Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

ChristianKirchner

Konzeption und Realisierung einer Anwendung zur mobilen Einsicht der Informationen des Hamburger Volleyball Verbandes

Fakultï¿œt Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

ChristianKirchner

Konzeption und Realisierung einer Anwendung zur mobilen Einsicht der Informationen des Hamburger Volleyball Verbandes

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultï¿œt Technik und Informatik
der Hochschule fï¿œr Angewandte Wissenschaften Hamburg

Betreuender Prï¿œfer: Prof. Dr. Stefan Sarstedt
Zweitgutachter: Prof. Dr. Michael Neitzke

Eingereicht am: 19. Oktober 2012

ChristianKirchner

Thema der Arbeit

Konzeption und Realisierung einer Anwendung zur mobilen Einsicht der Informationen des Hamburger Volleyball Verbandes

Stichworte

Android, MVC Entwurfsmuster, Factory Entwurfsmuster, Fassade Entwurfsmuster, HVBV, Risk-Map, OrmLite

Kurzzusammenfassung

Im Zeitalter der immer rasanter ansteigenden Anzahl von Smartphones und Tablet PC's, bekommt eine für diese Geräte optimierte Darstellung von Informationen eine immer größere Bedeutung. In dieser Arbeit geht es darum eine solche mobile Anwendung für den Hamburger Volleyball Verband zu konzipieren. Durch eine Anforderungsanalyse werden die funktionalen und nicht-funktionalen Anforderungen identifiziert und durch die anschließende Machbarkeitsstudie überprüft. In der darauf folgenden Designphase wird eine passende Software Architektur für die Anwendung entworfen. Abschließend wird in der Realisierungsphase ein Prototyp entwickelt, welcher durch Tests mit ausgewählten Benutzern evaluiert wird.

ChristianKirchner

Title of the paper

Conception and implementation of a mobile application for the information of the Hamburger Volleyball Verband

Keywords

Android, MVC Patter, Factory Pattern, Fassade Pattern, HVBV, Risk-Map, OrmLite

Abstract

The number of users of smartphones and tablets is growing faster and faster making an optimized utilization of applications for this devices ever more important. This thesis will discuss the design of such an application for the Hamburger Volleyball Verband. First, the functional and non-functional demands will be identified by conducting a demand analysis and then validated by a feasibility analysis. In the following design phase, based on the results of the demand analysis, a software architecture for this application will be created. Finally, a prototype will be developed, which will be evaluated by conducting user tests carried out by a number of chosen users.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Einleitung

Schaut man sich im täglichen Leben auf der Straße oder in der U-Bahn einmal um, kommt es einem so vor, als ob heutzutage jeder zweite Mensch ein eigenes Smartphone mit sich führt. Auch wenn laut einer Umfrage von Google im Mai 2012 erst jeder dritte Mensch in Deutschland ein Smartphone besitzt ist diese Zahl im Angesicht der Preise dieser Geräte doch sehr hoch. Das Smartphone ist zu einem Teil unseres Lebens geworden, so dass zwei von drei Smartphonebesitzern nicht ohne es aus dem Haus gehen.

Ausschlaggebend dafür könnten die Möglichkeiten sein die einem das Smartphone heutzutage bietet. Schon lange besitzt es außer dem Empfangen von SMS und E-Mails oder der Telefonie eine Menge weiterer Funktionen. Viel mehr dient das Smartphone zur Orientierung durch Navigationsangebote wie Goolge Maps oder dem Zugriff auf Informationen aus dem Internet oder den sogenannten Apps. Oftmals wird es auch zur Unterhaltung durch Musik oder Videos genutzt.

Betrachtet man die Entwicklung der vergangen Jahre, kann man den Trend zum Smartphone deutlich erkennen. Im 1. Quartal 2011 besaßen ca. 18% der Deutschen ein Smartphone, ein Jahr später hat sich diese Zahl mit 30% fast verdoppelt. Welch einen Stellenwert es im Alltag bereits erreicht hat erkennt man daran, dass 22% der deutschen Bevölkerung lieber auf den Fernseher als auf das Smartphone verzichten würden. Im Gegensatz zu früheren Zeiten, in denen der Fernseher, das Radio oder die Zeitungen für die Versorgung von Informationen aus der Welt zuständig waren, bedienen sich mittlerweile immer mehr Menschen aus dem reichhaltigen Angebot des Internets. Mehr als die Hälfte der Smartphonebesitzer greifen deshalb mindestens einmal am Tag über das Smartphone auf das Internet zu, was zu einem steigenden Angebot der mobilen Dienste führt (vgl. MediaCT (2012)).

Aus diesem Grund werden immer mehr Webseiten im Internet für einen Zugriff über das Smartphone optimiert oder sogar durch eine passende App ergänzt. Bei dem Wort App handelt es sich um die Abkürzung von Applikation, welche oft für kleine Anwendungen auf Smartphones oder Tablets verwendet wird. Ein Beispiel für ein Angebot welches alle diese Darstellungsarten anbietet ist das Fußball Magazin Kicker. Es gibt die normale Internetseite *www.kicker.de* und eine für Smartphones optimierte Seite *www.kicker.mobi*, auf die der Benutzer

umgeleitet wird, sobald er über einen Smartphone Browser auf die normale Internetseite handelt. Außerdem gibt es eine Kicker App, welche sowohl für das Betriebssystem Android, als auch für das von Apple derzeit verwendete Betriebssystem iOS verfügbar ist [1]. Aber auch Amazon, Paypal, Ebay und viele Andere bieten für jedes Gerät eine geeignete Darstellung an. Die mobile Optimierung der Webseiten dient zur Erleichterung der Bedienung über das Touchpad und reduziert die Anzahl der gleichzeitig dargestellten Informationen. Dadurch kann der Nutzer die Informationen genauer nach den eigenen Wünschen selektieren und reduziert seinen Zeitaufwand und vermeidet eine unnötig hohe Belastung des Datenvolumens seiner Internetverbindung.

Der Unterschied zwischen der mobilen Ansicht einer Website und einer für das jeweilige Gerät angepassten App zeichnet sich hauptsächlich durch folgende Aspekte aus:

Vorteile einer mobilen Webseite:

- Die Kosten und der Aufwand für die Entwicklung sind sehr viel geringer.
- Da es sich nur um eine Abwandlung der normalen Webseite handelt, kann es auf jedem Endgerät über den Browser aufgerufen werden.
- Es entsteht kein zusätzlicher Aufwand bei der Verbreitung der mobilen Seite, da die Endgeräte automatisch erkennen ob, die mobile oder die normale Webseite aufgerufen werden soll. Apps hingegen müssen publiziert und beworben werden, damit potentielle Benutzer auf sie aufmerksam werden.

Vorteile einer App:

- Durch das gezielte Einsetzten von Funktionen wie dem Kalender, GPS, Sensoren oder der Kamera können Apps eine größere Funktionsvielfalt ermöglichen.
- Apps können teilweise auch ohne Internetverbindung benutzt werden.
- Oftmals sind Apps durch gezieltes Caching oder andere Optimierungen deutlich performanter als die mobilen Webseiten.

Der Hamburger Volleyball Verband (HVBV) bietet auf seiner Internetseite *www.hvbv.de* [2] einen Ergebnisdienst an. Über diesen können sich Interessenten über anstehende Spieltage oder die Ergebnisse bestimmter Begegnungen von der Jugendliga bis hin zur Verbandsliga informieren. Auch die aktuellen Tabellen der jeweiligen Staffeln können dort eingesehen werden.

[1] Zuletzt aufgerufen am: 25.09.2012
[2] Zuletzt aufgerufen am: 20.09.2012

Für diese Internetseite existiert noch keine mobile Variante, was die Nutzung dieses Services speziell für Smartphones relativ aufwändig und kompliziert macht. Diese Arbeit wird sich damit beschäftigen ein Konzept für eine Umsetzung dieses Services als Android App für geeignete Smartphones und Tablets zu entwerfen. Während der Arbeit soll ein Prototyp dieser App entstehen, welcher die zuvor genannten Vorteile realisiert und dadurch den Benutzern einen angenehmen Umgang mit dem Service des HVBV ermöglicht.

1.1 Zielsetzung

Die Zielsetzung dieser Arbeit besteht aus der Konzeption und Realisierung einer Android App, welche die Informationen des HVBV auf eine übersichtliche, einfache und intuitiv zu handhabende Art und Weise darstellt. Dabei sollen in dem Projekt die Planungs- und Analysephase, die Designphase, die Realisierungsphase und final die Validierungs- und Verifikationsphase durchlaufen und bestmöglich Dokumentiert werden.

Ein besonderer Fokus der Arbeit liegt auf dem Entwurf einer geeigneten Architektur für die gesamte Anwendung, welche eine einfache Integration in das bestehende System des HVBV ermöglicht. Während des Projektes soll ein strukturierter, gut dokumentierter und vor allem nachvollziehbarer Software Entwurf entstehen. Dazu gehört die Verwendung von bekannten Entwurfsmustern und die Strukturierung der Anwendung in unterschiedliche Komponenten. Die wichtigste Aufgabe dabei ist es, die bestehende Architektur nur minimal zu verändern, um die daraus resultierenden Änderungen für bestehende Arbeitsabläufe so gut es geht zu vermeiden. Der zu entwickelnde Prototyp der Anwendung soll eine eingeschränkte Anzahl von Funktionen unterstützt. Um welche Funktionen es sich dabei handeln wird, soll mittels einer Anforderungsanalyse herausgefunden und durch eine Machbarkeitsstudie überprüft werden.

1.2 Persönliche Motivation des Autors

Für meine Motivation dieses Projekte zu realisieren waren im wesentlichen zwei Faktoren verantwortlich. Zuerst das Interesse an der Planung, Konzeption und Realisierung von Software, weshalb ich mich für den Studiengang der Angewandten Informatik entschieden habe. Zum zweiten der Spaß mit dem ich nun seit fast acht Jahren mit meiner Mannschaft am vom HVBV organisierten Spielbetrieb teilnehme. Wenn man als Interessent mit seinem Smartphone mobil auf den vom HVBV bereitgestellten Ergebnisdienst zugreifen möchte, ist der Komfort und die Handhabung der Internetseite sehr eingeschränkt. Deshalb würde ich gerne allen inter-

essierten Smartphone Besitzern eine komfortablere Möglichkeit bieten an die gewünschten Informationen zu gelangen.

1.3 Gliederung der Arbeit

Die Arbeit ist in sechs verschiedene Kapitel aufgeteilt, die teilweise mit den im Abschnitt 1.1 Zielsetzung angesprochenen Projektphasen übereinstimmen.

Das Kapitel 1 ist die *Einleitung*, die sowohl einen Überblick, als auch einen Einstieg in die behandelten Themen geben soll.

Das Kapitel 2 behandelt die *Grundlagen*. Dort werden Konzepte und Begriffe erklärt, die für das Verständnis der Arbeit von Bedeutung sind.

Das Kapitel 3 beinhaltet die *Analyse*, in der es darum geht die grundlegenden Rahmenbedingungen des Projektes zu erfassen und zu bewerten. Mittels einer Anforderungsanalyse wird der Ist-Zustand ermittelt und die Anforderungen an die Anwendung formuliert. Hierbei soll eine Befragung von potentiellen Nutzern der Anwendung behilflich sein. In der darauf folgenden Machbarkeitsstudie werden erste Lösungsansätze entworfen und anschließend auf ihre Risiken hin untersucht. Im abschließenden Fazit werden die Lösungsmöglichkeiten anhand ihrer Risiken und Eigenschaften wie Kosten, Nutzen, Flexibilität oder Robustheit gegeneinander abgewogen.

In Kapitel 4 *Design*, werden mit Hilfe der Ergebnisse aus der Analyse Phase und durch Verwendung von ausgewählten Entwurfsmustern, die sich über die Jahre bewährt haben, Designentscheidungen getroffen und begründet. Unterstützt und verdeutlicht werden diese Entscheidung durch UML Diagramme, die als eine Art Bauplan für die Realisierung dienen.

Das Kapitel 5 beschäftigt sich mit der *Realisierung*. Hier werden die Anforderungen unter Berücksichtigung der Designentscheidungen in einer konkreten Implementation umgesetzt und es entsteht ein erster Prototyp der Anwendung. In der anschließenden Evaluation wird das Ergebnis mit den in der Analyse Phase formulierten Anforderungen verglichen und es wird untersucht, welche der geplanten Anforderungen tatsächlich umgesetzt werden konnten.

In Kapitel 6, welches den *Schluss* der Arbeit darstellt, wird rückblickend noch einmal Bezug auf die am Anfang der Arbeit definierte Zielsetzung genommen. Außerdem werden Pläne und Vorstellungen für mögliche Erweiterungen der Anwendung vorgestellt.

2 Grundlagen

2.1 Android

Da es sich in dieser Arbeit unter anderem um einen Entwurf für das mobile Betriebssystem Android handelt, möchte ich zuerst eine Einführung in die Grundlagen dieses Systems geben. Dabei werde ich kurz die Geschichte von Android erläutern und anschließend genauer auf die Architektur, die Eigenschaften, sowie Vor- und Nachteile von Android eingehen.

Alles hat damit Angefangen, dass Andy Rubin als ehemalige Softwareentwickler von Apple im Jahre 2003 die Firma Andriod gründete, welche sich damals mit der Entwicklung von Software für Mobiltelefone beschäftigte (vgl. Markoff (2007)). Im Sommer 2005 wurde diese Firma dann von dem Großkonzern Google für 50 Millionen USD aufgekauft und von den Google Gründern Larry Page und Sergey Brin als *"best deal ever"* (Kowitt, 2011, Abschnitt 3, Zeile 3,4) bezeichnet.

Im November 2007 gründete die Firma Google dann zusammen mit 33 Partnern die Open Handset Alliance (OHA). Die Mitglieder der OHA bestehen hauptsächlich aus Hardware- und Mobilphoneherstellern (z.B. Acer, HTC, Samsung, NVidia, Intel), Netzbetreibern (z.B. T-Mobil, Vodafone), sowie Softwareherstellern (Google, eBay) aber auch Marketingunternehmen [1]. Das Ziel, dieser aus mittlerweile 84 Firmen bestehenden Gruppe, ist es offene Standards für Mobilgeräte zu schaffen. Als Hauptprodukt daraus entstand das am 22. Oktober 2008 veröffentlichte, erste komplett freie und offene mobile Betriebssystem Android 1.5 (vgl. OHA (2012)).

Laut einem Artikel einer der führenden Research Firmen Gartner liegt der Marktanteil von Android an den verkauften mobilen Betriebssysteme im Jahre 2012 bereits bei 56%, das Betriebssystem iOS von Apple hingegen nur bei 22% (vgl. Gartner (2012)). Eine Studie der Firma Bitcom hat zusätzlich ergeben, dass Android mit 40% die momentan am meisten verwendete mobile Plattform in Deutschland ist. Die Anzahl der Geräte hat sich innerhalb eines Jahres von 2011 bis 2012 mehr als verdoppelt. (vgl. Abbildung 2.1 [2]).

[1] Aktuelle und vollständige Mitgliederliste: `http://www.openhandsetalliance.com/oha_members.html` (zuletzt aufgerufen am: 11.06.2012)

[2] Quelle: `http://www.bitkom.org/de/presse/30739_72316.aspx` (zuletzt aufgerufen am: 13.07.2012)

Die Philosophie von Android ist darauf ausgelegt, dass ein Central Point of Failure (CPoF) so gut es geht vermieden wird. Genau darin sehe ich eine große Stärke dieses Produktes. Dabei wird sichergestellt, dass die Innovationen und Ideen, die zur Verbesserung des Produkts beitragen, nicht durch Restriktionen von Konzernen behindert werden. Durch die Offenheit des Systems und den frei verfügbaren Source Code ist jedem die Möglichkeit gegeben Android an seine speziellen Wünsche anzupassen und somit eventuell das gesamte Produkt für alle zu verbessern (vgl. Google (2012b)).

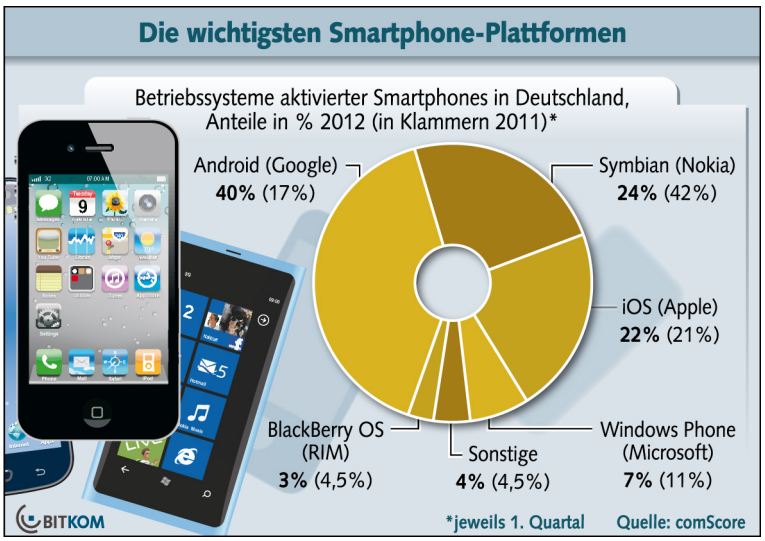


Abbildung 2.1: Die Verteilung der installierten mobilen Betriebssysteme in Deutschland.

2.1.1 Architektur

2.1.1.1 Die Schichten

Die Systemarchitektur von Android (vgl. Becker und Pant (2010), S. 15 ff) ist in vier verschiedene Schichten unterteilt (vgl. Abbildung 2.2). Alle beinhalten mehrere Komponenten und dienen einem gesonderten Zweck.

- *Kernel*
  Android basiert als Betriebssystemgrundlage auf einem Linuxkernel (aktuell 2.6, ab Android 4 auch 3.0), der die Schnittstelle zwischen Software und Hardware darstellt. Von hier aus wird z.B. die Speicherverwaltung, die Energieverwaltung oder die Prozessverwaltung gesteuert.

- *Android-Laufzeitumgebung und Bibliotheken*
  Das Herzstück der Android Laufzeitumgebung ist die Dalvik Virtual Machine (DVM). Die DVM ist eine besonders kleine extra für Android entwickelte Java Virtual Machine (JVM). Auf Grund der geringen Größe kann unter Android jede gestartete App in einer eigenen DVM, mit nur wenig zusätzlichen Ressourcen, laufen. Der Vorteil ist eine enorme Steigerung der Sicherheit und Verfügbarkeit, da unterschiedliche Apps sich niemals einen gemeinsamen Speicher teilen. Dadurch wird beim Absturz eines Prozesses in der DVM maximal eine App beendet. Eine Erweiterung der Android-Laufzeitumgebung bilden die Standardbibliotheken die in C/C++ geschrieben sind. Durch sie werden erforderlichen Funktionalitäten wie Datenbankzugriff, 3D-Grafikbibliotheken, Webzugriff, Multimedia-Verwaltung oder Oberflächenmanagement bereitgestellt. Die Standardbibliotheken sind fester Bestandteil des Systems und können von Anwendungsentwicklern nicht geändert werden.

- *Programmierschnittstelle/Anwendungsrahmen*
  Der Anwendungsrahmen bildet den entscheidenden Teil für die Entwicklung von Android Applikationen. Er beinhaltet alle Basisfunktionen auf die über definierte Schnittstellen aus den Apps heraus zugegriffen werden kann. Hier werden verschiedene Manager-Komponenten zur Verfügung gestellt z.B. zum Erstellen von Benachrichtigungen oder dem Wechseln von Prozessen.

- *Anwendungsschicht*
  Sie beinhaltet alle bereits in Android enthaltenen Anwendungen wie *Contacts* oder den *Dialer*, aber auch die aus dem Android Market heruntergeladenen oder eigenständig entwickelte Apps.

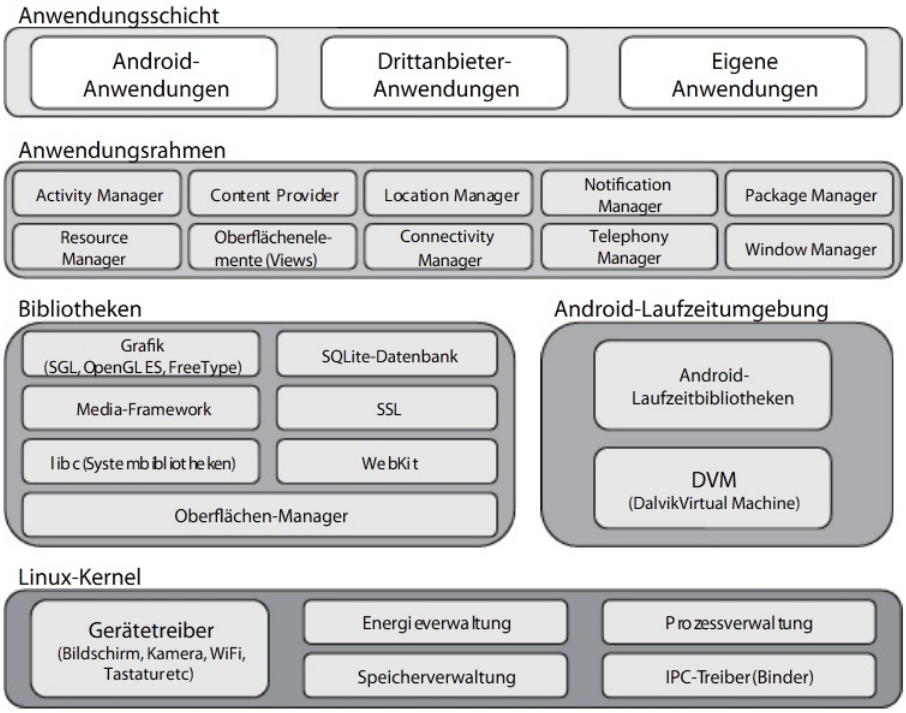


Abbildung 2.2: Android Systemarchitektur [3]

2.1.1.2 Sicherheit

Durch die Verwendung der DVM besitzt Android einen sehr hohen Sicherheitsgrad. Jede gestartete App bildet in ihrer eigenen DVM eine Sandbox und besitzt dadurch weder einen gemeinsamen Speicher mit anderen Apps, noch die Rechte auf Ressourcen des System zuzugreifen. Die für die Applikation benötigten Rechte müssen in ihrer Manifest Datei (vgl. Abschnitt 2.1.2.7) angefordert werden und können ohne die Berechtigung des Benutzers nicht erlangt werden. Möchte die Anwendung also auf das Internet oder die Kontakte zugreifen wird dies bei der Installation gekennzeichnet und der Benutzer kann sich gegen eine Installation entscheiden.

2.1.1.3 Die Context Klasse

Die Klasse *android.content.Context* ist eine Abstrakte Klasse von der die Klassen *Activity* und *Service* abgeleitet sind. Durch die vererbten Methoden von Context können Activities und Services auf die Laufzeitumgebung zugreifen. Über die Context Klasse wird z. B. auf die Ressourcen des Projektes zugegriffen.

[3] Quelle: Becker und Pant (2010)

2.1.1.4 Der Ressource Manager

Der Ressource Manager von Android dient zur Trennung zwischen Applikationslogik und dessen Präsentation. Durch diese Trennung ist man in der Lage seine Applikation an unterschiedliche Sprachen, Bildschirmgrößen und Darstellungsformen anzupassen. Der Ressource Manager verwaltet die Dateien, in denen es sich nicht direkt um Programmcode handelt und ist durch eine Ordnerstruktur aufgebaut. In dem Ordner */MeinProjekt/res* liegen, durch weitere Unterordner aufgeteilt, alle Ressourcen der Applikation. Über die automatisch generierte R.java Klasse ist der Entwickler in der Lage statisch auf die Ressourcen zuzugreifen. Die wichtigsten Ressourcen, die in diesem Projekt verwendet werden, habe ich hier kurz zusammengefasst:

- */res/drawable/*
  Dieser Ordner enthält alle Drawable Ressourcen. Das sind Dateien, welche unter Android auf den Bildschirm gezeichnet werden können. Dabei kann es sich um einfache Bilder handeln, die in Formaten wie z.B. *jpg*, *gif* oder *png* vorliegen. Aber auch Extensible Markup Language (XML) Dateien, mit denen man bestehende Bilder manipuliert oder sogar mit vordefinierten Drawables selber zeichnet [4].

- */res/layout/* In diesem Ordner befinden sich alle Layout Dateien, die unter Android in *xml* geschrieben sind. Diese Dateien enthalten die Struktur für den Aufbau der Präsentation durch die *Activities*. Ein Layout besteht aus einem oder mehreren in einander verschachtelten *Views*. Android bietet von sich aus mehrere verschiedene *Views* an, mit denen man Inhalte darstellen kann. Ein Beispiel dafür wäre der *TextView*, mit dem man einen Text ausgeben kann oder der *Spinner*, mit dem man ein Drop Down Menu realisieren kann. Aber auch ein simples *RelativLayout* mit dem man andere *View* Elemente relativ zu einander ausrichten kann [5].

- */res/values/* Hier werden einfache Werte in XML Dateien gespeichert. Diese Werte können *Colors*, *Strings* oder benötigte *ID's* sein. Über *ID's* greift man in dem Programmcode auf die in den Layouts definierten Views zu. Im Gegensatz zu den anderen Unterordnern, in denen jede Datei eine eigene Ressource war, können hier Listen von Werten hinterlegt werden. Auf diese Werte kann man sowohl aus anderen XML Dateien mit *@filename/value* zugreifen oder auch direkt aus dem Java Programmcode per *R.filename.value*.

[4] Eine vollständige Liste aller Drawable Ressources von Android: (zuletzt aufgerufen am: 13.07.2012)
[5] Ein Tutorial zu den vordefinierten Views: (zuletzt aufgerufen am: 13.07.2012)

2.1.2 Komponenten

Android ist eine moderne Plattform für komponentenbasierte Anwendungen. Das ergibt die Möglichkeit bereits bestehende Komponenten jederzeit wiederzuverwenden oder zu erweitern. Die unter Android zur Verfügung gestellten Anwendungen *Contacts* oder *Dialer* können somit von anderen Anwendungen über Intents (vgl. Abschnitt 2.1.2.5) mitverwendet werden. In den folgenden Abschnitten werde ich erklären, welche Komponenten es gibt und wie diese verwenden werden.

2.1.2.1 Content Provider

Durch Content Provider kann man den Zugriff von anderen Applikationen auf ausgewählte, freigegeben Daten ermöglichen. Dadurch kann man in seiner Applikation gezielt bestimmte Daten oder Dienste für andere Applikationen zur Verfügung stellen.

2.1.2.2 Status Notification

Bei Notifications handelt es sich um kleine Benachrichtigungen, die in der Android Statusleiste platziert werden können. In ihnen können kurze Informationen oder sogar Steuerelemente für die jeweilige Applikation enthalten sein.

2.1.2.3 Service

Ein Service ist der Programmteil der keine Oberfläche benötigt. Services übernehmen Aufgaben wie das Abspielen von Musik im Hintergrund oder die Verwaltung von Download Prozessen.

2.1.2.4 Activity

Eine Activity repräsentiert die sichtbare Bedienoberfläche für eine bestimmte Aktion. In ihr sind *Viewelemente* plaziert, welche in *Viewgroups* zusammengefasst werden können. *Viewelemente* sind zum Beispiel *DropDown Menüs*, *Textausgaben* oder *Buttons*. Über die *Viewelemente* erhält der Benutzer die Möglichkeit Informationen von der Applikation zu erhalten oder Interaktionen durchzuführen. Die Abbildung 2.3 zeigt den Lebenszyklus einer Activity.

- *onCreate()* wird aufgerufen, sobald die Activity das erste mal vom Betriebssystem in den Cache geladen wird. Solange eine Activity im Speicher des Betriebssystems vorhanden ist, wird diese Methode nicht mehr ausgeführt. Sie wird benutzt um eine initiale Konfiguration vorzunehmen, beispielsweise Views zu laden beziehungsweise zu erstellen oder einmalig zum Programmstart bestimmte Methoden auszuführen. Zusätzlich kann in der Methode ein Bundle Object übergeben werden, in welchem man den Zustand einer Activity speichern kann, um ihn beim erneuten Erstellen der Activity wiederherzustellen.
- *onStart()* wird ausgeführt, kurz bevor die *Activity* sichtbar wird, egal ob sie neu erzeugt oder zurück in den Vordergrund geholt wurde.
- *onResume()* aktiviert die Activity und ermöglicht dadurch die Interaktionen mit dem Benutzer.
- *onRestart()* wird immer dann ausgeführt, wenn eine gestoppte Activity erneut aufgerufen wird. Eine Activity gilt als gestoppt, wenn sie für den Anwender nicht mehr sichtbar ist, zum Beispiel durch das Starten einer anderen Activity in den Hintergrund verschoben wurde.

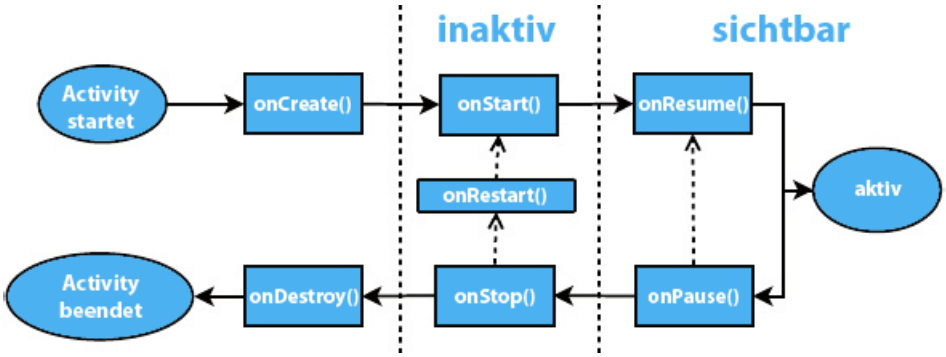


Abbildung 2.3: Lebenszyklus einer Activity

2.1.2.5 Intent

Bei der Entwicklung für Android handelt es sich um komponentenbasierte Anwendungsentwicklung. Da nur dem Betriebssystem alle Komponenten bekannt sind, wird eine Möglichkeit benötigt, zwischen Anwendung und Betriebssystem zu kommunizieren. Dies geschieht in Android über asynchrone Nachrichten, den sogenannten Intents. Über *explicit Intents* können ausgewählte Activities direkt aufgerufen werden. Mit Hilfe von *impliciten Intents* kann ein Anforderungsprofil übergeben werden, wofür das Betriebssystem eine passende Komponente aussucht. Dadurch können die Komponenten über eine sehr lose Kopplung miteinander kommunizieren und auch fremde Komponenten angesprochen werden (vgl. Becker und Pant (2010)).

2.1.2.6 Broadcast Receiver

Broadcast Receiver benötigt man, um auf Intents, die das ganze Betriebssystem betreffen und an alle Komponenten gesendet werden, zu reagieren. Das System kann zum Beispiel ein Intent verschicken, wenn die Kapazität des Akkus unter eine bestimmte Grenze fällt. Ist diese Information für eine Anwendung interessant, so benötigt diese eine Broadcast Receiver, der auf diesen Intent wartet und die gewünschte Reaktion auslöst.

2.1.2.7 Die Manifest Datei

Die Manifest Datei befindet sich in dem Root Ordner eines jeden Android Projektes und enthält wichtige Informationen über die Anwendung. Sie verfügt über Informationen für das Android System, die bereits vor der Ausführung der App bereit stehen müssen. Folgende Informationen sind in der Datei festgehalten: (vgl. Google (2012a):

- Der Packagename der App, welcher als einzigartiger Bezeichner für diese App dient.
- Eine Beschreibung der in der Applikation verwendeten Komponenten (vgl. Abschnitt 2.1.2), einschließlich der Information in welchen Prozessen diese ausgeführt werden.
- Die von der Applikation beanspruchten Zugriffsrechte, beispielsweise für das Internet, das GPS, oder die Kontaktinformationen.
- Die kleinstmögliche Android-API unter dem die Anwendung installiert werden kann.
- Einer Kennzeichnung der Activity, die bei Start der Applikation aufgerufen werden soll.
- Eine Liste der benötigten Libraries, ohne die ein Start der Anwendung nicht möglich ist.

2.2 Objektrelationale Abbildung

Die objektrelationalen Abbildung oder auch Object-Relational Mapping (ORM) (vgl. Abschnitt 2.2) stellt eine Möglichkeit dar, Objekte aus der objektorientierten Programmierung in eine relationale Datenbank zu speichern und daraus abzurufen.

Aufgrund der unterschiedlichen Eigenschaften der objektorientierten Programmierung und dem relationalen Datenbanken wurde Anfang der 1980er Jahre eine Unverträglichkeit dieser beiden Modelle festgestellt, die als *Impedance Mismatch* bezeichnet wird (vgl. Copeland und Maier (1984), S. 316-325).

Die Objektrelationale Abbildung soll dabei helfen, diese Unverträglichkeit zu umgehen und dadurch die Speicherung von Objekten in relationalen Datenbanken zu ermöglichen.In der einfachsten Variante werden dafür die verschiedenen Objekttypen jeweils auf eine eigene Tabelle abgebildet. Jede Instanz eines Objektes entspricht einer Tabellenzeile und für jedes Attribut wird eine Tabellenspalte benötigt. Die Identität der Objekte wird durch den Primärschlüssel der jeweiligen Tabellen realisiert. Besitzt ein Objekt eine Referenz auf ein anderes Objekt, so kann diese mit einer Fremdschlüssel-Primärschlüssel-Beziehung in der Datenbank dargestellt werden.

Eines der bekanntesten und größten ORM-Frameworks ist Hibernate, welches man unter www.hibernate.org [6] auffinden kann. Die Bibliothek ist unter einer LGPL Lizenzfrei verfügbar und kann kostenlos in Projekten verwendet werden.

[6] Zuletzt aufgerufen am: 6.9.2012

3 Analyse

Das Ziel der Analyse besteht darin, zuerst die bestehende Architektur des HVBV zu erfassen und die wesentlichen Anforderungen der Benutzer an die Applikation zu identifizieren. Durch eine Befragung potentieller Nutzer sollen die Prioritäten der einzelnen Funktionen festgelegt werden und gegebenenfalls um, im Vorfeld nicht bedachte Funktionen, ergänzt werden. In der Machbarkeitsstudie werden erste Lösungsansätze vorgestellt, die anhand ihrer Vorteile und Risiken bewertet werden. Das abschließende Fazit wird einen Überblick über die zu erfüllenden Anforderungen und präferierten Lösungsstrategien geben, die im darauf folgenden Kapitel ausschlaggebend für die Designentscheidungen sind.

3.1 Projektplan

Zu Beginn der Analysephase wurde ein Projektplan (vgl. Abbildung 3.1) angefertigt, in den im Laufe des Projektes die geplanten Aktivitäten eingetragen wurden. Mit Hilfe dieses Projektplan sollte die Übersicht über die noch anstehenden und bereits abgeschlossenen Aktivitäten gewährleistet sein. Außerdem ermöglichte er die zeitliche Planung der einzelnen Aktivitäten mit Hilfe eines festgelegten Fertigstellungstermins. In dem Projektplan sind die definierten Meilensteine blau markiert. Wenn immer ein Meilenstein erreicht wurde, fand eine Analyse des Projektes statt, die den Gesamtverlauf des Projektes betrachtet. Dadurch konnten Verzögerungen im Projekt identifiziert werden und die dazu passenden Gegenmaßnahmen eingeleitet werden. In der Planung wurde bewusst ein gewisser Puffer eingebaut, um unerwartete Ereignisse besser ausgleichen zu können.

	Vorgangsname	Anfang	Fertig stellen	Spätester Anfang	Spätestes Ende	Freie Pufferzeit
1	Feststellen des Ist-Zustandes	Di 01.05.12	Sa 05.05.12	Sa 08.09.12	Fr 14.09.12	44 Tage
2	Eigene Anforderungen formulieren	Do 26.04.12	Fr 27.04.12	Sa 21.07.12	Di 24.07.12	0 Tage
3	Umfrage erstellen	Sa 28.04.12	Mo 30.04.12	Di 24.07.12	Do 26.07.12	0 Tage
4	Umfrage durchführen	Di 01.05.12	Di 05.06.12	Do 26.07.12	Fr 31.08.12	0 Tage
5	Umfrage auswerten	Mi 06.06.12	Do 07.06.12	Fr 31.08.12	Mo 03.09.12	0 Tage
6	Umfrage abgeschlossen	Do 07.06.12	Do 07.06.12	Mo 03.09.12	Mo 03.09.12	0 Tage
7	funktionale Anforerungen für das Lastenheft formulieren	Fr 08.06.12	Mo 11.06.12	Mo 03.09.12	Do 06.09.12	3 Tage
8	nicht-funktionale Anforerungen für das Lastenheft formulieren	Di 12.06.12	Do 14.06.12	Mo 03.09.12	Do 06.09.12	0 Tage
9	Lösungsansätze entwickeln	Fr 15.06.12	Di 19.06.12	Do 06.09.12	Di 11.09.12	0 Tage
10	Risikoanalyse	Mi 20.06.12	Fr 22.06.12	Di 11.09.12	Fr 14.09.12	3 Tage
11	Analyse abgeschlossen	Di 26.06.12	Di 26.06.12	Fr 14.09.12	Fr 14.09.12	0 Tage
12	Benötigte Bibliotheken heraussuchen	Mi 27.06.12	Do 28.06.12	Do 20.09.12	Sa 22.09.12	33 Tage
13	Systemüberblick erstellen	Fr 29.06.12	Mo 02.07.12	Mi 19.09.12	Sa 22.09.12	30 Tage
14	neue Datenbank für den Ergebnisdienst entwerfen	Di 03.07.12	Sa 07.07.12	Mo 17.09.12	Sa 22.09.12	25 Tage
15	Views entwerfen	Mo 09.07.12	Mo 16.07.12	Fr 14.09.12	Sa 22.09.12	18 Tage
16	Komponentenstruktur entwerfen	Di 17.07.12	Sa 21.07.12	Mo 17.09.12	Sa 22.09.12	13 Tage
17	Recherche über die zu verwendenden Entwurfsmuster	Mo 23.07.12	Sa 28.07.12	Sa 15.09.12	Sa 22.09.12	7 Tage
18	Design abgeschlossen	Mo 06.08.12	Mo 06.08.12	Sa 22.09.12	Sa 22.09.12	0 Tage
19	Realisierung der Datenbankanbindung	Di 07.08.12	Do 09.08.12	Mo 01.10.12	Do 04.10.12	0 Tage
20	Realisierung der Basic Objekts	Sa 11.08.12	Do 16.08.12	Fr 28.09.12	Do 04.10.12	0 Tage
21	Realisierung der Datenbanksynchronisation	Sa 18.08.12	Sa 25.08.12	Mi 26.09.12	Do 04.10.12	0 Tage
22	Realisierung des MVC Entwurfsmusters	Di 28.08.12	Fr 07.09.12	Sa 22.09.12	Do 04.10.12	0 Tage
23	Realisierung des Factory Entwurfsmusters	Mo 10.09.12	Mi 12.09.12	Mo 01.10.12	Do 04.10.12	0 Tage
24	Realisierung des Fassade Entwurfsmusters	Fr 14.09.12	Mo 17.09.12	Mo 01.10.12	Do 04.10.12	0 Tage
25	Realisierung abgeschlossen	So 30.09.12	So 30.09.12	Do 04.10.12	Do 04.10.12	0 Tage
26	Test der Datenbankanbindung	Fr 10.08.12	Fr 10.08.12	Sa 06.10.12	So 07.10.12	4 Tage
27	Realisierung der Datenbankanbindung abgeschlossen	Mi 15.08.12	Mi 15.08.12	So 07.10.12	So 07.10.12	45 Tage
28	Test der Basic Objects	Fr 17.08.12	Fr 17.08.12	Sa 06.10.12	So 07.10.12	4 Tage
29	Realisierung der Basic Objects abgeschlossen	Mi 22.08.12	Mi 22.08.12	So 07.10.12	So 07.10.12	39 Tage
30	Test der Datenbanksynchronisation	Mo 27.08.12	Mo 27.08.12	Sa 06.10.12	So 07.10.12	2 Tage
31	Realisierung der Datenbank - synchronisation abgeschlossen	Mi 29.08.12	Mi 29.08.12	So 07.10.12	So 07.10.12	33 Tage
32	Test des MVC Entwurfsmusters	Sa 08.09.12	Sa 08.09.12	Sa 06.10.12	So 07.10.12	3 Tage
33	Realisierung des MVC Entwurfsmusters abgeschlossen	Mi 12.09.12	Mi 12.09.12	So 07.10.12	So 07.10.12	21 Tage
34	Test des Factory Entwurfsmusters	Do 13.09.12	Do 13.09.12	Sa 06.10.12	So 07.10.12	5 Tage
35	Realisierung des Factory Entwurfsmusters abgeschlossen	Mi 19.09.12	Mi 19.09.12	So 07.10.12	So 07.10.12	15 Tage
36	Test des Fassade Entwurfsmusters	Di 18.09.12	Di 18.09.12	Sa 06.10.12	So 07.10.12	7 Tage
37	Realisierung des Fassade Entwurfsmusters abgeschlossen	Mi 26.09.12	Mi 26.09.12	So 07.10.12	So 07.10.12	9 Tage
38	Test der gesamten Anwendung	Mo 01.10.12	Mi 03.10.12	Do 04.10.12	So 07.10.12	3 Tage
39	Tests abgeschlossen	Fr 05.10.12	Fr 05.10.12	So 07.10.12	So 07.10.12	1 Tag
40	Projekt fertig gestellt	So 07.10.12	So 07.10.12	So 07.10.12	So 07.10.12	0 Tage

Abbildung 3.1: Projektplan

3.2 Anforderungsanalyse

3.2.1 Feststellen des Ist-Zustandes

Bei der Analyse des Ist-Zustandes ergaben sich bereits die ersten Schwierigkeiten und damit auch die ersten Risiken für das Projekt. Aufgrund der bereits seit einigen Jahren fast unverändert existierenden Architektur befindet sich diese schon lange nicht mehr auf einem aktuellen Stand der Technik. Außerdem gab es beim HVBV niemanden, der über diesen Zeitraum den Überblick über die Arbeitsabläufe und die Funktionalität des Systems behalten hatte.

Durch ein Meeting mit dem für die Internetseite des HVBV verantwortlichen Mitarbeiter und einer Mitarbeiterin, die sich um die Pflege des Ergebnisdienstes kümmert, konnte jedoch ein erster Überblick des Systems erstellt werden. Dabei war es am wichtigsten, das festgestellt wurde, welche Funktionen wo und vor allem wie zu einer Ausführung kommen. Entgegen meiner anfänglichen Vermutung, dass die Internetseite auf eine Datenbank zugreift um die Informationen des Ergebnisdienstes zu erhalten, verhält sich das aktuelle System dort ein wenig anders:

Alle für das Projekt erforderlichen Daten des Ergebnisdienstes liegen gesondert in einer Microsoft Access Datenbank, auf die von außerhalb des Büros leider nicht zugegriffen werden kann. Um also überhaupt für die Benutzer der Applikation relevante Informationen bereitstellen zu können, mussten zuerst die benötigten Daten vom Internet aus zugänglich gemacht werden. In der Abbildung 3.2 ist zu erkennen, dass auf dem Webserver zwar brauchbare Informationen vorhanden sind, diese jedoch nur in Form von PHP Dateien vorliegen. Diese könnte man mittels Parsing durchsuchen, um an die benötigten Informationen zu gelangen, das jedoch dauert sehr lange und führt am Ende zu einer sehr unflexiblen Datenstruktur. Das liegt daran, dass in den PHP Dateien keine Primärschlüssel vorhanden sind, über die Beziehungen zu anderen Informationen hergestellt werden können.

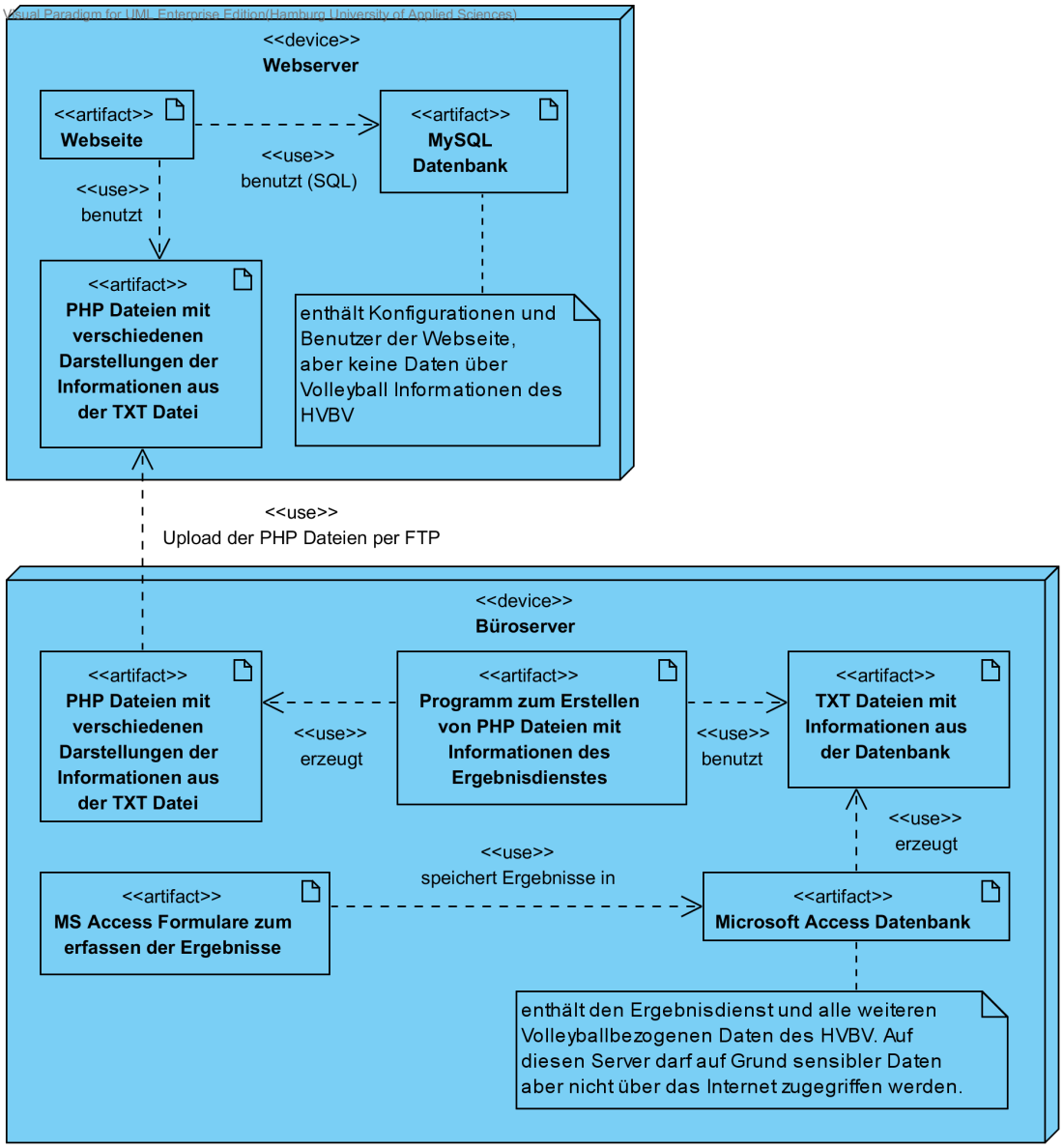


Abbildung 3.2: Verteilungssicht des Ist-Zustandes beim HVBV

Durch die Abbildung 3.3 wird der Arbeitsablauf zum Eintragen von Ergebnissen in das bestehende System des HVBV noch einmal chronologisch dargestellt. In diesem Prozess, müssen die Mitarbeiter des HVBV, zusätzlich zum eigentlichen Eintragen der Ergebnisse, noch viele weitere Aktionen durchführen.

Erst nach drei weiteren Schritten befinden sich die Informationen auf dem Webserver und stehen für die Internetseite zur Verfügung. Eigentlich könnten jedoch alle Aktionen von 2 bis 6 aus der Abbildung 3.3 automatisiert oder ersetzt werden. Jedoch wurde in den Gesprächen mit den betroffenen Person deutlich klar, dass Sie weiterhin gerne ihren gewohnten Arbeitsprozess ausführen wollen.

Bei dem Entwurf soll darauf geachtet werden, dass der bereits bestehende Prozess auch weiterhin verwendet werden kann. Um diese Voraussetzung zu erfüllen, könnte parallel zu dem bestehenden eine weitere Datenbank installiert werden. Diese müsste dann bei Änderungen in der alten Datenbank, automatisch mit den jeweils neuen Informationen erweitert werden. Oder die Informationen werden durch eine zusätzliche Anwendung direkt in die neue Datenbank geschrieben.

So kann weiterhin der alte Prozess aus Abbildung 3.3 benutzt werden und die Informationen stehen trotzdem in der neuen Datenbank zur Verfügung. Außerdem hätte die Internetseite so die Möglichkeit, weiter auf die PHP Dateien oder zusammen mit der App, auf die neue Datenbank als Informationsquelle zugreifen. Der Wunsch der Mitarbeiter wäre erfüllt und der alte Prozess könnte weiterhin zum Einfügen von neuer Informationen verwendet werden.

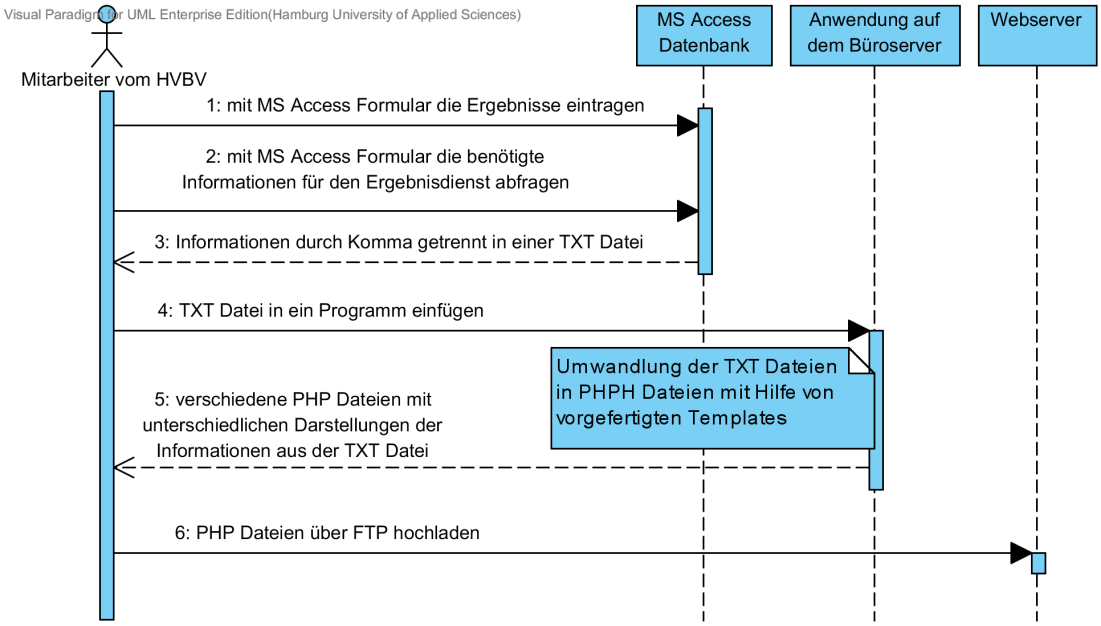


Abbildung 3.3: Sequenzdiagramm für das Eintragen von Ergebnissen

3.2.2 Funktionale Anforderungen

Die funktionalen Anforderungen an die Anwendung beschreiben die Funktionalität und das Verhalten der Anwendung. Normalerweise definiert der Auftraggeber in einem Lastenheft die Anforderungen, die er an die zu entwickelnde Software hat. Anschließend werden Unverständlichkeiten der Anforderungen beseitigt und die Anforderungen werden auf ihre Machbarkeit hin überprüft. Als Ergebnis entsteht das Pflichtenheft, in dem beschrieben wird, welche Anforderungen wie realisiert werden können.

Mit Hilfe der Ergebnisse der Analyse des Ist-Zustandes, konnten zwei verschiedene Kategorien von Anforderungen an die Anwendung identifiziert werden. Da es in diesem Projekt nicht einen konkreten Auftraggeber gibt, sondern zwei verschiedene Interessengruppen, möchte ich diese kurz benennen und die von ihnen gestellten Anforderungen erläutern.

Die erste Kategorie beinhaltet die Anforderungen an die Anwendung, welche hauptsächlich durch die potentiellen Benutzer gestellt werden. Komplikationen bei der Umsetzung dieser Anforderungen wirken sich im wesentlichen auf die Funktionalität der App aus.

Die zweite Kategorie betrifft die Anforderungen, die an die Bereitstellung der erforderlichen Informationen und die dafür benötigten Anpassung des bestehenden Systems des HVBV gerichtet sind (vgl. Abschnitt 3.2.1).

Bei diesen Anpassungen können Komplikationen nicht nur Auswirkungen auf die Anwendung haben, sondern auch negativen Einfluss auf das operative Geschäft des HVBV. Deshalb werden ihre Prioritäten wesentlich höher eingestuft, was dazu führt, dass mehr Ressourcen für diese Anforderungen aufgebracht werden. Außerdem werden Auswirkungen der Risiken, die diese Anforderungen betreffen als sehr viel gefährlicher eingestuft.

3.2.2.1 Anforderungen an die Applikation

Um die funktionalen Anforderungen zu bestimmten wurden zuerst die unterschiedlichen Nutzergruppen identifiziert. Es ergaben sich drei unterschiedliche Gruppen mit verschiedenen Anforderungen.

- Benutzer, deren Interesse sich auf eine bestimmte Mannschaft beschränkt,beispielsweise die Mutter eines Spielers, die gerne beobachten möchte wie ihr Kind gespielt hat.
- Die Spieler selbst, die nicht nur den Verlauf ihrer eigene Mannschaft verfolgen möchten, sondern auch das Geschehen in ihrer Staffel oder Liga.

- Benutzer, die auf die dargestellten Informationen einwirken wollen, indem sie beispielsweise Ergebnisse eintragen oder Spieltage verschieben. Für dieses Vorhaben würden jedoch nicht nur Leserechte, sondern auch Schreibrechte auf der Datenbank vom HVBV benötigt. Dadurch steigert sich die Wahrscheinlichkeit Fehler oder Sicherheitslücken zu generieren, die im schlimmsten Falle zu Änderungen oder Verlusten von Daten führen könnten. Daher wird sich dieses Projekt vorerst mit den Funktionen beschäftigen für die ein Lesender Zugriff ausreicht.

Aus den Anforderungen dieser drei Gruppen lassen sich bereits mehrere verschiedene Funktionen ableiten:

- *Wiederkehrender Zugriff auf ausgewählte Mannschaften*
  Voraussichtlich wird der Verlauf einer bestimmten, oftmals wohl der eigenen, Mannschaft öfter verfolgt als den Verlauf anderer Mannschaften. Deshalb würde sich eine Übersicht von favorisierten Mannschaften des Benutzers anbieten. Dafür muss es eine Möglichkeit geben Mannschaften als Favorit zu markieren und zu löschen.

- *Selektiver Zugriff auf ausgewählte Mannschaften*
  Es soll trotzdem die Möglichkeit bestehen Informationen über jede beliebige Mannschaft abzurufen. Dafür wird eine Suchfunktion benötigt über die man gezielt eine Mannschaft aus dem Gesamtbestand der Mannschaften auswählen kann.

- *Es muss festgelegt werden, welche Informationen angezeigt werden*
  Sowohl die Tabelle einer Staffel als auch der Spielplan mit den Spieltagen für die Mannschaft soll anzeigt werden.

- *Es müssen diejenigen Funktionen von Android identifiziert werden, die nützliche funktionale Anforderungen unterstützen könnten*
  Durch eine Anbindung an die Google Maps App wäre es möglich eine Navigation zu dem Austragungsort bevorstehenden Spieltagen zu realisieren. Durch einen Button in der Übersicht oder in der Ansicht der Spieltage öffnet sich direkt eine Routenplanung von dem aktuellen Standpunkt bis zum Austragungsort. Dadurch könnte sich der Benutzer die Suche nach der richtigen Adresse und den damit verbundenen Kopiervorgang in die Navigationsanwendung sparen. Außerdem wäre eine Synchronisation der Spieltage mit einem registrierten Google Kalender möglich. Damit könnten durch einen Knopfdruck alle bevorstehenden Spieltag in den Kalender einzutragen werden.

3.2.2.2 Umfrage

Durch eine eigenständig durchgeführte Umfrage wurden weitere Informationen über die Anforderungen von potentiellen Benutzer der Applikation gesammelt. Dadurch wurde versucht die mit der Anwendung verbundenen Prioritäten der Benutzer bei der Konzeption bestmöglich zu berücksichtigen. Zusätzlich ergab sich so die Möglichkeit weitere funktionale und nicht-funktionale Anforderungen der Benutzer zu erkennen. Der Kreis der Befragten Personen war bewusst auf diejenigen beschränkt, die später auch Interesse an der Applikation haben könnten und bestand zum größten Teil aus aktiven Volleyballern. Durch gezielte Fragen habe ich versucht Projektrisiken zu eliminieren oder bestmöglich zu reduzieren.

Mit Hilfe der ersten Frage wollte ich herausfinden wie die Verteilung der Smartphone Betriebssysteme im potentiellen Nutzerkreis aussieht. Besonders wichtig dabei ist, welche Version des Android Betriebsystems vorliegt. Diese hat direkte Auswirkung auf die Software Development Kit (SDK) Version von Android und dadurch auch die von Android bereitgestellten Funktionen.

1. Falls du ein Android Handy besitzt, welches Betriebssystem ist bei dir installiert?

	Beantwortung in Prozent	Anzahl Beantwortungen
ich besitze kein Android Handy	**61,5%**	**16**
ich habe ein Android Handy aber keine Ahnung welches Betriebssystem	7,7%	2
älter als 2.2.x Froyo	3,8%	1
2.2.x Froyo	0,0%	0
2.3.x Gingerbread	15,4%	4
3.x Honeycomb	0,0%	0
4.x Ice Cream Sandwich	11,5%	3
	beantwortete Frage	26

Abbildung 3.4: Benutzerumfrage: Frage 1

Da von allen Personen die ein Android Betriebssystem nutzen nur eine Person eine Version besitzt die älter ist als die Version 2.2, werde ich diese als minimale Voraussetzung für die geplante Applikation benutzten. Ausschlaggebend dafür sind einige wichtige Features die

erst mit der Version 2.2 eingeführt wurden. Das wichtigste dabei ist, dass die Applikation wahlweise auf dem externen Speicher des Gerätes installiert werden kann.

Die zweite Frage sollte einen Überblick über die Prioritäten der funktionalen Anforderungen an die Applikation aus Sicht der potentiellen Benutzer bekommen.

2. Für wie wichtig hälst du die folgenden Funktionen der Application?

	unwichtig				sehr wichtig	Bewertungs-Mittelwert	Anzahl Beantwortungen
Tabelle anzeigen	0,0% (0)	0,0% (0)	0,0% (0)	11,5% (3)	**88,5% (23)**	4,88	26
Spielplan anzeigen	0,0% (0)	0,0% (0)	0,0% (0)	11,5% (3)	**88,5% (23)**	4,88	26
Auflistung aller Mannschaften eines Vereins	0,0% (0)	20,0% (5)	**40,0% (10)**	28,0% (7)	12,0% (3)	3,32	25
Ansicht von den News des HVBV	8,7% (2)	**43,5% (10)**	39,1% (9)	8,7% (2)	0,0% (0)	2,48	23
Favoriten einrichten um ausgewählte Mannschaften schneller zu erreichen	0,0% (0)	15,4% (4)	19,2% (5)	15,4% (4)	**50,0% (13)**	4,00	26
Google Maps Navigation zu den Spieltagen	0,0% (0)	15,4% (4)	26,9% (7)	**38,5% (10)**	19,2% (5)	3,62	26
Warnungen bei Spielplanänderungen	0,0% (0)	0,0% (0)	3,8% (1)	30,8% (8)	**65,4% (17)**	4,62	26
					beantwortete Frage		**26**

Abbildung 3.5: Benutzerumfrage: Frage 2

Das Ergebnis zeigt recht deutlich, dass viele der Funktionen die vor der Umfrage definiert wurden auf ein recht hohes Interesse gestoßen sind. Außerdem kann man auf Grund der Einschätzungen die Tabelle, den Spielplan, die Warnungen bei Spielplanänderungen und die Favoritenliste als besonders erwünschte Kernfunktionen der Anwendung identifizieren. Auch die Google Maps Navigation zu den Spieltagen ergab eine recht große Nachfrage.

Die vierte Frage sollte dem Teilnehmerkreis die Möglichkeit geben eigene Vorschläge für weitere Funktionen zu äußern. Da hier jedoch nur sehr wenig Rückmeldung gekommen ist gehe ich durch die Kombination der Ergebnisse der beiden Fragen davon aus, das die wesentlichen Wünsche der Benutzer erkannt wurden.

In der dritten Frage ging es darum herauszufinden, welche der nicht funktionalen Anforderungen, die direkten Einfluss auf den Benutzer haben, besonders wichtig erscheinen.

3. Wie wichtig sind dir folgende Eigenschaften?

	unwichtig				sehr wichtig	Bewertungs-Mittelwert	Anzahl Beantwortungen
Die Application ist nicht mit Informationen überladen	0,0% (0)	3,8% (1)	23,1% (6)	**38,5% (10)**	34,6% (9)	4,04	26
Die Fülle an Funktionen der Application ist noch überschaubar	0,0% (0)	3,8% (1)	34,6% (9)	**38,5% (10)**	23,1% (6)	3,81	26
Die Application ist grafisch anspruchsvoll gestaltet	3,8% (1)	23,1% (6)	**42,3% (11)**	23,1% (6)	7,7% (2)	3,08	26
Die Application besitzt kurze Reaktionszeiten	0,0% (0)	0,0% (0)	3,8% (1)	46,2% (12)	**50,0% (13)**	4,46	26
					beantwortete Frage		**26**

Abbildung 3.6: Benutzerumfrage: Frage 3

Dabei hat sich die Reaktionszeit der Anwendung als wichtigstes Kriterium für die Benutzer herausgestellt. Auch die Bedienbarkeit und die Überschaubarkeit der Funktionen spielen eine Rolle, wobei es jedoch nicht so wichtig ist wie aufwendig das Design gestaltet ist.

Aus der fünften Frage lässt sich ableiten, dass zumindest in der Zielgruppe der Sportler die sich aktiv mit Volleyball befassen ein recht großes Interesse an einer solchen Anwendung besteht. Darüber hinaus könnte sich durch Familien und Bekannte, die sich so über den Verlauf des Sportlers informieren möchten, die Anzahl der Interessenten weiter erhöhen.

5. Hättest du Interesse an einer solchen Application?

	Beantwortung in Prozent	Anzahl Beantwortungen
ja	**92,3%**	**24**
nein	7,7%	2
	beantwortete Frage	**26**

Abbildung 3.7: Benutzerumfrage: Frage 5

Das die Mehrheit der Befragten sogar bereit wäre Geld für eine solche Anwendung zu bezahlen, bestärkt noch einmal das Ergebnis von Frage fünf und reduziert somit auch das Risiko eine Applikation ohne notwendiges Benutzerinteresse erstellt zu haben.

6. Wärst du bereit einen geringen Preis für die Entwicklung und die spätere Weiterentwicklung zu zahlen?

	Beantwortung in Prozent	Anzahl Beantwortungen
nein	**34,6%**	**9**
bis zu 50 Cent	11,5%	3
bis zu 1 Euro	**34,6%**	**9**
bis zu 2 Euro	15,4%	4
bis zu 3 Euro	0,0%	0
bis zu 4 Euro	0,0%	0
mehr als 4 Euro	3,8%	1
	beantwortete Frage	**26**

Abbildung 3.8: Benutzerumfrage: Frage 6

Aus den selbst überlegten Anforderungen ergaben sich, in Kombination mit den Umfrageergebnissen, folgende funktionale Anforderungen für das Lastenheft. Dabei bestimmt die Priorität der einzelnen Anforderungen, wie intensiv diese Anforderungen geplant und zu welchem Zeitpunkt im Projekt sie umgesetzt werden sollen. Zur Bestimmung der Prioritäten werden die Ausprägungen *unverzichtbar*, *sehr wichtig*, *wichtig*, *nicht wichtig* und *nebensächlich* in dieser absteigenden Reihenfolge verwendet werden. Eine Anforderung mit der Priorität *unverzichtbar* wird in diesem Projekt dem entsprechend besonders stark berücksichtigt und es wird versucht eine funktionsfähige erste Version am Ende der Arbeit sicherzustellen.

Nummer	Beschreibung	Priorität
FA1	Der Benutzer kann sich den Tabellenstand ausgewählter Mannschaften anzeigen lassen. Folgende Informationen werden angezeigt: • Platzierung der Mannschaft • Abkürzung des Namens der Mannschaft • Anzahl der gewonnen Punkte • Anzahl der abgegebenen Punkte • Verhältnis von gewonnenen zu fehlenden Punkten • Anzahl der gewonnenen bzw. verlorenen Sätze • Differenz von gewonnenen zu verlorenen Sätzen	sehr wichtig
FA2	Der Benutzer kann sich eine Liste der Spieltage ausgewählter Mannschaften anzeigen lassen. Ein Spieltag besteht aus folgenden Informationen: • Datum des Spieltages • Uhrzeit des Anpfiffs • Straßenname der Halle in der gespielt wird • jeweilige Paarungen aus Mannschaft, Gegner und Schiedsgericht • sofern vorhanden dem Ergebnis des Spiels.	sehr wichtig
FA3	Der Benutzer kann ausgewählte Mannschaften zu seinen Favoriten hinzufügen und erhält so die Möglichkeit von der Startseite aus direkt auf diese Mannschaften zuzugreifen.	sehr wichtig
FA4	Der Benutzer kann Mannschaften wieder aus seinen Favoriten entfernen.	sehr wichtig
FA5	Der Benutzer kann eine Google Maps Navigation zu ausgewählten Spieltagen öffnen.	wichtig
FA6	Der Benutzer kann Spieltage von Mannschaften mit seinem Google Kalender synchronisieren.	nicht wichtig
FA7	Auf jeder Seite mit Informationen des HVBV muss es die Möglichkeit geben die Inhalte zu aktualisieren	wichtig
FA8	Der Benutzer kann einstellen, dass nur bei einer aktiven WLAN-Verbindung auf das Internet zugegriffen wird.	nicht wichtig

Tabelle 3.1: Lastenheft - funktionale Anforderungen Teil 1

3.2.2.3 Anforderungen an die Anpassung des bestehenden Systems

Die nachfolgenden Anforderungen sind aus mehreren Gesprächen mit ausgewählten Mitarbeitern des HVBV entstanden. Dabei waren, mit dem Verantwortlichen für die Internetseite des HVBV und mir, zwei Personen mit gesteigertem Interesse an einer Anpassung des bestehenden Systems anwesend. Unterstützt wurden wir von einer Mitarbeiterin, welche in die von den Änderungen betroffenen Geschäftsprozesse des HVBV eingebunden ist. Ziel der Gespräche war es, eine für alle Parteien optimale Lösung zu finden. Als Ergebnis der gemeinsamen Gespräche wurden folgende Anforderungen an die Anpassung des bestehenden Systems festgelegt:

Nummer	Beschreibung	Priorität
FA9	Die Informationen für den Ergebnisdienst aus der Datenbank des HVBV müssen über das Internet zugänglich gemacht werden.	unverzichtbar
FA10	Die Arbeitsabläufe der Mitarbeiter des HVBV werden nicht verändert. Aktuell verwendete Formularmasken und Software bleibt bestehen und kann weiterhin verwendet werden.	unverzichtbar
FA11	Das bestehende System wird in seiner Funktionsweise erweitert. Die Erweiterungen setzten auf dem bestehenden System auf und können jeder Zeit rückgängig gemacht werden.	unverzichtbar
FA12	Durch die Änderungen kommen ausschließlich einfache Aufgaben wie das Ausführen einer Batch Datei hinzu, die zusätzlich zu den bestehenden Arbeitsabläufen ausgeführt werden muss.	sehr wichtig

Tabelle 3.2: Lastenheft - funktionale Anforderungen Teil 2

3.2.3 Nicht-Funktionale Anforderungen

Bei nicht funktionalen Anforderungen handelt es sich um Anforderungen, die nicht durch die Funktionalität der Anwendung bestimmt werden, sondern ihre Qualitätseigenschaften beschreiben. Da die nicht-funktionalen Anforderungen nicht an eine bestimmte Anwendung gekoppelt sind, gibt es die *DIN/ISO 9126 Norm* in der die wesentlichen Anforderungen festgehalten sind (vgl. Starke (2009) S. 60 ff).

Funktionalität

- *Angemessenheit und Richtigkeit*
  Angemessenheit und Richtigkeit sind zwei wesentliche Merkmale, wenn es um Software Qualität geht, da sie abhängig von der Anwendung unterschiedlich hohe Auswirkungen haben können. In dieser Applikation jedoch, geht es ausschließlich darum Informationen die vom HVBV bereitgestellt werden anzuzeigen. Dabei handelt es sich nicht um sensible Daten oder Daten, die bei einer fehlerhaften Darstellung zu finanziellen Auswirkungen führen könnten. Trotzdem soll sichergestellt werden, dass das System angemessen und richtig funktioniert, da diese Merkmale den Kunden besonders negativ auffallen könnten.

- *Interoperabilität*
  Eine gute Kooperation mit Fremdsystemen ist sehr wichtig, da die Anwendung in ein bestehendes System integriert werden soll.

- *Ordnungsmäßigkeit*
  Bei der Applikation handelt es sich um einen Angebot seitens des HVBV, bei der keinerlei Verpflichtungen zwischen den beteiligten Parteien eingegangen werden. Außerdem werden keine Daten benutzt oder gespeichert, für die besondere rechtlichen Bedingungen eingehalten werden müssen. Bei der Entwicklung jedoch muss darauf geachtet werden, dass keine rechtlich geschützten Erzeugnisse verwendet werden. Da die Applikation für jeden frei verfügbar sein soll, muss sichergestellt werden, dass die Informationen in der Applikation nicht gegen Gesetzte, wie beispielsweise das Jugendschutzgesetz, verstoßen.

- *Sicherheit*
  Da die in der Anwendung verendeten Daten keine besonderen Sicherheitsmaßnahmen verlangen, ist die Sicherheit der Anwendung nicht die oberste Priorität. Allerdings muss lediglich sichergestellt werden, dass durch die Anwendung keine Sicherheitslücke im bestehenden System des HVBV entsteht.

Zuverlässigkeit

- *Reife*
  Die Reife der Software spielt eine wesentliche Rolle. Nur eine fehlerfrei funktionierende Applikation wird die Benutzer dauerhaft motivieren diese App zu benutzen. Deshalb ist bei der Realisierung der Anforderungen ein hoher Reifegrad sicherzustellen.

- *Fehlertoleranz*
  Auch die Fehlertoleranz sollte hoch genug sein, um bei einem Ausfall der Datenbank oder dem Internet noch einen Teil der Leistung zu garantieren. Es muss daher möglich sein, zumindest den aktuellen Stand der Anwendung zu dem Zeitpunkt des Ausfalls korrekt anzuzeigen. Es sollten die Funktionen verfügbar sein, für die keine der Ausgefallenen Komponenten benötigt werden.

Benutzbarkeit

- *Verständlichkeit und Erlernbarkeit*
  Da es sich bei der Anwendung um eine interaktive Applikation handelt, sind diese Punkte besonders wichtig. Der Benutzer sollte ohne weitere Konfiguration in der Lage sein die Anwendung zu starten, nachdem er sie aus dem App-Store heruntergeladen hat. Danach muss er die Funktionen intuitiv verstehen oder erlernen können, ohne dass er dafür eine Anleitung benötigt.

- *Bedienbarkeit*
  Durch den Touchscreen eines Smartphones gibt es viel Möglichkeiten die Bedienung so einfach und intuitiv wie möglich zu gestalten. Allerdings erschwert das recht kleine Display die Strukturierung und Darstellung der abzubildenden Informationen und Funktionen. Die Bedienbarkeit ist bei solchen Apps ein wichtiges Kriterium und soll besonders beachtet werden.

Effizienz

- *Zeitverhalten*
  Da der Benutzer in eine ständigen Interaktion mit mit der Anwendung ist, sind starke Verzögerungen oder Wartezeiten zu vermeiden. Zusätzlich ist die Bedienung über ein Touchpad bei starken Verzögerungen deutlich erschwert, da der Benutzer zu spät ein Feedback bekommt ob die gewünschte Aktion auch wirklich ausgelöst wurde.

- *Verbrauchsverhalten*
  Das Verbrauchsverhalten hat eine hohe Relevanz, da die Ressourcen, die für die Anwendung zur Verfügung stehen, meistens durch das Smartphone stark limitiert sind. Deshalb sollte man versuchen die CPU und Speicherbelastung so gering wie möglich zu halten. Außerdem liegt oftmals eine Bandbreitenreduzierung des mobilen Internets vor, sobald ein bestimmtes Datenvolumen überschritten wurde. Deshalb ist es sehr hilfreich die Zugriffe auf das Internet so gering wie möglich zu halten.

Änderbarkeit

- *Analysierbarkeit*
  Durch eine gut strukturierte Architektur und einen ordentlichen und kommentierten Code soll versucht werden die Analysierbarkeit so hoch wie möglich zu halten.

- *Modifizierbarkeit und Stabilität*
  Da dieses Anwendung auch in der Zukunft weiterhin gepflegt und erweitert werden soll, sind die Modifizierbarkeit und die Stabilität sehr wichtige Aspekte. Durch die Verwendung von erprobten Entwurfsmustern und einer Unterteilung der Anwendung in verschiedene Komponenten sollen diese Eigenschaften gewährleistet werden.

Übertragbarkeit

- *Anpassbarkeit und Austauschbarkeit*
  Auf Grund der Verwendung von Android ist die Anpassbarkeit und die Austauschbarkeit auf Geräte mit einem lauffähigen Android Betriebsystem beschränkt. Dabei muss die Version des Android Betriebssystems die minimale SDK Version der Applikation unterstützten. Bei den Geräten kann es sich zum Beispiel um Smartphones oder Tablets handeln.

- *Installierbarkeit*
  Aufgrund des von Android bereitgestellten App-Stores und der besonderen Rechteverwaltung ist der Aufwand, der bei der Installation der Anwendung betrieben werden muss, für den Benutzer sehr gering.

Daraus ergibt sich hinsichtlich der nicht-funktionalen Anforderungen folgender Teil des Lastenhefts:

Nummer	Kategorie	Beschreibung	Priorität
NFA1	Funktionalität	Die Informationen aus der externen Datenbank des HVBV müssen vollständig und korrekt in der Anwendung verfügbar gemacht werden. Sofern die Funktionen aus den funktionalen Anforderungen auch auf der Internetseite des HVBV verfügbar ist, müssen die dargestellten Informationen entweder gleich sein oder zumindest dieselbe Bedeutung haben.	wichtig

Nummer	Kategorie	Beschreibung	Priorität
NFA2	Funktionalität	Die neue Anwendung muss ohne Beeinträchtigung der Funktionalität des bestehenden Systems integriert werden können. Als Beeinträchtigung zählen das Verändern, Löschen oder Hinzufügen von Daten in die Datenbank des HVBV, oder auch die Beeinträchtigung der Internetseite des HVBV. Außerdem ist ein Performance Verlust, der die ursprünglichen Funktionen für Mitarbeiter oder Benutzer spürbar verlangsamt nicht akzeptabel.	unverzichtbar
NFA3	Funktionalität	Die Sicherheit der Anwendung muss insofern gewährleistet sein, dass keine Sicherheitslücken für den alten Teil des Systems entstehen. Als Sicherheitslücken gelten die Einsicht in nicht öffentliche Informationen oder das unerwünschte Ändern oder Löschen von Daten.	unverzichtbar
NFA4	Zuverlässigkeit	Solange nichts an dem alten System des HVBV verändert wird, darf der Teil der Anwendung, der mit dem alten System kooperiert, nicht bedingt durch Fehlzustände versagen.	wichtig
NFA5	Zuverlässigkeit	Es soll bei Ausfällen des Internets oder der Datenbank des HVBV sichergestellt sein, dass die der Applikation bis dahin bekannten Informationen weiterhin korrekt angezeigt werden können. Abgesehen von den ausgefallenen Komponenten soll die Applikation weiterhin funktionieren.	sehr wichtig
NFA6	Benutzbarkeit	Der Benutzer soll ohne eine Anleitung, nur mittels Hilfe von Symbolen und Tooltipps, in der Lage sein die Funktionalität der Anwendung zu verstehen. Da es sich hierbei um eine besonders subjektive Anforderung handelt, soll durch regelmäßiges Feedback von bestimmten Testpersonen sichergestellt werden, dass die Anwendung für die Mehrheit gut verständlich ist.	sehr wichtig

Nummer	Kategorie	Beschreibung	Priorität
NFA7	Benutzbarkeit	Jede Funktionalität der Anwendung sollte für den Benutzer mit maximal fünf gezielten Aktionen erreichbar sein.	wichtig
NFA8	Effizienz	Die Reaktionszeit der Anwendung soll bei den Interaktionen mit dem Benutzer so gering sein, dass er die Verzögerung nicht als belastend empfindet. Die grafische Oberfläche der Anwendung muss auf Geräten, die nicht älter als zwei Jahre sind, unter normalen Bedingungen mit einer maximalen Verzögerung von einer Sekunde reagieren. Ist dies nicht der Fall, muss der Benutzer durch einen Ladebalken darauf hinwiesen werden, dass eine längere Berechnung durchgeführt wird.	wichtig
NFA9	Effizienz	Die Zugriffe auf die Datenbank mit den Informationen des Ergebnisdienstes und die dadurch übertragenen Daten sollen so gering wie möglich gehalten werden. Einmal abgerufen Daten müssen mindestens bis zum Schließen der Anwendung lokal verfügbar gemacht werden.	sehr wichtig
NFA10	Änderbarkeit	Die Dokumentation des Codes und die Exceptions müssen so gewählt und platziert werden, dass auftretende Fehler auch von nicht am Projekt beteiligten Personen mit der nötigen Ausbildung aufgefunden und verbessert werden können. Dafür muss sichergestellt werden, dass Interfaces und Funktionen, deren Name nicht direkt auf die Funktionalität schließen lässt, mit einer Javadoc Beschreibung versehen werden.	wichtig
NFA11	Modifizierbarkeit	Es soll sichergestellt werden, dass die Anwendung mit so wenig Aufwand wie möglich erweitert oder gewartet werden kann. Änderungen sollten deswegen immer nur kleine Teile der Anwendung betreffen und sich nicht auf die ganze Anwendung auswirken. Aus diesem Grund werden in dem Projekt sowohl das Fassade-, als auch das Factory-Entwurfsmuster verwendet.	wichtig

Tabelle 3.3: Lastenheft - nicht-funktionale Anforderungen

3.3 Machbarkeitsstudie

Nachfolgend wird auf Basis der Ergebnisse der Anforderungsanalyse die Realisierbarkeit der Anforderungen in der Anwendung überprüft. Dazu wird jede Anforderung aus dem Lastenheft daraufhin untersucht, ob sie mit den zur Verfügung stehenden Ressourcen und unter den definierten Bedingungen umgesetzt werden kann. Da für dieses Projekt keine finanziellen Mittel zur Verfügung stehen, wird besonders darauf geachtet, dass durch die Anforderungen keine Kosten entstehen. Außerdem sollte die Umsetzung der Anforderungen nicht so viel Zeit beanspruchen, dass am Ende des Projektes kein sichtbares Ergebnis präsentiert werden kann. Anschließend werden die Risiken für dieses Projekt identifiziert und bewertet. Anhand dieser Bewertungen kann dann entschieden werden, ob es überhaupt sinnvoll ist dieses Projekt fortzuführen.

3.3.1 Lösungsansätze

- *Ansatz für FA1 und FA2*
  Das Anzeigen der Tabellen und Spieltage mit Daten aus einer Datenbank ist unter Anroid recht einfach zu realisieren. Alle erforderlichen Funktionen sind bereits im Android Betriebssystem implementiert. Um die Abfragen aus der Datenbank noch einfacher zu gestalten, kann zusätzlich noch eine ORM-Software verwendet werden. Es gibt lediglich drei Faktoren die zu berücksichtigen sind:
  - Es wird eine gute Datenstruktur benötigt, um die Zusammenhänge zwischen Ligen, Staffeln, Mannschaften, Spieltagen und Paarungen darzustellen.
  - Die Datenstruktur aus der objektorientierten Sprache Java muss verlustfrei und möglichst performant in die Datenbank gespeichert und wieder abgerufen werden können.
  - Das Wichtigste ist jedoch, dass die dafür benötigten Daten über das Internet aus abrufbar sein müssen. Daher ist diese Anforderung stark an die Anforderung FA9 gekoppelt und hängt wesentlich von einem guten Datenbankschema ab.
- *Ansatz für FA3 und FA4*
  Auch für das Einrichten von Favoriten sind alle wichtigen Voraussetzungen bereits durch Android erfüllt. Wie schon bei FA1 und FA2 hängt das Ergebnis dieser Lösung stark von einer guten Datenstruktur ab.
- *Ansatz für FA5 und FA6*
  Da es sich bei Android um ein Betriebssystem von Google handelt, welches zusätzlich

noch sehr viel Wert auf Wiederverwendung von bestehenden Komponenten legt, gibt es bereits mehrere Möglichkeiten Google Maps aus anderen Anwendungen heraus aufzurufen. Mit Hilfe einer ausreichend detaillierten Adresse lassen sich Geodaten bestimmen, die dann an Google Maps übergeben werden können, um die Routen zu berechnen. Ebenso wird für die Verwendung des Google Kalenders eine fertige Schnittstelle zur Verfügung gestellt, die alle benötigten Funktionen anbietet.

- *Ansatz für FA7 und FA8*
  Die Voraussetzungen für die Umsetzung dieser beiden Anforderungen sind ebenfalls bereits durch Android abgedeckt.

- *Ansatz für FA9 - FA12*
  Um diese Anforderungen zu erfüllen, ist es wichtig eine Lösung zu entwickeln, die parallel zu dem bestehenden System des HVBV existieren kann. Eine Möglichkeit wäre das neue System zunächst losgelöst von dem bestehenden System zu entwerfen und anschließend eine Synchronisation der Datenbanken aus dem alten und dem neuen System zu entwerfen. Dabei muss darauf geachtet werden, dass die Synchronisation die Daten in der alten Datenbank nicht verändern kann und keine Sicherheitslücken durch diese Implementation entstehen. Durch die Analyse des Ist-Zustandes konnten die Informationen identifiziert werden, die in die neue Datenbank überführt werden können.

Bei einer ersten Analyse konnten keine Gründe identifiziert werden, weshalb eine der Anforderungen nicht umgesetzt werden kann. Dadurch ist jedoch nicht endgültig sichergestellt, dass alle Anforderungen auch tatsächliche umgesetzt werden können. Jedoch konnten die vorhersehbaren Risiken identifiziert und bewertet werden.

3.3.2 Risikoanalyse

Das Ziel dieses Abschnittes ist es, die Risiken für das Projekt zu identifizieren und anschließend zu bewerten. Mit Hilfe von bestimmten Gegenmaßnahmen kann man versuchen, besonders schwerwiegende Risiken zu eliminieren oder zu reduzieren. Diese Gegenmaßnahmen können in unterschiedliche Kategorien eingeordnet werden, wobei jede dieser Kategorien sowohl Vor- als auch Nachteile besitzt.

- ***Risikovermeidung***
  Aktivitäten ab einem bestimmten Risiko werden komplett unterlassen.
  Vorteile: Die Auswirkungen des Risikos können nicht eintreten.
  Nachteil: Die Aktivität wird nicht durchgeführt und muss durch eine Aktivität mit gleichem Ergebnis und anderem Risiko ersetzt werden. Ansonsten fehlt diese Aktivität in dem Projekt.

- ***Risikoverminderung***
  Durch bestimmte Maßnahmen wird das entstandene Risiko so gut es geht vermindert. Dazu können zusätzliche Mitarbeiter oder bestimmte Technologien eingesetzt werden.
  Vorteil: Die Auswirkungen und die Eintrittswahrscheinlichkeit können vermindert werden.
  Nachteil: Oft werden dazu zusätzliche Ressourcen benötigt und es bleibt trotzdem ein gewisses Restrisiko bestehen.

- ***Überwälzen***
  Risiken werden durch Versicherungen oder Vertragsklauseln abgesichert.
  Vorteil: Die Auswirkungen des Risikos werden so gut es geht abgesichert.
  Nachteil: Das Risiko wird nicht behoben, sondern nur auf andere Beteiligte übertragen. Personen zu finden, die bereit sind die Risiken zu tragen, ist nicht einfach oder sehr teuer.

- ***Risiko selbst tragen***
  Die letzte Möglichkeit ist es das Risiko selbst zu übernehmen. Dabei ist besonders wichtig, das die Entscheidung mit Hilfe der Eintrittswahrscheinlichkeit und den Auswirkungen gut abgewogen wird.
  Vorteil: Kann die letzte Möglichkeit sein ein Projekt trotz des Risikos durchzuführen.
  Nachteil: Die Auswirkungen des Risikos wurden in keinster Weise reduziert.

Ziel dieser Maßnahmen ist es das Restrisiko soweit zu reduzieren, dass sowohl der Auftraggeber, als auch die Projektleiter bereit sind das sie betreffende Restrisiko zu tragen.

In der nachfolgenden Tabelle sind die wesentlichen Risiken für das Projekt aufgelistet. Bei den gewählten Gegenmaßnahmen handelt es sich immer um Risikoverminderung, da die Risiken durch diese Variante auf ein akzeptables Restrisiko reduziert werden könnte, ohne erhebliche zusätzliche Ressourcen aufzuwenden. Dadurch könnten die schwerwiegenden Nachteile der anderen Kategorien von Gegenmaßnahmen vermieden werden. Die Bewertungsskalen für die Eintrittswahrscheinlichkeit und die Auswirkung sind in der Risk-Map (vgl. Tabelle 3.3.2) beschrieben.

Bezeichner	Beschreibung	Eintrittswahrscheinlichkeit	Auswirkung	Gegenmaßnahmen	Eintrittswahrscheinlichkeit nach Gegenmaßnahmen	Auswirkung nach Gegenmaßnahmen
R1	Probleme bei der Integration der Anwendung in das Fremdsystem des HVBV. Dazu zählen Probleme mit der MS Access Datenbank oder beim Übertragen der Daten aus der einen Datenbank in die andere.	3	4	Durch mehrere Meetings und eine Begutachtung des Fremdsystems sollen so viele Informationen wie möglich erlangt werden, um frühzeitig Probleme zu identifizieren	2	4
R2	Probleme seitens des HVBV, was Bereitstellung von Informationen oder Ressourcen angeht. Beispielsweise der Datenbank Zugriff oder zusätzlich benötigte Software.	2	5	Durch Gespräche und ausführliche Erläuterung des Vorhabens möglichst alle Details im Voraus besprechen und klären. Sich besonders Kooperativ zeigen, um die Kooperationsbereitschaft des HVBV zu maximieren.	1	5

Bezeichner	Beschreibung	Eintrittswahrscheinlichkeit	Auswirkung	Gegenmaßnahmen	Eintrittswahrscheinlichkeit nach Gegenmaßnahmen	Auswirkung nach Gegenmaßnahmen
R3	Fehlende Funktionalität von Android, die das zusätzliche Implementieren von Funktionen notwendig macht und somit zusätzliche Ressourcen beansprucht, wodurch die Umsetzung des Projektplans negativ beeinflusst werden könnte.	2	5	Sich im Vorwege über die Funktionalität der Anwendung Gedanken machen und die benötigten Funktionen überprüfen. Gegebenenfalls Zusatzbibliotheken suchen, die den benötigten Mehraufwand einschränken.	1	5
R4	Ausfall oder Verlust von benötigten Ressourcen wie beispielsweise Datenbank, Source Code oder auch dem Computer der zu Entwicklung der Software dient.	1	5	Zumindest den Verlust von Datenbank und Source Code durch regelmäßige redundante Backups absichern.	1	2
R5	Ausfall von Internet und Telefon, was zu einer Einschränkung der Kommunikation zwischen mir und dem HVBV oder zur Behinderung der Arbeit am Projekt führt.	1	3	Das Projekt von verschiedenen Orten aus verfügbar machen, um so gegebenenfalls alternative Arbeitsplätze zu benutzen.	1	2
R6	Krankheit oder Arbeitsunfähigkeit von Projektmitarbeitern und dadurch entstehender Zeitverzug im Projekt.	1	4	Einen gewissen Puffer in dem Projekt einbauen, der ungeplante Vorkommnisse ausgleichen kann.	1	3

Durch Eintragen der Risiken in eine Risk-Map (vgl. Tabelle 3.3.2) erhält man eine gute Übersicht über die identifizierten Risiken des Projektes. In den verschiedenen Gefahrenbereichen sind die dort einzuordnenden Risiken anhand ihrer Bezeichner verzeichnet. Dabei zeigen die in der Karte durch eckige Klammern markierten Risiken den Gefahrenbereich nach Berücksichtigung der Gegenmaßnamen.

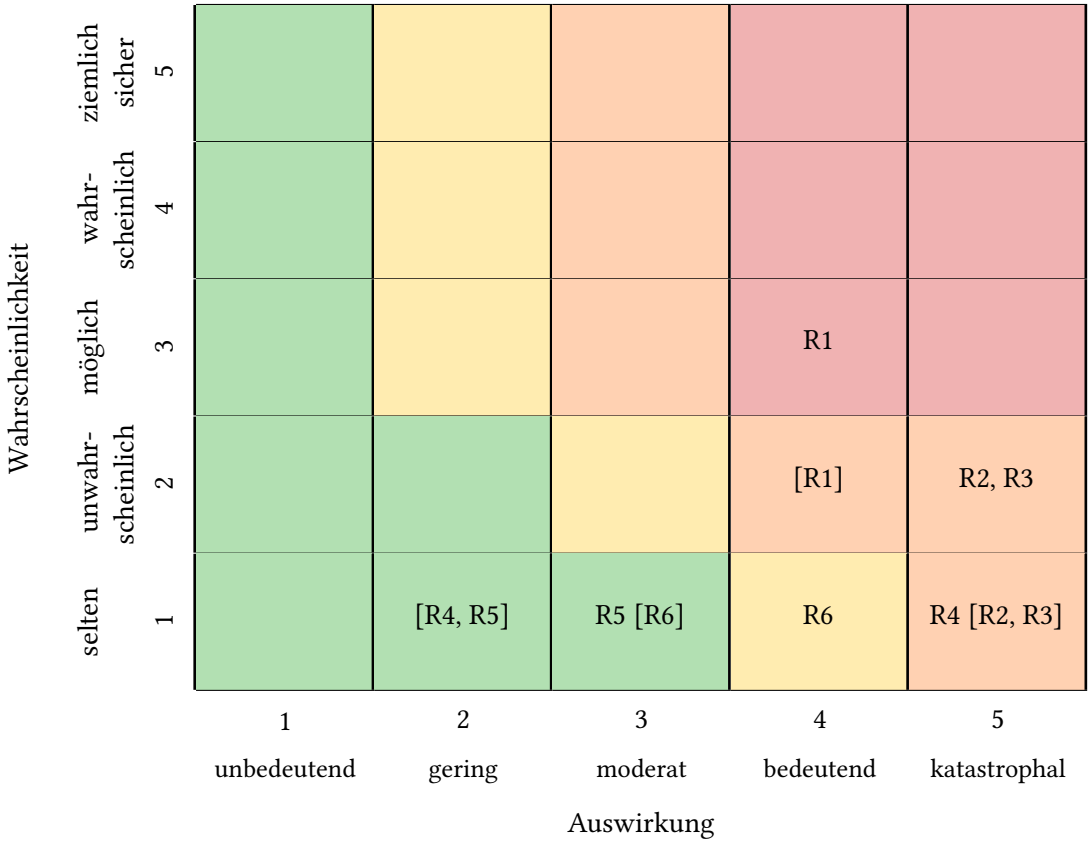


Tabelle 3.4: Risk-Map der für das Projekt identifizierten Risiken

Gut zu erkennen ist, dass unter Anwendung der Gegenmaßnahmen die Risiken des Projektes deutlich reduziert werden könnten. Durch sie befindet sich kein einziges Risiko mehr in der bedenklichen roten Zone und die Hälfte aller Risiken wurden dadurch in den sicheren grünen Bereich verschoben. Auf der Basis eines recht geringen Gesamtrisikos könnte das Projekt von ohne große Bedenken weitergeführt werden.

3.4 Fazit

Die Machbarkeitsstudie hat gezeigt, dass alle Anforderungen mit den zur Verfügung stehenden Ressourcen umsetzbar sind. Es ist nicht ausgeschlossen, dass unerwartet Probleme oder Risiken auftreten, die in der Risikoanalyse nicht berücksichtigt wurden. Durch diese Vorgehensweise war es möglich sich ein Überblick über die vorhersehbaren Risiken des Projektes zu verschaffen. Dadurch wurde versucht die Wahrscheinlichkeit der erfolgreichen Durchführung des Projektes zu erhöhen. Es wurde jedoch deutlich, dass selbst für ein kleines Projekte mit wenigen Beteiligten und ohne finanziellen Druck gewisse Risiken bestehen. Durch die definierten Gegenmaßnahmen könnte das Restrisiko so stark reduziert werden, so dass die beteiligten Parteien bereit wären das Projekt auf dieser Basis zu realisieren.

4 Design

Während der Analysephase wurden die Anforderungen an die Anwendung erfasst und spezifiziert. Auf Basis dieser Anforderungen soll jetzt mit Hilfe von Methoden des Software-Engineerings ein passendes Design für die Anwendung entwickelt werden.

4.1 Verwendete Bibliotheken

Zu Beginn der Designphase wurde festgelegt, welche Teile der Anwendung neu entwickelt werden und welche bestehenden Bibliotheken zur Verfügung gestellt oder erweitert werden. Die beiden folgenden Bibliotheken stehen jeweils unter Open Source Lizenz und konnten deshalb ohne Bedenken in diesem Projekt verwendet werden.

4.1.1 OrmLite

OrmLite ist eine auf die wesentlichen Grundfunktionen beschränkte Bibliothek um objektrelationale Abbildung unter Java vorzunehmen. Durch diese Beschränkung ist die Bibliothek wesentlich kleiner als andere Bibliotheken und eignet sich dadurch besonders gut für Projekte mit beschränkten Ressourcen.

OrmLite unterstützt die gängigsten Datenbanktypen wie MySQL, Postgres, Microsoft SQL Server, H2, Derby, HSQLDB und Sqlite. Auch DB2, Oracle, ODBC und Netezza werden provisorisch unterstützt und andere Datenbank können eigenständig hinzugefügt werden. Durch Annotationen in den Klassen die abgebildet werden sollen und einer kurzen Einstellungsdatei in der die DAO-Objekte erzeugt werden, kann man sehr schnell und unkompliziert eine Datenstruktur in einer Datenbank persistent machen. Zu finden ist diese Bibliothek unter http://ormlite.com/ [1].

[1] Zuletzt aufgerufen am 10.8.2012

4.1.1.1 Beispiel

```
1  ...
2  @DatabaseTable(tableName = "teams")
3  public class TeamImpl implements Team {
4          ...
5          @DatabaseField(columnName = COLUMN_ID, generatedId = true,
               allowGeneratedIdInsert = true)
6          int id;
7          @DatabaseField(columnName = COLUMN_NAME)
8          String name;
9          @DatabaseField(columnName = COLUMN_TAG)
10         String tag;
11         @DatabaseField(columnName = COLUMN_NR)
12         int nr;
13         @DatabaseField(foreign = true, columnName = COLUMN_RELAY)
14         RelayImpl relay;
15         ...
16 }
```

Listing 4.1: OrmLite Annotationen

Zunächst wird anhand der Annotation in Zeile 1 der Name der Tabelle festgelegt, in der die Objekte gespeichert werden sollen. Mit Hilfe der Annotation @DatabaseFile werden anschließend die Spaltennamen angegeben, in die bestimmte Variablen gespeichert werden. Außerdem können spezielle Eigenschaften wie die automatische Erhöhung von ID Werten (generateId = true) oder das Verhindern von automatisch generierten ID's, sobald der Benutzer explizit eine ID angibt (allowGeneratedIdInsert), festgelegt werden. Die Konvertierung von bekannten Java Typen in die jeweiligen kompatiblen Typen für die Datenbankspalten wird von OrmLite automatisch durchgeführt.

Durch die Angabe von *foreign = true* kann man eine Eins-zu-Eins Beziehung mit einem beliebigen anderen Objekt umsetzten. Anschließend kann man über einfache Java *Getter*- und *Setter*-Methoden auf die definierten Felder zugreifen.

Durch die von OrmLite bereitgestellten DAO Objekte werden Funktionen, wie das Einfügen, Löschen oder Verändern von Objekten, bereits zur Verfügung gestellt, genauso wie das Abrufen aller Datenbankeinträge von einem bestimmten Objekttyp. Alle anderen Datenbankabfragen können entweder mit Hilfe eines Querybuilders oder einfacher SQL Befehle umgesetzt werden. Die gesuchten Objekte werden unter OrmLite nicht komplett aus der Datenbank geladen, sondern zunächst nur mit der korrekten ID und Defaultwerten zurückgegeben. Möchte man auf alle Informationen des Objektes zugreifen, so muss man die Methode *reaload* auf dem

Objekt aufrufen. Dadurch kann man den Overhead reduzieren, indem man nur gezielt die Objekte lädt, die man tatsächlich verwendet.

4.1.2 Greendroid

Bei Greendroid handelt es sich um eine Java/Android Bibliothek, die für jede verschiedene Android Activity Klasse jeweils eine erweiternde Klasse zur Verfügung stellt. In diesen Klassen hat man zusätzliche Möglichkeiten, beispielsweise eine Navigationsleiste am oberen Bildschirmrand oder sogenannte QuickActionBar's, die dem Benutzer eine sehr komfortable Möglichkeit zur Interaktion bieten. Speziell für die Umsetzung der funktionalen Anforderungen und das Einhalten der nicht-funktionalen Anforderungen NFA6 und NFA7 ist diese Bibliothek besonders nützlich und erspart eine Menge zusätzliche Arbeit.

In den aktuellsten Android SDK Versionen sind viele dieser Erweiterungen bereits enthalten. Allerdings enthält die Android Version 2.2, die auf Grund der Umfrage aus dem Kapitel Analyse ausgewählt wurde, diese Funktionen noch nicht. Zu finden ist die Bibliothek unter folgender Adresse https://github.com/cyrilmottier/GreenDroid [2]

4.2 Architektur

Die Architektur eines Software Systems dient in einem Projekt als Übergang von der Analyse hin zur Implementation. Ihre Aufgabe besteht darin, einen groben Überblick über das gesamte System zu liefern und dadurch den beteiligten Personen im Projekt als Orientierung zu dienen (vgl. Starke (2009) S. 30 f). Die Architektur ist

> "eine strukturierte oder hierarchische Anordnung der Systemkomponenten sowie Beschreibung ihrer Beziehungen." (Balzert (2001), S. 716)

Mit Hilfe von Beschreibungssprachen für Strukturen und Abläufe wie beispielsweise UML werden Artefakte entworfen, die einen guten Überblick über die gesamten Komponenten des Systems ermöglichen. Mit Hilfe dieser Artefakte wird die Struktur der Komponenten und deren Zusammenwirken festgehalten, nicht jedoch ihre exakte Arbeitsweise. Die Auswirkungen von Designentscheidungen die hier getroffen werden, haben oftmals eine starke Auswirkung auf das Projekt und können erst lange nach der Entscheidung beurteilt werden (vgl. Starke (2009) S. 17).

[2] Zuletzt aufgerufen am: 16.9.2012

4.2.1 Systemüberblick

Durch das System des HVBV, in das die neue Anwendung integriert werden muss, gibt es für die neue Architektur gewisse Grundvoraussetzungen die beachtet werden müssen. Deshalb werde ich zuerst ein Überblick über das bestehende System des HVBV geben und dieses dann um die benötigten Komponenten erweitern.

4.2.1.1 Die Verteilungssicht

Durch die Verteilungssicht soll gezeigt werden, in welcher Umgebung das neu zu erstellende System ablaufen wird. Dafür werden bekannte Hardwarekomponenten und die zwischen ihnen verwendeten Kommunikationsprotokolle aufgezeigt und beschrieben.

In der Verteilungssicht (vgl. Abbildung 4.1) ist zu erkennen, dass das gesamte System über drei verschiedene Hardwarekomponenten verteilt ist. Die Kommunikation zwischen den alten Komponenten findet über das FTP Protokoll statt. Die neuen, in grün markierten, Komponenten sollen hingegen nur noch über wohldefinierte REST Web Services Schnittstellen miteinander kommunizieren.

- *Büroserver*
  Hier befindet sich die Microsoft Access Datenbank, welche die für den Ergebnisdienst zu modellierenden Daten enthält. Da in dieser Datenbank zusätzlich noch weitere wichtige und persönliche Daten stehen, ist der Zugriff auf sie nur von dem Büro aus möglich. Außerdem liegen hier die Formulare und zusätzlichen Programme, die benötigt werden, um die Ergebnisse in die Microsoft Access Datenbank zu schreiben. Als Erweiterung kommt nun ein Batch Programm hinzu, das die für den Ergebnisdienst benötigten Daten aus der Microsoft Access Datenbank ausliest und über eine REST Web Services Schnittstelle in eine neue Datenbank auf dem Webserver schreibt.

- *Webserver*
  Auf dem Webserver liegen alle für den Internet Auftritt des HVBV benötigten Dateien. Zusätzlich befindet sich dort eine Datenbank, welche Einstellungen, Benutzer oder Artikel von der Webseite enthält. Voraussetzung für den Zugriff via REST Web Services ist eine Unterstützung der Scriptsprache PHP, welche bereits im bestehenden System gegeben ist. Als Erweiterung wird hier eine weitere Datenbank installiert, die ausschließlich mit den Daten für den Ergebnisdienst des HVBV gefüllt wird. Diese Datenbank hätte auch an anderer Stelle installiert werden können, dabei muss nur berücksichtigt werden, dass man über das Internet auf die Datenbank zugreifen kann. Da sie allerdings von der Internetseite benutzt werden soll, ist es für die Zugriffszeit natürlich von Vorteil beide Komponenten auf einem Server zu betreiben.

- *Android Endgeräte*
  Die dritte Komponente bilden die Android Geräte, die natürlich in größerer Zahl vorhanden sein können. Auf ihnen findet man dann die aus dem Android Market installierte App und eine mit der App installierte SQLite Datenbank. Die zusätzliche Datenbank auf dem Android Gerät wird benötigt, um auch ohne eine bestehende Internetverbindung auf die erforderlichen Daten zuzugreifen. Dafür muss die interne Datenbank mit der externen Datenbank auf dem Webserver synchronisiert werden. Durch diese Maßnahme wird dadurch die Anzahl der Zugriffe über das Internet drastisch reduziert.

  Die rot markierten Artefakte aus der Verteilungssicht (vgl. Abbildung 4.1) könnten auch entfernt werden, da die Informationen jetzt alle in der neuen MySQL Datenbank zur Verfügung stehen. Allerdings werden sie vorerst erhalten bleiben, um einen nahtlosen Übergang in das neue System zu garantieren. In einem Notfall ist es dann zu jedem Zeitpunkt möglich die Informationen wieder über die alte Funktionalität abzurufen.

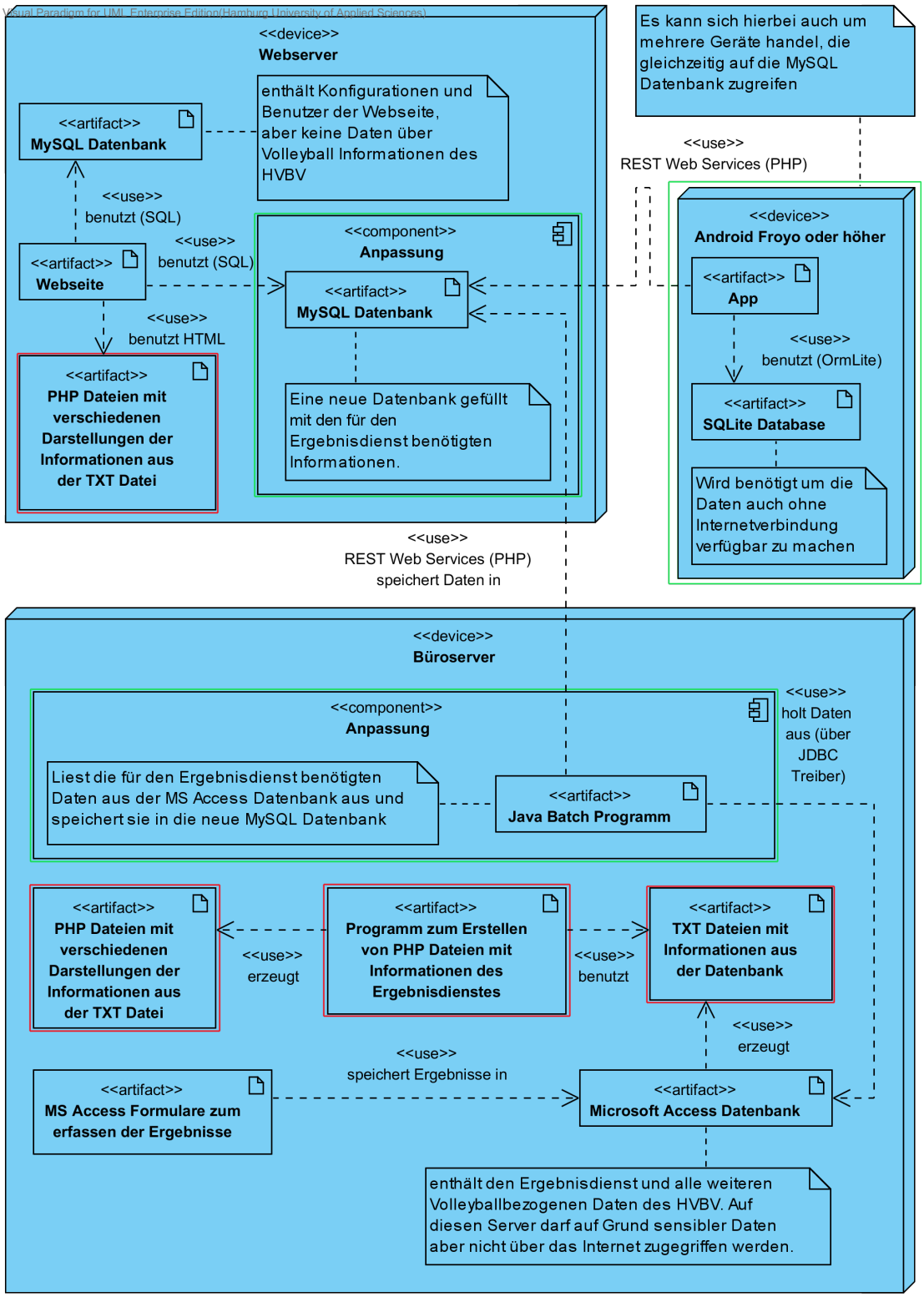


Abbildung 4.1: Verteilungssicht der Anwendung

4.2.2 Datenbanken

Auf jeder Hardwarekomponente aus der Verteilungssicht (vgl. Abbildung 4.1) befindet sich eine eigenständige Datenbank. Sie spielen für das Projekt eine wesentliche Rolle, weswegen ich sie gesondert betrachten möchte. Dabei wird der Aufbau und die Funktion der jeweiligen Datenbank genauer erläutert.

- *Microsoft Access Datenbank*
  Auf dieser Datenbank befinden sich alle für den HVBV benötigten Daten. Da es sich hierbei auch um sensible Stammdaten handelt, kann man diese Datenbank nicht von außen über das Internet ansprechen. Die für die Anwendung benötigten Informationen müssen deshalb von dem Büroserver aus in der Datenbank herausgesucht werden und anschließend in die MySQL Datenbank überspielt werden.

- *MySQL Datenbank*
  Diese Datenbank enthält, alle ohnehin auf der Internetseite frei einzusehenden Daten, für den Ergebnisdienst des HVBV. Dabei ist es besonders wichtig, dass die Datenbank möglichst flexibel ist, um auf Änderungen oder Ergänzungen des Ergebnisdienstes seitens des HVBV angemessen reagieren zu können. Aus den funktionalen Anforderungen wurde dann, zusammen mit dem Verantwortlichen für die Homepage des HVBV, ein Datenbankschema (vgl. Abbildung 4.2) entworfen. Dabei wurde die Struktur des HVBV folgendermaßen umgesetzt:

  Liga (League)
  Es gibt unterschiedliche Ligen welche einen Namen und ein für die Liga vorgesehenes Geschlecht (Gender) besitzen. Die league_id macht eine Liga einzigartig, so dass später auch Ligen mit gleichem Namen und Geschlecht möglich wären.

  Staffel (Relay)
  Außerdem besteht eine Liga aus mehreren Staffeln, welche durch ihre league_id eindeutig einer Liga zugeordnet werden. Staffeln haben neben ihrer relay_id und der schon erwähnten league_id noch eine Nummer (nr) und das Datum des Saisonstarts (season_start) und des Saisonendes (season_end). Einer Staffel können mehrere Mannschaften zugeordnet werden, die in dieser Staffel spielen.

Mannschaft (Team)
Dazu gibt es bei den Mannschaften eine relay_id die als Fremdschlüssel für die Liga dient, in der sie spielt. Weitere Eigenschaften sind der Vereinsname (name), eine passende Abkürzung zu dem Vereinsnamen (tag) und eine Nummer (nr), falls ein Verein mehrere Mannschaften besitzt.

Spieltag (Matchday)
Außerdem können einer Staffel mehrere Spieltage zugewiesen werden, indem in der Spalte relay_id die passende Staffel ID eingetragen wird. Zusätzlich besitzt ein Spieltag noch eine Adresse (Address), eine Mannschaft die den Spieltag ausrichtet (organizer_id), eine Nummer (nr) und ein Datum (date) an dem der Spieltag stattfindet. Einem Spieltag wiederum können über den Fremdschlüssel matchday_id mehrere Begegnungen zugewiesen werden.

Begegnung (Matching)
Eine Begegnung besteht aus einer Spielnummer (game_nr), zwei Mannschaften die gegeneinander antreten (team1_id, team2_id) und einem Schiedsrichter (referee_id). In team1_records und team2_records werden die gewonnen Sätze eingetragen und ein Attribut *confirmed* zeigt an, ob die Ergebnisse bereits vom HVBV bestätigt wurden.

Tabellensituation (Standing)
In der Tabelle *Standings* wird die Tabellensituation eines Teams für eine Saison gespeichert werden. Man kann an diese Informationen, auch über Beziehungen zwischen den Tabellen gelangen, was die Tabelle Standings redundant macht. Diese Redundanz hat zwar den Nachteil, dass bei dem Einpflegen, Löschen und Bearbeiten von Daten darauf aufgepasst werden muss, dass keine Synchronisationsfehler auf Grund der Redundanzen entstehen. Zusätzlich entsteht dadurch auch ein minimal höherer Speicherplatzbedarf. Es gibt aber gewisse Vorteile in der Performance und der Flexibilität dieser Variante. Die Informationen in der Tabelle Standings, welche oft von der Anwendung verwendet werden, können direkt aus der Tabelle abgelesen werden und müssen nicht jedes Mal durch eine komplizierte Anfrage über mehrere Tabellen abgefragt werden. Außerdem hat man durch die zusätzliche Tabelle die Möglichkeit, nach einer Saison, die Spieltage, Begegnungen und Ergebnisse zu löschen, ohne die Tabelle aus dieser Saison zu verlieren. Dadurch kann man die Daten in der Datenbank so gering wie möglich halten und ist trotzdem in der Lage eine Historie der Platzierungen eines Teams in den vergangen Spielzeiten festhalten.

Die Tabelle Timestamps

Es existiert noch eine weitere Tabelle *Timestamps*, die nicht in der Abbildung 4.2 mit aufgeführt ist. Sie wird genauso wie das Attribut (timestamp) aus den zuvor erwähnten Tabellen für die Synchronisation dieser Datenbank mit der jeweiligen SQLite Datenbanken auf den Android Geräten benötigt. Genauere Informationen hierzu befinden sich im Abschnitt 5.1.2.

- *SQLite Datenbanken*
  Die letzte Datenbank befindet sich auf den jeweiligen Android Endgeräten und dient zur Offline Verfügbarkeit der Daten und einer verbesserten Zugriffszeit. Es handelt sich dabei um eine Spiegelung der MySQL Datenbank. Dadurch sind Zugriffe auf die MySQL Datenbank über das Internet immer nur dann nötig, wenn sich die Informationen in der MySQL Datenbank verändert haben. Ansonsten kann die Anwendung direkt mit den Daten aus der SQLite Datenbank arbeiten.

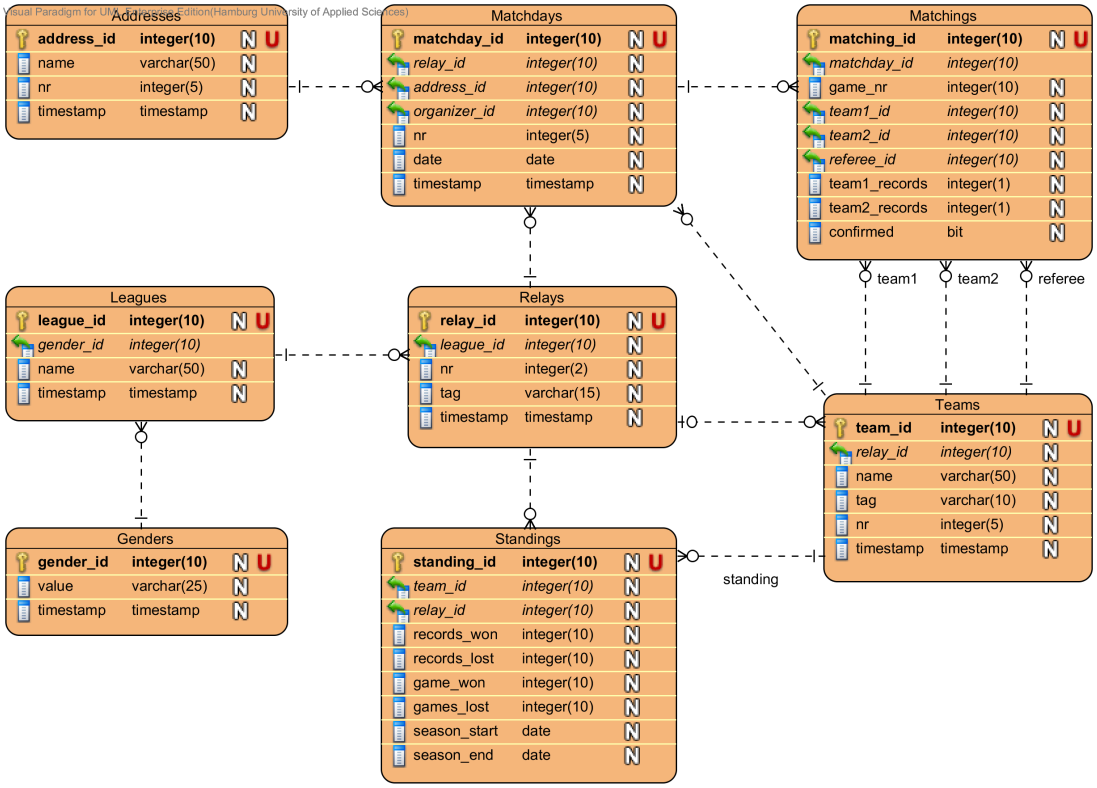


Abbildung 4.2: Datenbank Schema

4.2.3 Technische Strukturierung

Mit Hilfe des Entwurfsmusters Seperation of Concerns (SoC), wird die Anwendung in mehrere Komponenten, mit unterschiedlichen Verantwortlichkeiten unterteilt. Diese Unterteilung erleichtert die kompakte Darstellung einzelner Teilkomponenten und deren Interaktionen. Durch die technische Trennung der Anwendung können ähnliche Teile in Packages gruppiert werden und es entsteht eine Struktur.

Zunächst wurde die gesamte Anwendung in eine Android unabhängige (core) und eine Android abhängige (android) Komponente gegliedert. Dadurch können die Android unabhängigen Teilkomponenten für Umsetzungen auf anderen Java unterstützenden Plattformen weiterverwendet werden. Anschließend wurden die beiden Komponenten durch Packages weiter unterteilt (vgl. Abbildung 4.3), wobei jedes Package einen bestimmten technischen Teil der Anwendung abdeckt.

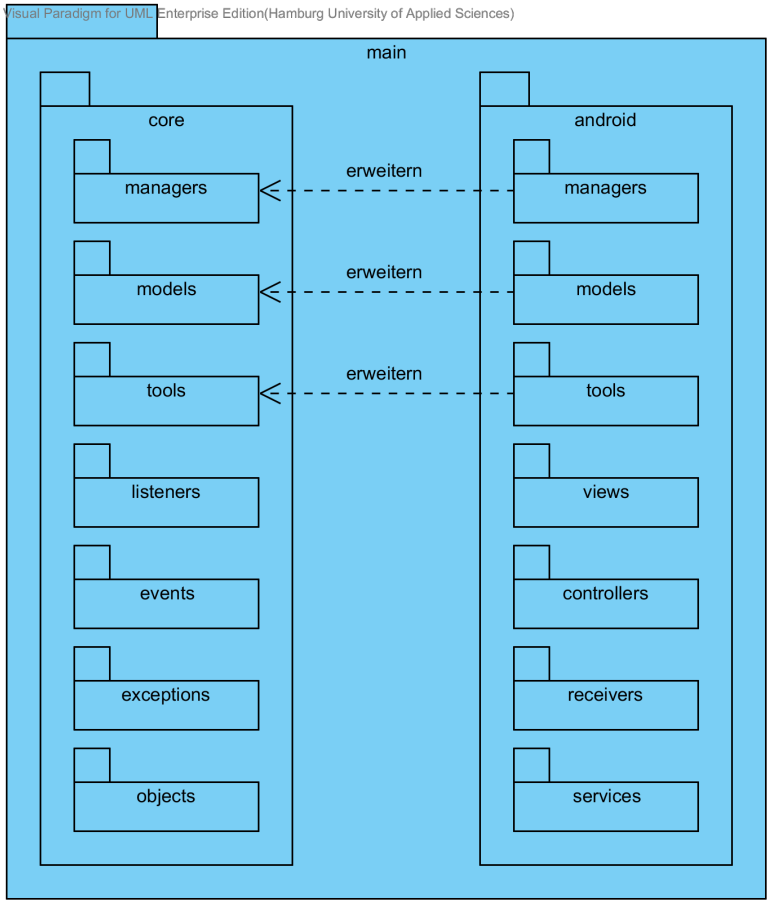


Abbildung 4.3: Unterteilung der Anwendung in verschiedene Packages

4.2.3.1 Core Komponente

Die Core Komponente stellt den eigentlichen Anwendungskern der Anwendung dar. Er enthält eine Fassade Klasse die nach dem Fassade Entwurfsmuster (vgl. Abschnitt 4.2.5) aufgebaut ist und über das implementierte Interface ApplicationCore eine einheitliche Schnittstelle für die darunter liegenden Teilkomponenten (Managers) darstellt.

Über das Application Interface gelangt man an den ApplicationCore mit dem man Zugriff auf alle für die Anwendung relevanten Methoden bekommt (vgl. Abbildung 4.4).

Für das Instanziieren von der Fassade ist die jeweilige Main Klasse der plattformabhängigen Erweiterung verantwortlich. Diese implementiert das Application Interface und erstellt die Manager Klassen und die Fassade.

4.2.3.2 Managers

Die Manager Komponenten kapseln gezielt technische Teilaufgaben der Anwendung, wie beispielsweise den Zugriff auf die interne Datenbank, die Synchronisation zwischen interner und externer Datenbank, das Prüfen der Erreichbarkeit von bestimmten Ressourcen oder die Verwaltung der Benutzer abhängigen Einstellungen. Sollten sich also Einflussfaktoren für die Manager Klassen ändern, existiert durch sie ein Single Point of Control (SPoC), der die Änderungen auf die Manager Klassen beschränkt und so der Rest der Anwendung nicht von den Änderungen betroffen ist.

4.2.3.3 Models

Die Models stellen die Daten bereit, die in den unterschiedlichen Views der Anwendung wiedergegeben werden. Was genau die Models bewirken, wie sie mit den Views und Controllern interagieren und wie sie entworfen wurden, wird in dem Abschnitt 4.2.6 detaillierter beschrieben.

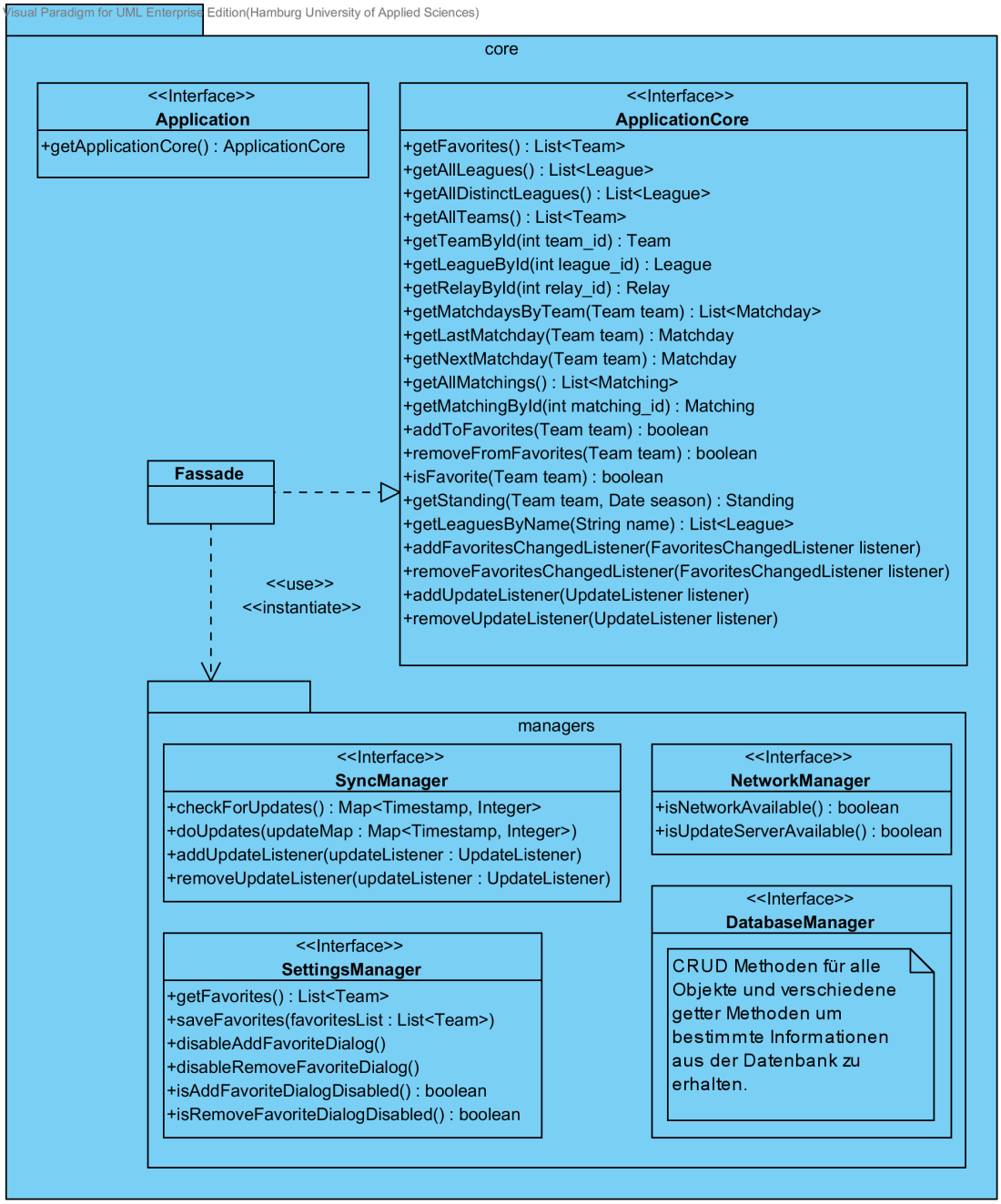


Abbildung 4.4: Aufbau der Core Komponente

4.2.3.4 Events

Um die Änderungen in den Model Klassen an die dazugehörigen Views weiterzugeben, werden von den Models sogenannte Events ausgelöst, für die sich die Views bei dem Model registrieren können. In diesem Package befindet sich die Funktionalität für die Models, um Listener zu registrieren und sie bei dem Eintreten von bestimmten Events zu informieren.

4.2.3.5 Listeners

Hier befinden sich die Listener Interfaces die implementiert werden müssen, wenn man sich beim ApplicationCore als Listener für bestimmte Events registrieren möchte. Momentan existieren Listener zum Verfolgen des Update Prozesses und zur Benachrichtigung von Änderungen an der Favoritenliste. Die dazu passenden Interfaces sind in der Abbildung 4.10 dargestellt.

4.2.3.6 Tools

In dem Package Tools findet man Hilfsklassen, die für mehrere andere Teilkomponenten nützlich sein können und keine direkte Zugehörigkeit zu speziellen Teilkomponenten haben. Zum Beispiel zur Konvertierung von Datumsformaten oder der Berechnung der Anzahl von Tagen zwischen zwei Terminen.

4.2.3.7 Basic Objects

Für die Anwendung wurde in dem Abschnitt 4.2.2 eine externe MySQL Datenbank erstellt, welche die Informationen des Ergebnisdienstes des HVBV enthält. Die Basic Objects repräsentieren genau diese Informationen in der Anwendung. Es handelt sich dabei um Java Klassen, die bestimmte Methoden zur Verfügung stellen, um auf die für die Anwendung wichtigen Informationen aus der Datenbank zugreifen zu können. Die Abbildung 4.2.3.8 zeigt eine Übersicht über die Funktionen und Verbindungen zwischen den Objekten.

4.2.3.8 Android Komponente

Die Android Komponente nutzt die Core Komponente und erweitert sie um die Android spezifischen Elemente.

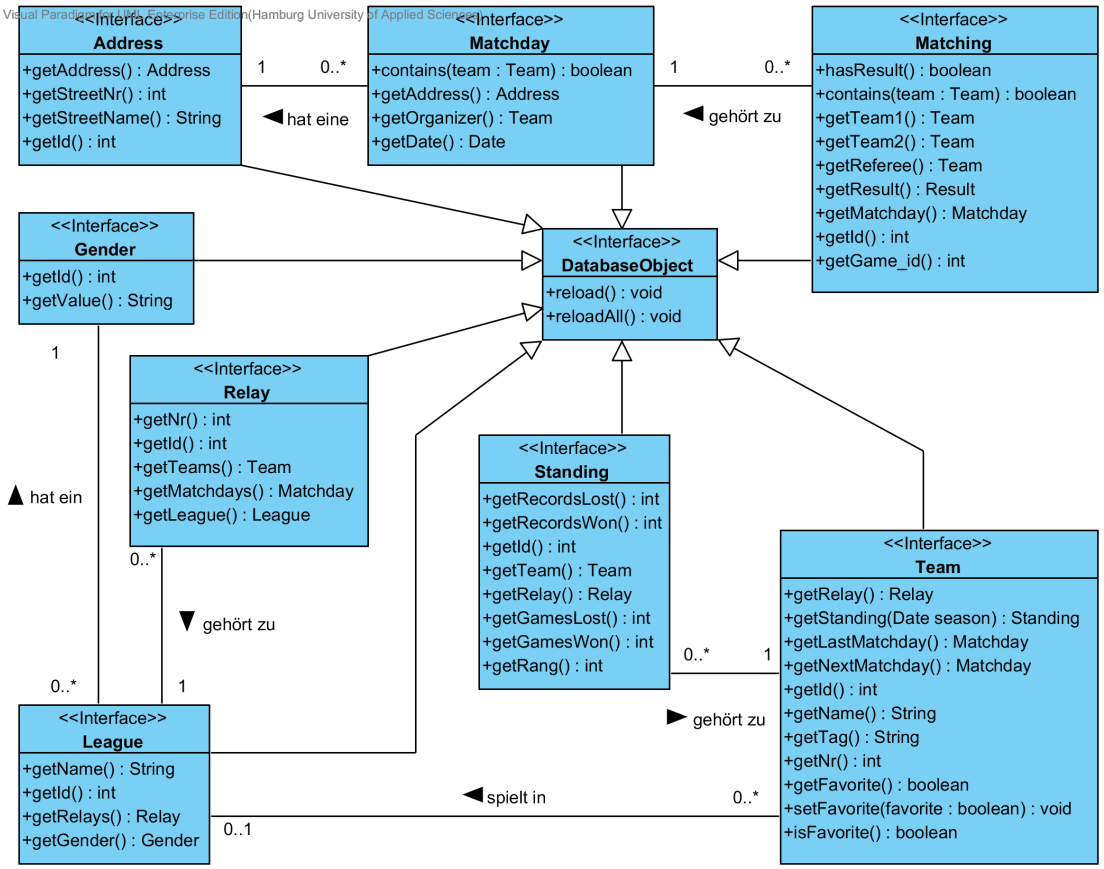


Abbildung 4.5: Klassendiagramm der Basic Objects

4.2.3.9 Managers

In diesem Package befinden sich die Manager für die Android Komponente. Dabei handelt es sich um Manager Implementierungen, die von der Core Komponente vorausgesetzt werden, wie zum Beispiel der *NetworkManager*, der *SettingsManager* oder der *DatabaseManager*. Es werden allerdings für die Funktionalität der Android Komponente zusätzlich spezielle Manager benötigt, die nicht in der Core Komponente implementiert werden können. Das liegt daran, dass sie stark von den jeweiligen Endgeräten abhängen, auf denen die Anwendung installiert werden soll. Dazu gehören Manager wie der *ServiceManager* oder der *NotificationManager*, die nur für die Umsetzung unter Android benötigt wird. Der *ServiceManager* startet und verwaltet die benötigten Services (vgl. Abschnitt 2.1.2.3) und der *NotificationManager* verwaltet die von der Anwendung benutzten Notifications (vgl. Abschnitt 2.1.2.2).

4.2.3.10 Models, Views und Controller

Für die Android Implementation müssen manche Models aus der Core Komponente erweitert werden. Die passenden Views dazu Um dem Benutzer die Informationen der Anwendung sichtbar zu machen werden Views benötigt, die jeweils einem Model zugeordnet werden. Zu jedem View aus der Anwendung muss es ein passenden Controller geben, der auf Benutzereingaben in den Views reagiert. Für jede dieser verschiedenen Objekte gibt es jeweils ein eigenes Package in dem sie gruppiert werden. Genauere Informationen hierzu gibt es im Abschnitt 4.2.6.

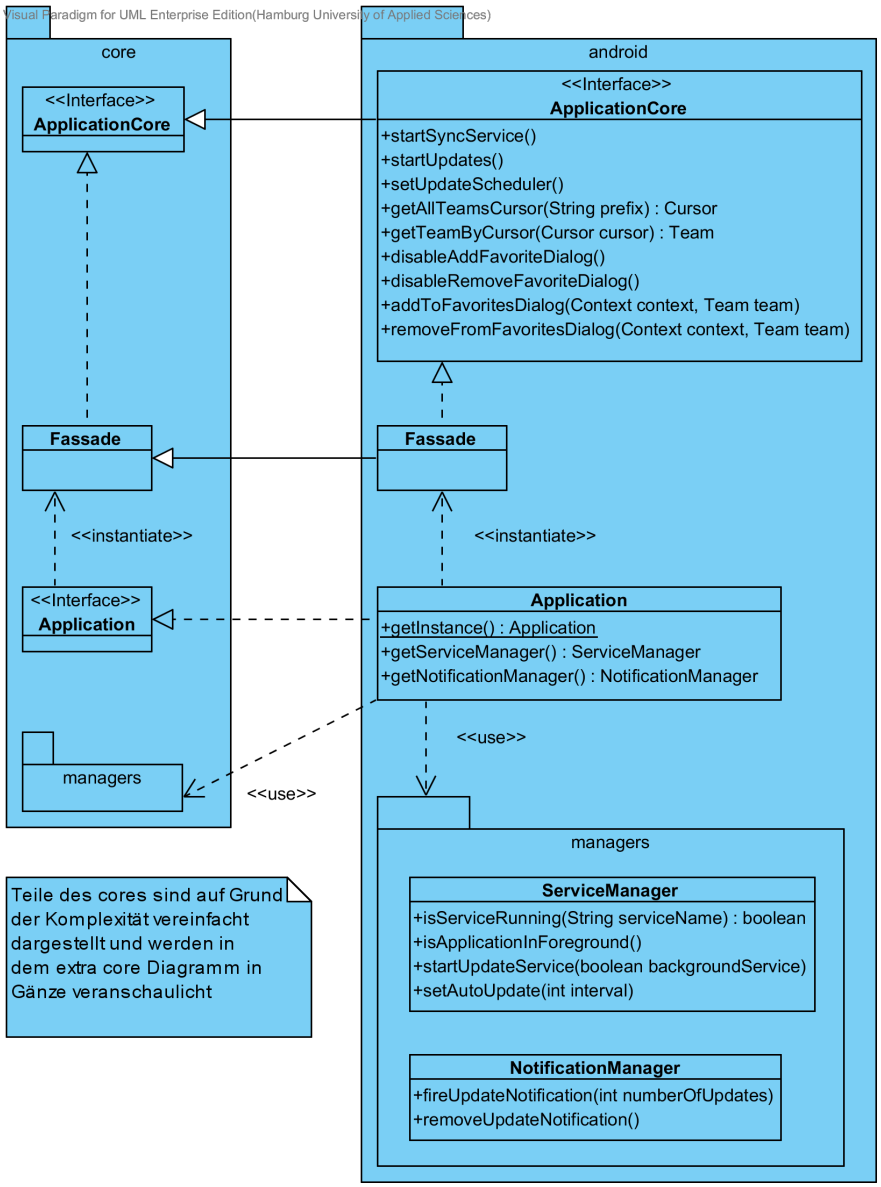


Abbildung 4.6: Aufbau der Android Komponente

4.2.3.11 Receivers

Für die Anwendung werden die folgenden drei verschiedene Broadcastreceiver (vgl. Abschnitt 2.1.2.6) benötigt:

- *AfterBootReceiver*
  Dieser Receiver wartet auf ein bestimmtes Intent (vgl. Abschnitt 2.1.2.5) des Android Systems, welches nach jedem Start des Gerätes verschickt wird. Daraufhin werden dann, je nach Benutzereinstellung, die automatischen Updateintervalle durchgeführt. Dies geschieht über einen von Android zur Verfügung gestellten *AlarmManager*, der zu bestimmten Zeiten und Intervallen Intents verschicken kann.

- *StartUpdateReceiver*
  Die vom *AfterBootReceiver* versandten Intents werden von diesem Receiver empfangen und es wird der Updateprozess über den *ApplicationCore* gestartet.

- *UpdateStatusReceiver*
  Der *UpdateStatusRecevier* dient zu dem Empfangen von neuen Update Benachrichtigungen. Er wird überall System benötigt, wo auf den Verlauf oder die Fertigstellung eines Updates reagiert werden soll. Zum Beispiel durch eine Aktualisierung der Benutzeroberfläche auf Grund neuer Informationen.

4.2.3.12 Services

Hier befinden sich alle Services (vgl. Abschnitt 2.1.2.3), die für die Anwendung benötigt werden. Dazu zählen der *AfterBootService*, der nach jedem Start des Android Gerätes durch den *AfterBootReceiver* gestartet wird und der SyncService, der für die Synchronisation der internen und externen Datenbank verantwortlich ist.

4.2.4 Factory Entwurfsmuster

Mit Hilfe des Factory Entwurfsmusters wurde ein SPoC zur Instanziierung der konkreten *Basic Objects* (vgl. Abbildung 4.2.3.8) geschaffen. Somit kann mit nur einer Änderung in der Factory Methode, die konkrete Implementierung einer Klasse durch eine andere Implementierung ersetzt werden. Die neue Implementierung muss dazu nur das passende Interface oder eine Erweiterung davon zur Verfügung stellen. Da jedes neue Objekt über die Factory Klasse erstellt wird, können dort zusätzlich noch bestimmte Vorbedingungen geprüft werden. So kann beispielsweise eine Übergabe von *null* als Parameter verhindert und durch einen Defaultwert ausgetauscht werden. Durch diese Eigenschaften verbessert man die Modifizierbarkeit der Anwendung, was sich positiv auf die NFA11 auswirkt. Außerdem wird durch die Prüfung der Parameter die Zuverlässigkeit des Systems erhöht, was der NFA4 zu Gute kommt. Die Architektur des Factory Entwurfsmusters lässt sich sehr gut durch die Abbildung 4.2.4 veranschaulichen (vgl. Gamma u. a. (2000) S. 107 ff).

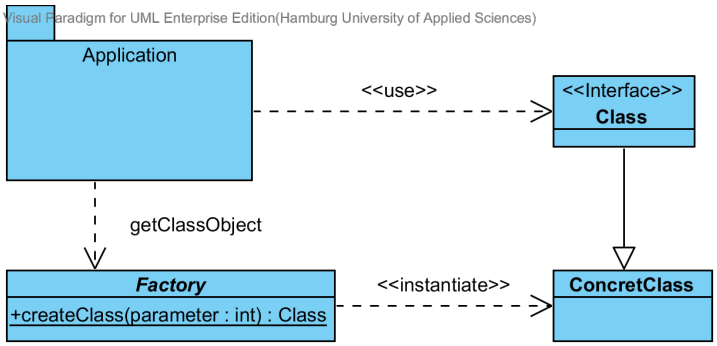


Abbildung 4.7: Strukturübersicht des Factory Patterns

In der Anwendung wird gegen ein Interface (in der Abbildung 4.2.4 die *Class*) programmiert. Das Erstellen eines neuen Basic Objects in der Anwendung, findet immer mit Hilfe der Factory Klasse statt. Diese erstellt dann eine dort festgelegte konkrete Klasse (in der Abbildung 4.2.4 die *ConcretClass*).

4.2.5 Fassade Entwurfsmuster

Das Fassade Entwurfsmuster fällt in die Kategorie der *Structural Patterns* und dient dazu eine vereinfachte zentrale Schnittstelle in der Anwendung bereitzustellen (vgl. Gamma u. a. (2000)). Dafür wird durch die Fassade eine weitere Indirektionsebene eingeführt, mit deren Hilfe die verschiedenen Subsysteme in einer Schnittstelle zusammengefasst werden.

Weil die Subsysteme hinter der Fassade versteckt werden, wird die Komplexität der Anwendung reduziert. Alle Zugriffe auf die Subsysteme finden über die Fassade statt, wodurch die Kommunikation zwischen den verschiedenen Komponenten auf wenige Schnittstellen reduziert wird. Hierdurch entsteht eine lose Kopplung, die dazu führt, dass Änderungen in den Subsystemen nur dann den Rest der Anwendung betreffen, wenn diese zu einer Veränderung der Schnittstellen führen. Dieser Faktor wirkt sich natürlich maßgeblich auf die Änderbarkeit und Wartbarkeit der Anwendung aus und hilft dadurch die NFA11 zu erfüllen.

Der Nachteil dieses Entwurfsmusters ist die Einführung der zusätzlichen Indirektionsebene, über die der Zugriff auf die benötigten Methoden an die Subsysteme weitergeleitet wird.

Trotz des zusätzlichen Overheads durch die benötigte Indirektionseben wird dieses Entwurfsmuster in dem Projekt verwendet, da die Vorteile der losen Kopplung die Nachteile deutlich überwiegen. Da diese Anwendung kontinuierlich mit möglichst geringem Aufwand weiterentwickelt werden soll, ist der sehr geringe Performance Nachteil zu vernachlässigen.

In dieser Anwendung stellen die Manager Klassen die technisch orientierten Subsysteme dar, die in der *Fassade* Klasse unter der Schnittstelle *ApplicationCore* zusammengefasst werden. Außer in der *Fassade* Klasse wird nirgendwo in der Anwendung auf die Manager zugegriffen. Ändert sich beispielsweise die Schnittstelle eines bestimmten Managers, so wirken sich diese Änderungen nur auf die Fassade Klasse aus und nicht auf den Rest der Anwendung.

Die *Fassade* Klasse ist zentraler Punkt im System und kann als Singleton über das *Application* Interface aufgerufen werden. Dieser SPoC wird dafür genutzt, Listener zu registrieren, die beim Auftreten von bestimmten Events benachrichtigt werden.

4.2.6 Model-View-Controller Entwurfsmuster

Die Grundidee des Model-View-Conroller (MVC) Entwurfsmusters wurde von Trygve Reenskaug eingeführt und existiert bereits seit 1979. Das Ziel des Entwurfsmusters ist es, eine interaktive Anwendung in drei Komponenten mit unterschiedlichen Aufgaben zu unterteilen (vgl. Buschmann u. a. (1996), S. 123 ff).

- *Model*
  Das Model enthält die Daten, die dem Benutzer dargestellt werden sollen und die Geschäftslogik die diese Daten verändert. Es ist unabhängig von View und Controller und hängt somit weder von der Repräsentation der Daten noch von den Benutzereingaben ab. Bei einer Änderung des Models werden alle View Komponente die dem Model zugeordnet sind über die Änderung informiert, so dass die Daten dort aktualisiert werden können.

- *View*
  Die View Komponente präsentiert dem Benutzer die Daten aus dem Model. Dabei kann es mehrere verschiedene Views für ein Model geben.

- *Controller*
  Die Controller Komponente ist an eine View Komponente gekoppelt und behandelt die Benutzereingaben die über die View Komponente getätigt werden. Außerdem kennt der Controller das Model der View Komponente und kann somit anhand des Benutzerinputs das Model manipulieren. Die Kommunikation zwischen View und Controller findet meist über Events statt.

Durch die Trennung der Funktionalitäten in diese drei Komponenten, wird einer hoher Grad an Flexibilität erreicht, welcher unterschiedliche Ansichten (Views) von bestimmten Inhalten ermöglicht. Diese Views können alle auf einem Model basieren, wodurch Zugriffe und Änderungen an den Daten zentral stattfinden. Zusätzlich entsteht somit eine Trennung zwischen der Anwendungslogik und der Darstellung der Anwendung.

Während der Vorbereitung und Recherche zu dem Thema MVC in Kombination mit Android stellte sich heraus, dass dieses Thema zum jetzigen Zeitpunkt noch keine fachliche Lektüre aufweist. Jedoch gibt es durchaus unterschiedliche Meinungen, in der sich mit Android befassenden Comunity (vgl. www.therealjoshua.com und www.stackoverflow.com).

Das größte Problem stellt dabei die Einordnung der von Android vorgegebenen Konstrukte in die Architektur des MVC Entwurfsmusters dar. Dies betrifft besonders die Activity, da sie

Aufgabenbereiche unterschiedlicher Komponenten in sich vereint. In viele Anwendungen wird sie als Teil des Views benutzt, indem sie verschiedene Layout Objekte erstellt und diese mit den benötigten Daten versieht. Zugleich dient sie jedoch auch als Einstiegspunkt in die Anwendung. Deshalb verfügt sie über Methoden aus dem Activity Lifecycle (vgl. Abbildung 2.3), welche normalerweise nicht in einem View Objekt untergebracht werden. Das führt dazu, dass unter Android die Erzeugung der MVC Struktur in der *onCreate* Methode der Activity stattfinden muss. Dies wiederum verstößt jedoch gegen die Unabhängigkeit von Model und View. Außerdem müssen Aufgaben, wie das Registrieren von *BroadcastReceivern* und das Verarbeiten von Intents in der Activity ausgeführt werden, welche normalerweise im Model passieren. Deshalb ist eine wirklich saubere Trennung zwischen den einzelnen Komponenten unter Android nicht so einfach möglich.

Trotzdem habe ich mich in diesem Projekt bewusst für eine Verwendung des MVC Entwurfsmusters entschieden. Zusammen mit der Trennung der Android und Core-Komponente entsteht dadurch eine sehr flexible Architektur, speziell im Hinblick auf die nicht-funktionalen Anforderungen NFA10 und NFA11 (vgl. Tabelle 3.2.3) ermöglicht.

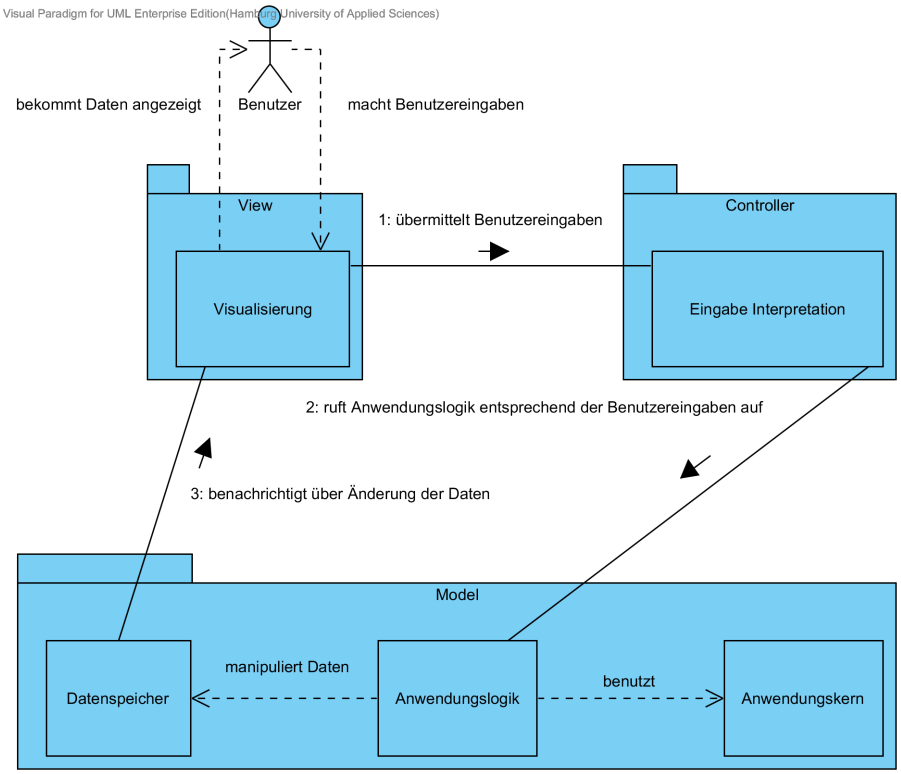


Abbildung 4.8: Komponentenübersicht der MVC Architektur

4.2.6.1 Views der Anwendung

Mit Hilfe der funktionalen Anforderungen wurden Prototypen für die Views entworfen. Da unter Android die Layouts entweder direkt im Java Code oder aber in separaten XML Dateien entworfen werden können, war es möglich die Prototypen losgelöst von dem Rest der Anwendung zu erstellen.

Um auch die nicht-funktionale Anforderung NFA11 im Falle von anstehenden Änderungen an den Views abzudecken, wurden die farblichen Einstellungen und bestimmte Eigenschaften der View Elemente in sogenannte *Themes* ausgelagert. Dadurch erhält man die Möglichkeit die Änderungen von diesen Einstellungen für die gesamte Applikation einheitlich und zentral durchzuführen.

Anhand der Prototypen konnte festgelegt werden, welche Daten das jeweils passende Model bereitstellen müssen. In dem folgenden Abschnitt der Arbeit möchte ich die, für den ersten Prototypen der Anwendung, entstandenen Views kurz vorstellen. Anhand des *Favoriten* Views, soll erklärt werden, wie die Anforderungen an das dazugehörige Model definiert wurden.

FavoritesView
In dieser Ansicht sollen die Informationen der vom Benutzer eingerichteten Favoriten in einer kompakten Form angezeigt werden. Darüber hinaus soll die Möglichkeit bestehen direkt zu der detaillierten Ansicht eines ausgewählten Favoriten zu gelangen. Der *FavoritesView* besteht im wesentlichen aus vier, in der Abbildung 4.2.6.1 markierten, Komponenten:

1. Der Navigationsleiste mit Zugriff auf die Teamsuche, Aktualisierung und Einstellungen der Anwendung (von links nach rechts). Das *FavoritesModel* benötigt dafür eine Methode zum auszulösen der Aktualisierung (vgl. Abbildung 4.10 *startUpdates*).

2. Einer Überschrift für die Ansicht in der sich der Benutzer gerade befindet. Hierfür werden keine Informationen aus aus dem Model benötigt, da der Text der Überschrift in dem jeweiligen View festgelegt werden kann.

3. Eine Liste mit jeweils einem *TeamItem* für jeden angelegten Favorit. TeamItems dienen zur kompakten Darstellung der wichtigsten Informationen einer Mannschaft. Dafür werden folgende Funktionen im *FavoritesModel* benötigt:
   - eine Liste mit jeweils einem TeamItemModel zu jedem Favoriten. (vgl. Abbildung 4.10 *getFavorites*).
   - eine Mannschaft zu der Liste der Favoriten hinzuzufügen (vgl. Abbildung 4.10 *addFavorite(Team team)*).
   - einen bisherigen Favoriten aus der Liste zu entfernen (vgl. Abbildung 4.10 *removeFavorite(Team team)*).

4. Eine sogenannten *UpdatePanel*, welches den Status eines Aktualisierungsvorgangs anzeigt, solange dieser ausgeführt wird. Dafür wird in dem *FavoritesModel* ein *UpdatePanelModel* benötigt, das die Daten des aktuellen Updateverlaufes enthält.

TeamItemView
Diese Ansicht gehört zu dem *TeamItem*, welches bereits im Abschnitt über die *FavoritesView* angesprochene wurde. In der *FavoritesView*, dient die *TeamItem* dazu, dem Benutzer einen Überblick über die wichtigsten Informationen seiner Favoriten zu ermöglichen. Außerdem kann er durch eine Aktion mit dem *TeamItem* eine Navigationsleiste öffnen, über die er Zugriff auf verschiedene Funktionen erhält. Die Informationen und die zur Verfügung gestellten Funktionen dieser Ansicht habe ich in der anschließenden Aufzählung zusammengefasst.

1. Die eindeutige Bezeichnung der Mannschaft durch das Vereinskürzel, den Rang der Mannschaft im Verein und der geschlechtlichen Ausrichtung. Diese Informationen können alle aus dem Team-Objekt ausgelesen werden (vgl. Abbildung 4.10 *getTeam* und *getGender*).

2. Die aktuelle Platzierung der Mannschaft in der laufenden Saison (vgl. Abbildung 4.10 *getRang*).

3. Eine Zusammenfassung des letzten Spieltages bestehend aus dem Datum und den Begegnungen mitsamt der dem Verband vorliegenden Ergebnisse. Dafür benötigt das TeamItemModel ein Matchday-Objekt des letzten Spieltages, welches das Datum enthält. Außerdem wird ein MatchdayTableModel benötigt, welches mit dem Matchday-Objekt initialisiert wird (vgl. Abbildung 4.10 *getLastMatchdayTableModel*). Das MatchdayTableModel enthält alle Informationen zur Darstellung der Begegnungen.

4. Eine Zusammenfassung des nächsten Spieltages bestehend aus dem Datum, der Adresse des Austragungsortes, so wie den angesetzten Begegnungen. Dafür werden ähnliche Daten wie für die Anzeige des letzten Spieltages gebraucht, jedoch handelt es sich hierbei um das Matchday-Objekt des nächsten Spieltages (vgl. Abbildung 4.10 *getNextMatchdayTableModel*).

5. Die Navigationsleiste mit den folgenden Funktionen:
   - Öffnet die TeamDetails Ansicht für diese Mannschaft.
   - Öffnet eine GoogleMaps Navigation zum Austragungsort des nächsten Spieltages.
   - Entfernt diese Mannschaft aus der Liste der Favoriten.

SearchTeamView
Diese Ansicht gibt dem Benutzer die Möglichkeit, gezielt nach bestimmten Mannschaften zu suchen. Anschließend können diese dann zu den Favoriten hinzugefügt werden oder es können die *TeamDetails* zu der Mannschaft angezeigt werden. Dabei stehen dem Benutzer für die Suche zwei verschiedene Möglichkeiten zu Verfügung, mit denen er zu seiner ausgewählten Mannschaft gelangen kann. Beide Varianten setzten unterschiedliches Vorwissen des Benutzers voraus, so dass er in der Lage ist mit so wenig Informationen wie möglich die gesuchte Mannschaft zu finden.

In der einen Variante kann der Benutzer eine Teil des Namens der Mannschaft oder des Vereinskürzels eingeben und die nötigen Angaben werden automatisch ergänzt. In der anderen Variante muss der Benutzer die Angaben von 2-5 eigenständig ausfüllen. Dabei ist die Mannschaftssuche folgendermaßen aufgebaut:

1. Ein Suchfeld mit Autovervollständigung, was die Suche nach dem Vereinsnamen oder dem Vereinskürzel unterstützt.
2. Ein Spinner um die Liga auszuwählen in der die gesuchte Mannschaft spielt.
3. Ein Spinner um geschlechtliche Ausrichtung der gesuchten Mannschaft auszuwählen.
4. Ein Spinner für die Staffel der gesuchten Mannschaft.
5. Ein Spinner mit der Liste aller Mannschaften, welche die ausgewählten Kriterien erfüllen.
6. Ein Button um die ausgewählte Mannschaft zu den Favoriten hinzuzufügen.
7. Ein Button um die *TeamDetails* für die selektierte Mannschaft zu öffnen.

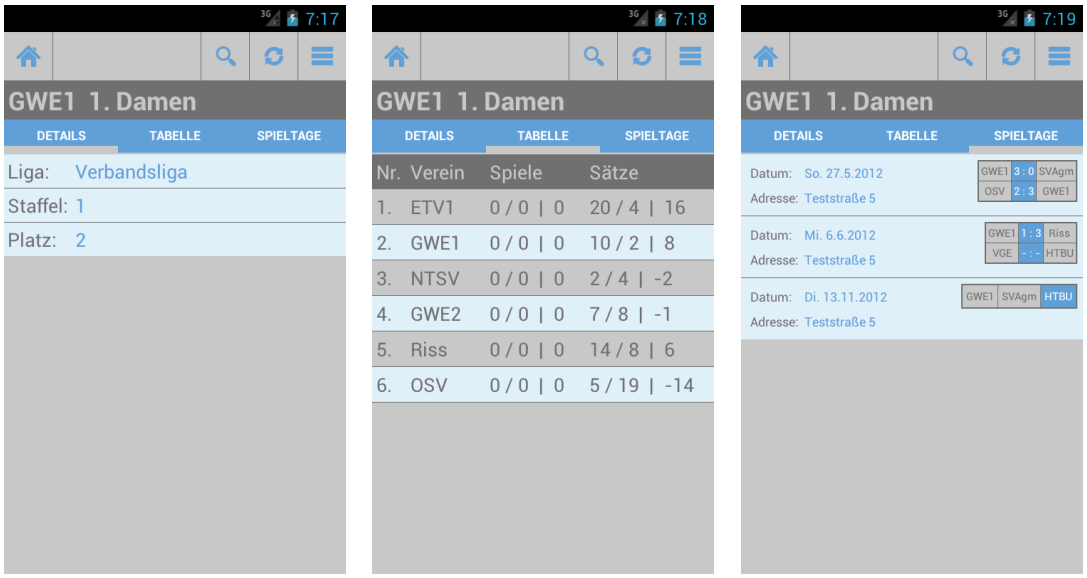
TeamDetails

In dieser Ansicht sollen alle Details einer Mannschaft vereint werden. Dafür werden sogenannte *Android Fragments* verwendet, die als unabhängige Bausteine überall in der Anwendung eingebaut werden können. Jedes Fragment deckt dabei ein bestimmtes Informationsgebiet ab. Durch eine eigene Navigation kann dann zwischen den verschiedenen Fragmenten gewechselt werden. Falls später weitere Details zu einer Mannschaften hinzukommen sollten, müssen nur die zusätzlichen *Android Fragmente* erstellt und in die Ansicht mit eingefügt werden. Der Prototyp enthält folgende drei Fragmente:

1. *InformationFragment*
   Hier werden einfache Informationen über die Mannschaft angezeigt. Dazu gehören der Vereinsname, die Liga, die Staffel oder die Platzierung der jeweiligen Mannschaft.

2. *StandingsFragment*
   Dieses Fragment zeigt die Tabelle für eine Mannschaft an. Die Informationen einer Spalte entsprechen dabei der funktionalen Anforderung FA1.

3. *MatchdaysFragment*
   Mit Hilfe dieses Fragments, wird eine Liste aller Spieltage für eine Mannschaft angezeigt. Die angezeigten Informationen eines Spieltages entsprechen der funktionalen Anforderung FA2.



(a) InformationFragment (b) StandingsFragment (c) MatchdaysFragment

Abbildung 4.9: Android Fragment Views

4.2.6.2 Models der Anwendung

Anhand der View Prototypen konnten die Funktionen und Daten für die entsprechenden Model Klassen identifiziert werden. Mit diesen Informationen wurde anschließend die Abbildung 4.10 erstellt, welche eine Übersicht über die Funktionen und Abhängigkeiten dieser Klassen ermöglicht. Diese Übersicht zeigt alle Model Klassen der Core Komponente, die eventuell in den konkreten Implementierungen für die Plattformen erweitert werden müssen.

4.2.6.3 Controllers der Anwendung

Die Controller implementieren wie die Anwendung auf Benutzereingaben in den Views reagiert. Deshalb sind sie sehr stark an die Views gekoppelt. Welche Funktionen ein Controller anbieten muss kann also anhand der View Prototypen herausgefunden werden. Eine exakte Formulierung der Schnittstellen ist jedoch nicht ohne weiteres möglich, da diese zu sehr von den konkreten Implementierungen der Views abhängen. Die wichtigsten Funktionen der Controller, die zu den vorgestellten Views gehören, sind hier aufgelistet.

- *FavoritesController*
  - eine Funktion, die bei einem Klick auf ein *TeamItem* die entsprechende *onTeamItemClicked* Funktion des Objektes aufruft.
- *TeamItemController*
  - ein Funktion, welche die *TeamDetails* zu der ausgewählten Mannschaft anzeigt.
  - eine Funktion mit der sich eine Google Maps Navigation zum Austragungsort des nächsten Spieltags öffnet.
  - eine Funktion, welche die ausgewählte Mannschaft aus den Favoriten entfernt.
- *SearchTeamController*
  - eine Funktion um beim Selektieren einer Liga, der geschlechtlichen Ausrichtung, der Staffel oder einer Mannschaft die Informationen im Model aktualisieren.
  - eine Funktion um die ausgewählte Mannschaft zu den Favoriten hinzufügen.
  - eine Funktion um die *TeamDetails* der ausgewählten Mannschaft anzuzeigen.
- *mehrere Controller*
  - Funktionen die bei Benutzung der oberen Navigationsleiste die entsprechenden Activities starteten oder bestimmte Ereignisse auslösen, wie beispielsweise die Aktualisierung der Anwendung.

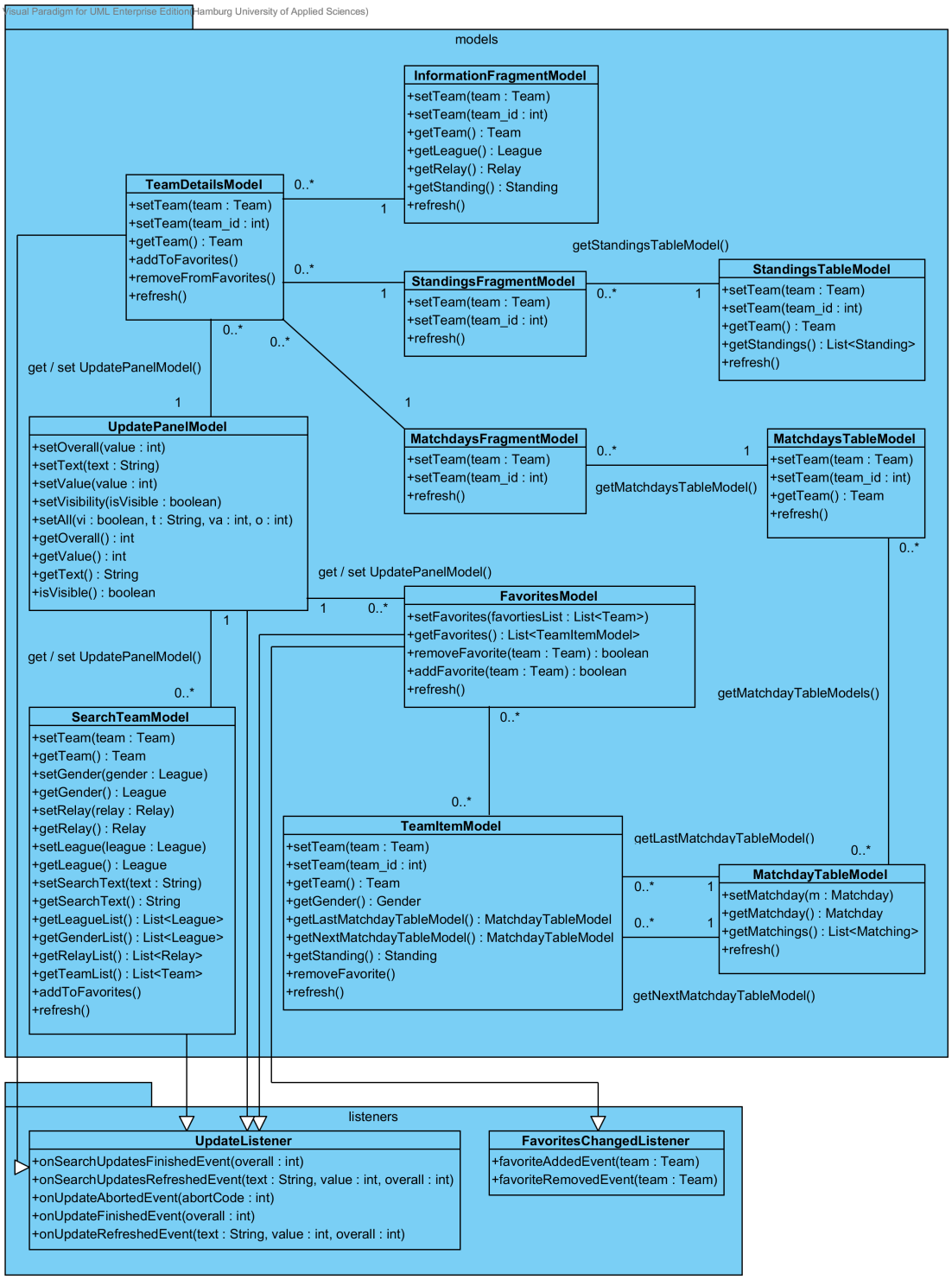


Abbildung 4.10: Modelle der Core Komponente

4.3 Fazit

In der Designphase wurde sichergestellt, dass die Architektur der Anwendung alle definierten Anforderungen unterstützt. Durch die Trennung in eine auf Standard Java basierende Core Komponente und eine plattformabhängige Android Komponente wurde eine erste technische Struktur für die Anwendung geschaffen. Diese wurde durch weitere Unterteilungen in verschiedene Packages weiter detailliert. Durch die Verwendung der drei beschriebenen Entwurfsmuster wurde der Architektur eine gewisse Flexibilität gegeben, welche Änderungen und Ergänzungen an den Komponenten vereinfachen soll.

Mit Hilfe der sehr aufwändigen Planung und den in der Designphase entstandenen Artefakten, konnte ein guter Überblick über die komplette Architektur hergestellt werden. In dieser Phase wurde bereits viel Vorarbeit für die Realisierungsphase geleistet. Dadurch wurde die Realisierungsphase deutlich verkürzt und bei einer korrekten Implementierung der Schnittstellen ist sichergestellt, dass die Kommunikation zwischen den Komponenten funktioniert.

Eine besondere Herausforderung bestand darin, eine Kombination aus Entwurfsmustern zu finde, die sich vorteilhaft in die Anwendung einbauen ließen. Dabei musste darauf geachtet werden, dass die Entwurfsmuster am Ende auch tatsächlich eine positive Auswirkung auf das Ergebnis haben. Speziell bei dem MVC Entwurfsmuster entstanden einige Schwierigkeiten und es musste abgewägt werden, ob es sich überhaupt sinngemäß in die Anwendung einbauen lässt.

5 Realisierung

In dem folgenden Kapitel werden die technischen Umsetzungen der wichtigsten Designentscheidungen kurz aufgezeigt und mittels einiger Source Code Auszüge erläutert. Außerdem werden zusätzlich noch die angewandten Testmethoden vorgestellt und eine abschließende Evaluation hinsichtlich der definierten Anforderungen durchgeführt.

5.1 Technische Umsetzung

5.1.1 Kommunikation zwischen Anwendung und externer Datenbank

Aus der Abbildung 4.1 geht hervor, dass sowohl eine externe MySQL Datenbank auf dem Webserver, als auch eine interne SQLite Datenbank auf dem Gerät benötigt wird. Im folgenden Text wird die Realisierung der Kommunikation zwischen Anwendung und externer Datenbank vorgestellt, indem genauer auf die Vorteile und Nachteile der gewählten Umsetzung eingegangen wird.

Um die Anwendung weitestgehend unabhängig von der ausgewählten externen Datenbank zu machen, wurde für das Projekt eine Representational State Transfer (REST) Schnittstelle mittels JavaScript Object Notation (JSON) implementiert. In der Umsetzung wird für jede Anfrage der Anwendung an die Datenbank eine PHP Datei erstellt. In diesem Fall liegen die Dateien auf dem Server, auf dem auch die externe Datenbank installiert ist. Über eine HTTP Anfrage kann jede Anwendung über das Internet diese Dateien mit bestimmten Parametern aufrufen. In diesen Dateien wird dann ein Query (Zeile 16) an die Datenbank abgesetzt und die Ergebnisse in ein gültiges JSON Objekt umgewandelt (Zeile 36, 44, 52). Dieses JSON Objekt ist dann der Rückgabewert der HTTP Anfrage und kann in den Anwendungen weiter verarbeitet werden.

```
1  <?php
2
3  if (isset($_GET["id"])) {
4      $id = $_GET['id'];
5
6      // get an address from addresses table
7      $result = mysql_query("SELECT id, street_name, street_nr FROM
           addresses WHERE id = $id");
8
9      if ($result) {
10
11         $result = mysql_fetch_array($result);
12
13         $address = array();
14         $address["id"] = $result["id"];
15                 $address["street_name"] = $result["street_name"];
16                 $address["street_nr"] = $result["street_nr"];
17
18         // echoing JSON response
19         echo json_encode($response);
20
21     } else {
22         // no address found
23         $response["success"] = 0;
24         $response["message"] = "No address found";
25
26         // echo no users JSON
27         echo json_encode($response);
28     }
29 ...
30 ?>
```

Listing 5.1: REST Zugriff auf die Datenbank

In diesem Beispiel wird eine Adresse anhand ihrer *address_id* aus der Tabelle *addresses* abgefragt und durch den Befehl *json_encode* (Zeile 16, 24) als JSON Objekt zurückgegeben. Der Vorteil dieser REST Implementation besteht darin, dass sie eine wohldefinierte Schnittstelle für alle Anwendungen zu Verfügung stellt. Sollte sich später einmal der Datenbanktyp geändert werden, müsste lediglich der Datenbankzugriff in den PHP Dateien angepasst werden. Dadurch wirken sich Änderungen nicht auf die Anwendungen aus, die diese Schnittstelle benutzen.

Um eine HTTP Anfrage von der Anwendung aus abzusetzen, wurde eine zusätzliche Klasse *JSONParser* erstellt. Mit Hilfe einer URL und Parametern, die als Argumente übergeben werden,

wird die Anfrage gesendet und das anschließende Parsen des JSON Objektes durchgeführt. Dafür wird geprüft, ob das JSON Objekt eine bestimmte Struktur besitzt. Wenn das Objekt wohlgeformt ist, wird es als Ergebnis der Anfrage zurückgegeben. Ansonsten findet in der Klasse die Behandlung der entsprechenden Exceptions statt.

Die Weiterverarbeitung der JSON Objekte findet in der *SyncManager* Klasse statt, welche die Informationen ausliest und in die passenden *BasicObjects* der Anwendung überführt.

Das JSON Format eignet sich besonders für Geräte mit limitierten Ressourcen, da es nur einen geringen zusätzlichen Overhead enthält.

5.1.2 Synchronisation zwischen interner und externer Datenbank

Um auf den jeweiligen Endgeräten immer die aktuellen Informationen anzeigen zu können, wird eine Synchronisation zwischen der externen und der internen Datenbank benötigt. Optimal wäre es, wenn sich zu jedem Zeitpunkt der gleiche Datenbestand auf beiden Datenbanken befindet. Dies kann in diesem Projekt jedoch nicht umgesetzt werden, da eine ständige Verbindung zwischen den betroffenen Datenbanken aufrecht erhalten werden muss. Der externen Datenbank muss jedes Endgerät bekannt sein, um es über Datenänderungen sofort zu informieren. Darüber hinaus muss gesichert sein, dass die Verbindung zwischen den Datenbanken immer aufrecht erhalten werden kann. Diese Voraussetzungen sind in diesem Anwendungsfall jedoch nicht gegeben, da die mobilen Endgeräte nicht immer eingeschaltet sind oder über eine bestehende Internetverbindung verfügen. Außerdem soll keine Registrierung zur Benutzung dieser Anwendung nötig sein, welche jedoch für eine eindeutige Zuordnung der Endgeräte benötigt würde.

Allerdings ist es für diese Anwendung auch nicht zwingend erforderlich, dass der Datenbestand zu jedem Zeitpunkt synchron ist. Wichtig ist lediglich der Zeitpunkt, in dem die Informationen tatsächlich vom Benutzer angefordert werden. Sobald der Benutzer die Anwendung öffnet, muss also sichergestellt werden, dass die Informationen dem aktuellen Stand der externen Datenbank entsprechen. Deshalb wird beim Starten der Anwendung oder in regelmäßigen Abständen geprüft, ob neue Daten in der externen Datenbank zur Verfügung stehen, die dann gegebenenfalls aktualisiert werden können.

Um den Prüfvorgang auf neue Daten so schnell wie möglich zu machen, wurde jeweils eine zusätzliche Tabelle *Timestamps* in den beiden Datenbanken hinzugefügt. Sie enthält für jede weitere Tabelle in der Datenbank eine Zeile, mit dem Tabellennamen und einem Timestamp der letzten Aktualisierung dieser Tabelle. Dadurch müssen bei der Suche nach neuen Daten nicht alle Tabellen komplett durchlaufen werden, sondern nur die Tabelle *Timestamps*. Beim

Durchlaufen der Tabelle wird jeweils der Timestamp, mit dem Timestamp der entsprechenden Zeile auf der anderen Datenbank verglichen. Erst bei einer Abweichung wird in der Tabelle auf der externen Datenbank nach allen Einträgen gesucht, die älter sind als der Timestamp der internen Datenbank. Die Tabellen werden deshalb erst dann vollständig durchlaufen, sobald wirklich eine Aktualisierung stattgefunden hat. Mit Hilfe des *Timestamp* Attributes in jeder Tabelle wird vermieden, dass alle Attribute verglichen werden müssen, um eine Aktualisierung zu erkennen.

Sobald die neuen Daten identifiziert wurden werden sie aus der externen Datenbank abgerufen und in die interne Datenbank geschrieben (vgl. Abschnitt 5.1.1). Dabei musste beachtet werden, dass der eindeutige Identifier der Einträge jeweils in beiden Datenbanken gleich ist, weil alle Fremdschlüssel Beziehungen über diesen Identifier laufen.

Die Updateroutine läuft in einem separaten Prozess, damit die Anwendung währenddessen nicht blockiert wird. Realisiert ist dies unter Android durch einen Service der gestartet wird und dann die entsprechenden Methoden der *SyncManager* Klasse aufruft.

5.1.3 Model-View-Controller

Das im Abschnitt 4.2.6 bereits angesprochene MVC Entwurfsmuster ist ein zentraler Punkt in der Architektur der gesamten Anwendung. Daher möchte ich in das Zusammenspiel zwischen der Core und der Android-Komponente, in Kombination mit dem MVC Entwurfsmuster, anhand eines ausführlichen Beispiels erläutern.

Sobald die App auf dem Handy gestartet wird, öffnet sich zuerst die Klasse *Application* aus der Android Komponente und sie durchläuft den Activity-Lifecycle (vgl. Abschnitt 2.1.2.4). Die *Application* Klasse implementiert das in der Core Komponente befindliche *Application* Interface, welches die Methode *getApplicationCore* für die Anwendung bereitstellt.

In der onCreate Methode werden die benötigten Manager Klassen instantiiert und damit ein neues Objekt der Klasse *Fassade* aus der Android Komponente erstellt. Diese Klasse erweitert die *Fassade* Klasse aus der Core Komponente um die speziell für die Android Implementierung benötigten Funktionen. Beide Fassade Klassen implementieren das jeweilige ApplicationCore Interface aus ihren Komponenten.

Nach Durchlaufen der *onCreate* Methode wird die in der *getHomeActivityClass* Methode angegebene Activity gestartet.

```
1  ...
2  @Override
3  public void onCreate() {
4    DatabaseHelper dbHelp = new DatabaseHelperImpl(this);
5    Factory.init(dbHelp);
6    DatabaseManager dbManager = new DatabaseManagerImpl(dbHelp);
7    NotificationManager notiManager = new NotificationManagerImpl(this);
8    NetworkManager netManager = new NetworkManagerImpl(this);
9    ServiceManager servManager = new ServiceManagerImpl(this);
10   SettingsManager setManager = new SettingsManagerImpl(this, dbManager)
         ;
11   SyncManager syncManager = new SyncManagerImpl(dbManager, netManager);
12   mApplicationCore = new Fassade(this, dbManager, notiManager,
         netManager, servManager, syncManager, setManager);
13   mApplicationCore.setAutoUpdates();
14 }
15
16 @Override
17 public Class<?> getHomeActivityClass() {
18   return FavoritesActivity.class;
19 }
20
21 @Override
22 public ApplicationCore getApplicationCore() {
23   return mApplicationCore;
24 }
25 ...
```

Listing 5.2: Die Application Klasse

Da Activities, wie im Abschnitt 4.2.6 erklärt, teilweise auch Aufgaben des Views übernehmen, wird ein entsprechender Controller benötigt. Deshalb wird in der Activity ein Interface erstellt, welches die benötigten Methoden für die Aktionen in der Activity definiert. Der verwendete Controller muss anschließend dieses Interface implementieren.

```
1 ...
2   public static interface Controller {
3     public void onUpdateItemClicked();
4     public void onSearchItemClicked();
5     public void onSettingsItemClicked();
6   }
7   ...
```

Listing 5.3: Die Favorites Activity - Controller

In der anschließenden *onCreate* Methode wird mit Hilfe der privaten Methode *setupMVC* ein Favorites Objekt erstellt (vgl. Listing 5.4). Es dient im wesentlichen dazu, den größten Teil der Logik des MVC Entwurfsmusters aus der Activity zu extrahieren. Dafür wird der *ApplicationCore* und der Context (vgl. Abschnitt 2.1.1.3) der Activity benötigt. Anschließend muss der Favorites Activity noch das Model, der Controller und die View, welche beim Erstellen des Objektes entstanden sind, zugewiesen werden. Bei der Zuweisung des Models registriert sich die Activity bei dem Model als Listener, um über bestimmte Events informiert zu werden.

```
1  ...
2    @Override
3    public void onCreate(Bundle savedInstanceState) {
4      setupMVC();
5      initActionBar();
6    }
7
8    private void setupMVC() {
9      final Application application = (Application) getApplication();
10     mFavorites = new Favorites(application.getApplicationCore(), this);
11     setModel(mFavorites.getModel());
12     setController(mFavorites.getController());
13     setActionBarContentView(mFavorites.getView());
14   }
15
16   private void setModel(final FavoritesModel model) {
17     mModel = model;
18     mModel.addListener(UpdateEvent.UPDATE_REFRESHED,
           updateRefreshedListener);
19     mModel.addListener(UpdateEvent.UPDATE_FINISHED,
           updateFinishedListener);
20     ...
21   }
22
23   private void setController(final FavoritesController controller) {
24     mController = controller;
25   }
26   ...
```

Listing 5.4: Die Favorites Activity - MVC

In der Favorites Klasse werden durch den Aufruf des Konstruktors die benötigten Komponenten für das MVC Entwurfsmuster erstellt (vgl. Listing 5.5). Durch ein Model, ein View und den dazugehörigen Controller entsteht ein Konstrukt, wie es in der Abbildung 4.8 beschrieben ist. Ansonsten enthält diese Klasse nur Getter und Setter Methoden für Model, View und Controller. Da sich nach der Definition des MVC Entwurfsmusters die Anwendungslogik in dem Model befindet, wird für dessen Erstellung der *ApplicationCore* benötigt, um auf die entscheidenden Funktionen der Anwendung zuzugreifen.Da der Context für viele Funktionen von Android benötigt wird, steht er sowohl im Model, als auch im View zur Verfügung.

```
1  ...
2  public Favorites(Application application, Context context) {
3      mModel = new FavoritesModel(application, context);
4      mView = (FavoritesView) View.inflate(context, R.layout.
           favorites_activity, null);
5      mController = new FavoritesController(mModel);
6      mController.setModel(mModel);
7      mView.setController(mController);
8      mView.setModel(mModel);
9    }
10   ...
```

Listing 5.5: Das Favorites Objekt als Container für MVC

Das Model was erstellt wurde, beinhaltet die Informationen, die in dem View dargestellt werden sollen. Sollte sich eine Information ändern, schickt es ein Event an seine Listener, die nach dem Empfangen des Events die Informationen aus dem Model neu auslesen. In diesem Beispiel wird ein Favorit aus der Liste entfernt. Mit dem Aufruf *notifyChange* in Zeile werden dann alle Listener des Models über das *FavoritesEvent.FAVORITE_REMOVED* Event informiert.

```
1  public boolean removeFavorite(final Team team) {
2      if (favorites.containsKey(team)) {
3        favorites.remove(team);
4        favoritesList = null;
5        notifyChange(FavoritesEvent.FAVORITE_REMOVED);
6        return true;
7      } else {
8        return false;
9      }
10   }
```

Listing 5.6: Das FavoritesModel

Der View besteht im wesentlichen aus zwei verschiedenen Komponenten. In einer XML Datei werden die zur Verfügung stehenden Elemente positioniert (vgl. Abschnitt 2.1.1.4). Über den *android:id* Eintrag (Zeile 19) kann das *ListView* Element, über den *RessourcenManager* (vgl. Abschnitt 2.1.1.4), angesprochen werden.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <bachelor.haw.main.android.views.FavoritesView xmlns:android="http://
       schemas.android.com/apk/res/android"
3      style="?attr/hvbv_favorites"
4      android:layout_width="match_parent"
5      android:layout_height="match_parent"
6      android:orientation="vertical" >
7      <LinearLayout
8          style="?attr/hvbv_favorites_headline"
9          android:layout_width="fill_parent"
10         android:layout_height="wrap_content"
11         android:orientation="horizontal" >
12         <TextView
13             style="?attr/hvbv_favorites_headline_text"
14             android:layout_width="wrap_content"
15             android:layout_height="wrap_content"
16             android:text="@string/favorites_headline" />
17     </LinearLayout>
18     <ListView
19         android:id="@+id/favorites_list"
20         android:layout_width="match_parent"
21         android:layout_height="match_parent"
22         android:layout_weight="1" />
23 </bachelor.haw.main.android.views.FavoritesView>
```

Listing 5.7: Das FavoritesView

In einer Java Klasse, die das Container Element aus der XML Datei erweitert, werden anschließend die View Elemente mit den Informationen aus dem Model gefüllt. Beim erstellen des Views wird die *onFinishInflate* Methode aufgerufen, in der eine Referenz auf die Liste aus der XML Datei erstellt wird. Sobald dem View ein Model zugewiesen wird, registriert es seine Listener beim Model und es wird die *init* Methode aufgerufen. Hier werden dann die Informationen aus dem Model an die Referenz der Liste weitergegeben. Bei einem *FavoritesEvent.FAVORITE_REMOVED* Event wird die Methode *bindFavoritesList* erneut aufgerufen und die Informationen werden noch einmal neu aus dem Model geladen und an die Referenz gebunden.

```
1  public class FavoritesView extends LinearLayout {
2    ...
3    public void setModel(final FavoritesModel model) {
4      if(mModel != null) removeListeners();
5      mModel = model;
6      addListeners();
7      init();
8    }
9    ...
10   private void init() {
11     bindUpdatePanel();
12     bindFavoritesList();
13   }
14
15   private void bindFavoritesList() {
16     mList.setAdapter(mModel.getFavoritesAdapter());
17     mList.setOnItemClickListener(onItemClickListener);
18     mList.invalidate();
19   }
20   ...
21   @Override
22   protected void onFinishInflate() {
23     super.onFinishInflate();
24     mList = (ListView) findViewById(R.id.favorites_list);
25   }
26
27   private final EventListener favoritesListChangedListener = new
         EventListener() {
28     @Override
29     public void onEvent(final Event event) {
30       bindFavoritesList();
31     }
32   };
33
34   private void addListeners() {
35     ...
36     mModel.addListener(FavoritesEvent.FAVORITE_REMOVED,
           favoritesListChangedListener);
37   }
38   ...
39 }
```

Listing 5.8: Das FavoritesView

Auf Grund der angesprochenen Problematik, bezüglich des MVC Entwurfsmusters unter der Verwendung von Android, mussten einige Funktionen des Views in der Activity implementiert werden. Dies betrifft zum Beispiel die Erstellung der Navigationsleiste durch die Methode *initActionBar* und die Weiterleitung der *onClick* Events an den Controller (vgl. Listing 5.9). Außerdem stehen in der Activity mit *onCreate* und *onDestroy* Methoden in denen jeweils die Listener für die Models hinzugefügt oder entfernt werden müssen. Dies ist notwendig, da ansonsten bei dem Schließen einer Activity durch das Betriebssystem, der Anwendungskern weiterhin versuchen würde nicht existente Listener weiterhin zu benachrichtigen. Außerdem kann in den Methoden *onPause* und *onResume*, der Zustand einer Activity durch eine Speicherung des Models gesichert oder wiederhergestellt werden. Dies ist zum Beispiel wichtig, wenn die Einstellungen des Benutzers in einer Activity beim Drehen des Displays nicht gelöscht werden sollen. Das liegt daran, dass beim Drehen des Displays die Activity neu aufgerufen wird. Da es in dieser Activity jedoch nichts zu Speichern gibt, sind die Methoden *saveState* und *restoreState* jeweils leer.

```
1  ...
2    public void initActionBar() {
3      setTitle(R.string.app_name);
4      addActionBarItem(Type.Search, R.id.action_bar_search);
5      mUpdateItem = (LoaderActionBarItem) addActionBarItem(Type.Refresh, R.
           id.action_bar_update);
6      addActionBarItem(Type.Settings, R.id.action_bar_settings);
7    }
8
9    @Override
10   public boolean onHandleActionBarItemClick(final ActionBarItem item,
         final int position) {
11     switch (item.getItemId()) {
12     case R.id.action_bar_search:
13       mController.onSearchItemClicked();
14       return true;
15     ...
16     default:
17       return super.onHandleActionBarItemClick(item, position);
18     }
19   }
20
21   @Override
22   public void onResume() {
23     super.onResume();
24     restoreState();
```

```
25   }
26
27   @Override
28   protected void onPause() {
29     super.onPause();
30     saveState();
31   }
32   ...
```

Listing 5.9: Die Favorites Activity - View Funktionen

Um auf die vom Model versendeten Events zu reagieren, werden bestimmte Listener benötigt. In der *setModel* Methode aus Listing 5.4 hat sich die Activity mit ihren Listenern für bestimmte Events registriert. Ein Beispiel für so ein Listener ist der updateRefreshedListener (vgl. Listing 5.10). Er wird ausgelöst, wenn von dem Model das *UpdateEvent.UPDATE_REFRESHED* Event ausgelöst wurde. Sobald dieses Event eintrifft, signalisiert ein Ladebalken in der Navigationsleiste, dass der Updateprozess ausgelöst wurde. Nach der Durchführung dieses Prozesses, verschickt das Model ein *UpdateEvent.UPDATE_FINISHED* Event, welches den Ladebalken wieder entfernt.

```
1  ...
2    private final EventListener updateRefreshedListener = new EventListener
         () {
3      @Override
4      public void onEvent(final Event event) {
5        mUpdateItem.setLoading(true);
6      }
7    };
8
9    private final EventListener updateFinishedListener = new EventListener
         () {
10     @Override
11     public void onEvent(final Event event) {
12       mUpdateItem.setLoading(false);
13     }
14   };
15   ...
```

Listing 5.10: Die Favorites Activity - Listener

Natürlich decken die aufgeführten Programmabschnitte nicht alle Funktionen der Anwendung ab und es werden auch nicht alle Verknüpfungen von den Models, den Views und den Controllern erläutert. Durch die wenigen ausgewählten Ausschnitte sollte jedoch erkenntlich sein, wie das MVC Entwurfsmuster in die Architektur eingebaut wurde.

5.2 Qualitätssicherung

Durch die in den folgenden Abschnitten beschriebenen Tests und Kennzahlen, wurde direkt zu Beginn des Projektes versucht eine gewisse Qualität des Ergebnisses sicherzustellen. Leider lässt sich nicht so einfach beurteilen, wie genau sich Qualität für das zu erwartende Ergebnis definieren lässt und wie sich die ermittelten Kennzahlen auf diese Qualität auswirken. Trotzdem helfen Tests und Kennzahlen dabei, die bisher erbrachte Arbeit zu hinterfragen und speziell im Hinblick auf die nicht-funktionalen Anforderungen zu bewerten.

5.2.1 Tests

Die Aufgabe der Anwendung besteht hauptsächlich darin, Informationen zu anzuzeigen und auf Benutzereingaben zu reagieren. Deshalb spielt die GUI eine wesentliche Rolle für die Qualität der Anwendung. Ein weiterer wichtiger Bestandteil der Anwendung ist die Synchronisation der Datenbanken, wofür eine Kommunikation über das Internet stattfinden muss. Zusätzlich ist der Anteil an konkreter Anwendungslogik (Formeln, bestimmte Abläufe) im Vergleich zu den restlichen Aufgaben (Darstellung und Interaktion) nur sehr sehr gering. Diese Kombination führt dazu, dass die normalerweise üblichen JUnit Tests oder das Überprüfen von Pre- und Postconditions, nur sehr schwer zu realisieren sind.

Aus diesem Grund habe ich mich bereits zu Beginn des Projektes für den Entwicklungsprozess des *Incremental prototypings* entschieden. Das bedeutet, dass die Applikation in vielen kleinen Iterationen erweitert wurde. Jede Iteration hatte jedoch als Zwischenergebnis einen funktionierenden Prototypen.

Im Laufe des Projektes veränderten sich die Ergebnisse immer weiter in Richtung des finalen Prototypen. Angefangen wurde mit einem horizontalen Prototyp, um die Funktionalität der entworfenen Views, mit dahinter liegenden Stubs, zu testen. Diese Prototypen wurden dann von einem sorgfältig ausgewählten Kreis von Nutzern getestet. Dabei wurde darauf geachtet, dass für diesen Kreis sowohl technisch erfahrenere, als unerfahrene Smartphone Besitzer ausgewählt wurden.

Im Anschluss daran änderte sich die Prototyp Art eher in Richtung eines vertikalen Prototypen. Nach und nach wurden die Activities erstellt und die Stubs wurden durch ihre tatsächliche Funktionalität ersetzt. Die Ergebnisse wurden erneut in regelmäßigen Abständen von den Benutzern getestet. Über die frei verfügbare Bug-Tracking Software *WebIssues* [1] konnten die Tester dann gefundene Fehler oder Verbesserungsvorschläge abgeben.

[1] Die Software ist frei verfügbar unter http://webissues.mimec.org/ (zuletzt aufgerufen am: 13.9.2012)

Ich habe mich für diese Art und Weise des Testens entschieden, da sie meines Erachtens nach die folgenden Vorteilen für das Projekt gebracht hat:

- Das Testen durch verschiedenen Personen, mit unterschiedlichen Android Geräten, konnte gerätespezifische Fehler aufdecken.
- Durch die Zusammenarbeit und das Feedback der potenziellen Nutzer bekommt man schneller ein Gespür dafür, wo die Interessen der Benutzer liegen und Prioritäten können somit besser gesetzt werden.
- Auf Grund des Drucks, der durch die regelmäßige Freigabe des Prototypen entsteht, arbeitet man in kleinen Etappen und vermeidet dadurch, dass sich Fehler durch das ganze Projekt ziehen.

5.2.2 Kennzahlen

Mit Hilfe der für Eclipse frei verfügbaren Plugins *CodePro AnalytiX* und *PMD*, wurde sich in regelmäßigen Abständen über den Status des Projektes hergestellt. Durch die Plugins können, direkt in der Eclipse Entwicklungsumgebung, sowohl Tipps zum aktuellen Aufbau des Programmcodes, als auch ausgewählte Kennzahlen für das Projekt angezeigt werden. Auf folgende Kennzahlen habe ich besonders geachtet:

5.2.2.1 zyklomatische Komplexität

Die zyklomatische Komplexität wurde 1976 durch Thomas J. McCabe eingeführt, weshalb sie auch McCabe-Metrik genannt wird. McCabe hatte erkannt, dass ungefähr die Hälfte der Entwicklungszeit einer Anwendung daraus besteht das Programm zu testen oder zu warten. Deshalb hat er eine mathematische Formel entwickelt, um die Komplexität einer Software Komponente zu messen und anschließend zu bewerten. Anhand der Bewertung kann dann festgestellt werden, ob diese Komponente noch mit einem akzeptablen Aufwand zu testen und zu warten ist. Die dabei entstandene Formel basiert auf dem Kontrollflussgraphen der jeweiligen Komponente. Bei dem Kontrollflussgraphen bilden die Anweisungen die Knoten und der Kontrollfluss zwischen den Anweisungen die Kanten (vgl. McCabe (1976)).

Formel

$M = e - n + 2p$

e: Anzahl Kanten im Graphen

n: Anzahl Knoten im Graphen

p: Anzahl der einzelnen Kontrollflussgraphen (ein Graph pro Funktion/Prozedur)

Ein großer Nachteil an diesem Verfahren ist jedoch, dass die Komplexität einzelner Anweisungen, wie beispielsweise die Schachtelungstiefe von Schleifen, nicht mit in die Wertung einbezogen werden.

Das Ergebnis gibt zwar die Komplexität der Funktion wieder, zur Bewertung jedoch muss noch eine obere Schranke festgelegt werden. McCabe ist der Ansicht, dass ab einem Wert von über 10 die Komplexität für einen Menschen zu hoch ist und er somit nicht mehr in der Lage ist die Funktion zu überschauen (vgl. McCabe (1976) S. 314). Daher empfiehlt er bei einem solchen Wert die Funktion aufzuteilen, um sie dadurch zu vereinfachen.

Meines Erachtens nach ist es sehr schwer zu definieren, ab wann eine Funktion als zu komplex anzusehen ist. Schließlich besitzt bereits eine einfache und übersichtliche *switch-case* Anweisung, mit 10 verschiedenen Möglichkeiten, eine Komplexität von 11. Allerdings kann die zyklomatische Komplexität dazu dienen, die vermeintlich einfachen Funktionen von den komplexen zu trennen. Anschließend kann man sich die Funktionen mit einem besonders hohen Wert noch einmal genauer anschauen. Die Entscheidung ob die Funktion anschließend überarbeitet werden sollte, liegt dann im Ermessen des Entwicklers.

In dem Ergebnis dieses Projektes besitzt nur eine Funktion eine zyklomatische Zahl die größer ist als 10. Dabei handelt es sich um eine solche *switch-case* Anweisung.

5.2.2.2 Commentratio

Die Commentratio ist ein sehr einfach zu berechnender Indikator. Sie beschreibt das Verhältnis der Kommentare im Code zu den gesamten Lines of Code (LoC) der Anwendung. Dadurch kann man sehr gut nachvollziehen, wie viel Arbeit man bereits in die Erstellung von Kommentaren investiert hat. Je größer das Verhältnis ist, desto wahrscheinlicher ist es, dass andere Projekt Mitarbeiter oder fremde Personen den geschriebenen Programmcode nachvollziehen können.

Jedoch sollte man auch bei diesem Verfahren das Ergebnis hinterfragen und nicht davon ausgehen, dass eine besonders hohe Commentratio auch automatisch einen gut dokumentierten Code impliziert. Theoretisch könnte das Verhältnis, durch einen besonders langen

Kommentar stark beeinflusst werden. Außerdem führt auch auskommentierter Teil des Programmcodes zu einem Anstieg der Commentratio, jedoch nicht zu einem besseren Verständnis des Anwendung.

Wenn man zur Berechnung das Plugin *CodePro AnalytiX* verwendet, kann man sich zusätzlich zu der Commentratio auch den Anteil der Javadoc Kommentare an den gesamten Kommentaren anzeigen lassen. Kombiniert man diese beiden Metriken und achtet auf die oben erwähnten Aspekte, erhält man einen guten Überblick darüber, ob eventuell etwas mehr Zeit in die Dokumentation des Programms investiert werden sollte. Denn eine sehr niedrige Commentratio bedeutet immer, dass nur sehr wenig Kommentare im Programmcode platziert wurden.

Der Prototyp des Projektes weist zur Zeit eine Commentratio von 5,1% auf und beinhaltet ca. 150 Javadoc Kommentare. Bei ca. 9000 LoC entspricht das ungefähr 450 Kommentaren.

5.3 Evaluation

Gegen Ende der Arbeit soll anhand der erzielten Ergebnisse noch einmal überprüft werden, welche der der definierten Anforderungen tatsächlich erfüllt wurden.

5.3.1 Erfüllung der Anforderungen

Ein Teil der Anforderungen lassen sich besonders gut anhand des erstellten Prototypens überprüfen. Dort wurden die Anforderungen FA1, FA2, FA3, FA4, FA5 und FA7 bereits umgesetzt und durch mehrere Benutzer getestet. Damit wurden zumindest alle wichtigen funktionalen Anforderungen umgesetzt. Die Anforderungen FA6 und FA8 konnten zwar noch nicht konkret realisiert werden, wurden jedoch bei der Planung und dem Design berücksichtigt. Sie können anschließend an das Projekt zur Anwendung hinzugefügt werden.

Die Anforderungen FA9, FA10, FA11 und FA12 haben den Designprozess der Anwendung maßgeblich beeinflusst. Der Teil der Anwendung, in dem diese Anforderungen tatsächlich realisiert werden, wurde noch nicht implementiert. Das liegt daran, dass dafür eine sehr enge Kooperation mit dem HVBV notwendig gewesen wäre, die in der begrenzten Projektzeit eventuell zu Problemen bei der Umsetzung der anderen Anforderungen geführt hätte. Aus diesem Grund wurde mit der neu erstellten MySQL Datenbank eine geeignete Schnittstelle für einen Adapter geschaffen, der die Entwicklung der Android App von dem Rest der Anwendung unabhängig machen sollte.

Um die Ergebnisse der Arbeit mit den Informationen des HVBV zu verknüpfen, wird noch die Komponente *Anpassung* auf dem Büroserver aus Abbildung 4.1 benötigt. Mit Hilfe eines

lesenden Zugriffs auf die MS Access Datenbank des HVBV müssen die erforderlichen Daten in die MySQL Datenbank portiert werden. Dadurch, dass der Zugriff nur lesend stattfindet, ist der Sicherheitsaspekt der berücksichtigt werden musste, sichergestellt. Da Daten weder gelöscht, geändert noch hinzugefügt werden können, werden die Anforderungen NFA2 und NFA3 auch erfüllt. Auf Grund der guten Java Database Connectivity (JDBC) Anbindungen für beide Datenbankentypen kann diese Komponente auch unter JAVA entwickelt und anschließend per Batch Programm ausführbar gemacht werden. Dieses Programm müsste nach dem Einfügen von neuen Informationen in die MS Access Datenbank entweder manuell oder automatisch ausgeführt werden. So könnten die in der Planung bereits berücksichtigten Anforderungen FA9 bis FA12 auch vollständig erfüllt werden. Die Erfüllung der nicht-funktionale Anforderung NFA1 ist teilweise auch an die Umsetzung der noch fehlenden Komponente gebunden und konnte daher noch nicht vollständig sichergestellt werden.

Die NFA5 konnte durch die eingeführte lokale SQLite Datenbank auf jedem Endgerät sichergestellt werden. Auch die NFA9 ist erfüllt, da die Informationen nach einmaligem downloaden immer lokal verfügbar sind.

In dem Prototypen sind alle Funktionen vom Start der Anwendung aus mit maximal vier Klicks zu erreichen, womit die NFA7 auch erfolgreich umgesetzt werden konnte.

Auf Grund der unterschiedlichen Gerätespezifikationen kann für die Anforderung NFA8 keine allgemein gültige Aussage getroffen werden. Durch die unterschiedlichen Testpersonen konnten jedoch mehrere positive Rückmeldungen eingeholt werden.

Die Anforderung NFA10 konnte soweit umgesetzt werden, dass alle verwendeten Interfaces mit den nötigen Javadoc Kommentaren versehen wurden.

5.3.2 Erkenntnisse aus der Evaluation

Es ist sehr gut zu erkennen, dass die Prioritäten der Anforderungen sehr gut berücksichtigt wurden. Letztendlich konnten zumindest alle *unverzichtbaren* und *sehr wichtigen* Anforderungen entweder bereits fertig implementiert oder aber bereits fertig geplant werden.

Während dieses Projektes konnten ca. 55% der Anforderungen konkret umgesetzt und in einem Prototypen realisiert werden. Für ungefähr 85% aller Anforderungen wurde bereits ein Konzept entwickelt, welches nur noch umgesetzt werden muss. Außerdem musste keine der Anforderungen verworfen werden, weil sie nicht umsetzbar ist.

In der Tabelle 5.1 sind die Anforderungen noch einmal übersichtlich aufgelistet. Die Farben stellen dabei folgenden Status dar:

- GRÜN : Die Anforderung wurde berücksichtigt und konnte erfolgreich im Prototypen realisiert werden.
- GELB : Die Anforderungen wurde in der Designphase berücksichtigt und es wurde ein konkretes Konzept entwickelt. Die Implementation im Prototypen konnte jedoch noch nicht umgesetzt werden.
- ORANGE : Die Anforderung wurde in der Designphase berücksichtigt, sodass es keine Probleme geben sollte sie später zu erfüllen. Es besteht jedoch noch kein konkretes Konzept der Umsetzung oder eine fertige Implementation.
- ROT : Die Anforderung konnte in dem fertigen Entwurf nicht erfüllt werden.

Nummer	Priorität	Status
FA1	sehr wichtig	
FA2	sehr wichtig	
FA3	sehr wichtig	
FA4	sehr wichtig	
FA5	wichtig	
FA6	nicht so wichtig	
FA7	wichtig	
FA8	nicht so wichtig	
FA9	unverzichtbar	
FA10	unverzichtbar	
FA11	unverzichtbar	
FA12	sehr wichtig	

Nummer	Priorität	Status
NFA1	wichtig	
NFA2	unverzichtbar	
NFA3	unverzichtbar	
NFA4	wichtig	
NFA5	sehr wichtig	
NFA6	sehr wichtig	
NFA7	wichtig	
NFA8	wichtig	
NFA9	sehr wichtig	
NFA10	wichtig	
NFA11	wichtig	

Tabelle 5.1: Status der Anforderungen

6 Schluss

Abschließend möchte ich wesentliche Aspekte des Projektes noch einmal zusammenfassen, ein persönliches Fazit ziehen und einen kurzen Ausblick über mögliche Ergänzungen geben.

6.1 Zusammenfassung

Die Zielsetzung des Projektes bestand darin, ein umfassendes Konzept für eine Android Applikation zu erstellen, die sich ohne wesentliche Veränderungen in das bestehende System des HVBV integrieren lässt. Die funktionalen und nicht-funktionalen Anforderungen an die Applikation wurden durch eine Befragung von potentiellen Benutzern identifiziert. Um eine fundierte Entscheidung zur Durchführung des Projektes treffen zu können, wurde eine Risk-Map erstellt und für die darin aufgeführten Risiken entsprechende Gegenmaßnahmen entwickelt.

Anschließend wurde eine Architektur erstellt, welche die Umsetzung der definierten Anforderungen ermöglichte. Dabei wurde mit erprobten Entwurfsmustern und bestimmten Software-Engeneering Methoden versucht, eine hohe Qualität der Software sicherzustellen. Die Verwendung des MVC und des Factory Entwurfsmusters sorgen für eine flexible Architektur. Zusammen mit der ausführlichen Dokumentation der Anwendung können Änderungen und Ergänzungen mit minimalem Aufwand durchgeführt werden. Damit wurden gute Voraussetzungen für die Weiterführung dieses Projektes geschaffen.

Durch regelmäßige Testdurchläufe mit ausgewählten Benutzern und die Überprüfung definierter Softwaremetriken wurde versucht die durch geforderte Qualität der Anwendung sicherzustellen.

Anhand der Evaluation konnte ein großer Teil der angestrebten Qualität nachgewiesen werden. Bis auf einige niedrig priorisierte Anforderungen, konnten alle anderen in dem entstanden Prototypen umgesetzt werden.

Mit dem Verlauf des Projektes und dem erarbeiteten Ergebnis bin ich sehr zufrieden, da in der Evaluation nachgewiesen werden konnte, dass die Zielsetzung des Projektes weitestgehend

erfüllt wurde. Meines Erachtens nach, haben folgende Faktoren für den Erfolg eine wesentliche Rolle gespielt:

- *Die besonders gründliche Planung in der Analyse Phase des Projektes*
  Hier wurde ein guter Überblick über das Projekt hergestellt und es wurden die grundlegenden Anforderungen an die Anwendung identifiziert.
- *Die enge Zusammenarbeit mit dem HVBV und den potentiellen Benutzern*
  Sie sorgte dafür, dass die Anwendung den Vorstellungen und Erwartungen eines bestimmten Benutzerkreises entspricht. Dadurch konnte sichergestellt werden, dass das Ergebnis des Projektes auch tatsächlich relevant ist.
- *Eine regelmäßige Kontrolle des Projektverlaufes*
  Ausgelöst durch bestimmte Kennzahlen oder Meilensteine wurden die bereits fertigen Teile der Anwendung und der aktuelle Stand des Projektes regelmäßig überprüft und im Hinblick auf die Zielsetzung erneut bewerten. Dadurch behielt man stets den Überblick über das gesamte Projekt und verlor die Zielsetzung nicht aus den Augen.

6.2 Ausblick

Ein großer Teil der Anwendung wurde bereits in dem Prototypen realisiert. Dennoch könnte das Ergebnis an bestimmten Stellen noch erweitert oder verbessert werden. Für den tatsächlichen Betrieb der Anwendung mit den vom HVBV zur Verfügung stehenden Daten, fehlt noch ein Teil der Anwendung. Bis jetzt wurde die Anwendung mit Daten der vergangen Saison getestet, die manuell aus der MS Access Datenbank entnommen und in die neue MySQL Datenbank integriert wurden. Für diesen Teil der Anwendung müssen noch folgende Aufgaben umgesetzt werden:

- Die Realisierung der Komponente *Anpassung* auf dem Büroserver aus Abbildung 4.1, die für die Übertragung der Daten, von der MS Access Datenbank in die neue MySQL Datenbank verantwortlich ist.
- Einrichten der MySQL Datenbank im Serversystem des HVBV.

Anschließend daran könnten folgende geplante, aber noch nicht realisierte Anforderungen umgesetzt werden:

- Dem Benutzer die Möglichkeit geben Einstellungen aus FA8 in der Applikation selber zu Verändern (vgl. Tabelle 3.1).
- Die Funktion bereitstellen, dass Spieltage in den Kalender des Benutzers eingetragen werden können (vgl. FA6 aus Tabelle 3.1).

Zum Abschluss möchte ich noch ein Paar Ideen für die Weiterentwicklung der Anwendung geben, die nicht bei der Planung des Projektes berücksichtigt wurden:

- Eine Ansicht mit der man sich die News der Internetseite des HVBV angezeigt werden.
- Eine Historie für Mannschaften, die den Verlauf von Liga, Staffel und Platzierung über mehrere Spielzeiten anzeigt.
- Bei Änderungen von Spieltagen eine Benachrichtigung für die betroffenen Mannschaften hinzufügen.

6.3 Anhang

Auf der beigelegten CD-Rom befinden sich folgende Dateien:

- Die gesamte Arbeit noch einmal im PDF Format.
- Ein Ordner *"SourceCode"*, in dem alle Dateien enthalten sind, die für die Ausführung der entwickelten Anwendung benötigt werden.
  - Ein Android Eclipse Projekt für die HVBV App.
  - Zwei Android Eclipse Bibliothek Projekte (Greendroid und com_viewpagerindicator), die zum Ausführen der HVBV eingebunden werden müssen.
  - Eine SQL Datei zum erstellen der externen Datenbank.
  - Ein Ordner hvbv, in dem sich die REST Schnittstelle für die externe Datenbank befindet.
  - Eine how_to_install.txt Datei, in der die wesentlichen Schritte zur Installation der Anwendung noch einmal genauer erklärt werden.

Acronyms

CPoF Central Point of Failure.

DVM Dalvik Virtual Machine.

HVBV Hamburger Volleyball Verband.

JDBC Java Database Connectivity.

JSON JavaScript Object Notation.

JVM Java Virtual Machine.

LoC Lines of Code.

MVC Model-View-Conroller.

ORM Object-Relational Mapping.

REST Representational State Transfer.

SDK Software Development Kit.

SoC Seperation of Concerns.

SPoC Single Point of Control.

XML Extensible Markup Language.

Glossar

App-Store
: Der Begriff App Store kommt ursprünglich von dem iPhone und beschreibt die Anwendung, über die sich weitere Applikationen für das Smartphone herunterladen lassen. Der Begriff ist jedoch auch auf andere Smartphone Betriebssysteme übertragbar, die eine solche Anwendung anbieten. Unter Android nennt sich diese Anwendung genau genommen Google Play Store.

CPoF
: Ein Bestandteil eines technischen Systems, dessen Ausfall den Ausfall des gesamten Systems nach sich zieht.

JDBC
: Eine einheitliche Java Schnittstelle zu verschiedenen relationalen Datenbanken.

JSON
: Ein kompaktes Datenformat zum Datenaustausch zwischen Anwendungen. Es handelt sich dabei um ein für Mensch und Maschine einfach lesbare Textform.

JVM
: Der Teil der Java-Laufzeitumgebung, der für die Ausführung des Java-Bytecodes verantwortlich ist.

LoC
: Anzahl an Programmzeilen.

REST
: Ein Architekturstil, der die Grundlage für das World Wide Web ist. Dabei wird eine Ressource, bestehend aus einer Sequenz von Bytes und Metadaten, zur Beschreibung der Darstellung, über eine URL zur Verfügung gestellt. Die Interaktionen mit der Ressource sind sowohl kontextfrei, als auch zustandslos. Sie finden über wohldefinierte GET, POST, PUT und DELETE Methoden statt.

Sandbox

Ein isolierten Bereich, innerhalb dessen jede Aktion keinerlei Auswirkung auf die äußere Umgebung hat.

Singleton

Ein Entwurfsmuster aus der Softwareentwicklung, welches sicherstellt, dass von einer Klasse genau ein Objekt existiert.

SoC

Unterteilt eine Anwendung in verschiedene Bereiche, die sich in ihrer Funktionalität möglichst wenig überlappen.

SPoC

Stellt sicher, dass Änderungen an dieser Stelle auf das gesamte System angewandt werden.

XML

Eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten in Form von Textdateien

Literaturverzeichnis

[Balzert 2001] Balzert, Helmut: *Lehrbuch der Software-Technik*. Spektrum Akademischer Verlag, 2001 (2. Auflage)

[Becker und Pant 2010] Becker, Arno ; Pant, Marcus: *Android 2. Grundlagen und Programmierung*. dpunkt.verlag, 2010 (1. Auflage). – ISBN 9783898646772

[Buschmann u. a. 1996] Buschmann, Frank ; Meunier, Regine ; Rohnert, Hans ; Sommerlad, Peter ; Stal, Michael: *Pattern - Oriented Software Architecture A System of Patterns*. Wiley, Oktober 1996. – ISBN 9780471958697

[Copeland und Maier 1984] Copeland, C. ; Maier, D.: Making smalltalk a database system. In: *ACM SIGMOD Records* 14 (1984), Nr. 2

[Gamma u. a. 2000] Gamma, Erich ; Helm, Richard ; Johson, Ralph ; Vlissides, John: *Design Patterns Elements of Reusable Object-Oriented Software*. Addison-Wesley, Mai 2000. – ISBN 9780201633610

[Gartner 2012] Gartner: *Gartner Says Worldwide Sales of Mobile Phones Declined 2 Percent in First Quarter of 2012*. Press Release. Mai 2012. – URL http://www.gartner.com/it/page.jsp?id=2017015

[Google 2012a] Google: *The AndroidManifest.xml File*. 2012. – URL http://developer.android.com/guide/topics/manifest/manifest-intro.html

[Google 2012b] Google: *Philosophy and Goals*. 2012. – URL http://source.android.com/about/philosophy.html

[Kowitt 2011] Kowitt, Beth: 100 million Android fans can't be wrong. In: *CNN-Money* (2011), 16. Juni. – URL http://tech.fortune.cnn.com/2011/06/16/100-million-android-fans-cant-be-wrong/

[Markoff 2007] Markoff, John: I, Robot: The Man Behind the Google Phone. In: *New York Times Magazine* (2007), 4. November. – URL http://www.nytimes.com/2007/11/04/technology/04google.html?_r=1&pagewanted=all

[McCabe 1976] McCabe, Thomeas J.: A Complexity Measure. In: *IEEE Transactions on Software Engineering* SE-2 (1976), Dezember, Nr. 4. – URL http://www.literateprogramming.com/mccabe.pdf

[MediaCT 2012] MediaCT, Ipsos O.: *Unser mobiler Planet: Deutschland.* Mai 2012. – URL http://services.google.com/fh/files/blogs/our_mobile_planet_germany_de.pdf

[OHA 2012] OHA: *Informationen über die OHA.* 2012. – URL http://www.openhandsetalliance.com/oha_overview.html

[Starke 2009] Starke, Gernot: *Effektive Software Architekturen.* Bd. 4. Carl Hanser Verlag, 2009. – ISBN 9783446420083

[www.stackoverflow.com] www.stackoverflow.com: *Diskussionen zur MVC Architektur unter Android 2.* – URL http://stackoverflow.com/questions/2925054/mvc-pattern-in-android

[www.therealjoshua.com] www.therealjoshua.com: *Diskussionen zur MVC Architektur unter Android 1.* – URL http://www.therealjoshua.com/2011/12/android-architecture-part-9-conclusion/

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 19. Oktober 2012 ChristianKirchner