Florian Sumann, BSc

Implementation und Validierung einer plattformspezifischen Datensynchronisation

MASTERARBEIT

zur Erlangung des akademischen Grades

Master of Science

Masterstudium Softwareentwicklung - Wirtschaft

eingereicht an der

Technischen Universität Graz

Betreuer
Univ.-Prof. Dipl.-Ing. Dr.techn. Wolfgang Slany

Institut für Softwaretechnologie

Graz, April 2016

Eidesstattliche Erklärung[1]

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Graz, am _____ _____

Datum Unterschrift

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Graz, _____ _____

Date Signature

[1] Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008

Kurzfassung

Das umfassende Angebot an mobilen Endgeräten ist gegenwärtig immens groß. Demnach steigt auch die Nachfrage nach Applikationen, welche aus diesen technischen Hilfsmitteln erst nützliche Universalwerkzeuge machen, rasant an. Doch viele Benutzer dürfen gleich mehrere Geräte ihr Eigen nennen und wollen die darauf befindlichen Informationen auch überall zur Verfügung haben. Aus diesem Grund werden im Allgemeinen die Daten nicht nur im lokalen Speicher des Endgeräts abgelegt, sondern sogleich zu einem Server transferiert, der daraufhin bei deren Verteilung behilflich ist.

Wenn Informationen auf verschiedene Weise erstellt, bearbeitet, oder auch gelöscht werden, dann gewinnt der Anwendungsbereich der Datensynchronisation an Bedeutung. Vor allem dann, wenn bestimmte Inhalte zur selben Zeit geändert werden. Um den Verlust von Daten zu vermeiden, müssen beim anschließenden Synchronisationsvorgang Entscheidungen darüber getroffen werden, welche Daten schlussendlich den aktuellen Informationsgehalt repräsentieren.

Die Thematik dieser Masterarbeit lässt sich in zwei Aufgabengebiete unterteilen. Die erste Aufgabe beinhaltet die Umsetzung eines Synchronisationsmechanismus mit speziellen Anpassungen für die Onlineplattform Almdesk. Diese stellt dem Benutzer eine Reihe von Anwendungen für den täglichen Gebrauch bereit. Zusätzlich wird noch eine Android-Applikation implementiert, die vorerst lediglich die Synchronisation von Kontaktdaten ermöglicht. Die zweite Aufgabe befasst sich mit der Ausführung von unterschiedlichen Validierungsmethoden. Der Großteil davon wird vorwiegend zu Optimierungszwecken eingesetzt, während einige wenige nur eine Kontrollfunktion haben. Damit soll verdeutlicht werden, dass ein sorgfältiges Testmanagement den Entwicklungsprozess unterstützt und die Softwarequalität positiv beeinflusst.

Abstract

These days, there is a large amount of mobile devices on the market. The functionality of such a device can be extended by installing or upgrading an application that incorporates new features or improvements. Numerous users use multiple devices at the same time and they expect that their data is constantly updated on all of them. For this reason the data is cached in the local storage of the device and also transferred to the external server that helps to distribute the given data to all kind of clients.

When information is created, edited or deleted in different ways at the same time, data is usually distributed with a special mechanism of synchronization. Otherwise data might be lost. Due to that approach it is possible to decide during the synchronization process, which changes should represent the current dataset.

The aim of this master thesis is to focus on two major topics. The first part of the thesis deals with the realization of a synchronization mechanism that is specially adapted to the needs of the online platform Almdesk. This platform provides a selection of helpful widgets for daily use. Additionally, an Android application is being implemented that, for now, only allows the synchronization of contacts. The second part of this thesis includes different techniques of software validation. Most of these methods are used to optimize the implemented functionality while some of them simply have a controlling function. The demonstration of the selected test methods emphasizes the various benefits of an accurate test management and its positive influences on the quality of the developed software.

Danksagung

Bedanken möchte ich mich bei der Unternehmensleitung der G+Z Software GmbH, die mich mit der Durchführung dieser Masterarbeit beauftragt hat. Im Besonderen seien hier die Herren Mario Ganster, Dipl.-Ing. Wolfgang F. P. Hutter und Christian Zorc erwähnt, die mir im Verlauf dieser Arbeit mit nützlichen Ratschlägen zur Seite gestanden haben. Des Weiteren bedanke ich mich bei Herrn Univ.-Prof. Dipl.-Ing. Dr.techn. Wolfgang Slany für seinen Einsatz bei der Betreuung meiner Arbeit.

Der wohl größte Dank gebührt jedoch meinen Eltern. Sie haben mich während meiner gesamten Studienzeit tatkräftig unterstützt und mir somit diese Ausbildung erst ermöglicht.

Zu guter Letzt möchte ich mich auch bei meiner Freundin für ihr Verständnis und ihre außerordentliche Hilfsbereitschaft bedanken.

Graz, April 2016 Florian Sumann, BSc

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

1. Einleitung

In den letzten Jahren hat sich das Angebot an Geräten für den typischen Softwareanwender sehr stark erweitert. Neben dem herkömmlichen PC gesellte sich eine Reihe von unterschiedlichen Gerätetypen hinzu, welche zum Großteil durch ihre kompakte Größe, Mobilität und Kommunikationsfähigkeiten überzeugen. Zu den beliebtesten Vertretern zählen wohl Laptops, Netbooks, Smartphones, Tablets, Smartwatches und Smart TV Geräte.

Das Softwareangebot besteht aus einer Vielzahl von Applikation, auch kurz Apps genannt, die dem Benutzer helfen sollen seinen Alltag zu erleichtern oder die Zeit zu vertreiben. Sowohl große als auch kleine Unternehmen sind deshalb sehr bemüht, so viele verschiedene Geräte wie nur möglich mit ihren Produkten zu versorgen und damit mehr Benutzer ansprechen zu können. Dabei wird das Aufgabengebiet der Datensynchronisation immer umfassender und wichtiger, da der Anwender seine gespeicherten Informationen, wie Dokumente, Bilder, Videos oder Spielstände, auf jedem seiner Geräte stets aktuell vorfinden möchte. Aus diesem Grund bieten zahlreiche App-Hersteller ihren Benutzern die Möglichkeit, ihre Daten zentral auf einem Server zu speichern und zwischen den einzelnen Geräten zu synchronisieren.

Damit diese Anforderungen an die Software zuverlässig und zur Zufriedenheit der Anwender erfüllt werden können, werden in der Softwareentwicklung mehrere Verfahren eingesetzt, um potenzielle Fehlerquellen bereits während der Entwicklung zu erkennen und zu beseitigen. Diese Vorgehensweisen sind Teil der Qualitätssicherung und unerlässlich bei der Entwicklung von modernen und komplexen Anwendungen. Dabei wird zum einen die Qualität der Software untersucht aber auch getestet und zum anderen der Entwicklungsprozess genau analysiert und optimiert. Bei genauerer Betrachtung der einzelnen Teilgebiete der Qualitätssicherung

wird jedoch schnell klar, dass mit Hilfe dieser Verfahren nicht annähernd alle Probleme gelöst werden können. Somit hat die Wahl der Testmethoden, Analysen und die Strukturierung des Entwicklungsumfeldes unmittelbaren Einfluss auf die Qualität einer Softwareanwendung. (Hoffmann, 2008, Seite 19-26)

1.1. Motivation

In der Softwareentwicklung beschränken sich viele Entwickler beim Testen ihrer Produkte auf reine Systemtests, oder auch auf Black-Box-Tests. Wie man in Abbildung 1.1 sehen kann, wird hierbei lediglich der Funktionsumfang der Anwendung mit den Vorstellungen des Auftraggebers verglichen. Kommt es zu diesem Zeitpunkt jedoch zu Fehlverhalten oder falschen Resultaten, können die Ursachen für auftretende Fehler in vielen Fällen nicht eindeutig lokalisiert werden. Somit verzögert die Suche und Behebung der Fehler nicht nur den Entwicklungsfortschritt, sondern könnte auch zusätzliche unvorhersehbare Kosten verursachen. Abbildung 1.2 zeigt die Relation zwischen dem Zeitpunkt der Fehlerursache und dem Zeitpunkt der Fehlerentdeckung während des Entwicklungsverlaufs. Folglich kann man daraus schließen, dass die steigenden Kosten aufgrund der Fehlerkorrekturen in direktem Bezug zum Zeitpunkt der Fehlerentdeckung stehen. Demnach ist ein systematischer Einsatz von unterschiedlichen Testverfahren während des Entwicklungsprozesses durchaus sinnvoll. (Kleuker, 2013b, Seite 34-35,309-312)

Wie in der Einleitung bereits erwähnt, spielt auch die Datensynchronisation in Verbindung mit mobilen Geräten eine immer wichtigere Rolle. Darüber hat sich auch das Unternehmen G+Z Software GmbH[1] Gedanken gemacht, welches eine Onlineplattform mit dem Namen Almdesk[2] betreibt. Dabei handelt es sich um einen Onlinedienst, der populäre und sehr häufig genutzte Anwendungen der digitalen Welt vereint. Neben der Verwaltung von Dokumenten, Aufgaben, Links oder Notizen besteht für den Benutzer auch die Möglichkeit mit anderen zu kommunizieren und Informationen

[1] http://www.gz-software.at/
[2] https://almdesk.com/

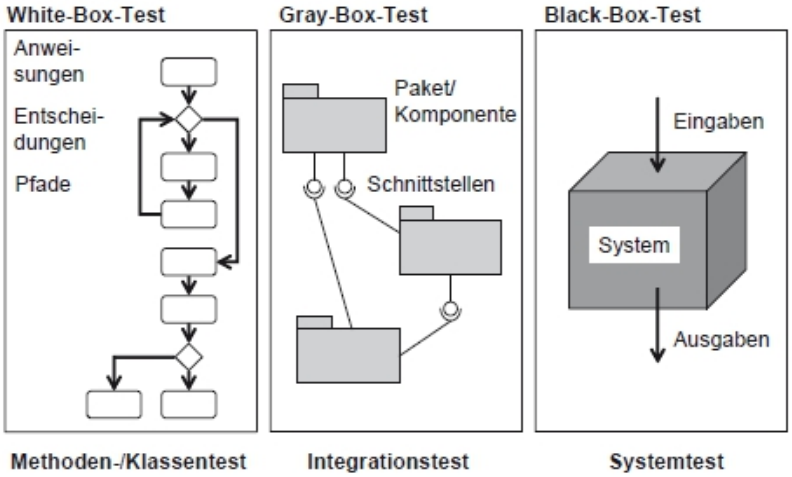

Abbildung 1.1.: Verschiedene Ansätze von Testverfahren (Kleuker, 2013b, Seite 311)

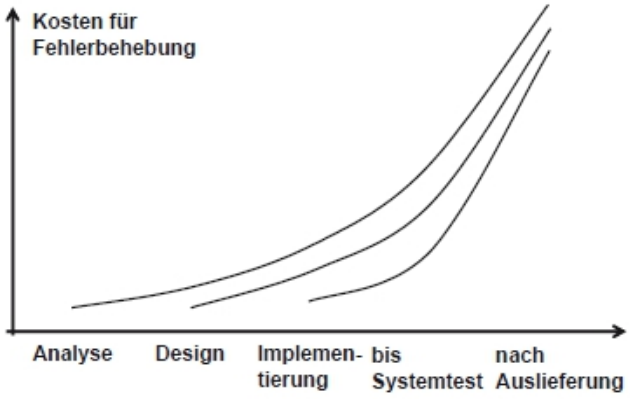

Abbildung 1.2.: Relation zwischen Fehlerbehebung und Kosten (Kleuker, 2013b, Seite 312)

Abbildung 1.3.: Almdesk - Oberfläche der Dokumentenverwaltung (G+Z Software GmbH, 2016a)

auszutauschen, sowie den Zugang zu den verschiedensten Nachrichtenportalen zu nutzen. Die Benutzeroberfläche unterteilt sich in einzelne Widgets[3], wobei jede zuvor genannte Aufgabe von einem Widget übernommen wird. Abbildung 1.3 zeigt die Ansicht des Widgets für Dokumentenverwaltung in zwei unterschiedlichen Varianten. (G+Z Software GmbH, 2016a)

Die gesamte Almdesk-Plattform ist jedoch zurzeit nur mit einer bestehenden Internetverbindung erreichbar. Notizen oder Aufgaben können beispielsweise nicht geändert werden und zu einem späteren Zeitpunkt auf dem System aktualisiert werden. Ein Synchronisationsmechanismus für bestimmte Widgets könnte hierbei Abhilfe schaffen, besonders wenn für den Zugriff auf die Plattform mehrere Geräte verwendet werden. (G+Z Software GmbH, 2016a)

[3] „Mini-Anzeigeprogramm [...]; grafisches Objekt“ (Technische Universität Chemnitz, 2016)

1.2. Anforderungen und Zielsetzungen

Diese Masterarbeit befasst sich mit der Erweiterung der Onlineplattform Almdesk und wurde vom Unternehmen G+Z Software GmbH in Auftrag gegeben. Das Aufgabengebiet beinhaltet das Entwickeln und Testen mehrere Module, die im laufenden Betrieb teilweise miteinander interagieren sollen.

Das erste Modul ist ein neues Widget für Almdesk, welches zum Anlegen, Bearbeiten und Löschen von Kontaktdaten eingesetzt werden kann. Die Benutzeroberfläche sollte sich dabei an die unterschiedlichsten Bildschirmauflösungen anpassen können. Das zweite Modul bezieht sich speziell auf die mobile Variante von Almdesk. Dabei handelt es sich im Rahmen dieser Arbeit ausschließlich um eine Applikation für jene Smartphones, die mit dem Betriebssystem Android[4] ausgestattet sind. Wie auch beim ersten Modul wird ein Widget benötigt, das die Bearbeitung von Kontaktdaten direkt am Smartphone ermöglicht. Das dritte und somit letzte Modul besteht aus einem Synchronisationsmechanismus, welcher für den Datenabgleich zwischen mobilen Geräten und dem Server über eine webbasierte Schnittstelle eingesetzt werden soll. Als ersten Anwendungsbereich der Synchronisation würde sich sogleich das Kontakt-Widget des Almdesk Apps anbieten um die Kontakte vom Smartphone zu synchronisieren.

Das Ziel dieser Arbeit ist jedoch nicht nur die Implementierung der einzelnen Module, sondern auch mit den geeigneten Testverfahren deren Funktionalität und Verhalten ausreichend zu validieren und damit die Qualität zu verbessern. Aus diesem Grund werden verschiedene Methoden genauer betrachtet und analysiert, um mit den daraus resultierenden Erkenntnissen eine angemessene Auswahl an Testverfahren zu treffen.

1.3. Einteilung der Arbeit

Dieser Abschnitt liefert einen kurzen Überblick über die weiteren Kapitel dieser Arbeit, welche sich in zwei große Themengebiete unterteilen lässt.

[4] http://www.android.com/

Dabei handelt es sich einerseits um die Entwicklung von Widgets sowie deren Synchronisationsmechanismen und andererseits um ausgewählte Testverfahren zur Steigerung der Softwarequalität.

Überlegungen und theoretische Grundlagen zur Umsetzung der Anforderungen werden in Kapitel 2 erläutert. Des Weiteren werden auch themenbezogene Arbeiten vorgestellt, um unterschiedliche Sichtweisen kennenzulernen.

In Kapitel 3 wird allgemein auf die Struktur und Funktionsweise von Widgets in Almdesk eingegangen und die Implementierungen genauer vorgestellt.

Die Vorgehensweise bei der Datensynchronisation wird in Kapitel 4 veranschaulicht. Dabei wird nicht nur der dazugehörige Mechanismus genauer betrachtet, sondern es werden auch alle Abläufe der Synchronisation ausführlich erklärt.

Kapitel 5 befasst sich ausschließlich mit der Auswahl und Umsetzung von geeigneten Testverfahren. Hier werden alle Testmethoden vorgestellt, die im Rahmen dieser Arbeit angewendet werden.

Kapitel 6 beinhaltet eine kurze Zusammenfassung zu den ausgearbeiteten Themengebieten. Dieser Teil enthält auch eine Diskussion über die erhaltenen Erkenntnisse und Ergebnisse.

Den Abschluss dieser Arbeit bildet Kapitel 7 mit einem Ausblick auf zukünftige Weiterentwicklungen und mögliche Erweiterungen.

2. Grundlagen und themenbezogene Bereiche

In diesem Kapitel werden die grundlegenden Aufgabengebiete der Qualitätssicherung in der Softwareentwicklung betrachtet. Da von einigen dieser Themenbereiche im Verlauf dieser Arbeit noch praktische Anwendungsmöglichkeiten vorkommen, wird auch in diesem Kapitel genauer auf deren Grundlagen eingegangen.

Darauf folgt ein kurzer Einblick in den Bereich der Datensynchronisation. Die Vorstellung von unterschiedlicher Projekte soll dabei zeigen, mit welchen Möglichkeiten Daten gegenwärtig synchronisiert werden.

2.1. Qualitätssicherung

In der modernen Softwareentwicklung hat die Qualitätssicherung dafür zu sorgen, dass der Benutzer eine Anwendung mit dem erwartungsgemäßen Funktionsumfang erhält. Mit den steigenden Qualitätsansprüchen der Kunden steigen auch die Anforderungen an die Qualitätssicherung, damit die entwickelten Produkte nicht nur hochqualitativ sondern auch rechtzeitig ausgeliefert werden können. Und um genau dies zu erreichen bedient man sich der Durchführung einer Qualitätssicherung, welche sich von Beginn an durch den gesamten Projektverlauf hindurch zieht. (O'Regan, 2014, Seite 5-7)

Um die Qualität einer Software wirklich einschätzen zu können, bedarf es einer Definition von Anforderungen, die ein hochwertiges Produkt erfüllen sollte. O'Regan (2014, Seite 5-6) und Wagner (2013, Seite 10) verweisen hierbei auf spezielle Standards, welche in ISO/IEC 9126 (2001) und in ISO/IEC

25010 (2011) beschrieben werden. Diese beinhalten diverse Merkmale, welche für eine genauere Betrachtung noch weiter unterteilt werden können und sind bei der Analyse und Spezifikation von Softwareanforderungen, sowie bei der Bewertung der Qualität behilflich. In Tabelle 2.1 werden die einzelnen Hauptmerkmale und deren Bedeutung kurz vorgestellt. Je nach Art und Einsatzgebiet der Software müssen nicht alle Merkmale stark ausgeprägt sein, um eine qualitativ hochwertige Lösung bieten zu können.(Wagner, 2013, Seite 10-12,15-16)

Für Kleuker (2013c, Seite 18) ist die Qualitätssicherung verantwortlich für das Auffinden jeglichen Fehlverhaltens. Dabei spielt es keine Rolle ob es sich um einen Programmierfehler handelt, oder ob die Anforderungen unzureichend umgesetzt wurden. Da somit das Aufgabengebiet der Qualitätssicherung sehr vielseitig ist, unterscheiden Kleuker und auch Hoffmann (2008, Seite 20) zwischen konstruktiver und analytischer Qualitätssicherung.

2.1.1. Konstruktive Qualitätssicherung

Laut Hoffmann (2008, Seite 20-22,65-66) handelt es sich bei der konstruktiven Qualitätssicherung um bestimmte Methoden, die den gesamten Entwicklungsprozess einer Software von Beginn an begleiten. Sie sollen dabei helfen, Fehler im Vorfeld und auch während der Entwicklung bestmöglich zu vermeiden. Die Anwendungsbereiche werden im folgenden Abschnitt kurz vorgestellt:

- **Software-Richtlinien:** Dabei handelt es sich um unterschiedliche Vorgaben zur Anwendung der gewählten Programmiersprache. Bei der Zusammenarbeit von mehreren Entwicklern kann eine einheitliche Form Zeit sparen. Auch die Vorgabe von bewährten Lösungsansätzen hilft dabei, Fehlerquellen frühzeitig zu eliminieren.

- **Typisierung:** Die Verwendung von Datentypen kann dazu führen, dass Unstimmigkeiten beim Programmablauf sofort erkannt werden.

Merkmal	Beschreibung
Funktionalität	Die Software soll den erwarteten Anforderungen hinsichtlich Funktionsweise und Umfang entsprechen.
Zuverlässigkeit	Dieses Merkmal bezieht sich auf die Häufigkeit von Softwarefehlern oder unerwarteten Ereignissen.
Effizienz	Beschreibt den Einsatz von effizienten Algorithmen und eine performante Nutzung von Hardware. Wenn der Fokus auf eine schnelle Bedienung, sowie auf kurze Reaktionszeiten gerichtet ist, spielt die Effizienz auch bei der Anwendung der Software eine große Rolle.
Benutzbarkeit	Wie gut und komfortabel sich die Software vom Benutzer bedienen lässt hängt von der benutzerfreundlichen Umsetzung ab. Dabei sind die bereits erwähnten Merkmale, wie Funktionalität, Zuverlässigkeit und Effizienz, von großer Bedeutung.
Wartbarkeit	Eine gute Wartbarkeit hängt davon ab, ob sich die Software schnell und problemlos erweitern oder ändern lässt.
Portabilität	Dieses Qualitätsmerkmal bezieht sich darauf, wie flexibel die Software bei der Übertragung auf andere bzw. neue Plattformen ist.
Kompatibilität	Die Kompatibilität gibt an, wie einfach die Software mit anderen Software- oder Hardwareelementen verknüpft werden kann. Dies betrifft nicht nur die Anwendung, sondern auch die Entwicklung.
Sicherheit	Hierbei handelt es sich um sicherheitsrelevanten Eigenschaften einer Software. Die Sicherheit soll Auskunft darüber geben, wieviel Schutz die Software gegen unbefugte Zugriffe bietet.

Tabelle 2.1.: Qualitätsmerkmale von Softwareanwendungen (ISO/IEC 25010, 2011)(Wagner, 2013, Seite 10-12)

- **Vertragsbasierte Programmierung:** Diese Art der Programmierung versucht mit Hilfe von Vor- und Nachbedingungen Fehler in bestimmten Programmabschnitten zu ermitteln.
- **Fehlertolerante Programmierung:** Mit diesem Programmieransatz wird nicht versucht alle Fehler zu beseitigen, sondern diese abzufangen und darauf entsprechend zu reagieren.
- **Portabilität:** Die Portabilität einer Software gibt an, ob sich diese ohne großen Aufwand auf unterschiedlichen Betriebssystemen ausführen lässt. Somit prägt die Portabilität auch den Programmierstil, was sich durchaus auf die Fehlervermeidung auswirken könnte.
- **Dokumentation:** Eine einheitliche und vollständige Dokumentation kann das Fehlerpotenzial in einem Softwareprojekt reduzieren.

2.1.2. Analytische Qualitätssicherung

Hoffmann (2008, Seite 20,22) und Kleuker (2013c, Seite 18) zählen jene Methoden zur analytischen Qualitätssicherung, welche während des Entwicklungsprozesses, aber auch nachdem die Entwicklung bereits abgeschlossen wurde, eingesetzt werden. Darunter befinden sich sämtliche Verfahren, welche zum Testen von Software angewendet werden. Diese werden in Abschnitt 2.1.3 genauer vorgestellt. Darüber hinaus gehören zu diesem Bereich der Qualitätssicherung auch unterschiedliche Analyseverfahren, sowie bestimmte Techniken zur Verifizierung von Softwareeigenschaften. Eine zusammenfassende Veranschaulichung dieser Verfahren gibt es in den folgenden Absätzen.

Metriken

Mit dem Einsatz von Metriken lässt sich die Qualität einer Software bewerten. Dazu werden bestimmte Faktoren definiert, welche für die Entwicklung einer hervorragenden Software ausschlaggebend sein können. Diesen Faktoren werden Wertebereiche zugeteilt, welche folglich bei der Evaluierung eine wichtige Rolle spielen. Wenn bei der Evaluierung bestimmte Schwerpunkte

gesetzt werden, können die Faktoren auch zuvor priorisiert werden. Bei der Entwicklung der Software ist es dann jederzeit möglich, Referenzwerte hinsichtlich der ausgewählten Faktoren zu berechnen und somit eine Evaluierung mit dem erstellten Wertesystem durchzuführen. Das Ergebnis einer solchen Evaluierung soll die Qualität der Software wiederspiegeln. Wobei es hier auf einen akkuraten Einsatz von Wertebereichen und Priorisierungen ankommt, um das gewünschte Resultat zu erhalten. (Kleuker, 2013c, Seite 21-22)

Manuelle Prüfung

Ein weiteres Verfahren zur analytischen Qualitätssicherung ist die manuelle Prüfung von Software. Im Gegensatz zu den bereits erwähnten Verfahren wird hier die entwickelte Software nicht maschinell geprüft, sondern die Analyse und Erstellung von Korrekturvorschlägen wird von verschiedenen Personen durchgeführt. Einerseits können dadurch nur schwer zu findende semantische Probleme entdeckt werden, und andererseits kommt hierbei die Routine der prüfenden Personen viel mehr zum Tragen als bei allen anderen Analyseverfahren. Für diese Art von Prüfung sind die drei folgenden Varianten bekannt:

- Walkthrough
- Review
- Inspektion

Bei einem *Walkthrough* erklärt der Entwickler seinen Kollegen die Funktionalität eines Code-Teils oder eines beliebigen Mechanismus. Allein durch die Präsentation, aber auch durch die Unterstützung der anwesenden Entwickler, können Fehler entdeckt werden und somit einen positiven Effekt auf die weitere Entwicklung ausüben. Die verfassten Software-Fragmente werden innerhalb des *Reviews*, analog zum Walkthrough, von anderen Entwicklern mit Hilfe eines zuvor festgelegten Anforderungskatalogs bewertet. Dies hat zwar den Vorteil, dass keine wesentlichen Punkte übersehen werden können, jedoch wird die Betrachtungsweise durch diese Vorgaben möglicherweise eingeschränkt. Bei der *Inspektion* handelt es sich ebenfalls um ein Instrument

zur Begutachtung von erstellten Software-Teilen, wobei hier die Bewertung in vorgegebenen Abschnitten durchgeführt wird. (Hoffmann, 2008, Seite 321-327)

Weitere Analyseverfahren

Hoffmann (2008, Seite 23) beschreibt noch weitere Analyseverfahren, welche dem Entwickler dabei helfen sollen, grundlegende Formalitäten einzuhalten, oder redundante Software-Teile zu erkennen. Dabei ist auch die Analyse von sicherheitsrelevanten Code-Fragmenten, sowie die Identifizierung von Sicherheitslücken ein wichtiger Bestandteil. Die damit gemeinten Analyseverfahren werden nachstehend angeführt:

- Konformitätsanalyse
- Exploit-Analyse
- Anomalienanalyse

Verifikation

Der Bereich der Verifikation von Software beschäftigt sich mit dem Problem, Quellcodeabschnitte auf ihre Fehlerfreiheit zu prüfen. Jedoch ist der Aufwand dafür beträchtlich, da alle möglichen Programm-Konstellationen miteinbezogen werden müssen. Aus diesem Grund gibt es auch noch ähnliche Ansätze, welche sich mit einer teilweisen Fehlerfreiheit begnügen und somit den Aufwand für die Überprüfung um ein Vielfaches reduzieren können. (Hoffmann, 2008, Seite 24)

2.1.3. Testverfahren

Dieser Abschnitt befasst sich allgemein mit möglichen Methoden zum Testen von Software-Fragmenten, sowie gesamten Software-Systemen. Die hier vorgestellten Verfahren werden im Rahmen dieser Arbeit auch zum

Teil in Kapitel 5 nochmals in Verbindung mit der Testimplementierung vorkommen.

Der Großteil der Testverfahren orientiert sich meistens nach einem ähnlichen Prinzip. Als Erstes werden die Testvoraussetzungen bestimmt. Diese unterscheiden sich je nach Umfang des Testfalles. Um aussagekräftige Resultate zu erhalten, sollten vor jedem Testdurchlauf immer die gleichen Voraussetzungen geschaffen werden. Im Anschluss daran wird der eigentliche Testdurchlauf durchgeführt. In diesem Teil wird folglich auch festgelegt, welche Arten von Tests durchgeführt werden. Nach Beendigung des Testdurchlaufs werden die Resultate analysiert und mit den angenommenen Werten verglichen. Bei näherer Betrachtung wird klar ersichtlich, dass die Qualität eines solchen Testfalles durch seine Voraussetzungen, den Test selbst und durch den Wertebereich der Ergebnisse bestimmt wird. (Kleuker, 2013c, Seite 25-27)

Einteilung der Testverfahren

Wie bereits in Abschnitt 2.1.2 erläutert, werden die meisten der hier vorgestellten Testverfahren bereits während der Entwicklung eingesetzt. In Bezug auf den Zeitpunkt des Einsatzes, sowie der entsprechenden Umsetzung der Testfälle, kann eine gewisse Einteilung der Testverfahren getroffen werden. Hoffmann (2008, Seite 158) beschreibt die oberste Stufe der Einteilung anhand folgender Hauptkategorien:

- Prüfebene
- Prüfkriterium
- Prüfmethodik

Abbildung 2.1 zeigt, aus welchen Teilen sich die drei Hauptkategorien zusammensetzen. Der *Prüfebene* werden jene Testverfahren zugeordnet, welche im Verlauf eines Projektes in den entsprechenden zeitgebundenen Abschnitten zum Einsatz kommen. Diese Verfahren befassen sich sowohl mit dem Testen von elementaren Programmbausteinen, als auch mit dem Zusammenwirken von unterschiedlichen Software-Komponenten. Im Vergleich zur Prüfebene werden beim *Prüfkriterium* die einzelnen Testverfahren

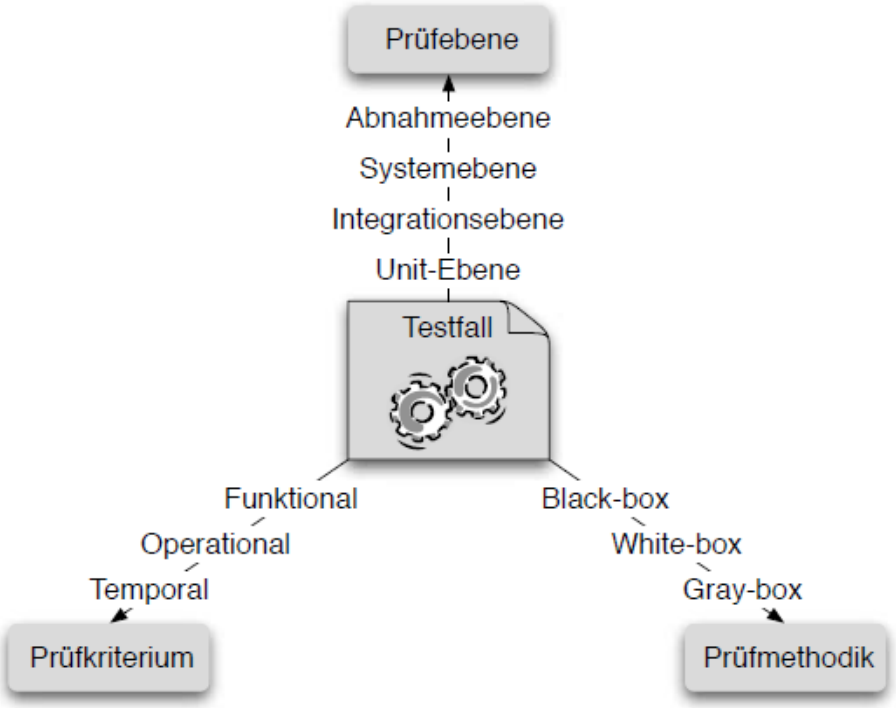

Abbildung 2.1.: Hauptkategorien der Testverfahren-Einteilung (Hoffmann, 2008, Seite 158)

hinsichtlich ihrer inhaltsbezogenen Kriterien gegliedert. Bei der Kategorie *Prüfmethodik* ergibt sich die Einteilung anhand der Zusammensetzung der Testmethoden. Dabei wird die Herangehensweise beim Aufbau und Ablauf der Tests zur Einordnung herangezogen. (Hoffmann, 2008, Seite 159,170,173)

Um eine konkrete Übersicht über die bedeutendsten Testverfahren während eines Entwicklungsprozesses zu erhalten, liefert O'Regan (2014, Seite 123) eine Auflistung von verschiedenen Arten von Testmöglichkeiten:

- Unit-Test
- Komponententest
- Systemtest

- Leistungstest
- Stresstest
- Browser Kompatibilitätstest
- Usability-Test
- Sicherheitstest
- Regressionstest
- Simulationstest
- Abnahmetest

Unit-Tests betrachten nur kleine Ausschnitte von Programmteilen. Die angefertigten Tests werden mit vordefinierten Ausgangswerten und unabhängig vom bestehenden Software-System durchlaufen. Anschließend wird überprüft, ob die erhaltenen Resultate mit den erwarteten Vorgaben übereinstimmen. Diese Testfälle werden im Laufe der Entwicklung ständig wiederholt, weshalb in der Regel alle erstellten Unit-Tests automatisiert abgearbeitet werden. Zu diesem Zweck werden Testumgebungen eingesetzt, welche sich um den automatischen Ablauf kümmern. Eine genauere Betrachtung solcher Testumgebungen folgt im Abschnitt 2.2. (Wagner, 2013, Seite 136-137)

Einzelne Funktionalitäten können zu größeren Komponenten zusammengeführt werden, welche dann universell im Projekt eingesetzt werden können, oder auch in anderen Projekten zur Anwendung kommen. Laut O'Regan (2014, Seite 123) ist der *Komponententest* dazu da, um die Sicherheit und Fehlerfreiheit der Komponenten zu testen. In diesem Zusammenhang sollte auch erwähnt werden, dass zusätzlich noch Integrationstests verwendet werden, um das Zusammenwirken unterschiedlicher Komponenten testen zu können. Hoffmann (2008, Seite 163-166) unterscheidet dabei zwischen den drei folgenden Integrationsmöglichkeiten: Big-Bang-Integration, strukturorientierte Integration und funktionsorientierte Integration.

Beim *Systemtest* wird das gesamte entwickelte System sorgfältig getestet. Dabei wird nochmals der geforderte Funktionsumfang, sowie die Benutzerfreundlichkeit und Leistungsfähigkeit des Systems überprüft. (Wagner,

2013, Seite 138-139)

Die Leistung des Systems wird mit sogenannten *Leistungstests* analysiert. Durch eigene Erhebungen, oder auch mit diversen Hilfsprogrammen können potenzielle Schwachstellen ermittelt werden. Die dadurch erhaltenen Leistungsdaten werden danach mit möglichen Projektspezifikationen verglichen. (O'Regan, 2014, Seite 123)

Durch den Einsatz von *Stresstests* können verschiedene Verhaltensweisen des Systems untersucht werden. Dazu zählt unter anderem das Verhalten eines Systems, das über die Leistungsgrenze hinaus beansprucht wird. Aufschlussreich ist danach aber auch die Reaktion auf das Zurückkehren zum Normalzustand. (Hoffmann, 2008, Seite 173)

O'Regan (2014, Seite 123) verweist darauf, dass der *Browser Kompatibilitätstest* vorwiegend bei Systemen mit webbasierten Benutzeroberflächen zum Einsatz kommt. Damit kann getestet werden, ob die Umsetzung der Oberflächengestaltung mit allen aktuellen oder zumindest den vorgegebenen Browsern kompatibel ist. Hoffmann (2008, Seite 171) bezieht sich beim Kompatibilitätstest auf das Testen des entwickelten Systems auf unterschiedlicher Hardware, sowie der Verträglichkeit mit anderen Software-Komponenten.

Beim *Usability-Test* wird die Benutzeroberfläche der erstellten Software im Detail betrachtet. Dabei wird darauf geachtet, ob die Bedienung der Anwendung für den Benutzer auch wirklich einfach und nachvollziehbar ist. (O'Regan, 2014, Seite 123)

Sicherheitstests werden durchgeführt, um zu prüfen, ob das System den spezifizierten Sicherheitsanforderungen gerecht wird. Zusätzlich können mit diesen Tests auch Verschlüsselungsmechanismen oder potenzielle Schwachstellen kontrolliert werden. (Hoffmann, 2008, Seite 172)

Regressionstests werden eingesetzt um sicherzustellen, dass Änderungen keine Auswirkungen auf die bereits erstellten Code-Teile haben. Dafür werden alle erstellten Testfälle mit ihren Spezifikationen und Resultaten gespeichert. Beim Ausführen des Regressionstests werden dann alle Testfälle durchlaufen. Falls die aktuellen Resultate nicht mit den gespeicherten übereinstimmen, sollte die letzte Änderung nochmals überarbeitet werden. Abbildung 2.2 zeigt, wie der Ablauf von Regressionstests aussehen könnte. (Kleuker, 2013b, Seite 313-314)

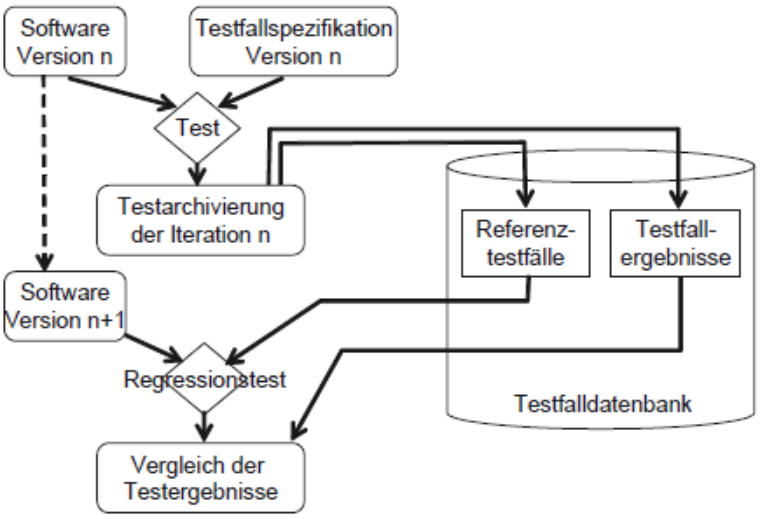

Abbildung 2.2.: Struktur eines Regressionstests (Kleuker, 2013b, Seite 313)

Mit Hilfe von *Simulationstests* können bereits während der Entwicklungsphase gewisse Funktionalitäten simuliert werden, noch bevor diese überhaupt umgesetzt wurden. Der Entwicklungsprozess wird somit nicht aufgehalten. Ein weiterer Vorteil ist die Simulation von Ereignissen oder Systemzuständen, welche nur selten auftreten und sich somit einfach wiederholen lassen. (O'Regan, 2014, Seite 123)

Beim *Abnahmetest* wird nochmals das gesamte System auf den spezifizierten Funktionsumfang geprüft und ist somit vergleichbar mit dem Systemtest. Jedoch wird bei diesem Test der Auftraggeber, falls nicht bereits zu einem früheren Zeitpunkt in den Entwicklungs- und Testprozess integriert, miteinbezogen. (Hoffmann, 2008, Seite 168-169)

2.2. Unit-Test Frameworks

In den Programmier- und Skriptsprachen, welche in dieser Arbeit zum Einsatz kommen, werden bereits Frameworks[1] zum Konstruieren von Unit-

[1] „Rahmen, Gerüst" (Technische Universität Chemnitz, 2016)

Tests bereitgestellt. Diese sind vor allem beim Verwalten und automatisierten Durchlaufen der entwickelten Testfälle behilflich. In den folgenden Abschnitten werden einige dieser Frameworks zusammengefasst dargestellt.

2.2.1. JUnit

Beim Test-Framework JUnit[2] wird zum Erstellen der Testfälle die Programmiersprache Java verwendet und ist dadurch auch bestens für Android Projekte geeignet. Dank der überaus weiten Verbreitung dieser Sprache gibt es eine beachtliche Auswahl an Beiträgen und Hilfswerkzeugen, die beim Aufbau der eigenen Testumgebung als Unterstützung dienen. (Beck, 2004, Seite 8-10)

Ein Grundbaustein von JUnit, sowie auch von allen anderen hier angeführten Test-Frameworks, sind Funktionen mit denen sich gewisse „Aussagen" definieren lassen. Diese befassen sich in den meisten Fällen mit Vergleichen von Werten oder Suchen in gegebenen Objektstrukturen. (JUnit, 2016a)

Eine elegantere und übersichtlichere Lösung zum Kombinieren solcher Aussagen ist die Funktion `assertThat`. Dabei gibt der erste Parameter den zu überprüfenden Wert an. Der zweite Parameter kann dann je nach Belieben eine einzige Aussagen-Funktion, oder auch eine Kombination aus mehreren sein. Die Lesbarkeit der Testfälle wird mit dieser Variante auf jeden Fall erhöht. (JUnit, 2016b)

Weitere nützliche Eigenschaften von JUnit sind natürlich auch das Planen der Testreihenfolge beim Durchführen von Testdurchläufen, sowie das Festlegen von abgekapselten Testumgebungen. Der Vorteil hierbei ist, dass man den Ausgangspunkt der Testausführung genau festlegen kann. Vorgefertigte Objekte und Datenbankinhalte garantieren für gleich bleibende Testvoraussetzungen und haben somit keine Auswirkungen auf den Ablauf der Validierung. (JUnit, 2016c)

[2] http://junit.org/

Funktion	Beschreibung
`async`	Mittels dieser Funktion werden asynchrone Anfragen berücksichtigt.
`equal`	Vergleicht zwei Werte miteinander. Die Abhängigkeiten von Typen und Werten kann durch die Verwendung von ähnlichen Methoden reguliert und verfeinert werden: `deepEqual`, `propEqual`, `strictEqual`.
`ok`	Überprüft, ob das erhaltene Resultat wahr ist.
`push`	Ermöglicht das Einbinden von benutzerdefinierten Aussagen.
`throws`	Kontrolliert, ob aufgetretene Fehler korrekt abgefangen werden.
`module`	Mit dieser Funktion können ähnliche Aussagen gruppiert werden.

Tabelle 2.2.: Grundlegende QUnit Funktionen (jQuery Foundation, 2016c)

2.2.2. QUnit

In JavaScript bietet sich QUnit[3] für die Realisierung einer Testumgebung an, da hierfür alle notwendigen Funktionen zum Testen von einzelnen Code-Fragmenten bis hin zu kompakten Software-Modulen angeboten werden. (Bongers und Vollendorf, 2011, Seite 631)

Bei diesem Test-Framework handelt es sich um ein Projekt der jQuery Foundation (jQuery Foundation, 2016b), welche eine umfangreiche Palette an Open-Source-Software für Entwickler zur Verfügung stellt. In Tabelle 2.2 werden die wichtigsten Funktionen von QUnit aufgelistet und kurz erläutert.

[3] https://qunitjs.com/

2.2.3. NUnit

Das Test-Framework NUnit ist wie auch die bereits erwähnten Frameworks JUnit und QUnit ein Open-Source-Projekt und wird in Verbindung mit dem .NET-Framework[4] eingesetzt. Ausgangspunkt der Entwicklung war JUnit, jedoch wurde die aktuelle Version von Grund auf neu konzipiert und umgesetzt. (NUnit, 2016)

Neben der herkömmlichen Evaluierung von definierten Aussagen, welche auch in den anderen Test-Frameworks in einer vergleichbaren Form auftreten, bietet NUnit weitere Hilfsmittel, um die Testumgebung noch besser an seine eigenen Bedürfnisse anpassen zu können. Dazu gehört beispielsweise eine umfangreiche Liste mit Attributen und Randbedingungen, die das Angebot an Testmöglichkeiten um ein Vielfaches erweitern. In Tabelle 2.3 befindet sich eine kleine Auswahl von Attributen mit der entsprechenden Beschreibung und Tabelle 2.4 verschafft einen kurzen Überblick über die Zusammenstellung der Randbedingungen. (NUnit Software, 2016c)

2.3. Komponententest

Komponententests und Unit-Tests sind sich im Wesentlichen sehr ähnlich, weshalb es auch durchaus vorkommen kann, dass zwischen diesen zwei Testarten kaum unterschieden wird. In beiden Fällen werden ausgewählte Programmteile gesondert betrachtet und in einer vordefinierten Testumgebung überprüft. Auch die Testdurchläufe und Auswertungen werden automatisiert durchgeführt. Ein auffallendes aber dennoch nicht eindeutiges Unterscheidungsmerkmal ist der zu testende Funktionsumfang. Während bei Unit-Tests bereits die kleinstmögliche Funktionalität getestet wird, betrachtet man bei Komponententests die Funktions- und Verhaltensweisen von gesamten Klassen. (Geirhos, 2011, Seite 823-824)

[4] Entwicklungsplattform von Microsoft - https://msdn.microsoft.com/

Attribut	Beschreibung
`Category`	Damit können einzelne Testfälle zu Kategorien verknüpft werden und somit gemeinsam getestet werden.
`Combinatorial`	Alle Varianten, die sich aus den Testdaten ergeben, werden getestet.
`Description`	Ermöglicht den Testfall zu beschreiben.
`Explicit`	Dieser Test wird nur dann durchgeführt, wenn dieser vor dem Testdurchlauf eigens ausgewählt wurde.
`Ignore`	Testfall wird ignoriert.
`MaxTime`	Die Laufzeit des Testfalles darf höchstens die angegebene Zeitspanne betragen.
`Parallelizable`	Mit diesem Attribut wird angegeben, ob bestimmte Testfälle zur selben Zeit durchgeführt werden sollen.
`Platform`	Für jeden Testfall kann die Version der Plattform angegeben werden, auf welcher dieser dann ausgeführt werden soll.
`Random`	Ermöglicht das Testen mit zufällig generierten Werten.
`Repeat`	Testfälle können mehrmals hintereinander aufgerufen werden.
`RequiresThread`	Dieses Attribut zeigt an, dass der Testfall in einem eigenen Thread ausgeführt werden soll.
`Retry`	Nach dem Fehlschlagen eines Tests kann dieser noch einmal wiederholt werden.
`TestOf`	Beim Testfall können Informationen über die betreffende Klasse hinzugefügt werden.
`Timeout`	Hiermit kann ein Zeitlimit gesetzt werden, damit ein Test nicht zu lange ausgeführt wird.

Tabelle 2.3.: Beschreibung von ausgewählten NUnit Attributen (NUnit Software, 2016a)

Randbedingung	Beschreibung
`AllItemsConstraint`	Mit dieser Bedingung können unter anderem ganze Listen überprüft werden.
`DelayedConstraint`	Ermöglicht eine verzögerte Ausführung von anderen Randbedingungen.
`FileOrDirectoryExistsConstraint`	Prüft, ob der Pfad zur angegebenen Datei oder zum angegebenen Ordner existiert.
`PropertyConstraint`	Damit können Eigenschaften von Objekten und deren Werte geprüft werden.
`RangeConstraint`	Vergleicht den definierten Wertebereich mit dem Testwert.
`RegexConstraint`	Hiermit können reguläre Ausdrücke eingesetzt werden.
`ReusableConstraint`	Diese Randbedingung kann im Gegensatz zu anderen wiederverwendet werden.
`UniqueItemsConstraint`	Erlaubt das Kontrollieren von beispielsweise Listen und überprüft, ob sich darin zwei gleiche Einträge befinden.
`XmlSerializableConstraint`	Überprüft, ob das angegebenen Objekt dem XML-Format entspricht.
`SubstringConstraint`	Durch diese Bedingung kann nach bestimmten Teilstücken von Texten gesucht werden.

Tabelle 2.4.: Beschreibung von ausgewählten NUnit Randbedingungen (NUnit Software, 2016b)

2.3.1. Komponenten in Android

Laut Becker und Pant (2015, Seite 483-484) kann es bei Projekten, welche mit Android umgesetzt werden, gegebenenfalls vorkommen, dass die zu testenden Komponenten auch Funktionen der Programmierschnittstelle von Android nutzen möchten. Solche Programmierschnittstellen werden in weiterer Folge auch als *Application Programming Interface* (API) bezeichnet. Betroffen sind davon vor allem jene Komponenten, die mit der Benutzeroberfläche kommunizieren oder mit anderen Komponenten Daten austauschen.

Nach den Angaben von Becker und Pant (2015, Seite 50) sind für die Interaktionen mit der Benutzeroberfläche die sogenannten `Activities` zuständig. Diese sorgen unter anderem für die Ausgabe des aktuellen Layouts und den darin enthaltenen `Views`, die sich schlussendlich in Form von Bildern, Texten oder Eingabefeldern präsentieren. Für die Verarbeitung von Daten bieten sich noch eine Reihe von weiteren Android-Komponenten an.

So legen Becker und Pant (2015, Seite 171-172, 291) dar, dass sich Services in Android hervorragend zur Handhabung von größeren Datenmengen eigenen, da sie ihre Aufgaben zur gleichen Zeit in einem anderen Prozess durchführen und sie die Benutzeroberfläche somit nicht beinträchtigen können. Des Weiteren erlaubt es die Komponente `Content Provider` interne Daten, sowie auch Daten von externen Anwendungen, mit einfachen Mitteln abzurufen. Bei Bedarf ermöglicht der `Content Provider` sogar die Bereitstellung von Informationen an außenstehende Applikationen.

Becker und Pant (2015, Seite 484-485) deuten jedoch darauf hin, dass sich trotz der Verweise zu anderen Software-Komponenten manche Tests dennoch in einer abgekapselten Testumgebung durchführen lassen. Die API von Android stellt dafür eigens zum Testen eine Sammlung von Klassen bereit. Diese ermöglichen nicht nur die Analyse der Verhaltensweisen von Android-Komponenten in bestimmten Situationen, sondern können auch beim Simulieren von Dateisystem- und Datenbankzugriffen behilflich sein. Die folgenden Klassen stellen eine kleine Auswahl der zu Verfügung stehenden Testklassen der API dar:

- `AndroidTestCase`
- `ActivityUnitTestCase`

- `ServiceTestCase`
- `ProviderTestCase2`

Nach der Beschreibung von Google Inc. (2016c) kommt vorzugsweise die Klasse `AndroidTestCase` zum Einsatz, wenn `Ressourcen` ausgelesen werden sollen oder Informationen benötigt werden, die auf dem `Context` der Anwendung basieren. Für Becker und Pant (2015, Seite 33, 54-55) kann es sich bei den Ressourcen um die unterschiedlichsten Daten handeln. Einen Großteil machen jedoch Layouts, Konfigurationsdateien und multimediale Inhalte aus. Die Context-Objekte sind die Bindeglieder zwischen dem eigentlichen System und den darauf laufendenden Applikationen. Sie werden auch für den Zugriff auf System-Ressourcen, sowie auf Ressourcen der eigenen Applikation, benötigt.

Die Klasse `ActivityUnitTestCase` erlaubt es das Verhalten von Activities mit unterschiedlichen Einstellungen und Abhängigkeiten zu testen. Da es sich nur um eine Simulation handelt, können nicht alle Funktionen einer Activity beim Testen verwendet werden. (Google Inc., 2016a)

Mit der Klasse `ServiceTestCase` können Services getestet werden. Wie auch beim Testen von Activities kann hierbei das Verhalten und die Laufzeit in Abhängigkeit von verschiedenen Einflüssen untersucht werden. (Google Inc., 2016l)

Beim Testen eines Content Providers ist die Klasse `ProviderTestCase2` behilflich. Sie ermöglicht, dass Testfälle, welche beispielsweise auf Systeminformationen zugreifen dürfen, in einer sicheren Testumgebung ausgeführt werden. Das System selbst wird dabei nicht beeinflusst. (Google Inc., 2016o)

2.4. Testen von grafischen Benutzeroberflächen

Ein wichtiges Qualitätsmerkmal einer Applikation sind die funktionstüchtigen Bedienungselemente der grafischen Benutzeroberfläche, auch *Graphical User Interface* (GUI) genannt. Denn diese steht im direkten Kontakt zum Anwender und soll seine Eingaben erwartungsgemäß durchführen. Um

alle notwenigen Bedienungselemente auch nach Änderungen schnell und vollständig testen zu können, werden automatisierte Tests eingesetzt. Im Gegensatz zu herkömmlichen Tests mit Mitarbeiten und Testanwendern geben die automatisierten Tests noch im selben Moment eine lückenlose Rückmeldung über die Resultate der durchgeführten Testläufe. Der Schwerpunkt beim Testen von Benutzeroberflächen liegt in der Überprüfung von Ein- und Ausgaben. Dabei werden möglichst viele Interaktionsvarianten des Anwenders mit den Testfällen nachgebildet. (Becker und Pant, 2015, Seite 477)

Dank der gut strukturierten Dokumentation von jQuery Foundation (2016a) ist klar ersichtlich, dass bei der Verwendung von QUnit das Testen von Oberflächenelementen mit relativ einfachen Mitteln realisierbar ist. Nach der Ansicht von Becker und Pant (2015, Seite 479) bietet Android zwar eigene Klassen zum Testen von grafischen Benutzeroberflächen an, jedoch sind die Umsetzungen, im Gegensatz zu QUnit, mit einem erhöhten Mehraufwand verbunden. Durch den Einsatz des Frameworks Robotium[5] kann das Anfertigen von solchen Tests durchaus erleichtert werden.

2.4.1. Robotium

Wie bereits im letzten Absatz erwähnt ist Robotium ein Test-Framework, welches speziell zum Testen von grafischen Benutzeroberflächen entwickelt wurde. Es kann ausschließlich für Android-Projekte verwendet werden, und ist sowohl auf virtualisierten Android-Umgebungen als auch auf Smartphones einsetzbar. Ziel des Frameworks ist die Erstellung von Tests schneller und komfortabler zu gestalten und dem Tester der Anwendung den Validierungsprozess zu erleichtern. (Robotium, 2016b)

Wenn Testfälle nicht nur automatisiert durchlaufen sondern auch automatisiert erstellt werden sollen, dann empfiehlt sich der Robotium Recorder. Mit Hilfe dieses Werkzeugs lassen sich laut Robotium (2016a) Testfälle in kürzester Zeit konstruieren.

[5] https://github.com/robotiumtech/robotium

2.5. Leistungsanalyse und Optimierungen

Auch wenn die modernen Endgeräte, auf denen die entwickelte Software schlussendlich laufen wird, sehr leistungsstark sind und über großzügigen Speicher verfügen, haben Analysen zur Leistungsverbesserung dennoch ihre Daseinsberechtigung. Denn zu lange Wartezeiten oder ständige Verzögerungen bei der Bedienung können für den Anwender sehr schnell frustrierend werden. Das Ziel solcher Leistungsanalysen ist es Schwachstellen zu entdecken, welche unnötig viele Ressourcen benötigen oder sich erst bei überdurchschnittlicher Laufzeit zu erkennen geben. (Kleuker, 2013c, Seite 305)

Gerade bei Software, die auf mobilen Endgeräten ausgeführt wird, ist die Performanz enorm wichtig. Eine limitierte Stromversorgung und teilweise sparsame Hardwareausstattungen sind ein Grund dafür, dass Applikationen nicht sorglos mit den Systemressourcen umgehen dürfen. (Becker und Pant, 2015, Seite 513)

Google Inc. (2016i) stellt eine Reihe von Hilfsprogrammen bereit, mit denen sich Speicherbedarf, Stromverbrauch oder die Auslastung der *Central Processing Unit* (CPU) einfach verfolgen lassen. Zur genaueren Veranschaulichung befinden sich in Tabelle 2.5 und 2.6 jeweils eine Liste von Werkzeugen, die sich mit dem Stromverbrauch und der Speicherverwendung auseinandersetzen. In den darauf folgenden Tabellen 2.7 und 2.8 werden zwei weitere Auswahlen von nützlichen Werkzeugen vorgestellt, welche sich mit der Auslastung von CPU und der *Graphics Processing Unit* (GPU) beschäftigen.

2.6. Methoden zur Synchronisation von Daten

Nachdem die Synchronisation von Daten gerade im Zusammenhang von mobilen Geräten eine große Rolle spielt, existieren davon viele verschiedene aber auch ähnlich aufgebaute Varianten. Dabei werden von den Unternehmen nicht nur selbst entwickelte Synchronisationsmechanismen verwendet, sondern es wird auch vermehrt mit anerkannten Standards gearbeitet.

Werkzeug	Beschreibung
Batterystats & Battery Historian Walkthrough	Mit diesem Werkzeug können bei Android-Projekten wichtige Informationen zum Stromverbrauch des ausführenden mobilen Geräts angezeigt und auch gespeichert werden.
Battery Historian Charts	Dieses Werkzeug wird für eine genauere Betrachtung der gespeicherten Daten verwendet.

Tabelle 2.5.: Werkzeuge zur Kontrolle des Stromverbrauchs von Android-Applikationen (Google Inc., 2016i)

Werkzeug	Beschreibung
Memory Monitor Walkthrough	Mit Hilfe dieses Werkzeugs kann der Speicherbedarf während der gesamten Laufzeit mitverfolgt werden.
Heap Viewer Walkthrough	Dieses Werkzeug zeigt Informationen zu jenen Objekten an, die sich zurzeit im Speicher befinden.
Allocation Tracker Walkthrough	Auch hiermit kann das Speichern und wieder Freigeben von Objekten überprüft werden und darüber hinaus überflüssige Zugriffe ermittelt werden.

Tabelle 2.6.: Werkzeuge zur Überwachung des Speicherbedarfs bei Android-Applikationen (Google Inc., 2016i)

Werkzeug	Beschreibung
Traceview Walkthrough	Hierbei handelt es sich um ein Werkzeug, mit dem die Ausführungsgeschwindigkeiten von einzelnen Funktionen genauer untersucht werden können.
Systrace Walkthrough	Übernimmt das Speichern von Informationen über den Ausführungsverlauf und liefert darüber eine detaillierte Übersicht.

Tabelle 2.7.: Werkzeuge zur Analyse der CPU Auslastung von Android-Applikationen (Google Inc., 2016i)

Werkzeug	Beschreibung
Hierarchy Viewer Walkthrough	Dieses Werkzeug liefert eine Übersicht über die Anordnung jener Bedienungselemente, welche am Bildschirm angezeigt werden sollen. Unvorteilhafte Verschachtelungen können somit schneller erkannt werden.
Profiling with Hierarchy Viewer	Die Darstellungsgeschwindigkeiten der einzelnen Bedienungselemente können mit diesem Werkzeug miteinander verglichen werden.

Tabelle 2.8.: Werkzeuge zur Analyse der GPU Auslastung von Android-Applikationen (Google Inc., 2016i)

Ein Beispiel dafür ist *Web Distributed Authoring and Versioning* (WEBDAV)[6] mit den dazugehörigen Erweiterungen CARDDAV und CALDAV, die von der *Internet Engineering Task Force* (IETF)[7] bereitgestellt werden. Daboo (2011) beschreibt CARDDAV als Standard für den Umgang mit Kontaktdaten im Internet. Dies beinhaltet das Lesen, Bearbeiten und Verwalten von Informationen der betreffenden Kontakte, welche dabei im vCard-Format[8] dargestellt werden. Um die beinahe selbe Funktionalität auch für Kalenderdaten zu erhalten, wurde nach Angaben von Daboo, Desruisseaux und Dusseault (2007) der Standard CALDAV entwickelt.

Die nachstehenden Unterteilungen bieten einen Überblick über die Umsetzung von Synchronisationsmechanismen namhafter Unternehmen. Diese Ausführungen sollten aber nicht nur als Vergleich dienen, sondern auch die mannigfaltigen Möglichkeiten aufzeigen.

2.6.1. Funambol

Das Unternehmen Funambol hat sich in Laufe der Jahre einen Namen mit sogenannten Cloud-Anwendungen gemacht, die sich um das Speichern und

[6] https://tools.ietf.org/html/rfc4918
[7] https://www.ietf.org/
[8] https://tools.ietf.org/html/rfc6350

Verteilen von Daten im Internet kümmern. Unter anderem ist auch ein Open-Source-Projekt für die Synchronisation von Aufgaben, sowie Kontakt- und Kalenderdaten dabei, welches eine Server-Version, sowie Client-Versionen für unterschiedliche Plattformen zum Testen und Weiterentwickeln bereitstellt. (Funambol Inc., 2016)

Funambol bietet unterschiedliche Varianten von Synchronisationsarten an, wobei der eigentliche Synchronisationsvorgang zwischen dem Client und dem Server immer aus denselben drei Schritten besteht. Im ersten und wichtigsten Schritt werden alle zuletzt getätigten Änderungen am Client und am Server lokalisiert und miteinander verglichen. Davon betroffen sind jene Datensätze, welche neu hinzugefügt, geändert oder gelöscht wurden. Abbildung 2.3 zeigt anhand von Schnittmengen mögliche Kombinationen von geänderten und gleichgebliebenen Datenobjekten. Alle Werte im Bereich *A* und *B* sind unverändert. In *Am* und *Bm* sind jene Werte enthalten, die seit der letzten Synchronisation entweder am Client oder am Server geändert wurden, jedoch noch nie synchronisiert wurden. Im Gegensatz dazu sind die Datenobjekte in *(A-Am)Bm* und *Am(B-Bm)* sowohl auf Client-Seite, als auch auf Server-Seite bekannt, und wurden seit der letzten Synchronisation auf einer dieser beiden Seiten geändert. Im Bereich *AmBm* sind nur synchronisierte Daten zu finden, welche auf Seite *A* und auf Seite *B* geändert wurden. (Funambol Inc., 2011, Seite 17-18)

Beim Vergleichen der Datenobjekte werden Operationen festgelegt, die beim eigentlichen Synchronisationsvorgang im Anschluss ausgeführt werden. Welche Operation für welchen Datensatz ausgeführt wird, ist nicht immer eindeutig. Es kann nämlich auch zu Konflikten zwischen zwei Datensätzen kommen, die entweder manuell, oder in den meisten Fällen durch eigens definierte Vorgaben aufgelöst werden. Bei der Bestimmung von Operationen und Konflikten ist eine Matrix behilflich, welche in 2.9 dargestellt wird. Diese beschreibt alle möglichen Zustände, die beim Vergleichen der Datenobjekte auftreten können. Die Buchstaben *A* und *B* zeigen an, dass diese Werte entweder in *Datenbank A* oder *Datenbank B* übernommen werden. Der Buchstabe *C* gibt an, dass es sich um einen Konflikt handelt, der aufgelöst werden soll. Der Buchstabe *D* bringt zum Ausdruck, dass ein Wert gelöscht wird und der Buchstabe *X* informiert darüber, dass sich nichts in der entsprechenden Datenbank ändern wird. Im zweiten Schritt der Synchronisation werden die ermittelten Operationen hintereinander ausgeführt.

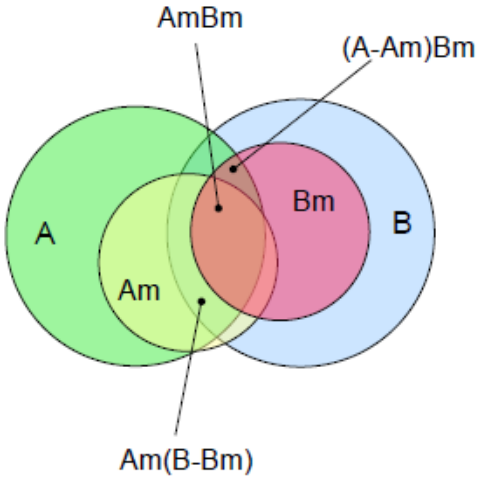

Abbildung 2.3.: Schnittmengen beim Ermitteln von Änderungen für die Datensynchronisation (Funambol Inc., 2011, Seite 18)

Um synchronisierte Werte in Zukunft schneller und eindeutig zuordnen zu können, wird jedes Datenobjekt mit einer einzigartigen Identifikation (ID) gekennzeichnet. Diese Zuordnung der einzelnen ID's muss auf beiden Seiten bekannt sein und wird falls notwendig im dritten und letzten Schritt der Synchronisation angepasst. (Funambol Inc., 2011, Seite 19-20)

2.6.2. Evernote

Evernote war von Beginn an eine Applikation zum Speichern von Notizen und konnte auf unterschiedlichen Plattformen ausgeführt werden. Dieses Produkt wurde in den vergangenen Jahren erfolgreich ausgebaut. Notizen, Aufgabenlisten oder Artikel werden mit allen verwendeten Geräten synchronisiert und können mit Freunden oder Arbeitskollegen geteilt werden. Aus der dadurch entstandenen Ansammlung von Texten und Bildern können Präsentationen erzeugt werden. Diese Funktion macht die Applikation somit zu einem nützlicher Helfer bei gemeinsamen Realisierungen von Projekten. (Evernote Corporation, 2016)

Für die Verwaltung der Daten auf unterschiedlichen Geräten bietet Evernote ein Protokoll mit dem Namen *Evernote Data Access and Management* (EDAM)

Database A → *Database B ↓*	*New*	*Deleted*	*Updated*	*Synchronized/ Unchanged*	*Not Existing*
New	C	C	C	C	B
Deleted	C	X	C	D	X
Updated	C	C	C	B	B
Synchronized/ Unchanged	C	D	A	=	B
Not Existing	A	X	A	A	X

Tabelle 2.9.: Matrix zur Bestimmung von Synchronisationsoperationen (Funambol Inc., 2011, Seite 19)

an. Wie beim zuvor beschriebenen Synchronisationsmechanismus von Funambol, findet hier die Synchronisation ausschließlich zwischen Client und Server statt. Des Weiteren können ebenfalls die gesamten Datenbank-Inhalte, aber auch nur die zuletzt geänderten Datenobjekte synchronisiert werden. Im Gegensatz zur Synchronisation von Funambol wird die Reihenfolge der geänderten Datenobjekte nicht anhand des Datums festgelegt, sondern durch die Vergabe einer *Update Sequence Number* (USN) nach erfolgreichem Eintragen von einzelnen Änderungen. Bei dieser USN handelt es sich um eine fortlaufende Nummer, welche für jeden Benutzerzugang separat verwendet wird und die genaue Abfolge von Datenänderungen bestimmt. Die USN stellt bei der Synchronisation auch eine Markierung dar, um herauszufinden, welche Objekte bereits synchronisiert wurden, und welche Daten noch transferiert werden müssen. Ein weiterer gravierender Unterschied im Vergleich zur vorherigen Vorgehensweise von Funambol ist der Ort, an dem die eigentliche Synchronisation stattfindet. Bei EDAM werden die geänderten Daten direkt am Client miteinander verglichen, während nach Angaben von Funambol Inc. (2011, Seite 12-13) diese Funktion von einem Service am Server übernommen wird. Der Client kümmert sich somit um das Abholen der zuletzt geänderten Daten vom Server, um den Abgleich und um das Senden seiner eigenen Änderungen. Für den Server bietet dieses Vorgehen einige große Vorteile. Der gesamte Datenabgleich wird ausgelagert und sämtliche Synchronisationsfortschritte der einzelnen Clients müssen nicht berücksichtigt werden. (Engberg und Hitchings, 2013, Seite 4-6)

3. Implementierung von Widgets

In diesem Kapitel wird die Onlineplattform Almdesk mit den dazugehörenden Widgets ausführlich vorgestellt. Es folgt somit ein kurzer Überblick über die Gestaltung der Benutzeroberfläche und eine genauere Betrachtung jener Informationen, die im Hintergrund transferiert und gespeichert werden.

Am Ende des Kapitels werden noch die Aufgabengebiete der beiden Kontakt-Widgets, welche für Android und die Web-Version entwickelt wurden, präsentiert, sowie deren Aufbau und Funktionsweise erläutert.

3.1. Almdesk

In Abschnitt 1.1 wurde bereits angeführt, dass es sich bei Almdesk um eine Onlineplattform handelt. Unter Verwendung eines Benutzerzugangs kann nach der Beschreibung von G+Z Software GmbH (2016a) eine große Auswahl an kompakt gehaltenen Programmen genützt werden, um alltägliche Aufgaben auf einer einzigen Plattform miteinander zu vereinen. Tabelle 3.1 enthält ausgewählte Widgets mit der dazugehörigen Beschreibung.

3.2. Unterstützte Plattformen

Zurzeit ist Almdesk sowohl in der mobilen Version, als auch in der Desktop-Version nur über den Browser erreichbar. Um eine korrekte Darstellung der Weboberfläche zu erhalten, empfiehlt G+Z Software GmbH (2016b) die Verwendung der Browser Firefox, Chrome, Internet Explorer (Version 9 oder 10) oder Safari.

Widget	Beschreibung
Aufgaben	Einzelne Aufgaben lassen sich bequem mit dem Aufgaben-Widget erstellen. Diese enthalten auch einen Status der angibt, ob eine Aufgabe bereits erledigt wurde, oder auch nicht.
Chat	Kurze Textnachrichten, sowie Dateien oder auch Aufgaben, können mit dem Chat-Widget an Freunde gesendet werden.
Dokumente	Dieses Widget bietet eine strukturierte Übersicht über alle gespeicherten Dokumente. Dabei kann es sich beispielsweise um Bilder, Audio-Dateien, Textdokumente und noch viele weitere Datei-Formate handeln. Die Dateien können mit anderen Nutzern geteilt, oder auch an beliebige Personen gesendet werden. Dabei darf die Datei eine maximale Größe von 2 Gigabyte (GB) nicht überschreiten.
Feedreader	Mit diesem Widget können komprimierte Inhalte von Webseiten ausgelesen und in Almdesk angezeigt werden.
Freunde	Ermöglicht das Suchen und Verwalten von Almdesk-Kontakten.
Linksammlung	Enthält eine Auflistung von Links zu eingetragenen Webseiten.
Notizen	Das Widget Notizen beherbergt eine Sammlung von Notiz-Texten, welche auch mit anderen Benutzern geteilt werden können.

Tabelle 3.1.: Almdesk Widgets (G+Z Software GmbH, 2016a)

Es gibt Bestrebungen, Almdesk als Applikation für unterschiedliche mobile Betriebssysteme, wie beispielsweise Android, iOS[1], Windows Phone 8[2] oder Windows 10[3], zu entwickeln. Dadurch könnten nicht nur die Bedienungselemente besser an das jeweilige Betriebssystem angepasst werden, sondern es wäre laut Franke und Ippen (2013, Seite 25-26) der Einsatz von hardwarenahen Funktionen wesentlich einfacher und umfangreicher.

3.3. Datenschnittstelle

Sämtliche Informationen, welche die Onlineplattform Almdesk anzeigt, sind auf einem Server abgelegt und werden beim Seitenaufbau geladen. Im Folgenden werden einige Technologien und Konzepte vorgestellt, die in Verbindung mit Almdesk eingesetzt werden und den Datentransfer erst ermöglichen.

3.3.1. ASP.NET

Bei ASP.NET handelt es sich um eine Architektur, die durch den Datenaustausch zwischen Client und Server geprägt wird. Der Client fordert dabei die notwendigen Informationen für die Darstellung der Benutzeroberfläche beim Server an. Der Server liefert hingegen nicht nur vorbereitete Daten, sondern hält auch Schnittstellen zu möglichen Datenbankanbindungen bereit. Für diesen Datentransfer wird vorzugsweise das *Hypertext Transfer Protocol* (HTTP) eingesetzt, wobei auch noch andere Protokolle zum Einsatz kommen können. Der Name ergibt sich einerseits durch die Beziehungen zum bereits erwähnten .NET Framework und andererseits durch die Unterstützung von sogenannten *Active Server Pages* (ASP). Diese können neben den in der Webprogrammierung eingesetzten Inhalten, wie beispielsweise *Hypertext Markup Language* (HTML) Elemente, noch zusätzliche serverspezifische Bausteine umfassen, die die Programmierung erleichtern oder den Funktionsumfang erweitern. (Schwichtenberg, 2013, Seite 50-51)

[1] http://www.apple.com/at/ios/
[2] http://www.windowsphone.com/de-at/how-to/wp8
[3] https://www.microsoft.com/de-de/windows/features

Ein Großteil der Daten wird bei Almdesk jedoch über das *Model View Controller* (MVC) Framework von ASP.NET bezogen. Dies ist die Umsetzung eines bekannten Entwurfsmusters, welches die Daten und deren Darstellung trennt. Für jedes Widget in Almdesk wird ein eigener *Controller* angelegt, der die Anfragen vom Client entgegennimmt. Nachdem der *Controller* die gewünschten Daten vom *Model* erhalten hat, werden diese in die entsprechende Form gebracht, bevor sie an den Client zurückgesendet werden. Das dabei verwendete Rückgabeformat ist *JavaScript Object Notation* (JSON). (Microsoft, 2016)

3.3.2. Internet Information Services

Um ASP.NET verwenden zu können, bedarf es einer Server-Konfiguration, welche mit den Komponenten von ASP.NET auch umgehen kann. Bei Almdesk kommen hierfür *Internet Information Services* (IIS) zum Einsatz. Dies ist eine Auswahl von verschiedenen Programmen, die dabei helfen, Webinhalte zuverlässig und flexibel bereitzustellen. (Schwichtenberg, 2013, Seite 192-194)

IIS bietet auch eine Reihe von Funktionen und Einstellungsmöglichkeiten, welche den Betrieb und die Verwaltung einer Onlineplattform einfacher gestalten. So lassen sich beispielsweise Prozesse und Verbindungen gesondert konfigurieren und protokollieren. Des Weiteren bringt IIS auch Funktionen für eine geordnete und sichere Authentifizierung mit. (Schwichtenberg, 2013, Seite 196)

3.3.3. Authentifizierung

Jeder Benutzer authentifiziert sich auf dem System mit seinen eigens ausgewählten Zugangsdaten. Diese beinhalten den Benutzernamen und das Passwort. Bereits beim Aufruf des Login-Bereichs erhält der Client vom Server eine ID, womit dieser nicht nur eine eindeutige Kennzeichnung erhält, sondern auch eine neue Sitzung für den Client startet. Die Sitzung endet entweder nach einer gewissen Zeitspanne, oder wenn die Client-Anwendung beendet wird. (Schwichtenberg, 2013, Seite 389)

Der Client speichert die ID zur Sitzung als Cookie. Mit dieser Methode kann eine bestimmte Anzahl an Zeichen mit der dazugehörigen Bezeichnung im Speicher abgelegt werden. (Schwichtenberg, 2013, Seite 381-382)

Sobald sich der Benutzer erfolgreich am System anmeldet, werden noch benutzerspezifische Daten in einem dafür vorgesehenen Session-Objekt gespeichert. Dieses beinhaltet nicht nur wichtige Einstellungen, sondern ist auch ein weiteres Merkmal für die Verknüpfung zwischen Benutzer und der aktuellen Sitzung. (Schwichtenberg, 2013, Seite 391)

3.4. Datenbankanbindung

Für das Speichern und Auslesen von größeren Datenmengen stellen die meisten Frameworks Schnittstellen zu diversen Datenbanksystemen bereit. Die in Almdesk implementierten Widgets haben die Möglichkeit auf folgende zwei relationale Datenbanken zugreifen zu können.

3.4.1. Microsoft SQL Server

Bei SQL Server handelt es sich um eine umfangreiche und vielseitige Datenbank-Lösung, die im Laufe der Zeit von den beiden Unternehmen Sybase und Microsoft ständig weiterentwickelt wurde. Als Grundlage für das Abfragen der Daten wird als Standard die *Structured Query Language* (SQL) verwendet. Jedoch bietet SQL Server darüber hinaus noch eine Vielzahl an Erweiterungen und Verbesserungen an. (TechTarget, 2006)

Nahezu jedes Widget in Almdesk verwaltet seine Daten, die von den Benutzern generiert werden, in der dafür vorgesehenen SQL Server Datenbank. Zusätzlich werden dort noch Benutzereinstellungen und Informationen zur statistischen Auswertung abgelegt.

3.4.2. SQLite

Widgets, die ihre Daten in einer eigenständigen Datenbank aufbewahren wollen, können auf die Dienste von SQLite zurückgreifen. Wie auch bei SQL Server wird bei SQLite auf die Grundfunktionalität von SQL gesetzt. Die Datenbank selbst findet in einer einzigen Datei Platz und lässt sich somit schnell und unkompliziert in die Struktur eines Widgets einbinden. (SQLite, 2016)

3.5. Oberflächengestaltung für Web-Plattform

Die Darstellung von Almdesk und den darin enthaltenen Widgets im Browser basiert auf der Nutzung von *Extensible HyperText Markup Language* (XHTML). Dabei handelt es sich um eine strukturierte Anordnung von *Extensible Markup Language* (XML) Elementen, die den eigentlichen Inhalt der Webseite enthalten. Um den Inhalt nicht nur optisch aufzuwerten, sondern auch an unterschiedliche Bildschirmauflösungen anpassen zu können, bedarf es noch der Verwendung von *Cascading Style Sheets* (CSS). Dies ist eine weitere Ansammlung von Definitionen darüber, wie die einzelnen Webseiten-Elemente angezeigt werden sollen. Je nach Browser-Kompatibilität sind hier Formen und Farben so gut wie keine Grenzen gesetzt. (Friedman, 2009, Seite 34-36)

3.5.1. jQuery-Framework

Vergleichbar mit anderen aktuellen Web-Plattformen wird bei Almdesk zu Beginn des Seitenaufbaus lediglich die Grundstruktur der Seite geladen. Der eigentliche Seiteninhalt folgt dann nach und nach, um die Ladezeiten für den Anwender zu optimieren. Dieses Vorgehen wird jedoch erst durch den Einsatz der Skriptsprache JavaScript ermöglicht, womit sich aus Sicht des Entwicklers das Einbinden eines JavaScript-Frameworks lohnt. JavaScript-Frameworks kümmern sich nicht nur um die Eigenheiten unterschiedlicher

Plattformen oder Browser, sondern liefern sogleich eine Vielzahl an Erweiterungen und tragen darüber hinaus zu einer schnelleren Entwicklung bei. (Bongers und Vollendorf, 2011, Seite 21)

In Verbindung mit Almdesk wird das JavaScript-Framework jQuery eingesetzt. Die Hauptgründe dafür sind vielfältig. jQuery wird wegen seiner vielen Anhänger ständig weiterentwickelt und verfügt über ein beachtenswertes Sortiment an Funktionen. Davon werden beispielsweise die vom Framework bereitgestellten Mechanismen zum Suchen und Bearbeiten von Elementen der aktuellen Seite, welche über das *Document Object Model* (DOM) genau definiert sind, häufig angewendet. Diese ordnen sämtliche Bedienungselemente, die in den Widgets enthalten sind, an und steuern außerdem noch deren Aufgaben. (Bongers und Vollendorf, 2011, Seite 18-19,709)

Ein weiteres unverzichtbares Konstrukt von jQuery sind Ajax-Anfragen. Mit Hilfe von Ajax werden sämtliche Widget-Inhalte vom Server abgerufen und darüber hinaus an die entsprechende Position weitergereicht. Die Client wird beim Erzeugen der Benutzeroberfläche keineswegs davon beeinflusst, da diese Datenströme größtenteils asynchron durchgeführt werden. Nachdem eine Ajax-Anfrage gesendet wurde, können die bereits vorhandenen Elemente weiter aufgebaut werden. Wenn zu einem späteren Zeitpunkt die Antwort der Ajax-Anfrage beim Client angekommen ist, wird dieser verständigt und kann sich sogleich darum kümmern. Mit dieser Methode ist es auch möglich gleichzeitig mehrere Anfragen zu senden, ohne dabei längere Wartezeiten herbeizuführen. Jedoch der wohl größte Vorteil bei dieser Methode ist die Wiederverwendbarkeit des bereits geladenen Inhalts. So muss nicht jedes Mal die gesamte Ansicht neu geladen werden, wenn es nur einer Aktualisierung von einzelnen Informationen bedarf. (Bongers und Vollendorf, 2011, Seite 241-242)

3.5.2. Sass

Für die Anordnung und das Aussehen der einzelnen Inhaltselemente ist bei der Web-Version von Almdesk CSS verantwortlich. Um sich die Arbeit beim Erstellen der unterschiedlichen Definitionen zu erleichtern, setzt Almdesk im Hintergrund auf *Syntactically Awesome StyleSheets* (SASS). Das Design

wird zwar dennoch mit herkömmlichen CSS erstellt, jedoch bietet der Einsatz von SASS weitere Möglichkeiten um die Anordnung, Verschachtelung und Ausgabe von CSS-Anweisungen zu beeinflussen. (Sass, 2016)

3.6. Widget-Entwicklung mit Android

Die Android-Version von Almdesk enthält natürlich dieselben Widgets und Einstellungsoptionen wie die bereits erwähnte Web-Variante. Allerdings ergeben Zugriffsmöglichkeiten auf das Betriebssystem hinsichtlich Hard- und Software einen deutlichen Mehrwert.

3.6.1. Benutzeroberfläche

Wie bereits in 2.3.1 erläutert, besteht die Bedienoberfläche in Android aus einzelnen `Views`, welche wiederum über unterschiedliche `Activities` am Bildschirm ausgegeben werden. Standardmäßig existiert für jedes Widget eine eigene `Activity`, welche den Inhalt des Widgets wiedergeben soll. Je nach Funktionsumfang des Widgets kann diese `Activity` unter anderem auch ein Menü beherbergen, oder auf weitere `Activity`-Ansichten verweisen. Dadurch lassen sich die Widgets auf vielfache Weise anpassen und erweitern, damit sie ihren Anforderungen gerecht werden können.

3.6.2. Möglichkeiten zur Datenverwaltung

Im Vergleich zur Web-Version von Almdesk bietet hier Android eine größere Auswahl an Datenverwaltungsformen an. Dies erlaubt das temporäre Zwischenspeichern von beliebigen Daten, falls die Verbindung zum Server abbrechen sollte.

Für das Verwalten von größeren Datenmengen wird in erster Linie die mitgelieferte Datenbank verwendet. Android setzt hierbei auf SQLite, dessen Eigenschaften bereits aus Abschnitt 3.4.2 bekannt sind. Der schonende Umgang mit Ressourcen und die Eigenständigkeit zeigen, dass SQLite

ideale Voraussetzungen für den Einsatz auf mobilen Geräten bietet. (Becker und Pant, 2015, Seite 249-250)

Ein weiteres Konzept zur Speicherung von geringfügigen Informationen bietet die Klasse `SharedPreferences` in Android. Diese Klasse ermöglicht das Speichern und Abrufen von einzelnen Werten unter Angabe einer eindeutigen Bezeichnung. Der Name der Datei, in welcher die Daten-Paare gespeichert werden, sowie der Modus mit dem die Datei geöffnet werden soll, können frei gewählt werden. (Google Inc., 2016m)

Die letzte Speichervariante, die in Verbindung mit Android vorgestellt wird, ist das Dateisystem. Alle Informationen werden dabei in Dateien gespeichert. Der Ort, an dem die Dateien angelegt werden, ist aus Sicherheitsgründen ausschließlich von der ausgehenden Applikation zugänglich. Sofern das verwendete Gerät dies unterstützt, können Dateien auch auf einer externen Speicherkarte verwaltet werden. Der Zugriff auf diese Dateien ist dann aber allen Applikationen, die auf dem Gerät installiert sind, gestattet. (Becker und Pant, 2015, Seite 279-280)

3.6.3. Kommunikation mit dem Server

Damit die Widgets ihren Inhalt auch präsentieren können, müssen zuvor alle dafür notwendigen Informationen vom Server geladen werden. Die Beschaffung der Daten funktioniert im Wesentlichen sehr ähnlich zu den in Abschnitt 3.5.1 vorgestellten Server-Anfragen der Web-Version. Nach dem Aufruf eines Widgets wird als erstes die `Activity` mit einem vordefiniertem Layout erstellt. Im Anschluss daran werden die Widget-Inhalte vom Server mittels asynchroner Anfragen angefordert. Erst wenn eine Antwort angekommen ist, werden die im Layout befindlichen Oberflächenelemente mit den erhaltenen Informationen befüllt.

Falls die gesamten Ladetätigkeiten etwas mehr Zeit in Anspruch nehmen, bietet Android unterschiedliche Ladeanimationen an, um die Wartezeit für den Anwender etwas zu verkürzen. (Becker und Pant, 2015, Seite 143)

3.7. Kontakt-Widget

Das Kontakt-Widget in Almdesk soll mit seiner Funkionalität verschiedene Bedürfnisse erfüllen können. Sofern der Anwender nicht auf die Backup-Funktion des Betriebssystems oder des Smartphone-Herstellers vertrauen möchte, erhält dieser die Möglichkeit seine Kontakte über Almdesk zu sichern. Ein weiterer Anwendungsfall wäre auch die Verknüpfung seiner Almdesk-Freunde mit den am Smartphone gespeicherten Kontakten. Im Folgenden werden die beiden Widget-Varianten, die sich in ihren Aufgaben wesentlich unterscheiden, kurz vorgestellt.

3.7.1. Kontakte auf der Web-Plattform

Die Web-Version von Almdesk verfügt über ein Kontakt-Widget, welches sich ausschließlich mit dem Einfügen, Bearbeiten und Löschen von Kontakten befasst. Das vereinfachte Klassendiagramm in Abbildung 3.1 veranschaulicht, aus welchen Teilen sich die Kontakt-Objekte zusammensetzen. Die Klasse `Contact` repräsentiert unter anderem persönliche Kontaktattribute, wie Name, Geburtsdatum oder auch Bilder. Die Klasse `ContactData` gibt die Definition für allgemeine Daten vor. Darunter fallen Telefonnummern, Mail- und Webadressen. Die Objekte der Klassen `ContactDataAddress` und `ContactDataOrganization` liefen, wie deren Namen bereits verraten, Informationen über Adressen und berufliche Gegebenheiten. Eine genaue Auflistung aller zur Verfügung stehenden Attribute, welche sich nach den Vorgaben von Perreault (2011) orientieren, folgt in Anhang C.1.

3.7.2. Kontakte auf der Android-Plattform

Im Gegensatz zur Web-Version kümmert sich das Kontakt-Widget der Android-Version um die Synchronisation der Kontakte mit dem Server. Deren Bearbeitung ist dennoch über die Kontakt-Applikation von Android möglich. Die Benutzeroberfläche beschränkt sich daher auf das Anzeigen der letzten Änderungen, sowie das Datum der letzten Synchronisation. Der Synchronisationsvorgang selbst wird entweder in bestimmten Intervallen

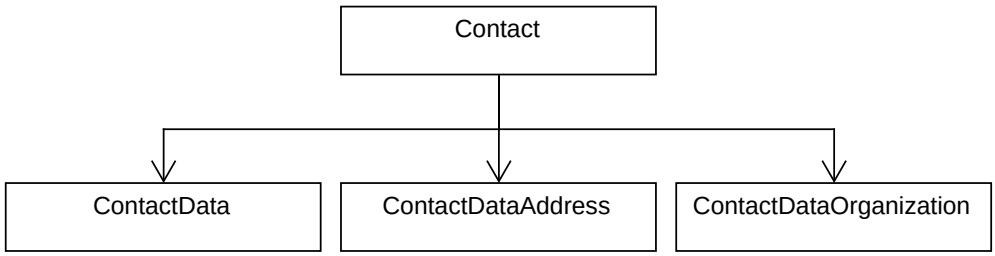

Abbildung 3.1.: Assoziationen der Kontaktdaten Klassen

automatisiert in Gang gesetzt, oder kann auch manuell über den dafür eingerichteten Menüeintrag gestartet werden.

3.7.3. Zugriff auf Android-Kontaktdaten

Um die Kontaktdaten der Android-Plattform synchronisieren zu können, sollten diese zuvor aus der Systemdatenbank ausgelesen werden. Für diesen Schritt muss die Applikation jedoch über bestimmte Genehmigungen verfügen, die bei der Installation durch den Anwender erteilt werden. Alle Genehmigungen werden im Laufe der Entwicklung im Manifest eingetragen, welches noch viele weitere Deklarationen für die Android-Anwendung enthält. Dazu gehören die empfohlene API oder auch die Definitionen zu dein einzelnen `Activities`. (Google Inc., 2016d)

Damit ein kontrollierter und sicherer Zugang zu den Daten anderer Applikationen gewährleistet werden kann, stellt Android eine Funktionalität bereit, die sich Content Provider nennt. Diese ermöglicht es, auf Teilbereiche von Datenbanken zuzugreifen, die im geschützten Bereich von fremden Applikationen liegen, und eigens zu diesem Zweck freigegeben werden. (Becker und Pant, 2015, Seite 291-292)

Eine spezielle Form des Content Providers ist der Contacts Provider, der insbesondere für den Zugriff auf Kontaktdaten bereitgestellt wird. Dabei setzt sich ein Kontakt aus Daten von drei unterschiedlichen Tabellen mit den Namen `Contacts`, `RawContacts` und `Data` zusammen. Die Tabelle `Contacts`

enthält zu jedem einzelnen Kontakt einen Eintrag. Es kann durchaus vorkommen, dass mehrere Benutzerkonten denselben Kontakt eintragen wollen. Damit ein Kontakt nicht mehrfach in der Datenbank liegt, werden identische Einträge mittels der Tabelle `RawContacts` miteinander verknüpft. Die eigentlichen Kontaktdaten, wie Telefonnummer oder Mailadresse, werden in der Tabelle `Data` gespeichert. (Google Inc., 2016e)

Zur Veranschaulichung zeigt der Quellcode 3.1 einen kurzen Ausschnitt darüber, wie das Geburtsdatum eines Kontaktes geändert werden kann. Mit Hilfe der Klassen `Data` und `Event` können Konstanten angesprochen werden, welche entweder die Spaltenbezeichnung der betreffenden Datenbank liefern, oder eine bestimmte Eigenschaft repräsentieren. Dieses Beispiel orientiert sich an der Methode `newInsert`, deren Aufbau und Funktionsweise in Google Inc. (2016e) ausführlich dargestellt werden. Darüber hinaus sind auch eigene Funktionen zur Formatierung des Geburtsdatums enthalten.

```
ops.add(
  ContentProviderOperation.newUpdate(Data.CONTENT_URI)
    .withSelection(Data.CONTACT_ID + "=? and " +
          Data.MIMETYPE + "=? and " +
          Event.TYPE + "=?",
      new String[]{
        String.valueOf(id),
        Event.CONTENT_ITEM_TYPE,
        String.valueOf(Event.TYPE_BIRTHDAY)
        }
    )
  .withValue(Event.START_DATE,
    Utils.getDateStringByPattern(
      Config.FORMAT_DATE,
      birthday)
  )
  .build());
```

Quellcodeverzeichnis 3.1: Aktualisieren des Geburtsdatums

4. Datensynchronisation

Dieses Kapitel beschreibt den allgemeinen Vorgang der Datensynchronisation, welcher ausschließlich vom Client ausgeführt wird. Der Client ist in diesem Fall eine Android Applikation für die Plattform Almdesk, die ein Widget enthält, das sich wiederum um den Status und die Abwicklung der Synchronisation von den Kontaktdaten des Benutzers kümmert. Es folgt daher eine genaue Beschreibung aller beteiligten Komponenten, die sowohl am Client als auch am Server zum Einsatz kommen, und außerdem die Darstellung sämtlicher Abläufe, die den Synchronisationsmechanismus ermöglichen.

Die an dieser Stelle präsentierte Datensynchronisationslösung orientiert sich an den wesentlichen Grundzügen des von Choi u. a. (2010) beschriebenen Algorithmus.

4.1. Bestandteile der Synchronisation

Eine elementare Überlegung zur Synchronisation ist, dass nur jene Daten abgeglichen werden, die sich einerseits seit dem letzten Durchlauf verändert haben oder sich andererseits auf dem Client und auf dem Server unterscheiden. Dafür gibt es bereits bestimmte Strategien, um Kontaktdaten, welche in der Datenbank in verschiedenen Tabellen gespeichert sind, einfach miteinander vergleichen zu können.

4.1.1. Hash und Kennzeichen

Ein möglicher Ansatz wäre nach jedem Bearbeitungsschritt eines Kontaktes eine eindeutige Zeichenkette, bestehend aus Buchstaben und Zahlen, mittels sogenannter Hash-Funktion anzulegen. Spitz, Pramateftakis und Swoboda (2011, Seite 95) beschreiben, dass es unterschiedliche Varianten von Hash-Funktionen gibt. Das Ergebnis einer solchen Funktion enthält jedoch immer die gleiche Anzahl an Zeichen. Die Länge der Zeichenkette kann jedoch durch die Verwendung unterschiedlicher Funktionen variieren. Nicht nur am Client sondern auch am Server wird der *Secure Hash Algorithm* (SHA) zum Erstellen solcher Zeichenketten verwendet, welcher laut Spitz, Pramateftakis und Swoboda (2011, Seite 101-102) für seine Schnelligkeit bekannt ist.

Mit Hilfe dieser Zeichenkette, im Folgenden auch Hash genannt, können Unterschiede zwischen zwei Kontakten mit nur einer Abfrage entdeckt, und im weiteren Verlauf der Synchronisation auch entstandene Konflikte behandelt werden. (Choi u. a., 2010, Seite 393)

Als Ergänzung zum Hash-Wert wäre es noch vorteilhaft, einen Kontakt nach diversen Änderungen auch zu kennzeichnen. Mit dieser Kennzeichnung kann zu Beginn der Synchronisation sofort erkannt werden, ob ein Kontakt seit dem vorherigen Synchronisationsdurchlauf geändert wurde und dadurch automatisch bei der Synchronisation berücksichtigt werden. (Choi u. a., 2010, Seite 394)

4.1.2. Kontaktzuordnung

Die Kontaktdaten werden am Client und auch am Server in einer Datenbank hinterlegt. Um die einzelnen Daten auch gesondert ansprechen, sowie die verschiedenen Kontakteigenschaften mit dem Kontakt selbst verknüpfen zu können, erhalten diese beim erstmaligen Eintragen eine ID. Sobald ein Kontakt, welcher zuvor am Server angelegt wurde, mittels Datenabgleich auf den Client gelangt, wird dieser Datensatz auch am Client nochmals neu eingetragen und erhält dadurch eine weitere ID. Jene Identifikation, die bereits vom Server mitgegeben wurde, dient lediglich als Zuordnung zu den korrespondierenden Daten auf dem Server. Nur aus diesem Grund werden

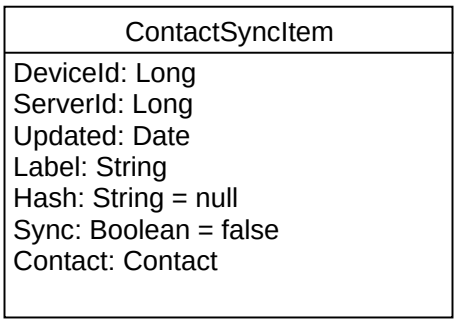

Abbildung 4.1.: Attribute der Klasse `ContactSyncItem`

am Client zwei, in den meisten Fällen unterschiedliche, Identifikationen gespeichert, um bei zukünftigen Änderungen den entsprechenden Kontakt am Server ermitteln zu können.

4.1.3. Synchronisationsobjekt

Um die zuvor kennengelernten Bestandteile zu vereinen, folgt in dieser Textpassage noch eine komprimierte Veranschaulichung der Synchronisationsobjekte. Dabei handelt es sich um Objekte der Klasse `ContactSyncItem`, die bei der Datensynchronisation zwischen Client und Server transferiert werden. Abbildung 4.1 zeigt einen Ausschnitt der Klassenstruktur, welche nur die Attribute betrifft. Darin sind alle bereits bekannten Informationen, wie die beiden ID's, der Hash und das Attribut `Sync`, das sich auf die Kennzeichnung einer Änderung bezieht. Das Attribut `Updated` enthält das Datum der letzten Änderung, und `Label` gibt einen formatierten Ausdruck des Kontaktnamens wieder. Der wohl wichtigste Teil wird im Attribut `Contact` gespeichert. In einem Objekt der Klasse `Contact` sind, wie bereits in Abschnitt 3.7.1 erwähnt, die eigentlichen Kontaktdaten verborgen.

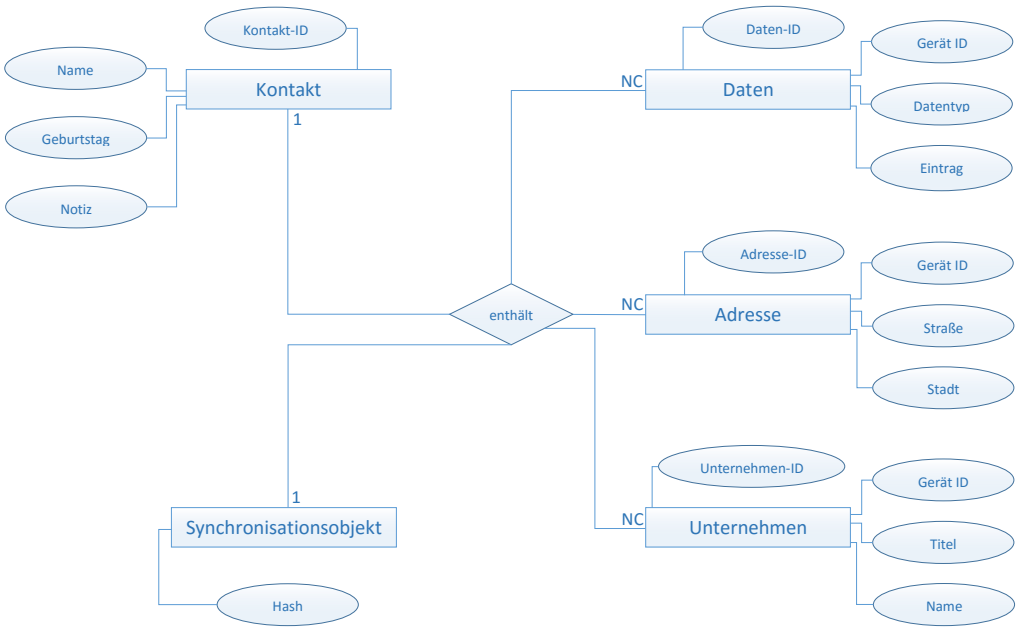

Abbildung 4.2.: Entity-Relationship-Modell der Datenbankstruktur des Kontakt-Widgets

4.2. Datenbankstruktur

Wie schon zuvor in Abschnitt 3.4.2 angeführt, haben Widgets am Server nicht nur das Recht auf die Systemdatenbank zuzugreifen, sondern sie können ihre Daten auch in einer eigenständigen SQLite Datenbank verwahren. Genau diese Alternative wird vom Kontakt-Widget genutzt.

Um einen groben Überblick über die Beziehungen zwischen den Tabellen in der Datenbank zu erhalten, kann, wie auch von Kleuker (2013a, Seite 30) beschrieben, ein *Entity-Relationship-Modell* (ERM) Abhilfe schaffen. Abbildung 4.2 stellt ein solches Modell von der Datenbankstruktur des Kontakt-Widgets am Server dar. Es handelt sich hierbei um eine vereinfachte Form und enthält dadurch nur die Basisattribute, die zum besseren Verständnis dienen sollen.

Erwähnenswert ist beim ERM die ID des Geräts, da diese im bisherigen Verlauf noch nicht besprochen wurde. Aufgrund dessen, dass sich die

erweiterten Kontaktdaten am Client, in diesem Fall ein beliebiges mobiles Gerät, ebenfalls in verschiedenen Tabellen befinden, müssen diese dort eindeutig zugeordnet werden können. Dies garantiert wiederum eine am Client vergebene ID, die dann auch auf den Server übertragen wird.

4.3. Synchronisationsalgorithmus

In diesem Teil des Kapitels wird der Synchronisationsalgorithmus mit seinen einzelnen Ausführungsschritten vorgestellt. Erweitert werden die angeführten Beschreibungen durch aufschlussreiche Diagramme. Diese beinhalten die essentiellsten Komponenten und Informationen und sollen zu einem wesentlich besseren Verständnis beitragen.

4.3.1. Ablauf

Das Aktivitätsdiagramm, zu sehen in Abbildung 4.3, präsentiert den geordneten Ablauf des Synchronisationsmechanismus. Es enthält alle notwendigen Aktionen und beschreibt deren mögliche Reihenfolge bei der Ausführung. Gewisse Abläufe und Ideen orientieren sich an jenem Algorithmus, welcher von Choi u. a. (2010, Seite 394-397) veröffentlicht wurde. Dieser wurde um mehrere Komponenten erweitert und an die speziellen Anforderungen, welche die Plattform Almdesk an die Datensynchronisation stellt, angepasst.

Der Ablauf der Datensynchronisation lässt sich in 3 Stufen unterteilen. Zu Beginn werden zur Vorbereitung die Kontakte am Client durchsucht und auf Änderungen überprüft. Die dadurch erhaltenen Synchronisationsobjekte werden an den Server gesendet, wo ebenfalls alle Änderungen abgerufen werden.

Ab diesem Zeitpunkt beginnt die eigentliche Synchronisation, wobei die Kontaktdaten beider Seiten miteinander verglichen und der Umgebung entsprechend zuerst am Server aktualisiert werden. In diesem Schritt kann es in manchen Fällen zu Konflikten kommen, wenn ein und derselbe Kontakt sowohl am Client als auch am Server geändert wurde. Nachdem die

Konfliktbeseitigung, über welche in Abschnitt 4.5 noch weitere Einzelheiten folgen, durchgeführt wurde und die Kontakte am Server aktualisiert wurden, können die verbleibenden Änderungen an den Client zurückgesendet werden. Um zu gewährleisten, dass die Daten am Client auch wirklich angekommen sind, erfolgt nochmals eine Rückmeldung an den Server und der Synchronisationsschritt wird somit abgeschlossen.

Es kann durchaus vorkommen, dass sich doppelte Kontakteinträge in der Datenbank befinden. Deshalb werden am Ende des Synchronisationsmechanismus sämtliche Kontaktdaten noch einmal durchleuchtet und darüber hinaus nach Duplikaten gesucht.

4.3.2. Kommunikation zwischen Client und Server

In diesem Abschnitt liegt der Schwerpunkt in der Kommunikation, die beim Synchronisationsmechanismus zwischen Client und Server stattfindet. Das in Abbildung 4.4 dargestellte Sequenzdiagramm soll dabei helfen, die Abläufe der Synchronisation noch einmal aus einer anderen Perspektive zu betrachten. Jene Aufgaben des Clients, welche das Überprüfen auf Änderungen und das Suchen nach Duplikaten betreffen, wurden bereits im vorangegangenen Abschnitt erläutert und deshalb wird hier an dieser Stelle nicht erneut darauf eingegangen.

Ebenso der Teil mit dem Transferieren von Datenobjekten ist bereits bekannt, jedoch wird anhand des Sequenzdiagramms ersichtlich, dass sich dieser Schritt mehrmals wiederholen könnte. Grund dafür ist die potenzielle Anzahl an Kontakten, die vom Client und vom Server ausgehen. Das Senden und Bearbeiten von größeren Datenmengen nimmt zweifelsohne mehr Zeit in Anspruch, wodurch sich auch die Wahrscheinlichkeit erhöht, unnötige Zeitüberschreitungen bei den asynchronen Anfragen an den Server herbeizurufen. Um dies zu umgehen werden alle Kontaktdaten, die synchronisiert werden sollen, in mehrere Blöcke aufgeteilt und nacheinander versendet. Wenn der Fall eintritt, dass sich am Client keine Daten geändert haben, werden dennoch solange Anfragen an den Server gestellt, bis alle seine Änderungen am Client angelangt sind.

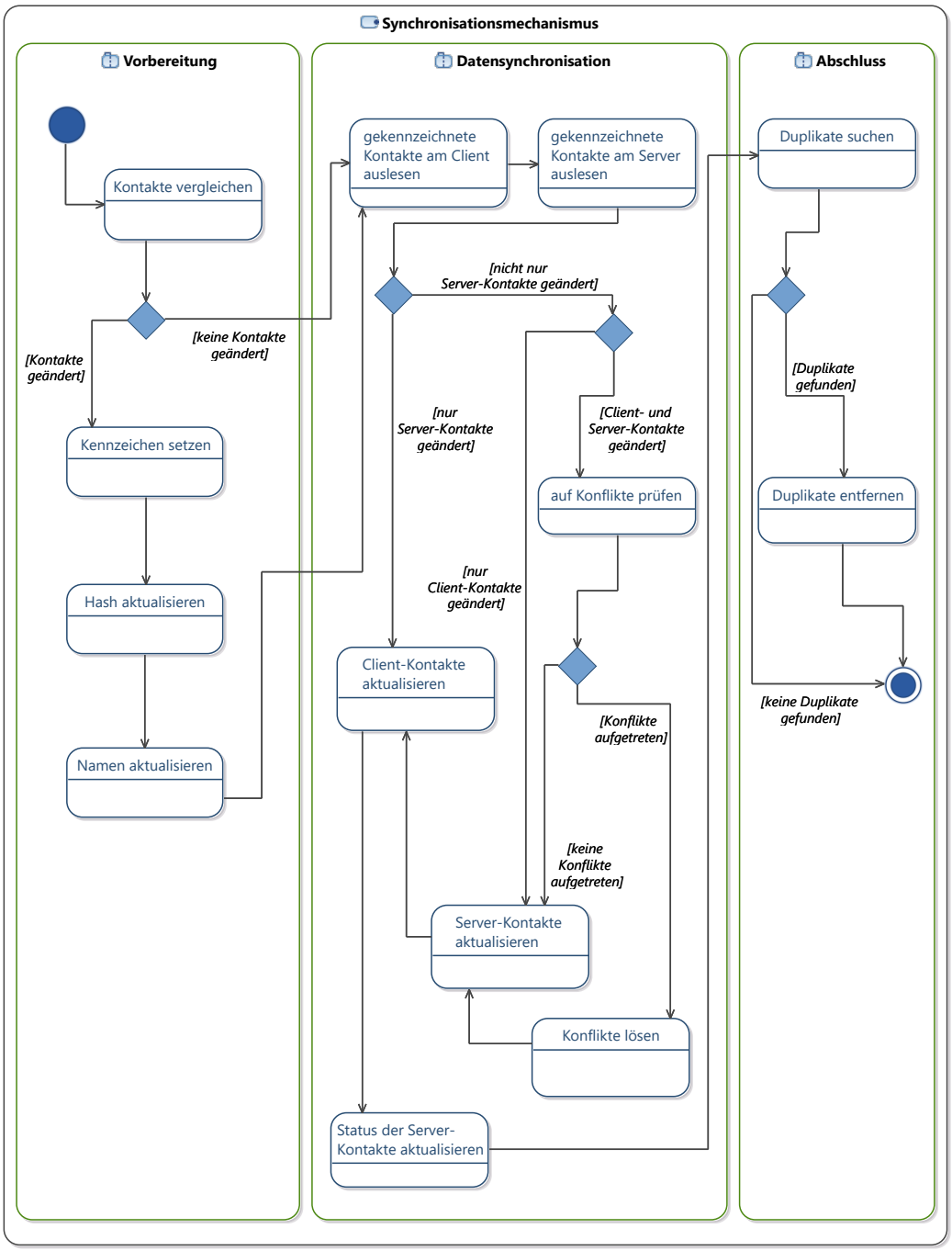

Abbildung 4.3.: Aktivitätsdiagramm zum Synchronisationsmechanismus

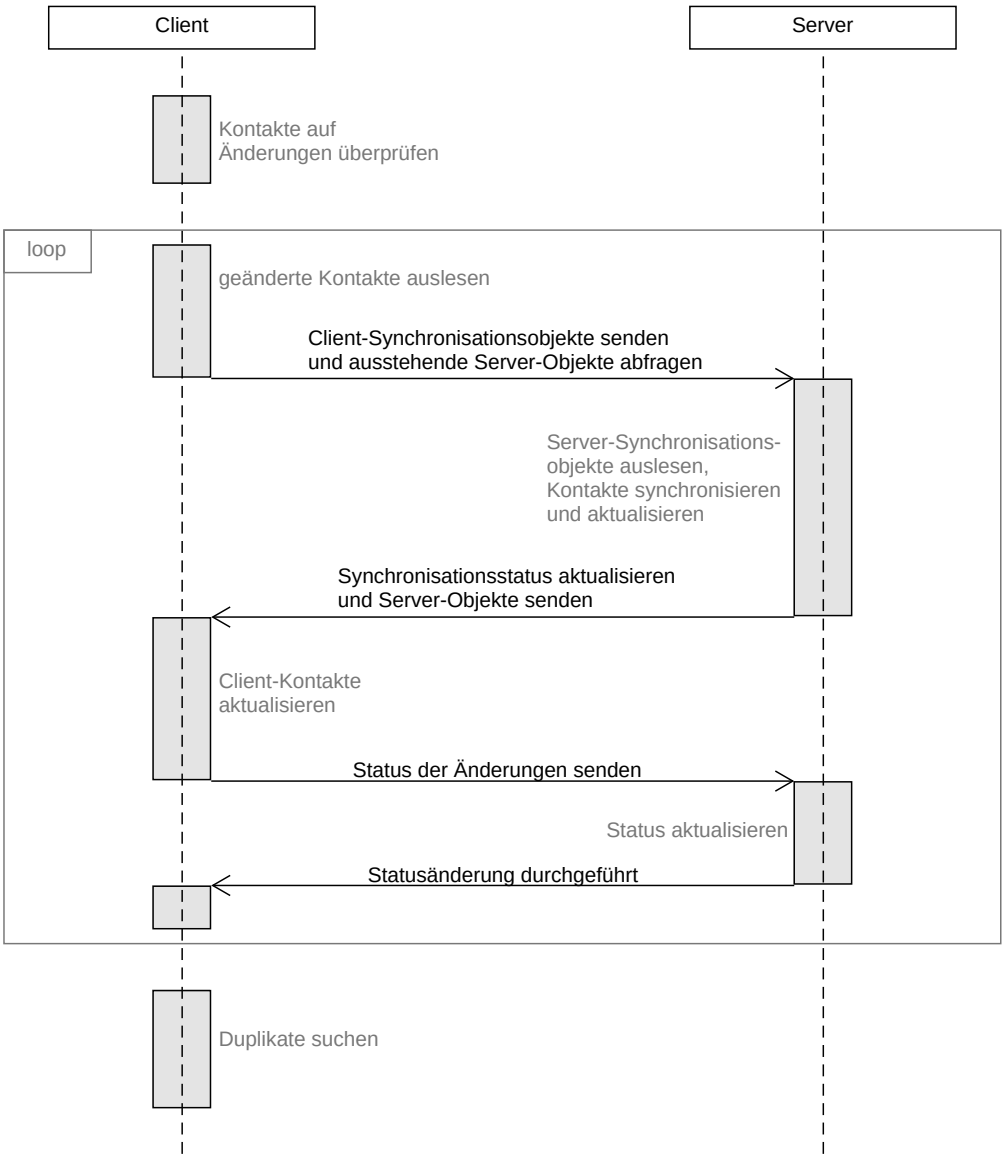

Abbildung 4.4.: Sequenzdiagramm zur Kommunikation zwischen Client und Server

4.4. Datentransfer

Die Datenobjekte der Klasse `ContactSyncItem` können beim Aufruf des Synchronisationsmechanismus nicht in ihrer ursprünglichen Definition übergeben werden, sondern müssen zuvor in eine Form gebracht werden, welche für den Datentransfer geeignet ist und der Server auch verarbeiten kann.

Zu diesem Zweck eignet sich das Datenformat JSON hervorragend. Es ist weit verbreitet und in nahezu allen bekannten Programmiersprachen einsetzbar. Die Struktur ist einfach und variabel gehalten, sodass sich Datenobjekte, wie jene bei der Datensynchronisation von Almdesk, in kürzester Zeit erstellen lassen. Zur besseren Veranschaulichung zeigt Beispiel 4.1 einen kurzen Ausschnitt eines Synchronisationsobjekts, wie es im Format JSON an den Server gesendet wird. Ergänzend dazu befindet sich in Anhang C.2 ein ausführlicheres Beispiel. (Bray, 2014)

```
{"Contact":{
  "Birthday":"1991-01-01 00:00:00",
  "FirstName":"Testbenutzer1",
  "UrlList":[{
    "Data":"https://almdesk.com",
    "DataType":"5",
    "Type":"1",
    "Id":-1,
    "DeviceId":35
  }],
  "Photo":"/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDA...",
  "Id":4
},
"Updated":"2016-02-16 12:10:12",
"Hash":"e21380ede0587add2fec25c0477dda88c9865de7",
"Label":"Testbenutzer1",
"ServerId":18,
"DeviceId":4,
"Sync":true}
```

Quellcodeverzeichnis 4.1: Ausschnitt eines Synchronisationsobjekts im Format JSON

4.5. Konflikte

Konflikte treten auf, wenn ein Kontakt in derselben Synchronisationsperiode sowohl am Client als auch am Server geändert wird. Wenn beispielsweise auf beiden Seiten eine bestimmte Eigenschaft mit dem gleichen Wert belegt wird, so lässt sich der Konflikt ohne Aufwand einfach beheben. Werden jedoch unterschiedliche Eigenschaften verändert, so müssen beide Kontakte getrennt voneinander betrachtet werden.

Für das Auflösen von Konflikten gibt es unterschiedliche Strategien. Eine möglichst benutzerfreundliche Variante wäre es, die betroffenen Kontakte zu speichern. Der Anwender würde zudem eine Meldung erhalten, dass ein Konflikt aufgetreten ist. Im Anschluss daran löst dieser dann den Konflikt manuell auf und erhält am Client und am Server den von ihm gewählten Kontakt. Diese Umsetzung ist vor allem dann sinnvoll, wenn verschiedene Anwender auf dieselben Kontakte zugreifen.

Im Fall von Almdesk verwaltet jeder Benutzer seine eigenen Kontakte, wodurch Konflikte in der Regel nur sehr selten auftreten. Deshalb wird bei der Konfliktbewältigung eine automatisierte Strategie herangezogen. Auch hierbei gibt es wieder vielfältige Möglichkeiten einen Konflikt zu beseitigen. Eine davon ist es, dem Server ein Vorzugsrecht zu geben. Das heißt, dass bei unterschiedlichen Kontakteigenschaften im Zweifel immer die Eigenschaft jenes Kontaktes den Vorzug erhält, welcher am Server gespeichert ist. Wenn nun beim Synchronisieren von Kontaktdaten am Almdesk-Server Differenzen auftreten wird genau diese Strategie für deren Lösung eingesetzt.

4.6. Sync Adapter in Android

Der Synchronisationsmechanismus in Almdesk lässt sich mit der dazugehörigen Android-Applikation auf zwei verschiedene Arten auslösen. Zum einen kann der Mechanismus über das Menü des Kontakt-Widgets aktiviert

werden und zum anderen kann ein Adapter, dessen Grundgerüst von Android bereitgestellt wird, dazu verwendet werden, die Synchronisation in einem vorgegebenen Intervall automatisiert starten zu lassen.

Wenn man in Android immer wiederkehrende Aufgaben im Hintergrund erledigen möchte, führt an der Verwendung des Sync Adapters, laut Google Inc. (2016f), fast kein Weg vorbei. Dieser lässt sich problemlos in eigene Android-Projekte einbinden und vereint einige wichtige Vorteile:

- Applikation muss nicht aktiv sein
- Aufgaben werden getrennt von der Applikation ausgeführt
- Aufgaben werden automatisiert aufgerufen
- Sync Adapter kümmert sich um die Netzwerkverbindung

Auch hinsichtlich der Ausführungszeiten bietet der Sync Adapter eine beachtliche Anzahl an Möglichkeiten. Neben der manuellen und der in regelmäßigen Intervallen automatisch initiierten Aufrufe kann der Sync Adapter auch auf Befehl des Servers aktiviert werden. Darüber hinaus wäre es auch denkbar, diesen Vorgang mit einer weiteren Android-Komponente zu koppeln, welche Änderungen in der Datenbank erkennt und diese über den Sync Adapter an den Server weiterreicht. (Google Inc., 2016k)

5. Umsetzung der Testumgebungen

Welche Testmethoden und Analyseverfahren zur Anwendung kommen, um die Qualität der entwickelten Programme zu erhöhen, wird sich in diesem Kapitel zeigen. Dabei werden hier nicht nur die einzelnen Verfahren vorgestellt, sondern es wird auch anhand von Auswertungen und Vergleichen versucht deren Nutzen zu verdeutlichen.

5.1. Unit-Test Vergleich

Mit Unit-Tests werden in der Regel kleine Code-Teile, sowie einzelne Funktionen durch Angabe von Vergleichswerten auf ihre Richtigkeit überprüft. In dieser Arbeit kommen dafür die beiden Frameworks JUnit und NUnit zum Einsatz, mit denen sich die erstellten Unit-Tests einfach verwalten und automatisiert ausführen lassen.

Ein weiterer Grund für die Verwendung dieser Test-Frameworks ist die Vergleichbarkeit von Funktionalitäten. So werden in der hier besprochenen Umsetzung diese Tests nicht nur zur Identifikation von Fehlverhalten angewendet, sondern sie erlauben es auch, Funktionen, die auf unterschiedlichen Plattformen ausgeführt werden, miteinander zu vergleichen. Das kann auf der einen Seite die Fehleranfälligkeit reduzieren und auf der anderen Seite einen positiven Einfluss auf den weiteren Entwicklungsprozess haben.

Als aussagekräftiges Beispiel wäre hier die Generierung des Hash-Wertes bei der Kontaktsynchronisation zu nennen. Diese Hash-Werte werden am Client wie auch am Server unabhängig voneinander, und durch die Verwendung von zwei unterschiedlichen Programmiersprachen, aus den Kontaktdaten

generiert. Ihre Gleichwertigkeit lässt sich vor allem durch die Anwendung von Unit-Tests schnell und einfach überprüfen, wie auch im Quellcode 5.1 zu sehen ist. Die Funktion `GenerateContactHash` definiert dabei einen Kontakt, der mit allen seinen Attributen zu einer Zeichenkette zusammengefügt wurde. Zudem gibt die Funktion noch jenen Hash-Wert vor, welcher mit dem Ergebnis der Generierung ident sein soll.

```
[Test]
public void GenerateContactHash() {
  Contact contact = new Contact();
  String value =
    "PrefixVornameNachnameSuffixKurzname" +
    "198005051" +
    "mail@test.com";

  String result =
    "d62d66bdb5d39867d97aac058700a7dc4cde4ad6";
  Assert.AreEqual(
    result,
    contact.GenerateContactHash(value)
  );
}
```

Quellcodeverzeichnis 5.1: Überprüfung der Hash-Generierung mittels Unit-Test in C#

5.2. Untersuchungen zur Datenbehandlung

Bei der Erstellung von Android-Apps machen die Entwickler auch zwangsläufig Gebrauch von den dafür bereitgestellten Komponenten. Diese Komponenten sind vor allem bei der Gestaltung der Oberfläche, beim Speichern und Auslesen von Daten, sowie beim Zugriff auf Ressourcen von externen Anwendungen nicht wegzudenken. Auch in diesem Fall können spezielle Testmethoden angewendet werden, um deren Einbindung in die App-Umgebung während der Entwicklung fortlaufend testen zu können. Eine genauere Ausführung darüber ist in den nachfolgenden Unterteilungen dargestellt.

5.2.1. Datenbank

Der Zugriff auf die interne SQLite Datenbank einer App wird über die Klasse `SQLiteOpenHelper` ermöglicht. Sie bietet eine große Auswahl an Funktionen mit deren Hilfe sich die Interaktionen mit der Datenbank steuern lassen. Damit die Datenbank zur selben Zeit nicht mehrmals geöffnet wird, was durchaus Probleme beim Zugriff auf die Daten verursachen könnte, regelt bei der Almdesk-Applikation die Klasse `DatabaseHandler` die Ausführung der Datenbankfunktionen. Diese ist als *Singleton* konzipiert und sorgt dafür, dass es nur eine einzige Verbindungsmöglichkeit zur Datenbank gibt. (Becker und Pant, 2015, Seite 252)

Um die Datenbankverbindung und das korrekte Verhalten der Datenbankzugriffe zu überprüfen, werden anhand von verschiedenen Testfunktionen bestimmte Datenbankaufrufe simuliert. Die Verbindung zwischen Applikation und Datenbank stellt allerdings nicht das herkömmliche `Context`-Objekt her, sondern es wird ein eigens zu Testzwecken zur Verfügung gestelltes Objekt der Klasse `RenamingDelegatingContext` dafür herangezogen. Hierbei erhält der Entwickler die Möglichkeit für die Datenbankzugriffe eine Testdatenbank zu nützen, wobei die Daten in der originalen Datenbank der App unberührt bleiben. (Becker und Pant, 2015, Seite 485)

Das Grundgerüst der Testfälle wird durch die Basisfunktionalität der Klasse `AndroidTestCase` bestimmt. Die Überprüfung der Aussagen basiert auch weiterhin über das Test-Framework JUnit. Der Quellcode 5.2 veranschaulicht nur den wichtigsten Teil einer Testmethode, die prüft, ob jener Kontaktdatensatz aus der Datenbank richtig ausgelesen wird, der zuvor beim Initialisierungsvorgang der Testklasse eingefügt wurde.

```
public void testReadContact() throws Exception {
  String label = "Testbenutzer1", expectedLabel = "";
  SQLiteDatabase db = dbHandler.getReadableDatabase();
  Cursor cursor = db.query("ContactSync",null,"?",
            new String[]{"1"},null,null,null);
  if(cursor != null){
    if(cursor.moveToFirst()){
      expectedLabel = cursor.getString(
        cursor.getColumnIndex("Label"));
    }
```

```
  }
  assertEquals(label, expectedLabel);
}
```

Quellcodeverzeichnis 5.2: Auslesen von Kontaktinformationen aus der SQLite Datenbank

5.2.2. Provider

Wie bereits in Abschnitt 2.3.1 erwähnt, dienen Provider unter anderem als Datenschnittstelle zu anderen Apps. Diese Schnittstelle wird bei der Kontaktsynchronisation genutzt, um an die Daten der Kontakt-Applikation des Betriebssystems zu gelangen. Da es sich hierbei um den Zugriff auf externe Daten handelt, sollte beim Testen stets darauf geachtet werden, dass die Kontaktliste der Testumgebung nicht beeinflusst wird und dadurch weitere Testfälle nicht beeinträchtigt werden. Zu diesem Zweck existieren verschiedene Ansätze, damit die Kontaktdaten ohne weitere Auswirkungen manipuliert werden können. Eine Möglichkeit davon wäre der Einsatz von Robolectric[1].

Bei Robolectric handelt es sich um ein weiteres Framework, das sich auf die Ausführung von Unit-Tests spezialisiert hat. Die Grundfunktionalitäten der Unit-Tests bauen weiterhin auf jene von JUnit auf. Die Besonderheit bei diesem Framework ist jedoch die Tatsache, dass zusätzlich auch Android-Komponenten in die Testfälle integriert werden können, ohne diese auf einem mobilen oder einem emulierten Gerät ausführen zu müssen. Der Grund dafür ist die Bereitstellung und Überschreibung ausgewählter Klassen von Android's *Software Development Kit* (SDK). Somit ist es möglich die Testfälle in der herkömmlichen *Java Virtual Machine* (JVM) laufen zu lassen, woraus sich ein weiterer essentieller Vorteil ergibt. Die Geschwindigkeit, mit der die Tests ausgeführt werden können, wird dadurch wesentlich erhöht und Wartezeiten minimiert. (Robolectric, 2016b)

Mit Hilfe der von Robolectric bereitgestellten Funktionen lassen sich alle benötigten Operationen in Bezug auf die Kontaktliste bequem simulieren. Das System und auch andere Testfälle werden durch Änderungen nicht

[1] http://robolectric.org/

gestört. Falls der Funktionsumfang beim Erstellen von individuellen Tests nicht ausreichend erscheint, kann dieser durch die Erweiterung der so genannten *Shadow*-Klassen beliebig angepasst werden. Dabei handelt es sich um Abbilder von den zuvor erwähnten Android-Klassen, die eigens für einzelne Testfälle ausgebaut werden können. (Robolectric, 2016a)

5.2.3. Shared Preferences

Auch beim Testen der Shared Preferences Funktionalität empfiehlt es sich, die Testdaten von der restlichen Umgebung abzukapseln. Die geänderten Daten bleiben sonst über den Testdurchlauf hinaus bestehen und könnten andere App-Bereiche beeinträchtigen. Eine einfach anwendbare Methode wäre es, für das Abarbeiten der Testfälle eine eigene Dateibezeichnung zu wählen. Die originalen Einträge sind somit von den Teständerungen unabhängig. (Google Inc., 2016m)

Eine weitere Variante wäre, wie auch bereits beim Testen von Providern, der Einsatz von Robolectric. Die Funktionen werden nicht in der herkömmlichen Testumgebung von Android aufgerufen und die Änderungen gehen nach dem Beenden von Robolectric wieder verloren.

Beispiel 5.3 verschafft einen kurzen Einblick in die Anwendungsweise von Robolectric und enthält auch einen Test, der sich mit dem Setzen von Werten über Shared Preferences beschäftigt. Zu Beginn des Tests erhält Robolectric Informationen über die App-Einstellungen, sowie über die zu verwendende SDK-Version. Der Zugriff auf die Shared Preferences Datei geschieht beim Almdesk-App über die Klasse `StorageUtils`, welche Funktionen zum Speichern und Auslesen bereitstellt. Das dazu notwendige `Context`-Objekt wird wiederrum von Robolectric geliefert.

```
@Test
@Config(constants = BuildConfig.class, sdk = 19)
public void testSharedPreferences() throws Exception {
  Context context = RuntimeEnvironment
      .application
      .getApplicationContext();
  String value = "2016-01-01";
```

```
  StorageUtils.setValue(context, "Datum", value);
  String result =
    StorageUtils.getValue(context, "Datum", "");

  assertEquals(value, result);
}
```

Quellcodeverzeichnis 5.3: Interner Speicherzugriff über die Klasse `SharedPreferences`

5.3. Oberflächenbedienung

Um den Anwender von der Qualität einer Benutzeroberfläche überzeugen zu können, reichen moderne Gestaltungsmöglichkeiten, wie ein selbst anpassendes Design oder ereignisgesteuerte Effekte, nicht aus. Es muss auch ein gewisser Grad an Stabilität, sowie eine möglichst kurze Ladezeit gegeben sein.

Die folgenden Untergliederungen sind geprägt von verschiedenen Methoden zur Überprüfung der Darstellung und Aufgaben von einzelnen Oberflächenelementen. Dabei ermitteln die vorgestellten Verfahren, ob die Benutzeroberfläche auch wirklich den Anforderungen entspricht, und tragen dadurch zur Benutzerfreundlichkeit bei.

5.3.1. Elemente des Web-Widgets

Nahezu alle HTML-Elemente, die sich in einem Almdesk-Widget befinden, werden über Funktionen des jQuery-Frameworks eingefügt. Im Zuge dessen können ausgewählte Elemente, die womöglich zur Steuerung der Oberfläche bestimmt sind, ihre Aufgaben erhalten. Diese Aufgaben werden dann zu gegebenen Anlässen, in den häufigsten Fällen auf Grund einer Interaktion des Anwenders mit der Oberfläche, ausgeführt. Je nach Definition löst bereits die Auswahl eines Elements, oder auch die Eingabe von Buchstaben in ein Textfeld, das entsprechende Event aus.

Das Kontakt-Widget in Almdesk beinhaltet viele solcher Steuerungselemente. Sie veranlassen das Hinzufügen und Entfernen von zusätzlichen Eingabefeldern, oder leiten das Speichern des Kontaktes nach getätigten Änderungen ein. Um sicher zu gehen, dass auch wirklich alle Events ordnungsgemäß aufgerufen werden, können diese Auslösevorgänge automatisiert getestet werden. Dabei ist das Test-Framework QUnit behilflich. Es ermöglicht das Erstellen von Testfällen, die sich in diesem Fall um das Auslösen von Events, sowie um die Überprüfung der Ergebnisse kümmern. Bei deren Umsetzung können auch jQuery-Funktionen zur Unterstützung eingesetzt werden. Am Ende eines Testdurchlaufs erhält der Entwickler eine übersichtliche Auflistung mit allen ausgeführten Tests und den dazugehörigen Resultaten.

Das Event, welches beim Kontakt ein neues Eingabefeld für eine weitere Mailadresse hinzufügt, ist ein demonstratives Beispiel dafür, weshalb die genannten Tests überhaupt erst benötigt werden. Dazu liefert der Quellcode 5.4 einen kurzen Überblick über das automatisierte Prüfen von Events in Almdesk. Der hier dargestellte Test zählt mittels Befehl *length* zuerst die Anzahl aller bereits vorhandenen Eingabefelder für Mailadressen. Nach dem Ausführen des Events wird ein weiteres Eingabefeld erwartet. Stimmt die Anzahl mit den Erwartungen überein, ist der Test positiv verlaufen.

```
QUnit.test("contact: add mail", function (assert) {

  var numberBefore = $("table.mail", dialog).length;
  $(".btnAdd.mail", dialog).trigger($.Event("click"));
  var numberAfter = $("table.mail", dialog).length;

assert.deepEqual(numberAfter, numberBefore + 1);
});
```

Quellcodeverzeichnis 5.4: Mail-Event mit QUnit testen

5.3.2. `Activity`-Layout

Zum Testen von Oberflächeneigenschaften stellt Android neben der bereits beschriebenen Klasse `ActivityUnitTestCase` unter anderem auch die

Klasse `ActivityInstrumentationTestCase2` bereit. Mit dieser werden beim Almdesk-App die einzelnen Oberflächenelemente des Kontakt-Widgets auf ihre Präsenz und korrekte Darstellung überprüft.

Um eine bessere Vorstellung von der Umsetzung solcher Kontrollen zu bekommen, ist ein wesentlich vereinfachter Testfall im Quellcode 5.5 enthalten. Hierbei wird getestet, ob die Synchronisationsobjekte auch tatsächlich im Layout der `Activity` geladen werden. Zu diesem Zweck wird mit Mockito[2] ein Testobjekt erstellt und an den entsprechenden Adapter weitergeleitet. Am Ende des Testfalls sollte sich dann das erzeugte Synchronisationsobjekt im Layout befinden.

Mockito ist ein Framework zum Simulieren und Testen von Klassenfunktionen. Dabei kann es sich um Klassen handeln, die entweder noch gar nicht vollständig implementiert wurden, oder auch um Klassen, von denen man nur eine Teilfunktionalität benötigt. Entsprechend der jeweiligen Situation lassen sich die erstellten Testobjekte, auch Mocks genannt, nach Belieben anpassen. Auch deren Verhalten kann bei der Ausführung von Funktionen beobachten werden. (Mockito, 2016)

```
@UiThreadTest
public void testActivityGrid() throws Exception {
  GridView grid = (GridView)activity
    .findViewById(R.id.contact_grid);
  assertNotNull(grid);
  grid.setAdapter(null);

  List<ContactSyncItem> items = new ArrayList<>();
  Mockito mockito = new Mockito();
  ContactSyncItem item =
    mockito.mock(ContactSyncItem.class);
  mockito.when(item.getLabel()).thenReturn("Objekt1");
  items.add(item);

  activity.setSyncItemsAdapter(items);
  assertEquals(1, grid.getCount());
}
```

Quellcodeverzeichnis 5.5: Testfall zum Überprüfen von `View`-Elementen einer `Activity`

[2] http://mockito.org/

5.3.3. Hilfsmittel zur Analyse der Benutzeroberfläche

Bei der Überprüfung der Benutzeroberfläche werden noch zusätzliche Helfer miteinbezogen, mit denen sich gewisse Testvorgänge einfacher gestalten lassen. Sie befassen sich alle mit der Simulation von Benutzereingaben auf der App-Oberfläche und werden in den folgenden Unterteilungen kurz angeführt.

Robotium

Die Grundfunktionalität von Robotium wurde bereits in Abschnitt 2.4.1 diskutiert. Die Struktur der Testfälle baut auf die zuvor kennengelernte Android-Testklasse `ActivityInstrumentationTestCase2` auf. In der Testumgebung von Almdesk wurde Robotium vor allem wegen des erweiterten Funktionsumfangs aufgenommen. Dadurch können Testfälle einfacher und somit schneller erstellt werden, was sich auch positiv auf die Strukturierung und Lesbarkeit auswirkt. (Vogel, 2016)

Espresso

Mit den Funktionen des Frameworks Espresso können ebenfalls Benutzereingaben gesteuert und ausgewertet werden. Espresso erlaubt es auch mehrere aufeinanderfolgende Eingaben zu definieren, die dann sequentiell durchlaufen werden. Bei diesem Vorgang kommt auch das hervorstechendste Merkmal zum Vorschein. Espresso erkennt nämlich selbstständig, ob die Benutzeroberfläche für mögliche Eingaben bereit ist, oder ob im Hintergrund gerade andere Funktionen ausgeführt werden. Durch diese Tatsache muss sich der Entwickler nicht um etwaige Wartezeiten kümmern, was das Erstellen von umfangreicheren Testfällen erheblich vereinfacht. (Google Inc., 2016n)

Monkey

Eine etwas andere Vorgehensweise beim Überprüfen von Oberflächenelementen bietet das Programm Monkey, das sich nicht, wie seine oben genannten Vorgänger, mit vordefinierten Methoden beschäftigt. Die Idee dahinter ist, durch eine große Menge an zufälligen Interaktionen mit der Benutzeroberfläche, beliebige Schwachstellen zu entdecken. Die Interpretation der Ergebnisse bleibt dem Entwickler dann selbst überlassen. (Google Inc., 2016q)

5.4. Laufzeitoptimierungen

In diesem Teil der Arbeit werden sämtliche Methoden gezeigt, die im Zusammenhang mit der Entwicklung der Datensynchronisation unternommen wurden, um sowohl die Ausführungs- und Darstellungsgeschwindigkeiten, als auch die Übertragungsgeschwindigkeiten zu verbessern.

Besondere Aufmerksamkeit wird hierbei der mobilen Applikation von Almdesk zu Teil, da sich performante Lösungsansätze positiv auf die Leistung des Geräts, sowie den Stromverbrauch auswirken können.

5.4.1. Darstellungsverhalten

Der Status des letzten Synchronisationsvorganges befindet sich in der `Activity` des Kontakt-Widgets. Darin enthalten sind das Datum und alle Kontakte, die bei der Synchronisation beteiligt waren. Gerade nach dem ersten Durchlauf, bei dem alle Kontakte erfasst und in die Datenbank eingetragen werden, sind mit großer Wahrscheinlichkeit die meisten Kontaktelemente in der `Activity` angeordnet. Aus diesem Grund sollte eine performante Methode zum Befüllen der Oberfläche angewendet werden, um lange Wartezeiten möglichst zu vermeiden. Es gibt in Android mehrere Varianten, wie eine Gruppe von Elementen, deren Inhalt zuvor aus der Datenbank geladen werden muss, in die Benutzeroberfläche eingepflegt werden kann.

Die hinsichtlich der Umsetzung einfachste Methode ist es, für jedes Element das vordefinierte Layout zu laden, mit Informationen zu füllen und in die dafür bereitgestellte Liste einzufügen. Hierbei müssen jedoch alle Elemente bereits während der Erstellung der `Activity` bereitstehen. Bei einer größeren Anzahl von Elementen kann dies allerdings eine gewisse Zeit in Anspruch nehmen, wodurch der App-Anwender zum Warten gezwungen wird.

Um die Ladezeiten zu verringern, können `AdapterViews` eingesetzt werden. Dies sind spezielle Listen oder Raster, bei denen die darin enthaltenen Elemente über einen Adapter geladen werden. Beim Kontakt-Widget in Almdesk kommt hierfür eine Raster-Ansicht zum Einsatz, da sich damit auf größeren Bildschirmen mehrere Kontakte in einer Zeile anzeigen lassen. Mittels Adapter werden nur jene Elemente hinzugefügt, die auch wirklich auf der Benutzeroberfläche angezeigt werden, weshalb die Ladezeiten wesentlich kürzer sind als im Vergleich zur vorherigen Methode. Auch das Aussehen und die Position werden vom Adapter gesetzt. (Google Inc., 2016g)

Die dritte und letzte Variante, die hier vorgestellt wird, nutzt ebenfalls den Adapter als Datenschnittstelle. Im Unterschied zur vorherigen Umsetzung werden jedoch nicht alle Informationen auf einmal aus der Datenbank geladen, sondern es kommt hierfür ein Loader zur Anwendung. Dieser erhält eine direkte Verbindung zur Datenbank und lädt immer nur einen Bruchteil der Daten, welche gerade für die Darstellung benötigt werden. (Becker und Pant, 2015, Seite 317-318,320)

Tabelle 5.1 enthält die durchschnittlichen Ladezeiten, angegeben in Millisekunden (ms), der `Activity` mit dem jeweiligen Verfahren zur Darstellung der synchronisierten Kontakte. Daraus lässt sich ablesen, dass die erste Variante bei einer größeren Menge von Kontakten zu ineffizient ist. Die Ladetätigkeiten über die `AdapterView` halten sich gerade bei sehr wenigen Kontakten in Grenzen. Wenn jedoch die gesamten Ergebnisse betrachtet werden, dann ist die dritte Variante eindeutig am schnellsten. Sie vereint die Vorteile des Adapters mit jenen der Datenbankverbindung.

Beim Kontakt-Widget von Almdesk kommt, nach der Auswertung dieses Vergleichs, eine modifizierte Form der `AdapterView` zum Einsatz. Abgesehen von der zu Beginn stattfindenden Initialisierung, werden in der Regel nur wenige Kontakte zwischen zwei Synchronisationsdurchläufen geändert

Variante	8 Kontakte	500 Kontakte	1000 Kontakte
einfache Liste	153 ms	5690 ms	10630 ms
AdapterView	36 ms	412 ms	703 ms
Loader	115 ms	119 ms	160 ms

Tabelle 5.1.: Ladezeiten der Kontakt-Widget `Activity`

und dadurch auf der Benutzeroberfläche angezeigt. Somit wäre die zweite Variante die beste Wahl.

5.4.2. Übertragungsgeschwindigkeit

Die Übertragung der Kontaktinformationen spielt im Zuge der Synchronisation eine wichtige Rolle. Dafür werden die Synchronisationsobjekte in das textbasierte Format JSON umgewandelt. Bei der Client-Applikation übernimmt diese Aufgabe Gson[3], währenddessen auf der Server-Seite die JSON-Konstrukte mit dem weit verbreiteten Framework Json.NET[4] erstellt werden.

Im Verlauf der App-Nutzung werden zum Teil nur sehr geringe Datenmengen übertragen. Lediglich beim Initialisierungsschritt, bei dem alle Kontakte synchronisiert werden, kommt es zu einer größeren Ansammlung von Kontaktdaten. Die Übertragungsgeschwindigkeit könnte dabei noch zusätzlich beeinträchtigt werden, wenn viele der betroffenen Kontakte auch Bilder enthalten. Diese werden nämlich ebenfalls als Text übertragen, wodurch sich die Datenmenge signifikant erhöht.

An Hand eines Beispiels wird die Problematik der Datenübertragung hinsichtlich Anzahl und Umfang der synchronisierten Kontakte erläutert. Dabei werden insgesamt 500 Synchronisationsobjekte an den Server gesendet, wobei jede Anfrage nur 50 Kontakte beinhaltet. Zur besseren Veranschaulichung dienen Vergleichswerte und Diagramme, die einerseits aus den Messungen der durchgeführten Testdurchläufe und andererseits aus den

[3] https://github.com/google/gson
[4] http://www.newtonsoft.com/json

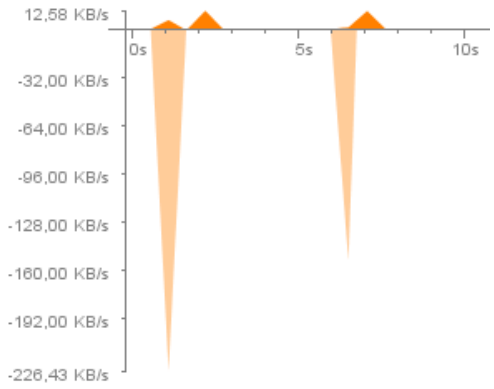

Abbildung 5.1.: Datentransfer von Kontakten ohne Bilder

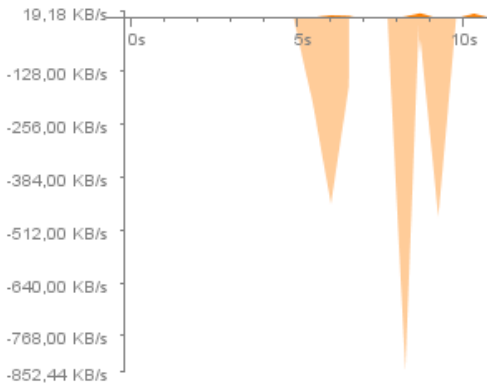

Abbildung 5.2.: Datentransfer von Kontakten mit Bilder

Aufzeichnungen der von Google Inc. (2016b) vorgestellten Hilfswerkzeuge stammen.

Abbildung 5.1 berichtet über den durchschnittlichen Datentransfer von Objekten, die keine Kontaktbilder enthalten. Die negativen Werte kennzeichnen dabei jene Daten, die an den Server gesendet werden und die positiven Werte stehen für die Menge der erhaltenen Daten. Im Vergleich dazu lässt Abbildung 5.2 bereits erkennen, dass sich die Datenmenge durch das Hinzufügen von Kontaktbildern um ein Vielfaches erhöht. Deshalb werden bei der Client-Applikation von Almdesk die Synchronisationsobjekte vor dem Absenden komprimiert, und dann am Server wieder in ihre ursprüngliche Form gebracht. Für diesen Vorgang kommt der Gzip-Mechanismus zum Einsatz, welcher von Gailly (2016) beschrieben wird. Das daraus resultierende Ergebnis erscheint in den beiden Abbildung 5.3 und 5.4, woraus auch eine deutliche Reduktion des Datenvolumens zu erkennen ist.

Der verminderte Datentransfer hat auch Einfluss auf die gesamte Laufzeit der Datensynchronisation. Während die Auswirkungen bei Objekten ohne Kontaktbilder nicht gerade überragend sind, können bei Kontakten mit Bildern doch deutliche Zeitverbesserungen festgestellt werden. Verstärkt wird dieser Effekt noch durch eine höhere Anzahl von Kontakten, in der Tabelle auch als „K/A“ bezeichnet, die bei den Anfragen an den Server gesendet werden. Zum Vergleich sind in Tabelle 5.2 die Laufzeiten der

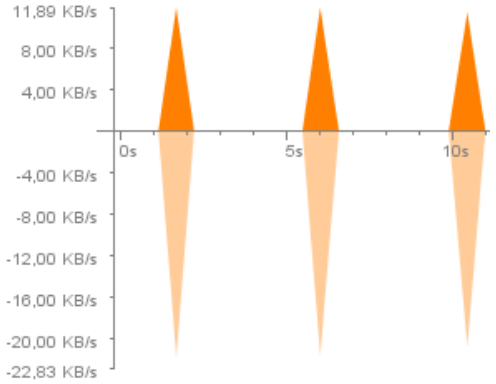

Abbildung 5.3.: Datentransfer von komprimierten Kontakten ohne Bilder

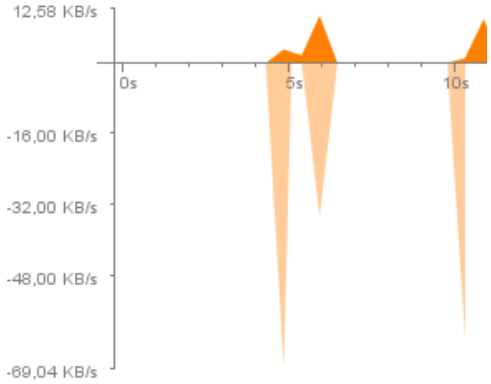

Abbildung 5.4.: Datentransfer von komprimierten Kontakten mit Bilder

	50 K/A	100 K/A	300 K/A
Kontakte ohne Bilder	79 s	70 s	64 s
Kontakte ohne Bilder (komprimiert)	76 s	68 s	59 s
Kontakte mit Bilder	94 s	81 s	54 s
Kontakte mit Bilder (komprimiert)	76 s	67 s	66 s

Tabelle 5.2.: Synchronisationszeiten mit unterschiedlicher Anzahl von Kontakten pro Server-Anfrage

Synchronisation unter den verschiedenen Konstellationen aufgelistet. Die Zeit wird dabei in Sekunden (s) angegeben. Auffällig ist jedoch dabei, dass die Erhöhung der Kontaktmenge nicht unbedingt ein Garant für einen schnelleren Synchronisationsablauf ist.

5.5. Analyse der Android-Applikation

Gerade bei der Erstellung von Android-Applikationen erhält der Entwickler den Zugang zu einem breiten Spektrum von Hilfsmitteln, mit denen sich das Verhalten von Anwendung genau beobachten und analysieren lässt. Egal, ob es sich um die Lokalisierung von Speicher-Problemen handelt, oder ob die Hierarchie der Bedienelemente zu verschachtelt ist. Mit den bereitgestellten Programmen können die unterschiedlichsten Schwachstellen erkannt und dadurch beseitigt werden. Google Inc. (2016i)

5.5.1. Bedienungselemente

Die Grundlage für eine schnelle und fehlerfreie Oberfläche bilden die einzelnen Bedienungselemente. Die Darstellungsgeschwindigkeit, sowie die Verschachtelungen der einzelnen Elemente spielen hierbei eine wichtige Rolle. Mit dem Hilfswerkzeug Hierarchy Viewer können genau diese Eigenschaften der Benutzeroberfläche untersucht werden. (Google Inc., 2016j)

Bei nochmaliger Betrachtung der drei Darstellungsmöglichkeiten, die in Abschnitt 5.4.1 angeführt wurden, ist es mit dem Hierarchy Viewer möglich, die enormen Unterschiede der erstellten `Activity`-Layouts herauszuheben. Die Menge der im Layout eingebetteten `View`-Elemente ist sowohl bei der `AdapterView`-Variante, als auch bei der Loader-Variante dieselbe. Die Raster-Ansicht enthält, auf Grund der Displaygröße des Testgeräts, fünf Kontakt-Elemente, wodurch sich eine Gesamtsumme von 50 Elementen ergibt. Im Vergleich dazu werden bei der langsamsten Variante bei 500 synchronisierten Kontakten insgesamt 5000 Elemente eingefügt, womit sich auch der äußerst aufwendige und langsame Darstellungsprozess erklären lässt.

Eine weitere sehr nützliche Funktion des Hierarchy Viewers ist die Analyse des Aufbaus von einzelnen `View`-Elementen. Dabei werden die betrachteten Elemente mit farblich gestalteten Punkten versehen, die Auskunft über die Performanz der Darstellung in den einzelnen Phasen der Layout-Gestaltung geben. Rote Punkte sind ein Zeichen dafür, dass die betroffenen Elemente ein hohes Optimierungspotenzial haben. Beim Kontakt-Widgets von Almdesk

sind davon jene `TextViews` betroffen, die ein Standardbild anzeigen sollen, wenn für den jeweiligen Kontakt kein Bild existiert. Um dieses Bild für alle Bildschirmauflösungen in einer respektablen Qualität anzuzeigen, wird es über eine eigens dafür definierte Schriftart in einem Textfeld geladen. Diese Umsetzung ist nach Ansicht des Hierarchy Viewers jedoch um einiges langsamer, als wenn das Standardbild direkt über einen Ressourcen-Eintrag gesetzt wird. Mit diesem kleinen Beispiel soll veranschaulicht werden, wie sich mit dem Hierarchy Viewer die Benutzeroberfläche Schritt für Schritt optimieren lässt. (Google Inc., 2016j)

5.5.2. Traceview

Mit dem Hilfswerkzeug Traceview können laut Google Inc. (2016p) sämtliche Abläufe und Methodenaufrufe über einen selbst gewählten Zeitraum aufgenommen werden. Die dadurch angesammelten Daten enthalten zweckmäßige Informationen zur Dauer und Ausführungshierarchie aller aufgerufenen Funktionen und sind somit bei der Beobachtung von verzweigten Mechanismen und Ereignissen hilfreich. Eine kurze Zusammenfassung über den Informationsgehalt der Ergebnisse, welche Traceview zu den einzelnen Funktionen bereitstellt, befindet sich in der nachstehenden Auflistung:

- Anzahl der Funktionsaufrufe
- Anzahl der rekursiven Aufrufe
- Dauer der Funktionsausführung
- Gesamtdauer aller Funktionen, die von der betreffenden Funktion ausgeführt werden

In Verbindung mit der mobilen Applikation von Almdesk wurden nochmals jene Funktionen genauer analysiert, bei denen sich die längste Ausführungszeit ergab. Diese standen unweigerlich in direkter Verbindung mit der Datensynchronisation. Dabei konnten nicht nur unnötige Datenbankzugriffe lokalisiert und entfernt, sondern auch aufwendige und zeitintensive Abläufe optimiert werden.

6. Zusammenfassung

Zum einen wurde für die Plattform Almdesk ein Widget zur Darstellung und Bearbeitung von Kontaktinformationen entwickelt. Dies beinhaltet sowohl eine Version für den Web-Bereich, als auch eine Version für mobile Geräte, speziell für das Betriebssystem Android. Ein Synchronisationsmechanismus dient dabei zur Unterstützung. Zum anderen wurden die erstellten Widgets validiert und durch den Einsatz von unterschiedlichen Frameworks analysiert und optimiert.

Hinsichtlich der Umsetzung der beiden Widget-Versionen wurde die Oberflächenbedienung möglichst einfach und benutzerfreundlich gestaltet. Im Zusammenhang dazu findet die Datensynchronisation des Android-Widgets im Hintergrund statt und gibt nur die wichtigsten Informationen an die Oberfläche weiter.

Bei der Implementierung der Widgets und der Synchronisation wurden auch Testfälle erstellt, weshalb bereits während des Entwicklungsprozesses einige aufgetretene Probleme in kurzer Zeit erkannt und behoben werden konnten. Im Anschluss daran erfolgte eine ausführliche Validierung der entwickelten Komponenten, wobei größtenteils die Android-Applikation davon profitierte. Die daraus resultierenden Optimierungen verbesserten nicht nur den Ladevorgang der Oberflächenelemente, sondern verminderten auch die Zahl der Zugriffe auf die Datenbank, sowie auf den Kontakt-Provider. Jedoch führten nicht alle Bemühungen zum Ziel. Die Laufzeiten der Datensynchronisation wurden mit verschiedenen Konstellationen, in Abhängigkeit von Datenkomprimierung und Anzahl der gesendeten Objekte, miteinander verglichen. Trotzdem lieferten die Ergebnisse keine eindeutigen Aussagen darüber, welches die bestmögliche Variante davon gewesen wäre. Des Weiteren konnten aus zeitlichen Gründen die Speicherverwaltung, sowie der Stromverbrauch der App nur sehr oberflächlich behandelt werden.

7. Ausblick

Dieser Abschnitt beschreibt noch diverse Anregungen und Ideen, die sich für weiterführende Entwicklungen und Validierungen anbieten. Dabei handelt es sich sowohl um Verbesserungsvorschläge, als auch um wünschenswerte Erweiterungen des derzeitigen Entwicklungsstandes.

Eine mögliche Erweiterung des Funktionsumfangs betrifft den Umgang mit Konflikten beim Synchronisationsprozess. Falls es beim aktuellen System zu einem Datenkonflikt kommt, werden jene Änderungen bevorzugt, die am Server getätigt wurden. Durch das Angebot von wenigen Lösungsstrategien können den Benutzern allerdings mehr Freiheiten beim Bearbeiten ihrer eigenen Daten verschafft werden.

Um die Anzahl der Server-Anfragen, die von der Android-Applikation in regelmäßigen Abständen ausgehen, erheblich zu reduzieren, muss der Auslösemechanismus der Datensynchronisation geändert werden. Hierfür beschreibt Google Inc. (2016k) zwei Methoden, die in Verbindung mit dem bereits vorgestellten Sync Adapter angewendet werden können. Dadurch wird der Synchronisationsmechanismus nur gestartet, wenn sich die Daten entweder am Server, oder am Client ändern.

Im Bereich der Software-Validierung ist auch noch einiges an Potenzial vorhanden. So lassen sich, in Bezug auf die Leistungsfähigkeit, mit gewissen Feinabstimmungen durchaus bessere Resultate erzielen. Davon betroffen ist beispielsweise der Datentransfer beim Synchronisieren. Dieser kann durch genauere Untersuchungen der Datenmenge, sowie der zuvor angesprochenen Ausführungsintervalle noch weiter optimiert werden.

In weiterer Folge können solche Anpassungen auch unmittelbare Auswirkungen auf den Stromverbrauch der Applikation haben, welcher speziell bei mobilen Geräten nicht unbeachtet bleiben darf. Aus diesem Grund

stellt Google Inc. (2016h) Informationen bereit, mit deren Hilfe sich der Strombedarf von Applikationen unter bestimmten Umständen verringern lässt. Dabei werden hier, hinsichtlich der Datensynchronisation, wichtige Hinweise zur Netzwerkverbindung angeführt, die für das Almdesk-App einen weiteren positiven Effekt haben könnten.

Anhang

Anhang A.

Abkürzungsverzeichnis

API	Application Programming Interface
ASP	Active Server Page
CardDAV	vCard Extensions to Web Distributed Authoring and Versioning
CalDAV	Calendaring Extensions to Web Distributed Authoring and Versioning
CPU	Central Processing Unit
CSS	Cascading Style Sheets
DOM	Document Object Model
EDAM	Evernote Data Access and Management
ERM	Entity-Relationship-Modell
GB	Gigabyte
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ID	Identifikation

IEC	International Electrotechnical Commission
IETF	Internet Engineering Task Force
IIS	Internet Information Services
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
MVC	Model View Controller
ms	Millisekunde
SASS	Syntactically Awesome StyleSheets
SDK	Software Development Kit
SHA	Secure Hash Algorithm
SQL	Structured Query Language
USN	Update Sequence Number
WebDAV	Web Distributed Authoring and Versioning
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup Language

Anhang B.

Entwicklungsumgebung

Im Rahmen dieser Arbeit kamen sowohl bei der Umsetzung, als auch bei der Validierung der erstellten Software verschiedene Entwicklungsumgebungen zum Einsatz.

B.1. Web-Plattform

Sämtliche Neuentwicklungen und Anpassungen, welche für die Widget-Funktionalität auf der Web-Plattform, sowie für den serverseitigen Synchronisationsteil notwendig waren, wurden mit der von Microsoft bereitgestellten Umgebung Visual Studio 2013[1] getätigt. IIS 8.0 Express[2] fungierte dabei als Webserver.

Damit alle Clients, die sich im lokalen Netzwerk befinden, Zugriff auf die Web-Inhalte haben, muss die Firewall angepasst werden. Wie von Gopalakrishnan (2016) beschrieben, ist auch der folgende Eintrag des `binding`-Elements in der Server-Konfiguration notwendig.

```
<binding protocol="http"
  bindingInformation="*:[Port]:[IP-Adresse]"
/>
```

[1] https://www.visualstudio.com/

[2] http://www.iis.net/learn/extensions/introduction-to-iis-express/iis-80-express-readme

B.2. Android

Für die Erstellung der Android-Applikation wurde die Entwicklungsumgebung Android Studio[3] eingesetzt. Jene Frameworks, die zur Durchführung der Testabläufe verwendet wurden, können bei dieser Umgebung über das darin enthaltene Build-System Gradle[4] hinzugefügt werden.

```
dependencies {
  testCompile 'junit:junit:4.12'
  testCompile 'org.robolectric:robolectric:3.0'
  androidTestCompile fileTree(
    include: 'robotium-solo-5.5.4.jar', dir: 'libs'
  )
  androidTestCompile 'org.mockito:mockito-core:1.+'
  androidTestCompile
  'com.android.support.test.espresso:espresso-core:2.2.1'
}
```

Zusätzlich wird für die Ausführung noch eine aktuelle Version der Android SDK benötigt. Die Mindestanforderung liegt beim Almdesk-App bei Versionsnummer 10.

```
defaultConfig {
  applicationId "com.almdesk"
  minSdkVersion 10
  targetSdkVersion 23
}
```

[3] http://developer.android.com/tools/studio/index.html
[4] https://gradle.org/

Anhang C.

Kontakteigenschaften

C.1. Attribute

Die hier angeführte Liste von Attributen ist an die Definitionen von Perreault (2011) angelehnt und umfasst alle Kontaktinformationen, die beim Synchronisationsvorgang miteinbezogen werden.

Allgemein:

- Vorname
- Zweiter Vorname
- Nachname
- Präfix
- Suffix
- Kurzname
- Geburtsdatum
- Jahrestag
- Foto
- Notiz

Adresse:

- Straße
- Stadt
- Postleitzahl
- Region
- Land

Unternehmen:

- Titel
- Beschreibung der Aufgabe
- Name des Unternehmens
- Bezeichnung der Abteilung

C.2. JSON-Datenstruktur

Beispiel C.1 zeigt eine Liste von Synchronisationsobjekten im Format JSON. In dieser Form werden die Kontakte zum Synchronisieren an den Server gesendet.

```
[{"Contact":{"Id":4,
    "Birthday":"1991-01-01 00:00:00",
    "FirstName":"Testbenutzer1",
    "PhoneList":[{
      "Data":"0123456789",
      "DataType":"2",
      "Type":"1",
      "Id":-1,
      "DeviceId":45
    }],
    "UrlList":[{
      "Data":"https://almdesk.com",
      "DataType":"5",
      "Type":"1",
      "Id":-1,
      "DeviceId":35
    }],
    "OrganizationList":[{
      "Name":"G+Z Software GmbH",
      "Type":"3",
      "Id":-1,
      "DeviceId":47
    }],
    "Photo":"/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDA..."
  },
  "Updated":"2016-02-16 12:10:12",
  "Hash":"e21380ede0587add2fec25c0477dda88c9865de7",
  "Label":"Testbenutzer1",
  "ServerId":18,
  "DeviceId":4,
  "Sync":true
}]
```

Quellcodeverzeichnis C.1: Vollständiges Synchronisationsobjekt im Format JSON

Literatur

Beck, Kent (2004). *JUnit. Pocket Guide*. 1. Aufl. O'Reilly Media. ISBN: 0596007434 (siehe S. 18).

Becker, Arno und Marcus Pant (2015). *Android 5. Programmieren für Smartphones und Tablets*. 4. Aufl. dpunkt.verlag. ISBN: 9783864902604 (siehe S. 23–26, 40, 42, 57, 65).

Bongers, Frank und Maximilian Vollendorf (2011). *jQuery. Das Praxisbuch*. 2. Aufl. Galileo Press. ISBN: 9783836218108 (siehe S. 19, 38).

Bray, Tim (2014). *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC 7159. Internet Engineering Task Force. URL: https://tools.ietf.org/html/rfc7159 (besucht am 14. 02. 2016) (siehe S. 52).

Choi, Mi-Young u. a. (2010). „A Database Synchronization Algorithm for Mobile Devices". In: *IEEE Transactions on Consumer Electronics* 56, S. 392–398. ISSN: 0098-3063. DOI: 10.1109/TCE.2010.5505945 (siehe S. 44, 45, 48).

Daboo, Cyrus (2011). *CardDAV: vCard Extensions to Web Distributed Authoring and Versioning (WebDAV)*. RFC 6352. Internet Engineering Task Force. URL: https://tools.ietf.org/html/rfc6352 (besucht am 19. 01. 2016) (siehe S. 28).

Daboo, Cyrus, Bernard Desruisseaux und Lisa Dusseault (2007). *Calendaring Extensions to WebDAV (CalDAV)*. RFC 4791. Internet Engineering Task Force. URL: https://tools.ietf.org/html/rfc4791 (besucht am 19. 01. 2016) (siehe S. 28).

Engberg, Dave und Seth Hitchings (2013). *Evernote Synchronization via EDAM*. Techn. Ber. Evernote Corporation. URL: https://dev.evernote.com/media/pdf/edam-sync.pdf (besucht am 24. 01. 2016) (siehe S. 31).

Evernote Corporation (2016). *Evernote – der Ort für alle deine Aufgaben | Evernote*. URL: https://evernote.com/intl/de/ (besucht am 24. 01. 2016) (siehe S. 30).

Franke, Florian und Johannes Ippen (2013). *Apps mit HTML5 und CSS3. für iPad, iPhone und Android*. 2. Aufl. Galileo Press. ISBN: 9783836222372 (siehe S. 34).

Friedman, Vitaly (2009). *Praxisbuch Web 2.0. Moderne Webseiten programmieren und gestalten*. 2. Aufl. Galileo Press. ISBN: 9783836213424 (siehe S. 37).

Funambol Inc. (2011). *Funambol Developer's Guide*. Techn. Ber. Funambol Inc. URL: http://sourceforge.net/projects/funambol/files/docs/v9/Funambol-developers-guide-v9.0.pdf/download (besucht am 22. 01. 2016) (siehe S. 29–31).

Funambol Inc. (2016). *Overview*. URL: http://sourceforge.net/projects/funambol/ (besucht am 22. 01. 2016) (siehe S. 29).

Gailly, Jean-loup (2016). *Introduction*. URL: http://www.gzip.org/#intro (besucht am 04. 03. 2016) (siehe S. 67).

Geirhos, Matthias (2011). *Professionell entwickeln mit Visual C# 2010. Das Praxisbuch*. 1. Aufl. Galileo Press. ISBN: 9783836214742 (siehe S. 20).

Google Inc. (2016a). *ActivityUnitTestCase*. URL: http://developer.android.com/reference/android/test/ActivityUnitTestCase.html (besucht am 16. 01. 2016) (siehe S. 24).

Google Inc. (2016b). *Android Monitor*. URL: http://developer.android.com/tools/help/android-monitor.html (besucht am 03. 03. 2016) (siehe S. 67).

Google Inc. (2016c). *AndroidTestCase*. URL: http://developer.android.com/reference/android/test/AndroidTestCase.html (besucht am 16. 01. 2016) (siehe S. 24).

Google Inc. (2016d). *App Manifest*. URL: http://developer.android.com/guide/topics/manifest/manifest-intro.html (besucht am 10. 02. 2016) (siehe S. 42).

Google Inc. (2016e). *Contacts Provider*. URL: http://developer.android.com/guide/topics/providers/contacts-provider.html (besucht am 11. 02. 2016) (siehe S. 43).

Google Inc. (2016f). *Creating a Sync Adapter*. URL: http://developer.android.com/training/sync-adapters/creating-sync-adapter.html (besucht am 15. 02. 2016) (siehe S. 54).

Google Inc. (2016g). *Layouts. Building Layouts with an Adapter*. URL: http://developer.android.com/guide/topics/ui/declaring-layout.html#AdapterViews (besucht am 01. 03. 2016) (siehe S. 65).

Google Inc. (2016h). *Optimizing Battery Life*. URL: http://developer.android.com/training/monitoring-device-state/index.html (besucht am 19. 03. 2016) (siehe S. 73).

Google Inc. (2016i). *Performance Profiling Tools*. URL: http://developer.android.com/tools/performance/index.html (besucht am 20. 01. 2016) (siehe S. 26–28, 69).

Google Inc. (2016j). *Profiling with Hierarchy Viewer*. URL: http://developer.android.com/tools/performance/hierarchy-viewer/profiling.html (besucht am 04. 03. 2016) (siehe S. 69, 70).

Google Inc. (2016k). *Running a Sync Adapter*. URL: http://developer.android.com/training/sync-adapters/running-sync-adapter.html (besucht am 15. 02. 2016) (siehe S. 54, 72).

Google Inc. (2016l). *ServiceTestCase*. URL: http://developer.android.com/reference/android/test/ServiceTestCase.html (besucht am 16. 01. 2016) (siehe S. 24).

Google Inc. (2016m). *Storage Options. Using Shared Preferences*. URL: http://developer.android.com/guide/topics/data/data-storage.html#pref (besucht am 08. 02. 2016) (siehe S. 40, 59).

Google Inc. (2016n). *Testing UI for a Single App*. URL: http://developer.android.com/training/testing/ui-testing/espresso-testing.html (besucht am 27. 02. 2016) (siehe S. 63).

Google Inc. (2016o). *Testing Your Content Provider*. URL: http://developer.android.com/training/testing/integration-testing/content-provider-testing.html (besucht am 16. 01. 2016) (siehe S. 24).

Google Inc. (2016p). *Traceview Walkthrough*. URL: http://developer.android.com/tools/performance/traceview/index.html#WorkingWithTraceview (besucht am 04. 03. 2016) (siehe S. 70).

Google Inc. (2016q). *UI/Application Exerciser Monkey*. URL: http://developer.android.com/tools/help/monkey.html (besucht am 27. 02. 2016) (siehe S. 64).

Gopalakrishnan, Vaidy (2016). *Handling URL Binding Failures in IIS Express*. URL: http://www.iis.net/learn/extensions/using-iis-express/handling-url-binding-failures-in-iis-express (besucht am 13. 03. 2016) (siehe S. 77).

G+Z Software GmbH (2016a). *Almdesk. Eine Plattform - viele Tools für deinen Alltag*. URL: https://almdesk.com/ (besucht am 18. 01. 2016) (siehe S. 4, 32, 33).

G+Z Software GmbH (2016b). *Fragen und Antworten*. URL: https://almdesk.com/de/faq/ (besucht am 02. 02. 2016) (siehe S. 32).

Hoffmann, Dirk W. (2008). *Software-Qualität*. 1. Aufl. Springer. ISBN: 9783540763222 (siehe S. 2, 8, 10, 12–17).

ISO/IEC 25010 (2011). *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. ISO/IEC 25010:2011. Geneva, Switzerland: International Organization for Standardization (siehe S. 7, 9).

ISO/IEC 9126 (2001). *Software engineering - Product quality - Part 1: Quality model*. ISO/IEC 9126-1:2001. Geneva, Switzerland: International Organization for Standardization (siehe S. 7).

jQuery Foundation (2016a). *Cookbook*. URL: http://qunitjs.com/cookbook/ (besucht am 17. 01. 2016) (siehe S. 25).

jQuery Foundation (2016b). *jQuery Foundation*. URL: https://jquery.org/ (besucht am 12. 01. 2016) (siehe S. 19).

jQuery Foundation (2016c). *QUnit API Documentation*. URL: http://api.qunitjs.com/ (besucht am 12. 01. 2016) (siehe S. 19).

JUnit (2016a). *Assertions*. URL: https://github.com/junit-team/junit/wiki/Assertions (besucht am 12. 01. 2016) (siehe S. 18).

JUnit (2016b). *Matchers and assertthat*. URL: https://github.com/junit-team/junit/wiki/Matchers-and-assertthat (besucht am 12. 01. 2016) (siehe S. 18).

JUnit (2016c). *Test fixtures*. URL: https://github.com/junit-team/junit/wiki/Test-fixtures (besucht am 12. 01. 2016) (siehe S. 18).

Kleuker, Stephan (2013a). *Grundkurs Datenbankentwicklung. Von der Anforderungsanalyse zur komplexen Datenbankanfrage*. 3. Aufl. Springer Vieweg. ISBN: 9783658015879 (siehe S. 47).

Kleuker, Stephan (2013b). *Grundkurs Software-Engineering mit UML. Der pragmatische Weg zu erfolgreichen Softwareprojekten*. 3. Aufl. Springer Vieweg. ISBN: 9783658006419 (siehe S. 2, 3, 16, 17).

Kleuker, Stephan (2013c). *Qualitätssicherung durch Softwaretests. Vorgehensweisen und Werkzeuge zum Test von Java-Programmen*. 1. Aufl. Springer Vieweg. ISBN: 9783834809292 (siehe S. 8, 10, 11, 13, 26).

Microsoft (2016). *ASP.NET MVC Overview*. URL: https://msdn.microsoft.com/de-de/library/dd381412.aspx (besucht am 07. 02. 2016) (siehe S. 35).

Mockito (2016). *Features And Motivations*. URL: https://github.com/mockito/mockito/wiki/Features%20And%20Motivations (besucht am 27. 02. 2016) (siehe S. 62).

NUnit (2016). *NUnit*. URL: http://nunit.org/ (besucht am 12. 01. 2016) (siehe S. 20).

NUnit Software (2016a). *Attributes*. URL: https://github.com/nunit/nunit/wiki/Attributes (besucht am 13. 01. 2016) (siehe S. 21).

NUnit Software (2016b). *Constraints*. URL: https://github.com/nunit/nunit/wiki/Constraints (besucht am 13. 01. 2016) (siehe S. 22).

NUnit Software (2016c). *NUnit 3.0 Documentation*. URL: https://github.com/nunit/nunit/wiki (besucht am 13. 01. 2016) (siehe S. 20).

O'Regan, Gerard (2014). *Introduction to Software Quality*. 1. Aufl. Springer. ISBN: 9783319061054 (siehe S. 7, 14–17).

Perreault, Simon (2011). *vCard Format Specification*. RFC 6350. Internet Engineering Task Force. URL: https://tools.ietf.org/html/rfc6350 (besucht am 19. 01. 2016) (siehe S. 41, 79).

Robolectric (2016a). *Extending Robolectric*. URL: http://robolectric.org/extending/ (besucht am 27. 02. 2016) (siehe S. 59).

Robolectric (2016b). *Robolectric. Test-Drive Your Android Code*. URL: http://robolectric.org/ (besucht am 27. 02. 2016) (siehe S. 58).

Robotium (2016a). *FAQ*. URL: http://robotium.com/pages/faq (besucht am 17. 01. 2016) (siehe S. 25).

Robotium (2016b). *User scenario testing for Android*. URL: https://github.com/robotiumtech/robotium/ (besucht am 17. 01. 2016) (siehe S. 25).

Sass (2016). *Sass (Syntactically Awesome StyleSheets)*. URL: http://sass-lang.com/documentation/file.SASS_REFERENCE.html (besucht am 08. 02. 2016) (siehe S. 39).

Schwichtenberg, Dr. Holger (2013). *Microsoft ASP.NET 4.5 mit Visual C# 2012. Das Entwicklerbuch*. 1. Aufl. Microsoft Press. ISBN: 9783866455702 (siehe S. 34–36).

Spitz, Stephan, Michael Pramateftakis und Joachim Swoboda (2011). *Kryptographie und IT-Sicherheit. Grundlagen und Anwendungen*. 2. Aufl. Vieweg+Teubner. ISBN: 9783834814876 (siehe S. 45).

SQLite (2016). *About SQLite*. URL: https://www.sqlite.org/about.html (besucht am 08. 02. 2016) (siehe S. 37).

Technische Universität Chemnitz (2016). *BEOLINGUS Online-Wörterbuch*. URL: http://dict.tu-chemnitz.de/ (besucht am 11. 01. 2016) (siehe S. 4, 17).

TechTarget (2006). *SQL Server*. URL: http://searchsqlserver.techtarget.com/definition/SQL-Server (besucht am 08. 02. 2016) (siehe S. 36).

Vogel, Lars (2016). *Android user interface testing with Robotium - Tutorial*. URL: http://www.vogella.com/tutorials/Robotium/article.html (besucht am 27. 02. 2016) (siehe S. 63).

Wagner, Stefan (2013). *Software Product Quality Control*. 1. Aufl. Springer. ISBN: 9783642385704 (siehe S. 7–9, 15).