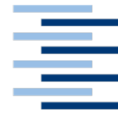

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Stephan Ohm

Testgetriebene Entwicklung einer Android Applikation

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Stephan Ohm

Testgetriebene Entwicklung einer Android Applikation

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Bettina Buth
Zweitgutachter : Prof. Dr. Stefan Sarstedt

Abgegeben am 20.11.2014

Stephan Ohm

Thema der Arbeit
Testgetriebene Entwicklung einer Android Applikation

Stichworte
Testgetriebene Entwicklung, Android, mobile App, Continuous Integration, Test First, Unit-Test, UI-Test, Robolectric, Espresso, Jenkins

Kurzzusammenfassung
Testgetriebene Entwicklung ist eine Methode der Softwareentwicklung, die vom ersten Schritt an ermöglichen soll, eine ganzzeitliche Qualitätssicherung einer Software während der gesamten Entwicklungsphase zu gewährleisten. In dieser Arbeit wird unter Verwendung dieser Methode eine Android App beispielhaft entwickelt, wobei die Eignung verschiedener Werkzeuge für die Anwendung testgetriebener Entwicklung unter Android untersucht wird.

Stephan Ohm

Title of the paper
Testdriven development of an Android application

Keywords
Testdriven development, Android, mobile app, continuous integration, test first, unit-test, ui-test, Robolectric, Espresso, Jenkins

Abstract
Testdriven development is a method of software engineering that enables from the first step quality assurance during the whole phase of development. This paper investigates the suitability of different tools for applying the method of testdriven development for Android.

Inhaltsverzeichnis

1 Einleitung 7

1.1 Motivation 7

1.2 Zielsetzung 7

1.3 Inhaltlicher Aufbau der Arbeit 8

2 Grundlagen 9

2.1 Grundlagen der testgetriebenen Entwicklung 9

2.1.1 Test First 9

2.1.2 Refactoring 10

2.1.3 Continuous Integration 11

2.2 Mobile Applikationen 12

2.2.1 Native App 12

2.2.2 Web App 12

2.2.3 Hybrid App 12

2.3 Android 13

2.3.1 Entstehung 13

2.3.2 Systemaufbau 14

2.3.2.1. Linux-Kernel 14

2.3.2.2. Bibliotheken und Android-Laufzeitumgebung 15

2.3.2.3. Anwendungsrahmen 15

2.3.2.4. Anwendungsschicht 15

2.3.3 Komponenten 15

2.3.3.1. Content Provider 15

2.3.3.2. Status Notification 16

2.3.3.3. Service 16

2.3.3.4. Activity 16

2.3.3.5. Intent 17

2.3.3.6. Broadcast Receiver 17

2.3.4 Android Manifest 18
2.4 Android Test Framework 19
2.4.1 Die Test API 19
2.4.2 Monkey und Monkeyrunner 21
2.5 Projekt Setup / Entwicklungswerkzeuge 22
2.5.1 Development Kits 22
2.5.2 Entwicklungsumgebung: Eclipse 22
2.5.3 Build-Management-Tool: Ant 22
2.5.4 Unit-Test-Framework: Robolectric 23
2.5.5 UI-Test-Framework: Robotium 24
2.5.6 UI-Test-Framework: Espresso 24
2.5.7 Versionskontrollsystem: Git / GitHub 24
2.5.8 Integrationsserver: Jenkins 24
2.5.9 Statische Code-Analyse: Android Lint 25
2.5.10 Android-Plattform 25
3 Testgetriebene Entwicklung mit Android 26
3.1 Anforderungen an das Demonstrationsbeispiel 26
3.1.1 Design der Activities 28
3.1.2 Akzeptanzkriterien 33
3.1.3 Akzeptanztests 34
3.2 Umsetzung von Test First 34
3.2.1 Android Application Project anlegen 34
3.2.2 Projekt unter Versionskontrolle stellen 34
3.2.3 Unit-Test mit dem Android Test Framework 36
3.2.3.1. Anlegen eines Testprojektes 36
3.2.3.2. Umsetzung von Unit-Tests mit dem Android Test Framework 37
3.2.3.3. Ausführung des Testfalls 38
3.2.3.4. Implementierung der Android Komponente 38
3.2.3.5. Erneutes Ausführen des Testfalls 39
3.2.3.6. Integration in den Jenkins 40
3.2.4 Unit-Test mit Robolectric 43
3.2.4.1. Anlegen eines Testprojektes 43
3.2.4.2. Umsetzung von Testfall A1_WelcomeScreen mit Robolectric 44
3.2.4.3. Ausführung des Robolectric Testfalls 45
3.2.4.4. Integration in den Jenkins 46
3.2.4.5. Bewertung 46
3.2.5 UI-Test mit Robotium 46

3.2.5.1. Anlegen eines Testprojektes *46*
3.2.5.2. Umsetzung von Testfall “Registrierung” mit Robotium *47*
3.2.5.3. Umsetzung von Testfall A3_Interaktion_WebView_Nativ mit Robotium .. *49*
3.2.5.4. Implementierung der WebViewNativeInteractionActivity *50*
3.2.5.5. Integration in den Jenkins *52*
3.2.5.6. Bewertung *53*
3.2.6 UI-Test mit Espresso 53
3.2.6.1. Anlegen eines Testprojektes *53*
3.2.6.2. Umsetzung von Testfall A2_Registrierung_Daten mit Espresso *53*
3.2.6.3. Integration in den Jenkins *55*
3.2.6.4. Bewertung *55*
3.3 Weitere Werkzeuge für die Qualitätssicherung 55
3.3.1 Stresstest mit Monkey 55
3.3.1.1. Ausführung von Stresstests mit Monkey *55*
3.3.1.2. Bewertung *57*
3.3.2 Statische Code-Analyse mit Android Lint 57
3.3.2.1. Ausführung von Android Lint *57*
3.3.2.2. Bewertung *58*
4 Fazit 58
Literaturverzeichnis 60
Anhang 62
Abbildungsverzeichnis 62
Tabellenverzeichnis 63

1 Einleitung

Mobile Apps sind aus unserem Alltag nicht mehr wegzudenken. Ständig sieht man Leute, deren Blick auf den Bildschirm ihres Smartphones gerichtet ist. Für fast jeden Zweck gibt es heutzutage die passende App. Der geläufige Motto: „Eine App für alles“ beschreibt das enorme Angebot an unterschiedlichen Apps für Smartphones und Tablets. Der Markt für mobile Apps ist in den letzten Jahren stetig gewachsen. Laut dem Brachen Verband Bitkom [Bitkom2014] werden im Jahr 2014 in Deutschland voraussichtlich 717 Millionen Euro mit Apps umgesetzt, was einer Steigerung von 31% gegenüber dem Vorjahr entspräche, und 3,4 Milliarden Apps heruntergeladen. Dabei sind die Umsätze von Unternehmen, die einen steigenden Anteil ihrer Umsätze über mobile Apps generieren, z.B. Online Versandhändler, noch gar nicht mitgerechnet.
Der wirtschaftliche Erfolg einer mobilen App hängt oftmals direkt mit der Qualität zusammen. So werden gute Apps von Nutzern positiv bewertet. Diese steigen daraufhin im Ranking des jeweiligen App-Stores. Aufgrund der großen Konkurrenz droht eine App mit schlechter Qualität, gar nicht wahrgenommen zu werden.
Die Methode der testgetriebenen Entwicklung soll eine kontinuierliche Qualitätssicherung vom ersten Schritt der Entwicklung an ermöglichen.

1.1 Motivation

Während meines Studiums der Informatik ging ich einer mehrjährigen Tätigkeit als Werkstudent in der Software Qualitätssicherung nach. Seit dieser Zeit beschäftige ich mich viel mit Testautomatisierung und mobilen Apps, insbesondere für Android. Daher möchte ich mich in dieser Arbeit auf Android konzentrieren.

1.2 Zielsetzung

Ziel dieser Bachelorarbeit ist die exemplarische Entwicklung einer App für Android mit der Methode der testgetriebenen Entwicklung. Dabei sollen verschiedene Werkzeuge hinsichtlich ihrer Eignung zur Umsetzung dieser Methode evaluiert werden.

1.3 Inhaltlicher Aufbau der Arbeit

In Kapitel 2 werden die Grundlagen der testgetriebenen Entwicklung erläutert und ein Überblick über Android gegeben. Zudem wird der Begriff „Mobile App" eingegrenzt und die Werkzeuge vorgestellt, die im Rahmen dieser Arbeit zum Einsatz kommen.

Kapitel 3 beinhaltet die Umsetzung testgetriebener Entwicklung mit Android. Zuerst werden die Anforderungen an die zu entwickelnde App formuliert. Anschließend erfolgt die Entwicklung einer Komponente nach der Test First Methode. Hierbei wird der Einsatz zweier Test Frameworks für Unit-Tests demonstriert.

Anschließend erfolgt die testgetriebene Entwicklung unter Verwendung von Akzeptanztests. Hierbei kommen automatisierte Oberflächentests zum Einsatz. Auch hier sollen zwei unterschiedliche Test Frameworks miteinander verglichen werden.

Am Ende von Kapitel 3 wird der Einsatz von Werkzeugen für die Durchführung von Stresstests und statischer Code-Analysen beschrieben.

Den Schluss der Bachelorarbeit bildet das Fazit in Kapitel 4, in welchem die Erfahrungen bei der Umsetzung mit den verschiedenen Werkzeugen zusammengefasst werden.

2 Grundlagen

2.1 Grundlagen der testgetriebenen Entwicklung

Testgetriebene Entwicklung ist eine qualitätsbewusste Methode der Softwareentwicklung, mit der eine Software in kleinen Schritten entwickelt werden kann. Für die Anforderungen an den Code, werden automatisierte Unit Tests (Komponenten Tests) geschrieben. Diese Tests prüfen eine Software in kleinen unabhängigen Einheiten. Sie helfen, die *Software korrekt* zu entwickeln. Zudem werden automatisierte Akzeptanztests für die Anforderung der Kunden geschrieben. Diese Tests prüfen eine Software als große integrierte Systemeinheit. Sie helfen die *richtige Software* zu entwickeln, also eine Software, die diese Kundenanforderungen erfüllt.
Testgetrieben zu arbeiten bedeutet, erst Test Code zu schreiben, bevor der eigentlich zu testenden Programmcode geschrieben wird. Jede funktionale Programmänderung wird durch einen gezielten Test motiviert. Dieser Test wird so entworfen, dass er zunächst fehlschlägt. Das erste Fehlschlagen des Tests ist notwendig, weil die Software die gewünschte Funktionalität noch gar nicht besitzt. Erst im Anschluss wird der Code geschrieben, der die Funktionalität und somit auch den Test erfüllt [Westphal2006].

2.1.1 Test First

Die beschriebene Vorgehensweise mit dem Schreiben von Tests zu beginnen, wird **Test First** genannt. Test First ist im Grunde keine Test- sondern eine Designstrategie. Die Schnittstellen einer zu testenden Komponente werden so bereits benutzt, bevor die Komponente existiert. Dies lässt frühzeitig Aussagen über die Verwendbarkeit des Designs zu.

Im Idealfall sorgt die konsequente Anwendung dieses Vorgehens dafür, dass nicht nur eine hohe Testabdeckung erreicht wird, sondern auch dafür, dass die Architektur unter Aspekten der Testbarkeit entworfen wird. In der testgetriebenen Entwicklung werden zwei Arten von Tests unterschieden: Unit- und Akzeptanztests. Bei beiden Test-Arten handelt es sich um toolunterstütze automatisierte Tests [Westphal2006].

Bei den im weiteren Verlauf dieser Arbeit beschriebenen Unit-Tests handelt es sich ausschließlich um automatisierte, funktionsorientierte Tests auf der Ebene einer einzelnen Komponente. Akzeptanztests werden als automatisierte, funktionsorientierte Tests auf Systemebene behandelt.

Westphal (siehe [Westphal2006] S. 2 f.) stellt drei wesentliche Direktiven auf um testgetriebene Entwicklung weiter zu charakterisieren:

1. Direktive: Motivieren Sie jede Änderung des Programmverhaltens durch einen automatisierten Test.

2. Direktive: Bringen Sie Ihren Code immer in die Einfache Form.

3. Direktive: Integrieren Sie Ihren Code so häufig wie nötig.

Diese drei Direktiven lassen sich drei Techniken zuordnen: Test First, Refactoring und Continuous Integration, im Folgenden kurz CI genannt.

2.1.2 Refactoring

Refactoring ist eine kontrollierte Technik zur Verbesserung der Gestaltung einer vorhandenen Codebasis. Sein Wesen besteht darin, eine Reihe von das Verhalten erhaltenen Änderungen durchzuführen, die für sich gesehen „zu klein“ sind um „Wert zu sein“ angefasst zu werden. Die kumulative Wirkung jeder dieser Transformationen ist jedoch ganz erheblich. Indem man sie in kleinen Schritten durchführt, reduziert man das Risiko dabei Fehler einzuführen [Fowler1999].

Refactorings verfolgen das Ziel einer verbesserten Lesbarkeit und Verständlichkeit des Codes. Weitere Gesichtspunkte, die Refactorings verfolgen, sind die Sicherstellung der Erweiterbarkeit und die Vermeidung von Redundanz.

Im Arbeitszyklus der testgetriebenen Entwicklung, werden Refactorings vor allem zu dem Zeitpunkt durchgeführt, nachdem eine Funktionalität implementiert und der zugehörige Test erfolgreich bestanden wurde. Die erneute Ausführung des Testfalls im Anschluss an das Refactoring sichert zu, dass keine neuen Fehler eingeführt wurden. Die meisten integrierten Entwicklungsumgebungen unterstützen bei der Durchführung von Refactorings.

2.1.3 Continuous Integration

Im klassischen Wasserfallmodell wird meist nur sehr selten und besonders spät im Projektverlauf integriert. Die einzelnen Komponenten, die das fertige Produkt ausmachen werden also erst spät zusammengefügt. Dies kann zu einer Reihe von Problemen führen. Der Ansatz einer kontinuierlichen Integration vom Beginn eines Softwareprojektes an versucht, diese zu reduzieren und die Softwareentwicklung zu beschleunigen [Fowler2006].

Fowler [Fowler2006] beschreibt die folgenden 10 Grundsätze für eine wirkungsvolle CI:

1. **Gemeinsame Codebasis:** Durch Verwendung einer Versionsverwaltung und eines gemeinsamen Repository steht den Entwicklern eine gemeinsame Codebasis zur Verfügung.

2. **Automatisierter Build:** Das Produkt muss vollautomatisch aus seinen Grundbestandteilen übersetzt und zusammengebaut werden können.

3. **Selbsttestender Build:** Durch die Ausführung automatisierter Tests bei jedem Build wird das entstandene Produkt automatisch auf Korrektheit überprüft.

4. **Häufige Integration:** Entwickler sollen ihre Arbeit am Projekt mindestens einmal pro Tag integrieren. Es soll also mindestens einmal am Tag der aktuelle Stand in das Versionskontrollsystem eingecheckt werden.

5. **Jede Änderung (Commit) soll einen automatischen Build auslösen:** Fehler, die durch Änderungen entstehen, können so schneller aufgefunden und behoben werden, sofern die Änderungen klein sind.

6. **Einen fehlschlagenden Build sofort beheben:** Dies soll umgehend erfolgen, da CI ja genau dafür sorgen soll, auf einem bekannten und stabilen Stand zu entwickeln.

7. **Der Build soll schnell gehalten werden:** Nach dem Einchecken einer Änderung sollte das CI-System eine schnelle Rückmeldung geben können, möglichst nach nur wenigen Minuten.

8. **Tests sollen in einem Klon der Produktionsumgebung stattfinden:** Die Umgebung in der die Tests ausgeführt werden sollte der Produktivumgebung möglichst ähnlich sein.

9. **Die ausführbaren Programmdateien sollen leicht zugänglich sein:** Alle Beteiligten sollen einfach an den aktuellen und ausführbaren Stand des Produkts kommen können. Sei es zum Testen, zur Demonstration oder einfach nur um zu sehen, was sich geändert hat.

10. **Jeder kann sehen was passiert**: Die Informationen über den Build, insbesondere über seinen Zustand und Verlauf, sollen allen Beteiligten leicht zugänglich sein.

2.2 Mobile Applikationen

Bei der Entwicklung mobiler Applikationen (im Folgenden kurz App genannt) werden meist drei Arten unterschieden: „Native App“, „Web App“ und „Hybride App“. Im Folgenden werden kurz die wesentlichen Merkmale und Unterschiede erläutert.

2.2.1 Native App

Eine Native App [Verclas2011] wird meist über den jeweiligen App Store (z.B. Apple App Store oder Google Play) heruntergeladen und auf dem Gerät installiert. Die App hat somit die Möglichkeit, auf Funktionen des Gerätes wie z.B. Bewegungssensoren, Kamera, Mikrofon, GPS etc. zuzugreifen. Native Applikationen können außerdem auf das Benachrichtigungssystem der Plattform zugreifen und sogenannte Push-Benachrichtigungen empfangen.

2.2.2 Web App

Eine Web App [Verclas2011] ist im Grunde nicht mehr als eine Webseite, welche im Browser des mobilen Endgerätes angezeigt wird. Web Apps versuchen oft das Design von nativen Applikationen nachzuempfinden, wobei die Darstellung der Inhalte zumeist für die Bildschirmgöße und Eingabemöglichkeiten mobiler Geräte optimiert wird. Techniken, wie z.B. HTML 5, ermöglichen es, Inhalte mit Hilfe des Browser-Cache eine bestimmte Zeit auch ohne aktive Internetverbindung anzuzeigen. Der Zugriff auf Systemfunktionen ist allerdings nur sehr eingeschränkt möglich.

2.2.3 Hybrid App

Hybride Apps [Verclas2011] verbinden die Eigenschaften von Nativen Apps und Web Apps. Es sind praktisch Native Apps, die Inhalte als HTML-Webseiten in einem WebView darstellen, d.h. in einem Browserfester innerhalb der Nativen App. Hybride Apps können somit auf die Systemfunktionen im vollen Umfang zugreifen.

Hybride Apps sind meist nur mit aktiver Verbindung zum Internet im vollen Umfang zu benutzen. Inhalte im WebView werden meist nicht persistiert, was zu häufigeren Nachladen der Seiten führt.

Ein Vorteil der Hybriden App ergibt sich bei der Entwicklung für verschiedene Plattformen, da dieselben HTML-Strukturen für die verschiedenen mobilen Betriebssysteme verwendet werden können. Dies kann zu einer möglichen Kostenreduzierung bei der Cross-Plattform-Entwicklung führen.

2.3 Android

2.3.1 Entstehung

Im Jahr 2003 gründete der Softwareentwickler und ehemalige Apple-Mitarbeiter Andy Rubin in Kalifornien die Firma Android Inc. Bis zum Verkauf der Firma an Google im Jahr 2005 wurde das Betriebssystem in Eigenregie entwickelt. Android wurde von Google 2007 der Öffentlichkeit präsentiert und steht seit Oktober 2008, mit der von Google freigegebenen Version 1.0, Herstellern frei zur Verfügung [Wikipedia2014].

Android ist heute das das meistverbreitetste mobile Betriebsystem der Welt. Dies belegen aktuelle Zahlen des US Informationsdienstleisters IDC.

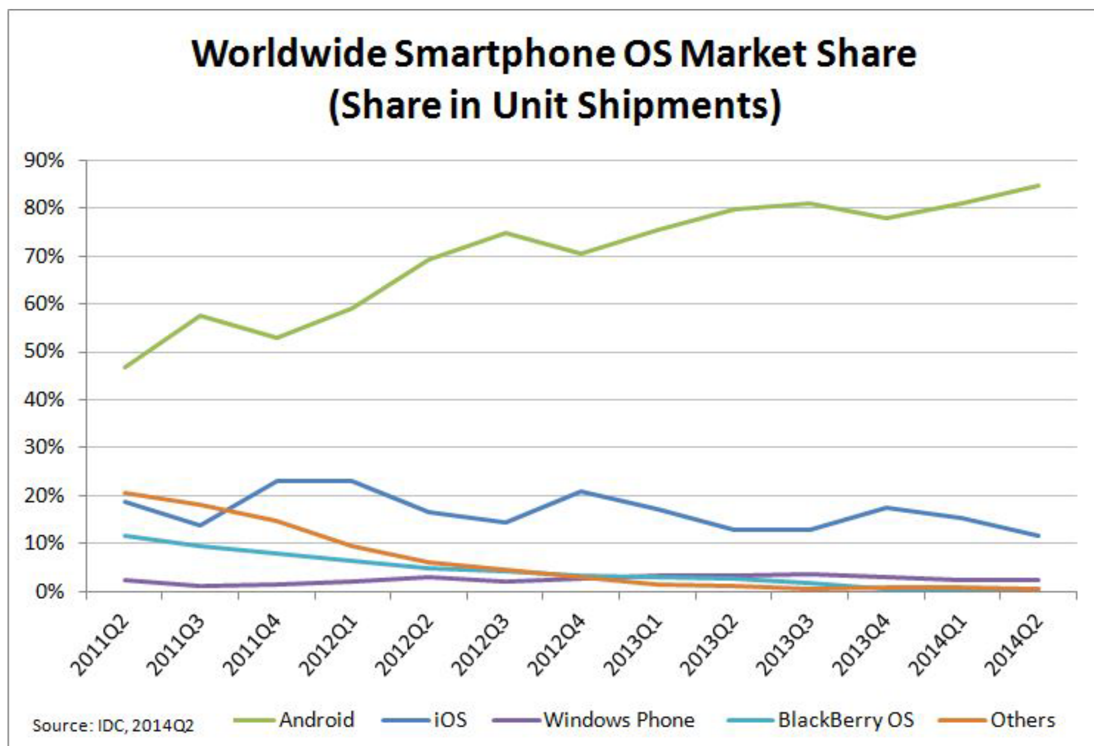

Abbildung 2.1 Weltweite Marktanteile mobiler Betriebssysteme, [IDC2014]

Abbildung 2.1 zeigt, dass Android im 2. Quartal 2014 bei ausgelieferten Smartphones einen weltweiten Marktanteil von über 80% besitzt. Der sich fortsetzende Trend des Android Betriebssystems mag auch damit zusammenhängen, dass es weltweit eine steigende Nachfrage nach low-cost Smartphones gibt (vgl. [IDC2014]) und diese meist mit dem für die Hersteller kostenlosen Betriebssystem Android ausgeliefert werden.

2.3.2 Systemaufbau

Die Systemarchitektur von Android (vgl. [Becker2010] S. 15 ff) ist in vier verschiedene Schichten unterteilt (vgl. Abbildung 2.2). Alle Schichten beinhalten mehrere Komponenten und dienen einem gesonderten Zweck.

2.3.2.1. Linux-Kernel

Android basiert als Betriebssystemgrundlage auf einem Linux-Kernel (anfangs 2.6, ab Android 4.x auch 3.x), der die Schnittstelle zwischen Software und Hardware darstellt. Von hier aus wird z.B. die Speicherverwaltung, die Energieverwaltung oder die Prozessverwaltung gesteuert.

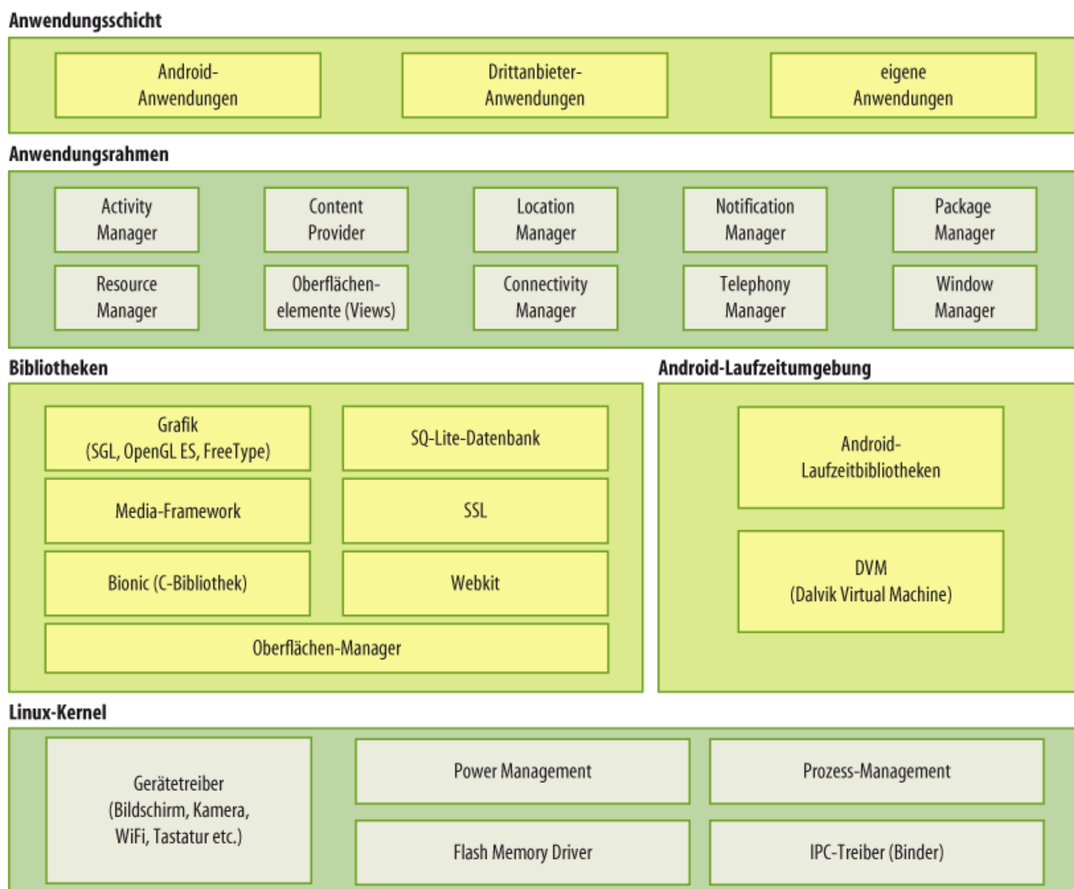

Abbildung 2.2 Android Systemarchitektur, [Becker2010]

2.3.2.2. Bibliotheken und Android-Laufzeitumgebung

Das Herzstück der Android Laufzeitumgebung ist die Dalvik Virtual Machine (DVM). Die DVM ist eine besonders kleine, extra für Android entwickelte Java Virtual Machine (JVM). Aufgrund der geringen Größe kann unter Android jede gestartete App in einer eigenen DVM mit nur wenig zusätzlichen Ressourcen laufen. Da sich die jeweiligen Apps dadurch niemals einen gemeinsamen Speicher teilen, führt dies dazu, dass bei einem Absturz eines Prozesses in der DVM maximal eine App beendet wird.

Eine Erweiterung der Android-Laufzeitumgebung bilden die Standardbibliotheken, die in C/C++ geschrieben sind. Durch sie werden die erforderlichen Funktionalitäten wie Datenbankzugriff, 3D-Grafikbibliotheken, Webzugriff, Multimedia-Verwaltung oder Oberflächenmanagement bereitgestellt. Die Standardbibliotheken sind fester Bestandteil des Systems und können von Anwendungsentwicklern nicht geändert werden.

2.3.2.3. Anwendungsrahmen

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

2.3.2.4. Anwendungsschicht

Sie beinhaltet alle bereits in Android standardmäßig enthaltenen Apps wie Contacts oder den Dialer, aber auch die aus dem Google Play Store heruntergeladenen oder eigenständig entwickelten Apps.

2.3.3 Komponenten

Android ist eine moderne Plattform für komponentenbasierte Anwendungen. Das ergibt die Möglichkeit, bereits bestehende Komponenten jederzeit wieder zu verwenden oder zu erweitern. Die unter Android zur Verfügung gestellten Anwendungen Contacts oder Dialer können somit von anderen Anwendungen über Intents (vgl. Abschnitt 2.3.3.5 Intents) mitverwendet werden. Im nächsten Abschnitt werden die Komponenten von Android näher erläutert.

2.3.3.1. Content Provider

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

2.3.3.2. Status Notification

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

2.3.3.3. Service

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

2.3.3.4. Activity

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

- **onCreate()** wird aufgerufen, sobald die Activity das erste Mal vom Betriebssystem in den Cache geladen wird. Solange eine Activity im Speicher des Betriebssystems vorhanden ist, wird diese Methode nicht mehr ausgeführt. Sie wird benutzt, um eine initiale Konfiguration vorzunehmen, beispielsweise Views zu laden beziehungsweise zu erstellen oder einmalig zum Programmstart bestimmte Methoden auszuführen. Zusätzlich kann in der Methode ein Bundle Object übergeben werden, in welchem man den Zustand einer Activity speichern kann, um ihn beim erneuten Erstellen der Activity wiederherzustellen.

- **onStart()** wird ausgeführt, kurz bevor die Activity sichtbar wird, egal ob sie neu erzeugt oder zurück in den Vordergrund geholt wurde.

- **onResume()** aktiviert die Activity und ermöglicht dadurch die Interaktionen mit dem Benutzer.

- **onRestart()** wird immer dann ausgeführt, wenn eine gestoppte Activity erneut aufgerufen wird. Eine Activity gilt als gestoppt, wenn sie für den Anwender nicht mehr sichtbar ist, z.B. durch das Starten einer anderen Activity in den Hintergrund verschoben wurde.

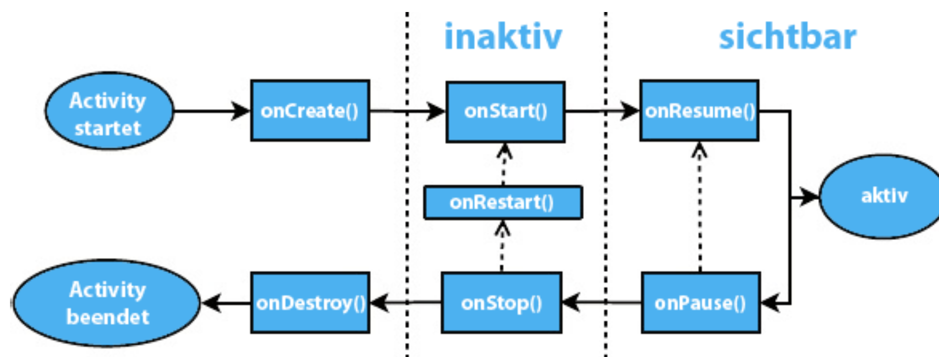

Abbildung 2.3 Lebenszyklus einer Activity, [Becker2010]

2.3.3.5. Intent

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

2.3.3.6. Broadcast Receiver

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

2.3.4 Android Manifest

Die Manifest Datei befindet sich in dem Root Ordner eines jeden Android Projektes und enthält wichtige Informationen über die App. Sie verfügt über Informationen für das Android System, die bereits vor der Ausführung der App bereit stehen müssen. Folgende Informationen sind in der Datei festgehalten [Google2013]:

- der Package name der App, welcher als einzigartiger Bezeichner für diese App dient
- eine Beschreibung der in der App verwendeten Komponenten (vgl. Abschnitt 2.3.3 Komponenten), einschließlich der Information, in welchen Prozessen diese ausgeführt werden
- die von der App beanspruchten Zugriffsrechte, beispielsweise für das Internet, das GPS oder die Kontaktinformationen
- die kleinstmögliche Android-API unter dem die App installiert werden kann
- eine Kennzeichnung der Activity, die bei Start der App aufgerufen werden soll
- eine Liste der benötigten Libraries, ohne die ein Start der App nicht möglich ist

2.4 Android Test Framework

Die Anwendungsentwicklung für Android baut auf der Programmiersprache Java auf und nutzt deshalb auch deren Testframework JUnit als Grundlage für das eigene Testframework, welches im Android SDK enthalten ist. Abbildung 2.4 gibt einen Überblick über dessen Architektur [Google2013a].

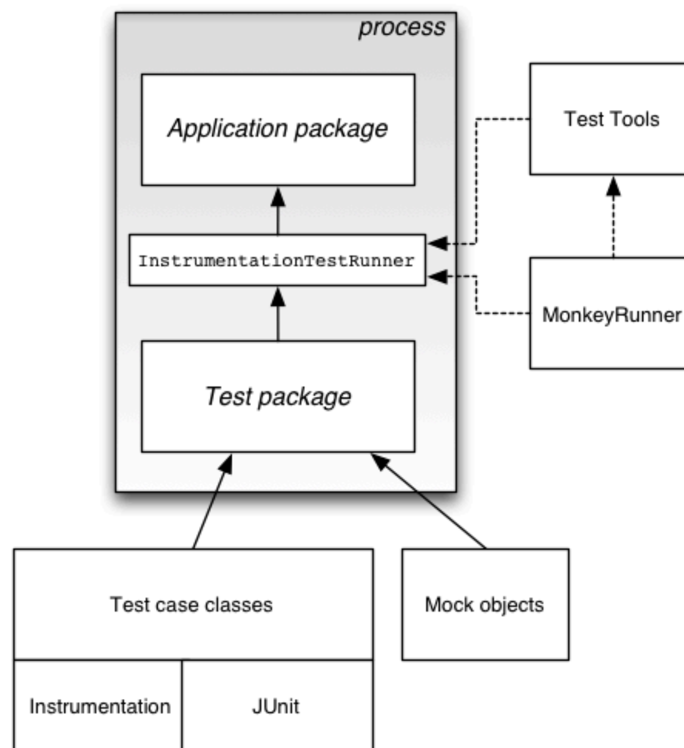

Abbildung 2.4 Android Test Framework, [Google2013a]

Die **Test Tools** beinhalten die Werkzeuge des Android SDK, die zum Bauen des Android Apllication Projects und der Tests benötigt werden. Sie sind entweder in der Entwicklungsumgebung Eclipse (siehe Kapitel 2.5.2) enthalten oder stehen anderen Entwicklungsumgebungen per Kommandozeile zur Verfügung.

2.4.1 Die Test API

Die Android Test API basiert auf der JUnit API und erweitert diese durch das Instrumentation Framework und Android spezifische Testklassen [Google2013a].

JUnit

Für Klassen die keine Methoden der Android API aufrufen, können Unit Tests auch durch die JUnit TestCase Klasse umgesetzt werden. TestCase ist außerdem die Basisklasse für AndroidTestcase mit der Android abhängige Objekte getestet werden können. AndroidTestcase erweitert JUnit durch Android spezifische Setup-, Teardown- und Hilfsmethoden. [Google2013a]

Instrumentation

Die Android Instrumentation ist ein Sammlung von Kontrollmethoden oder „Hooks“ im Android System. Diese Hooks kontrollieren eine Android Komponente unabhängig von ihrem normalen Lebenszyklus. Über die Instrumentation können die Callback Methoden des Activity Lebenszyklus (siehe Kapitel 2.3.5, Abbildung 2.3) innerhalb des Test-Codes aufgerufen werden. Hierdurch wird es möglich, Schritt für Schritt durch den Lebenszyklus einer Activity, ähnlich eines Debug Prozesse, zu gehen [Google2013a].

Eine der Schlüsselmethoden ist getActivity() als Teil der Instrumentation API. Die getestete Activity wird bis zu diesem Methodenaufruf nicht gestartet.

Die folgende Abbildung 2.5 zeigt das Klassendiagramm der Testklassen im Android Test Framework. Im Rahmen des Kapitels 3 stehen die Klassen ActivityUnitTestcase<Activity> und AcivityInstrumentationTestCase2<Activity> im Fokus.

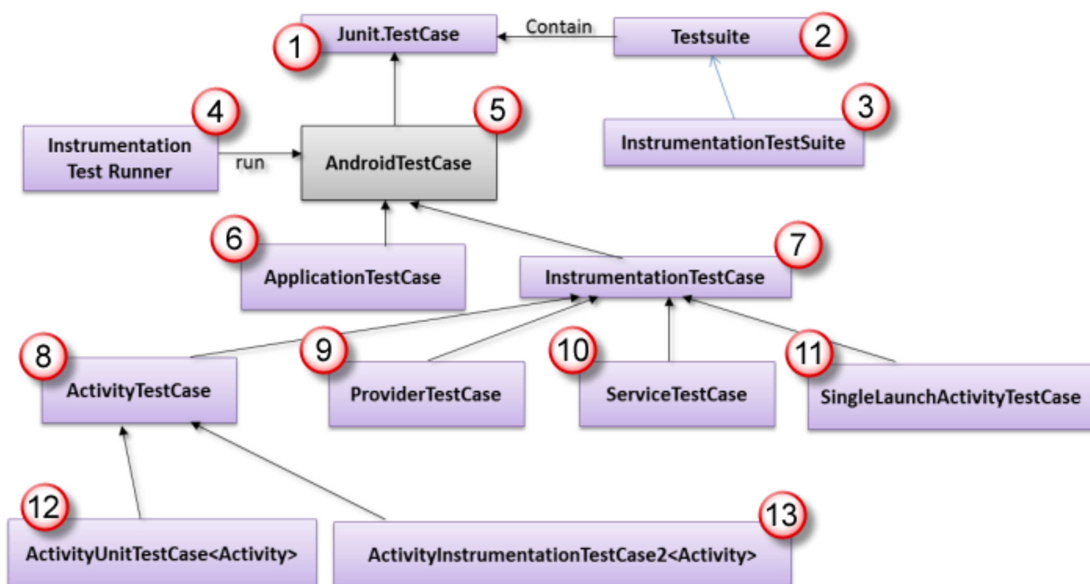

Abbildung 2.5 Klassendiagramm Android Test Framework, [Guru99.com2013]

Im folgenden werden die einzelnen Klassen kurz erläutert [Guru99.com2013].

1. **TestCase** beinhaltet JUnit Methoden um JUnit Tests durchzuführen.
2. **TestSuite** kann benutzt werden, um eine definierte Menge von Tests zu starten.
3. **InstrumentationTestSuite** ist eine TestSuite, die vor der Ausführung Instrumentation in den InstrumentationTestCase injiziert.
4. **InstrumentationTestRunner** ist der Testtreiber, um Testfälle auf der Zielapplikation auszuführen.
5. **AndroidTestCase** erweitert JUnit TestCase. Er beinhaltet Methoden, um Zugang zu Ressourcen wie z.B. dem Activity Kontext zu erhalten.
6. **ApplicationTestCase** überprüft Application Klassen in einer kontrollierten Umgebung.
7. **InstrumentationTestCase** überprüft ein besonderes Feature oder Verhalten der Zielapplikation, z.B. die Ausgaben der Benutzeroberfläche.
8. **ActivityTestCase** ist eine Basisklasse, die das Testen von Activities unterstützt.
9. **ProviderTestCase** ist eine Klasse zum Testen eines einzelnen ContentProvider.
10. **ServiceTestCase** wird benutzt, um Service Klassen in einer Testumgebung zu testen. Er unterstützt außerdem den Lebenszyklus eines Service.
11. **SingeLauchActivityTestCase** wird benutzt, um eine einzelne Activity mit einem InstrumentationTestcase zu testen.
12. **ActivityUnitTestCase <Activity>** wird benutzt, um eine einzelne isolierte Activity zu testen.
13. **ActivityInstrumentationTestCase2<Activity>** erweitert die JUnit TestCase Klasse und stellt eine Verbindung zur Zielapplikation durch die Instrumentation her. Mit Hilfe dieser Klasse können die Komponenten der GUI einer App angesprochen und UI Events (Tastatureingaben oder Touch Events) an die UI gesendet werden.

2.4.2 Monkey und Monkeyrunner

Das Android SDK stellt außerdem zwei Tools für die funktionale Testebene zur Verfügung.

Monkey ist ein Tool, das die automatische Ausführung von Stresstests auf der Androidoberfläche vereinfacht. Monkey simuliert randomisierte Benutzereingaben, wie z.B. Klicks, Gesten oder Berührungen. Außerdem können mit Monkey Systemnachrichten erzeugt werden. Monkey wird über die Android Debug bridge (adb) gestartet [Google2013b].

Monkeyrunner ist eine API und Ausführungsumgebung für in Python geschriebene Testprogramme. Die API beinhaltet Funktionen, um sich mit einem Android Device zu verbinden, Programmpakete zu installieren und zu deinstallieren, Screenshots zu machen und Testpakete gegen eine App auszuführen [Google2013c].

2.5 Projekt Setup / Entwicklungswerkzeuge

Dieses Kapitel gibt einen Überblick über die Werkzeug, die im Rahmen der beispielhaften Implementierung einer Android App genutzt werden.

2.5.1 Development Kits

- JAVA 7 JDK
- Android SDK in der Version 23.0.2

2.5.2 Entwicklungsumgebung: Eclipse

Als Entwicklungsumgebung diente Eclipse in der Version 4.2 Juno. Die Entwicklungsumgebung steht unter der Eclipse Public License [Eclipse2014] frei zur Verfügung.

2.5.3 Build-Management-Tool: Ant

Apache Ant, im folgenden nur Ant genannt, ist ein Projekt der Apache Software Foundation und steht für „Another Neat Tool". Als Build-Management-Tool dient es der automatischen Generierung von ausführbaren Computerprogrammen aus Quelltext [Matzke2005].

Die Ablaufsteuerung für den Build Prozess eines Projektes wird in einer XML-Datei namens build.xml beschrieben und kann von Ant ausgeführt werden. Ein solches Buildfile folgt einer festen Syntax. Einige XML Tags sollen hier kurz erläutert werden.

- Target: ein Container für ein oder mehrere zusammengehörige Tasks
- Task: kommt innerhalb eines Targets vor, erledigt eine bestimmte eindeutige Aufgabe und ist vergleichbar zu einer Anweisung in einer Methode
- Properties: definieren die von Ant-Tasks benötigte JAR-PATH/Literale/Ordner
- Classpath: spezifiziert Paths von JARs/Verzeichnisse zur Ausführung von Tasks

2.5.4 Unit-Test-Framework: Robolectric

Robolectric [Robolectric2013] wurde als Unit-Test-Framework speziell für Android Applikationen entwickelt. Der Vorteil gegenüber JUnit bzw. dem Android Test Framework [siehe Kapitel 2.4] ist die Möglichkeit, die Tests in der JVM ablaufen zu lassen. Tests können somit ohne Emulator oder externes Gerät durchgeführt werden. Dies wird mit sogenannten Shadow Objects realisiert.

- **Shadow Objects**: Robolectric fängt das Laden von Android-Klassen ab und überschreibt den Methoden-Rumpf mittels Javassist, einem Unterprojekt von Jboss, das Java-Byte-Code-Manipulation während des Ladens von Java Klassen ermöglicht. Anschließend werden sogenannte Shadow Objects an neue Android Objekte gebunden. Die veränderten Android Objekte leiten dann alle Methodenaufrufe als Proxy an die Shadow Objects [Greb2012].

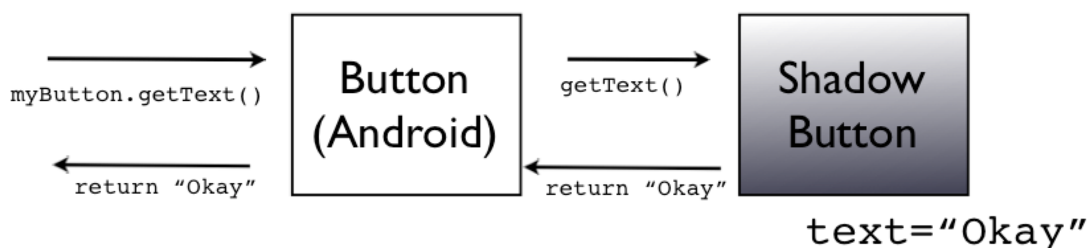

Abbildung 2.6 Shadow Objects unter Robolectric, [Greb2012]

- **Laden von Views und Ressourcen**: Robolectric parsiert die Layout Files des Android Projektes und baut daraus einen neuen View Object Tree, bestehend aus Android View Objekten und ihren Shadows.

Durch die Ausführung in der JVM entfällt das sogenannte Dexing, also die Konvertierung von Java kompilierten JAR Dateien in auf der Dalvik VM ausführbare Dateien. Ebenso entfällt die Installation der zu testenden App und des Testprojektes auf dem Emulator bzw. Android Device. Da auf einen Emulator verzichtet werden kann, entfällt daher auch die Zeitdauer beim Start eines solchen. Diese Faktoren sollen für einen erheblich verminderten Zeitaufwand bei der Ausführung von Robolectric Tests, gegenüber Tests, die auf dem Android Test Framework basieren, führen.

2.5.5 UI-Test-Framework: Robotium

Robotium [Robotium2012] ist ein Testautomatisierungsframework für Android, das es ermöglicht, Oberflächentests aus Nutzersicht zu schreiben. Das Framework ist unter Apache 2.0 Lizenz veröffentlicht und unterstützt sowohl native als auch hybride Apps (ab Version 4). Im Rahmen dieser Arbeit wurde Robotium in der Version 5.0.1 verwendet.

2.5.6 UI-Test-Framework: Espresso

Einen verlässlichen Test für Android zu schreiben, sollte nicht mehr Zeit in Anspruch nehmen, als sich am Kaffeeautomaten einen Espresso zu ziehen. Mit diesen Worten beschreibt einer der Google Mitarbeiter und Espresso Entwickler V. Zakharov die Motivation hinter der Entstehung des Test Frameworks Espresso [GoogleTechTalk2013].

Espresso stellt drei Paradigmen in den Fokus: „Einfachheit“, „Zuverlässigkeit“ und „Haltbarkeit/Beständigkeit“ [Google2013d].

- **Einfachheit** wird in Form von Methoden angeboten, die dem Verhalten realer Nutzer nachgebildet sind: ein UI-Element auswählen, Aktion auf dem Element ausführen und abschließend das erwartete Ergebnis prüfen.

- **Zuverlässigkeit** versucht Espresso dadurch zu erreichen, dass es bewusst für jede Aktion innerhalb eines Tests so lange wartet, bis die App im Zustand „idle“ ist und somit der UI-Thread gefahrlos und exklusiv durch den Test genutzt werden kann.

- **Haltbarkeit/Beständigkeit** soll erreicht werden, indem Espresso Methoden zur Verfügung stellt, die es ermöglichen UI-Elemente über ihre Ressource-ID anzusprechen.

Für die Umsetzung von Tests mittels Espresso wurde das Framework in der Version 1.1 verwendet.

2.5.7 Versionskontrollsystem: Git / GitHub

Das verteilte Versionskontrollsystem Git [Git2014] bietet die Möglichkeit, auch ohne Verbindung zu einem zentralen Server zu arbeiten. Es ist daher keine ständige Netzwerkverbindung erforderlich. Aus Gründen der Datensicherheit wurde ein Remote-Repository auf GitHub angelegt. Für den Rahmen dieser Abschlussarbeit wird ein kostenloses öffentliches Repository als völlig ausreichend erachtet.

2.5.8 Integrationsserver: Jenkins

Aufgrund seiner großen Verbreitung wurde Jenkins [Jenkins2014] als CI-Server für dieses Projekt gewählt. Jenkins Unterstützt neben den gängigen Build-Tools wie Ant, Maven oder Gradle auch die hier verwendete Versionsverwaltung Git.

2.5.9 Statische Code-Analyse: Android Lint

Android Lint [Google2013e] ist ein Tool zur statischen Code-Analyse. Es durchsucht den Sourcecode eines Android Projektes nach potenziellen Fehlerquellen und Optimierungsmöglichkeiten und listet diese nach den Kategorien Korrektheit, Sicherheit, Leistung, Benutzerfreundlichkeit und Zugänglichkeit auf. Android Lint ist in der Lage, folgende Fehler und Optimierungsmöglichkeiten zu erkennen:

- fehlende oder unbenutzte Übersetzungen
- Performance Probleme des Layouts
- unbenutzte Ressourcen
- inkonsistente Größe von Arrays
- hartcodierte Strings
- Fehler im Manifest

2.5.10 Android-Plattform

Für das Demonstrationsbeispiel wurde als Zielumgebung API-Level 18, Android 4.3 gewählt. Die Mindestanforderung beträgt API-Level 16 (Android 4.1).

3 Testgetriebene Entwicklung mit Android

3.1 Anforderungen an das Demonstrationsbeispiel

Diese App dient in erster Linie als Demonstrationsbeispiel für die Umsetzung testgetriebener Entwicklung im Rahmen dieser Arbeit. Des Weiteren soll sie möglichst viele, für eine mobile App typischen Elemente besitzen, um die Verwendung von Frameworks für Testautomatisierung zu erproben. Durch die Verwendung von WebViews und Interaktion zwischen HTML Inhalt und nativen Elementen der App sollen die Grundzüge einer Hybrid App abgebildet werden.

Folgende Android Standardelemente sollen in der App vorhanden sein:

Android Element	**Beschreibung**
Button	Der Standard Button ist hier in seiner Ausprägung ein Taster, der beim Klick eine Aktion auslöst.
TextView	ein per Default durch den Benutzer nicht editierbares Textfeld
EditText	ein per Default durch den Benutzer editierbares Textfeld
Spinner	eine DropDown Liste
AlertBox	Dialog Box zum Anzeigen einer Warnung
Toast	Textnachricht, die auf der Oberfläche angezeigt wird und nach einer bestimmten Zeit wieder verschwindet
CheckBox	ein Markierungsfeld, das zwei Zustände annehmen kann: nichtmarkiert oder markiert
WebView	View zum Anzeigen von Webseiten

Tabelle 3.1 Android UI Standardelemente

Die Activities der App sollen folgende Hierarchie besitzen:

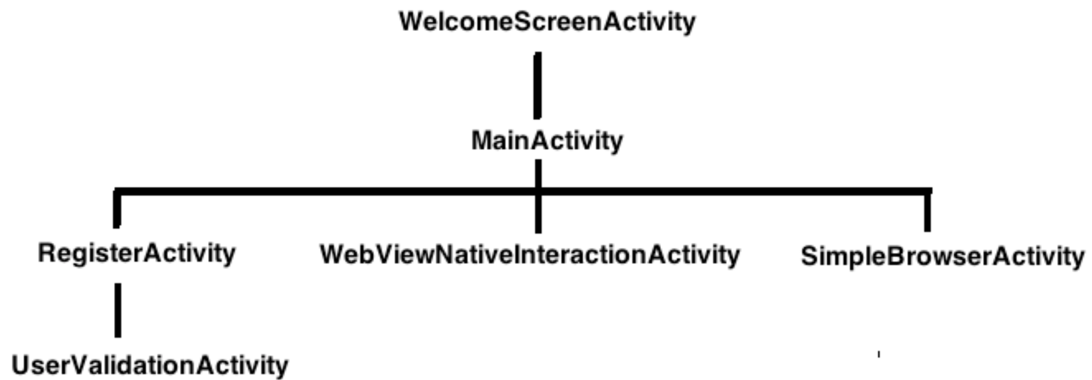

Abbildung 3.1 Hierarchie der Activities

Beim Start der App ist die WelcomeScreenActivity zu sehen. Sie wird in Richtung der MainActivity verlassen, die den Hauptbildschirm darstellt. Über die MainActivity kann man zur RegisterActivity, WebViewNativeInteractionActivity und der SimpleBrowserActivity gelangen. Die UserValidationActivity ist ein Child der RegisterActivity.

3.1.1 Design der Activities

Als Teil des Designprozesses der App wurden mit Hilfe des Online Dienstes MockFlow [MockFlow2014] und einer Bildbearbeitungssoftware WireFrames erstellt. Diese sollen dokumentieren, wie die Benutzeroberfläche der App aussehen und welche Funktionalität sie besitzen soll.

WelcomeScreenActivity:

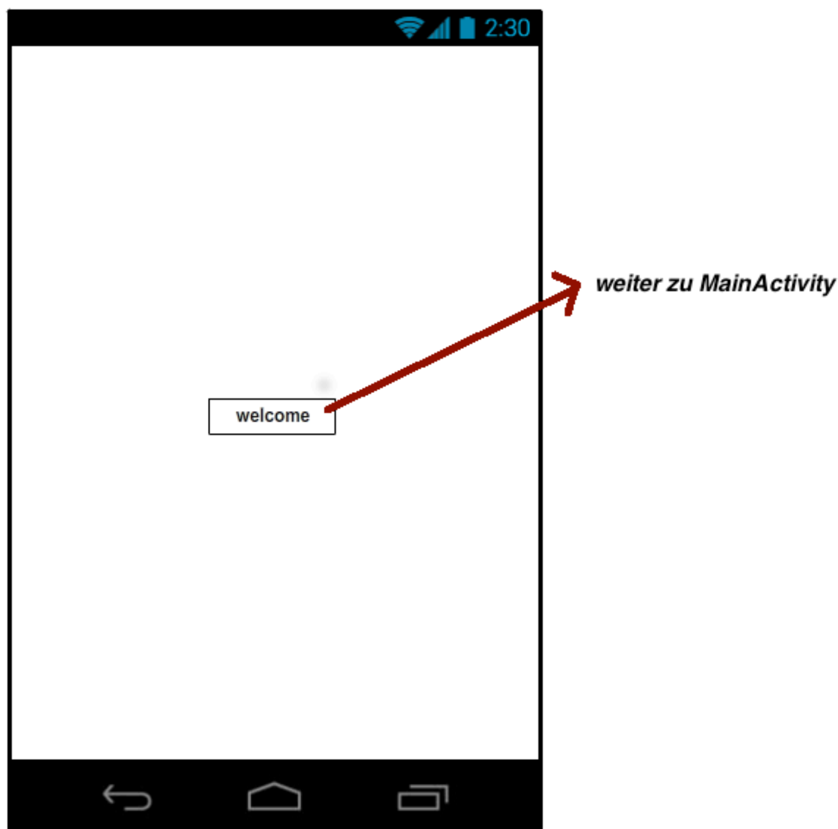

Abbildung 3.2 Design WelcomeScreenActivity

Die WelcomeScreenActivity besitzt als einziges Element einen Button, über den man zur MainActivity gelangt. Anhand dieser Activity wird die Verwendung von Unit-Tests unter Android demonstriert.

Der Hauptbildschirm, die MainActivity:

Abbildung 3.3 Design MainActivity

Zusätzlich zu den Buttons, über die man zu anderen Activities gelangt, besitzt die MainActivity ein Textfeld für Benutzereingaben, dessen Inhalt nach Klick auf den Button an anderer Stelle ausgegeben wird. Weitere Buttons sollen die Anzeige einer Toast Message auslösen oder eine AlertBox anzeigen.

RegisterActivity und UserValidationActivity:

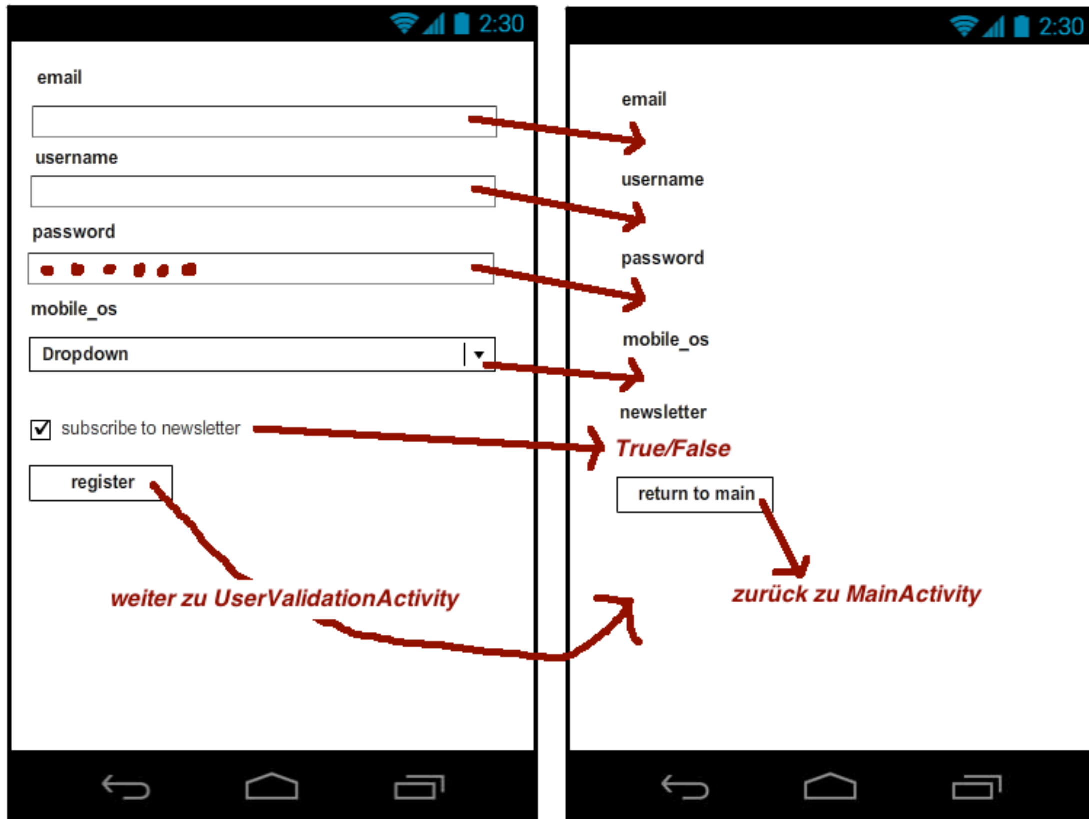

Abbildung 3.4 Design RegisterActivity und UserValidationActivity

Die RegisterActivity soll ein gängiges Registrierungsformular abbilden. Die Eingaben des Benutzers sollen nach Klick auf den „register“ Button in der UserValidationActivity angezeigt werden. Die UserValidationActivity dient der Demonstration des Aufrufs einer Activity bei gleichzeitiger Übergabe von Daten.

WebViewNativeInteractionActivity:

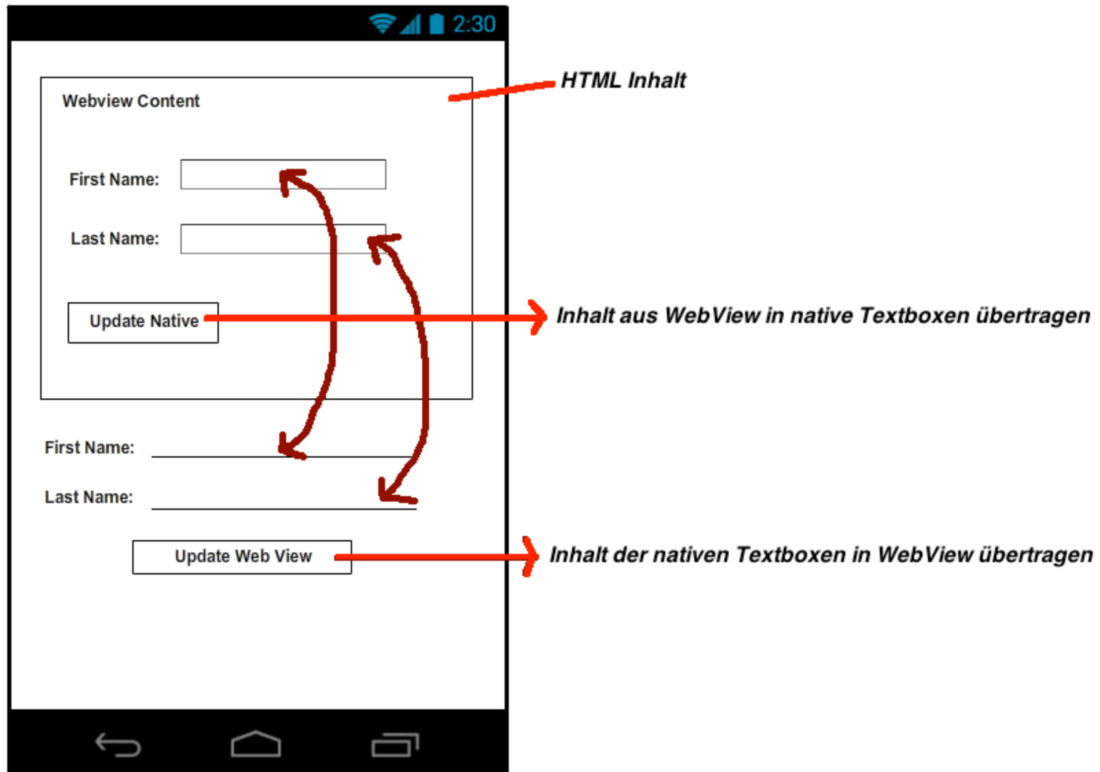

Abbildung 3.5 Design WebViewNativeInteractionActivity

Diese Activity soll Funktionen einer Hybrid App besitzen. Der in einem WebView geladene HTML Inhalt soll mit den nativen Elementen der App kommunizieren. Eingaben im nativen Teil sollen in den WebView kopiert werden und umgekehrt.

SimpleBrowserActivity:

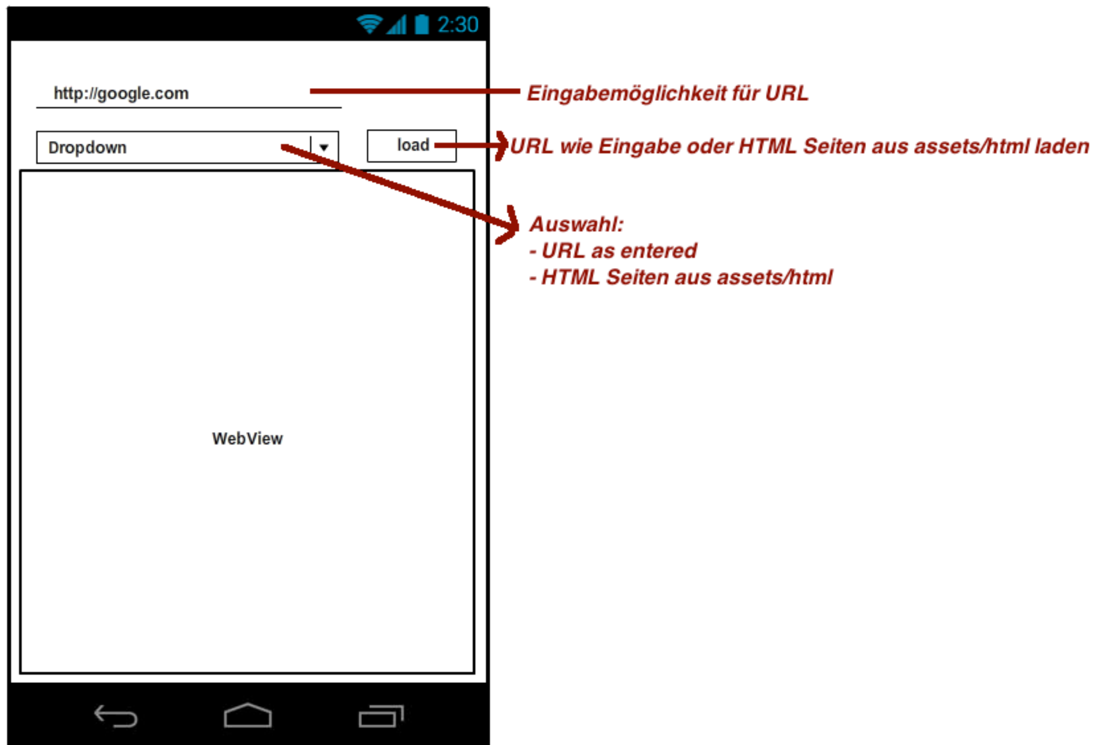

Abbildung 3.6 Design SimpleBrowserActivity

Diese Activity soll eine Implementierung eines einfachen Webbrowsers innerhalb der nativen App darstellen. Sie soll die Möglichkeit bieten, entweder eine Webseite über eine durch den Benutzer eingegebene URL aufzurufen und im WebView anzuzeigen oder statische HTML Seiten anzuzeigen, die vom Entwickler in der App hinterlegt wurden.

3.1.2 Akzeptanzkriterien

Diese Anforderungen lassen sich als Akzeptanzkriterien formulieren. Die Umsetzung der Anforderungen wird im Rahmen dieser Arbeit mittels der WelcomeScreenActivity, der RegisterActivity und UserValidationActivity sowie der WebViewNativeInteractionActivity demonstriert.

WelcomeScreen

- Nach dem Start der App wird ein Willkommensbildschirm angezeigt.
- Durch Klick auf einen Button gelangt man zur MainActivity.

Registrierung

- Es können Daten für E-Mail-Adresse, Benutzername und Passwort eingegeben werden.
- Über eine Liste kann ein mobiles Betriebssystem ausgewählt werden.
- Durch Klick auf einen Button wird die UserValidationActivity angezeigt.
- Sämtliche Eingaben aus der RegisterActivity werden auf der UserValidationActivity angezeigt.
- Über einen Button gelangt man zurück zur MainActivity.

Interaktion WebView Nativ

- Beim Klick auf den Button im WebView werden die Einträge der WebView-Textboxen in die nativen Textboxen übertragen.
- Beim Klick auf den nativen Button werden die Einträge aus den nativen Textboxen in die WebView-Textboxen übertragen.

Aus den Akzeptanzkriterien können nun die Akzeptanztests entwickelt werden.

3.1.3 Akzeptanztests

A1_WelcomeScreen

- über Klick auf den Button zur MainActivity gelangen

A2_Registrierung_Daten

- Daten eingeben
- über den Button zur UserValidationActivity gelangen
- Daten überprüfen

A3_Interaktion_WebView_Nativ

- Eingaben im WebView machen
- Klick auf Update-Button im WebView
- Übergabe der Daten in nativen Textboxen überprüfen
- neue Eingaben in nativen Textboxen machen
- Klick auf nativen Update-Button
- Übergabe der Daten im WebView überprüfen

3.2 Umsetzung von Test First

In diesem Kapitel erfolgt die praktische Umsetzung der testgetriebenen Entwicklung anhand einer Android App. Dabei wird der Grundsatz des Test First verfolgt. Zuerst erfolgt die Implementierung des Testfalls und anschließend die Implementierung der Funktionalität in Form einer oder mehrerer Komponenten. Bevor allerdings der erste Test implementiert wird, erfolgt zunächst das Anlegen des Projektes für die App in der Entwicklungsumgebung und die Einrichtung der Versionskontrolle.

3.2.1 Android Application Project anlegen

In Eclipse wird ein neues Android Application Project unter dem Namen TDDAndroid angelegt. Als Mindestanforderung für das Android SDK wird API-Level 16 (Android 4.1) und als Zielversion API-Level 18 (Android 4.3) gewählt. Der App wird zu Anfang eine noch leere Activity namens MainActivity hinzugefügt.

3.2.2 Projekt unter Versionskontrolle stellen

Nach Installation von Git müssen zuerst die globalen Variablen für den Namen und die E-Mail-Adresse des Nutzers gesetzt werden. Dies geschieht per Kommandozeile:

```
git config --global user.name „Vorname Nachname‟

git config –-global user.email „E-Mail-Adresse‟
```

Um für unser Projekt ein lokales Repository anzulegen, bedarf es nur eines Befehls in der Kommandozeile. Hierfür wechselt man in das Projektverzeichnis und benutzt den folgenden Befehl:

```
git init
```

Daraufhin wird das lokale Git Repository unter .git angelegt. Vor dem ersten Commit werden nun alle Dateien „gestaged“ oder, anders ausgedrückt, für den folgenden Commit markiert.

```
git add .
```

Jetzt kann der erste Initiale Commit erfolgen.

```
git commit –m ‚First commit’
```

Um als Team an dem Projekt arbeiten zu können, benötigt man zusätzlich ein Remote Repository, in das alle Teammitglieder ihre Änderungen einchecken können. Hierzu muss zuerst ein Account auf GitHub eingerichtet werden. Anschließend kann ein neues öffentliches Repository erstellt werden. Dem GitHub Repository wird derselbe Name wie dem Android Application Project gegeben und es wird eine kurze Beschreibung hinzugefügt. Abschließend kopiert man die URL des GitHub Repository und teilt sie Git mit.

```
git remote add origin <remote repository URL>
```

Jetzt kann der Inhalt des lokalen Repository auf GitHub gepushed werden.

```
git push origin master
```

Unser Projekt ist nun erfolgreich auf GitHub gehosted.

Abbildung 3.7 TDDAndroid Projekt auf GitHub

Bewertung

Ein Projekt mit Git unter Versionskontrolle zu stellen, gestaltet sich mit wenig Aufwand.

3.2.3 Unit-Test mit dem Android Test Framework

Die Akzeptanzkriterien der WelcomScreenActivity sollen nun mittels Test First und dem Android Test Framework als Unit-Test umgesetzt werden.

3.2.3.1. Anlegen eines Testprojektes

Es wird ein Android Test Project angelegt. Dabei ist das zu testende Android Application Project auszuwählen.

Als Testklasse wird eine neue Klasse vom Typ JUnit Test Case erzeugt. Die Superclass muss aus dem Ordner „android.test“ gewählt werden, da Android nicht mit den ursprünglichen JUnit Tests zurechtkommt.

3.2.3.2. Umsetzung von Unit-Tests mit dem Android Test Framework

In diesem Beispiel soll das Verhalten einer Activity getestet werden. Als Superclass wurde daher „ActivityUnitTestCase“ gewählt.

```
public class WelcomeScreenActivityTest extends
    ActivityUnitTestCase<WelcomeScreenActivity> {
```

Abbildung 3.8 Klassendefinition JUnit

Innerhalb der Methode setUp() wird die zu testende Activity erstellt und gestartet.

```
protected void setUp() throws Exception {
    super.setUp();
    // neuen Intent aus dem Context der Apllication under Test instanzieren
    Intent intent = new Intent(getInstrumentation().getTargetContext(),
            WelcomeScreenActivity.class);
    // ruft die onCreate() Methode der zu testenden Activity auf
    startActivity(intent, null, null);
    // Referenz auf die gestartete Activity holen
    welcomeScreenActivity = getActivity();
}
```

Abbildung 3.9 Setup() Methode ActivityUnitTestCase

Jetzt kann die Testmethode geschrieben werden. Aufgrund der Tatsache, dass an dieser Stelle mit dem JUnit3 Framework gearbeitet wird, muss die Testmethode mit dem Wort „test“ beginnen, um als solche erkannt zu werden. Innerhalb der Methode testShouldStartAnotherActivity() wird zuerst ein Button erzeugt. Nach Klick auf den Button wird überprüft, ob eine Activity gestartet wurde und ob es bei ihr um die erwartete MainActivity handelt (siehe Abbildung 3.10).

```
public void testShouldStartAnotherActivity() {
    // zu testender Button der WelcomeScreenActivity
    Button btnLaunchMain = (Button) welcomeScreenActivity
            .findViewById(R.id.btnLaunchMain);

    // Klick auf Button
    btnLaunchMain.performClick();
    // Zuweisung der gerade gestarteten Activity
    Intent triggeredIntent = getStartedActivityIntent();
    // Überprüfung das ausgelöster Intent not Null ist
    assertNotNull("ERROR: Intent was null", triggeredIntent);
    // Überprüfung das es sich bei gestarteter Activity um MainActivity handelt
    assertEquals("Error: Wrong Activity", MainActivity.class.getName(),
            triggeredIntent.getComponent().getClassName());
}
```

Abbildung 3.10 Testmethode ActivityUnitTestCase

3.2.3.3. Ausführung des Testfalls

Der Test wird nun ausgeführt. Auch wenn es offensichtlich erscheint, dass der Test zu diesem Zeitpunkt kein positives Ergebnis liefern kann, schon allein aufgrund der Tatsache, dass an dieser Stelle Objekte referenziert werden, die noch nicht existieren, so sollte man nicht darauf verzichten. Man kann somit ausschließen, dass der Test aufgrund einer falschen Implementierung immer positiv ausfällt. Ein solcher Test hätte dann keinen Nutzen bzw. Erkenntnisgewinn.

3.2.3.4. Implementierung der Android Komponente

Es wird eine leere Activity ohne Action Bar erzeugt und per Layout Manager ein Button hinzugefügt. Anschließend wird, um den Übergang zur nächsten Activity zu realisieren, die Methode startMainActivity() erstellt. Zuletzt wird dem Button ein OnClickListener zugewiesen, der diese Methode aufruft (siehe Abbildung 3.11).

```
public class WelcomeScreenActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_welcome_screen);

        Button btnLaunchMain = (Button) findViewById(R.id.btnLaunchMain);

        btnLaunchMain.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                startMainAvtivity();
            }
        });
    }

    public void startMainAvtivity(){
        Intent nextScreen = new Intent(this, MainActivity.class);
        startActivity(nextScreen);
    }
}
```

Abbildung 3.11 Implementierung WelcomeScreenActivity

Die folgende Abbildung 3.12 zeigt die Oberfläche der implementierten WelcomeScreenActivity.

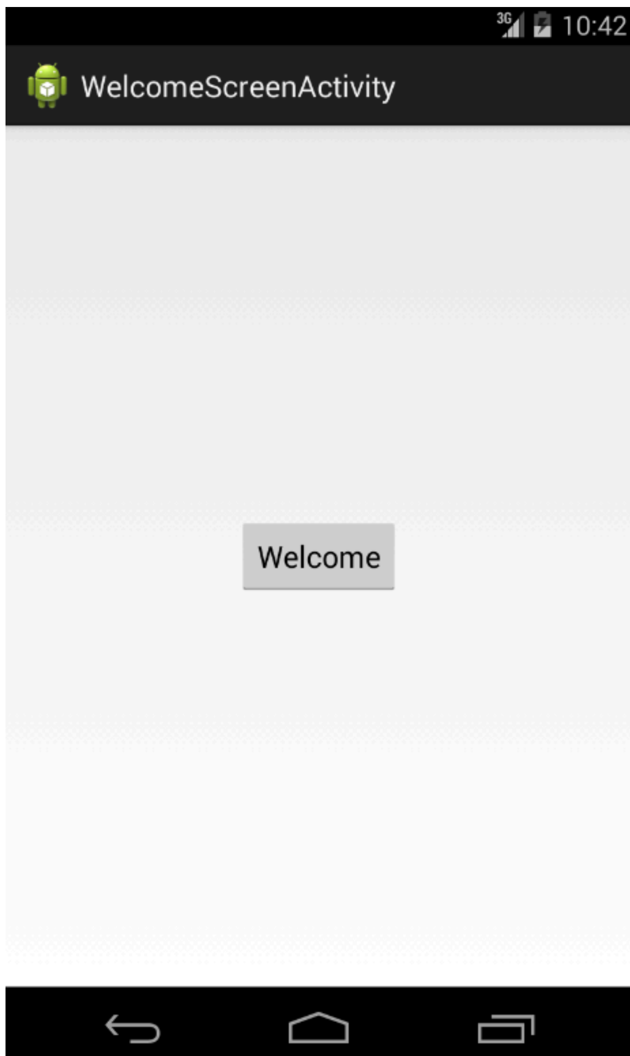

Abbildung 3.12 WelcomeScreenActivity

3.2.3.5. Erneutes Ausführen des Testfalls

Nach erfolgter Implementierung der WelcomeScreenActivity stellen wir zuerst fest, ob innerhalb unseres Testprojektes keine Fehler mehr angezeigt werden und der Test Code, wie erwartet, erfolgreich kompiliert.

Um den Testlauf zu starten wird *RunAs -> Android JUnit Test* gewählt.

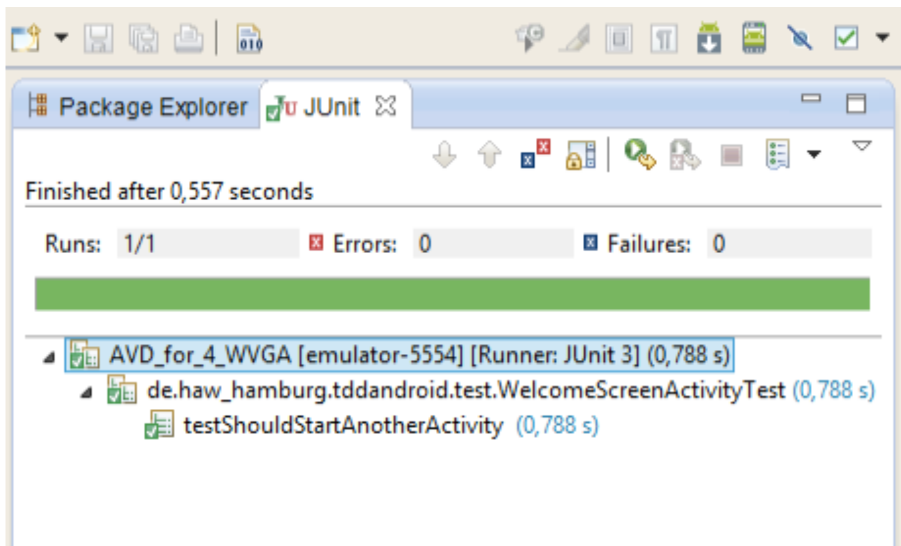

Abbildung 3.13 JUnit Testergebnis

Der Test ist nun erfolgreich, wie Abbildung 3.13 zeigt.

3.2.3.6. Integration in den Jenkins

Um CI zu realisieren, gilt es nun, das Android Application Project und das Testprojekt in den Jenkins CI-Server zu integrieren. Nach erfolgreicher Installation von Jenkins und Ant, ist zunächst ein Ant Build File für das Android Application Project zu generieren.

```
android update project -p TDDAndroid
```

Anschließend wird ebenfalls ein Ant Build File für das Testprojekt generiert.

```
android update test-project -m ../TDDAndroid -p TDDAndroidJunitTests
```

Jetzt lässt sich überprüfen, ob sich das Android Application Project mit Ant bauen lässt.

```
ant clean debug
```

Bevor wir das Testprojekt ebenfalls mit Ant bauen, muss sichergestellt sein, dass ein Android Device oder Emulator verbunden ist. Danach wird der folgende Befehl im Verzeichnis des Testprojektes ausgeführt:

```
ant clean debug install test
```

Das Android Application Project und das Testprojekt werden daraufhin gebaut und auf dem Android Device oder Emulator installiert. Anschließend führt Ant die Tests aus.

Nun kann ein neuer Jenkins Build Job angelegt werden. Um auf die Benutzeroberfläche von Jenkins zu gelangen, muss im Webbrowser nur die Adresse http://localhost:8080/ eingegeben werden. Um einen neuen Jenkins Build Job anzulegen, wählt man zuerst „Element anlegen“ und „Free Style-Softwareprojekt bauen“. Um die Ressourcen des

Android Application und Testprojektes automatisch aus der Versionsverwaltung zu beziehen, müssen die zugehörigen Repository URLs und die GitHub Zugangsdaten angegeben werden.

Vor der Ausführung der folgenden Build Anweisungen, kann durch Jenkins, nach Installation des Android Emulator Plug-Ins, ein Emulator gestartet werden. Hierbei stehen zwei Möglichkeiten zur Verfügung. Die erste Möglichkeit besteht darin, eine bereits existierende Instanz eines Emulators zu starten. Diese muss zuvor im AVD (Android Virtual Device) Manager eingerichtet worden sein. Die zweite Möglichkeit ist, im Jenkins die benötigten Details zur Erstellung eines Android Emulators anzugeben und bei jeder Ausführung des Jenkins Build Projektes eine neue Instanz zu erzeugen.

Um das Android Application Projekt und das Testprojekt über Jenkins zu bauen, auf dem Emulator zu installieren und die Tests auszuführen, müssen die beiden zuvor beschriebenen Ant Anweisungen als „Ant Build Schritt“ hinzugefügt werden. Dazu ist es erforderlich, den Ort der build.xml Dateien anzugeben.

Die Konfiguration des Jenkins Build Jobs ist damit abgeschlossen. Dieser kann nun über „jetzt bauen“ gestartet werden.

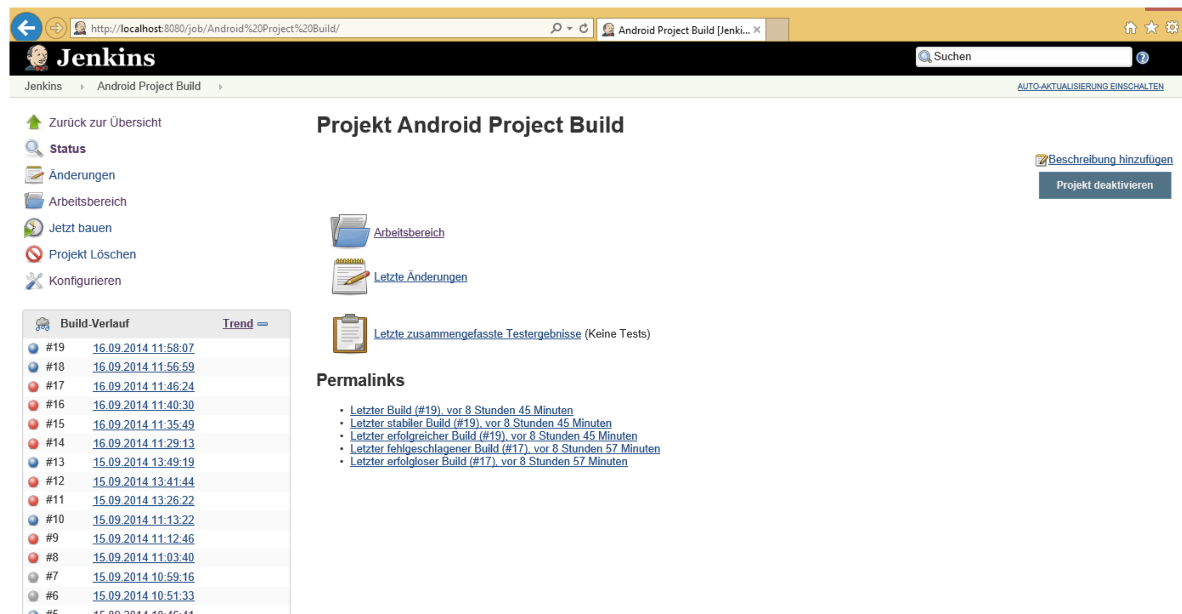

Abbildung 3.14 TDDAndroid im Jenkins

Die beschriebene Konfiguration von Jenkins ist die minimal notwendige Konfiguration. Diese kann durch weitere Anweisungen jederzeit ergänzt werden.

Bewertung

Um die Eignung von Jenkins als CI-Server zu bewerten, soll an dieser Stelle überprüft werden, inwieweit die „10 Grundsätze für eine wirkungsvolle CI" aus Kapitel 2.1.3 erfüllt sind:

1. Gemeinsame Codebasis:
Durch Verwendung einer Versionsverwaltung und eines gemeinsamen Repositories steht den Entwicklern eine gemeinsame Codebasis zur Verfügung.

- Ist erfüllt durch die Verwendung von Git und GitHub.

2. Automatisierter Build:
Das Produkt muss vollautomatisch aus seinen Grundbestandteilen übersetzt und zusammengebaut werden können.

- Ist erfüllt durch die Build Automatisierung mit Ant.

3. Selbsttestender Build:
Durch die Ausführung automatisierter Tests bei jedem Build wird das entstandene Produkt automatisch auf Korrektheit überprüft.

- Ist erfüllt, da bei jedem Build die Tests ausgeführt werden.

4. Häufige Integration: Entwickler sollen ihre Arbeit am Projekt mindestens einmal pro Tag integrieren. Es soll also mindestens einmal am Tag der aktuelle Stand in das Versionskontrollsystem eingecheckt werden.

- Dies liegt in der Verantwortung des jeweiligen Entwicklers.

5. Jede Änderung (Commit) soll einen automatischen Build auslösen: Fehler, die durch Änderungen entstehen, können so schneller aufgefunden und behoben werden, sofern die Änderungen klein sind.

- Ist noch nicht erfüllt, kann aber im Jenkins konfiguriert werden. Jenkins bietet die Möglichkeit, das Versionskontrollsystem nach Änderungen abzufragen.

6. Der Build soll schnell sein: Nach dem Einchecken einer Änderung sollte das CI-System schnell eine Rückmeldung geben können, möglichst nach nur wenigen Minuten.

- Ist zu diesem Zeitpunkt erfüllt. Der Jenkins Job benötigt zum gegenwärtigen Zeitpunkt nur 28 Sekunden.

7. Einen fehlschlagenden Build sofort beheben: Dies soll umgehend erfolgen, da CI ja genau dafür sorgen soll auf einem bekannten und stabilen Stand zu entwickeln.

- Dies liegt ebenfalls in der Verantwortung des Entwicklers.

8. Tests sollen in einem Klon der Produktionsumgebung stattfinden: Die Umgebung, in der die Tests ausgeführt werden, sollte der Produktivumgebung möglichst ähnlich sein.

- Ist durch den Einsatz eines Emulators erfüllt.

9. Die ausführbaren Programmdateien sollen leicht zugänglich sein: Alle Beteiligten sollen einfach an den aktuellen und ausführbaren Stand des Produkts kommen können, sei es zum Testen, zur Demonstration oder einfach nur um zu sehen, was sich geändert hat.

- Ist noch nicht erfüllt. Der Jenkins Job kann aber auch so erweitert werden, dass die resultierenden Installationsdateien, in diesem Fall die APK Dateien, im Jenkins archiviert werden.

10. Jeder kann sehen was passiert: Die Informationen über den Build, insbesondere über seinen Zustand und Verlauf, sollen allen Beteiligten leicht zugänglich sein.

- Ist erfüllt. Diese Informationen sind über die Weboberfläche von Jenkins einzusehen.

Die Anforderungen von CI aus Kapitel 2.1.3 sind daher weitestgehend durch den Einsatz von Jenkins erfüllt. Selbstverständlich werden nicht alle diese Anforderungen unmittelbar durch Jenkins erfüllt aber unterstützt.
Die Benutzung von Jenkins gestaltet sich intuitiv und übersichtlich. Die weiteren Konfigurationsmöglichkeiten von Jenkins bieten viel Spielraum zur Erweiterung des Build Prozesses.

3.2.4 Unit-Test mit Robolectric

Eine alternative Möglichkeit Unit-Tests umzusetzen bietet Robolectric. Im Folgenden soll der gleiche Testfall mit diesem Framework ebenfalls umgesetzt werden.

3.2.4.1. Anlegen eines Testprojektes

Der Aufwand bis zum erfolgreichen Anlegen eines Robolectric Testprojektes ist um einiges größer als mit JUnit bzw. dem Android Test Framework.

Im Stammverzeichnis des Android Application Projects wird ein Folder (nicht Source Folder) mit dem Namen „test“ angelegt. Anschließend muss im libs Folder ebenfalls ein Ordner namens „test“ angelegt werden. In diesen Ordner sind nun die Robolectric JAR Files zu kopieren.

Das Robolectric Testprojekt wird als reines Java Project angelegt. Dabei wird unter Project Layout der Punkt „Use project folder as root for sources and class files” ausgewählt.

Unter „Add external JARS“ muss außerdem ein entsprechendes android.jar hinzugefügt werden. Dieses sollte den gleichen API Level besitzen wie das Android Application Project.

Unter „Add Library“ wird JUnit in der Version 4 hinzugefügt. Dies ist notwendig, da Robolectric auf die Annotationen von JUnit 4 angewiesen ist.

Zuletzt muss die Reihenfolge im classpath angepasst werden. Dazu wird im Tab „Order and Export“ das android.jar ausgewählt und in der Reihenfolge unterhalb des robolectric jars platziert.

Um die Testfälle auch ausführen zu können, bedarf es einer neuen Run Configuration für JUnit Tests. Hierbei ist der RadioButton auf “Run all tests in the selected project, package or source folder” zu setzen und das Android Application Project auszuwählen. Als TestRunner dient JUnit 4. Abschließend wird unter „Arguments“ das working directory auf „other“ gesetzt.

3.2.4.2. Umsetzung von Testfall A1_WelcomeScreen mit Robolectric

Für die Lauffähigkeit von Robolectric Tests ist es erforderlich, mittels JUnit 4 Annotation @RunWith() den RobolectricTestRunner als TestRunner festzulegen:

```
@RunWith(RobolectricTestRunner.class)
public class WelcomeScreenActivityTest {
```

Abbildung 3.15 Klassendefinition Robolectric

Methoden, die vor dem Test ausgeführt werden sollen, müssen mit @before annotiert werden, wie es in der folgenden Abbildung 3.16 zu sehen ist.

```
@Before
public void setup() {
    //Roboelectric build chain zum Erzeugen einer Activity
    welcomeScreenActivity = Robolectric
            .buildActivity(WelcomeScreenActivity.class).create().start().visible()
            .get();
    //Shadowobject erzeugen
    shadowWelcomeScreen = Robolectric.shadowOf(welcomeScreenActivity);
}
```

Abbildung 3.16 Setup() Methode Robolectric

Da Robolectric auf JUnit4 aufbaut, müssen Testmethoden mit der Annotation @Test versehen werden.

```
@Test
public void shouldStartAnotherActivity() throws Exception {
    // zu testender Button der WelcomeScreenActivity
    Button btnLaunchMain = (Button) welcomeScreenActivity
            .findViewById(R.id.btnLaunchMain);
    // Klick auf Button
    Robolectric.clickOn(btnLaunchMain);
    // Überprüfung das es sich bei gestarteter Activity um MainActivity handelt
    assertThat(
            shadowWelcomeScreen.peekNextStartedActivityForResult().intent
                    .getComponent(),
            equalTo(new ComponentName(welcomeScreenActivity, MainActivity.class)));
}
```

Abbildung 3.17 Testmethode mit Robolectric

3.2.4.3. Ausführung des Robolectric Testfalls

Im Gegensatz zur Ausführung von Android JUnit Tests benötigt Robolectric keinen Emulator oder angeschlossenes Android Device. Die Tests werden direkt in der JVM ausgeführt.

3.2.4.4. Integration in den Jenkins

Wie beim Anlegen eines Testprojektes ist auch die Integration von Robolectric in den Jenkins aufwändiger als bei einem Android Test Projekt. Hierzu ist die die manuelle Erstellung eines Ant Build Skriptes für das Robolectric Test Projekt erforderlich. Da es sich hier um kein Android Test Project handelt, kann das Build Skript nicht automatisch über die Kommandozeile mittels `android update test-project...` erzeugt werden.

Um letztendlich ein lauffähiges Ant Build Skript zu erzeugen, wurde ein Beispiel aus Github [Schultz 2012] verwendet. Wichtig ist hierbei insbesondere, dass im classpath des Build Skriptes die junit.jar Datei vor der android.jar Datei definiert ist. Ansonsten kommt es bei der Ausführung zu einem Fehler.

3.2.4.5. Bewertung

Eines der wesentlichen Argumente für Robolectric ist die Ausführung der Tests in der JVM. Im Rahmen der Ausführung der Testfälle konnte kein Geschwindigkeitsvorteil gegenüber dem Android Test Framework und der Ausführung als Android JUnit Test ausgemacht werden.
Dies mag aber auch daran liegen, dass die getestete App nicht besonders groß ist (unter 350 KB) und der Emulator bereits lief und nicht jeweils extra gestartet werden musste. Bei wesentlich größeren Apps ist anzunehmen, dass der Prozess des Dexings und der Installation der App und des Testprojektes beim Android JUnit Test in einer viel längeren Ausführungszeit resultiert und der von den Robolectric Entwicklern propagierte Geschwindigkeitsvorteil deutlich wird.

An dieser Stelle soll noch einmal die Bedeutung von Ausführungszeiten bei Unit-Tests verdeutlicht werden. Wie bereits in Kapitel 2.1 beschrieben, ist das übliche Vorgehen bei testgetriebener Entwicklung: Test schreiben, Test ausführen (schlägt fehl), Produktiv-Code schreiben, Test ausführen (erfolgreich), Refactoring durchführen, Test erneut ausführen (hoffentlich immer noch erfolgreich) und anschließend mit dem nächsten Test weitermachen. Wenn die Ausführungszeit eines Tests zu lang wird, dann stellt dies eine erhebliche Störung im Arbeitsprozess dar. Die Testausführung kann dann als unangenehme Wartezeit wahrgenommen werden.

3.2.5 UI-Test mit Robotium

In diesem Kapitel sollen die Anforderungen der RegisterActivity, der UserValidationActivity sowie der WebViewNativeInteractionActivity umgesetzt werden. Hierzu soll zuerst ein Akzeptanztest mit Robotium implementiert werden.

3.2.5.1. Anlegen eines Testprojektes

Ein Robotium Testprojekt wird als Android Test Project angelegt. Nachdem das zu testende Android Application Project ausgewählt wurde, muss lediglich im Build-Path die .jar Datei

von Robotium als Library eingebunden werden. Als Superclass für eine Robotium Testklasse dient die Klasse android.test.ActivityInstrumentationTestCase2.

3.2.5.2. Umsetzung von Testfall “Registrierung” mit Robotium

Zentraler Bestandteil einer Robotium Testklasse ist das Objekt der Klasse Solo. Es ermöglicht den Zugriff auf Views und deren Manipulation genau so wie ein Benutzer dies über die Oberfläche täte.

```
import com.robotium.solo.Solo;

public class RegistrationTest extends ActivityInstrumentationTestCase2<RegisterActivity> {
    // Robotium Solo Klassenobjekt
    private Solo solo;
```

Abbildung 3.18 Klassendefinition Robotium

Robotium baut auf der Struktur der Testklassen von JUnit auf. Daher ist eine Setup-Methode zu implementieren. Das Robotium Klassenobjekt solo bekommt die zu testende Activity übergeben und erhält Zugriff auf die Instrumentation.

```
@Override
protected void setUp() throws Exception {
    super.setUp();
    solo = new Solo(getInstrumentation(), getActivity());
}
```

Abbildung 3.19 Setup() Methode Robotium

Die Eingabe eines Textes in ein Eingabefeld erfolgt in zwei Schritten. Zuerst wird ein View, in diesem Fall ein TextView erzeugt, der mittels der Methode solo.getView(R.id.emailEditText) die entsprechende Referenz erhält. Anschließend wird mittels solo.enterText(„monika@mustermann.org“) der Wert in den TextView eingetragen. Um einen Wert aus einem View auszulesen, muss dieser zuerst wieder erzeugt werden. Anschließend muss die getText() Methode des Views aufgerufen werden.

Es wäre auch möglich, die Methoden wie z.B. solo.clickOnView() auszuführen, indem man den Text angibt, der im jeweiligen View angezeigt wird. Robotium würde die Aktion dann auf dem ersten Element ausführen, dessen Text passt. Es ist aber empfehlenswert, mit den IDs der Elemente zu arbeiten, da der Text zu Anfang der Entwicklung womöglich nicht final ist oder die App später in mehreren Sprachen lokalisiert wird. Um im Testfall Werte zu überprüfen, werden hier die Assertions aus dem JUnit Framework benutzt.

```
public void testRegistration() {
    // Textfeld holen
    EditText emailEditText = (EditText) solo.getView(R.id.emailEditText);
    // Text eintragen
    solo.enterText(emailEditText, "monika@mustermann.org");
    EditText usernameEditText = (EditText) solo.getView(R.id.usernameEditText);
    solo.enterText(usernameEditText, "moni99");
    EditText passwordEditText = (EditText) solo.getView(R.id.passwordEditText);
    solo.enterText(passwordEditText, "qwertz12345");
    // Spinner holen
    Spinner mobileSpinner = (Spinner) solo.getView(R.id.mobileSpinner);
    // Zuerst auf den Spinner klicken damit sich die Auswahl öffnet
    solo.clickOnView(mobileSpinner);
    // Robotium den Text "webOS" suchen lassen um anschließend einen Klick auszuführen
    solo.clickOnText("webOS");
    // CheckBox holen
    CheckBox newsletterCheckbox = (CheckBox) solo.getView(R.id.tosCheckBox);
    // Auf CheckBox klicken um sie zu setzen
    solo.clickOnView(newsletterCheckbox);
    // Button holen
    Button registerButton = (Button) solo.getView(R.id.registerButton);
    // Auf Button klicken. Daraufhin wird die UserValidationActivity gestartet und
    // die zuvor eingegebenen Daten werden angezeigt
    solo.clickOnView(registerButton);
    //Textfeld holen
    TextView emailTextView = (TextView) solo.getView(R.id.emailTextView);
    // Wert der Textfelder überprüfen
    assertEquals("monika@mustermann.org", emailTextView.getText());
    TextView usernameTextView = (TextView) solo.getView(R.id.userNameTextView);
    assertEquals("moni99", usernameTextView.getText());
    TextView passwordTextView = (TextView) solo.getView(R.id.passwordTextView);
    assertEquals("qwertz12345", passwordTextView.getText());
    TextView mobileTextView = (TextView) solo.getView(R.id.mobileTextView);
    assertEquals("webOS", mobileTextView.getText());
    TextView newsletterTextView = (TextView) solo.getView(R.id.newsletterTextView);
    assertEquals("true", newsletterTextView.getText());
}
```

Abbildung 3.20 Testfall A2_Registrierung_Daten mit Robotium

Nach jeder Testmethode soll sichergestellt werden, dass alle geöffneten Activities sauber beendet werden. Dies wird durch das Überschreiben der tearDown() Methode und dem Aufruf der entsprechenden Solo Klassenmethode erreicht. Ohne diesen Schritt kann es bei der Ausführung mehrerer Testmethoden hintereinander zu Konflikten kommen.

```
// Alle offenen Activities werden beendet
@Override
protected void tearDown() throws Exception {
    solo.finishOpenedActivities();
    super.tearDown();
}
```

Abbildung 3.21 tearDown() Methode Robotium

Auf die Implementierung der zugehörigen Android Komponenten RegisterActivity und UserValidationActivity wird in dieser Arbeit nicht näher eingegangen.

3.2.5.3. Umsetzung von Testfall A3_Interaktion_WebView_Nativ mit Robotium

Robotium besitzt die Fähigkeit, mit WebViews und HTML Inhalten umzugehen. Die Methoden hierzu sind der bekannten Selenium2/Webdriver API nachempfunden. Um mit Elementen im WebView zu interagieren, wird eine Implementierung der Webdriver Klasse By benutzt, um diese zu lokalisieren. Neben der Möglichkeit, ein Webelement z.B. anhand des Namens, des Textinhaltes, des Klassennamens oder XPath bzw. CSS Ausdruck zu identifizieren, kann hier auch die ID des Elements verwendet werden. Dies ist auch hier die eindeutigste Methode auf ein Element zuzugreifen. Natürlich muss im HTML Inhalt eine ID vergeben sein. Hinsichtlich der Testbarkeit der Anwendung empfiehlt es sich also, dies gleich bei der Entwicklung sicherzustellen.

```
public void testWebViewNativeInteraction(){
    // Locator für die WebElemente
    By inputFirstName = By.id("fname");
    By inputLastName = By.id("lname");
    By buttonUpdateNative = By.id("updateNativeBtn");
    // Warten auf ein WebElement
    solo.waitForWebElement(inputFirstName);
    // Text im WebView eingeben
    solo.typeTextInWebElement(inputFirstName, "Hans");
    solo.typeTextInWebElement(inputLastName, "Webtester");
    // Auf Button im WebView klicken
    solo.clickOnWebElement(buttonUpdateNative);
    // Eine Sekunde lang warten
    solo.sleep(1000);
    // native Elemente
    EditText fnameEditText = (EditText) solo.getView(R.id.fnameEditText);
    EditText lnameEditText = (EditText) solo.getView(R.id.lnameEditText);
    Button updateWebViewButton = (Button) solo.getView(R.id.updateWebViewButton);
    // Überprüfen ob Werte aus dem WebView in den nativen Textfelder eingetragen wurden
    assertEquals("Hans", fnameEditText.getText().toString());
    assertEquals("Webtester", lnameEditText.getText().toString());
    // Inhalt der Textfelder löschen
    solo.clearEditText(fnameEditText);
    solo.clearEditText(lnameEditText);
    // neue Werte eintragen
    solo.enterText(fnameEditText, "Herbert");
    solo.enterText(lnameEditText, "Nativetester");
    solo.clickOnView(updateWebViewButton);
    // wieder eine Sekunde lang warten
    solo.sleep(1000);
    // Überprüfen ob Werte aus den Nativen Elementen im WebView angezeigt werden
    assertEquals("Herbert", solo.getWebElement(inputFirstName,0).getText());
    assertEquals("Nativetester", solo.getWebElement(inputLastName,0).getText());
}
```

Abbildung 3.22 Testfall A3_Interaktion_WebView_Nativ mit Robotium

3.2.5.4. Implementierung der WebViewNativeInteractionActivity

Die Beschreibung der Implementierung dieser Activity konzentriert sich auf den Datenaustausch zwischen dem HTML Inhalt des WebViews und den nativen Elementen der App. Der Datenaustausch wird mittels eines JSON Objektes realisiert, dass der Methode setNames() im Javascript der geladenen Webseite übergeben wird.

```
public void sendNamesToWebView() {
    JSONObject namesJson = new JSONObject();
    try {
        namesJson.put("fname", fnameEditText.getText().toString());
        namesJson.put("lname", lnameEditText.getText().toString());
        webview.loadUrl( "javascript:setNames(" + namesJson.toString() + ")" );
    } catch (JSONException e) {
        Log.e(getPackageName(), "Failed to create JSON object for web view");
    }
}
```

Abbildung 3.23 Klasse WebViewInterface

Um den Datenaustausch mit dem WebView zu gewährleisten, bedarf es der Implementierung eines JavascriptInterface. Dies kann innerhalb der Activity als innere Klasse umgesetzt werden.

```
public class WebViewInterface {
    Context mContext;

    // Instanzierung des Interface und setzen des Kontext
    WebViewInterface(Context c) {
        mContext = c;
    }
```

Abbildung 3.24 Klasse WebViewInterface

Jetzt fehlt noch eine Methode, um den Inhalt der Textfelder aus dem WebView in die nativen Textfelder zu schreiben.

```
@JavascriptInterface
public void updateNames(String namesJsonString) {
    Log.d(getPackageName(), "Sent from webview: " + namesJsonString);
    try {

        JSONObject namesJson = new JSONObject(namesJsonString);
        final String firstName = namesJson.getString("fname");
        final String lastName = namesJson.getString("lname");

        runOnUiThread(new Runnable() {
            public void run() {
                fnameEditText.setText(firstName);
                lnameEditText.setText(lastName);
            }
        });

    } catch (JSONException e) {
        Log.e(getPackageName(), "Failed to create JSON object from web view data");
    }
}
```

Abbildung 3.25 updateNames Methode

Damit die Methode updateNames(String namesJsonString) dem Javascript zur Verfügung steht, muss sie ab targetSDkVersion 17 oder höher mit @JavascriptInterface annotiert werden. Zuletzt muss die onStart() Methode überschrieben werden, die direkt nach onCreate() ausgeführt wird. Der WebView erhält das JavascriptInterface und die Berechtigung Javascript ausführen zu dürfen (siehe Abbildung 3.16).

```
@SuppressLint("SetJavaScriptEnabled")
@Override
public void onStart() {
    super.onStart();
    WebSettings webSettings = webview.getSettings();
    webSettings.setJavaScriptEnabled(true);
    webview.loadUrl("file:///android_asset/html/webViewContent.html");

    webview.addJavascriptInterface(new WebViewInterface(this), "Android");

}
```

Abbildung 3.26 onStart() Methode

Die WebViewNativeInteractionActivity verfügt nun über die Fähigkeit, mit den Inhalten des WebViews zu interagieren.

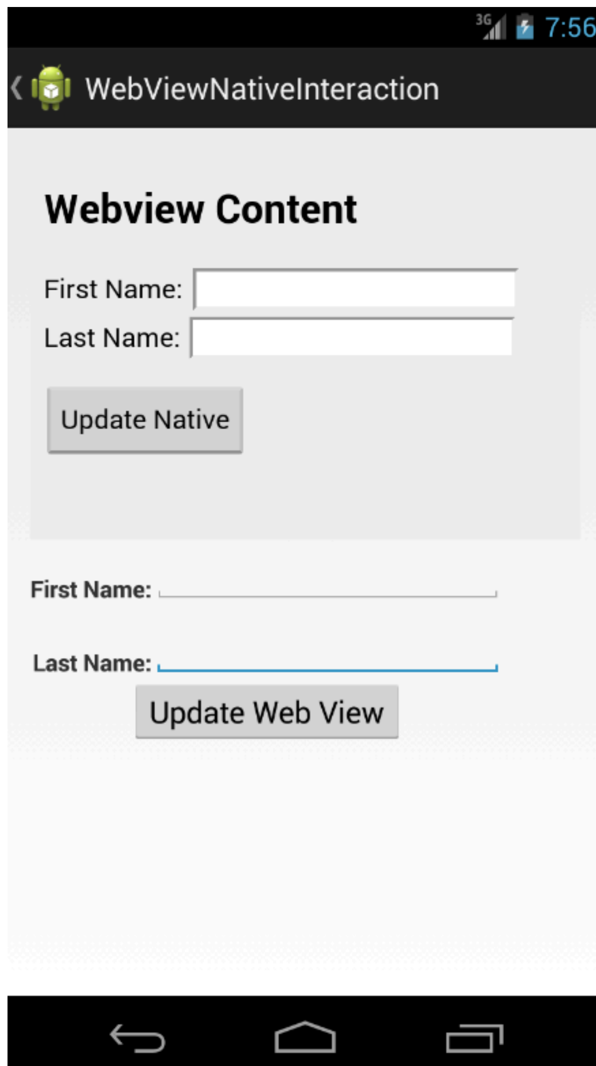

Abbildung 3.27 WebViewNativeInteractionActivity

Abbildung 3.27 zeigt die Oberfläche der fertig implementierten Activity.

3.2.5.5. Integration in den Jenkins

Zuerst gilt es wieder ein Ant Build File für das Testprojekt zu erzeugen.

```
android update test-project -m ../TDDAndroid -p
TDDAndroidRobotiumTests
```

Als Android Test Project gestaltet sich die Integration in den Jenkins analog zu Kapitel 3.2.3.6.

3.2.5.6. Bewertung

Robotium bietet die Möglichkeit, schnell und einfach aussagekräftige UI-Tests zu implementieren. Über die Klasse Solo kann man auf alle wichtigen Elemente der UI zugreifen. Ein gutes Argument für den Einsatz von Robotium ist zudem die Unterstützung von WebViews, die bei Hybrid Apps einen wesentlichen Bestandteil der Funktionalität bilden.
Wie bereits erwähnt, orientieren sich die Methoden für das Testen von Inhalten im WebView an dem bekannten Framework Selenium2/Webdriver. Wer Erfahrung mit Selenium2/Webdriver hat, sollte keine Schwierigkeiten haben, effektive Tests für Hybrid-Apps zu schreiben.

3.2.6 UI-Test mit Espresso

Im folgenden Kapitel soll der Testfall A2_Registrierung_Daten noch einmal mit Espresso umgesetzt werden. Aufgrund fehlender Unterstützung für WebViews, kann der Testfall A3_Interaktion_WebView_Nativ nicht mit Espresso umgesetzt werden.

3.2.6.1. Anlegen eines Testprojektes

Ein Espresso Testprojekt wird wie bei Robotium als Android Test Project angelegt. Das zu testende Android Application Project wird ausgewählt und im Build-Path die .jar Datei von Espresso als Library eingebunden. Als Superclass für eine Espresso Testklasse dient ebenfalls die Klasse android.test.ActivityInstrumentationTestCase2.

Um Espresso Tests auszuführen, bedarf es aber einer Änderung in der Run Configuration. Während Robotium Tests den standard InstrumentationTestRunner aus dem Paket android.test benutzt, verlangt Espresso den sogenannten GoogleInstrumentationTestRunner. Hierbei handelt es sich um eine Weiterentwicklung, die laut Google [Google2013f] für eine höhere Stabilität der Tests sorgen soll, so garantiert z.B. der GoogleInstrumentationTestRunner, dass die jeweilige onCreate() Methode auf Seiten der zu testenden App beendet ist, bevor der Test startet.

3.2.6.2. Umsetzung von Testfall A2_Registrierung_Daten mit Espresso

Die folgende Abbildung 3.17 zeigt die wichtigsten zu importierenden Espresso Methoden zum Auswählen, Ausführen und Überprüfen von UI-Elementen.

```
// Espresso Importe
import static com.google.android.apps.common.testing.ui.espresso.Espresso.onView;
import static com.google.android.apps.common.testing.ui.espresso.Espresso.onData;
import static
com.google.android.apps.common.testing.ui.espresso.action.ViewActions.*;
import static
com.google.android.apps.common.testing.ui.espresso.matcher.ViewMatchers.*;
import com.google.android.apps.common.testing.ui.espresso.assertion.ViewAssertions;
```

Abbildung 3.28 Espresso Importe

Auch bei Espresso muss die setUp() Methode implementiert sein.

```
@Override
protected void setUp() throws Exception{
    super.setUp();
    getActivity();
}
```

Abbildung 3.29 Setup() Methode Espresso

Wie bereits in Kapitel 2.5.6 beschrieben, orientiert sich die Umsetzung der Testmethode mit Espresso daran, wie ein realer Benutzer vorgehen würde, d.h. ein UI-Element auswählen, darauf eine Aktion ausführen und das erwarte Ergebnis prüfen. Hierzu stellt Espresso drei Methoden zur Verfügung:

- Auswählen: onView(Matcher<View>)
- Ausführen: perfom(ViewAction)
- Prüfen: check(ViewAssertion)

Die Verkettung der Methoden führt zu kompaktem und gut lesbarem Test-Code. Wie Robotium besitzt auch Espresso die Möglichkeit, die UI-Elemente mittels des ViewMatchers withText auszuwählen. Da aber, wie bereits in Kapitel 3.2.5.2 beschrieben, dies die Wartbarkeit des Test-Codes negativ beeinflussen kann, ist es auch hier besser, die Auswahl über die eindeutigen IDs der Elemente durchzuführen.

```
public void testRegistration(){
    // Textfeld holen und Text eingeben
    onView(withId(R.id.emailEditText)).perform(typeText("monika@mustermann.org"));
    onView(withId(R.id.usernameEditText)).perform(typeText("moni99"));
    onView(withId(R.id.passwordEditText)).perform(typeText("qwertz12345"));
    // Spinner holen und durch klick öffen
    onView(withId(R.id.mobileSpinner)).perform(click());
    // Element mit dem Text "webOS" suchen und auswählen
    onData(allOf(is(instanceOf(String.class)), is("webOS")))
      .perform(click());
    // CheckBox holen und klicken
    onView(withId(R.id.tosCheckBox)).perform(click());
    // Button holen und klicken
    onView(withId(R.id.registerButton)).perform(click());
    // Textfeld holen und Inhalt überprüfen
    onView(withId(R.id.emailTextView)).check(ViewAssertions.matches(withText("monika@mustermann.org")));
    onView(withId(R.id.userNameTextView)).check(ViewAssertions.matches(withText("moni99")));
    onView(withId(R.id.passwordTextView)).check(ViewAssertions.matches(withText("qwertz12345")));
    onView(withId(R.id.newsletterTextView)).check(ViewAssertions.matches(withText("true")));
}
```

Abbildung 3.30 Testfall A2_Registrierung_Daten mit Espresso

Bei der Betrachtung des mit Espresso implementierten Testfalls in Abbildung 3.30 wird deutlich, dass dieser im Vergleich zur Implementierung mit Robotium wesentlich kompakter ausfällt.

3.2.6.3. Integration in den Jenkins

Da es sich beim Espresso Projekt ebenfalls um ein Android Test Project handelt, ist die Integration in den Jenkins analog zu Robotium bzw. dem Android Test Framework durchzuführen (siehe Kapitel 3.2.3.6).

3.2.6.4. Bewertung

Durch die fehlende Unterstützung für WebViews ist Espresso für die Entwicklung von Hybrid Apps leider nicht geeignet. Für Native Apps ist Espresso allerdings eine gute Wahl, vor allem aufgrund des kompakten und gut lesbaren Test-Codes. Das stärkste Argument für den Einsatz von Espresso dürften aber die im Vergleich zu Robotium deutlich kürzeren Ausführungszeiten der Tests sein. Bei der Ausführung gleicher Tests, die sowohl mit Robotium als auch mit Espresso implementiert wurden, konnte bei Espresso eine zwischen 40 und 50 Prozent kürzere Ausführungszeit gemessen werden. In Anbetracht eines Testprojektes mit einer großen Anzahl von UI-Tests ist dieser Zeitfaktor ein großer Pluspunkt für Espresso bei der testgetriebenen Entwicklung von Apps.

Eine höhere Stabilität von Espresso im Vergleich zu Robotium konnte im Rahmen dieser Arbeit nicht festgestellt werden, da es bei keinem der Testframeworks zu Stabilitätsproblemen bei den implementierten Tests kam.

3.3 Weitere Werkzeuge für die Qualitätssicherung

Im vorhergegangenen Kapitel 3.2 wurde der Einsatz von Unit- und UI-Tests beschrieben. Für die Entwicklung von Android Apps stehen aber noch weitere Werkzeuge zur Verfügung, die während des gesamten Entwicklungsprozesses einer Android App bei der fortlaufenden Qualitätssicherung unterstützen.

3.3.1 Stresstest mit Monkey

Die im Kapitel 3.2 beschriebenen Tests folgen einer gewissen Geschäftslogik. Ein realer Endnutzer klickt aber auch mal so in der Anwendung herum, wie es sich der Entwickler der App nicht unbedingt vorher gedacht hat. Dieses Verhalten simuliert Monkey mittels randomisierter Eingaben.

3.3.1.1. Ausführung von Stresstests mit Monkey

Ein Stresstest mit Monkey kann direkt über die Shell des Android Gerätes oder Emulators gestartet werden. Kommt es dabei zu einem Absturz der App oder es tritt ein anderer Fehler auf, stoppt Monkey die App und gibt den Fehler im LogCat aus. Der Detailgrad für die Ausgabe des Testverlaufs kann gewählt werden.

Um den Stresstest zu starten, benötigt man den Paketnamen der App auf dem Device und wählt die Anzahl der zufälligen Ereignisse aus. Durch die Angabe eines oder mehrerer Pakete wird erreicht, dass Monkey nur innerhalb dieser Pakete testet und Versuche, in weitere Pakete zu navigieren, blockiert. Die Ausgabe in der Konsole gibt u.a. Feedback, wie lange die Ausführung der Tests gedauert hat oder wie lange die Netzwerkverbindung durch Monkey unterbrochen worden war.

```
adb shell monkey -p de.haw_hamburg.tddandroid -v 1000
```

```
C:\adt-bundle-windows-x86\sdk\platform-tools>adb shell monkey -p de.haw_hamburg.
tddandroid -v 1000
:Monkey: seed=1413370352477 count=1000
:AllowPackage: de.haw_hamburg.tddandroid
:IncludeCategory: android.intent.category.LAUNCHER
:IncludeCategory: android.intent.category.MONKEY
// Event percentages:
//   0: 15.0%
//   1: 10.0%
//   2: 2.0%
//   3: 15.0%
//   4: -0.0%
//   5: 25.0%
//   6: 15.0%
//   7: 2.0%
//   8: 2.0%
//   9: 1.0%
//   10: 13.0%
:Switch: #Intent;action=android.intent.action.MAIN;category=android.intent.categ
ory.LAUNCHER;launchFlags=0x10200000;component=de.haw_hamburg.tddandroid/.Welcome
ScreenActivity;end
    // Allowing start of Intent { act=android.intent.action.MAIN cat=[android.in
tent.category.LAUNCHER] cmp=de.haw_hamburg.tddandroid/.WelcomeScreenActivity } i
n package de.haw_hamburg.tddandroid
:Sending Trackball (ACTION_MOVE): 0:(4.0,3.0)
:Sending Touch (ACTION_DOWN): 0:(644.0,206.0)
:Sending Touch (ACTION_UP): 0:(636.6435,190.29468)
:Sending Touch (ACTION_DOWN): 0:(508.0,255.0)
:Sending Touch (ACTION_UP): 0:(416.0349,208.01707)
:Sending Touch (ACTION_DOWN): 0:(100.0,240.0)
:Sending Touch (ACTION_UP): 0:(98.61426,238.52473)
:Sending Trackball (ACTION_MOVE): 0:(4.0,-5.0)
:Sending Flip keyboardOpen=false
:Sending Touch (ACTION_DOWN): 0:(279.0,148.0)
:Sending Touch (ACTION_UP): 0:(294.631,153.76427)
    // Rejecting start of Intent { act=android.intent.action.MAIN cat=[android.i
ntent.category.LAUNCHER] cmp=com.android.email/.activity.Welcome } in package co
m.android.email
:Sending Touch (ACTION_DOWN): 0:(671.0,44.0)
:Sending Touch (ACTION_UP): 0:(674.2681,46.28614)
:Sending Trackball (ACTION_MOVE): 0:(2.0,2.0)
:Sending Trackball (ACTION_MOVE): 0:(3.0,2.0)
:Sending Trackball (ACTION_MOVE): 0:(-2.0,2.0)
    //[calendar_time:2014-10-30 15:06:57.268  system_uptime:285233898]
    // Sending event #100
:Sending Trackball (ACTION_MOVE): 0:(3.0,-2.0)
```

Abbildung 3.31 Konsolenausgabe Monkey

In Abbildung 3.31 ist die Ausgabe eines Tests zu sehen, welcher 1000 Nutzerevents simuliert hat. Man kann erkennen, an welcher Position der Oberfläche ein Touchevent ausgelöst und wann ein anderer Intent geöffnet wurde.

Durch den Befehl `-throttle <Zeit in Milisekunden>` kann eine Verzögerung zwischen den einzelnen Ereignissen eingestellt werden.

3.3.1.2. Bewertung

Monkey ist ein hilfreiches Werkzeug, wenn es darum geht, die Robustheit einer App mittels randomisierter Eingaben zu testen. Man hat keinen Einfluss darauf, wann und welche Events auf der App ausgeführt werden, vergleichbar mit dem Verhalten eines realen Endnutzers.

3.3.2 Statische Code-Analyse mit Android Lint

Während die bisher vorgestellten Test Frameworks und auch Monkey die funktionalen Anforderungen testen, eignet sich Android Lint dazu, strukturelle Probleme im Code zu erkennen.

3.3.2.1. Ausführung von Android Lint

Um die Analyse durch Android Lint auszuführen, muss in Eclipse lediglich unter *Window -> Show View -> Other* der *Lint Warnings View* und das Android Projekt ausgewählt werden.

Problems | Javadoc | Declaration | Console | Progress | LogCat | Lint Warnings

10 errors, 200 warnings

Description	Category	Location
This LinearLayout layout or its LinearLayout parent is useless	Performance	abc_activity_chooser_view_list_ite
[Accessibility] Missing contentDescription attribute on image (13 items)	Accessibility	abc_search_view.xml:64 in layout
[Accessibility] Missing contentDescription attribute on image	Accessibility	abc_search_dropdown_item_icon
[Accessibility] Missing contentDescription attribute on image	Accessibility	abc_search_dropdown_item_icon
[Accessibility] Missing contentDescription attribute on image	Accessibility	abc_search_dropdown_item_icon
[Accessibility] Missing contentDescription attribute on image	Accessibility	abc_list_menu_item_icon.xml:17 i
[Accessibility] Missing contentDescription attribute on image	Accessibility	abc_activity_chooser_view_list_ite
[Accessibility] Missing contentDescription attribute on image	Accessibility	abc_activity_chooser_view_includ
[Accessibility] Missing contentDescription attribute on image	Accessibility	abc_activity_chooser_view_includ
[Accessibility] Missing contentDescription attribute on image	Accessibility	abc_action_mode_close_item.xml
[Accessibility] Missing contentDescription attribute on image	Accessibility	abc_action_bar_title_item.xml:25 i
[Accessibility] Missing contentDescription attribute on image	Accessibility	abc_action_bar_home.xml:29 in la
[Accessibility] Missing contentDescription attribute on image	Accessibility	abc_action_bar_home.xml:22 in la
[Accessibility] Missing contentDescription attribute on image	Accessibility	abc_action_bar_decor_overlay.xm
Missing density variation folders in TDDAndroidRobotiumTests\res: drawable-xxhdpi (3 items)	Usability:Icons	TDDAndroid
The following unrelated icon files have identical contents: abc_textfield_search_selected_holo_dark.9.png, abc_t	Usability:Icons	abc_textfield_search_selected_hol
The resource R.layout.support_simple_spinner_dropdown_item appears to be unused (68 items)	Performance	support_simple_spinner_dropdow
Possible overdraw: Root element paints background ?attr/actionBarItemBackground with a theme that also pai	Performance	abc_action_bar_home.xml:21 in la
Nested weights are bad for performance (2 items)	Performance	abc_search_view.xml:90 in layout
Use a layout_width of 0dp instead of wrap_content for better performance (3 items)	Performance	abc_search_view.xml:78 in layout
Should explicitly set android:allowBackup to true or false (it's true by default, and that can have some security i	Security	AndroidManifest.xml:13 (TDDAnc
Consider adding android:layout_alignStart="@+id/fnameEditText" to better support right-to-left layouts (56 ite	Bi-directional Text	activity_web_view_native_interact
[I18N] Hardcoded string "Update Web View", should use @string resource (10 items)	Internationalization	activity_web_view_native_interact
This text field does not specify an inputType or a hint (12 items)	Usability	activity_web_view_native_interact
The abc_list_divider_holo_light.9.png icon has identical contents in the following configuration folders: drawab	Usability:Icons	abc_list_divider_holo_light.9.png i
This tag and its children can be replaced by one <TextView/> and a compound drawable	Performance	abc_activity_chooser_view_list_ite
Invalid layout param in a LinearLayout: layout_alignParentTop	Performance	abc_action_bar_decor_overlay.xm
Attribute "showAsAction" is only used in API level 11 and higher (current min is 10) (11 items)	Correctness	web_view_native_interaction.xml:
Not targeting the latest versions of Android; compatibility modes apply. Consider testing and updating this ver	Correctness	AndroidManifest.xml:9 (TestApp)
<uses-sdk> tag should specify a target API level (the highest verified version; when running on later versions, c	Correctness	AndroidManifest.xml:7 (TDDAndr
Wrong orientation? No orientation specified, and the default is horizontal, yet this layout has multiple children	Correctness	abc_action_bar_decor_overlay.xm
Windows file separators (\) must be escaped (\\); use c:\\adt-bundle-windows-x86\\sdk (3 items)	Correctness	local.properties:10 (TDDAndroidJu
Attribute is missing the Android namespace prefix (2 items)	Correctness	abc_search_dropdown_light.xml:1
Invalid package reference in library; not included in Android: javax.inject. Referenced from dagger.internal.Keys	Correctness	espresso-1.1-bundled.jar in libs (T
Class referenced in the manifest, de.haw_hamburg.testapp.WebviewScreenActivity, was not found in the projec	Correctness	AndroidManifest.xml:28 (TestApp
This class should be public (android.support.v7.internal.widget.ActionBarView.HomeView)	Correctness	TDDAndroid
Duplicate id @+id/image, already defined earlier in this layout	Correctness	abc_activity_chooser_view_includ

Abbildung 3.32 Android Lint Warnings für TDDAndroid

Abbildung 3.32 zeigt die Lint Warnings View von TDDAndroid. Beim mehrfachen Auftritt einer Warnung, wird eine Gesamtüberschrift gebildet unter welcher alle Codezeilen aufgelistet werden, die dieser Warnung entsprechen. Durch Klick auf eine dieser Warnungen gelangt man direkt zur entsprechenden Codezeile, um eine Verbesserung vornehmen zu können.

3.3.2.2. Bewertung

Android Lint ist ein einfach zu benutzendes Werkzeug zur schnellen Erkennung möglicher Fehlerquellen und Verbesserungsmöglichkeiten im Sourcecode. Die Darstellung der gefunden Warnungen ist übersichtlich und ermöglicht dem Entwickler, deren Ursache schnell zu beheben.

4 Fazit

Ziel dieser Arbeit war es, eine Android App testgetrieben zu entwickeln und einzelne Werkzeuge nach ihrer Eignung für diesen Zweck zu evaluieren. Alle an die App gestellten Anforderungen aus Kapitel 3.1 konnten umgesetzt und erfüllt werden.

Ich komme zu dem Schluss, dass testgetriebene Entwicklung von Android Apps möglich und ohne größere Schwierigkeiten umzusetzen ist. Es stehen eine ganze Reihe von Werkzeugen zur Verfügung, um diese Methode der Softwareentwicklung auch effektiv einzusetzen. Alle in dieser Arbeit vorgestellten Werkzeuge haben sich dafür als geeignet erwiesen. Das Programmieren nach der Test First Methode hat bereits beim Schreiben der Tests zu hilfreichen Designentscheidungen geführt. Die dadurch forcierte Überlegung, was man mindestens braucht, um die geforderte Funktionalität zu gewährleisten, und die anschließende Implementierung genau dieser Anforderung führte, nach meiner Meinung, zu effizientem Code. Es entsteht dabei außerdem von Anfang an gut testbarer Code, z.B. durch die konsequente Verwendung eindeutiger IDs von UI-Elementen. Dies wurde vor allem bei der Umsetzung der HTML-Inhalte für die WebViewNativeInteractionActivity deutlich.

Bei der Frage, ob man für Unit-Tests besser das Android Test Framework oder Robolectric einsetzt, würde ich mich für Robolectric entscheiden. Handelt es sich bei der zu entwickelnden App um ein umfangreiches Projekt, das einerseits zu vielen Klassen und andererseits zu einer großen Installationsdatei führt, wodurch das Dexing und die

Installation der App vor der eigentlichen Testausführung stärker ins Gewicht fallen, ist nur durch Robolectric eine kurze Testausführung gewährleistet.

Im Bereich der UI-Tests empfehle ich Espresso, da hier sehr kompakte und gut lesbare Tests entstehen. Entscheidendes Argument für den Einsatz von Espresso ist aber auch hier die deutlich kürzere Ausführungszeit der Tests im Vergleich zu Robotium. Jedoch gilt diese Empfehlung nur bei der Entwicklung nativer Apps. Aufgrund der nicht vorhandenen Unterstützung von WebViews kann Espresso bei Hybrid Apps nicht eingesetzt werden. Hier kann dann aber auf Robotium und seine an der populären Selenium2/WebDriver API nachempfundenen Methoden zurückgegriffen werden.

Der Einsatz von Versionsverwaltung hat sich dank Git als äußerst problemlos dargestellt. In wenigen Minuten war das Projekt unter Versionsverwaltung gestellt und auf GitHub gehostet. Die Eigenschaft von Git als verteiltes Versionskontrollsystem ermöglichte es zudem, weiter den aktuellen Stand zu versionieren, auch wenn ohne Verbindung zum Remote Repository gearbeitet werden musste. Zudem hat sich Jenkins als zuverlässiger Integrationsserver erwiesen. Die Einrichtung des Jenkins, um damit die Android App und die Testprojekte automatisiert zu bauen und anschließend die Tests durchführen zu lassen, hat sich als unkomplizierter herausgestellt, als ich im Vorfeld dieser Arbeit angenommen habe. Lediglich die Integration des Robolectric Test Projektes erforderte einigen Aufwand, da das benötige XML Skript manuell erstellt werden musste.

Stresstests und statische Codeanalyse stehen mit dem Android SDK in Form der Werkzeuge Monkey und Lint von Anfang an zur Verfügung. Während Monkey den Entwickler unterstützt, die Robustheit der App zu prüfen, ermöglicht Lint vor allem mögliche Fehlerquellen und Ursachen für Performanceverluste zu finden.

Die testgetriebene Entwicklung einer Android App ist nicht nur dank der Unterstützung durch eine Vielzahl von Werkzeugen möglich, sondern, meiner Meinung nach, auch die beste Methode, um vom ersten Schritt an die Qualität einer App zu sichern und dadurch mit dieser am Markt bestehen zu können.

Literaturverzeichnis

[Becker2010] Becker, Arno; Pant, Marcus: Android 2. Grundlagen und Programmierung (2010); dpunkt Verlag; 2. Auflage; ISBN 3-8986-4677-7

[Bitkom2014] Bitkom: App-Markt wächst rasant (2014); Abgerufen am 10.11.2014; URL http://www.bitkom.org/de/presse/8477_79327.aspx

[Eclipse2014] Eclipse Foundation: Eclipse Public License - v 1.0 (2014); Abgerufen am 05.11.2014; URL http://www.eclipse.org/org/documents/epl-v10.php

[Fowler1999] Fowler, Martin; Beck, Kent; Brant, John; Opdyke, William; Roberts, Don: Refactoring: Improving the Design of Existing Code (1999); Addison-Wesley Professional; ISBN 0-2014-8567-7

[Fowler2006] Fowler, Martin: Continuous Integration (2006); Abgerufen am 20.07.2014; URL http://www.martinfowler.com/articles/continuousIntegration.html

[Git2011] Git: git –fast version control (2011); Abgerufen am 20.09.2014; URL http://git-scm.com/about

[Google2013a] Google: Android Developers - Testing Fundamentals (2013); Abgerufen am 01.06.2014; URL http://developer.android.com/tools/testing/testing_android.html

[Google2013b] Google: UI/Application Exerciser Monkey (2013); Abgerufen am 05.06.2014; URL http://developer.android.com/tools/help/monkey.html

[Google2013c] Google: monkeyrunner; Abgerufen am 05.06.2014; URL http://developer.android.com/tools/help/monkeyrunner_concepts.html

[Google 2013d] Google: android-test-kit – a fun little Android UI test API (2013); Abgerufen am 15.07.2014; URL https://code.google.com/p/android-test-kit/wiki/Espresso

[Google 2013e] Google: Android Tools Project Site – Lint (2013); Abgerufen am 15.07.2013; URL http://tools.android.com/tips/lint

[Google 2013f] Google: GoogleInstrumentationTestRunner – Google´s Testing Tools For Android (2013); Abgerufen am 15.07.2014; URL https://code.google.com/p/android-test-kit/wiki/GoogleInstrumentationTestRunner

[GoogleTechTalk2013] GTAC 2013: Espresso: Fresh Start to Android UI Testing (2013); Abgerufen am 04.06.2014; URL http://www.youtube.com/watch?v=T7ugmCuNxDU

[Greb2012] Greb, Chuck: Android TDD (2012); Abgerufen am 11.08.2014; URL http://de.slideshare.net/ecgreb/android-tdd-23738940

[Guru99.com2013] Guru99.com: Complete Guide to Android Testing & Automation; Abgerufen am 15.08.2014; URL http://www.guru99.com/why-android-testing.html

[IDC2014] IDC: Smartphone OS Market Share, Q2 2014 (2014); Abgerufen am 11.11.2014; URL http://www.idc.com/prodserv/smartphone-os-market-share.jsp

[Jenkins2014] Jenkins-Ci.org: Meet Jenkins (2014); Abgerufen am 10.09.2014; URL https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins

[Matzke2005] Matzke, Bernd: Ant: Eine praktische Einführung in das Java-Build-Tool (2005); dpunkt; ISBN 3-8986-4327-1

[MockFlow2014] MockFlow: Online Wireframe Tool and Design Cloud (2014); Abgerufen am 20.10.2014; URL http://www.mockflow.com

[Robolectric2013] Robolectric.org; Robolectric, Test-Drive Your Android Code (2013); Abgerufen am 07.06.2014; URL http://robolectric.org

[Robotium2013] Robotium: The world's leading Android™ test automation framework (2013); Abgerufen am 10.07.2014; URL https://code.google.com/p/robotium/

[Schultz2012] Schultz, Justin: Github. Android-Robolectric-Example (2012); Abgerufen am 08.06.2014;
URL https://github.com/jmschultz/Eclipse-Robolectric-Example/blob/master/android-project-test/build.xml

[Verclas2011] Verclas, Stephan; Linnhoff-Popien, Claudia: Smart Mobile Apps (2011); Springer-Verlag, ISBN 3-6422-2258-7

[Westphal2006] Westphal, Frank: Testgetriebene Entwicklung - JUnit & FIT (2006), dpunkt Verlag; ISBN 3-8986-4220-8

[Wikipedia2014] Wikipedia: Android (Betriebssystem) (2014); Abgerufen am 25.10.2014; URL http://de.wikipedia.org/wiki/Android_(Betriebssystem)

Anhang

Abbildungsverzeichnis

Abbildung 2.1 Weltweite Marktanteile mobiler Betriebssysteme, [IDC2014] 13
Abbildung 2.2 Android Systemarchitektur, [Becker2010] 14
Abbildung 2.3 Lebenszyklus einer Activity, [Becker2010] 17
Abbildung 2.4 Android Test Framework, [Google2013a] 19
Abbildung 2.5 Klassendiagramm Android Test Framework, [Guru99.com2013] 20
Abbildung 2.6 Shadow Objects unter Robolectric, [Greb2012] 23
Abbildung 3.1 Hierarchie der Activities 27
Abbildung 3.2 Design WelcomeScreenActivity 28
Abbildung 3.3 Design MainActivity 29
Abbildung 3.4 Design RegisterActivity und UserValidationActivity 30
Abbildung 3.5 Design WebViewNativeInteractionActivity 31
Abbildung 3.6 Design SimpleBrowserActivity 32
Abbildung 3.7 TDDAndroid Projekt auf GitHub 36
Abbildung 3.8 Klassendefinition JUnit 37
Abbildung 3.9 Setup() Methode ActivityUnitTestCase 37
Abbildung 3.10 Testmethode ActivityUnitTestCase 37
Abbildung 3.11 Implementierung WelcomeScreenActivity 38
Abbildung 3.12 WelcomeScreenActivity 39
Abbildung 3.13 JUnit Testergebnis 40
Abbildung 3.14 TDDAndroid im Jenkins 41
Abbildung 3.15 Klassendefinition Robolectric 44
Abbildung 3.16 Setup() Methode Robolectric 44
Abbildung 3.17 Testmethode mit Robolectric 45
Abbildung 3.18 Klassendefinition Robotium 47
Abbildung 3.19 Setup() Methode Robotium 47
Abbildung 3.20 Testfall A2_Registrierung_Daten mit Robotium 48
Abbildung 3.21 tearDown() Methode Robotium 48
Abbildung 3.22 Testfall A3_Interaktion_WebView_Nativ mit Robotium 49
Abbildung 3.23 Klasse WebViewInterface 50
Abbildung 3.24 Klasse WebViewInterface 50
Abbildung 3.25 updateNames Methode 51
Abbildung 3.26 onStart() Methode 51
Abbildung 3.27 WebViewNativeInteractionActivity 52
Abbildung 3.28 Espresso Importe 53
Abbildung 3.29 Setup() Methode Espresso 54
Abbildung 3.30 Testfall A2_Registrierung_Daten mit Espresso 54
Abbildung 3.31 Konsolenausgabe Monkey 56
Abbildung 3.32 Android Lint Warnings für TDDAndroid 57

Tabellenverzeichnis

Tabelle 3.1 Android UI Standardelemente 26

Versicherung über Selbstständigkeit

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

Hamburg, den ________________ ____________________________