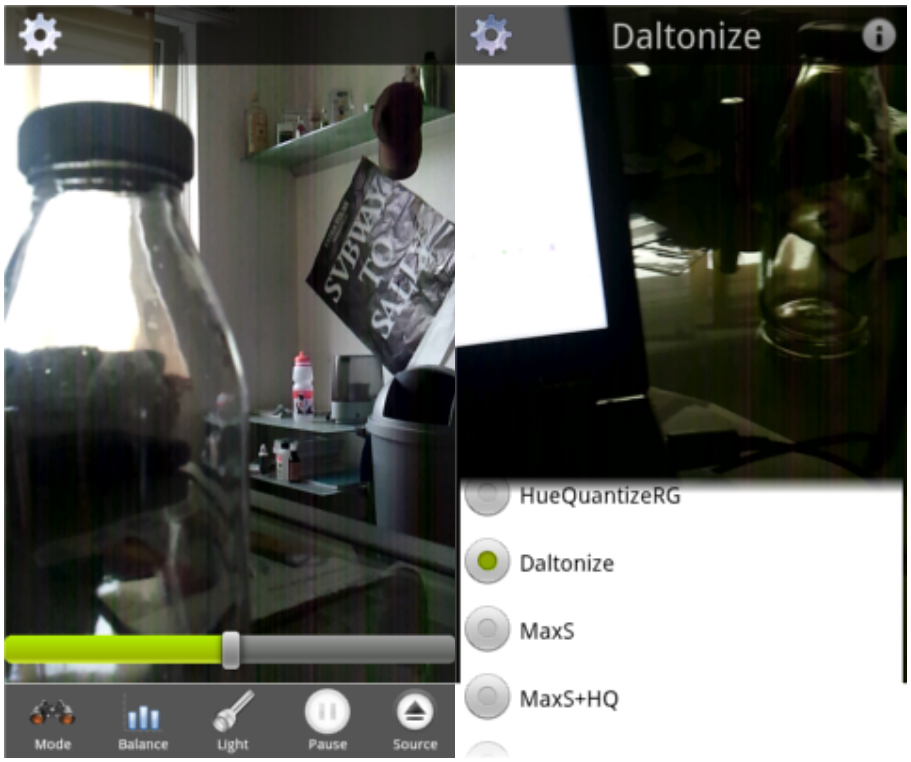
Es ist aber nicht alles schlecht. Was absolut genial war, war der Stop-Button. Dieser Button veranlasst die App die Live-Übersetzung zu unterbrechen und das Aktuelle Bild in Ruhe zu betrachten.

2.2.4 Fazit

Alles im Allen ist die Idee und die Umsetzung der App nicht schlecht. Wir wagen es zu bezweifeln ob es wirklich so viele Einstellungsoptionen braucht. Des Weitern ist die Performance der App wirklich miserabel. Vom Akku-Verbrauch mal ganz zu schweigen. Wie das ganze auf dem iPhone aussieht war nicht Bestandteil unseres Testes.

Am schluss wollen wir noch das eine oder andere Bild der Applikation zeigen damit jeder auch noch die Chance hat sein eigens Bild des UI zu machen.

1 http://www.appbrain.com/app/dankam-a2-0/com.dankaminsky.dankam.dankam_a2



(a) DanKam Applikation **(b)** DanKam Settings

Abbildung 2.1: DanKam Applikation Screenshots

3 Evaluation

Tabelle 3.1: Dokumenthistory - Evaluation

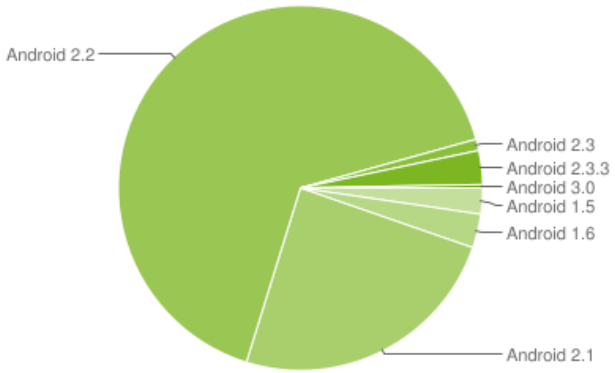
Rev.	Datum	Wer	Änderung
0.1	26.05.2011	Patrice Müller	Technologieentscheid erstellt

3.1 Technologieentscheid

Die Entscheidung welche Technologie wir verwenden werden war einfach. Da schon in der Aufgabenstellung definiert wurde das es eine Android-App gibt, stellte sich uns nur noch die Frage welche Version des Android unser Zielpaltfrom sein wird.

Zur Verfügung standen uns die Versionen Froyo (2.2) und Gingerbread (2.3) nun galt es noch zu klären welche die am weitest verbreitet Version ist. Dank Google lässt sich diese Frage schnell und einfach beantworten, Froyo (2.2).

Abbildung 3.1: Android Versionsverteilung



Platform	API	Vert.
Android 1.5	3	2.3%
Android 1.6	4	3.0%
Android 2.1	7	24.5%
Android 2.2	8	65.9%
Android 2.3	9	1.0%
Android 2.3.3	10	3.0%
Android 3.0	11	0.3%

4 Umsetzungskonzept

Tabelle 4.1: Dokumenthistory - Umsetzungskonzept

Rev.	Datum	Wer	Änderung
0.1	30.05.2011	Dominik Spengler	Umsetzungskonzepte erstellt

4.1 Allgemein

Ziel der Arbeit war es, eine lauffähige und stabile Applikation für Android zu erstellen, welche Farbenblinde Personen in Alltagssituation unterstützen kann. Als primäre Informationsquelle diente die offizielle Android Dokumentation auf http://developer.android.com. Eine besonders wichtige Anforderung an die Applikation war die Geschwindigkeit. Um die Anforderung von mindestens 15 Bildern pro Sekunde auch auf älteren Geräten erfüllen zu können wurde die Transformationslogik der Kamerabilder in eine C-Bibliothek ausgelagert.

4.2 Bildtransformation

Ziel der Bildtransformation ist es, dass Farbenblinde sonst für sie nicht sichtbare Unterschiede erkennbar zu machen. Um dies zu erreichen wurden folgende Algorithmen implementiert:

- **Daltonize**[1]**:** Algorithmus welcher die von Farbenblinden Unterschiedlich gesehenen Farben sinnvoll anpasst.
- **Falsche Farben:** Eigener Algorithmus, welcher im kompletten Bild die Farben vertauscht.
- **Farbunterschiede hervorheben:** Eigener Algorithmus, welcher die Grün Werte verstärkt und die Rot Werte abschwächt.

1 http://www.vischeck.com/daltonize

4.3 Farberkennung

Die Farberkennung wurde durch eine Umwandlung in den HSL Farbraum realisiert. Dieser Farbraum eignet sich besonders gut für die Farberkennung, da er die Farbe von Sättigung und Helligkeit trennt. Für eine exakte Erklärung der Implementation wird auf Implementationskonzepte verwiesen.

4.4 Erreichte Ziele

Da ein Ziel unserer Applikation war mit möglichst wenig Benutzerinteraktion auszukommen, werden die erreichten Ziele nicht mittels erfüllen der Use Cases definiert, sondern über die Erfüllung der Szenarien. Leider müssen wir offen zugestehen, dass wir mit lediglich zwei Probanden eine zu geringe Anzahl an Testpersonen hatten um eine qualitative Aussage über die erreichten Ziele zu machen.

5 Resultate

Tabelle 5.1: Dokumenthistory - Resultate

Rev.	Datum	Wer	Änderung
0.1	26.05.2011	Dominik Spengler	Erreichte Ziele, Ausblick erstellt
0.2	30.05.2011	Dominik Spengler	Persönlicher Bericht, Lessons Learned erstellt
0.3	31.05.2011	Patrice Müller	Persönlicher Bericht erstellt

5.1 Erreichte Ziele

Da ein Ziel unserer Applikation war mit möglichst wenig Benutzerinteraktion auszukommen, werden die erreichten Ziele nicht mittels erfüllen der Use Cases definiert, sondern über die Erfüllung der Szenarien. Leider müssen wir offen zugestehen, dass wir mit lediglich zwei Probanden eine zu geringe Anzahl an Testpersonen hatten um eine qualitative Aussage über die erreichten Ziele zu machen.

5.1.1 Szenarien

Tabelle 5.2: Resultate – Szenarien

Szenario	Erfüllungsgrad
Ishihara-Test	erfüllt
Farbstifte auswählen	erfüllt
Öffentliche Toilette	teilweise erfüllt
Glastrennung	teilweise erfüllt

Trotz der geringen Anzahl an Probanden, können wir mit einiger Zuversicht sagen, dass der Ishihara-Test kein Problem mit Hilfe unserer Applikation darstellt. Bei den Szenarien bezüglich der Farberkennung spielen die Lichtverhältnisse eine zentrale Rolle. Bei guten Lichtverhältnissen (Farbstifte auswählen) funktionierte die Farberkennung erwartungsgemäss. Bei schummrigen Licht und sehr dunklen oder hellen Gegenständen wird jedoch häufig Schwarz, Grau oder Weiss erkannt.

Leider führte das einschalten des LED oft zu einer Überbelichtung. Obwohl es in HTC Geräten die Möglichkeit gibt die Stärke des LED einzustellen, handelt es sich dabei nicht um ein offizielles Android Feature und konnte deshalb im Umfang dieser Semesterarbeit nicht behoben werden.

5.1.2 Nichtfunktionale Anforderungen

Die Nichtfunktionalen Anforderungen können grösstenteils mit den Statistiken des Android Market verfolgt werden. Zwar kann man nicht mit Gewissheit sagen, dass die Anforderungen komplett erfüllt werden, aber die Statistiken geben ein Indiz für die Erfüllung der Anforderungen. Die hier abgebildeten Statistiken zeigen den stand der Applikation nach zwei Wochen im Market.

Tabelle 5.3: Resultate – Nichfunktionale Anforderungen

Anforderung	Erfüllungsgrad
Performance	erfüllt
Hardwarekompatibilität	erfüllt
Softwarekompatibilität	erfüllt
Stabilität	erfüllt (mit Vorbehalt)
Bedienbarkeit	erfüllt

Performance

Die Anforderung an die Performance wurde mit dem HTC Hero getestet. Selbst beim berechnungsintensiven Daltonize Algorithmus ist die Framerate genügend hoch, um noch benutzbar zu sein. Uns wurde berichtet, dass es auch auf dem HTC G1 brauchbar läuft.

Hardwarekompatibilität

Während der Entwicklung wurde die Applikation immer wieder auf folgenden Geräten getestet:

- HTC Hero
- HTC Desire
- Samsung Galaxy S
- Samsung Galaxy Tab

Die Gerätestatistik des Android Market zeigt, dass unsere Applikation bereits auf verschiedensten Geräten installiert wurde. Da noch keine negativen Kommentare aufgetaucht sind, können wir von der Erfüllung der Hardwarekompatibilität ausgehen.

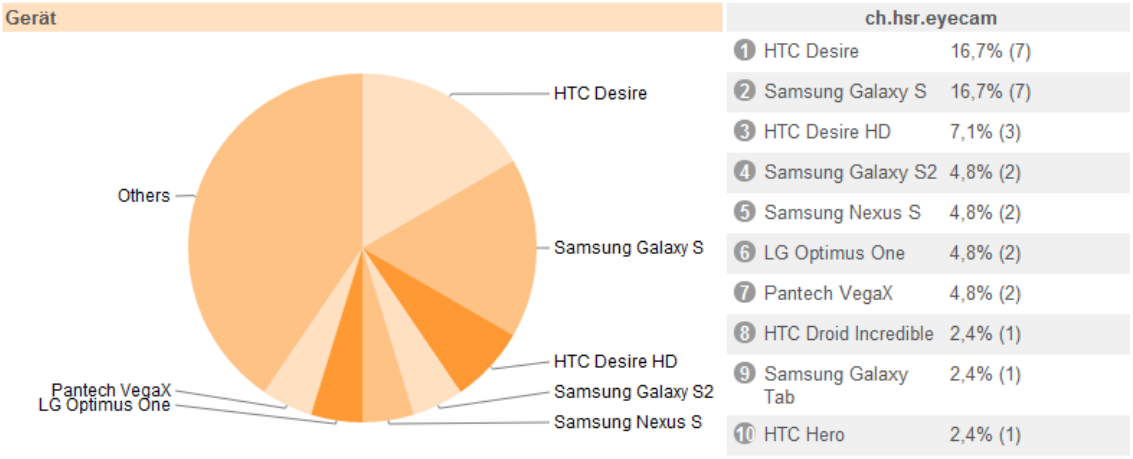


Abbildung 5.1: eyeCam Geräteverteilung

Softwarekompatibilität

Analog zu der Hardwarekompatibilität können wir von der Erfüllung der Softwarekompatibilität ausgehen.

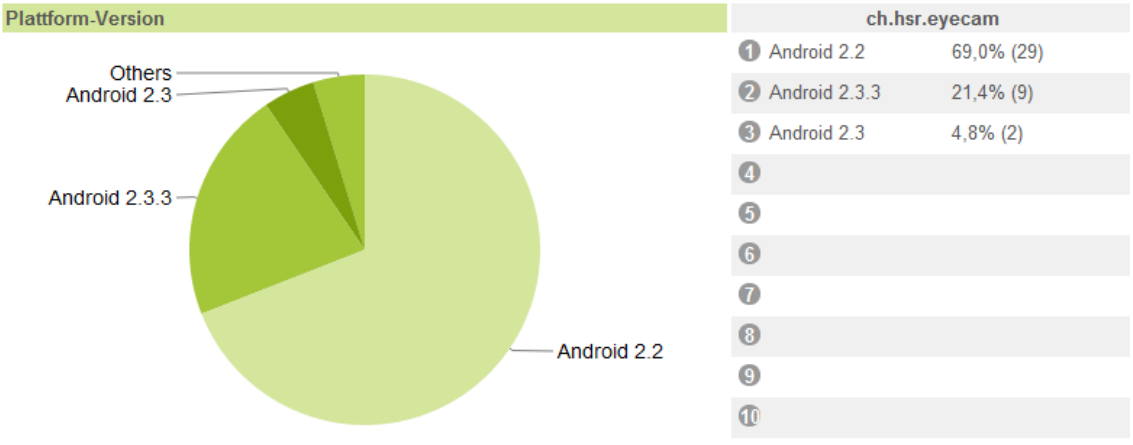


Abbildung 5.2: eyeCam Platform Versionen

Stabilität

Es wurde eine NullPointerException gemeldet. Da wir die Grenze von 100 Installationen noch nicht erreicht haben, können wir die Anforderung von 1% Fehlverhalten nur mit Vorbehalt als erfüllt ansehen. Allerdings konnten wir den Fehler nicht rekonstruieren, auch nicht mit dem “Affentest".

Abbildung 5.3: eyeCam Installationen

Bedienbarkeit

Für die Bedienbarkeit beziehen wir uns auf unsere eigenen Erfahrungen mit der Applikation und auf die Usability Tests. Es wurden keine Probleme mit der Einhändigen Bedienbarkeit festgestellt.

5.2 Ausblick

Für die Weiterentwicklung wurden folgenden Punkte als die wichtigsten identifiziert:

- verbesserung Farberkennung
- erweitertes Usability Testing

Ein detaillierterer Ausblick befindet sich in einem separatem Kapitel Möglichkeiten der Weiterentwicklung.

5.3 Persönliche Berichte

5.3.1 Dominik Spengler

Projektverlauf

Zu Beginn der Semesterarbeit war uns noch nicht klar was das Thema unserer Projektarbeit sein wird. Anfangs stand ein Finger Duell Spiel auf dem MS Surface Table und eine Comparis Applikation für Google TV zur Auswahl. Herr Stolze überraschte uns immer wieder mit neuen spannenden Ideen für unsere Semesterarbeit. Eine dieser Ideen war auch eine Applikation für Android als Hilfsmittel für Farbenblinde. Weil ich schon mein SE2 Projekt auf Android gemacht habe, und bisher noch keinerlei Erfahrung mit Microsoft Technologien und HTML5 gemacht habe, war für mich persönlich die Android Applikation am Interessantesten. Glücklicherweise war Patrice Mueller ähnlicher Meinung und wir konnten uns auf die Android Applikation einigen. Ein sehr interessanter Aspekt der eyeCam Applikation war sich mit der Problematik der Farbenblindheit auseinander zu setzen.

Es war sehr Interessant, die Anforderungsanalyse direkt mit dem Endbenutzer zu machen. Da ich ursprünglich keinen Informatik-Beruf gelernt habe, war dies eine wertvolle Erfahrung für mich. Oftmals wurde in den Interviews klar dass ich mir zu viele Gedanken über Funktionalitäten machte, welche gar nicht benötigt wurden. Ebenfalls konnten in den Interviews einige nahe liegende, überraschende Alltagsprobleme identifizieren werden, welche mir immer wieder von neuem zeigten wie wichtig der enge Kontakt zu dem Endnutzer ist.

Durch die Wahl von Scrum als Vorgehensmodell konnten wir uns schnell auf die Implementation der Prototoypen stürzen um das zentrale Risiko der Performance zu minimieren. Zu beginn wurde um jede Architekturentscheidung lange diskutiert, was zu einem eher langsamen Vorgehen führte, da wir eigentlich vermeiden wollten C-Code auf Android zu schreiben. Durch dieses Risiko-orientierte Vorgehen wurde uns aber früh in der Entwicklung klar dass es nötig ist JNI zu nutzen um die Performance Anforderungen zu erfüllen.

Die ersten drei Sprints liefen sehr unproblematisch ab, und wir konnten den Projektplan weitgehend einhalten. Allerdings haben wir die Komplexität des User Interfaces komplett unterschätzt. Weil wir die Orientationsänderungen des Gerätes selbst handhaben müssen, war es nötig eigene Views für alle benötigten GUI Elemente zu schreiben. Dies hat uns im zweiten Teil der Semesterarbeit etwas aus dem Konzept gebracht. Glücklicherweise konnten wir auch diese Hürde überwinden und eine fertige Applikation am Ende der Semesterarbeit abliefern. Dadurch ging die Dokumentation leider etwas unter, aber durch den Codefreeze zwei Wochen vor Abgabe konnten wir den Rückstand wieder gut machen.

Rückblick

Rückblickend muss ich sagen, dass der nahe Kontakt zu dem Endnutzer eines der Wichtigsten Punkte für gute Software ist. Leider kam bei uns das Usability Testing zu kurz. Es wäre sicherlich spannend ein erweitertes Usability Testing, möglicherweise mit A/B Testing, durchzuführen. Ebenfalls ist aufgefallen, dass wir gegen Mitte des Projektes eher Dokumentationsfaul wurden. Rückblickend muss man sagen, dass Scrum vielleicht nicht die beste Lösung für eine SA/BA. Es gibt viele Aktivitäten einer SA/BA, wie z.B. Dokumentation, Infrastruktur, welche sich nicht direkt einer User Story zuordnen lassen. Was unserem Projekt in der Schlussphase sehr viel gebracht hat, war der Code-freeze zwei Wochen vor Abgabetermin.

Gelerntes

Am meisten habe ich im Projektmanagement gelernt. Mit Scrum haben wir eine Vorgehensweise benutzt, welche für uns beide neu war. Durch den sehr geringen Overhead und die Forcierung der agilen Softwareentwicklung stellt Scrum für mich die beste Softwareentwicklungsmethode dar mit der ich bisher gearbeitet habe. Trotz des einfachen Grundkonzeptes von Scrum gibt es einige Fehler, welche sich bei uns eingeschlichen haben. Zum Beispiel haben wir unsere “Definition of Done"nicht konsequent durchgezogen und die User Stories waren in der Regel zu lang, was zu seltsamen Burndown Charts führte. Bezüglich Android Entwicklung, konnte ich nach dem SE2 Projekt einen tieferen Einblick in die Funktionsweise von Android gewinnen.

Ein weiterer Punkt der nicht vernachlässigt werden darf ist die Wichtigkeit des Usability Testing. In einem nächsten Projekt werde ich sicherlich mehr Zeit dafür investieren.

Fazit

Zusammenfassend kann ich sagen, dass ich äusserst stolz auf unsere Semesterarbeit bin. Die langen Architekturdiskussionen haben sich durchaus bezahlt gemacht. Obwohl mit JNI, eigenen GUI Elementen, Farberkennung einiges an Schwierigkeiten und neuen Technologien aufgetreten sind, konnten die Kernfunktionen der Applikation zur Zufriedenheit unserer Probanden implementiert werden. Das ist schlussendlich das Wichtigste in der Entwicklung einer Applikation.

5.3.2 Patrice Müller

Projektverlauf

Als wir hörten das wir eine Arbeit zur Unterstützung von Farbenblinden machen würden, dachten wir das wird eine schön einfache Sache und wir könnten uns auf die Architektur, Dokumentation und Projektautomation konzentrieren und die eine oder andere Fancy-Funktion implementieren.

Aber als erste mussten wir uns erst mal mit der Materie vertraut machen. Das hiess, sich mal um zuhören wer einen Farbenblinden kennt und wenn ja ob er bereit wäre uns zu helfen. Wie der Zufall so einem zu Spielt gab es sogar zwei Farbenblinden in meinem nahen bekannten bzw. Freundeskreis.

Als nächsten gingen wir an die Herausforderung, uns ein Mindset zu bilden das einem Farbenblinden möglichst nahe kommt. Dazu führten wir lange Interview mit unser Probanden, auf Grund diesen Interviews spezifizierten wird dann unser Features. Die wohl grösste Überraschung war für mich, als der ein Proband uns schilderte das immer wieder auf das Problem stiess auf öffentlichen Toiletten nicht zu wissen ob die Kabine nun besetzt ist oder nicht. Ihm blieb bis anhin nichts weiter über als die Toilettentür auf gut Glück zu öffnen.

Im Rahmen der zu vor genannten Interviews machten wir auch eine Konkurrenzanalyse der „DanKam 2+“ dies ist eine Android App die ihren Ursprung beim iPhone hat. Somit wurde die App einfach vom iPhone aufs Android portiert. Die Portierung gelang mehr schlecht als recht – So was von langsam -. Dennoch lernten wir viel. Denn die Essens dieser Konkurrenzanalyse war das unsere Probanden eine schnell und einfache Applikation wollten.

Einfach hiess in diesem Fall, wenig Knöpfe und wenig Möglichkeiten. Was uns als Informatiker durchaus überraschte, wer will schon wenig Möglichkeiten? Auf jeden Fallen machten wir uns das KISS-Prinzip zum Kredo.

Am Anfang der Implementierung lief alles hervorragend, sogar die JNI-Schnittstelle wurde schnelle implementiert als gedacht. Dann kam das „simple“ GUI. Das GUI an sich ist ja wirklich simpel gehalten. Aber die Rotation die es mit machen muss stellte uns vor eine Herausforderung, an der wir manchmal schier verzweifelten. Das Problem war das normaler weise die Acitivity neu gestartet wird so bald eine Orientierungswechsel kommt. Um die Activity dem entsprechend anzupassen. Dies war wegen der Kamera nicht möglich. Also mussten wir alles selber implementieren.

Als dann alles so ausgesehen hat wie wir das wollten und wir auch bei den letzten Usability-Tests noch die eine oder andere Finesse entdeckten die wir mit wenigen Zeilen implementieren konnten und der Features bzw. Codefreeez hinter uns lag. War es zeit die Applikation auf dem Android-Market zu publizieren.

Nach anfänglichen Schwierigkeiten mit dem eröffnen eines Developeraccount und dem Signieren der Applikation, öffnete sich unser eine neue Welt von Möglichkeiten. Die wir aber leider im Rahmen dieser Arbeit nicht mal ankratzen konnten.Nun ja, es gibt ja noch eine Bachelorarbeit. Codefreez, Featurefreez und das Publizieren auf dem Markt lag hinter uns. Nun war es an der Zeit den Sourcecode der Welt vorzustellen. Wir entschieden uns unser Projekt auf github zu publizieren, da uns dort das Gesamtpaket am besten gefiel.

Die Letzten zwei Wochen standen vor der Tür und wir mussten noch so viel erledigen. Da war auf der eine Seite die allgemein beliebte Dokumentation und auf der Andren Seite noch das Video und das A0-Plakat.

Im Zuge der Plakat erstellen eröffnete sich uns ein weiteres Fenster für pushen unsere Applikation nämlich mein Optiker. Welcher uns u.a. Schon in früheren Phasen unseres Projektes ausgeholfen hatte, Thema Ishihara-Test.

Denn als er sich erkundigte wie es so in unserm Projekte liefe und ich ihm davon berichtet meinte ich würde uns gerne helfen und Plakat bei sich aufhängen. Nun waren vollends motiviert, ja sogar ein bisschen für die Dokumentation.

Rückblick

Rückblicken, war das Projekt ein voller Erfolg. Vor allem habe ich den Wert der Interviews mit den Endbenutzer kennen gelernt. Gut wenn wir das ganze ehrlich im Kontext eine SA beurteilen. Muss ich aber leider gestehen das die Dokumentation etwas unter ging. Der Grund dafür sehe ich einerseits bei der Art und Weise wie wir Scrum umgesetzt haben. Wir machten zwar die „Definitoin of Done“ und ja dort drin war die Dokumentation mit drin. Aber dies haben wir zu oft gänzlich miss achtet, es fehlte wohl der Scrum-Master der uns ab und zu mal auf die Finger klopfte. Auch die administrativen Tätigkeiten stellten uns zum Teil vor Schwierigkeiten im Bezug auf Scrum weil wir oft nicht wussten wo wir diese Tätigkeiten einordnen sollten. Codefreez war aber wohl das beste was wir machen konnten und diesen haben wir auch voll und ganze eingehalten.

Gelerntes

Scrum: Seine Stärken und Schwächen. Ich glaube Scrum ist was die Softwareentwicklung angeht der Weg den man gehen sollte. Voraussetzung ist einfach das die ganze Infrastruktur steht und das man die „Definition of Done“ einhält.

Android: JNI, UI-Entwicklung

Architektur: Harte Diskussion über die Architektur zahlen sich am Schluss mehrfach aus.

Fazit

Zum Schluss muss ich sagen das echt stolz auf unser Arbeit bin. Ins besonders auf unsere Architektur und unsern Code.

5.4 Lessons Learned

Aus den Persönlichen Erfahrungen sind folgende Lessons Learned hervorgegangen:

Android-Konzepte: Android bietet die Möglichkeit eigene GUI Elemente zu entwickeln, welche sich wie in Android üblich in einem XML Layout definieren lassen. So erreicht man eine sehr schöne Trennung von UI Code und Applikationslogik.

Android: Nach wie vor hinkt die Dokumentation von Android, besonders bei fortgeschrittenen Konzepten, hinterher und es ist zum Teil schwierig Lösungen für Probleme zu finden.

Farberkennung: Die Farberkennung ist eine komplexe Thematik. Wir haben für unsere Applikation einen sehr einfachen Ansatz gewählt, welcher bestimmt noch Verbesserungspotential bietet.

Farbtransformation: Farbransformationen sind schwierig nachzuvollziehen und zu testen.

Scrum: Scrum ist simpel und agil, jedoch nicht einfach. Besonders für eine SA/BA wo viel Wert auf Dokumentation gelegt wird vielleicht nicht die geeignetste Vorgehensweise.

Usability Testing: Das Usability Testing stellt einen wichtigen Bestandteil in der Applikationsentwicklung dar. Aufgrund des engen Zeitplanes ging es leider in unserem Projekt etwas unter.

Teil II

Software Projektdokumentation

6 Anforderungsspezifikation

Tabelle 6.1: Dokumenthistory - Anforderungsspezifikation

Rev.	Datum	Wer	Änderung
0.1	08.04.2011	Patrice Müller	Personas, Szenarios, Priorisierung der Szenarien,Nichtfunktionale Anforderungen, Use Cases erstellt

6.1 Personas

Die Personas wurde aus den Interviews mit unseren Probanden abgeleitet. Aus Datenschutzgründen wurden die Daten anonymisiert. Aus den beiden Interviews sind folgende Verhaltensvariablen hervorgegangen:

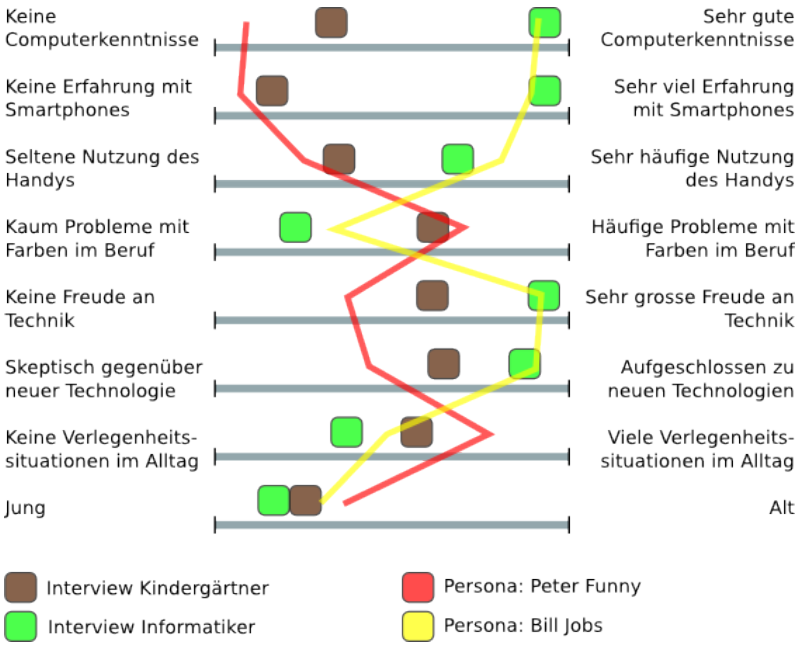


Abbildung 6.1: Behavior pattern Diagramm

Anhand der so identifizierten Verhaltensmustern haben wir unten beschriebenen Personas erstellt.

6.1.1 Peter Funny, Kindergärtner

Abbildung 6.2: Peter Funny

Tabelle 6.2: Charakteriesierung von Peter Funny

Alter	27 Jahre
Funktion	Kindergärtner
Verpflichtung	Verantwortlich für 28 Kinder, Freundin ohne Kinder
Werdegang	Sekundarschule, HMS+,PH

Peter arbeitet in einem durchschnittlichen Kindergarten in der Deutschschweiz. Er ist mit zwei anderen Mitarbeiter dafür verantwortlich das die Kinder eine altersgerechte Ausbildung bekommen.

Sein Arbeitsalltag ist alles andere als EDV lastig. Er verbringt viel Zeit mit den Kindern draussen, spielt und malt mit ihnen. Viele der Spiele und der Lieder die er mit den Kinder spielt bzw. singt kommen aus seiner eigenen Feder. D.h. er kreiert die Spiele bzw. Lieder selber.

Obschon, wie erwähnt, die EDV nicht den Alltag in seinem Kindergarten bestimmt ist sie allgegenwertig. Die Kinder haben Zugang zu zwei iMac's aus dem Jahre 2000. Klar sind das nicht die Power-Maschinen aber für die Lernspiele u.a. Globi reichen sie allemal.

Er selbst ist eine sehr gesellige und kommunikative Persönlichkeit mit einem grossen Gerechtigkeitssinn, welche nur schwer aus der Ruhe zu bringen ist. Leider kommt es hin und wieder vor, dass er Aufgrund seiner Farbsehschwäche in Verlegenheit vor seinen Schülern gerät.

Peter's Ziele

- **Verlegenheit vermeiden.** Da Peter durch seinen Beruf als Kindergärtner sehr häufig mit Farben zu tun hat, stellt seine Farbenblindheit in verschiedensten Situationen eine Schwierigkeit dar. Beispielsweise wenn er an der Wandtafel ein Bild vorzeichnen will. Sein Ziel ist es diese Schwierigkeiten zu minimieren und somit Unangenehme Situationen zu vermeiden.
- **Keine Einschränkung im Arbeitsalltag.** So sehr es ihm auch unangenehm ist, wenn die Farbenblindheit seine Arbeit beeinträchtigt, so hat er sich damit arrangiert. Obwohl er froh um ein Hilfsmittel wäre, soll es keine Einschränkungen mit sich bringen, wie zum Beispiel umständliche Bedienung, lange Startzeit etc.

6.1.2 Bill Jobs, Informatik Student

Abbildung 6.3: Bill Jobs

Tabelle 6.3: Charakterisierung von Bill Jobs

Alter	26 Jahre
Funktion	Informatik Student
Verpflichtung	Studium, Arbeit, Freundin ohne Kinder
Werdegang	Sekundarschule, Lehre als Applikationsentwickler, Informatik Studium an einer Hochschule

Bill Jobs, ist ein Informatik Student an der HSR und stürzt sich voller Elan auf jede neue Technologie die er in die Hände bekommen kann.

Er arbeitet zuverlässig auch wenn es für manchen einen chaotischen Eindruck macht. Mit einem ruhigen Plätzchen für seinen Laptop, welches auch Strom und Internetverbindung bietet ist er zufrieden.

Während der Vorlesung versucht er es zu vermeiden seinen Laptop mitzunehmen da er sonst zu stark abgelenkt ist, es sei denn er ist gerade an einem Projekt welches er nicht los lassen kann und daher in der Vorlesung weiter arbeitet.

Menschlich ist er, entgegen dem gängigen Informatikcliché, sehr gesellig und kommunikativ. Des Weiteren dient er seinen Kommilitonen oft als Ideenfabrik. Für ihn stellt die Farbenblindheit selten ein Problem in seinem Umfeld dar.

Bill's Ziele

- **Neues „Gadget" kennenlernen.** Bill interessiert sich sehr stark für jede neue Technologie. Besonders interessant ist für ihn neue Technologie, welche er gleich selbst ohne grossen Aufwand testen und nutzen kann, da er ohnehin schon alle erforderlichen Komponenten besitzt.
- **Nützliche Applikation auf seinem Smartphone.** Bill spielt häufig mit seinem Smartphone und ist immer auf der Suche nach neuen, mehr oder weniger sinnvollen Applikationen um sich die Zeit zu vertreiben oder sein Leben zu erleichtern.

6.2 Szenarien

6.2.1 Ishihara-Test

Ist-Szenario

In einem Gespräch mit einem Kollegen bei Peter zu Hause kommt seine Farbenblindheit ans Licht. Voller Neugier bittet der Kollege um eine Demonstration.

Da Peter einen Ishihara-Test zu hause hat, hohlt er diesen und die beiden machen den Test komplett durch. Wie erwartet erkennt Peter kaum eine Zahl oder einen Weg auf den Bildern. Sein Kollege jedoch besteht den Test ohne jegliche Probleme. Etwas frustriert, durch die klare Demonstration seiner Sehschwäche verabschiedet sich Peter von seinem Kollegen.

Soll-Szenario

Peter hat sich mit seinem ersten Lohn ein Andoird-Smartphone gekauft. Von einem Freund hat er von der “eyeCamApp gehört und das sie extra für Leute mit Rot-Grün-Schwäche entwickelt wurde.

Da Peter etwas skeptisch ist und er ja einen Ishihara-Test, durch seien Tätigkeit als Kindergärtner, zu hause hat. Macht er den Ishihara-Test einmal ohne die App und notiert sich die Ergebnisse. Nun macht er den gleichen Test noch mal, aber dies mal mit der App. Er notiert sich die Ergebnisse erst gar nicht da er so erstaunt ist das er auf einmal alle Zahlen erkennt.

6.2.2 Die öffentliche Toilette

Ist-Szenario

Es ist Samstag Abend und Bill Jobs sitzt mit Freunden in einer Bar und geniesst das Wochenende. Im Verlauf des Abends wird der Druck auf die Blase immer grösser und Bill macht sich auf den Weg zur Toilette.

Leider besitzt die Toilette keine Pissoirs. Da Peter doch ziemlich dringend auf die Toilette muss, will/kann er nicht darauf warten bis jemand anderes fertig wird. Da er bei den Toiletten nicht erkennen kann welcher Schieber auf Grün ist, versucht er die Türen zu öffnen und hat bei der dritten Türe endlich erfolg. Schnell huscht er in die Toilette aber nicht ohne den irritierten Blick der Person, welche gerade aus der Toilette nebenan kommt, zu bemerken.

Soll-Szenario

Nach einer längeren Sitzung und viel Kaffee, fordert die Natur auch bei Bill Jobs Ihren Tribut. Dummerweise hat es gerade vor 3 Minuten zur grossen Pause geläutet und wie üblich sind fast alle Toiletten besetzt.

Früher musste er alle Türen testen um eine freie Toilette zu finden. Heute kann er diese peinliche Situation etwas entschärfen indem er sein Smartphone raus holt und die performante eyeCam-App startet welche ihm durch seinen gespeicherten Lieblingsfilter bei der Findung einer freien Toilette hilft.

6.2.3 Farbstifte auswählen

Ist-Szenario

Eines der Kinder von Peter's Klasse bittet Peter darum einen schönen Sonnenuntergang mit blauem Himmel ganz oben an die Wandtafel zu malen um den Klassenraum etwas freundlicher zu gestalten.

Weil Peter Schwierigkeiten hat gewisse Farbstifte voneinander zu unterscheiden, und somit ganz genau weiss dass die Möglichkeit besteht, sich vor der Klasse zu blamieren, muss er leider nein sagen. Ganz enttäuscht von ihrem sonst so netten Klassenlehrer geht das kleine Mädchen an ihren Platz zurück.

Soll-Szenario

Peter will für den nächsten Tag ein Bild vorbereiten welches die Kinder dann nachmalen sollen. In diesem Bild, wird es eine grüne Wiese geben, auf welcher eine rote Blume wächst.

In einem Zweiten Schritt sollen dann die Kinder und somit auch er, eine blauen Himmel mit einer gelben Sonne mahlen. Wahlweise kann die ganze Situation auch in einen Sonnenuntergang getaucht werden. Dh. gelbe bzw. orange Sonne mit rötlichen bis orangen Hintergrund.

Um den minimalen Farbunterschieden Herr zu werden will Peter die "eyeCamApp benützen. Leider stellt er fest das die App mit den speziellen Lichtverhältnissen nicht ganz klar kommt. Da die eyeCam Applikation auch noch die Funktion der Farberkennung hat, probiert er diese Funktionalität gleich mal aus.

Er schaut das alle die Farbstifte die er verwenden will auf einem Bild platz haben und klickt auf Standbild. Nun Tipp er mit seinen Finger auf die Einzelnen Farbstifte und die eyeCam sagt ihm was für eine Farbe sie haben.

6.2.4 Glastrennung

Ist-Szenario

Bill Jobs muss mal wieder seinen Haushaltspflichten nachgehen und wurde von seinen Mitbewohnern gebeten die Glassentsorgung zu Übernehmen. Weil Bill die Flaschenentsorgung bisher noch nie übernommen hatte, willigte er ein.

Am Entsorgungscontainer angekommen, wirft er ganz selbstverständlich alle grünen und braunen Flaschen in die braun gekennzeichneten Öffnungen. Er ist etwas verwirrt darüber dass es so viele braune Öffnungen und verhältnismässig wenige weisse Öffnungen hat, aber denkt sich nichts weiter dabei.

Soll-Szenario

Es ist Samstag Mittag nach dem letzten Schultag und Bill Jobs hat den kürzeren gezogen und muss alle Bier und Wein Flaschen entsorgen.

Beim Entsorgungscontainer angekommen wird ihm bewusst, dass er doch weiss dass es grüne und braune Weinflaschen gibt. Ein kurzer Griff zum Smartphone mit unserer Applikation unterstützt ihn in der Erkennung der unterschiedlichen Farben der Flaschen. Dank dem einfachen User Interface der Applikation ist dies auch in seinem lädierten Zustand ohne Probleme möglich.

Voller Selbstzufriedenheit geht er nach dem Entsorgen nach Hause wo hoffentlich bereits Aufgeräumt wurde.

6.3 Priorisierung der Szenarien

Da es bei unsern Szenarios, im Prinzip immer auf das gleiche raus kommt. Nämlich auf die Rot-Grün-Erkennung bzw. ebben nicht Erkennung, haben wir uns entschieden das für uns der Isihara-Test als Referenz dient. Da er ebben auch universell einsetzbar ist. Sprich wir können den Test überall mitnehmen und so das Szenario in diversen Umfelder transportieren.

D.h. wenn man die Szenarios in **“Must"** und **“Optional"** unterteilt. Sieht es wie folgt aus:

- **Must**
  - Ishihara-Test
- **Optional**

- Farbstifte auswählen (Farberkennung auf dem Standbild)
- Öffentlich Toilette (Rot-Grün-Erkennung bei schummrigen Licht)

6.4 Use Cases

Unser Use Cases sind ganz simpel. Da unser Applikation auch den Anspruch für sich erhebt simpel zu sein.

6.4.1 UC1: Farberkennung

Beschreibt den Ablauf, bis zur Farberkennung

1. Der Benutzer startet eyeCam
2. Richtet die Kamera auf das Objekt von Interesse
3. Pausiert das Priviewing per Knopfdruck
4. Tippt auf das Objekt
5. eyeCam sagt im die Farbe

6.4.2 UC2: Die Previewing pausieren

Der Benutzer kann auf Previewing anhalten.

1. Direkt per Knopfdruck
2. Direkt auf das Objekt des Interesses tippen.

6.4.3 UC3: Filter einstellen

Der Benutzer tipp auf den Filter Kopf und kann den Filter wechseln.

6.4.4 UC4: Licht ein-/ausschalten

Der Benutzer kann auf das Lampen-Icon tippen und das Licht geht an bzw. aus.

6.4.5 UC4: Schriftgröße für PupUp's ändern

Der Benutzer Tippt auf Einstellung und das Menu geht auf. Hier wählt er seine Schriftgröße aus. Die Schriftgröße wird gleich angepasst.

6.4.6 UC5: Menugrösse anpassen

Der Benutzer Tippt auf Einstellung und das Menu geht auf. Hier wählt er seine Menugrösse aus. Die Menugrösse wird gleich angepasst.

6.4.7 UC6: Partial ein/aus

Der Benutzer Tippt auf Einstellung und das Menu geht auf. Hier wählt er dann ob der Filter partial wirken soll oder nicht.

6.5 Nicht funktionale Anforderungen

Die nicht funktional Anforderungen wurden in diesem Projekt besonders angeschaut. Gerade weil diese Applikation Leuten helfen soll ihren Alltag einfacher gestallten zu können. Natürlich werden wir auch in diesem Projekt nicht um die “standard"nicht funktionale Anfroderungen wie Performace, Stabilität und Bedinbarkeit rum kommen.

In den folgenden Unterkapitel, welches für sich jedes eine nicht funktionale Anforderung abbildet, werden wir jeweils genauer auf die jeweiligen Anforderungen zu sprechen kommen. Viele der kommenden Anforderungen sind aus den Feedback von unseren Probanden bezüglich der DanKam entstanden.

6.5.1 Performance

Die Applikation sollte **nie ins Ruckeln kommen**. D.h. es sollte wenn möglich immer über 20 Frame pro Sekunden angezeigt werden. Damit der Benutzer wirklich ein “Realtime-Erlbenis"hat.

6.5.2 Hardwarekompatibilität

Das soll heissen, die Applikation soll auf möglichst allen Android-Smartphone laufen ohne grösse Funktionseinbussen.

6.5.3 Softwarekompatiblität

Die Applikation soll so geschrieben werden das nach einem Upgrade auf ein neuere Version wenig bis keine Anpassungen an der Applikation gemacht werden müssen. Davon ausgenommen ist wenn die Architektur von Android grundlegend geändert wird.

6.5.4 Stabilität

Die Applikation sollte in **weniger als 1%** der Fälle ein Fehlverhalten aufweisen.

6.5.5 Bendinbarkeit

Die App soll für Rechts- so wie auch für Linkshänder im Hoch- wie auch Querformat mit einer Hand bedienbar sein. Des Weitern sollte **das UI Farbenblind gerecht designet werden**. Am wichtigen ist vor allem die **intuitive Bedienbarkeit**.

7 Analyse

Tabelle 7.1: Dokumenthistory - Analyse

Rev.	Datum	Wer	Änderung
0.1	28.03.2011	Dominik Spengler	Domain Model, Objektkatalog erstellt
0.2	24.05.2011	Dominik Spengler	Domain Model, Objektkatalog Anpassungen

7.1 Domain Model

Für die Domain Analyse haben wir uns an dem MVC-Pattern orientiert. Aufgrund des Android Frameworks wird eine Activity benötigt, welchen den Kern der Applikation darstellt. Von dieser Activity werden die benötigten Elemente (Controller und View) instanziert und verwaltet. Der Controller reagiert dabei auf die Benutzereingabe und reagiert darauf mit Aktionen auf der StartActivity. Die StartActivity wiederum instruiert die View wenn notwendig über die Änderungen.

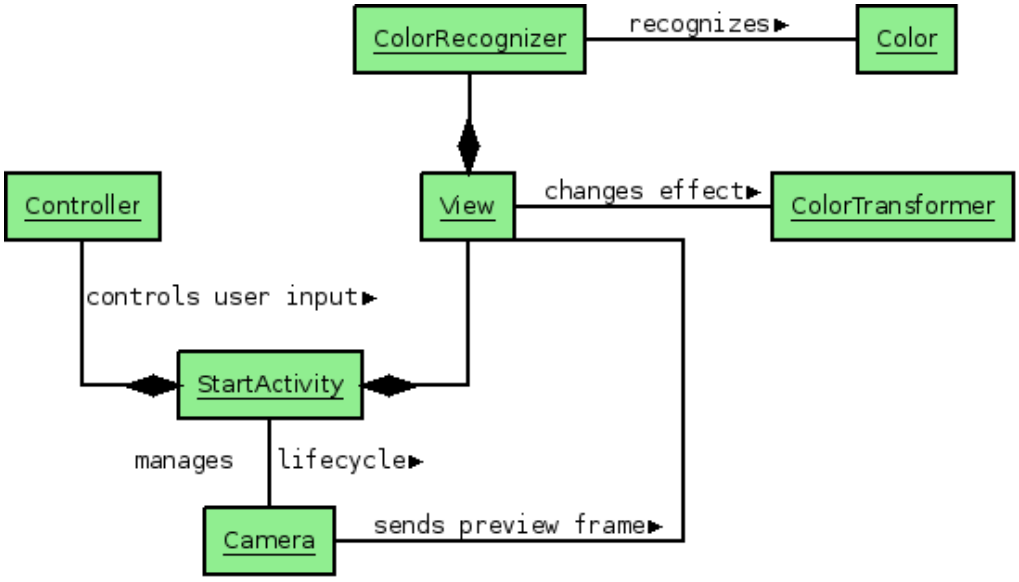


Abbildung 7.1: Domain Model

Die Kamera stellt dabei ein spezielles Objekt dar welches vom Android Framework zur Verfügung gestellt wird. Da es sich bei der Kamera um eine geteilte Ressource handelt, muss jedoch der Lebenszyklus ebenfalls durch die Activity verwaltet werden.

Der generelle Ablauf der Bildtransformation wurde wie folgt Designed:

- Kamera sendet Preview Frames über einen Data Callback an die View
- View sendet Preview Frames an das ColorModel zur Bearbeitung
- Das ColorModel nutzt die C-Bibliothek für die Bildtransformation
- Die View stellt das Transformierte Bild dar

Ein grosser Vorteil dieser Architektur ist die Tatsache, dass die StartActivity der Information Expert aller wesentlichen Aspekte der Applikation ist. Somit konnten die Eigenschaften der Applikation an zentraler Stelle, in der StartActivity, verwaltet werden.

7.2 Objektkatalog

7.2.1 Controller

Das Controller Objekt stellt die Schnittstelle für die Benutzerinteraktion dar. Es werden primär Änderungen in den Filtern und Einstellungen an das StartActivity Objekt weitergeleitet.

7.2.2 View

Das View Objekt hat folgende Verantwortlichkeiten:

- Darstellung des bearbeiteten Kamerapreview
- Farberkennung bei Berührung der Stelle im Bild

Generell ist es gemäss MVC-Pattern schlecht, bearbeitung der Benutzerinteraktion in der View zu haben. Jedoch macht es für die Farberkennung durchaus Sinn, da die View die Informationen über das dargestellte Bild hat.

7.2.3 ColorRecognizer

Dies repräsentiert ein Hilfsobjekt für die Funktionalität der Farberkennung. Dabei wird mit dem Color Objekt interagiert.

7.2.4 Color

Das Color Objekt repräsentiert die effektive Funktionalität der Farberkennung.

7.2.5 ColorTransformer

Dieses Objekt ist verantwortlich für die Bearbeitung der Preview Frames der Kamera. Es werden verschieden Filter bereitgestellt. Aufgrund von Performance Problemen mit der Implementation komplett in Java, musste auf eine native Bibliothek ausgewichen werden.

7.2.6 StartActivity

Dieses Objekt stellt den Kern der Applikation dar. Hier werden alle nötigen Initialisierungen durchgeführt und der Lebenszyklus der Applikation verwaltet. Für genauere Informationen zu den Activities wird auf die offizielle Dokumentation von Google verwiesen.

7.2.7 Camera

Das Camera Objekt ist lediglich für das bereitstellen der Preview Frames zuständig. Es handelt sich dabei um ein Objekt des Android Frameworks.

8 Design

Tabelle 8.1: Dokumenthistory - Design

Rev.	Datum	Wer	Änderung
0.1	10.03.2011	Dominik Spengler	Architekturdokument erstellt
0.2	26.05.2011	Dominik Spengler	Packagestruktur, Klassendiagramm erstellt
0.3	28.05.2011	Patrice Müller	UI-Design erstellt

8.1 Architektur

Die Grundarchitektur unserer Applikation ist denkbar einfach. Die einzelnen Komponenten der Applikation selbst lassen sich in Anlehnung an das 7.1 wie folgt unterteilen:

- **ColorView** stellt das View Objekt des Domain Models dar
- **ControlBar** hat die Funktionalität des Controllers
- **ColorRecocnizer, ColorTransform** sind für die Farberkennung und -umwandlung zuständig
- **EyeCamActivity** ist das Model der Applikation

Generell wird versucht soweit möglich Elemente des Android Framework selbst zu verwenden um die Hardwarekompatibilität möglichst gross zu halten. Weil die Applikation aber gezwungenermassen im Landscape Modus gestartet werden musste, damit die Kamera korrekt funktioniert, wurden noch diverse GUI Klassen benötigt, welche die Orientationsänderungen des Gerätes losgelöst vom Android Framework handhaben können. Alle diese Klassen sind in einem seperaten Package names **widgets** abgelegt. Für eine genauere Beschreibung der einzelnen Komponenten wird auf **??** verwiesen.

Da die Geschwindigkeit der Applikation ein Schlüsselelement in der Benutzbarkeit darstellt, wurden verschiedenste Architekturen mit steigender Komplexität evaluiert bis eine zufriedenstellende Geschwindigkeit erreicht wurde. Siehe Prototypen für genauere Informationen zu den getesteten Architekturen.

8.2 Packegestruktur

Die nachfolgenden Kapitel zeigen die Abhängigkeiten zwischen den einzelnen Packeten auf. Der Übersichtlichkeit halber werden lediglich die public Methoden und Members gezeigt. Für eine Darstellung der kompletten Klassen wird auf das Klassendiagramm verwiesen.

8.2.1 ch.hsr.eyecam

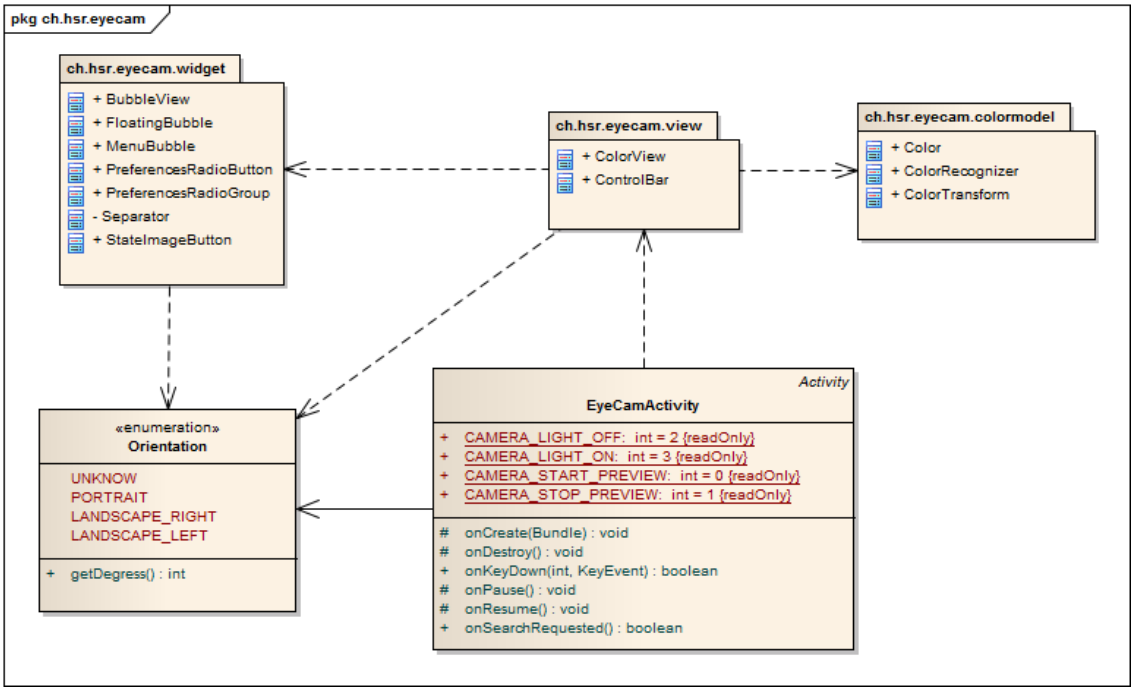


Abbildung 8.1: package ch.hsr.eyecam

Dieses Packet stellt das Model unserer Architektur dar. Auffallend ist dabei dass eine weitere Klasse namens Orientation vorhanden ist. Diese wird benötigt um die Gerätrotationen losgelösst vom Android Framework handhaben zu können.

8.2.2 ch.hsr.eyecam.view

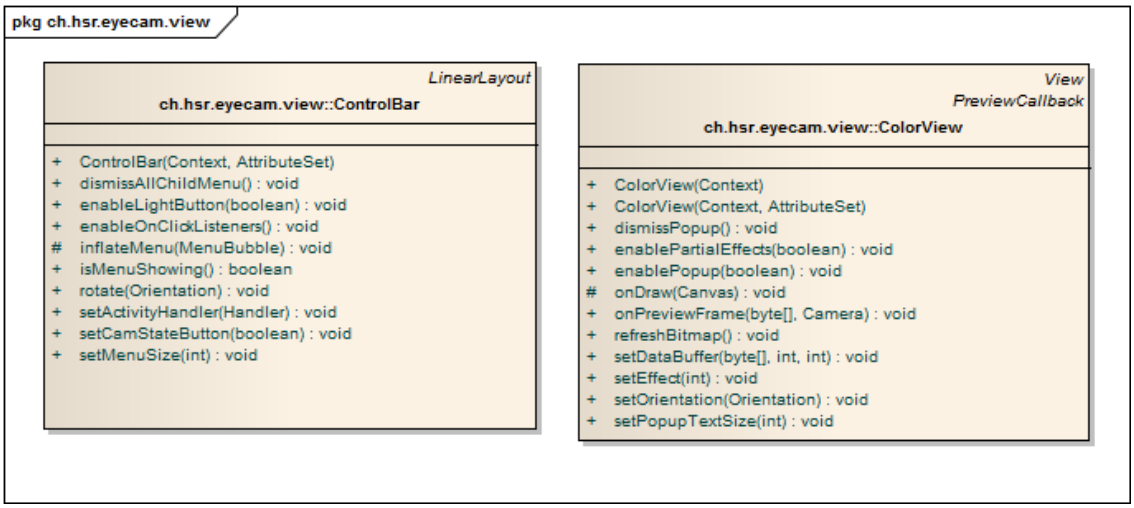


Abbildung 8.2: package ch.hsr.eyecam.view

In diesem Packet befindet sich sowohl die View als auch der Controller unserer Applikation.

8.2.3 ch.hsr.eyecam.colormodel

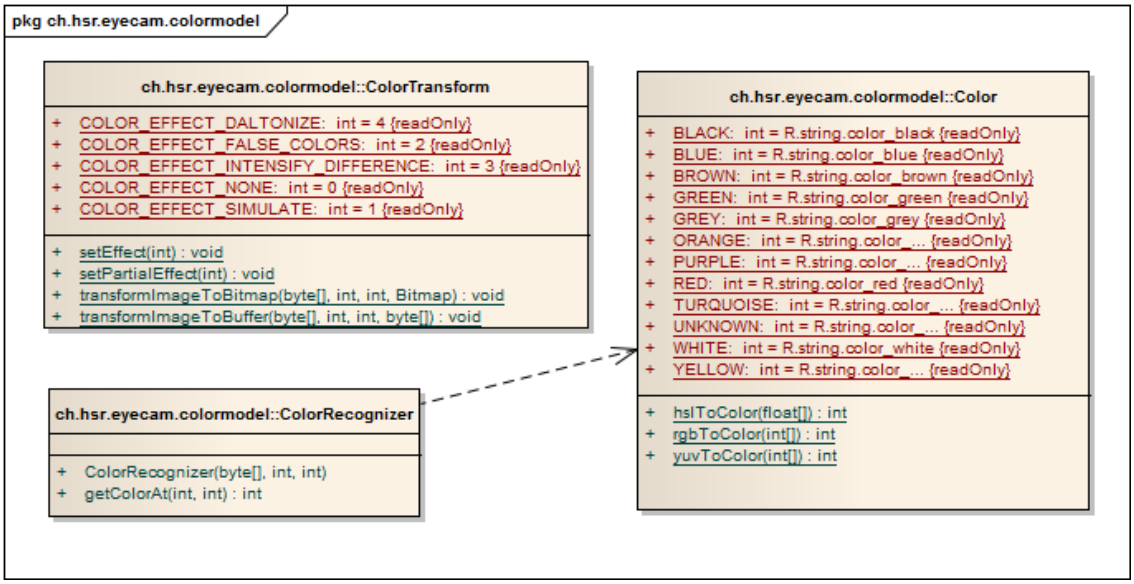


Abbildung 8.3: package ch.hsr.eyecam.colormodel

Das ch.hsr.eyecam.colormodel Packet ist für alle Funktionalitäten bezüglich Farberkennung und -umwandlung verantwortlich. Speziell zu erwähnen ist dabei, dass dieses Packet nur von der ColorView Klasse referenziert wird.

8.2.4 ch.hsr.eyecam.widget

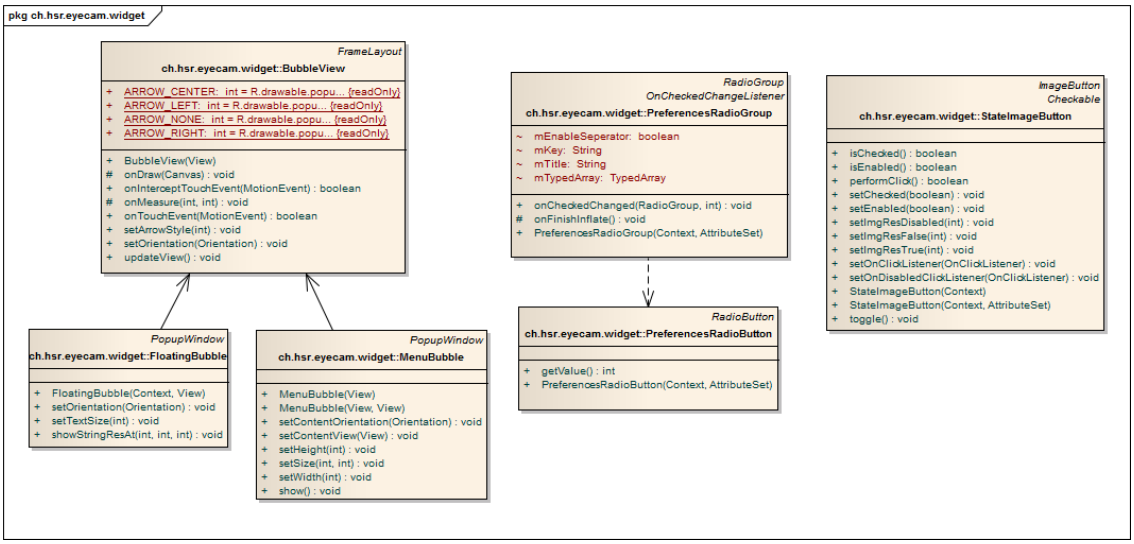


Abbildung 8.4: package ch.hsr.eyecam.widget

Da die Android Views nicht losgelöst vom Framework rotiert werden können, mussten Custom Views implementiert werden. Diese befinden sich alle in diesem Packet.

8.3 Klassendiagramm

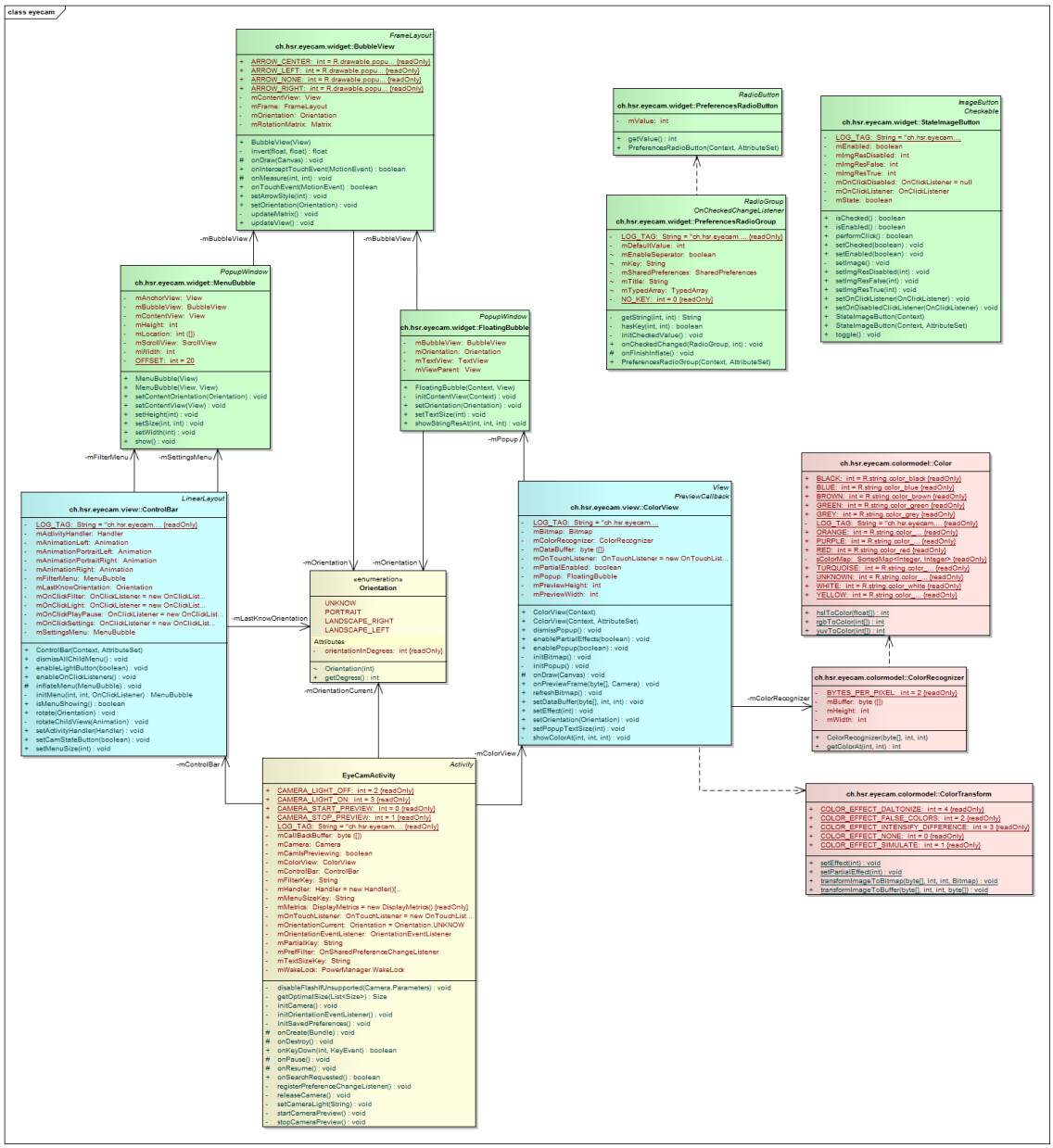


Abbildung 8.5: Klassendiagramm

In diesem Diagramm werden alle Klassen unserer Applikation und ihr Zusammenspiel abgebildet. Die Klassen sind dabei anhand ihrer Packetzugehörigkeit farblich unterschieden. Auffall-

end ist im Klassendiagramm die hohe Abhängigkeit von den verschiedenen Packeten an die Orientation Klasse. Dies ist jedoch nicht weiter verwunderlich, da alle Views Kenntniss über die Orientierung des Gerätes benötigen.

Ebenfalls auffallend ist, dass die drei Klassen *PreferencesRadioButton, PreferencesRadio-Group und StateImageButton* keinerlei Assoziation zu anderen Klassen aufweisen. Dies ist der Fall, weil diese Klassen über eine XML Datei vom Android Laufzeitsystem instanziert werden.

8.4 UI-Design

Wir haben beim UI-Desing uns auf folgende Punkte ganz speziell konzentriert.

- Farbenblindgrecht
- Intuitiv
- Clean and Simpel
- Fancy

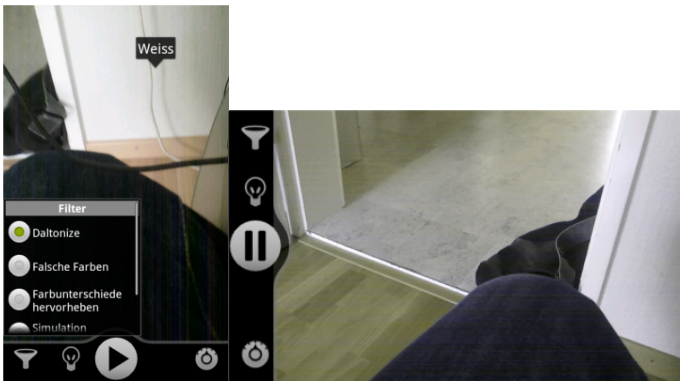
In den Folgenden Kapitel werden wir Ihnen zeigen wie wir diese Punkte in unser UI versucht haben best möglichst umzusetzen. Und wo wir in der Retrospektive vielleicht sagen müssen, da haben wir es nicht ganz geschafft.

8.4.1 Farbenblindgrecht

Da es ja neben der Rot-/Grün Schwäch auch noch die komplette Farbenblindheit gibt versuchten wir dies in unser UI-Design einfliessen zu lassen. Also machten wir uns auf um die Perfekt Farbpalette zu finden. Durch die Interview mit unsern Probanden hat sich recht schnell die Schwarz/Weiss/Grau Farbenpalette durchgesetzt, da hier sicher alle die gleichen Farben sehen.

Somit haben wir beim Design unseres UI speziell darauf geschaut alles mit der Schwarz/Weiss/Grau Farbenpalette zu gestallten. Das Ergebnis sieht man hier.

Abbildung 8.6: eyeCam User Interface



(a) Portrait Modus

(b) Landscape Modus

Das Grün beim Radiobutton wird vom Androidtheme gesetzt. Der Aufwand dies zuändern wäre zu gross gewesenen. Da aber das ganze restliche UI ebben mit der Schwarz/Weiss/Grau Farbenpalette designet wurde ist der Kontrast für die Farbenblinde genug gross um den Unterschied zu erkennen.

8.4.2 Intuitiv

Wir versuchte die Applikation so Intuitiv wie nur möglich zu machen. Am besten ist es uns wohl bei der Farberkennung gelungen.

Man kann ganz einfach während des Previewing auf das Objekt des Interesses tippen. Dann die Applikation haltet das Previewing auch schon an und verrät dann auch gleich einem die Farbe des Objektes.

Icons

Die Icons sind natürlich extrem wichtig wenn man Wert auf eine intuitive Bedienung legt. Also versuchten wir hauptsächlich bekannte Icons zu verwenden die jeder kennt.

Abbildung 8.7: Filter Icon

Dies ist das Icon um die Filter auszuwählen. Wir mussten feststellen das für Leute die nicht im Büro arbeiten nicht gerade das intuitivste Icon ist. Aber so bald sie einem Mal darauf getippt haben ist ihnen alles klar.

(a) kein Licht | **(b)** Licht off | **(c)** Licht on

Abbildung 8.8: Licht Icon

Hier muss man vielleicht schnell das erste Icon erklären. Dieses Icon sehen nur die Leute die ein Smartphone haben welches kein Licht besitzen. Wir wollen hier aber trotzdem zeigen das die Applikation die Möglichkeit unterstütz Licht ein bzw. auszuschalten wenn man denn eins hat. Vielleicht beeinflusst das ja den nächsten Smartphonekauf.

(a) Pause | **(b)** Play

Abbildung 8.9: Play/Pause Icons

Da ja das Previewing aufhört so bald man auf das Object seines Interesses tipp, wollten wir das sich dem Benutzer sofort klar wird das nun das Previewing angehalten hat und das er es jeder Zeit wieder einschalten kann. Und da gibt es wohl keine bekanntere Icons als unser zwei hier.

Abbildung 8.10: Settings Icon

Zum Icon für die Einstellungen gibt es nicht viel zu sagen aussehr das es das Standard Icon von Android ist und somit einen hohen Wiedererkennung Effekt hat.

8.4.3 Clean and Simpel

Clean and Simpel das war uns ganz wichtig. Die Benutzer sollen sich ja nicht durch einen Irrgarten von Einstellung bewegen müssen bis sie zur gewünschten Funktionalität kommen.

Aus den Interviews mit dem Probanden hat sich nämlich herauskristallisiert das sie gar keine grossen Einstellungen brauchen. Sondern eher mal das Licht bzw. kurz auf Stop drücken wollen um gleich mehre Farben mit einander zu vergleichen. Und dann evtl. die Filter zu wechseln.

Diese Erkenntiss haben versuchten wir best möglichst um zu setzen, dabei kam unser Controlbar raus (siehe nächstes Bild).

Abbildung 8.11: Controlbar

Über diese vier Knöpfe kann man die ganze Applikation steuern, mehr braucht es nicht. Es kam sogar schon vermehrt das Feedback das man die Einstellungen doch nicht bräuchte. Somit könnte es sein das das UI noch simpler und sauberer wird. Aber dies sind Entscheidungen die diese Arbeit nicht mehr betreffen und somit auch nicht weiter drauf eingegangen wird.

8.4.4 Fancy

Mit Fancy sind all bei unsere Applikation hauptsächlich die Animationen gemeint. Durch die Animationen versuchten wir einen Hauch von Leben in die Applikation zu bringen. Dies hat das Ziel dem Benutzer ein möglichst angenehmes Erlebnis zu verschaffen. So das er richtig Freude hat wenn er mit der Applikation interagiert.

Besondern Wert legten wir auf die Drehanimation. Welche beim Drehen des Smartphone vom der Portrait Stellung in die Landscape Stellung startet bzw. endet. Gerade durch diese Animation verlieht unsre Applikation ihr “virtuelles Gesicht".

9 Implementation

Tabelle 9.1: Dokumenthistory - Implementation

Rev.	Datum	Wer	Änderung
0.1	22.03.2011	Dominik Spengler	Prototypen Dokumentation erstellt
0.2	27.03.2011	Dominik Spengler	Prototypen Dokument um Prototyp 2 erweitert
0.3	19.04.2011	Dominik Spengler	Implementationskonzepte erstellt
0.4	26.05.2011	Dominik Spengler	Implementationskonzepte überarbeitet
0.5	28.05.2011	Patrice Müller	UI erstellt

9.1 Prototypen

Da es sich bei der Anforderung an die Performance der Applikation um ein grosses Risiko handelt, wurden verschiedene Architekturprototypen entwickelt. Im folgenden soll auf die einzelnen Implementationen und Erkenntnisse eingegangen werden

9.1.1 Prototyp 1

Beim ersten Prototypen handelt es sich um eine Implementation komplett in Java. Über ein SurfaceHolder#Callback Interface wird die Methode *onPreviewFrame()* aufgerufen und das so erhaltene Preview Frame bearbeitet. Um die Performance besser evaluieren zu können, wurde keinerlei Bildbearbeitung durchgeführt sondern lediglich die erhaltenen Frames so bearbeitet, damit man sie darstellen konnte. Der dazu nötige Code ist sieht folgendermassen aus:

```
@Override
public void onPreviewFrame(byte[] data, Camera cam) {
    int w = cam.getParameters().getPreviewSize().width;
    int h = cam.getParameters().getPreviewSize().height;

    YuvImage yuv = new YuvImage(data, NV21_FORMAT, w, h, null);
    if (yuv == null) Log.i(VIEW_LOG_TAG,"Bitmap null");
    Rect rect = new Rect(0, 0, w, h);
    ByteArrayOutputStream output_stream = new ByteArrayOutputStream();
    yuv.compressToJpeg(rect, 50, output_stream);

```

```
    // Convert from Jpeg to Bitmap
13  Bitmap bmp = BitmapFactory.decodeByteArray(
        output_stream.toByteArray(), 0, output_stream.size());
15
    Canvas canvas = getHolder().lockCanvas();
17  canvas.drawBitmap(bmp, matrix, paint);
    getHolder().unlockCanvasAndPost(canvas);
19 }
```

Klassendiagramm

Um das Klassendiagramm kleiner zu halten, sind nur die wirklich genutzten Methoden der Android Klassen dargestellt.

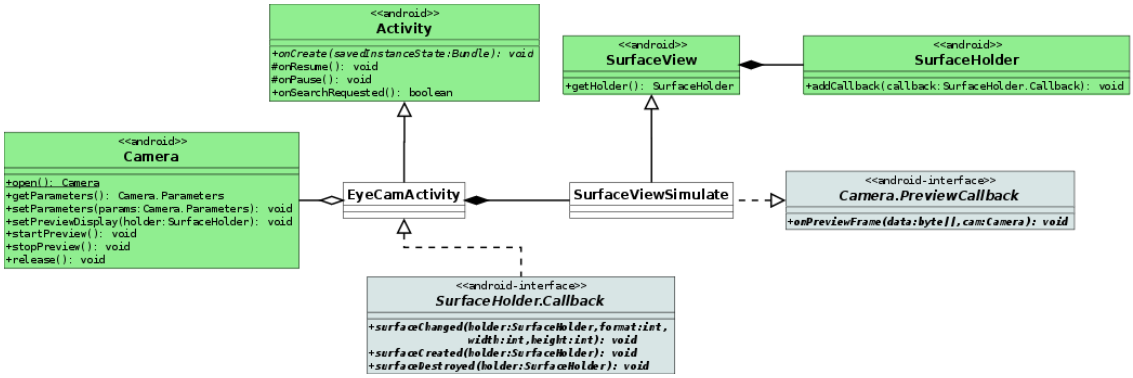


Abbildung 9.1: Prototyp 1

Der Code ist unter dem Tag prototype-1 im Repository verfügbar.

Erkenntnisse

- Kamera sendet Preview Frames in YUV Kodierung
- Performance mit Java-Implementation sehr schlecht
  - es muss mit native Code gearbeitet werden um Performanceanforderungen zu erfüllen

9.1.2 Prototyp 2

Da der Prototyp 1 hinsichtlich der Performance nicht akzeptabel war, wurde in einer zweiten Iteration der Prototyp 2 entwickelt, welcher die Bildbearbeitung in eine Native Bibliothek auslagert. Die Frames der Kamera werden ebenfalls wieder über einem *Camera.PreviewCallback* Interface entgegengenommen. Dieses Mal geschieht die Bildbearbeitung jedoch nicht mehr im Java Code, sondern in einer C-Bibliothek welche die bearbeiteten Frames direkt in den Pixel Buffer eines bereitgestellten Bitmap schreibt. Der Pixel Buffer des Bitmaps wird über eine, seit Android 2.2 zur Verfügung stehende, stabile Schnittstelle aus dem C-Code angesprochen.

Klassendiagramm

Wie bereits im Prototyp 1 werden der Einfachheit halber nur die genutzten Methoden der Android Klassen dargestellt. Das Orange eingefärbte Objekt stellt die C-Bibliothek dar.

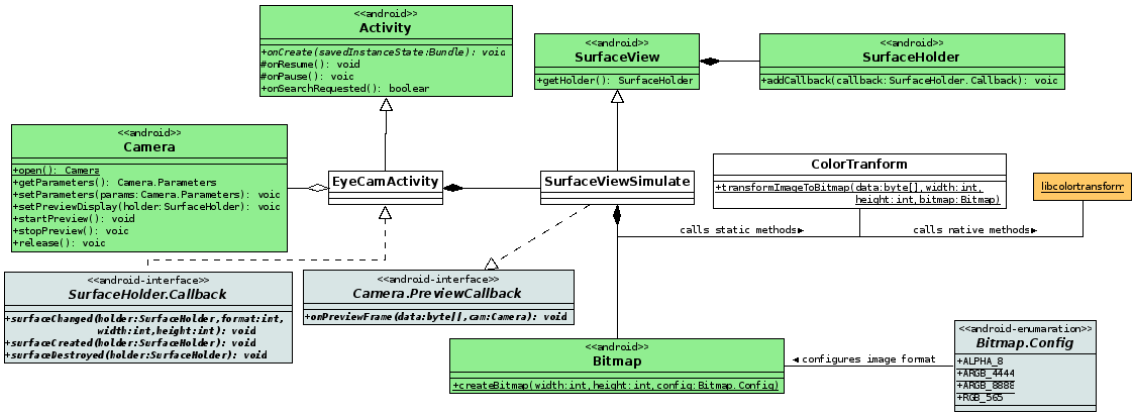


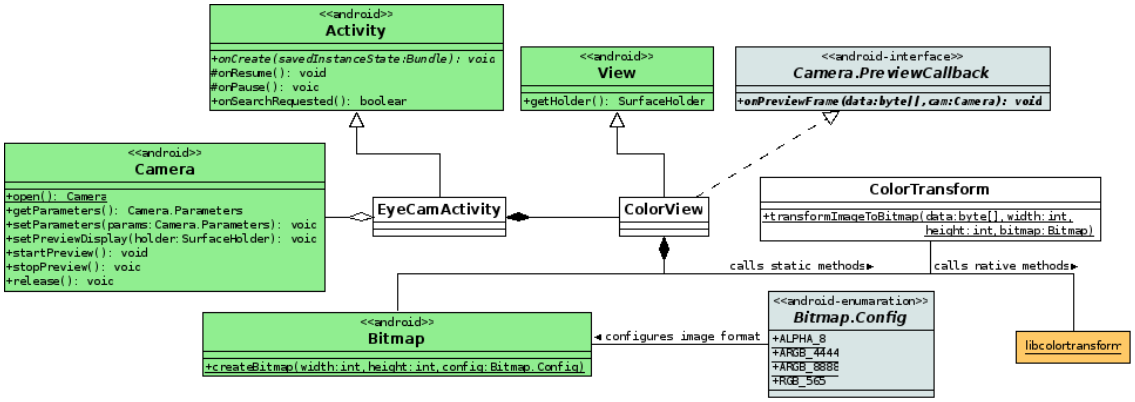
Abbildung 9.2: Prototyp 2

Der Code ist unter dem Tag prototype-2 im Repository verfügbar.

Erkenntnisse

- C++ Schnittstelle vom Android JNI verhält sich eigenartig
  - es musste C-Code für die Bibliothek verwendet werden
- SurfaceView wird eigentlich nicht benötigt
  - SurfaceHolder wird nicht benötigt
  - SurfaceHolder.Callback wird nicht benötigt

Allgemein lässt sich sagen dass durch die oben beschriebenen Erkenntnisse die erste Architektur überdenkt werden musste. Nachdem die Erkenntnisse des zweiten Prototypen umgesetzt wurden, reduzierte sich die Architektur auf folgendes Klassendiagramm.



9.2 Implementationskonzepte

Da wir bisher keinerlei Erfahrung in der Bild- und Videobearbeitung gemacht haben, stellte die Bildbearbeitung eine grosse Schwierigkeit unseres Projektes dar. Ebenfalls haben wir mit der Entwicklung der Nativen Bibliothek in C Neuland betreten. Im nachfolgenden wird auf die einzelnen Implementationskonzepte eingegangen.

9.2.1 Native Bibliothek

Weil das Android Framework keine performante Möglichkeit bietet, das von der Kamera erhaltene Bildformat im Java Code zu bearbeiten, musste auf eine native C-Bibliothek ausgewichen werden. Theoretisch wäre auch C++ möglich gewesen, allerdings wird C vom Android Native Development Kit (NDK) besser unterstützt und war für unsere Zwecke ausreichend.

Seit Android 2.2 existiert eine minimale API für den Zugriff auf den Pixel Buffer von Java Bitmap Objekte, welche sich ideal für unsere Zwecke eignet. Es werden folgende Methoden bereitgestellt:ne

```
1 /**
  *
3 * Given a java bitmap object, fill out the AndroidBitmap struct
  * for it. If the call fails, the info parameter will be ignored
5 */
```

```
7  int AndroidBitmap_getInfo(JNIEnv env, jobject jbitmap,
                             AndroidBitmapInfo* info);
9
   /**
11  * Given a java bitmap object, attempt to lock the pixel address.
    * Locking will ensure that the memory for the pixels will not move
13  * until the unlockPixels call, and ensure that, if the pixels had
    * been previously purged, they will have been restored.
15  *
    * If this call succeeds, it must be balanced by a call to
17  * AndroidBitmap_unlockPixels, after which time the address of the
    * pixels should no longer be used.
19  *
    * If this succeeds, *addrPtr will be set to the pixel address. If
21  * the call fails, addrPtr will be ignored.
    */
23
   int AndroidBitmap_lockPixels(JNIEnv env, jobject jbitmap,
25                              void** addrPtr);

27 /**
    * Call this to balanace a successful call to
29  * AndroidBitmap_lockPixels
    */
31 int AndroidBitmap_unlockPixels(JNIEnv} env, jobject jbitmap);
```

Um die notwendige Art des Polymorphismus für das Setzen der Effekte zu erreichen, musste auf Funktionspointer ausgewichen werden. In der Hauptfunktion der Bildtransformation werden die Werte der YUV Pixel ausgelesen und anhand des mittels setEffect() gesetzten Funktionspointer umgewandelt.

9.2.2 Farbtransformation

Die Android Kamera liefert das Bild standardmässig in NV21 (oder yuv420sp) kodierten Frames[1]. Dabei handelt es sich um ein Bildformat im YUV Farbraum mit 4:2:0 Farbunterabtastung, wobei die vier für den Faktor der Abtastrate des Helligkeitsignals Y, die zwei für die beiden Farbsignale U und V ist[2]. Das heisst es werden vier Pixel des Y-Kanals mit jeweils einem Pixel des U- und des V-Kanals dargestellt, wobei jeder Pixel ein byte gross ist. Die interne Repräsentation sieht folgendermassen aus:

1 http://developer.android.com/reference/android/graphics/ImageFormat.html#NV21
2 http://de.wikipedia.org/wiki/Farbunterabtastung

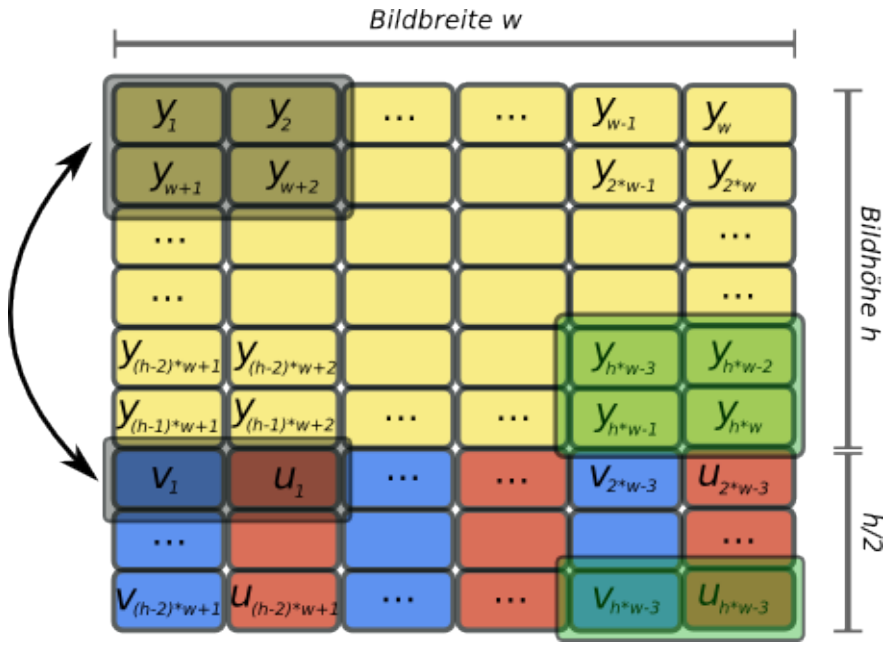


Abbildung 9.3: Aufbau von yuv420sp

Da es sich beim YUV Farbmodell um einen in der High-Level Informatik unüblichen Farbraum handelt und das Android Framework auch keine Möglichkeit bietet dieses Format performant darzustellen, stellte eine erste Schwierigkeit die Konvertierung des Eingangssignals dar. Als Zielformat der Konvertierung standen durch die Nutzung des Pixel Buffer Zugriffs für Java Bitmap Objekte[1] standen die von der Bitmap Klasse bereitgestellten Formate zur Auswahl:

- ALPHA_8
- ARGB_4444
- ARGB_8888
- RGB_565

Da der Alpha-Kanal für die Applikation nicht benötigt wird, stellte RGB_565 die naheliegenste Wahl dar.

Die Konvertierung des YUV Farbraums in den RGB Farbraum ist in der ITU Recommendation ITU-R BT.601-7[2] beschrieben. Unglücklicherweise handelt es sich bei dem beschriebenen Verfahren um Berechnungen in Floating-Point Arithmetik. Floating-Point Berechnungen sind

1 http://developer.android.com/sdk/ndk/overview.html#tools
2 http://www.itu.int/rec/R-REC-BT.601-7-201103-I/en

typischerweise langsamer als Fixed-Point Berechnungen, ausser natürlich es existiert eine spezieller Prozessoreinheit für Floating-Point Arithmetik (z.B. GPU). Aufgrund der Generizität der Android Platform und unserem Wunsch die Applikation auch auf älterer Hardware performant Nutzen zu können, mussten die Berechnungen durch Integer approximiert werden. Diese Approximation wurde durch Skalierung des Wertebereichs erreicht.

Aus der ITU Recommendation gehen die folgenden Transformationsmatrizen hervor:

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1.40763 \\ 1 & -0.345518 & -0.717 \\ 1 & 1.77916 & 0 \end{pmatrix} \times \begin{pmatrix} Y \\ U \\ V \end{pmatrix}$$

Weil die Eingabewerte acht bit breit sind, wird durch Multiplikation mit 2ˆ16 wird eine Skalierung des Wertebereichs auf [2ˆ16,2ˆ24] erreicht. Im Anschluss an die Berechnungen muss dann noch durch 2ˆ16 geteilt werden, was effizient mit einem Bitshift um 16 erreicht werden kann.

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 65536 & 0 & 92250 \\ 65536 & -22644 & -46990 \\ 65536 & 1165961 & 0 \end{pmatrix} \times \begin{pmatrix} Y \\ U \\ V \end{pmatrix}$$

9.2.3 Simulation der Farbenblindheit

Die Simulation der Farbenblindheit basiert auf einem Algorithmus beschrieben von Hans Brettel, Francoise Vienot und John Mollon in ihrem Paper “Computerized simulation of color appearance for dichromats". Es handelt sich dabei um dieselbe Simulation welche auch für den Daltonize Algorithmus beschrieben bei Vischeck verwendet wird. Wie im Paper beschrieben, wird als erstes eine Transformation in den LMS Farbraum vorgenommen, in diesem die Fehlermatrix angewandt und anschliessend die Transformation in den RGB Farbraum durchgeführt.

9.2.4 Farberkennung

Die Farberkennung stellte ein weiteres Risiko unserer Applikation dar. Weder der RGB noch YUV Farbraum eignen sich sonderlich gut für die Farberkennung, deshalb musste auf einen dritten Farbraum ausgewichen werden. Durch die Identifikation der Farbe losgelöst von Helligkeit und Sättigung bieten sich die beiden Farbräume HSL und HSV für die Farberkennung an[1]. Weil die Erkennung von Weiss und Grau auf den ersten Blick einfacher im HSL Farbraum erscheinen, haben wir uns dafür entschieden.

1 http://de.wikipedia.org/wiki/HSL-Farbraum

Da es in unserer Applikation nötig ist, die Farbe zu erkennen, welche von der Kamera aufgenommen wird, und nicht jene die auf dem Bildschirm dargestellt wird, ist es nötig direkt die Daten der Kamera im YUV zu verwenden. Der Einfachheit und der Leserlichkeit halber werden die YUV Daten erst in den RGB Farbraum umgewandelt und erst danach in HSL wo die effektive Farberkennung stattfindet. Diese Funktionalität konnte glücklicherweise ohne grosse Probleme in Java implementiert werden. Es handelt sich dabei um eine sehr nahe liegende Implementation mit folgenden Schritten:

1. wenn Sättigung klein => Schwarz
2. wenn Sättigung hoch => Weiss
3. wenn Wert klein => Grau
4. die Farbe wird bestimmt je nach Hue Wert
5. wenn Farbe == Rot oder Orange => überprüfe Sättigung und Wert ob es sich eher um Braun handelt

Die Schwellenwerte für Schwarz, Wiess, Grau und Braun sind dabei empirisch bestimmt worden.

sectionUI

Unser Gedanken bezüglich dem UI-Design haben wir ja im gleichnamigen Kapitel erläutert. In diesem Kapitel wollen wir uns der Implementation verstärkt widmen.

Einer der wichtigsten Punkt bezüglich UI-Implementation war die Trennung vom UI-Code und dem restlichen Code. Das schöne an Android ist es bietet einem die Möglichkeit die UI über XML zu definieren. Was die Trennung das Code extrem vereinfacht. Es besteht sogar die Möglichkeit seine eigne View's und Widget mit ihren speziellen Attributen über XML zu definieren.

Da so gut wie alles über Javadoc schon dokumentiert ist. Wollen wir hier nur noch gewisse spezial Fälle, Implementierungsentscheidungen und evtl. Stolperfallen beschreiben.

9.2.5 Spezielle Anforderung wegen der Kamera

Um die spezielle Anforderung der Views und Widgets erfüllen zu können. Mussten wir fast alle UI-Komponenten die wir verwenden wollten erweitern oder selber schreiben. Das spezielle an unser Applikation ist, das wir wegen der Kamera die Applikation in die Lanscape-Orientierung zwingen müssen und in dieser auch bleibt, egal was komme.

Dies ist nötig weil die Kamera das Bild immer im Landscapemodus liefert.

Dies erreicht man in dem die **android:screenOrientation** im AndroidManifest.xml auf **“landscape"** setzt, siehe Zeile 3.

```
1 <activity android:name="eyeCam"
          android:label="@string/app_name"
3         android:screenOrientation="landscape"
```

9.2.6 BubbleView–View

Die BubbleView Klasse fungiert als Container für alle Views die in den Popup Windows erscheinen sollen. Dies war nötig um die Orientierungsänderung der Views zu realisieren. Wir haben die *onMeasure* Methode als Einstiegspunkt verwendet.

Am besten stellt man sich so eine View wie eine Bild vor. Also, es hat einen Rahmen mit einer Höhe und einer Breite, in diesem Rahmen befindet sich dann das eigentlich Bild. Genau wie die Bilder haben auch die Views eine Höhe und eine Breite, aber anstelle der Leinwand haben sie ein Canvas Objekt welches den eigentlichen Inhalt beinhaltet.

Das Canvas Objekt orientiert sich an Hand eines Kartesisches Koordinatensystem was einem die Möglichkeit gibt mit bisschen Linearen Algebra das Bild zu drehen.

Das ganze sieht dann wie folgt im Code aus:

```
1 @Override
protected void onMeasure(int widthMeasureSpec
3                        , int heightMeasureSpec) {
        super.onMeasure(widthMeasureSpec, heightMeasureSpec);
5       int h = getMeasuredHeight();
        int w = getMeasuredWidth();
7
if (mOrientation == Orientation.PORTRAIT)
9       setMeasuredDimension(h, w);
updateMatrix();
11 }

13 private void updateMatrix() {
        mRotatioMatrix.reset();
15      float width = getWidth();
        float height = getHeight();
17
switch(mOrientation){
19      case LANDSCAPE_RIGHT:
                mRotationMatrix.setRotate(180, width/2.0f
21                      , height/2.0f);
                break;
23      case PORTRAIT:
```

```
                    mRotationMatrix.setRotate(-90, 0.0f, 0.0f);
25                  mRotationMatrix.postTranslate(0.0f, height);
                    break;
27          }

29  invalidate();
    }
```

Das Tauschen der von Höhe und Breite geschied auf der Zeile 7.Was man leider jetzt gerade ebben nicht sieht, ist das die Reihenfolge der Parameter der Methode *setMeasuredDimension* nicht Höhe, Breite sonder Breite, Höhe ist. Der wirkliche Schlüsselpunkt ist aber die Zeile 8 bzw. die Methode *updateMatrix* hier wird nämlich die Rotationmatrix für das Canvas Objekt berechnet.

9.2.7 ControlBar–View

Die Klasse *ControlBar* hat zur Aufgabe alle Belangen ihrer Kinderviews zu managen. Sei das jetzt das Austauschen von *OnClickListener*, das starten von Animation oder das Kommunizieren mit der *Activity*.

9.2.8 FloatingBubble / MenuBubble–Widgets

Die beiden Klassen FloatingBubble und MenuBubble sind Erweiterung der Klasse BubbleView und realisieren die Menus (MenuBubble) bzw. die Popup Windows (FloatingBubble).

9.2.9 PreferencesRadioGroup / PreferencesRadioButton–Widgets

Die beiden Klassen *PreferencesRadioGroup* und *PreferencesRadioButton* werden dazu verwendet die Menus zu realisieren. Die Idee dahinter ist das wir pro RadioGroup eine Map Abbildung haben. D.h. jede *PreferencesRadioGroup* wir als Map angeschaut und die Menutext werden als Schlüssel verwendet um auf die Einstellungswerte zu zugreifen, welche wir hinterlegt haben.

9.2.10 XML

Wie schon oben beschrieben, war es unser Ziel den ganzen UI-Code ins XML auszulagern. Das ganze UI ist in vier XML-Datein aufgeteilt, was einem das Wiederverwenden und Pflegen vereinfacht. Die vier Dateien sind main.xml, cam_control.xml, filter_menu.xml und settings_menu.xml sie werden gleich hier unten beschrieben.

main–XML

Diese main.xml sieht bisschen anderst aus als bei den meisten Android Applikationen, man siehe Zeile 2 bzw. Zeile 10. Hier wurde ein kleiner Trick verwendet um einen Overlay zu realisieren. D.h. die Controlbar wird über die Colorview gelegt. Dies war wichtig um das Kamerabild nicht zu verzehren.

Die genau Anleitung wie man das bewerktstelligt und noch weiter Vorteile dieser Technik findet man unter dem Link: http://developer.android.com/resources/articles/layout-tricks-merge.html.

Die Zeile 4 zeigt wie man View bzw. Widget anspricht das man selber geschrieben hat.

```
1  <?xml version="1.0" encoding="utf-8"?>
   <merge xmlns:android="http://schemas.android.com/apk/res/android">
3
       <ch.hsr.eyecam.view.ColorView
5          android:id="@+id/cameraSurface"
           android:layout_width="match_parent"
7          android:layout_height="match_parent">
       </ch.hsr.eyecam.view.ColorView>
9
       <include
11         layout="@layout/cam_control"
           android:id="@+id/cam_control"/>
13
   </merge>
```

cam_control–XML

Die XML-Datei ist im Kontext Android UI-Entwicklung eigentlich selbsterklärten. Dennoch wollen wir hier kurz erklären wie man selbst definierte Attribute anspricht, wie man sie definiert ist in einem späteren Kapitel beschrieben.

Wie man auf Zeile 4 sehen kann mussten wir zu erste einen eigenen Namensraum definieren bevor wir unser Attribute nutzen konnten. Obschon wir den vollständigen Namen der Klasse angeben mussten.

```
   <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout
     ...
4    xmlns:eyecam="http://schemas.android.com/apk/res/ch.hsr.eyecam"
     ...>
6    ...
           <ch.hsr.eyecam.widget.StateImageButton
```

```
8                   style="@style/ImageButton.Center"
                    android:id="@+id/imageButton_Pause"
10                  eyecam:imgResTrue="@drawable/ic_menu_play"
                    eyecam:imgResFalse="@drawable/ic_menu_pause"/>
```

settings_menu / filter_menu–XML

Was etwas speziell an diesen Beiden XML-Dateien ist, dass sie in keiner anderen XML-Datei referenziert werden. Das liegt daran das wir diese UI's zur Laufzeit aus dem XML generieren. Das Generieren wird von Klasse *ControlBar* angestossen und wird dann von der Android Klasse *LayoutInflater* übernommen. Wie der *LayoutInflater* genau funktioniert finden sie hier http://developer.android.com/reference/android/view/LayoutInflater.html.

Eigene Attribute im XML definiert

Als wir eigene Attribute im XML ansprechen wollten mussten wir erst mal dem Android sagen das wir was zum ansprechen haben. Nun dies geschah in dem wir im Order *res/values/* eine neue XML-Datei namens *attrs.xml* definierten. In dieser Datei mussten wir dann die View bzw. Widget Klassen als *declare-styleable* definieren, siehe Zeile 1.

In einem zweiten schritt konnten wir dann die eigentliche Attribute definieren. Beim Definieren muss darauf geachtet werden das man das richtig Format nimmt sonst kann man sie später nicht ansprechen oder es gibt eine *Exception* . In unserm Fall hier mussten wir das Format *reference* nehmen da wir ja Ressourcen aus der *R*-Datei ansprechen wollten.

```
1 <declare-styleable name="StateImageButton">
          <attr name="imgResTrue" format="reference" />
3         <attr name="imgResFalse" format="reference"/>
          <attr name="imgResDisabled" format="reference"/>
5 </declare-styleable>
```

Im Code XML-Werte auslesen

Im vorhergegangen Kapitel haben wir kurz gezeit wie wir unser Attribute selbst definiert haben. Um nun das Bild abzurunden zeigen wir noch wie wir es das auslesen realsiert haben. Dafür mussten wir dank dem Androidframwork nicht mehr all zu viel tun, nach dem mir mal die Dokumentation gefunden hatten ;P.

Dennoch gab es einen kleinen Stolperstein. Um der Klasse mitzuteilen welche Attribute wir gesetzt hatten mussten wir den entsprechenden Konstruktor überschreiben, siehe Zeile 1.

Was uns am Anfang nicht ganz klar war. Aber danach war die Sache wirklich keine grosse Hexerei mehr. Wie man an Zeilen 3–7 sieht.

```
1 public StateImageButton(Context context, AttributeSet attrs) {
...
3 TypedArray typedArrayAttr = context.obtainStyledAttributes(attrs
    ,R.styleable.StateImageButton);
5
mImgResTrue = typedArrayAttr.getResourceId(
7         R.styleable.StateImageButton\_imgResTrue,
          R.drawable.ic\_menu\_sad);
```

Die zuvor angesprochene Dokumentation findet man hier http://developer.android.com/resources/tutorials/views/hello-gallery.html

10 Testing

Tabelle 10.1: Dokumenthistory - Testing

Rev.	Datum	Wer	Änderung
0.1	23.05.2011	Dominik Spengler	Unit-Tests erstellt
0.2	24.05.2011	Dominik Spengler	Systemtests erstellt

10.1 Unit-Tests

In unserer Applikation gibt es abgesehen von den Systemtests zwei zentrale Funktionalitäten, welche man testen könnte:

- Farbumwandlung (YUV $\rightarrow$ RGB)
- Farberkennung

Theoretisch wäre es auch möglich die verschiedenen Farbfilter wie zum Beispiel die Simulation der Farbenblindheit zu testen. Allerdings liegt es in der Natur der Farbwahrnehmung, dass sie äusserst subjektiv und somit schwer zu testen ist. Wäre dem nicht so, gäbe es kein Zielpublikum für unsere Applikation. Somit macht es nicht viel Sinn die Simulation und andere Filter zu testen, ohne genau definieren zu können wie das erwartete Resultat auszusehen hat.

10.1.1 Farberkennung und -umwandlung

Eine Schwierigkeit im testen der Farbumwandlung ist das von Android verwendete Format yuv420sp (oder NV21). Nach unserem Wissen gibt es kein Bildbearbeitungsprogramm welches dieses Format unterstützt. Deshalb wurden für die Tests byte buffer mit den abgeschätzten Werten für die Farben verwendet. Wegen der Schwierigkeit die exakten YUV byte Werte für die Farben zu bestimmen, decken unsere Tests nur die Farben Rot, Grün, Blau, Schwarz, Weiss und Gelb ab. Siehe Implementationskonzepte für genauere Details.

Zusätzlich wurden die tests erschwert durch den für die Farbumwandlung verwendeten Algorithmus. Es handelt sich dabei um eine etwas modifizierte Methode nach ITU-R Recommendation

BT.601. Obwohl es sich um die gängige Art und Weise der Transformation handelt, wird reines Rot nicht in RGB 255,0,0 umgewandelt.

10.2 Systemtests

Android bietet die Möglichkeit für automatisierte Affentests . Der Affentast hat bewiesen dass unsere Applikation schnell und stabil läuft[1].

10.3 Usability Tests

Die Usability Tests haben wir in den Gewohnten Umgebungen der Probanden durchgeführt. Was heisst das? Da wir die Probanden persönlich kannten bzw. kennen war es für uns ein leichtes jeder Zeit mit ihnen in Kontakt zu tretten un sie nach ihrer mein zu fragen bzw. ihnen gewisse Aufgaben zu stellen. Diese Aufgaben war immer auf unser Szenarios bezogen, sei das mal zwei Glasflasche in einem schummrigen Pub zu unterscheiden oder heraus zu finden ob jetzt in einem Club die Toilette besetzt ist oder nicht.

Wir gingen diesen etwas unkonventionellen da wir daran glaubten das Test im Labor, von Ausnahme des Isihara-Test für unser Applikation nicht der richtig Weg wäre. Da es gerade bei einer Applikation für den Alltag so viele Variationen der Probleme gibt und noch viel mehr Unbekannte an man Anfang nicht dachte. Darum hatten wir uns für diesen etwas unkonventionellen Weg entschieden.

Dabei haben wir immer auf die folgen Kriterien geachtet:

- Perforamnce
- Einhändig bedienbar.
- Intuitiv bedienbar
- Kein Absturz der Applikation

Wir für uns haben immer wieder den Isihara-Test verwendet um unser Applikation zu testen. Auch beim letzten Usability den wir mit einem Video Dokumentiert haben, diente der Isihara-Test als Einstiegs-Test.

Das Video findet man im open.eyecam@gmail.com-Account auf Youtube, unter Private Videos.

1 Besonders konnte die NullPointerException aus dem Usability Test und dem Market nicht rekonstruiert werden.

11 Resultate und Weiterentwicklung

Tabelle 11.1: Dokumenthistory - Resultate und Weiterentwicklung

Rev.	Datum	Wer	Änderung
0.1	26.05.2011	Dominik Spengler	Resultate, Möglichkeiten der Weiterentwicklung erstellt

11.1 Resultate

Die Resutate wurden bereits im Technischen Bericht unter Erreichte Ziele beschrieben. Für eine Beschreibung der Problemlösung und der gemachten Überlegungen wird auf Implementationskonzepte verwiesen.

11.2 Möglichkeiten der Weiterentwicklung

Obwohl die Grundfunktionalität unserer Applikation implementiert wurde, gibt es immer Möglichkeiten der Weiterentwicklung. In diesem Kapitel werden wir die aus unserer Sicht sinnvollen Ergänzungen zur Applikation vorstellen.

11.2.1 Verbesserung der Farberkennung

Ein wesentliches Kernfeature unserer Applikation stellt die Farberkennung dar. Wie in Erreichte Ziele kurz beschrieben hat diese allerdings etwas Probleme in schlechteren Lichtverhältnissen. Ebenfalls benötigt die bereits vorhandene Logik weiteres Testing und möglicherweise Erweiterung.

Ein erster Schritt der Weiterentwicklung sollte deshalb sein, die Farberkennung zu verbessern.

11.2.2 Farberkennung über Bereich

Momentan funktioniert die Farberkennung auf einen einzelnen Pixel. Dies ist nicht ganz optimal, da man auf einem Touch Screen in der Regel mit dem Finger nicht genau auf die Stelle tippt, auf die man wollte. Dies könnte man dadurch Umgehen, dass man die Farberkennung nicht nur auf einen einzelnen Pixel, sondern auf einen Bereich von Pixeln durchführt. Da die Farberkennung komplett in Java implementiert ist, sollte dies keine grosse Schwierigkeit darstellen. Für die Berechnung des Durchschnitts könnte man beispielsweise in einem ersten Schritt eine simple Durchschnittsfunktion implementieren.

Wenn man nun einen Bereich wählt für die Farberkennung, ist aber die jetzige Sprechblase nicht gerade optimal um den Bereich zu kennzeichnen. Generell scheint die Sprechblase in einem ersten Moment etwas verwirrend zu sein (ist es jetzt genau der Pixel unter dem Pfeil?). Aus diesem Grund würde sich ein Kreis um den zu erkennenden Bereich anbieten. Zusätzlich wäre es sinnvoll wenn die Kreislinie einige Pixel dick ist und in der ermittelten Farbe des Bereiches eingefärbt wird.

11.2.3 User Interface Notifications

Obwohl das User Interface sehr simpel aufgebaut ist, wäre es durchaus sinnvoll dem Benutzer Rückmeldung mittels OSD Notifications geben würde, wenn immer nötig. Wann genau diese Rückmeldungen nötig sind, gilt es mit erweiterten Usability Testing herauszufinden.

11.2.4 Pinch to Zoom

In den Usability Tests und auch beim Vorführen der Applikation kam es häufiger vor, dass man die Farbe eines Bereiches erkennen wollte, welcher sehr klein war und daher schwierig zu treffen. Aus diesem Grund wäre es durchaus sinnvoll eine “Pinch to Zoom"Funktionalität im Pause-Modus der Applikation zu implementieren.

Eine Herausforderung wäre dabei die aktuelle Stelle der Farberkennung, und somit der Sprechblase, zu halten und mit dem Zoom zu skalieren.

11.2.5 Erweitertes Usability Testing

Wie bereits in 5.1 erwähnt, wurden für das Usability Testing lediglich zwei Testpersonen benutzt. Um eine bessere Aussage über die Verbesserungsmöglichkeiten der Applikation machen zu können, ist ein erweitertes Usability Testing nötig.

11.2.6 Report Funktionalität

Ein allgemeines Phänomen im Android Market ist es, dass im Verhältnis zu den Downloads wenige Leute Kommentare zu den Applikationen abgeben. Eine möglich Erklärung dafür ist, dass man Kommentare nicht direkt aus der Applikation abgeben kann, sondern diese aus der Market Applikation auf dem Smartphone machen muss. Damit mehr Leute Rückmeldung zu der Applikation machen, wäre es durchaus sinnvoll eine Report Funktionalität direkt in die Applikation einzubauen. So wäre es für Benutzer auch viel einfacher Feature Requests und Bug Reports zu erstellen.

12 Projektmanagement

Tabelle 12.1: Dokumenthistory - Projektmanagement

Rev.	Datum	Wer	Änderung
0.1	07.03.2011	Patrice Müller	Prozessmodell,Q-Massnahemen erstellt
0.2	07.03.2011	Dominik Spengler	Risikomanagement erstellt
0.3	15.03.2011	Dominik Spengler	Entwicklungsumgebung erstellt
0.4	22.03.2011	Dominik Spengler	Risikomanagement ergänzt

12.1 Entwicklungsumgebung

In diesem Kapitel wird die Konfiguration unserer Entwicklungsumgebung genauer beschrieben. Im Detail wird auf Redmine, Jenkins und Git eingegangen.

12.1.1 Redmine

Die Redmine Serverumgebung wurde um einige Plugins erweitert um uns in der Entwicklung zu Unterstützen:

- **Scrum-PM:** Dieses Plugin erweitert Redmine um ein Scrum Dashboard und Backlog. Es ermöglicht das erstellen von User-stories und Burndown Charts.
- **Hudson Plugin:** Dieses Plugin erweitert Redmine um einen Hudson Tab. Dadurch ist es möglich den Status der Builds des Jenkins Continuous Integration Server zu verfolgen.
- **DocPu Plugin:** Dank diesem Plugin kann man die gesamte Projektdokumentation im Redmine Wiki erstellen und danach anhand eines Templates LaTeX code generieren lassen.

Zusätzlich zu den oben erwähnten Plugins sind noch weitere installiert worden, welche lediglich der Bequemlichkeit dienen und nicht zur Erleichterung der Entwicklung beitragen.

- **Embedded Plugin:** Durch dieses Plugin ist es möglich HTML Dateien in einem weiteren Tab in Redmine einzubinden. Es wird benutzt um direkten Zugriff auf JavaDoc zu haben.

- **Gitrevision Download Plugin:** Dieses Plugin ermöglicht es die Revisionen des Git Repository Browsers herunterzuladen.

12.1.2 Jenkins

Beim Jenkins Continuous Integration Server handelt es sich um ein Community Projekt zur kontinuierlichen Integration geschrieben in Java. Das Projekt war ursprünglich unter dem Namen Hudson bekannt. Aufgrund eines Lizenz-Streits mit Oracle wurde das Projekt jedoch in Jenkins umbenannt[1] wobei jedes Projekt das jeweilige andere als Fork betrachtet.

Eine besondere Stärke von Jenkins ist die sehr gute und umfangreiche Erweiterbarkeit durch Plugins. Um Jenkins komplett in unsere Projektentwicklung einzubinden wurden folgende Plugins installiert:

- **Android Emulator Plugin:** Durch dieses Plugin wird das automatisierte starten des Android Emulators möglich. Besonders zu erwähnen ist, dass es auch möglich ist verschiedenste Emulator-konfigurationen nacheinander im selben Job zu testen.
- **Jenkins Git Plugin:** Ermöglicht die Einbindung eines Git Repositories in Jenkins.

Zusätzlich wurden noch folgende Plugins installiert um Benutzeroberfläche angenehmer zu gestalten:

- **Green Balls:** Dieses Plugin ersetzt die Standardmässig blauen Bälle für erfolgreiche Builds mit grünen Bällen.
- **Dashboard View:** Durch dieses Plugin ist es möglich personalisierte grafische Auswertungen der einzelnen Jobs anzuzeigen.
- **ChuckNorris Plugin:** Dieses Plugin zeigt anstelle des Jenkins dem Butler ein Bild von Chuck Norris (inklusive Programmier Fakten) auf jeder Build Seite.

Für den Build Prozess, sind zwei Jobs konfiguriert worden:

- **eyeCam–release build:** Bei diesem Job handelt es sich um einen wöchentlichen Build des master Branches. Er wird nur dann ausgeführt, wenn der “eyeCam–master"test Build erfolgreich für verschiedene Emulator Konfigurationen war.
- **eyeCam–debug build:** Dabei handelt es sich um einen täglichen Build des development Branches. Er wird nur dann ausgeführt, wenn der “eyeCam–development"test Build erfolgreich für die Standard Emulator Konfigurationen war.

1 http://jenkins-ci.org/content/hudsons-future

12.1.3 Git

Als Software zur Versionsverwaltung wird Git eingesetzt. Git wurde gewählt aufgrund der Einfachheit branches zu erstellen und wieder zu mergen. Ebenfalls zeigt die Erfahrung dass Git zu deutlich weniger Probleme bei der Arbeit im Team an der selben Codebase führt.

Bei der Entwicklung mit Git wird der Ansatz der Feature Branches[1] verfolgt. Grundlegend existieren zwei Branches:

- **master:** Der master Branch wird für voll funktionsfähige Releases genutzt. Er wird am Ende jedes Sprints mit dem development Branch gemerged.
- **development:** Im development Branch findet die Entwicklung statt. Dabei wird für jedes neue Feature und jeden neuen Bug ein weiterer Branch erstellt. Dies führt dazu, dass zu einem bestimmten Zeitpunkt niemals mehr als vier Branches existieren: master, development, feature von Herr Müller, feature von Herr Spengler. Damit beim push auf den Server die Revisions-History komplett erhalten bleibt, wird nach folgender Methodik gearbeitet:

```
$ git pull origin development
$ git checkout -b featureXY development
...
<Arbeit am featureXY>
...
\$ git pull origin development
\$ git checkout development
\$ git merge --no-ff featureXY
...
Updating ea1b82a..05e9557
...
\$ git branch -d featureXY
...
Deleted branch featureXY
...
\$ git push origin development
```

Nach dieser Vorgehensweise entsteht ein Branch Graph nach folgendem Vorbild:

1 http://martinfowler.com/bliki/FeatureBranch.html

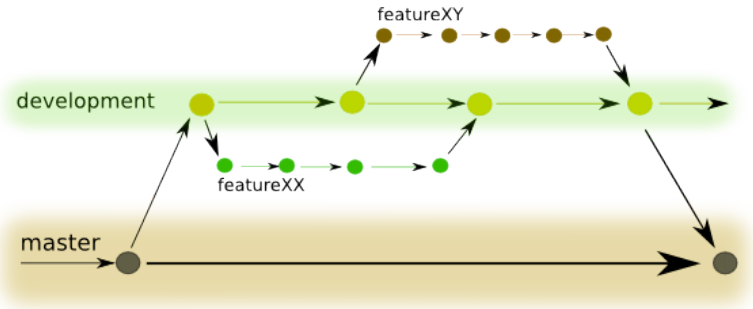


Abbildung 12.1: Git Feature Branch graph

12.2 Projektplan

Da wir uns für Scrum entschieden haben wird auch unser Projektplan in Sprints definiert. Das Frühjahrssemester (21.02.11–03.06.11) hat gesamt 15 Wochen. Diese 15 Wochen wurden wie folgt aufgeteilt.

12.2.1 Wochenaufteilung in Sprints

Tabelle 12.2: Projektplan

Nr	Datum	Sprint	Title	Spezielle Termine
1 2	21.02.11 28.02.11	Sprint 0	Infrastruktur Setup	Themenfindung
3 4	07.03.11 14.03.11	Sprint 1	Architektur Prototype	
5 6	21.03.11 28.03.11	Sprint 2	Algorithmen & Framework	
7 8	04.04.11 11.04.11	Sprint 3	UI & Threading	
9 10	18.04.11 25.04.11	Sprint 4	User Interface	 RC-Featurefreeze
11 12	02.05.11 09.05.11	Sprint 5	Usability Testing & Review	Codereview und Usability Testing Video Final Featurefreeze
13 14	16.05.11 23.05.11	Sprint 6	Finalization	Kriterienreview, Publishing, Code-freeze Plakat, Documentationsreview, Applikationsvideo
15	30.06.11		Hell Week	Abgabe (03.06.11)

12.2.2 Kurzbeschreibung der Sprints und Termine

Themenfindung

Als wir mit dem Projekt anfingen gingen wir davon aus, das wir für eine Partnerfirma eine Applikation für Google-TV entwickeln. Leider war das SDK für Google-TV noch nicht veröffentlicht. Somit mussten wir uns umschauen.

Infrastruktur Setup[1]

Entscheid der Infrastruktur, d.h welches Prozessmodell, was für eine Entwicklungsumgebung und wie sieht eigentlich unser Arbeitsumfeld Umfeld der Arbeit aus. So wie das Erstellen der ersten Dokument.

1 http://152.96.56.18/redmine/projects/eyecam/sprints/show?sprint%5Bid%5D=2&commit=Choose

Architektur Prototype[1]

- Sich in die Bildbearbeitung einlesen
- Dokumentation automatisieren
- Erste Bilder von der Kamera

Algorithmen & Framework[2] **Farbfilter Framwork:** Ein C-Framwork auf dem Android implementieren damit wir möglichst perfomant die Bilder der Kamera Live bearbeiten können. **Verschiedene Farbfilter testen:** Welche Farbfilter haben für die an Achromatopsie leidende Menschen die beste Wirkung. **Farbfilter auf Computer implementieren:** Als erste wird eine Simulation implementiert, welche ja nicht auf dem Android gebraucht wird. Um dann später die Farbfilteralgorithmen auf Performanz und Korrektheit zu testen.

UI & Threading

User Interface implmentieren mit all seinen Must-Have-Features. Beim Threading wollen wir schauen das die Bildbearbeitung ihren eigenen Thread bekommt, ob es gebraucht wird oder nicht ist die Entscheidung dieses Sprints.

Usability Testing & Review

Wir testen unser Applikation an unsern zwei Probanden und schauen wie dessen Feedback bezüglich unser Applikation ausfällt und ziehen daraus unser Fazit und ob noch was gemacht werden muss. Da wir mit unsern Must-Have-Features fertig sind, gibt es ein Codereview um die Qualität des Codes hoch zu halten.

Finalization

Dieser Sprint wird dazu verwendet die Verbesserungsvorschläge des Usability Testings umzusetzen, damit auch sicher gestellt ist das unser Applikation die Hilfe ist die sie sein sollte.

1 http://152.96.56.18/redmine/projects/eyecam/sprints/show?sprint%5Bid%5D=3&commit=Choose
2 http://152.96.56.18/redmine/projects/eyecam/sprints/show?sprint%5Bid%5D=4&commit=Choose

Hell Week

Letzte Refactoring am Code und Dokumentation Überprüfung. So wie Poster erstellen und sich auf die Präsentation vorbereiten.

12.3 Team und Verantwortlichkeiten

Das Team für diese Arbeit bildet sich aus zwei Personen, Herrn Dominik Spengler und Herrn Patrice Müller. Auf der folgende Tabelle sind die Verantwortlichkeiten aufgelistet.

Tabelle 12.3: Verantwortlichkeiten

Projektbereich / Dokument	**Verantwortlicher**
Dokumentation	Patrice Müller
Dokumentationautomation, Wiki => LaTex => PDF	Patrice Müller
Q-Massnahemen	Patrice Müller
Interviews	Patrice Müller
Szenarios	Patrice Müller
Konfiguarationmanagement	Dominik Spengler
Server	Dominik Spengler
Redmine-Installation	Dominik Spengler
Redmine-Instandhaltung	Dominik Spengler
Test-Environment	Dominik Spengler
Testen	Dominik Spengler, Patrice Müller
Domain Model	Dominik Spengler, Patrice Müller
Technisch-Implementierung	Dominik Spengler, Patrice Müller

12.4 Risikomanagement

12.4.1 Risikoanalyse

Risiko Nr.	Titel	Beschreibung	max. Schaden [h]	Wahrscheinlichkeit	gewichteter Schaden [h]	Indikator	Massnahmen
Allgemeine Risiken							
R1.1	Streit im Team	Konflikte im Team	40	1.00%	0.4	Schlechte Umgangssprache , Gereiztheit	Kommunikation im Team verbessern
R1.2	Ausfall eines Teammitglied	Ein Teammitglied fällt wegen Krankheit oder Ähnlichem aus	40	5.00%	2		
R1.3	Ausfall der Infrastruktur	Redmine, Jenkins Fällt aus	400	1.00%	4	Hardware, Software Teilausfälle	Backup
Technologische Risiken							
R2.1	Cross Device Kompatibilität	Die Applikation läuft nur auf bestimmten Android Geräten	50	5.00%	2.5	Applikation verhält sich unterschiedlich auf verschiedener Emulatorhardware	Testing auf verschiedenenen Emulatoren in Jenkins
R2.2	Versionskompatibilität	Die Applikation läuft nur auf bestimmten Android Versionen	50	5.00%	2.5	Applikation verhält sich unterschiedlich auf verschiedener Emulatorversion	Native Code gegen stabile Interfaces programmieren
R2.3	Native Code nötig	Aufgrund von Performance Issues muss auf native Code ausgewichen werden	100	10.00%	10	Code komplett in Java geschrieben ist viel zu langsam	frühe Implementation des ersten Prototypen in Java
R2.4	Probleme in der Entwicklung mit JNI	Die Entwicklung mit dem Android NDK gestaltet sich schwieriger als erwartet	75	10.00%	7.5	JNI Code verhält sich nicht erwartungsgemäss, Native Methoden performen nicht	Native Code gegen stabile Interfaces programmieren
R2.5	Probleme bei den Bildtransformationsalgorithmen	Die verwendeten Algorithmen verhalten sich nicht erwartungsgemäss.	75	10.00%	7.5	Das Transformierte Bild enthält Fehler	Entwicklung Testprogramm am PC mit offenen Bildbearbeitungs-Bibliotheken
R2.6	Probleme bei der Entwicklung des UI	Der Aufwand für die Entwicklungs des User Interfaces wurde unterschätzt.	35	100.00%	35	Das gewünschte User Interface kann nur mit grossem Aufwand implementiert werden.	Mehr Zeit investieren, da das User Interface eine sehr wichtige Komponente der Applikation darstellt
Entwicklungsrisiken							
R3.1	Fehleinschätzung des Aufwandes	Der Aufwand der Einzelnen Issues wird falsch eingeschätzt	50	10.00%	5	Die Sprint Backlogs werden immer verkleinert	Userstories in kleinere Tasks unterteilen
R3.2	Vision nicht Umgesetzt	Die Features der Vision können nicht umgesetzt werden	100	1.00%	1	Implementation der Kernfeatures wird immer weiter aufgeschoben	Features der Vision als erstes Implementieren
R3.3	Performance	Applikation läuft viel zu langsam	100	5.00%	5	Erste Test auf Dev. Device laufen sehr langsam	Kontinuierliches Refactoring nach Performance Guidelines von Google
R3.4	Usability	Software kann nicht benutzt werden	100	5.00%	5	Usability Test zeigen Probleme in der Nutzung der Applikation auf	Frühe und häufige Usability Tests
R3.5	Architektur	Kamerapreview lässt sich nicht gut bearbeiten	200	5.00%	10	Architekturprototypen laufen sehr langsam	Frühe Implementation verschiedener Prototypen steigender Komplexität

12.4.2 Eingetretene Risiken

R2.3 Native Code nötig

Aufgrund der sehr schlechten Performance des Prototyp 1 wurde es nötig eine native C-Bibliothek zu implementieren. Die neuen Risiken “R2.4 Probleme in der Entwicklung mit JNIünd “R2.5 Probleme bei den Bildtransformationsalgorithmen"wurden deshalb identifiziert und aufgenommen.

Dieses Risiko konnte durch die Entwicklung des Prototyp 2 entschärft werden.

R2.6 Probleme bei der Entwicklung des UI

Weil wir die gesamte Applikation im Landscape modus starten, und die Device Orientation selbst handhaben, gestaltete sich das User Interface als ausserordendlich schwierig. Die Applikation muss in den Landscape modus forciert werden, damit die Kameravorschau korrekt funktioniert. Da Android von sich aus sehr schlechte Möglichkeit bietet die Orientation der einzelnen Widgets zu beeinflussen, ohne dass die gesamte Applikation “gedreht"wird, musste mit Rotationen auf dem Canvas gearbeitet werden. Dies erhöhte die Komplexität des User Interface batrachtlich.

12.4.3 Entschärfte Risiken

R2.4 Probleme in der Entwicklung mit JNI

Erwartungsgemäss gestalltete sich die Entwicklung mit JNI im ersten Moment eher schwierig. Nach Einlesen in JNI und gelegentlichem Blick in die Android NDK Beispiele konnte das Entwicklungstempo aber dennoch hoch gehalten werden.

12.5 Prozessmodell

Die Arbeit wurde nach dem Vorbild des Scrum-Projektmanagement[1] organisiert. Wo bei wir als Projektteam uns die Freiheit liesen das Scrumkonstrukt unseren Bedürfnissen anzupassen. Also Dokumentationsvorlage haben wir die Vorlage von Herrn Prof. Dr. Stefan Keller genommen, wobei wir hier gleich verfahren sind wie mit dem Scrumkonstrukt und es unsern Bedürfnissen angepasst haben.

1 http://de.wikipedia.org/wiki/Scrum

12.6 Q-Massnahmen / Definiton of Done

Da wir uns im Prozessmodell ja für Scrum entschieden haben werden sich viele der hier aufgeführten Q-Massnahemen in den “Definition of Done"wieder finden.

Definiton of Done: Ein Feature ist erst dann implementiert wenn alle Punkte auf der Liste abgearbeitet sind|

13 Projektmonitoring

Tabelle 13.1: Dokumenthistory - Projektmonitoring

Rev.	Datum	Wer	Änderung
0.1	30.05.2011	Dominik Spengler	Codestatistik, Protokolle, Code-Reviews erstellt
0.2	30.05.2011	Patrice Müller	Soll-Ist-Zeitvergleich erstellt

13.1 Soll-Ist-Zeitvergleich

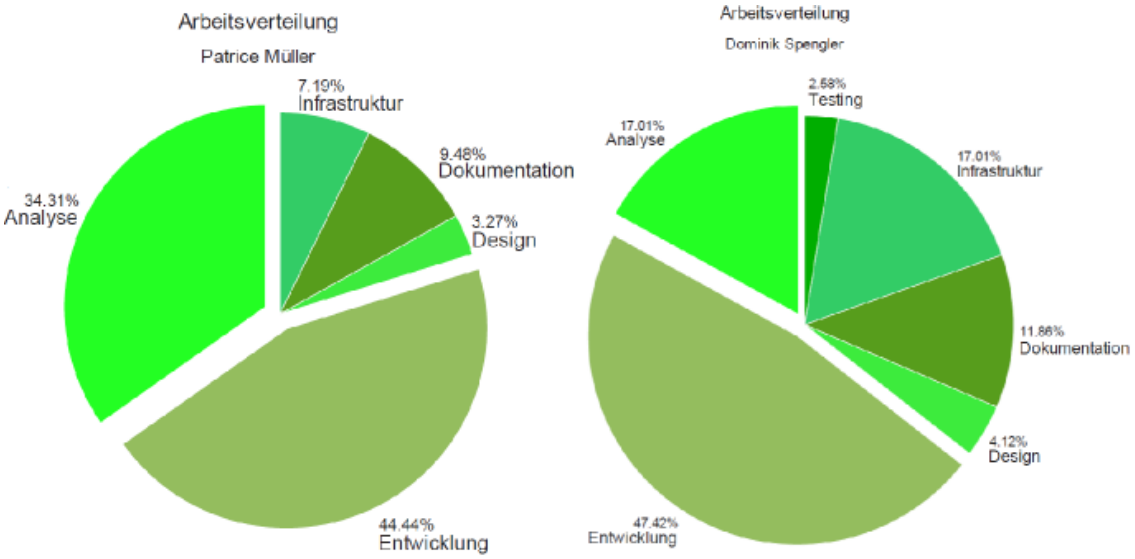


Abbildung 13.1: Arbeitsaufteilung der Teammitglieder

An den obigen zwei Diagrammen sieht man wer für was, wie viel Zeit investiert hat. Die “PersönlichenDiagramme wiederspiegeln wunder schön unseren Fokus in diesem Projekt. Noch deutlich wird das Bild wenn man sich das Diagramm zur allg. Zeitverteilung über das ganze Projekt anschaut.

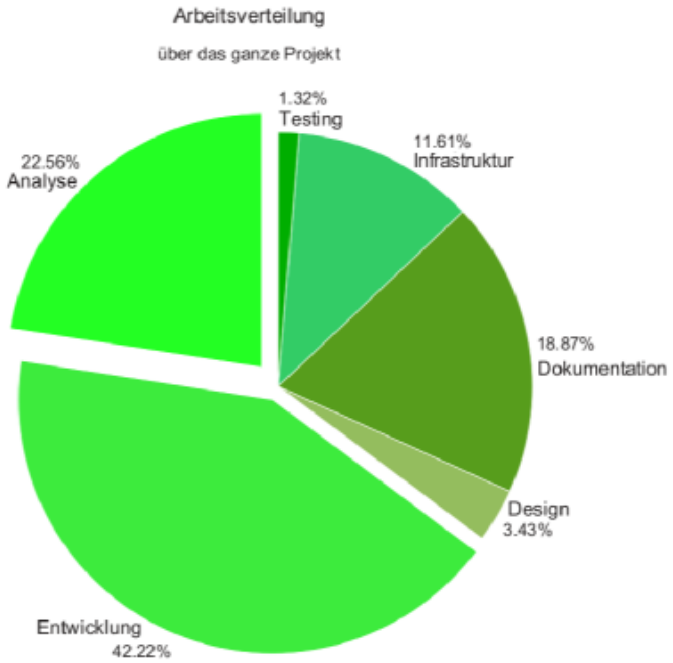


Abbildung 13.2: Arbeitsaufteilung des gesamten Projektes

Hier sieht man gleich das unser Fokus auf der Analyse bzw. Entwicklung lag. Wenn man sich noch mal unser Problemstellung in die Erinnerung ruft– **Eine augmented reality App für Farbenblinde**–wird einem sehr schnell klar, dass der Erfolg unser Applikation mit dem Verständnis für unser Endbenutzer steht oder fällt.

Was wahrscheinlich etwas speziell ist, ist der sehr hohe Entwicklungsanteil der fast bei 50% liegt. Dies ist aber gewollt. Denn wir wollten ja am Ende dieses Projektes zu mindest eine Beta-Version unserer Applikation auf dem Market sehen.

13.2 Codestatistik

Die Codestatistiken wurden mittels Structure101 und dem State Of Flow Eclipse Metrics Plugin bestummen. Es werden nur die nach unserer Meinung relevanten Metriken angegeben. Für eine genauere Übersicht aller ermittelten Metriken ist in der Abgabe die Exportierten HTML Dateien mitgeliefert.

13.2.1 Allgemeine Statistiken

Tabelle 13.2: Allgemeine Codestatistik

Beschreibung	Anzahl
Packages:	4
Classes (outer):	14
Classes (all):	41
NI (Number of bytecode Instructions):	3,651
LOC (Non Comment Non Blank Lines Of Code):	1,349

13.2.2 Cyclomatic Complexity

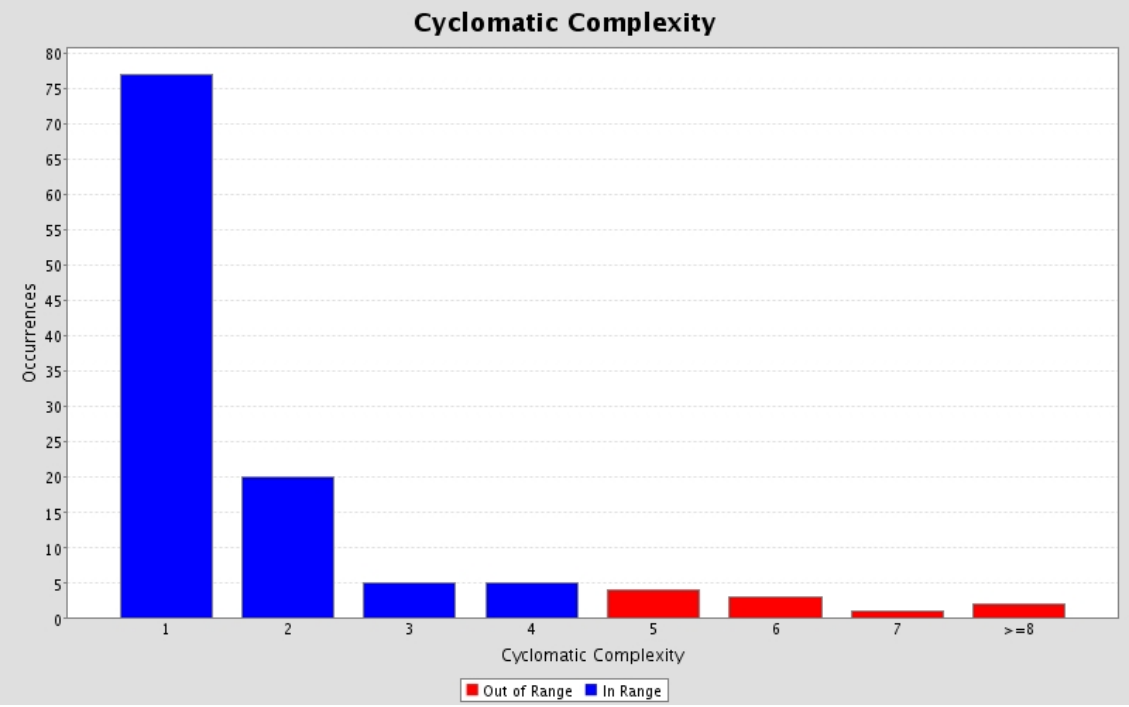


Abbildung 13.3: Cyclomatic Complexity Metrik

Cyclomatic Complexity definiert die Anzahl an “Branchesïn einer Methode, d.h. Anzahl for, while, do, case, catch und des ternary Operator. Methoden der folgenden Klassen unserer Applikation werden dabei als problematisch identifiziert:

- Color
- EyeCamActivity
- Anonyme Innere Klassen von EyeCamActivity
- ControlBar

Dass innerhalb der Color Klasse hohe zyklische Komplexität auftaucht ist nicht überraschend, da sie die Logik der Farberkennung enthält. Diese Logik hat gezwungener massen hohe zyklische Komplexität. Die weiteren Problematischen Methoden sind bis auf EyeCamActivity#getOptimalSize(sizeList) und des Activity Handlers im Zusammenhang mit der Orientation und der SharedPreference. Der Activity Handler und die Interaktion mit der SharedPreference

Instanz sind aufgrund des Designs von Android mit hoher zyklischer Komplexität verbunden. Bei den Orientation Methoden und getOptimalSize könnte man die Komplexität durch Verbesserung des Designs eventuell etwas Verbessern.

13.2.3 Feature Envy

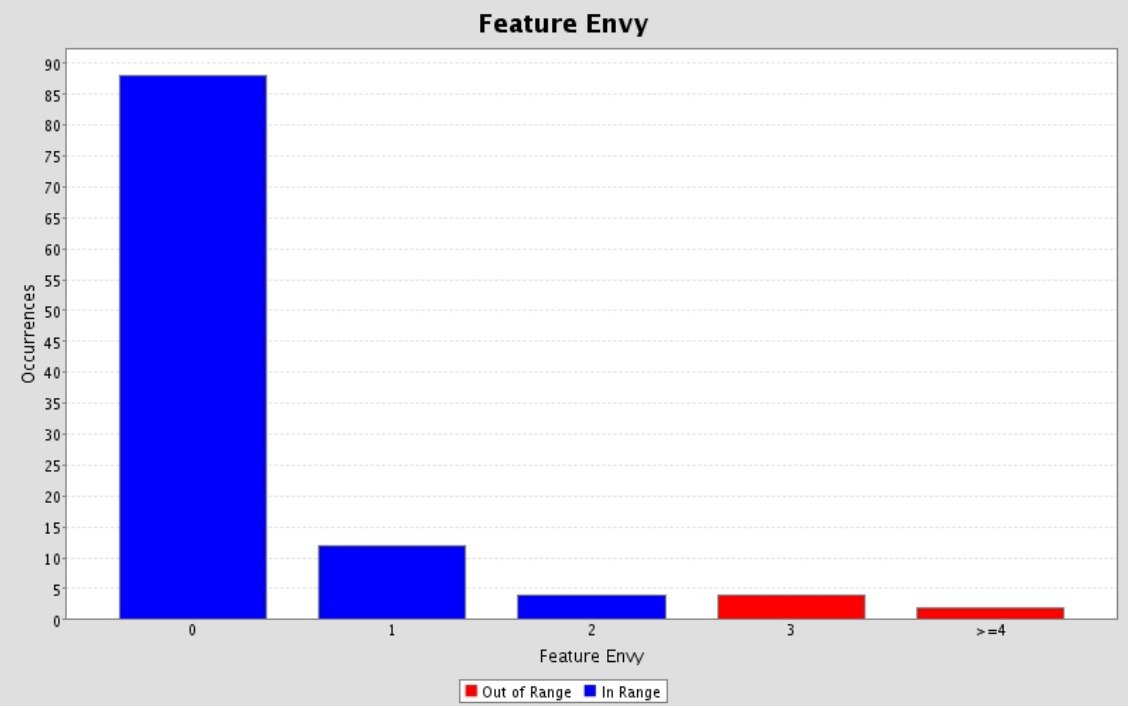


Abbildung 13.4: Feature Envy Metrik

Feature Envy tritt auf, wenn eine Methode grösseres Interesse an Methoden und Members einer anderen Klasse als der eigenen hat. Feature Envy deutet auf schlechte Kohäsion in den Klassen hin und sollte daher vermieden werden. In unserem Fall handelt es sich bei den sechs auftretenden Problemfällen ausschliesslich um anonyme, private, innere Klassen. Um genau zu sein handelt es sich um den Handler und die verschiedenen Listeners. Aus diesem Grund werden die sechs Fälle nicht als Problematisch angesehen.

13.2.4 Efferent Couplings

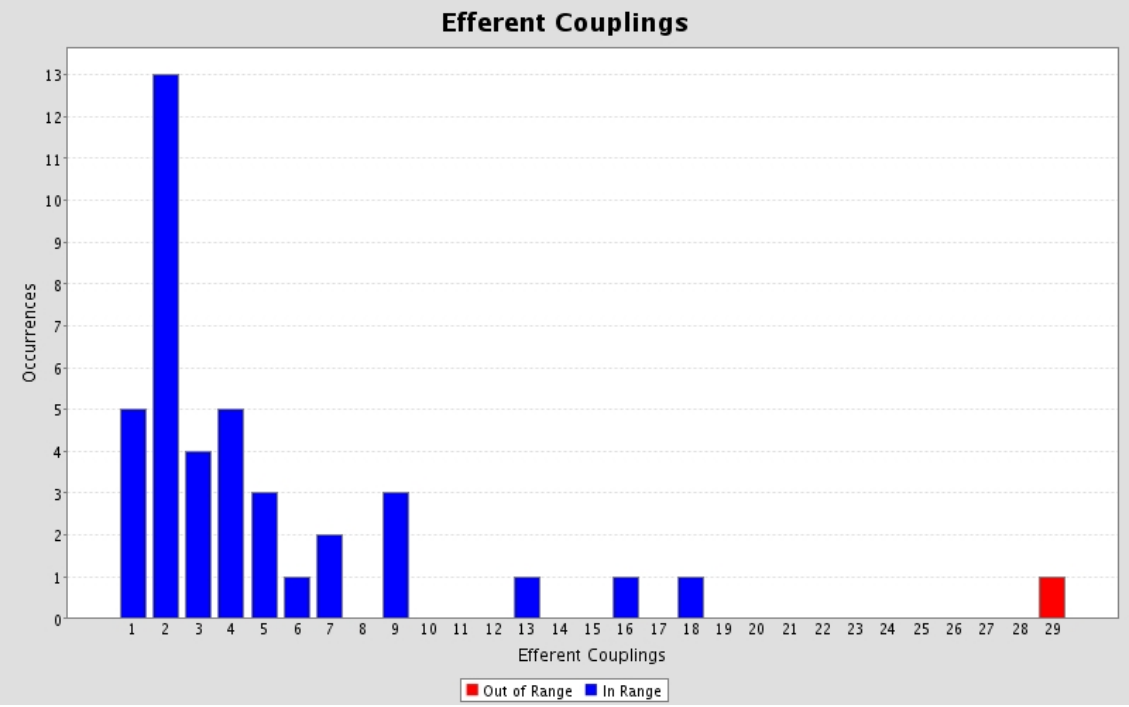


Abbildung 13.5: Efferent Couplings Metrik

Efferent Coupling beschreibt die Anzahl an Typen (Klassen, Interfaces, Exceptions) welche die gemessene Klasse “kennt". Eine hohe Anzahl an fremden Typen deutet auf eine lose Kopplung hin. In unserem Fall handelt es sich bei der Klasse, welche aus der Reihe tanzt um EyeCamActivity. Da es sich um das Model handelt und aufgrund des Android Framework kern der Applikation ist, war dies zu erwarten. Man kann sich aber für später überlegen ob es nicht sinnvoll wäre eine neue Model Klasse zu erstellen und die Activity lediglich für das Management des Lebenszyklus der Applikation zu verwenden.

13.2.5 Lack of Cohesion in Methods

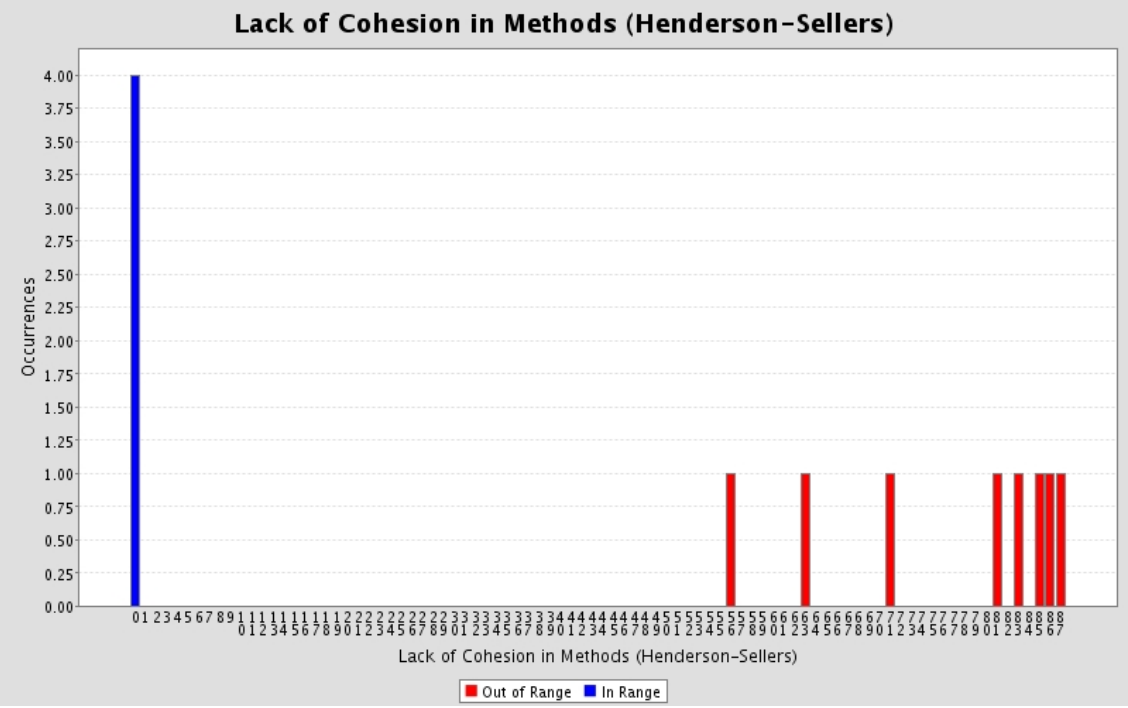


Abbildung 13.6: Lack of Cohesion in Methods Metrik

Für Lack of Cohesion in Methods gibt es verschiedenste Berechnungsmethoden, die zugrunde liegende Idee ist jedoch immer gleich: Es wird die Verwendung der Member Variablen durch die Methoden verglichen. Abgebildet im Bild sieht man die Metrik nach Henderson-Sellers. Generell ist aufgefallen dass unser Code bezüglich Lack of Cohesion in Methods eher schlecht abschneidet. Das heisst, dass häufiger Member Variablen von wenigen Methoden benutzt werden. Das schlechte Abschneiden kann man dadurch erklären, dass wir beim Design und der Implementation primär Fokus auf die logischen Verantwortlichkeiten der Klassen gelegt haben und nicht auf die Kohäsion nach Member Variablen. Hier sieht man dass unser Code doch noch Verbesserungpotential hat.

13.3 Protokolle

Die Sitzungsprotokolle sind im unter **??** dokumentiert.

13.4 Code-Reviews

Es wurde ein Code Review am Montag, 9.5.2011 von Michael Klenk durchgeführt. Es handelte sich dabei um den Code der Version 0.4-alpha. Folgende Anpassungen wurden dabei von Herr Klenk vorgeschlagen:

- EyeCamActivity
  - Methoden isNotNull() und isNull() => sind diese wirklich sinnvoll/nötig
- Orientation
  - Vorberechnete Konstanten verwenden => für die Grenzwerte der Orientierung
  - Orientierung für auf dem Kopf
- StateImageButton
  - Methode isChecked() => gibt lediglich privaten Member zurück, sinnvoller direkte Nutzung des Members
- colortransform.c
  - Duplicated Code => für das Auslesen der U- und V-Werte des Frames
- Unit Testing
  - Die Testabdeckung lässt noch sehr zu wünschen übrig
- Farberkennung verbessern
- Update des Bildes bei Auswahl des Filters im Pause Modus

Die meisten der oben erwähnten Kritikpunkte wurden für die folgende Version angepasst. Jedoch sind die folgenden Punkte ganz bewusst nicht verändert worden:

- **Orientation auf Kopf:** Unserer Meinung nach ist dieses Feature aus Usability Sicht nicht sinnvoll
- **StateImageButton#isChecked():** Es handelt sich dabei um einen Override
- **colortransform.c:** Die Werte der Transformation sind nach wie vor schwer nachzuvollziehen. Es handelt sich um eine Transformation nach ITU-R Recommondation und auch für uns eher schwer nachzuvollziehen. Aufgrund von Performance Überlegungen halten wir auch an den vorgerechneten Werten für die Filter fest.
- **Unit Testing:** Die Code Abdeckung ist nach wie vor eher schlecht. Auf diese Problematik wird in Unit-Tests eingegangen.

14 Softwaredokumentation

Tabelle 14.1: Dokumenthistory - Softwaredokumentation

Rev.	Datum	Wer	Änderung
0.1	26.05.2011	Patrice Müller	Installation erstellt

14.1 Installation

Auf den Folgenden Seiten stehen die Installationsanweisungen für den Anwender wie auch für den Entwickler. Dabei ist zu beachten das in diesem Dokument der Entwickler im Vordergrund steht.

14.1.1 Voraussetzungen für Anwender und Entwickler

Ein Smartphone mit folgende Parameter: |Android|>= 2.2| |Hardware|Smartphone mit Kamera|

14.1.2 Anwender

Für den Anwender ist die Installation absolut Intuitiv, so fern er schon eine mal eine Applikation vom Android-Market installiert hat. Denn genau so funktioniert es! Einfach **eyeCam** im Marketsuchfeld eintippen und auf suchen klicken. Nun sollte eine Applikation mit dem Logo

Abbildung 14.1: eyeCam Applicatoin Icon

auftauchen. Drauf tippen und dann auf *Installieren* tippen und das wars.

14.1.3 Entwickler

Die folgende Anleitung ist für Entwickler die unter Linux entwickeln. Falls Sie ein Entwickler sind die der unter Mac OS X oder Windows entwickelt bitten wir Sie die entsprechenden Seiten zu konsultieren ob es für Sie noch spezielle Konfigurationen gibt. Die entsprechenden Seiten sind jeweils aufgeführt.

Voraussetzungen

Tabelle 14.2: Softwarevoraussetzungen für Entwickler

Software	Version	Anleitung
Java-SDK	>= 1.6	-
Eclipse	>= 3.6.2	-
Android-SDK	>= r11	http://developer.android.com/sdk/installing.html
Android-SDK-Tools	>= r11	-
ADT Plugin für Eclipse	>= 10.0.1	http://developer.android.com/sdk/eclipse-adt.html#installing
Android-NDK	>= r5b	Die Anleitung findet man bisschen weiter unten
Git	>= 1.7.1	-
Ant	>= 1.7	-

Falls Ihr System die Voraussetzung nicht erfüllt, ist es nötig Ihr System nach zu rüsten. Am besten gleich in der Reihenfolge wie sie oben in der Tabelle aufgeführt. Bei Unklarheiten wird Ihnen diese Seite http://developer.android.com/sdk/installing.html weiter helfen, dort steht noch mal alles Schritt für Schritt

Ubuntu 64bit

Beim der 64bit-Version von Ubuntu gibt es noch eine kleine Spezialität man muss noch die ia32-libs nach installieren.

```
sudo apt-get install ia32-libs
```

Git-Repository holen

Das Projekt **eyeCam** finden Sie auf github.com der genau Link lautet https://github.com/tom-and-jerry/eyeCam. Hier können Sie sich gerade über den neusten Stand des Projektes informieren.

Um nun eine eigene Version zu kriegen, brauchen Sie nur das Git-Rpository in Ihr gewünschten Ordner zu klonen.

```
1 git clone git://github.com/tom-and-jerry/eyeCam.git
```

Projekt in Eclipse importieren

Als erste wollen wir vorausschicken das wir Ihnen stärksten empfehlen das Projekt jeweils mit **Ant** zu builden und *nicht mit* **Eclipse**. Da wir u.a. das Buildscript für unser Projekt angepasst haben. **Importieren**

Nun können Sie in Eclipse ein neues Android Projekt erstellen. Nun sollte folgender Dialog auftauchen:

Abbildung 14.2: Eclipse Dialog - neues Android Projekt

- *Projectname* leer lassen
- Auf *Create from existing source* klicken
- *Location* zum geklonten Git-Repository browsen und dort den Root-Ordner auswählen

Jetzt sollte das ganze in etwa so aussehen

Abbildung 14.3: Eclipse Dialog - ausgefülltes Android Projekt

- Auf *Finish* klicken.

Bevor Ihr nun aber anfangen könnt zu Arbeiten muss noch eine letzte Datei erstellt werden, und zwar die **local.properties**. In dieser Datei wird noch dem Buildscript gesagt wo eure Android-SDK liegt. Am einfachsten in den Projekt-Root-Ordner gehen und dann folgenden Befehl eintippen.

```
1 /eyeCam$ echo "sdk.dir=/YOUR␣SDK␣PATH/android-sdk-linux_x86" >> local.properties
```

Am besten Test Ihr eure Konfiguration mit den folgenden Befehlen (das Smartphone sollte angeschlossen sein):

```
/eyeCam$ ndk-build -B
....
/eyeCam$ ant install
....
```

Debugen

Android stellte eine statisch **LOG** Klasse (android.util.Log) zur Verfügung. Welche wirklich super, da man zwischen den Message-Typen **VERBOSE**, **DEBUG**, **INFO**, **WARN**, **ERROR** und **ASSERT** wählen kann. Diese kann man dann auch gleich als Filter in der DDMS-View im Eclipse benützen.

So toll diese Klasse ist, hat sie einen grossen Nachteil, man kann das Loggin nicht über ein Flag oder über ein Ant-Target ausschalten. Dieser Zustand war für uns nicht akzeptabel und so kam der **debug_config** Ordner zu standen. In diesem Ordner befindet sich eine Wraper-Klasse um die android.util.Log Klasse. Die Wraper-Klasse namens **Debug** hat die Aufgabe das Loggin per Flag ein- bzw. ausschaltbar zu machen.

Was heisst das nun für Sie?. Ganz einfach, an jeder Stelle wo Sie es für nötig empfinden eine Debug-Ausgabe zumachen, verwenden Sie anstelle **Log.d(LOG_TAG,“msg“)** die **Mehtode Debug.msg(LOG_TAG,“msg“)**. Wo bei zu bemerken ist das der LOG_TAG der voll qualifizierte Name der Klasse ist, wo Sie den Debugausgebe machen wollen.

Das ganze wird über die Datei **build.properties** gesteuert. Welche sich im Rootverzeichnis des Projektes befindent. Das wichtigste für Sie ist das Attribut **debug.config.logging=false**. Mit diesem Attribut schalten Sie das Loggin an (true) bzw. aus (false) unabhänig vom Buildtarget.

Was zum Abschluss noch zu bemerken ist, das im Moment nur die Debugausgabe in der Klasse Debug implementiert ist. Falls Sie aber finden Sie benötigen noch die anderen Optionen fühlen Sie sich frei diese zu implementieren.

Publishing / Updating

Bevor wir Ihnen gleich erklären wie das das Publishing bzw. Updating auf dem Android-Market geht, wollen wir klar stellen das dieser Teil der Anleitung für Entwickler des **eyeCam** Teams gedacht sind. D.h. diese Teammitglieder haben alle Zugangsdate zu allen Konten des **eyeCam-Projekt**, so wie alle Signierungskey des Projektes.

Aber falls Sie einfach eine kurze Anleitung brauchen wie man seine Applikation für den Market vorbereitet und signiert sind Sie herzlich dazu eingeladen diese Anleitung zu konsultieren.

Falls Sie aber an der genauen Vorgehensweise interessiert sind, finden Sie diese unter diesem Link http://developer.android.com/guide/publishing/publishing.html

Das Publishing / Update auf dem Android-Market ist eigentlich ganz einfach, wenn man mal weiss wie. Hier wollen wir Ihnen das wichtigst in kürze zeigen. Wir setzen folgende Bedienungen voraus

- Sie haben die Zugangsdaten zum eyeCam Entwickleraccount auf dem Android-Market.
- Sie haben die Datei **eyeCam-relaease-key.kestore** in welcher sich der Signierungskey befindet.
- **jarsigner** ist auf Ihrem System installiert
-
- **Das Debugging ist ausgeschaltet!**

Jetzt brauchen es nur noch drei kleine Befehle und das ganze ist fertig. Als erste müssen Sie das Projekt neu Builden und zwar mit dem Target **release**

```
ant release
....
-release-nosign:
     [echo] No key.store and key.alias properties found in build.properties.
     [echo] Please sign YOURPAHT/eyeCam/bin/eyeCam-unsigned.apk manually
     [echo] and run zipalign from the Android SDK tools.
```

Erschrecken Sie nicht ab der letzen Meldung **No key.store and key.alias properties found in build.properties.** die ist ganz normal, wenn man wie wir den Key in einer separaten Datei verwalten. Des Weitern muss man auch die Signierung und Komprimierung der Applikation selber machen, wie das auch oben zu lesen ist.

Aber gehen wir Schritt für Schritt vor. Als nächstes kommt jetzt nämlich das Signieren der Applikation. Wie wir oben gesehen haben hat das Buildscript des Target release keine **eyeCam-unsigned.apk** erstellt. Das ist unser Applikation die wir noch zu signieren haben. Am einfachsten navigieren sie gleich ins **bin-Verzeichnis** des Projektes. Dort angekommen können Sie dann gleich den folgenden Befehl aus führen welcher die Applikation für den Market signiert.

```
#jarsigner [ options ] jar-file aliasInKeyStore
jarsigner -verbose -keystore YOUR_PATH/eyeCam-release-key.keystore eyeCam-unsigned.a
```

So jetzt sind wir nur noch einen Schritt vom Publishing weg. Damit die Applikation möglichst perfomant läuft müssen wir sie mit **zipalign** für das Smartphone optimieren. Dies geschieht wie folgt:

```
#Copyright (C) 2009 The Android Open Source Project
#
#Usage: zipalign [-f] [-v] <align> infile.zip outfile.zip
#       zipalign -c [-v] <align> infile.zip
#
#  <align>: alignment in bytes, e.g. '4' provides 32-bit alignment
#  -c: check alignment only (does not modify file)
#  -f: overwrite existing outfile.zip
#  -v: verbose output

zipalign -v 4 eyeCam-unsigned.apk eyeCam.apk
```

ACHTUNG! Benütz nur die **4** beim Ableichparameter für die Bytes.

Abbildungsverzeichnis

Tabellenverzeichnis

Index