Otto-von-Guericke-Universität Magdeburg

Fakultät für Informatik
Institut für Technische und Betriebliche Informationssysteme

Diplomarbeit

Cross-Platform-Entwickung unter iOS und Android: Technologieüberblick und Prototyp-basierte Bewertung

Verfasser:
Felix Alcalá Toca

31. Dezember 2011

Betreuer:
Prof. Dr. rer. nat. habil. Gunter Saake
Dr.-Ing. Sebastian Günther (Vrije Universiteit Brussel)

Universität Magdeburg
Fakultät für Informatik
Postfach 4120, 39016 Magdeburg

Es gibt *keinen* Grund,
nicht glücklich zu sein.

Sprachliche Anmerkungen

Im Interesse erhöhter Sprachverständlichkeit und verringerter Redundanz wird bei Personenbezeichnungen stets nur die maskuline Form angegeben, ohne eine Einschränkung auf maskuline Personen zu implizieren.

Ebenfalls mit dem Ziel der erhöhten Verständlichkeit wurde davon abgesehen, englische Fachbegriffe durch deutsche zu ersetzen, wenn der englische Begriff mindestens so geläufig ist wie der deutsche. Beispiele hierfür sind *App Store*, *open source* und *cross-platform*. Es wurde bei diesen Begriffen die englische Schreibweise beibehalten. Bei Wortzusammensetzungen mit deutschen Wortteilen wurden die deutschen Regeln zur Wortkomposition angewandt, so zum Beispiel bei den Begriffen *App-Store-fähig* und *Cross-Platform-Entwicklung*.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Abkürzungen

2D zweidimensional

3D dreidimensional

AHP Analytic Hierarchy Process

API Application Programming Interface

ASK Ackerschlagkartei

CSS Cascading Style Sheets

DOM Document Object Model

DVD Digital Versatile Disk

ERP Enterprise Resource Planning

etc et cetera

FMIS Farm Management Information System

fps frames per second

GB Gigabyte (= 1024*1024*1024 Byte)

GPL GNU General Public Licens

GPS Global Positioning System

GWT Google Web Toolkit

ha Hektar (= 10.000 m^2)

HTML Hyper Text Markup Language

JS JavaScript

JSON JavaScript Object Notation

kb Kilobyte (= 1024 Byte)

LED Light-Emitting Diode (Leuchtdiode)

LOC Lines of Code

MB Megabyte (= 1024*1024 Byte)

MIT Massachusetts Institute of Technology

MVC Model View Controller

NDK Native Development Kit

OS Operating System (Betriebssystem)

PC Personal Computer

PDA Personal Digital Assistant

RAM Random-Access-Memory (Arbeitsspeicher)

RIM Research in Motion (Smartphone-Hersteller)

ROC Rank Order Centroid

SDK Software Development Kit

SLOC Source Lines of Code

SMS Short Message Service (Kurznachricht)

UI User Interface

usw. und so weiter

XML eXtensible Markup Language

z.B. zum Beispiel

„Write Once, Run Anywhere."
Werbeslogan für Java (Sun Microsystems [CY99])

1 Einleitung

1.1 Motivation

Durch die zunehmende Verbreitung von Tablets und Smartphones boomt der Markt für mobile Softwareanwendungen (*Apps*). Im Jahr 2011 werden geschätzte 15 Milliarden US-Dollar mit Apps umgesetzt.[1]

Auf zwei von drei derzeit verkauften Smartphones ist Android oder iOS installiert, weswegen diese beiden Plattformen eine zentrale Rolle für die Entwicklung von Apps spielen. Allerdings sind Android und iOS zueinander inkompatibel. Das bedeutet, dass Apps von einer Plattform auf die andere portiert oder sogar komplett neu entwickelt werden müssen, um auf beiden Plattformen laufen zu können. Softwarehersteller sehen sich folglich hohen Entwicklungskosten gegenüber.

Der Smartphonemarkt ist dabei einem äußerst starken Wandel unterworfen. Während beispielsweise das Smartphone-Betriebssystem Symbian im Jahr 2010 noch einen Marktanteil von 36 % aufwies, hat sich dieser Anteil binnen Jahresfrist auf 17 % reduziert und sank damit deutlich stärker als vorhergesagt worden war (vgl. Abschnitt 2.3). Plattformgebundene App-Entwicklung ist also ein riskantes Unterfangen.

Die plattformunabhängige Entwicklung von Apps bietet eine Antwort auf diese beiden Herausforderungen. So können Entwicklungskosten reduziert werden, da nicht für mehrere Plattformen gesondert implementiert werden muss. Gleichzeitig sinkt die Abhängigkeit von einzelnen Plattformen, was das Risiko der Softwarehersteller verringert.

Es stehen zahlreiche Cross-Platform-Technologien für den Mobilbereich zur Verfügung. Allerdings gestaltet sich die Auswahl einer geeigneten Technologie aus zwei Gründen als unerwartet schwierig: Erstens kennt weder die Literatur noch das World Wide Web einen umfassenden Überblick über die verfügbaren Technologien. Eine fundierte Technologieauswahl scheitert also bereits daran, dass unklar ist, welche Technologien überhaupt in Betracht zu ziehen sind. Zweitens gibt es kaum belastbare Informationen über die tatsächliche Funktionsfähigkeit der einzelnen Technologien. Oft stehen Her-

[1] http://www.gartner.com/it/page.jsp?id=1529214

stellerangaben im deutlichen Widerspruch zur tatsächlichen Leistungsfähigkeit eines Produkts.

Für die Auswahl einer Cross-Platform-Technologie steht somit keine solide Entscheidungsgrundlage zur Verfügung. Die vorliegende Arbeit entstand im Rahmen einer industriellen, Prototyp-basierten Technologiestudie und befasst sich mit den beiden dargestellten Wissenslücken.

1.2 Projektumfeld

Das dieser Arbeit zugrundeliegende Projekt entstammt der betrieblichen Praxis. Projektinhaber war die PC-Agrar Informations- und Beratungsdienst GmbH („*PC Agrar GmbH*“) in Pfarrkirchen, ein Hersteller von landwirtschaftlicher Branchensoftware. Das Projektziel bestand in der Sichtung und fundierten Bewertung der verfügbaren Cross-Platform-Technologien für die mobilen Produkte des Unternehmens. Daher ist die zu Bewertungszwecken prototypisch umgesetzte Anwendung in diesem Kontext angesiedelt.

1.3 Einschränkungen

In dieser Arbeit liegt der Fokus auf betrieblichen Datenanwendungen, wie etwa die mobile Stammdaten- und Vorgangsanzeige. Dynamische Anwendungen wie Navigationssoftware oder Spiele sowie hardwarenahe Tools können zwar auch plattformübergreifend entwickelt werden, liegen jedoch außerhalb des vom Projektinhaber vorgegebenen Rahmens.

Von den Endgeräten her beschränkt sich die vorliegende Arbeit auf Smartphones, da Android-Tablets noch nicht hinreichend Verbreitung gefunden haben. Die gewonnenen Ergebnisse lassen sich aber problemlos auf Tablets übertragen.

Des Weiteren erfolgt aus praktischen Gründen eine Konzentration auf die Plattformen Android und iOS. Diese beiden Plattformen decken momentan zwei Drittel des Smartphonemarkts ab und sind daher hinreichend allgemeingültig.

1.4 Überblick über die Arbeit

Ziel der Arbeit ist, Grundlage für eine fundierte Technologieentscheidung bei der Cross-Platform-Entwicklung für Smartphones zu sein. Zunächst werden in Kapitel 2 zahlreiche für die Arbeit notwendige Grundlagen gelegt. In Kapitel 3 wird die wissenschaftliche Fragestellung hergeleitet und der Forschungsansatz erörtert. Das für die Technologieent-

scheidung zwingend notwendige Wissen, welche Technologien überhaupt verfügbar sind, findet sich in Form eines tabellarischen und strukturierten Überblicks in Kapitel 4.

Auf diesem Fundament kann im Folgenden eine Prototyp-basierte Technologiebewertung durchgeführt werden. Kapitel 5 befasst sich mit der Konzeption des Prototypen und wählt auf dieser Grundlage aus den verfügbaren Technologien vier besonders geeignete aus. In Kapitel 6 wird die Implementierung der Prototypen in den jeweiligen Technologien beschrieben.

Kapitel 7 bewertet die Prototypen und damit die ihnen zu Grunde liegenden Technologien mit Hilfe von empirischen Messungen und logischen Herleitungen. Dies bildet die Grundlage für die Diskussion in Kapitel 8, aus der eine konkrete Technologieempfehlung für den vorliegenden Anwendungsfall sowie der Versuch der Übertragung auf allgemeinere Fälle hervorgeht.

Die Arbeit schließt mit einer zusammenfassenden Betrachtung und dem Ausblick auf weitere notwendige Forschung in Kapitel 9.

„Man soll die Dinge so einfach wie möglich machen, aber nicht einfacher."
Albert Einstein (deutscher Physiker, 1879–1955, [Rau04])

2 Grundlagen

Ziel dieses Kapitels ist es, die für das Verständnis dieser Arbeit notwendigen Grundlagen zu erläutern. Gerade bei einer Arbeit, die einen Überblick über zahlreiche Technologien bietet, ist es aus Platzgründen unmöglich, sämtliche Grundlagen zu allen Fachbegriffen und Fachgebieten in erschöpfender Tiefe darzustellen. Daher werden die angesprochenen Themen stark komprimiert und oft auch vereinfacht dargestellt: stets so einfach wie möglich, um den Ausführungen der Arbeit folgen zu können, niemals aber einfache1r oder gar unrichtig.

2.1 Begriffsbestimmung

Im Folgenden werden wichtige Begriffe festgelegt. Dabei wird nicht versucht, allgemeingültige Definitionen zu erarbeiten. Vielmehr ist das Ziel dieses Abschnitts, eine für diese Arbeit gültige und nützliche Terminologie zu etablieren.

2.1.1 Hardware

Im Rahmen dieser Arbeit bezeichnet der Begriff **Rechner** etwas, das im Englischen mit *computing device* ausgedrückt werden kann und im Grunde genommen jede programmierbare Maschine umfasst, insbesondere Desktoprechner und **Mobilgeräte** wie Smartphones oder Tablets.

Der Begriff **Smartphone** ist in der Literatur nicht einheitlich definiert [Mer11]. So wird das Smartphone in [BO11] beispielsweise als Mobiltelefon mit Internetzugang, leistungsstarken Webbrowsern und Multimediafähigkeit umrissen. Für die Unternehmensberatung Gartner ist das Betriebssystem das Hauptmerkmal; hier gilt als Smartphone ein mobiles Kommunikationsgerät, welches ein identifizierbares, offenes Betriebssystem verwendet. Als solche werden von Gartner unter anderem Android und iOS angesehen [Gar11c]. In dieser Arbeit wird jedoch die pragmatischere Definition von Kirk verwendet: Smartphones sind Mobiltelefone mit Internetzugang, auf denen man Softwareanwendungen ausführen kann [Kir11].

Als **Tablets** gelten in Anlehnung an [Gar11c] Geräte, die weniger als 1,5 kg wiegen und durch direkten Bildschirmkontakt per Stift oder Touchscreen bedient werden. Sie sind jedoch, wie bereits weiter oben erwähnt, nicht Fokus der Arbeit.

2.1.2 Cross-Platform

Die zur Verwendung auf Smartphones oder Tablets vorgesehenen Betriebssysteme, z.B. Android oder iOS, werden hier als **Plattform** bezeichnet.

Überraschenderweise ist der Begriff **cross-platform** recht selten Gegenstand formaler Definitionen. So kommt beispielsweise die Monographie „Cross-Platform Development für Smartphones“ komplett ohne Definition des Begriffs aus [Eck10]. Eine der gefundenen Definitionen besagt, Software sei cross-platform, wenn sie auf allen Plattformen läuft [CY99]. In [Won00] wird definiert, dass Anwendungen cross-platform sind, wenn sie auf verschiedenen Plattformen ohne Veränderung funktionieren. Diese letzte Definition deckt sich gut mit dem deutschen Begriff **plattformübergreifend** und wird daher hier verwendet.

Als **Cross-Platform-Technologie** gilt im Rahmen dieser Arbeit jede Technologie, mit Hilfe derer Cross-Platform-Anwendungen programmiert werden können. Diesen Prozess bezeichnet man als **Cross-Platform-Entwicklung**.

2.1.3 Software

Eine **App** ist eine autarke Softwareanwendung, die von Smartphones heruntergeladen und von ihnen ausgeführt werden kann [FT11]. Smartphone-Nutzer verwenden einen so genannten *App Store*, um Apps auf ihr Smartphone herunterzuladen und zu installieren. Während der Begriff App Store zunächst lediglich einen entsprechenden Dienst der Firma Apple[1] bezeichnete, ist er nun auch ein Sammelbegriff für diesbezügliche Angebote anderer Hersteller geworden, wie z.B. Android Market[2], Windows Phone Marketplace[3] und BlackBerry App World[4] [Gar11c]. Der Begriff **App Store** bezeichnet also den vom Hersteller eines Smartphones vorgesehenen Distributionskanal für Apps. Eine App, die per App Store verbreitet werden kann, ist **App-Store-fähig**.

2.1.4 Native Apps und Web Apps

Das Begriffspaar *native App* und *Web App* dient vorrangig der Abgrenzung voneinander. **Native App** bezeichnet dabei eine App, die für eine spezifische Plattform kompiliert

[1] `http://www.apple.com`
[2] `https://market.android.com/`
[3] `http://www.windowsphone.com/de-DE/marketplace`
[4] `http://de.blackberry.com/services/appworld/`

wurde und per App Store auf dem Mobilgerät installiert werden kann. Im Gegensatz dazu steht der Begriff der *Web App*, der im Folgenden hergeleitet wird. Einige ursprünglich aus dem Desktopbereich stammende Definitionen sind weit gefasst: Laut [Hal08] ist eine Webanwendung ein Softwaresystem, das den Nutzern über das Internet zur Verfügung steht. Spezifischer wird der Begriff in [Cho+07] festgelegt: Webanwendungen sind Client-Server-Anwendungen, bei denen ein Webbrowser das User-Interface liefert. [Hae10] definiert pragmatisch: Eine Web App ist schlicht eine Webseite, die für die Darstellung auf Mobilgeräten optimiert ist.

Aus diesen Definitionen ist ersichtlich, dass das Internet im Allgemeinen und der Webbrowser im Speziellen eine zentrale Rolle für Web Apps spielen, aber nicht zum üblichen Zwecke des Surfens eingesetzt werden. Vielmehr soll per Browser eine Softwareanwendung bereitgestellt werden. Der Begriff sei daher wie folgt definiert: Eine **Web App** ist eine Internetseite, die für die Darstellung auf Mobilgeräten optimiert ist und das „Look-and-Feel“ einer nativen App nachbildet.

2.1.5 Programmierung

Abstrakt gesehen kann gesagt werden, dass der Sinn und Zweck der **Programmierung** von Rechnern darin besteht, sie zu einem erwünschten Verhalten zu führen. Analog kann man feststellen, dass der Sinn und Zweck von **Cross-Platform-Programmierung** darin besteht, Rechner verschiedener Plattformen zum gleichen erwünschten Verhalten zu führen.

Dieses erwünschte Verhalten wird zu Beginn des Softwareentwicklungsprozesses typischerweise zunächst als natürlichsprachliches Konzept verfasst, beispielsweise in Form einer Spezifikation oder eines Lasten- bzw. Pflichtenhefts. Im Laufe des Entwicklungsprozesses wird das Konzept sukzessive in einen neuen Text überführt, der in einer formal definierten Programmiersprache verfasst ist, den so genannten **Quellcode** oder Quelltext.

2.2 Programmiersprachen

Programmiersprachen sind Sprachen, in denen Softwareentwickler Befehle für den Rechner verfassen [Wag08]. Die für das weitere Verständnis der Arbeit wichtigen Programmiersprachen werden in komprimierter Form eingeführt. Für eine vertiefende Betrachtung wird jeweils auf entsprechende Literatur verwiesen.

2.2.1 Java

Die objektorientierte Programmiersprache Java wurde von Sun Microsystems[5] Mitte der 1990er Jahre veröffentlicht und auch nach der Übernahme von Sun durch Oracle[6] stetig weiterentwickelt [Sch10]. Java erlaubt die Cross-Platform-Programmierung von Servern, Desktoprechnern und Mobilgeräten. Java-Runtime-Umgebungen sind dabei im Mobilbereich weit verbreitet: Im Jahr 2010 gab es nach Angaben von Oracle drei Milliarden Java-fähige Mobiltelefone [Ora10]. Allerdings ist auf diesen Geräten meist nur *Java Platform, Micro Edition* (*Java ME*, vormals *J2ME*) installiert, welche einen deutlich reduzierten Leistungsumfang gegenüber der regulären *Java 2 Platform, Standard Edition* (*Java SE*, vormals *J2SE*) aufweist. Für vertiefende Informationen zu Java sei auf [SB05], [Sch10] und [Ull11] verwiesen.

2.2.2 JavaScript

Das ebenfalls Mitte der 1990er Jahre von Netscape[7] der Öffentlichkeit vorgestellte JavaScript war integriert in Netscapes Internetbrowser und diente der Manipulation des vom Browser dargestellten HTML-Codes [Koc11b][Wen10]. Die Datenschnittstelle stellte und stellt das so genannte *Document Object Model (DOM)* bereit. JavaScript und DOM-Manipulation werden inzwischen von allen Desktop- und Smartphone-Browsern unterstützt.

JavaScript ist eine interpretierte Sprache, d.h. der Quellcode wird direkt durch eine Komponente des Browsers, die so genannte *JavaScript-Engine*, gelesen und ausgeführt. Bis auf die Syntax der Sprache hat JavaScript mit Java wenig gemein [Ull11], vielmehr bestehen in allen wichtigen Bereichen gravierende Unterschiede [Har97]. Die ursprünglich LiveScript genannte Sprache wurde lediglich aus Marketinggründen in JavaScript umbenannt [Koc11b][Wen10]. Vertiefende Betrachtungen zu JavaScript finden sich beispielsweise in [Koc11b] und [Wen10].

2.2.3 C/C++

1972 von Ritchie und Thomson veröffentlicht [Wol09], ist C immer noch eine der meistgenutzten Programmiersprachen. Das liegt im Wesentlichen daran, dass C und sein objektorientierter und rückwärtskompatibler Abkömmling C++ Allzweckprogrammiersprachen sind [Str00]. Sowohl C als auch C++ sind ANSI-normierte Sprachen, die sich per Compiler in sehr effizienten Maschinencode überführen lassen [Wol09][KP10]. Im Zusammenhang mit plattformübergreifender Smartphone-Programmierung ist relevant,

[5] http://de.wikipedia.org/wiki/Sun_Microsystems
[6] http://www.oracle.com/
[7] http://de.wikipedia.org/wiki/Netscape_Communications_Corporation

dass sich sowohl Android als auch iOS per C/C++ programmieren lassen, worauf in Abschnitt 4.8 näher eingegangen wird.

Es existiert umfassende Literatur zum Thema C/C++, unter anderem [Wol09], [Str00] und [KP10].

2.2.4 Objective-C

Die für Macintosh- und iOS-Programmierung eingesetzte C-Erweiterung *Objective-C* wurde Anfang der 1980er Jahre von Cox und Love entworfen [Kol11]. Sie erbte viele Elemente von C, ist rückwärtskompatibel zu C und – wie der Name nahe legt – stark objektorientiert [Kol11]. Obwohl eine direkte Kompatibilität zu C++ nicht besteht, können bei sorgfältiger Implementierung durchaus C++ und Objective-C parallel verwendet werden [Sta11a].

Eine fundierte Einführung in Objective-C für Macintosh und iOS bietet [Koc11a], während [Kol11] sich auf Objective-C für Smartphones konzentriert.

2.2.5 Flash

Die ursprünglich für animierte Vektorgrafiken im Internet entwickelte Technologie Flash wird inzwischen als Entwicklungsumgebung für interaktive Multimedia-Inhalte aller Art eingesetzt. Flash wurde erstmals 1996 von der kalifornischen Firma Macromedia[8] veröffentlicht. Auch nach der Macromedia-Übernahme durch Adobe[9] im Jahr 2005 wurde es kontinuierlich weiterentwickelt [WG10].

Flash ist eine Cross-Platform-Technologie und läuft auf so gut wie allen Desktop-Betriebssystemen, jedoch nur auf wenigen Smartphones: Während zumindest einige Android-Geräte Flash-fähig sind, unterstützen iOS-Geräte Flash nicht. Ein Grund ist der verhältnismäßig hohe Rechenbedarf von Flash, der die Leistungsfähigkeit der meisten Geräte überfordert oder die Akkulaufzeit über Gebühr verringert [Job10].

Vertiefende Betrachtungen zu Flash finden sich in [GJJ11] und [WG10].

2.3 Der Smartphone-Markt

Während im vorangegangenen Kapitel Programmiersprachen im Fokus standen, gibt dieses Kapitel einen Überblick über die verfügbaren Smartphone-Plattformen (Abschnitt 2.3.1) und ihre gegenwärtige (Abschnitt 2.3.3) und zukünftige (Abschnitt 2.3.4) Verbreitung.

[8] http://de.wikipedia.org/wiki/Macromedia
[9] http://www.adobe.com/

2.3.1 Smartphone-Plattformen im Überblick

Der Überblick über die wichtigsten Smartphone-Plattformen wird in tabellarischer Form gegeben (Tabelle 2.1). Der Aufbau und der überwiegende Teil der Daten entstammen [Der10].

Plattform	iOS[1]	Android[2]	Windows Phone[3]	BlackBerry[4]	Symbian[5]
OS-Hersteller	Apple	Open Headset Allicance	Microsoft	Research in Motion (RIM)	Nokia
Telefon-Hersteller	Apple	Samsung, HTC, andere	Nokia, Samsung, HTC, andere	RIM	Nokia
Programmier-sprache	Objective-C/C++	Java SE	C#[6]	Java SE	Java ME, C++, andere
Markt-einführung	2007	2008	2010	2002	2001

[1] http://www.apple.com/de/ios/
[2] http://www.android.com/
[3] http://symbian.nokia.com/
[4] http://www.microsoft.com/windowsphone/
[5] http://de.blackberry.com/services/blackberry7/
[6] http://msdn.microsoft.com/de-de/windowsphone/hh442446

Tabelle 2.1 – Überblick über verfügbare Smartphone-Plattformen in Anlehnung an [Der10]

2.3.2 Methodik der Marktanteilermittlung

Um die Marktanteile der Plattformen zu ermitteln, stehen prinzipiell zwei offene Informationsquellen zur Verfügung: Web-Zugriffsstatistiken bezüglich mobiler Browser sowie Pressemitteilungen von Marktforschungsinstituten und Unternehmensberatungen.

Erstere sagen über den realen Verbreitungsgrad der Plattformen ansatzbedingt wenig aus, da nur diejenigen Geräte erfasst werden, mit denen auf das Internet zugegriffen wird. Zudem erzeugen Geräte, die häufig zum Surfen verwendet werden, mehr PageViews (Zugriffe) als Geräte, die nur selten zum Surfen verwendet werden. In der Folge sind solche Zahlen zwar für Entwickler von Webseiten relevant, weniger aber für Entwickler von Apps. Diese benötigen primär Informationen über die Anzahl der verfügbaren Geräte und nicht über das Ausmaß der Internetnutzung.

Daher kommt der zweiten genannten Informationsquelle hier mehr Bedeutung zu: Pressemitteilungen von Marktforschungs- und Beratungsunternehmen bezüglich aktueller, historischer und prognostizierter Smartphone-Verkaufszahlen. Diese Mitteilungen dienen dem Zweck, den Produktverkauf der Beratungsunternehmen zu fördern – in

diesem Fall zumeist umfassende Marktstudien, aus denen die Zahlen entnommen sind. Entsprechend enthalten die Publikationen vorrangig Aufmerksamkeit erregende Befunde und selten Informationen über die zu ihrer Ermittlung benutzte Methodik.

Der Erwerb einer umfassenden Marktstudie war vom Projektinhaber kostenbedingt als nicht notwendig eingestuft worden, weswegen die öffentlich zugänglichen Zahlen hier als Ausgangsbasis dienen. Dabei ist zu beachten, dass jeder Anbieter solcher Studien eine andere Methodik verwendet, insbesondere bei Randfällen. So stellt sich beispielsweise die Frage, ob in den Verkaufszahlen von Smartphones die Verkaufszahlen von Tablets, iPods oder PDAs eingeschlossen werden oder nicht. Sinnvoll vergleichen lassen sich die Zahlen nur bei gleichbleibender Systematik, was dazu führt, dass im Folgenden ausschließlich die Angaben eines Anbieters verwendet werden: die recht ergiebigen Verkaufsstatistiken und -prognosen von Gartner[10].

2.3.3 Marktanteile

Zunächst werden die aktuellen Marktanteile der Plattformen untersucht. In Tabelle 2.2 finden sich die am 15. November 2011 veröffentlichten weltweiten Smartphone-Verkaufszahlen nach Plattform aufgeschlüsselt für das dritte Quartal 2011 in Gegenüberstellung mit dem dritten Quartal des Vorjahres (vgl. [Gar11b]).

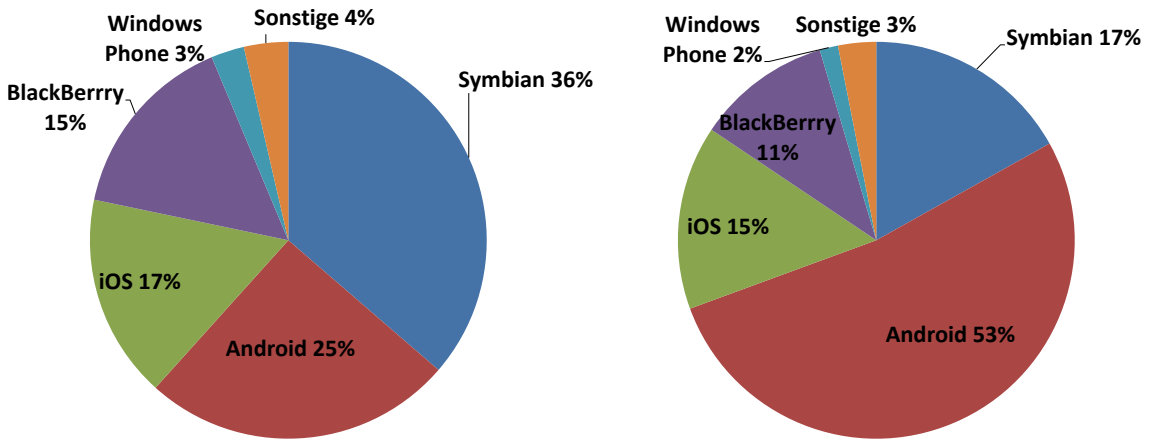
	3. Quartal 2011		3. Quartal 2010		Veränderung	
	Verkaufte Geräte [Mio.]	Marktanteil	Verkaufte Geräte [Mio.]	Marktanteil	Verkaufte Geräte	Marktanteil
Android	60,5	52,5 %	20,5	25,3 %	+195 %	+108 %
Symbian	19,5	16,9 %	29,5	36,3 %	−33 %	−53 %
iOS	17,3	15,0 %	13,5	16,6 %	+28 %	−10 %
BlackBerrry	12,7	11,0 %	12,5	15,4 %	+2 %	−26 %
Windows Phone[1]	1,7	1,5 %	2,2	2,7 %	−19 %	−44 %
Sonstige	3,5	3,1 %	2,9	3,6 %	+21 %	−14 %
Gesamt	115,2	100 %	81,1	100 %	+42 %	0 %

[1] inklusive auslaufendem Vorgänger Windows Mobile

Tabelle 2.2 – Weltweite Smartphone-Verkaufszahlen an Endbenutzer nach Plattform im 3. Quartal 2011 und im 3. Quartal 2010 (vgl. [Gar11b])

In Abbildung 2.1 findet sich als grafische Darstellung obiger Daten eine Gegenüberstellung der Marktanteile aus den Jahren 2011 und 2010 in Form von Tortendiagrammen. Es wird deutlich, dass der Smartphonemarkt sehr volatil ist: Innerhalb eines Jahres halbierte sich der Marktanteil des ehemaligen Marktführers Symbian, während sich der Anteil des vormals zweitplatzierten Android mehr als verdoppelte. Solche sprunghaften

[10] http://www.gartner.com



(a) 2010 **(b)** 2011

Abbildung 2.1 – Weltweite Marktanteile der Smartphone-Plattformen im 3. Quartal 2011 und im 3. Quartal 2010 (Daten aus: [Gar11b])

Änderungen von Marktanteilen sind dem Markt für Mobilgeräte nicht fremd: So verdrängten beispielsweise Mitte der 2000er Jahre PDAs mit Windows Mobile in kurzer Zeit diejenigen des vormaligen Marktführers Palm.

2.3.4 Markttrends

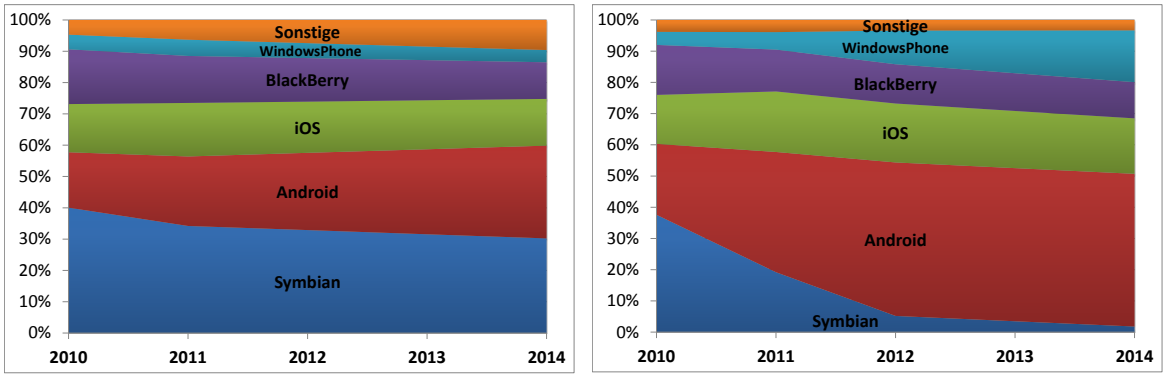
„Prognosen sind schwierig, besonders wenn sie die Zukunft betreffen."
(zugeschrieben u.a. Niels Bohr, Karl Valentin, Winston Churchill)[11]

Auch in Zukunft sind deutliche, unvorhergesehene Marktveränderungen nicht ausgeschlossen. Daher sollten die Marktprognosen nicht überbewertet werden, denn zukünftige Prognosen könnten von der gegenwärtigen signifikant abweichen. Um dies zu verdeutlichen, wurden zwei Prognosen von Gartner bezüglich der Smartphone-Marktanteile ausgewertet. Beide Studien versuchen, den Marktverlauf der jeweils nächsten vier Jahre zu prognostizieren. Eine der Studien wurde 2010 erstellt [Gar10], die andere 2011 [Gar11a]. Abbildung 2.2 visualisiert die Prognosen als Flächendiagramme.[12]

Bei stabilen Prognosen wäre zu erwarten, dass im Jahr 2011 relativ ähnliche zukünftige Marktanteile vorhergesagt werden wie im Jahr 2010. Dies ist offenbar nicht der Fall: Die Prognosen unterscheiden sich deutlich. Exemplarisch wurde die Abweichung der prognostizierten Werte für das Jahr 2012 untersucht. Abbildung 2.3 zeigt die Abweichung der beiden Vorhersagen voneinander in Prozentpunkten.

[11] vgl. `http://www.werhatdasgesagt.de/sonstige-zitate/prognosen-sind-schwierig-besonders-wenn-sie-die-zukunft-betreffen/` sowie `http://de.wikipedia.org/wiki/Prognose#Zitate`

[12] Hinweis zur Methodik: Die Prognose aus dem Jahr 2010 enthält Daten zu den Jahren 2009, 2010, 2011 und 2014; die Prognose aus dem Jahr 2011 enthält Daten zu den Jahren 2010, 2011, 2012 und 2015. Die Überschneidung der beiden Prognosen (Jahre 2010 bis 2014) wird im Diagramm dargestellt. Fehlende Angaben zu einzelnen Jahren wurden linear interpoliert.



(a) Prognose aus dem Jahr 2010 **(b)** Prognose aus dem Jahr 2011

Abbildung 2.2 – Prognosen der Marktentwicklung bei Smartphone-Verkäufen

Es ist ersichtlich, dass die Prognosen von zukünftigen Smartphone-Marktanteilen zu unbeständig sind, um eine stabile Entscheidungsgrundlage darzustellen. Dies bedeutet, dass Planungssicherheit bei der App-Entwicklung für spezielle Smartphone-Plattformen nicht gegeben ist und unterstreicht den Vorteil der Risikoreduzierung durch plattformunabhängige App-Programmierung.

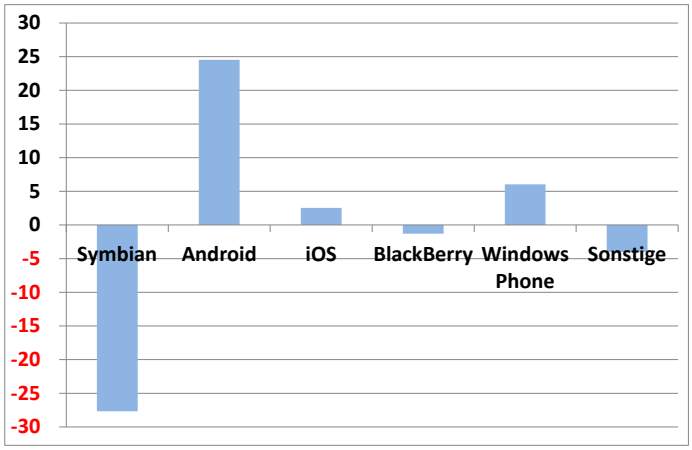


Abbildung 2.3 – Abweichung der für 2012 prognostizierten Marktanteile in Prozentpunkten zwischen der Prognose aus dem Jahr 2010 und der Prognose aus dem Jahr 2011.

Im Falle der PC Agrar GmbH wäre zudem eine plattformspezifische Entwicklung von Apps wirtschaftlich nicht durchzuführen gewesen: Der für eine einzelne Plattform prognostizierte Umsatz durch App-Verkäufe und Wartungsverträge war geringer als die geschätzten Entwicklungs- und Wartungskosten. Erst durch Verwendung einer Cross-Platform-Lösung konnte ein wirtschaftlich rentables Produkt geplant werden.

2.4 Überblick über die Plattformen iOS und Android

Nachdem die verschiedenen, verfügbaren Smartphone-Plattformen vorgestellt wurden, werden in diesem Abschnitt wichtige Eigenschaften der beiden für diese Arbeit zentralen Plattformen iOS und Android erörtert.

2.4.1 Vergleich der Hardware

Zunächst wird auf die Hardware der iOS- und Android-Smartphones eingegangen. Aus der Übersichtstabelle 2.3 geht hervor, dass Android-Geräte deutlich unterschiedlicher sind als iOS-Geräte: Es gibt über 50 verschiedene Android-Modelle, während es lediglich drei aktuelle iOS-Smartphones gibt. Auch sind bei Android deutlich mehr Ausstattungsvarianten vorhanden. Die Vielfalt bei Android-Geräten gegenüber iOS-Geräten verdeutlicht Abbildung 2.4, welche die drei verfügbaren iPhone-Modelle drei populären Android-Geräten gegenüberstellt.[13]

	iOS[1]	**Android** [con11][And11]
Hardware-Hersteller	Apple	HTC, SonyEricsson, Samsung u.a.
Betriebssystem-Hersteller	Apple	Open Handset Alliance[2]
Anzahl aktueller Modelle	3 (iPhone 3GS / 4 / 4S)	> 50
Display-Diagonale	3,5 Zoll	2,5 Zoll bis 5,3 Zoll
Display-Auflösung	480 x 320, 960 x 640	240 x 320 bis 800 x 1280
Bildverhältnis	3:2	Unterschiedlich
GPS	Ja	Meistens

[1] vgl. `http://www.apple.com/de/iphone/compare-iphones/`
[2] `http://www.openhandsetalliance.com/` – Firmenzusammenschluss unter Führung von Google

Tabelle 2.3 – Gegenüberstellung zentraler Hardware-Eigenschaften von iOS- und Android-Smartphones

(a) iOS-Geräte **(b)** Android-Geräte

Abbildung 2.4 – Populäre iOS- und Android-Geräte

[13] Bildquellen: `http://www.apple.com/iphone/compare-iphones/` sowie `http://www.google.com/phone/`

2.4.2 Vergleich des Benutzerkonzepts

Die allgemeine Gestaltung des User Interface und der Nutzerführung von iOS und Android weicht in wesentlichen Punkten voneinander ab, die anhand eines Beispiels besprochen werden. Abbildung 2.5 zeigt exemplarisch den System-Dialog zur Auswahl des aktiven WLAN-Netzwerks auf iOS (links) und Android (rechts).[14]

Abbildung 2.5 – Gegenüberstellung typischer User Interfaces von iOS und Android

Zunächst fällt die unterschiedliche allgemeine Gestaltung (1) auf: Während der iOS-Screen in hellen Weiß-, Grau- und Blau-Tönen gestaltet ist, ist die Grundfarbe des Android-Screens dunkel. Zudem fällt auf, dass die Gestaltung bei iOS runder und weicher gehalten ist als bei Android. Die Titelzeile (2) bei iOS ist größer als bei Android; ebenso die dortige Schrift, was bedeutet, dass die Titelzeile unter Android deutlich längere Texte darstellen kann als unter iOS. Wichtigster Unterschied ist aber der Zurück-Button (3): Während dieser bei iOS immer oben links als Teil der Titelzeile zu sehen ist, sieht Android eine Hardware-Taste für diese Benutzeraktion vor. Deswegen kommen Android-Apps ohne ein entsprechendes User-Interface-Element aus. Andere User-Interface-Elemente hingegen sind auf beiden Plattformen vorhanden, weisen aber typischerweise unterschiedliche Gestaltungen auf (2, 4, 5).

[14] Bildquelle iOS-Screenshot: `http://www.scc.kit.edu/img/net/iOSWLAN8.PNG` am 09.12.2011

2.4.3 Jailbreak und Rooting

Sowohl unter iOS als auch Android gibt es eine Reihe von Operationen, die der Nutzer, bedingt durch die im Betriebssystem hinterlegten Rechte, nicht durchführen darf. Dazu gehören das Entfernen vorinstallierter Programme und das Ersetzen von Hardware-Treibern. Das Umgehen dieser Einschränkungen wird bei iOS als *Jailbreak* und bei Android als *Rooting* bezeichnet. [Lab11][App10b]

Smartphone-Hersteller stehen Jailbreak und Rooting im Allgemeinen kritisch gegenüber: Beispielsweise kündigte Apple an, für Geräte mit Jailbreak die Garantie erlöschen zu lassen, weil durch das Jailbreaking die Sicherheit und Stabilität des Endgerätes gefährdet würden [App10b]. Allein schon deswegen können Jailbreak und Rooting im Rahmen dieser im betrieblichen Kontext angesiedelten Arbeit nicht als zielführend eingestuft werden und finden somit keine weitere Berücksichtigung (siehe auch Abschnitt 3.4).

2.4.4 Plattformspezifische Entwicklung für iOS und Android

Nachdem die beiden Plattformen iOS und Android nun vorgestellt sind, wird in diesem Abschnitt ein Überblick darüber gegeben, wie Apps für diese Plattformen üblicherweise entwickelt werden. Tabelle 2.4 fasst die wichtigsten diesbezüglichen Eckdaten zusammen.

	iOS [Kol11]	**Android** [Kün11]
Betriebssystem basiert auf	Mac OS X	Linux
Entwicklungssprache für native Apps	Objective-C	Java bzw. C/C++
User Interface API	Cocoa Touch API	android.view.*
Entwicklungsumgebung	XCode (nur Macintosh)	Freie Wahl, z.B. Eclipse
Vorinstallierter Browser	Safari (Webkit-basiert)	Android Browser (Webkit-basiert)

Tabelle 2.4 – Gegenüberstellung ausgewählter entwicklungsrelevanter Eigenschaften von iOS und Android

Für Apps, die unter iOS ausgeführt werden sollen, findet die Entwicklung zwingend in Apples Entwicklungsumgebung *XCode*[15] statt, welche nur für Macintosh-Computer zur Verfügung steht. Als Entwicklungssprache kommt Objective-C zum Einsatz. Zudem ist ein iOS-Software-Development-Kit (SDK) notwendig, welches die benötigten Klassen und Werkzeuge für die Entwicklung von iOS-Apps auf dem Entwicklungsrechner bereitstellt. [Kol11]

Android-Apps werden in Java programmiert und können auf allen Plattformen erstellt werden, auf denen Java-Entwicklung möglich ist. Als Entwicklungsumgebung empfiehlt Google die Open-Source-Software Eclipse[16]. Die für die App-Entwicklung notwendigen

[15] `http://developer.apple.com/technologies/tools/`

[16] `http://www.eclipse.org/`

Tools und Klassenbibliotheken werden per SDK bereitgestellt, zudem steht für die Entwicklung von Apps in C/C++ ein so genanntes Native Development Kit (NDK) bereit. [Kün11]

Die Entwicklung für die beiden Plattformen basiert dabei stark auf den Vorgaben der jeweiligen Plattform-SDKs. Beispielsweise werden unter iOS die Bildschirmmasken programmatisch durch Objective-C-Befehle erzeugt, während unter Android ein deskriptives, XML-basiertes Verfahren zum Einsatz kommt (vgl. [Kol11][Kün11]). Selbst wenn man also den Objective-C-Code einer iOS-App nach Java oder C/C++ portieren könnte, wäre diese App unter Android nicht lauffähig, da unter Android andere API-Befehle aufgerufen werden müssen als unter iOS.

2.4.5 Unterschiede der Smartphone- und Desktop-Entwicklung

Zur weiteren Einführung in die App-Entwicklung unter iOS und Android wird verglichen, worin diese sich von der bekannten, traditionellen Desktop-Entwicklung unterscheidet. Der besseren Lesbarkeit wegen werden in diesem Abschnitt Smartphones mit iOS und Android vereinfachend mit dem generellen Begriff *Smartphones* bezeichnet.

Ein offensichtlicher Unterschied zwischen der Anwendungsentwicklung für Smartphones und Desktoprechner besteht in der Größe des Bildschirms: Während Smartphones Bildschirmdiagonalen zwischen 3 und 5 Zoll aufweisen (vgl. Tabelle 2.3), gibt es praktisch keine Laptops mit Bildschirmen unter 10 Zoll.[17]

Zudem müssen Smartphones als Mobilgeräte mit deutlich weniger Hardware-Ressourcen auskommen. Das iPhone 4S hat beispielsweise nur 512 MB Arbeitsspeicher und maximal 64 GB Speicher für Daten und Apps [AL11], während so gut wie alle aktuellen Laptops mit mindestens 4 GB RAM und 240 GB Festplatte ausgestattet sind.[18] Auch verringert jede CPU-Operation die Akku-Laufzeit, weswegen unnötige Berechnungen und Status-Abfragen unbedingt zu vermeiden sind [App11b].

Unterschiedlich ist auch die Art der Benutzereingaben. Die klassischen diesbezüglichen Desktop-Konzepte – Tastatur, Maus und Mausklicks – werden durch Konzepte ersetzt, die der fingerbasierten Touchscreen-Bedienung Rechnung tragen: *Berührungen* und *Gesten*. Berührungen (engl. *taps*) entsprechen dabei den am Desktoprechner üblichen Mausklicks. Bei Gesten wird zudem der zeitliche Verlauf der Berührungen berücksichtigt und interpretiert, beispielsweise als Zoom-Geste oder als Geste zum schnellen Scrollen. Auch sind bei Smartphones mehrere Berührungen zur gleichen Zeit an verschiedenen Orten möglich, was *Multi-Touch* genannt wird. [App11a]

[17] Beim Internethändler notebooksbilliger.de hatten am 11.12.2011 alle verfügbaren Laptops und Netbooks mindestens einen 10,1-Zoll-Bildschirm.

[18] Beim Internethändler notebooksbilliger.de hatten am 11.12.2011 über 94 % der verfügbaren Laptops mindestens 4 GB RAM. 91 % der Laptops besaßen mehr als 240 GB Festplattenspeicher.

Weitere Eingaben können Apps durch Beschleunigungssensoren, GPS-Sensoren und Kompasse erhalten.[19] Bei den Möglichkeiten zur Ausgabe aus Smartphone-Apps hingegen gibt es auf Smartphones typischerweise keinen Drucker. Dafür stehen dort zumeist Generatoren für haptisches Feedback („Vibration“) und LEDs als Blitzlicht oder Taschenlampe zur Verfügung.[20]

2.5 Cross-Platform-Grundlagen

Nachdem in den vorhergehenden Abschnitten die Plattformen iOS und Android vorgestellt wurden, können in diesem Abschnitt die wichtigsten Grundprinzipien der Cross-Platform Entwicklung knapp vorgestellt werden. Das Ziel ist, das notwendige Hintergrundwissen für das Verständnis der in dieser Arbeit untersuchten Cross-Platform-Technologien zu vermitteln.

2.5.1 Compiler, Interpreter und Laufzeitumgebung

Generell muss festgestellt werden, dass Anwendungs-Quellcode nicht unmittelbar von Rechnern ausgeführt werden kann. Der Grund dafür ist, dass Programmiersprachen zwar bereits formale Beschreibungen des erwünschten Verhaltens der Software sind, aber noch nicht in solche Befehle übersetzt wurden, die der Rechner direkt ausführen kann. [Mak09]

Es gibt zwei grundsätzliche Möglichkeiten, Quellcode für den Rechner verständlich und damit ausführbar zu machen: Kompilieren und Interpretieren. Beim **Kompilieren** wird der Quellcode zur Entwicklungszeit durch einen so genannten *Compiler* in plattformspezifischen Maschinencode überführt, der von der Plattform ausgeführt werden kann. Beim **Interpretieren** hingegen wird der Quellcode weitestgehend unverändert auf die Zielplattform überführt und dort durch einen so genannten *Interpreter* ausgeführt. Der Interpreter übersetzt die im Quellcode vorhandene Logik zur Laufzeit in für die Plattform verständliche Befehle. (vgl. [Wag08][Mak09])

Code, der direkt für eine jeweilige Plattform kompiliert wurde, ist im Regelfall deutlich performanter als interpretierter Code, da der Übersetzungsvorgang der in der Programmiersprache abgefassten Logik in Maschinenbefehle einen beträchtlichen Anteil der verfügbaren Rechenleistung in Anspruch nimmt [Pre00]. Vorteil der Interpreter-Lösung hingegen ist, dass – geeignete Interpreter vorausgesetzt – der gleiche Quellcode auf unterschiedlichen Plattformen ausgeführt werden kann.

[19] vgl. `http://developer.apple.com/technologies/ios/features.html`

[20] vgl. `http://developer.apple.com/technologies/ios/accessibility.html` und `http://developer.apple.com/technologies/ios/features.html`

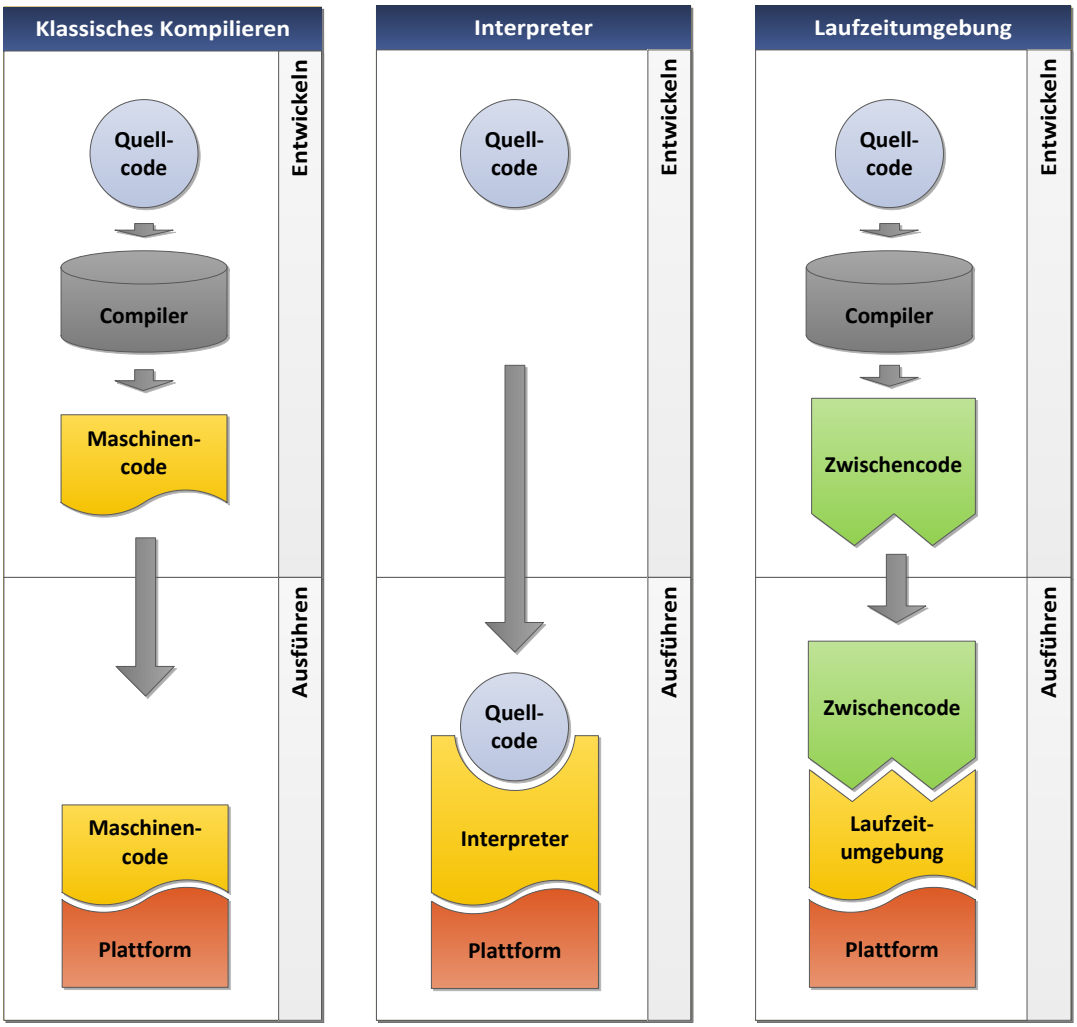


Abbildung 2.6 – Schematischer Vergleich von Compiler, Interpreter und Laufzeitumgebung

Eine Zwischenlösung ist der Einsatz einer so genannten **Laufzeitumgebung**. Hier wird der Quellcode durch einen Compiler in maschinennahen aber plattformunabhängigen *Zwischencode* übersetzt, der zur Laufzeit von einem Interpreter in Plattform-spezifische Befehle umgewandelt wird [Ull11]. Dieser Interpreter wird Laufzeitumgebung genannt. Da der Zwischencode den Maschinenbefehlen deutlich ähnlicher ist als der urspüngliche Quellcode, erreichen Laufzeitumgebungen typischerweise deutlich höhere Performance-Werte als Quellcode-Interpreter [Pre00].

In Abbildung 2.6 werden diese drei Ansätze schematisch gegenübergestellt.

2.5.2 Shared Code Base

Um die gleiche Anwendungs-Funktionalität auf mehreren Smartphone-Plattformen zu erhalten, wird klassischerweise für jede Plattform ein separates Programm entwickelt.

Beispiele für dieses Vorgehen sind die Apps der Deutschen Bahn[21], der Tagesschau[22] und von Twitter[23]. Die erstellten Programme gleichen sich zwar in Funktionalität und im Aussehen, greifen aber nicht auf den gleichen Quellcode zurück.

Abbildung 2.7 visualisiert dieses Vorgehen: zwei separate Quellcode-Sets werden durch zwei separate Compiler in zwei Plattform-spezifische Maschinencodes übersetzt. Diese können schließlich auf den beiden Plattformen ausgeführt werden. Während dieses Vorgehen optimale Laufzeit-Performance gewährleistet, ist es oft nicht wirtschaftlich durchzuführen, wie in Abschnitt 2.3.4 verdeutlicht wurde.

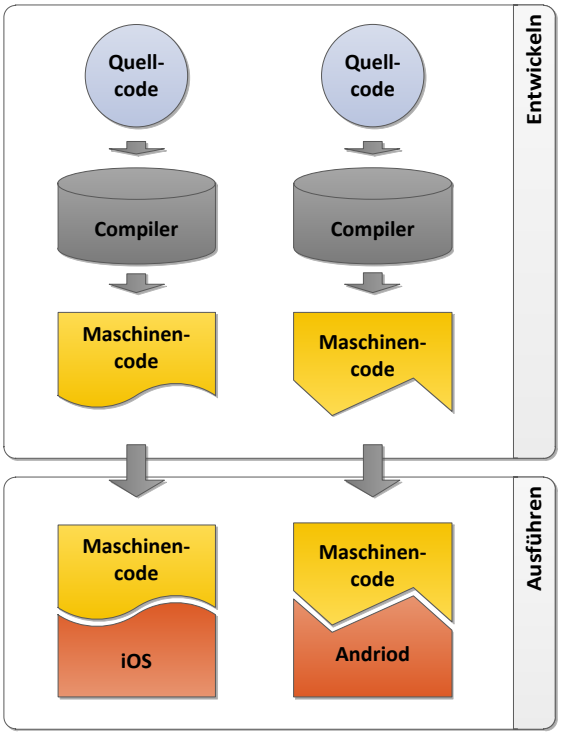


Abbildung 2.7 – Entwicklung von Apps für mehrere Plattformen ohne Shared Code Base

Hauptziel der Cross-Platform-Entwicklung ist daher die Nutzung einer so genannten *Shared Code Base*. Das bedeutet, dass anstelle mehrerer, separater Quellcode-Sets die Apps für alle Plattformen mittels *eines einzigen* Quellcode-Sets erzeugt werden.

Eine Shared Code Base hat eine Reihe wichtiger Vorteile: Zunächst sinkt der Entwicklungsaufwand bei Erstellung und Wartung. Dies führt dazu, dass die zu entwickelnden Anwendungen schneller auf den Markt gebracht werden können. Zudem ist es möglich, mit dem gleichen Entwicklungsaufwand mehr Plattformen abzudecken und damit mehr potenzielle Kunden zu erreichen. Das trägt zur Reduzierung des Fehlschlagrisikos von App-Projekten bei.

[21] `http://www.bahn.de/p/view/buchung/mobil/mobile-apps.shtml`
[22] `http://www.tagesschau.de/app/index.html`
[23] `http://twitter.com/#!/download`

Um die Shared Code Base zu ermöglichen, bedarf es einer Cross-Platform-Technologie. Ihre Aufgabe besteht bildlich gesehen darin, eine Brücke zwischen den Plattformen zu schlagen und die bestehenden Unterschiede in Programmiersprachen, API-Befehlen und Hardware zu überbrücken. In Kapitel 4 werden die verfügbaren Lösungen vorgestellt.

2.6 Grundlagen von Web Apps

Im Prinzip kann der Quelltext einer Internetseite als Quellcode einer Anwendung betrachtet werden und der Webbrowser als der entsprechende Interpreter. Da auf jeder Smartphone-Plattform ein Webbrowser zur Verfügung steht, kommen in dieser Arbeit Webtechniken im Allgemeinen und Web Apps im Speziellen eine wichtige Rolle zu. Die grundsätzlichen Webtechniken wie HTML, CSS und JavaScript sowie die entsprechenden Client-Server-Architekturen werden dabei als bekannt vorausgesetzt und könnten nötigenfalls in [Bal08] nachgelesen werden. In den folgenden Abschnitten werden relevante Fortentwicklungen dieser Standard-Technologien vorgestellt.

2.6.1 HTML5, CSS3 und WebKit-Engine

HTML5 und CSS3 bezeichnen zwei sich im Standardisierungsprozess befindliche neue Webstandards. Es gibt zusätzlich mehrere verwandte Standardisierungsbemühungen, die oft als Bestandteil von HTML5 und CSS3 angesehen werden, obwohl es tatsächlich separate Entwicklungen sind. Alle aktuellen Browser unterstützen diese Standards in einem mehr oder weniger großen Ausmaß. Die meisten der Neuerungen sind für diese Arbeit nicht relevant, weswegen vertiefend auf die Literatur verwiesen sei: [LS11][Gil11][Hog10] [Bos11].

Für die neuen Möglichkeiten der Webseitengestaltung ist hier die Aussage hinreichend, dass durch **CSS3** nützliche Features für Web Apps eingeführt werden, beispielsweise native Farbverläufe und abgerundete Ecken. Dies ist zur glaubwürdigen Imitation von nativen Apps unverzichtbar und wird von der für die Zwecke dieser Arbeit bedeutsamen **WebKit Browser Engine** unterstützt. Diese von Apple entwickelte Open-Source Browser-Engine ist Grundlage der nativen Browser von iOS und Android und liegt auch zahlreichen Desktop-Browsern zu Grunde, unter anderem Safari und Google Chrome [Gil11]. Viele Features aus den neuen Standards HTML5 und CSS3 werden von der WebKit-Engine bereits implementiert, so dass sowohl unter Android als auch unter iOS eine gute Unterstützung dieser Features gewährleistet ist.

Drei für die Zwecke dieser Arbeit bedeutsame Innovationen an **HTML** werden im Folgenden vorgestellt.

2.6.2 Offline Web Applications

Das Feature *Offline Web Applications* ist Bestandteil der entstehenden HTML5-Spezifikation [Hic11a] und bezweckt es, Webapplikationen auch ohne bestehende Internetverbindung verwendbar zu machen. Die Technik verfolgt damit das gleiche Ziel wie Google Gears[24] und Adobe AIR[25] und macht beide im Grunde überflüssig. Google hat in der Folge Gears mittlerweile eingestellt [Fet10].

Damit eine Webseite offline genutzt werden kann, müssen alle für ihre Funktionalität notwendigen Dateien auf dem Endgerät vorliegen. Es müssen also alle notwendigen Ressourcen gecacht werden, wofür der Standard [Hic11a] ein sogenanntes *Manifest* vorsieht, eine Liste aller zu cachenden Dateien. Das Manifest ist eine eigenständige Textdatei, die in der Webseite verlinkt ist und vom Server mit dem Mime-Typ „text/cache-manifest“ bereitgestellt wird. Listing 2.1 enthält als einfaches Beispiel eine HTML-Datei und die zugehörige Manifest-Datei.

```
1  (Datei index.html):
2  <!DOCTYPE html>
3  <html manifest="cache.manifest">
4    <!-- ... -->
5    <img src="img/logo.png"/>
6    <!-- ... -->
7  </html>
8
9  (Datei cache.manifest):
10 CACHE MANIFEST
11 index.html
12 img/logo.png
13 # ggf. weitere Dateien
```

Listing 2.1 – Beispielhafte Anwendung des HTML5-Features *Offline Web Applications*

Der komplexe Caching-Algorithmus aus [Hic11a] ist in einem Entscheidungsdiagramm in Abbildung 2.8 vereinfacht dargestellt. Dem Diagramm lassen sich einige wichtige Beobachtungen entnehmen: Es wird deutlich, dass die Webanwendung stets aus dem Cache geladen wird, d.h. auch im Onlinefall werden auf dem Server veränderte Dateien nicht im Browser berücksichtigt. Das Verhalten der Webanwendung lässt sich folglich nur anpassen, indem man den Browser dazu veranlasst, die Anwendung erneut zu cachen, was nur möglich ist, indem die Manifest-Datei modifiziert wird. In diesem Fall werden *alle* im Manifest aufgeführten Dateien vom Server in den Cache heruntergeladen, unabhängig davon, ob sich einzelne Dateien geändert haben oder nicht.

Dieses Verhalten ist nachteilig für die mobile Nutzung von offline-fähigen Webanwendungen, denn es negiert die dem http-Protokoll inhärente Transfersparsamkeit: Auch wenn nur ein einziges Byte in einer einzigen Datei verändert beim Nutzer angezeigt

[24] http://gears.google.com/

[25] http://www.adobe.com/de/products/air.html

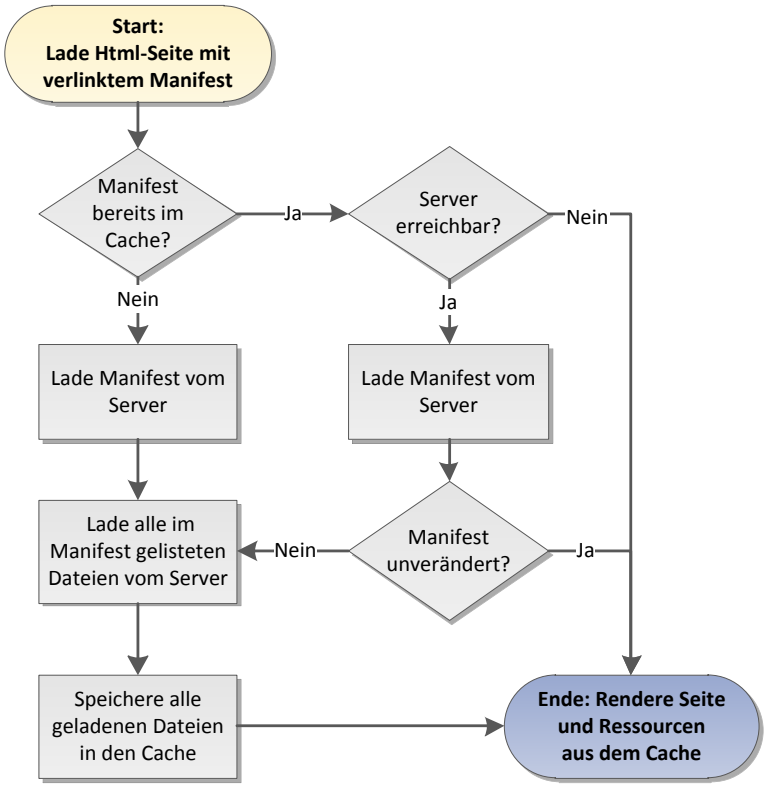


Abbildung 2.8 – Vereinfachtes Entscheidungsdiagramm des Caching-Algorithmus von HTML5 *Offline Web Applications*

werden soll, müssen sämtliche Ressourcen, inklusive aller Bilder, JavaScript-Libraries und Unterseiten erneut über die potenziell schmalbandige Mobilfunkverbindung geladen werden.

Relativiert wird dieses Problem durch eine andere gegenwärtige Einschränkung: Die maximale Größe des Caches ist nicht im Standard vorgegeben und variiert von Browser zu Browser, wobei keine einheitliche Dokumentation der Begrenzungen existiert. Informellen Quellen entstammt die Information von 5 MB als momentan übliche Obergrenze; auf dem iPhone offenbar sogar, ohne dass der Nutzer der Anwendung mehr Speicherplatz einräumen könnte [Sta10a].

2.6.3 Web-Storage-API

Die mittlerweile aus dem HTML5-Standard herausgelöste und separat weiterentwickelte Web-Storage-API hat mehrere Ziele; eines davon ist es, Webanwendungen eine persistente, lokale Speichermöglichkeit von Informationen zu gewähren, um beispielsweise Nutzdaten dauerhaft lokal abrufbar zu halten [Hic11b]. Insbesondere im Zusammenhang mit dem HTML5-Feature *Offline Web Applications* ist die Web-Storage-API hilfreich, denn individuelle Nutzereinstellungen, die traditionell am Server gespeichert würden, können hiermit lokal per JavaScript gespeichert und gelesen werden.

Die Web-Storage-API ist ein lokaler, persistenter Speicher, der über JavaScript die Möglichkeit zur Speicherung von String-Tupeln der Form *Schlüssel* $\mapsto$ *Wert* bietet [Hic11b]. Listing 2.2 illustriert anhand eines einfachen Beispiels die Funktionsweise der API, ohne jedoch alle Möglichkeiten abzudecken.

```
1 <script>
2 if (!localStorage.loadCount)
3    localStorage.loadCount = 0;
4 localStorage.loadCount = parseInt(localStorage.loadCount) + 1;
5 document.getElementById('count').textContent = localStorage.loadCount;
6 </script>
```

Listing 2.2 – Anwendungsbeispiel der Web-Storage-API [Hic11b]

Vertiefend sei auf [LS11] hingewiesen sowie auf zwei hier nicht weiter betrachtete entstehende Standards für lokale Datenhaltung per Datenbank, namentlich *Web SQL Database* [Hic10] sowie *Indexed Database API* [Meh+11]. Diese finden in den hier vorgestellten Cross-Platform-Technologien keinen Einsatz und werden somit an dieser Stelle nicht weiter erörtert.

2.6.4 Gleichzeitige Nutzung fixierter und scrollender Elemente

Bei Web Apps ist es häufig wünschenswert, bestimmte Bildschirminhalte stets sichtbar zu halten (z.B. eine Toolbar oder die Titelleiste) und andere Bildschirminhalte scrollbar zu gestalten (beispielsweise eine Liste mit Stammdaten). Prinzipiell sieht CSS für nicht scrollbare Elemente die Eigenschaft `position:fixed` vor: „fixed boxes do not move when the document is scrolled“ [Bos+11]. Jedoch sind alle gängigen Mobilbrowser, insbesondere auch alle WebKit-basierten Mobilbrowser, abweichend implementiert, so dass solcherart ausgezeichnete Elemente mit der restlichen Seite mitscrollen.

Es gibt zahlreiche Versuche, dieses Problem zufriedenstellend zu lösen [CL11], von denen einige in Abschnitt 4.4 vorgestellt werden. Der Einsatz von Frames – eine traditionelle Technik zur Aufteilung einer Webseite in mehrere unabhängige Bereiche – scheitert auf mobilen Browsern an der Tatsache, dass sich die Browser nicht an die angegebenen Aufteilungen halten. Die erwähnten Versuche zielen daher darauf ab, dem Browser einen nicht-scrollbaren Content „vorzugaukeln“, per JavaScript die durch Scroll-Gesten entstehenden MouseClick-Events abzufangen und das Scrollen per JavaScript zu emulieren. Die Performance dieser Lösung ist jedoch oft nicht zufriedenstellend [CL11], wie auch die Auswertung der empirischen Messungen in Abschnitt 7.8 belegt.

Nachdem in diesem Kapitel alle notwendigen Grundlagen für das Verständnis dieser Arbeit gelegt wurden, können im nächsten Kapitel ihre wissenschaftliche Fragestellung und ihr Forschungsansatz erörtert werden.

„Die Kritik an anderen hat noch keinem die eigene Leistung erspart."
Noël Coward (1899-1973, englischer Künstler)[1]

3 Wissenschaftliche Fragestellung und Forschungsansatz

Aufgabe dieser Arbeit ist es, für die konkrete Aufgabenstellung der PC Agrar GmbH die am besten geeignete Cross-Platform-Technologie zu ermitteln. Im ersten Schritt wird untersucht, ob sich diese Frage mit der bestehenden Literatur beantworten lässt.

3.1 Stand der Forschung

Es wurde daher nach dedizierter Literatur in Journals und Fachzeitungen gesucht. Unter den Suchbegriffen „[choosing|evaluation|comparison] cross platform [technologies|frameworks]" konnten im Juli 2011 in den Datenbanken ACM, SpringerLink, IEEE, Elesevier, ScienceDirect und Google Scholar keine relevanten Zeitschriftenartikel ausfindig gemacht werden. Die Suche brachte jedoch vier Monographien zu diesem Thema zu Tage sowie einen sehr fundierten Aufsatz zu einem ähnlichen Thema. Diese werden im Folgenden besprochen.

3.1.1 Monographien

Allen vier gefundenen Fachbüchern ist gemein, dass sie jeweils eine kleinere Auswahl der verfügbaren Technologien besprechen, ohne auf die Existenz weiterer Technologien hinzuweisen oder zu erklären, warum diese nicht besprochen wurden.

In der **Bachelor-Arbeit von Paananen** wird nach der am besten geeigneten Cross-Platform-Technologie für ein finnisches Medienunternehmen geforscht [Paa11]. Fünf Technologien werden untersucht, und in jeder Technologie wird eine Beispielanwendung umgesetzt. Die anschließende Bewertung ermittelt einen Punktewert je Technologie und muss aus mehreren Gründen kritisiert werden: Erstens spielen die Beispielanwendungen bei der Bewertung keine Rolle, weswegen sich die Frage stellt, wozu überhaupt

[1] http://www.zitate-online.de/sprueche/allgemein/18285/die-kritik-an-anderen-hat-noch-keinem-die.html und http://en.wikipedia.org/wiki/No%C3%ABl_Coward

implementiert wurde. Zweitens ist nicht klar, warum die verwendeten Bewertungskriterien benutzt werden und nicht andere. Drittens ist nicht immer nachvollziehbar, wie von den geschilderten Beobachtungen auf Bewertungspunkte geschlossen wurde. Und viertens erfolgt die Addition der Bewertungspunkte ungewichtet, so dass jedem Kriterium die gleiche Bedeutung beigemessen wird, was angesichts stark unterschiedlicher Granularität und Bedeutsamkeit der Kriterien (z.B. „Unterstützt das iPhone“, „kann Audio abspielen“, „veröffentlicht unter der MIT Lizenz“) nicht sinnvoll erscheint.

In der **Diplomarbeit von Eckl** wird im Rahmen einer Fallstudie für den Lebensmittel-Einzelhandel eine Cross-Platform-Smartphone-App entwickelt [Eck10]. Sieben Cross-Platform-Technologien werden vorgestellt, von denen zwei ohne weitere Begründung als „Favoriten“ benannt werden. Die Anwendung wird in einer der beiden Technologien umgesetzt, da der andere „Favorit“ zum damaligen Zeitpunkt ein benötigtes Feature nicht unterstützte. Unerwähnt bleibt, dass auch mehrere Nicht-Favoriten dieses Feature unterstützt hätten.

Im Buch **Pro Smartphone Cross-Platform Development** von Allen et al. werden sechs Cross-Platform-Technologien weitgehend unzusammenhängend vorgestellt [AGL10]. Die Techniken werden nicht evaluiert und auch nicht verglichen; das Buch befasst sich allein mit ihrer Verwendung.

Das Werk **HTML5-Apps für iPhone und Android** stellt vier Cross-Platform-Technologien vor [HS11]. Auch hier findet keine kritische Auseinandersetzung mit den Technologien statt und auch hier bleiben zahlreiche weitere Technologien unerwähnt.

Insgesamt muss somit festgestellt werden, dass die verfügbare Literatur die Fragestellung des Projektinhabers nicht angemessen beantworten kann, da in keinem Werk alle Technologien benannt, geschweige denn solide untersucht werden. Auf dieser Basis ist keine hinreichend begründete Technologieentscheidung zu treffen. Es müssen folglich die in Frage kommenden Technologien recherchiert und evaluiert werden, um die Aufgabenstellung der PC Agrar GmbH zu beantworten.

3.1.2 Verwandte Themengebiete

In der breiteren Recherche wurde eine sehr fundierte Second-Level-Studie zur Evaluierung und Auswahl von Softwarepaketen (z.B. Office-Pakete oder ERP-Software) gefunden [JS09]. Dort wurden 130 Beiträge zum Thema gesichtet und 60 Beiträge für relevant befunden und genauer untersucht. Die Ergebnisse der Studie lassen sich teilweise auf die vorliegende Aufgabenstellung übertragen.

Zunächst wird der Auswahlprozess betrachtet. Die Autoren leiten aus den untersuchten Studien einen generischen Prozess für die Auswahl von Softwarepaketen ab (vgl. [JS09]):

1. Investitionsbedarf feststellen und Vorabuntersuchung über verfügbare Softwarepakete durchführen
2. Kurzaufstellung der Auswahl-Kandidaten erstellen
3. Ausschluss der meisten Kandidaten, beispielsweise wegen Inkompatibilität zu bestehender Infrastruktur, fehlender Features oder Ähnlichem
4. Verbleibende Kandidaten durch eine Evaluierungsmethode bewerten
5. Weiteres Lichten der Liste durch empirische Evaluierung und Pilot-Installationen
6. Vertrag aushandeln
7. Kauf abschließen, Software einführen

Es ist zu beachten, dass keine Studie sämtliche Schritte dieses Auswahlprozesses umsetzt, insbesondere werden die beiden Schritte 4 (Evaluierung) und 5 (Pilot-Installationen) in keiner Studie zusammen verwendet, sondern immer nur wahlweise.

Auf diesem Auswahlprozess baut die vorliegende Arbeit auf – mit einigen im Folgenden beschriebenen Anpassungen.

3.2 Auswahl von Cross-Platform-Technologien

Der Auswahlprozess von [JS09] basiert auf einer Kurzaufstellung der in Frage kommenden Alternativen. Eine solche Aufstellung muss bei der Auswahl von Cross-Platform-Technologien in Anbetracht nicht vorhandener Erfahrungsberichte in der Literatur auf Herstellerangaben und weiteren, im Internet verfügbaren Informationsquellen wie Blog-Einträgen oder Wikipedia-Artikeln beruhen. Zahlreiche solcher Quellen wurden gesichtet [Anu11][ODe10][iph09][Sta10b][Wik11a][Wik11b][Woo10] und hinsichtlich der dort besprochenen Technologien ausgewertet. Technologien, die den Angaben gemäß iOS und Android unterstützen, und solche, bei denen dies zu vermuten war, wurden daraufhin auf den Webseiten der jeweiligen Hersteller detailliert recherchiert.

Dabei wurde mehrmals festgestellt, dass die Quellen kein verlässliches Fundament für Entscheidungen darstellen: Angaben in Blogs waren teils unrichtig; von Herstellern beschriebene Features funktionierten mitunter nicht, noch nicht oder nicht mehr. So warb beispielsweise der Hersteller Rhomobile[1] bis Mitte 2011 für sein Produkt Rhodes mit der Angabe, es funktioniere unter „iPhone, Windows Mobile, RIM, *Symbian* and Android“.[2] In einer vertiefenden Recherche stellte sich heraus, dass die Unterstützung für Symbian mangels Nachfrage bereits 2009 eingestellt worden war.[3] Daraus wird ersichtlich, dass für

[1] `http://rhomobile.com/`

[2] `http://web.archive.org/web/20101127020423/http://rhomobile.com/products/rhodes/` – Hervorhebung durch Verfasser

[3] `http://groups.google.com/group/rhomobile/browse_thread/thread/ec0f5db173c046bc/dae62e322be7cd87?lnk=gst&q=symbian+#dae62e322be7cd87` in Verbindung mit `http://groups.google.com/group/rhomobile/browse_thread/thread/bfdf4055f208cafc/d1ac9eabd9deda58?lnk=gst&q=symbian+1.*#d1ac9eabd9deda58`

eine aussagekräftige Kurzaufstellung eine tief gehende Recherche notwendig ist. Um den Rechercheaufwand im vertretbaren Rahmen zu halten, wurden Mindestanforderungen an die zu untersuchenden Technologien gestellt (vgl. Abschnitt 3.4).

Eine Literatur- und Internet-Recherche allein kann jedoch die Frage nach der tatsächlichen Funktionsfähigkeit und Güte einer Technologie nicht beantworten. Im Rahmen dieser Arbeit wird daher zum Mittel des Prototypen gegriffen. Eine solche beispielhafte Implementierung der Zielanwendung wird in mehreren Technologien durchgeführt und entspricht dem Punkt 5 in der Liste von [JS09] („Pilot-Anwendungen“). Aus Gründen des Aufwandes wurde dies nicht für alle in der Kurzaufstellung enthaltenen Technologien durchgeführt, so dass analog zu [JS09] eine Vorauswahl erfolgte. Der Prozess der Technologieauswahl und -bewertung in dieser Arbeit ist somit folgender:

1. Mindestanforderungen für Technologien festlegen (Abschnitt 3.4)
2. Technologieüberblick erstellen (Kapitel 4)
3. Prototyp definieren und die zu untersuchenden Technologien anhand seiner Anforderungen auswählen (Kapitel 5)
4. Prototyp in den verschiedenen Technologien implementieren (Kapitel 6)
5. Technologien anhand der Prototypen mittels einer Evaluierungsmethode untersuchen (Kapitel 7)
6. Ergebnisse der Evaluierung prüfen und Konsequenzen ermitteln (Kapitel 8)

3.3 Wissenschaftliche Fragestellung

Aus diesem Prozess lassen sich folgende Forschungsfragen ableiten:

Forschungsfrage 1 *Welche Technologien zur plattformübergreifenden Programmierung für iOS und Android gibt es und welche Eigenschaften haben sie?*

Forschungsfrage 2 *Welche Cross-Platform-Technologie ist für das geplante mobile Software-Produkt der PC Agrar GmbH am besten geeignet?*

In Kapitel 4 wird Forschungsfrage 1 beantwortet, während die Kapitel 5 bis 8 der Beantwortung von Forschungsfrage 2 dienen. Zunächst sind aber einige weitere Teilfragen zu klären. Da diese nicht im Zentrum der Untersuchung stehen, werden sie als Nebenfragen bezeichnet und bündiger als die Forschungsfragen beantwortet:

Nebenfrage 1 *Welches sind die Mindestanforderungen an Cross-Platform-Technologien im Rahmen dieser Arbeit?*

Nebenfrage 2 *Anhand welcher Kriterien kann man Cross-Platform-Technologien bewerten?*

Nebenfrage 3 *Welche Evaluierungsmethode ist für die Auswahl einer Cross-Platform-Technologie im Rahmen dieser Arbeit geeignet?*

Diese Nebenfragen werden nun nacheinander untersucht.

3.4 Mindestanforderungen

Zunächst muss geklärt werden, welche Mindestanforderungen an Cross-Platform-Technologien zu stellen sind, die für die PC Agrar GmbH eingesetzt werden sollen. Diese Anforderungen dürfen keine geeigneten Technologien vorzeitig verwerfen, sollen aber den Rechercheaufwand durch Ausschluss irrelevanter Technologien reduzieren.

Bei den zu unterstützenden Smartphone-Plattformen wird daher mindestens **iOS und Android** vorausgesetzt, da dies nach Auffassung des Projektinhabers – und auch in der Wahrnehmung der Fachpresse – die beiden momentan wichtigsten Smartphone-Plattformen sind.

Jede relevante Cross-Platform-Technologie muss im Sinne einer möglichst langen Nutzungsdauer der zu entwickelnden Apps **vom Technologiehersteller aktiv weiterentwickelt** werden, weswegen nur solche Technologien berücksichtigt werden, an denen innerhalb der letzten drei Monate erkennbar gearbeitet wurde, z.B. durch Veröffentlichungen auf der Webseite oder durch Commits in ein eventuelles Open-Source-Repository.

Zudem ist es notwendig, dass die Technologie **für vollwertige Apps geeignet** ist, d.h. es muss im Gegensatz zu „Baukasten"-Lösungen beliebige Business-Logik ausführbar sein. Auch werden keine Lösungen berücksichtigt, die nur für bestimmte Branchen vorgesehen sind.

Weiterhin darf die Technologie **kein Jailbreak oder Rooting** des Endgeräts voraussetzen, da dies aufgrund des geschäftlichen Einsatzes der Apps einen unseriösen Eindruck bei potenziellen Kunden hinterlassen könnte (vgl. Abschnitt 2.4.3).

Technologien, die diesen Erfordernissen entsprechen, werden in Kapitel 4 detailliert vorgestellt. Technologien, die während der Recherche geprüft wurden und den Mindestanforderungen nicht entsprechen, sind in Anhang A aufgeführt. Selbiges gilt auch für Technologien, die erst nach Abschluss der Prototyp-Implementierungen bei einer Nachrecherche entdeckt wurden.

3.5 Bewertungskriterien

Für eine fundierte Bewertung der Technologien stellt sich auch die Frage nach den Kriterien, anhand derer die Technologie beurteilt werden soll. Auch nach intensiver Recherche konnte in der Literatur kein Katalog von Bewertungskriterien für Cross-Platform-Technologien ausfindig gemacht werden. Ansatzpunkt für die Erarbeitung

eines solchen Katalogs war daher die umfassende Kriterienliste der Second-Level-Studie [JS09] bezüglich der Auswahl von Softwarepaketen.

Die Kriterien dieser Liste wurden – soweit möglich – auf die gegebene Aufgabenstellung übertragen und durch weitere Kriterien aus anderen Quellen ergänzt. Dabei wurde darauf geachtet, die Granularität der Kriterien auf einem etwa gleichen Niveau zu halten. Dafür wurden teils mehrere Kriterien zusammengefasst. Beispielsweise listet [Paa11] 21 einzelne Geräte-Features – Bluetooth, SMS, Klingeltöne, etc. – auf, die unter dem Punkt „Native Gerätefunktionen“ als ein einzelnes Kriterium in den Katalog eingehen.

Der erarbeitete Katalog orientiert sich an den Bedürfnissen von Apps für den geschäftlichen Einsatz (*Business-Apps*), weswegen viele nur für Spiele relevante Punkte wie „OpenGL-Unterstützung“ oder „Physik-Engine“ nicht aufgeführt werden. Gleiches gilt auch für Kriterien, die keine Auswahlrelevanz zu besitzen scheinen, wie die Frage, welches Speichermanagement eine Technologie verwendet.

Einige der aufgelisteten Kriterien ohne Quellenangaben wurden im Laufe der Erstellung dieser Arbeit für wichtig befunden und in den Katalog mit aufgenommen. So untersuchen zwar viele Quellen, welche Smartphone-Plattformen *momentan* unterstützt werden, aber keine Quelle fragt, welche Plattformen voraussichtlich *in Zukunft* unterstützt werden, obwohl die Entwicklung einer App offenbar eine Investition in *zukünftig* zu erzielenden Umsatz ist.

Trotzdem sollte nicht vermutet werden, dass Kriterien ohne Quellenangabe eine originäre Leistung des Verfassers dieser Arbeit darstellen, denn sie sind zum Großteil recht nahe liegend. Vielmehr steht zu vermuten, dass alle Kriterien bereits von einem Kollegen irgendwann irgendwo gedacht und formuliert wurden – auch wenn kein expliziter Quellennachweis ausfindig gemacht werden konnte. Die Leistung des Autors liegt eher in der Strukturierung und Bündelung der zahlreichen vorliegenden Kriterien in den vorliegenden Katalog.

Der durch die Recherche entstandene Kriterienkatalog findet sich in Tabelle 3.1. Dort werden die Kriterien gruppiert aufgelistet und – angeregt von [JS09] – für jedes Kriterium eine beschreibende Frage angeführt, die bei der Evaluierung angewendet werden kann.

Kriterium	**Beschreibung**
Plattformen	
Unterstützung aktueller Smartphone-Plattformen	Welche Smartphone-Plattformen werden unterstützt? (vgl. u.a. [Eck10][GE11][JS09])
Unterstützung zukünftiger Smartphone-Plattformen	Wie gut ist die Technologie gerüstet, um zukünftige Smartphone-Plattformen zu unterstützen?

(Fortsetzung auf nächster Seite)

(Forts.) **Kriterium**	**Beschreibung**
Mehraufwand für zusätzliche Plattformen	Wie hoch ist der Aufwand, um mehrere Plattformen zu unterstützen?
Portierbarkeit als Webanwendung	Wie leicht lassen sich Zielanwendungen als Webanwendung portieren?
Technologie	
Entwicklungsstadium	In welchem Entwicklungsstadium befindet sich die Technologie (z.B. Hobby-Projekt, Beta-Version)? (vgl. [Eck10][ODe10])
Aktiv	Wird die Technologie aktiv weiter entwickelt? (vgl. [Eck10][ODe10])
Kontinuität	Wie wahrscheinlich ist es, dass die Technologie vom Hersteller weiter gepflegt und entwickelt wird?
Verbreitung	Wie verbreitet ist die Technologie? (vgl. [GE11][JS09][MC09])
Auszeichnungen	Ist die Technologie mit relevanten Preisen ausgezeichnet worden? (vgl. [ODe10])
Lizenz	Unter welchen Lizenzbedingungen kann die Technologie eingesetzt werden? (vgl. [Paa11][Wik11a])
Open Source	Ist der Quellcode der Technologie frei verfügbar? (vgl. [JS09][ODe10])
Einsatzmöglichkeiten	
Domänenspektrum	Wie breit ist das Spektrum möglicher Zielanwendungen in Bezug auf verschiedene Branchen und Geschäftsfelder? (vgl. [JS09])
Kommerziell verwendbar	Gibt es Einschränkungen für die kommerzielle Verwendung?
Vollwertige Entwicklung	Ist beliebiger Code ausführbar oder sind nur vorgegebene Bausteine kombinierbar?
Skalierbarkeit	Wie gut ist die Technologie für umfangreiche Anwendungen geeignet?
Plattform-Integration	
Native Gerätefunktionen	Auf welche nativen Gerätefunktionen haben Zielanwendungen Zugriff (z.B. Kompass, Telefonbuch, usw.)? (vgl. u.a. [Anu11][GE11][MC09])
Native Erweiterbarkeit	Wie gut kann die Technologie um selbsterstellte, native Module erweitert werden? (vgl. [Paa11])
Lokale Datenbank	Steht eine lokale Datenbank zur Verfügung und wie leistungsstark ist sie? (vgl. [GE11][MC09])
Parser für JSON/XML	Steht der Zielanwendung ein Parser für Daten aus webtypischen Datenformaten zur Verfügung? (vgl. [JS09])
Social Features	Wie einfach lassen sich soziale Netzwerke wie Twitter und Facebook in die Zielanwendungen integrieren? (vgl. [ODe10])
Multimedia-Fähigkeiten	Wie gut lassen sich 2D- und 3D-Grafiken sowie Sound und Videos verwenden? (vgl. [GE11][MC09][ODe10])
Server-Interaktion	Wie einfach ist es, auf Server-Backends zuzugreifen? (vgl. [GE11][MC09])

(Fortsetzung auf nächster Seite)

(Forts.) **Kriterium**	**Beschreibung**
Verwendungsanalyse	Lässt sich ermitteln, wie die Anwendung genutzt wird (z.B. Nutzungshäufigkeit, verwendete Features)? (vgl. [Anu11])
Offline-Fähigkeit	Kann die Zielanwendung auch offline eingesetzt werden?
App-Store-Fähigkeit	Kann die Zielanwendung über den App Store vertrieben werden? (vgl. [MC09][Wik11a])
Jailbreak-freie Installation	Ist ein *Jailbreak* oder *Rooting* der Endgeräte erforderlich, um die Zielanwendungen zu verwenden? (vgl. Abschnitt 2.4.3)
User Interface und Interaktion	
Allgemeine Bedienbarkeit	Wie gut lassen sich die Zielanwendungen bedienen? (vgl. [JS09])
Aussehen des User Interface	Wie ansehnlich ist das User Interface der Zielanwendung? Gibt es Darstellungsfehler?
Datenvisualisierung	Wie effektiv lassen sich Daten darstellen und visualisieren? (vgl. u.a. [GE11][JS09][MC09])
Animationen	Werden gerätetypische Animationen unterstützt?
Multi-Touch	Gibt es Multi-Touch-Unterstützung?
Gesten	Werden Gesten unterstützt (z.B. Wischen, Streichen)?
Einarbeitung und Dokumentation	
Einfach zu erlernen	Wie leicht lässt sich die Nutzung der Technologie erlernen? (vgl. [JS09][GE11])
Vorhandenes Know-How	Gibt es im Unternehmen bereits Know-How zur Technologie oder ihren Grundlagen? (vgl. [Anu11][Eck10][Woo10])
Programmiersprache(n)	In welchen Programmiersprachen werden Anwendungen entwickelt? (vgl. u.a. [Anu11][GE11][JS09])
Dokumentation	Wie gut ist die Technologie dokumentiert? Ist die Dokumentation aktuell, zutreffend, umfassend, verständlich, mit Beispielen und/oder Tutorials? (vgl. u.a. [GE11][JS09][MC09])
Community	Wie groß, aktiv und hilfsbereit ist die Entwickler-Community der Technologie? (vgl. [Eck10][MC09])
Entwicklungsprozess	
Entwicklungszeit	Wie schnell lassen sich Anwendungen mit der Technologie umsetzen? (vgl. [Anu11][Paa11])
Entwicklungsumgebung (IDE)	Steht eine dedizierte Entwicklungsumgebung zur Verfügung? (vgl. [GE11][Wik11a])
Entwicklungs-Hardware	Welche Hardware wird zur Entwicklung benötigt? (vgl. [JS09][Sta10b])
Emulator	Stellt die Technologie einen Emulator bereit? (vgl. [Eck10][Wik11a])
Build-Service	Gibt es einen Online-Service zum Kompilieren der Apps? (vgl. [Wik11a])
Debugging	Wie effektiv lassen sich Zielanwendungen debuggen? (vgl. [GE11][Wik11a])

(Fortsetzung auf nächster Seite)

(Forts.) **Kriterium**	**Beschreibung**
Logging	Gibt es Logging-Unterstützung? (vgl. [JS09])
Quellcode der Zielanwendungen	
Quellcode-Umfang	Wie viel Quellcode ist nötig, um die Zielanwendung umzusetzen? (vgl. [GE11])
Shared Code Base	Können Anwendungen für alle Plattformen mit einem gemeinsamen Quellcode erstellt werden?
Separation of Concerns	Ist eine Separation of Concerns vorgesehen (z.B. MVC)?
Technische Kriterien (vgl. [GE11])	
Anwendungsperformance	Wie performant laufen die Zielanwendungen? (vgl. u.a. [GE11][JS09][MC09])
Stabilität	Wie stabil laufen die Zielanwendungen? (vgl. [JS09])
Akkuverbrauch	Wie viel Akkuleistung verbrauchen die Zielanwendungen? (vgl. [GE11])
Installergröße	Wie viel Speicherplatz benötigen die Installationspakete der Zielanwendungen? (vgl. [GE11])
Arbeitsspeicherbedarf	Wie viel Arbeitsspeicher wird bei der Ausführung der Zielanwendungen belegt? (vgl. [GE11])
Hersteller (vgl. [JS09])	
Expertise des Herstellers	Wie kompetent und erfahren ist der Technologiehersteller? (vgl. [JS09])
Referenzen	Welche internen und externen Referenzen und Erfahrungen bestehen zum Hersteller? (vgl. [JS09])
Hersteller-Support	Bietet der Hersteller technischen Support, Beratung, Trainingskurse an? (vgl. [Eck10][GE11][JS09])
Zusätzliche Dienstleistungen	Welche zusätzlichen relevanten Dienstleistungen stehen zur Verfügung? (vgl. [Anu11][ODe10])
Kosten (vgl. u.a. [Anu11][GE11][JS09])	
Direkte Kosten	Welche unmittelbaren Kosten entstehen durch den Einsatz der Technologie? (Lizenzen/Hardware/...) (vgl. [GE11][JS09][Wik11a])
Folgekosten	Welche Kosten zieht der Einsatz der Technologie nach sich (Schulungen/Wartung/Updates/...)? (vgl. [JS09])
Kostenvorteile	Welche direkten und indirekten Kostenvorteile entstehen durch den Einsatz der Technologie? (vgl. [JS09])
Meinungen (vgl. [JS09])	
Meinungen	Wie ist die Meinung von Kunden, Experten, Angestellten etc. über die Technologie? (vgl. [JS09])

Tabelle 3.1 – Bewertungskriterien für Cross-Platform-Technologien

3.6 Evaluierungsmethode

Die dritte zu beantwortende Nebenfrage befasst sich mit der geeigneten Evaluierungsmethode für die Auswahl einer Cross-Platform-Technologie. Aufgabe dieser Methode ist es, aus den verfügbaren Technologien die am besten geeignete Technologie zu ermitteln. Bei der Auswahl von Softwarepaketen werden hierfür in der Literatur vorrangig drei Methoden verwendet: Analytic Hierarchy Process (AHP), gewichtetes Scoring sowie Fuzzy-basierte Ansätze [JS09].

Sowohl AHP als auch die Fuzzy-basierten Ansätze zeichnen sich durch eine komplexe mathematisch-theoretische Fundierung aus [MH10]. Auch in der Durchführung sind diese Methoden aufwändig: Allein der Versuch, die Durchführungsvorschriften als Grundlagen zu formulieren, würde den Rahmen dieser Arbeit überschreiten. Andererseits wäre ohne diese Grundlagen die Technologiebewertung und damit das Ergebnis der Arbeit nicht nachvollziehbar, was nicht akzeptabel ist. Somit scheiden AHP und Fuzzy-basierte Methoden für die Zwecke dieser Arbeit als Evaluierungsmethoden aus. Das hat zur Folge, dass hier die einfach durchzuführende Methode des gewichteten Scorings angewandt wird.

Aufgabe des gewichteten Scorings ist es, jeder Alternative einen Betrag an Bewertungspunkten, den so genannten *Score*, zuzuweisen. Je höher der Score, desto besser ist die Alternative für eine gegebene Aufgabenstellung geeignet. Zur Ermittlung des Scores wird zunächst jede Alternative hinsichtlich verschiedener Kriterien untersucht, wobei für jedes Kriterium zu jeder Alternative ein Befund ermittelt wird, der im Anschluss in einen Punktewert überführt wird. Diese Punktewerte werden schließlich gewichtet summiert. (vgl. [JS09])

In der Literatur existiert eine sehr große Anzahl verschiedener Vorschläge, wie das gewichtete Scoring praktisch umzusetzen ist [MH10]. Das hier verwendete Scoring-Verfahren wurde für die gegebene Fragestellung entwickelt, mit der Maßgabe, möglichst einfach und nachvollziehbar zu sein. Es entspricht einer vereinfachten Version des Verfahrens *SMARTER* [EB94][MH10] und funktioniert folgendermaßen:

Zunächst werden die für eine Entscheidung relevanten Kriterien festgelegt, im Falle dieser Arbeit durch den Projektinhaber. Alsdann wird für jede Alternative ermittelt, wie gut sie die Kriterien erfüllt. Diese so genannten *Realbefunde* können numerischer oder natürlichsprachlicher Natur sein.

Im zweiten Schritt werden die Realbefunde in Bewertungspunkte *(Teilscores)* überführt. Diese geben an, wie gut eine Alternative das jeweilige Kriterium erfüllt. Das entspricht dem Verfahren in Fachzeitschriften wie denjenigen der Stiftung Warentest: Aus einer Beobachtung wie „Die Waschmaschine verbraucht pro Waschgang 12 l Wasser“ wird ein Score wie „Wasserverbrauch: ++“. Das hier verwendete Scoring-Verfahren ver-

wendet Dezimalbruch-Teilscores zwischen 0 und 1, wobei 0 den schlechtesten Realbefund abbildet und 1 den besten.

Die ermittelten Teilscores jeder Alternative werden anschließend in einen Gesamtscore für die Alternative überführt. Dies muss gewichtet erfolgen, denn nicht jedes Kriterium ist für die Entscheidung gleichermaßen relevant. Mathematisch ausgedrückt: Der Score s_a einer Alternative a ist die gewichtete Summe aller Teilscores v_{ia}, wobei w_i das dem Kriterium i zugeordnete Gewicht darstellt (vgl. [MH10]):

$$s_a = \sum_{i=1}^{n} w_i \cdot v_{ia} \quad (1 \ldots n \text{ Kriterien}) \tag{3.1}$$

Die Ermittlung korrekter Gewichte w_i ist schwierig [JS09], denn sie sollen die Präferenzen des Entscheiders möglichst genau widerspiegeln. Dass dies nicht ohne Weiteres zu bewerkstelligen ist, wird allein aus der schieren Anzahl in der Literatur vorgeschlagener diesbezüglicher Verfahren ersichtlich [MH10].

In dieser Arbeit wird ein einfaches, wirkungsvolles Verfahren zur Approximation der Gewichte verwendet: der so genannte *Rank Order Centroid* (ROC). Es konnte nachgewiesen werden, dass durch ROC-Approximation die Präferenzen von Entscheidern hinreichend genau abgebildet werden können [BB96]. Die Verwendung des ROC erfordert vom Entscheider statt einer expliziten Gewichtung der Kriterien lediglich das Aufstellen einer Rangfolge der Kriterien, d.h. der Entscheider muss die Kriterien nach ihrer relativen Wichtigkeit sortieren. Aus dieser Rangfolge errechnet sich nach Formel 3.2 das approximative Gewicht des jeweiligen Kriteriums (vgl. [BB96][MH10]). Die entstehenden ROC-Gewichte für ein bis neun Kriterien sind in Tabelle 3.2 aufgeführt.

$$w_{ROCj} = \frac{1}{n} \sum_{k=j}^{n} \frac{1}{k} \quad (n = \text{Anzahl der Kriterien; } j = \text{Rang des Kriteriums}) \tag{3.2}$$

Wichtig bei der Aufstellung der Rangfolge ist, dass der Entscheider die Kriterien nicht abstrakt sortiert, sondern anhand der ermittelten Realbefunde. Hintergrund dieser Anforderung ist die Überlegung, dass ein Kriterium, das von allen Alternativen gleich gut erfüllt wird, nicht entscheidungsrelevant sein kann [HKR99]. Erst wenn eine Alternative ein Kriterium besser (oder schlechter) erfüllt als andere, wird das Kriterium die Entscheidung beeinflussen können. Entsprechend werden auch Kriterien, deren Werte sehr ähnlich sind („Preis: 199 Euro“ und „Preis: 195 Euro“), weniger Tragweite in der Entscheidung haben als Kriterien, deren Realbefunde stärker divergieren („Lieferzeit: 24 Stunden“ und „Lieferzeit: 3 Wochen“). Der Entscheider muss also die Kriterien dahingehend sortieren, welche Wichtigkeit er den unterschiedlichen Realbefunden beimisst.

	Anzahl der Kriterien								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Rang 1	1,000	0,750	0,611	0,521	0,457	0,408	0,370	0,340	0,314
Rang 2	-	0,250	0,278	0,271	0,257	0,242	0,228	0,215	0,203
Rang 3	-	-	0,111	0,146	0,157	0,158	0,156	0,152	0,148
Rang 4	-	-	-	0,062	0,090	0,103	0,109	0,111	0,111
Rang 5	-	-	-	-	0,040	0,061	0,073	0,079	0,083
Rang 6	-	-	-	-	-	0,028	0,044	0,054	0,061
Rang 7	-	-	-	-	-	-	0,020	0,033	0,042
Rang 8	-	-	-	-	-	-	-	0,016	0,026
Rang 9	-	-	-	-	-	-	-	-	0,012
Summe	**1,000**	**1,000**	**1,000**	**1,000**	**1,000**	**1,000**	**1,000**	**1,000**	**1,000**

Tabelle 3.2 – ROC-Gewichte für ein bis neun Kriterien

Nachdem die Fragestellungen und die Methodik in diesem Kapitel geklärt wurde, kann im folgenden Kapitel nun die Forschungsfrage 1 beantwortet werden.

„Ordnung ist das halbe Leben.“ – Deutsches Sprichwort

4 Technologie-Überblick

Ziel dieses Kapitels ist es, einen Überblick über die verfügbaren Cross-Platform-Technologien zu geben. Das Kernanliegen ist also nicht, alle Technologien umfassend zu beschreiben, sondern einem Entscheider aufzuzeigen, welche Technologien für eine konkrete Aufgabenstellung grundsätzlich in Betracht kommen könnten und näher untersucht werden sollten.

4.1 Gruppierung

Die 18 in diesem Kapitel vorgestellten Technologien verwenden teilweise sehr ähnliche technische Ansätze, um Cross-Platform-Entwicklung zu ermöglichen. Um die Verständlichkeit der Erläuterungen zu erhöhen, werden die Technologien nach diesen zugrundeliegenden technischen Ansätzen gruppiert. Dies erleichtert auch die Diskussion ihrer Eigenschaften, denn das grundlegende Prinzip determiniert viele Eigenschaften einer Technologie. Die Technologien werden in sechs Gruppen unterteilt.

Portierte Desktop-Runtimes bezeichnet eine Gruppe von Cross-Platform-Technologien, die für Desktop-Rechner entwickelt wurden und im Nachhinein für mobile Geräte portiert wurden. Sie nutzen das in Abschnitt 2.5.1 vorgestellte Prinzip der Laufzeitumgebung („Runtime“). Wie gezeigt werden wird, sind diese Lösungen für den Mobilbereich nicht optimal einzusetzen.

Die anderen fünf Technologie-Gruppen sind speziell für Mobilgeräte entwickelt worden und lassen sich am besten durch ein Schaubild einführen (Abbildung 4.1). Das Diagramm zeigt ein Spektrum von Technologien: Links ist der Pol der Webtechnologien, rechts der Pol der nativen Technologien. Dazwischen sind die Namen der fünf Technologie-

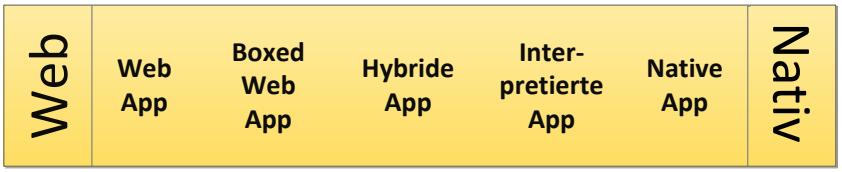


Abbildung 4.1 – Anordnung der Technologie-Gruppen im Spektrum zwischen Webtechnologien und nativen Technologien

Gruppen aufgeführt, in welche die gefundenen Cross-Platform-Technologien eingeordnet werden.

Die Technologien für *Web Apps* sind dem Web-Pol am nächsten und nutzen keine nativen Bestandteile. *Boxed Web Apps* verwenden größtenteils Webtechniken und nur wenige native Komponenten. Bei den *hybriden Apps* herrscht in etwa Gleichgewicht zwischen nativen und webbasierten Bestandteilen, während bei *interpretierten Apps* die nativen Bestandteile überwiegen und schließlich bei den *nativen Apps* überhaupt keine Webkomponenten mehr vorhanden sind.

Jedem dieser grundsätzlichen Cross-Platform-Prinzipien ist in dieser Arbeit ein separater Abschnitt gewidmet, welcher zunächst das Prinzip erklärt und anschließend die zugehörigen Technologien knapp vorstellt. Der Großteil der recherchierten Fakten wird hierbei in Tabellen aufgeführt, deren Aufbau im Folgenden beschrieben wird.

4.2 Aufbau der Übersichtstabellen

Im jeweils **ersten Abschnitt** der Tabellen finden sich Angaben zum Hersteller der Technologie und seiner geografischen Position, was vorrangig der Orientierung dient. Zur vertiefenden Recherche ist die Webseite des Produkts angegeben, und im Sinne der Nachvollziehbarkeit wird zudem die Produktversion dokumentiert, auf der die Angaben in der Tabelle beruhen. Ebenfalls aufgeführt wird, ob die Technologie *Open Source* ist, sowie die Lizenz, unter der die Technologie genutzt werden kann. Mögliche Werte sind: MIT-Lizenz[1], GPL v2[2], GPL v3[3], Apache 2.0[4] und Hersteller-eigene, kommerzielle Lizenzen.

Im Abschnitt **Features** wird aufgeschlüsselt, welche Entwicklungsaufgaben durch die Technologie plattformübergreifend gelöst werden können. Ziel ist es, eine effektive Vorauswahl der Technologien dadurch zu erleichtern, dass angegeben wird, welche Aufgabenbereiche die Technologie lösen kann. In der Tabelle wird zu jeder Technologie die Antwort auf die folgenden Fragen gegeben:

User Interface: Lässt sich das User Interface plattformübergreifend gestalten?

Persistenz: Stellt die Technologie einen Weg bereit, um Daten dauerhaft und strukturiert zu speichern, z.B. eine Datenbank?

[1] `http://www.opensource.org/licenses/mit-license.php`
[2] `http://www.gnu.de/documents/gpl-2.0.de.html`
[3] `http://www.gnu.de/documents/gpl-3.0.de.html`
[4] `http://www.apache.org/licenses/LICENSE-2.0`

Hardware-Zugriff: Kann die Technologie plattformunabhängig auf mindestens ein Geräte-Feature (z.B. Dateisystem, Kontakte, Telefon) zugreifen?

App-Store-fähig: Können die mit der Technologie erstellten Apps in einem App Store vertrieben werden?

Entwicklungsumgebung: Steht dem Entwickler eine für die Zwecke der Technologie optimierte Entwicklungsumgebung zur Verfügung?

Programmiersprachen: In welchen Programmier- und/oder Auszeichnungssprachen werden die Anwendungen entwickelt?

Die unterstützten Plattformen finden sich im Abschnitt **Plattformen**. Ziel ist auch hier, eventuelle Anforderungen von Entscheidern schnell beantworten zu können.

Quelle dieser Angaben sind – wenn nicht anders angegeben – die referenzierten Webseiten der Hersteller oder eine von dort aus verlinkte Seite. Es wurde zu jeder Technologie die API-Beschreibung gesichtet und gegebenenfalls eine Demoversion heruntergeladen, um die in der Übersicht notwendigen Angaben zu ermitteln. Recherchezeitraum war der Juli 2011. Die Angaben wurden im Rahmen einer Nachrecherche im November 2011 auf den Datenstand vom 20.11.2011 aktualisiert.

4.3 Portierte Desktop-Runtimes

Zunächst werden die klassischen Desktop-Cross-Platform-Technologien Java, Flash und .NET (Mono) auf ihre Eignung für plattformübergreifende, mobile App-Entwicklung untersucht.

4.3.1 Java

Im Fall von Java lässt sich ein klares Ergebnis ermitteln: Java ist nicht zur plattformunabhängigen Android- und iOS-Programmierung geeignet, denn Java-Anwendungen laufen nicht unter iOS. Hintergrund ist, dass Apple Java als veraltete Technologie einstuft und in der Konsequenz keine Java-Runtime für iOS bereitgestellt hat [App10a][Pog07]. Auch ließ Apple bis September 2010 nur solche Apps zum Vertrieb im App Store zu, die in Objective-C, C, C++ oder JavaScript entwickelt waren, weswegen Java-Anwendungen nicht auf iOS-Geräten installiert werden konnten [MGL10]. Auch nach Aufhebung dieser Restriktion wurde bislang keine Java-Laufzeitumgebung für iOS veröffentlicht.

4.3.2 Flash

Wie in Abschnitt 2.2.5 festgestellt, wird Flash auf iOS nicht nativ unterstützt, weitgehend aus den gleichen Gründen wie auch Java nicht [Job10]. Der Versuch von Adobe, Flash dennoch auf iOS-Geräten zu etablieren, wurde durch die eben erwähnte App-Store-Richtlinie konterkariert. Erst seit Apple diese Richtlinie im September 2010 aufhob, können Adobe FlashBuilder[5] sowie Adobe Flex SDK[6] zur Erstellung mobiler Apps mit Flash genutzt werden. Auch ein französischer Softwarehersteller veröffentlichte ein entsprechendes Tool namens *OpenPlug ELIPS*[7], welches auf Adobe Flex basiert.

Trotzdem ist es nicht unwahrscheinlich, dass in absehbarer Zukunft Adobe die Unterstützung für Flash-Apps einstellen wird. Immerhin wurde die Entwicklung des mobilen Flash-Plug-Ins für Browser bereits eingestellt [Win11]. Zudem ist Apples Kritik an Flash nicht unfundiert: Flash sei im PC-Zeitalter für PCs und für Mäuse entwickelt worden und versage bei den im mobilen Zeitalter wichtigen Themen wie Energiesparsamkeit, Touch-Interfaces und offene Webstandards [Job10]. Auch die Entwicklung von OpenPlug ELIPS wurde mittlerweile eingestellt [Ens11]. Aufgrund dieser negativen Perspektive sei hier lediglich auf die Möglichkeit der Flash-basierten Cross-Platform-Entwicklung hingewiesen, ohne die Techniken aber im Detail zu erläutern.

4.3.3 .NET (Mono)

Die ursprünglich unter der Marke *Mono* von Novell[8] entwickelten .NET-Laufzeitumgebungen für Linux, Mac OS, iOS und Android wurden nach der Übernahme von Novell durch Attachmate[9] nicht weiterentwickelt [Kah11]. Teile der entlassenen Entwicklermannschaft des Mono-Projekts gründeten im Mai 2011 die Firma Xamarin[10], um die Mono-Projekte weiterzuführen [Ica11]. Das besondere Augenmerk der Firma liegt auf Mono für Android und Mono für iOS (sog. *MonoTouch*) [Ica11].

Allerdings sind MonoTouch und Mono für Android zwei separate Produkte. Hauptziel war nicht das Erstellen einer Cross-Platform-Umgebung, sondern das Implementieren einer .NET-Laufzeitumgebung samt plattformspezifischer Erweiterungen. Das hat zur Folge, dass die Mono-API in weiten Teilen ein C#-Wrapper für die nativen API-Befehle der jeweiligen Plattform ist. Dies betrifft unter anderem sämtliche User-Interface-Elemente, die Befehle für Netz- und Hardwarezugriff sowie die Datenbank-Anbindung.[11]

[5] `http://www.adobe.com/products/flash-builder.html`
[6] `http://www.adobe.com/products/flex/`
[7] `http://www.openplug.com/`
[8] `http://www.novell.com/`
[9] `http://www.attachmate.com/`
[10] `http://xamarin.com/`
[11] vgl. `http://docs.xamarin.com/android` und `http://docs.mono-android.net/`

So können Apps für Android und iOS zwar in einer einheitlichen Programmiersprache entwickelt werden, aber eine beträchtliche Menge an Quellcode muss plattformspezifisch – und damit mehrfach – implementiert werden. Da in einem Vorgängerprodukt des Projektinhabers allein der Code für das User Interface knapp die Hälfte des gesamten Quellcodes ausmachte, wird Mono in dieser Arbeit nicht als vollwertige Cross-Platform-Technologie eingestuft.

Somit ist festzustellen, dass keine der klassischen Desktop-Runtimes eine tragfähige Lösung zur Cross-Platform-Entwicklung für Mobilgeräte darstellt.

4.4 Technologien für Web Apps

Die zweite ermittelte Gruppe von Cross-Platform-Technologien umfasst vier Ansätze, die das Problem der plattformunabhängigen Web-App-Entwicklung lösen. Ansatzpunkt ist die Imitation des Aussehens nativer Apps durch speziell gestaltete HTML/CSS-Webseiten. Solcherart erstellte Apps lassen sich prinzipbedingt nur im Browser des Smartphones ausführen und können nicht wie reguläre Apps über den App Store vertrieben werden. Erst durch Kombination mit anderen Cross-Platform-Technologien können mit den in diesem Abschnitt vorgestellten Technologien auch eigenständige Apps erzeugt werden (vgl. Abschnitt 4.5).

Die Technologien *jQTouch* und *jQuery mobile* bauen auf der JavaScript-Library jQuery[12] auf und divergieren vorrangig in der Tatsache, dass *jQTouch* nur in WebKit-basierten Browsern funktioniert, während *jQuery mobile* die Lauffähigkeit auf so gut wie allen Smartphone- und Desktop-Plattformen anstrebt.

Sencha Touch ist, wie die beiden eben vorgestellten Technologien, komplett in HTML, CSS und JavaScript geschrieben, basiert aber nicht auf jQuery. Es implementiert ein eigenes, umfassendes App-Framework in JavaScript. Anstatt Webseiten durch HTML-Code zu erstellen, nutzen Entwickler eine JavaScript-API, welche eine High-Level-Syntax für das Erstellen von Toolbars, Listen, Buttons und dergleichen bietet.

Die Technologie *GwtMobile* erweitert das *Google Web Toolkit* (*GWT*)[13] um Fähigkeiten für mobile Web-Apps. Von Google ursprünglich für Desktop-Webanwendungen entwickelt, können mittels GWT Webanwendungen in Java implementiert und durch Kompilieren in HTML-, CSS- und JavaScript überführt werden [Dew08]. Die dem Hobbybereich entstammende Technologie GwtMobile ist beim Erstellen von User-Interfaces allerdings weniger leistungsstark als die drei anderen Technologien dieser Gruppe.

[12] `http://jquery.com/`

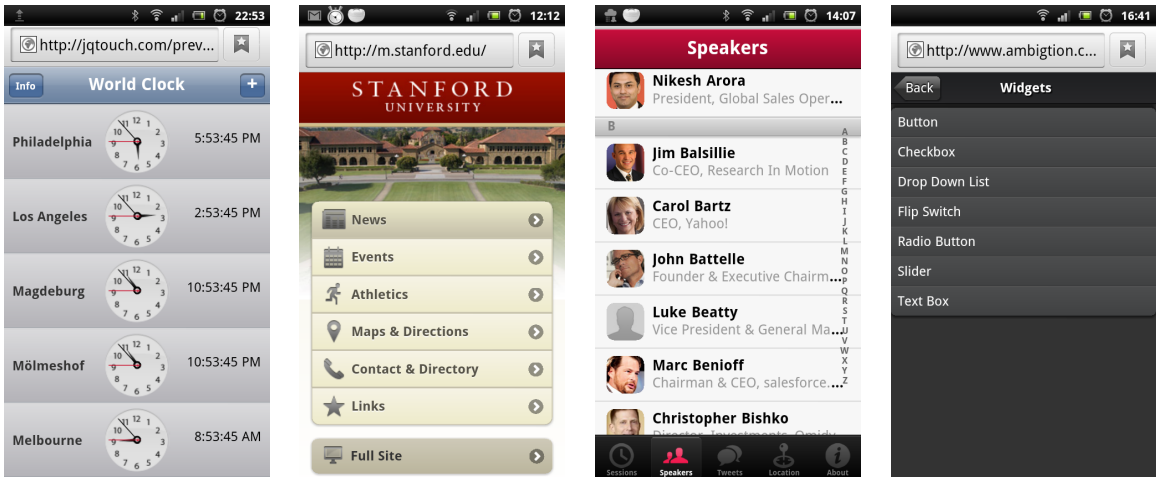
[13] `http://code.google.com/intl/de-DE/webtoolkit/`

	jQTouch[1]	jQuery mobile[2]	Sencha Touch[3]	GwtMobile[4]
Hersteller	Kaneda, D; Stark, J.	The jQuery Project	Sencha Inc.	Jiang, D. et al.
Sitz	USA	vorwiegend USA	USA	k.A.
OpenSource	ja	ja	ja	ja
Lizenz	MIT	MIT, GPL v2	Kommerziell / GPL v3	Apache 2.0
Vorgestellte Version	Version 1.0 beta 3	Version 1.0	Version 1.1.0	Version 1.1
Features				
User Interface	ja	ja	ja	ja
Persistenz	nein	nein	bis 5 MB[5]	bis 5 MB[5]
Hardware-Zugriff	nein	nein	nein	nein
App-Store-fähig	nein	nein	nein	nein
Entwicklungsumgebung	nein	nein	nein	nein
Programmiersprachen	HTML/CSS/JS	HTML/CSS/JS	JavaScript	Java
Plattformen				
iOS / Android	ja / ja	ja / ja	ja / ja	ja / ja
Weitere Plattformen	ja (WebKit)	ja (zahlreiche)	ja (WebKit)	ja (WebKit)

[1] `http://sencha.com/products/touch/`
[2] `http://jqtouch.com/`
[3] `http://jquerymobile.com/`
[4] `https://github.com/dennisjzh/GwtMobile`
[5] Persistenz-Unterstützung im Rahmen der Möglichkeiten des Browsers (vgl. Abschnitt 2.6.3)

Tabelle 4.1 – Überblick über die Technologien für Web Apps

Dies wird auch in Abbildung 4.2 deutlich, welche Screenshots von mit den Technologien umgesetzten Web Apps zeigt[14]. In Tabelle 4.1 werden die Eigenschaften aller vier Technologien gegenübergestellt.



(a) jQTouch **(b)** jQuery mobile **(c)** Sencha Touch **(d)** GwtMobile

Abbildung 4.2 – Imitation nativer User Interfaces per HTML5 und CSS3

[14] Quellen: `http://m.stanford.edu/` `http://jqtouch.com/preview/demos/clock/#home` `http://dev.sencha.com/deploy/touch/examples/oreilly/` und `http://www.ambigtion.com/gwtmobileui/`

4.5 Technologien für Boxed Web Apps

Die beiden Haupteinschränkungen von Web Apps sind, dass sie sich als reine Internetseiten nicht über den App Store vertreiben lassen und nicht auf die Smartphone-Hardware zugreifen können. Abhilfe schaffen Technologien, für die in dieser Arbeit der Begriff *Boxed Web Apps* eingeführt wird.

Bei Boxed Web Apps handelt es sich um Apps, deren Anwendungsfunktionalität als Internetseite umgesetzt wird, beispielsweise mit einer der eben aufgeführten Web-Technologien. Den Rahmen zur Ausführung der Web App stellt aber nicht der Browser des Smartphones bereit, sondern eine eigenständige, native App, deren Aufgabe darin besteht, die Web App bildschirmfüllend anzuzeigen. Als native App kann sie zudem auf das Dateisystem, das Telefonbuch und die sonstige Hardware zugreifen und diese Zugriffsmöglichkeit der gehosteten Web App zur Verfügung stellen. Abbildung 4.3 stellt Web Apps und Boxed Web Apps gegenüber.

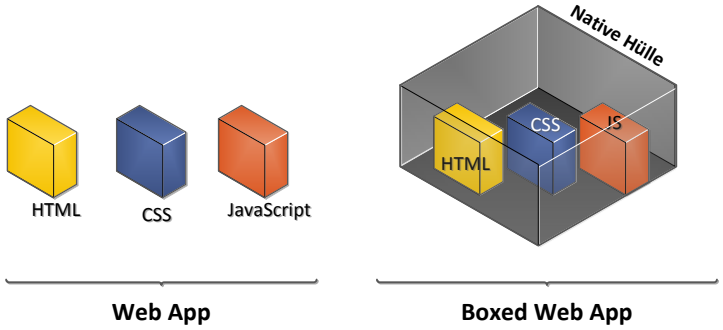


Abbildung 4.3 – Web Apps und Boxed Web Apps

Prominentester Vertreter dieses Ansatzes ist *PhoneGap*, dessen Hersteller Nitobi[15] im Oktober 2011 von Adobe aufgekauft wurde [Ado11a]. Adobe Dreamweaver CS 5.5[16] unterstützt bereits die Erzeugung von PhoneGap-Apps [Ado11b].

Ein anderer Vertreter ist das Produkt *QuickConnectFamily Hybrid*, welches federführend von L. Barney entwickelt wird. Bei der Recherche entstand beim Verfasser allerdings der subjektive Eindruck einer „One-Man-Show“, die mehr verspricht als hält.[17]

Dritter Vertreter ist der Cloud-Dienst *FeedHenry Studio*, der die App-Entwicklung in eine Webanwendung verlagert. Eine versuchsweise erstellte Demo-Anwendung wies

[15] `http://www.nitobi.com/`

[16] `http://www.adobe.com/de/products/dreamweaver.html`

[17] Unter anderem, weil der überwiegende Teil der Änderungen an Dokumentation und Quellcode von L. Barney stammt, wobei der Umfang der angepriesenen Features mutmaßlich einer ganzen Entwicklermannschaft bedürfte.

	PhoneGap[1]	**QuickConnect-Family Hybrid[2]**	**FeedHenry Studio[3]**
Hersteller	Adobe Systems Inc.	Barney, L. et al.	FeedHenry Inc.
Sitz	USA	k.A.	USA, Irland
Open Source	ja	ja	nein
Lizenz	MIT	MIT	k.A.
Vorgestellte Version	Version 1.2.0	Version 2.1	Stand 19.11.11
Features			
User Interface	nein	nein	ja
Persistenz	bis 5 MB[4]	ja	ja
Hardware-Zugriff	ja	ja	ja
App-Store-fähig	ja	ja	ja
Entwicklungsumgebung	ja	nein	ja
Programmiersprachen	HTML/CSS/JS	HTML/CSS/JS	HTML/CSS/JS
Plattformen			
iOS / Android	ja / ja	ja / ja	ja / ja
Weitere Plattformen	ja (zahlreiche)	ja (WebKit)	ja (zahlreiche)

[1] `http://www.phonegap.com/`
[2] `http://www.quickconnectfamily.org/`
[3] `http://www.feedhenry.com/`
[4] Persistenz-Unterstützung im Rahmen der Möglichkeiten des Browsers (vgl. Abschnitt 2.6.3)

Tabelle 4.2 – Überblick über die Technologien für Boxed Web Apps

allerdings unter Android erhebliche Darstellungsmängel auf und reagierte nicht auf Nutzereingaben. Sie befindet sich zu Referenzzwecken auf der beiliegenden DVD.

Tabelle 4.2 listet die Eigenschaften der drei Technologien auf.

4.6 Technologien für hybride Apps

Ausgehend vom Konzept der Boxed Web App werden bei *hybriden Apps* zusätzliche native Features genutzt. Der Übergang zwischen Boxed Web Apps und hybriden Apps ist fließend, denn auch Boxed Web Apps verwenden einige native Zugriffe. Grenzmerkmal ist die Verwendung nativer User-Interface-Elemente: Technologien, welche dies ermöglichen, werden hier als Technologien für hybride Apps aufgeführt. Es wurden bei der Recherche zwei solche Technologien ausfindig gemacht: *Worklight* und *Rhodes*.

Analog zu PhoneGap stellen diese beiden Technologien eine native Hülle zur Ausführung von Webanwendungen bereit; zusätzlich können native User-Interface-Elemente wie Tabbars und Kontextmenüs verwendet werden. Der App-Entwickler kombiniert für das User Interface folglich Webelemente mit nativen Elementen, wobei der Großteil der Anwendung – Texte, Grafiken, Listen, Buttons etc. – webbasiert umgesetzt wird.

Vom Leistungsumfang her ist Rhodes Worklight überlegen: Während Rhodes Unterstützung für alle zentralen User-Interface-Aufgaben bietet, verweist Worklight auf die

Möglichkeit, diese selbst nativ (und somit mehrfach) zu implementieren. Möglich sei auch die Einbindung von Sencha Touch und PhoneGap.

Weiterhin bietet Rhodes einen integrierten lokalen Webserver, der sich stark am Web-Framework Ruby On Rails[18] orientiert. Somit können die anzuzeigenden Webseiten dynamisch auf dem Gerät erzeugt werden, was große Flexibilität erlaubt. Der Hersteller von Rhodes, das kalifornische Start-Up *Rhomobile*[19], wurde im Oktober 2011 von Motorola Solutions[20] übernommen [Mot11].

Zusammenfassend werden die Eigenschaften von Rhodes und Worklight in Tabelle 4.3 aufgelistet.

	Worklight[1]	**Rhodes**[2]
Hersteller	Worklight Inc.	Rhomobile Inc.
Sitz	USA, Israel	USA
Open Source	nein	ja
Lizenz	Kommerziell	MIT
Vorgestellte Version	Version 4.2	Version 3.1.0
Features		
User Interface	bedingt[3]	ja
Persistenz	ja	ja
Hardware-Zugriff	nein[4]	ja
App-Store-fähig	ja	ja
Entwicklungsumgebung	ja	ja
Programmiersprachen	HTML/CSS/JS, ggf. Objective-C bzw. Java	Ruby und HTML/CSS/JS
Plattformen		
iOS / Android	ja / ja	ja / ja
Andere	ja (wenige)	ja (zahlreiche)

[1] `http://www.worklight.com/`
[2] `http://rhomobile.com/products/rhodes`
[3] Wenige native UI-Elemente; keine Web-UI-Elemente
[4] Entwickler muss auf PhoneGap zurückgreifen oder den Zugriff selbst nativ implementieren

Tabelle 4.3 – Überblick über die Technologien für hybride Apps

4.7 Technologien für interpretierte Apps

Der Ansatz, native User-Interface-Elemente durch plattformübergreifende API-Befehle zu erzeugen, steht im Mittelpunkt der beiden in dieser Gruppe vorgestellten Technologien.

Bei *Titanium Mobile* steht dem Entwickler eine JavaScript-API zur Verfügung, über die sich sämtliche User-Interface-Elemente erzeugen lassen. Der vom Entwickler erstellte

[18] Vertiefung zu Ruby on Rails: [Har11], Vertiefung zu Ruby: [Bov07][FM08]
[19] `http://rhomobile.com/`
[20] `http://www.motorolasolutions.com`

	Titanium Mobile[1]	**Corona SDK**[2]
Hersteller	Appcelerator Inc.	Ansca Inc.
Sitz	USA	USA
Open Source	ja	nein
Lizenz	Apache v2	Kommerziell
Vorgestellte Version	Version 1.7.2	Version 2011.689
Features		
User Interface	ja	bedingt[3]
Persistenz	ja	ja
Hardware-Zugriff	ja	ja
App-Store-fähig	ja	ja
Entwicklungsumgebung	ja	nein
Programmiersprache	JavaScript	Lua
Plattformen		
iOS / Android	ja / ja	ja / ja
Weitere Plattformen	nein	nein

[1] `http://www.appcelerator.com/products/`
[2] `http://www.anscamobile.com/corona`
[3] kein Zugriff auf die typischen UI-Elemente der Plattform

Tabelle 4.4 – Überblick über die Technologien für interpretierte Apps

JavaScript-Code wird auf den Endgeräten durch eine Laufzeitumgebung interpretiert.[21] Es sind so auch Zugriffe auf Hardware-Komponenten, Netzwerk und das Dateisystem möglich.

Die Technologie *Corona SDK* verwendet ebenfalls diesen Ansatz [Sta11b], allerdings kommt als Programmiersprache die Scriptsprache Lua[22] zum Einsatz. Corona SDK ist schwerpunktmäßig für Spiele konzipiert und bietet eine Vielzahl entsprechender Features wie eine Physik-Engine und eine Storyboard-API. Für die Umsetzung von geschäftlichen Apps sind hingegen kaum dedizierte Features vorhanden.

Tabelle 4.4 führt die wichtigsten Eigenschaften beider Technologien auf.

4.8 Technologien für native Apps

Die letzte Gruppe von Cross-Platform-Technologien setzt gänzlich auf nativen Quellcode. Wie auch das Corona SDK sind diese Technologien vorrangig für Spiele konzipiert. Ansatzpunkt ist, dass sich sowohl iOS als auch Android per C/C++ programmieren lassen. Die Entwicklung in diesen Sprachen ist zwar üblicherweise deutlich aufwändiger als in moderneren Programmiersprachen, bietet aber auch eine ungleich höhere Anwendungsperformance [Phi99][Pre00].

[21] `http://developer.appcelerator.com/question/45001/is-appcelerator-titanium-mobile-really-a-cross-compiler`
[22] `http://www.lua.org` – vertiefend: [Ier06]

	Marmalade[1]	**MoSync**[2]
Hersteller	Ideaworks3D Ltd.	MoSync AB
Sitz	Vereinigtes Königreich	Schweden
Open Source	nein	ja
Lizenz	Kommerziell	Kommerziell / GPL v2
Vorgestellte Version	Version 5.1	Version 2.7
Features		
User Interface	ja	ja
Persistenz	ja	ja
Hardware-Zugriff	ja	ja
App-Store-fähig	ja	ja
Entwicklungsumgebung	nein	ja
Programmiersprachen	C/C++	C/C++
Plattformen		
iOS / Android	ja / ja	ja / ja
Weitere Plattformen	ja (zahlreiche)	ja (zahlreiche)

[1] `http://www.madewithmarmalade.com/` [2] `http://www.mosync.com/`

Tabelle 4.5 – Überblick über die Technologien für vollständig native Apps

Beide Technologien stellen zahlreiche relevante Bibliotheken bereit, beispielsweise für den plattformunabhängigen Zugriff auf die Smartphone-Hardware. Tabelle 4.5 beschreibt die beiden hier nur am Rande relevanten Technologien *Marmalade* und *MoSync*.

Dies beschließt die Kurzvorstellung der verfügbaren Cross-Platform-Technologien für iOS- und Android-Entwicklung. Es wird deutlich, dass vielfältige Technologien mit unterschiedlichen Stärken und Schwächen vorliegen, die näher analysiert werden müssen, ehe eine fundierte Technologie-Entscheidung möglich ist. Daher wird nun ein Prototyp spezifiziert, anhand dessen die Leistungsfähigkeit der Technologien bezüglich der konkreten Aufgabenstellung der PC Agrar GmbH überprüft werden kann.

5 Konzeption des Prototypen

Damit die Evaluierungsergebnisse möglichst relevant für die konkrete Aufgabenstellung der PC Agrar GmbH ausfallen, orientiert sich der Prototyp an der geplanten Endanwendung. Zu beachten ist, dass dieser Teil der vorliegenden Arbeit eine in der betrieblichen Praxis konzipierte Softwareanwendung beschreibt, weswegen nicht alle getroffenen Entscheidungen wissenschaftlich hergeleitet werden können. Nichtsdestotrotz wurde auf fachliche Fundierung, Nachvollziehbarkeit und Schlüssigkeit der Ausführungen geachtet. Zunächst sei in diesem Sinne der inhaltliche Rahmen umrissen, innerhalb dessen die Anwendung angesiedelt ist.

5.1 Anwendungsdomäne Landwirtschaft

Die Landwirtschaft befasst sich mit der „wirtschaftliche[n] Nutzung des Bodens“, um pflanzliche und tierische Produkte zu erzeugen, und gehört zum primären Wirtschaftssektor, dem so genannten *Ursektor* [Bro06]. Viele Landwirte nehmen darüber hinaus auch Aufgaben aus anderen Wirtschaftssektoren wahr, beispielsweise in der Biogas-Produktion (sekundärer Sektor) oder im Winterdienst (tertiärer Sektor).

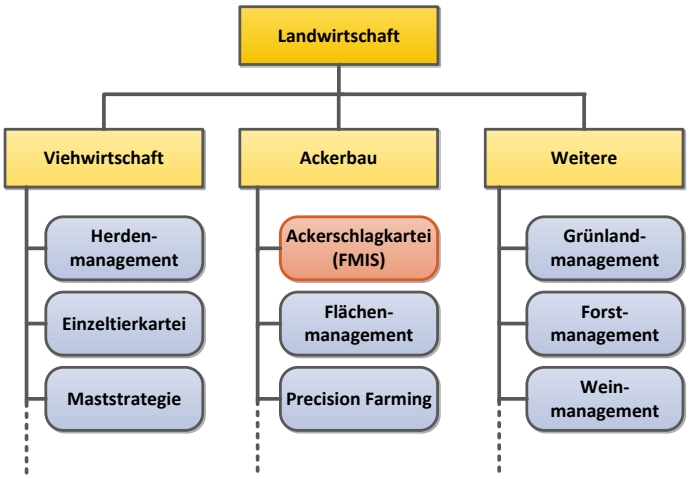


Abbildung 5.1 – Einordnung der vorliegenden Arbeit in den Kontext landwirtschaftlicher Softwareprodukte

Dabei werden sie in allen ihren Aufgabenbereichen durch Softwareprodukte unterstützt. Abbildung 5.1 zeigt einen Ausschnitt aus einer umfangreicheren Aufstellung

aus [DD02]. Die landwirtschaftlichen Produktionsbereiche befinden sich im oberen Teil des Diagramms. Unter jedem Bereich sind einige Beispiele für entsprechende Anwendungsprogramme aufgeführt. Die vorliegende Arbeit ist im Domänenbereich Ackerbau angesiedelt und dort im Bereich der so genannten *Farm Management Information Systems (FMIS)*, die im deutschen Sprachraum zumeist als *Ackerschlagkartei* bezeichnet werden.

5.1.1 Landwirtschaftliches Fachvokabular

Auf landwirtschaftliches Fachvokabular wird weitgehend verzichtet, da diese Arbeit an einer Fakultät für Informatik betreut wird. Einige wichtige Begriffe werden in diesem Abschnitt erläutert, wobei es sich nicht um allgemeine Definitionen handelt, sondern um solche, die das weitere Verständnis der Arbeit fördern sollen und in ihrem Kontext gültig sind:

- Ein **Schlag** bezeichnet ein Stück Land, das der Landwirt nutzen kann. Vereinfachend kann es gedanklich meist durch das Wort „Anbaufläche“ ersetzt werden, jedoch werden auch andere Flächen als Schlag bezeichnet, z.B. Wiesenflächen.

- Als **Ackerschlag** wird ein Schlag bezeichnet, der als Acker bewirtschaftet wird.

- Eine **Ackerschlagkartei** *(ASK)* ist ein integriertes Softwaresystem, welches den Landwirt bei allen für den Ackerbau relevanten Aufgaben unterstützt. Typische Module einer Ackerschlagkartei sind unter anderem Flächenmanagement, Düngeplanung, Lagerhaltung, Vorgangsdokumentation sowie betriebliche und landwirtschaftliche Auswertungen. [DD02]

- Der Begriff **Fruchtart** bezeichnet das auf einem Schlag angebaute Nutzgut, wie beispielsweise *Weizen*, *Kartoffeln* oder *Mais*. Meistens werden vom Landwirt allerdings spezifischere Fruchtartbezeichnungen gewählt, z.B. *Sommerweizen* oder *Silomais*. Die angebaute Fruchtart ist für den Landwirt eine der wichtigsten Informationen über einen Schlag, da sich aus ihr ergibt, wann welche Maßnahmen durchzuführen sind.

- Die **Schlagfläche** wird in Hektar *(ha)* angegeben und bezeichnet die dem Schlag zugeordnete Fläche.

- Ein **Erntejahr** ist ein vom Kalenderjahr abweichendes Wirtschaftsjahr, das in der Ackerschlagkartei zum Einsatz kommt. Wie auch in anderen Wirtschaftsbereichen (z.B. bei Wintersportbetrieben) wird im Ackerbau ein abweichendes Wirtschaftsjahr genutzt. Hintergrund ist, dass zwischen Saatvorbereitung, Aussaat

und Ernte zumeist ein Jahreswechsel liegt, diese Arbeiten jedoch wirtschaftlich zusammengehören und daher im selben Wirtschaftsjahr berücksichtigt werden sollen. Beginn und Ende des Erntejahres sind nicht einheitlich geregelt und können sich sogar innerhalb eines Betriebes von Schlag zu Schlag unterscheiden.

5.1.2 Die Ackerschlagkartei *AO AgrarOffice*

Wie einige andere Softwarehersteller[1] auch vertreibt die PC Agrar GmbH eine Ackerschlagkartei. Abbildung 5.2 zeigt einen Screenshot ihrer Software *AO AgrarOffice*.

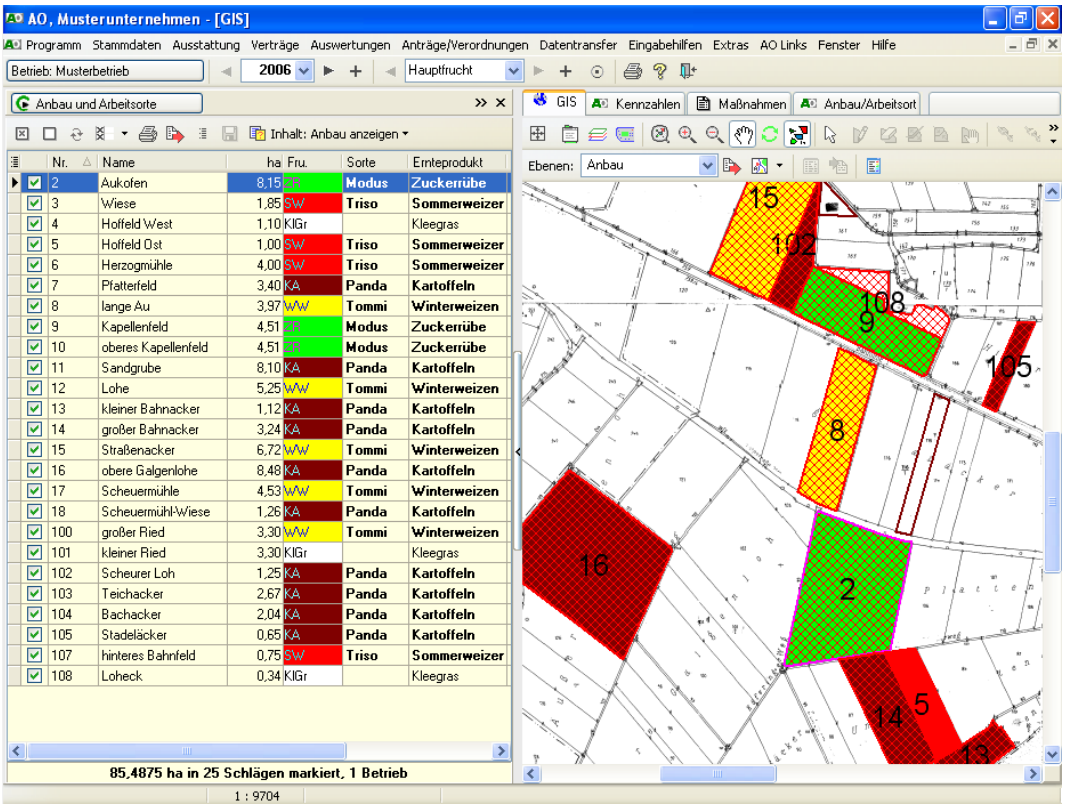


Abbildung 5.2 – AO AgrarOffice mit geöffnetem Geo-Informations-System

Im Kopfteil kann der Betrieb („Musterbetrieb“) und das Erntejahr („2006“) ausgewählt werden. In der linken Bildhälfte ist die Liste der Schläge des Betriebs zu sehen. Zu jedem Schlag sind unter anderem Schlagnummer, Schlagname, Schlagfläche sowie die angebaute Fruchtart angegeben; die Fruchtart ist zudem farblich kodiert. Die rechte Bildhälfte zeigt im Geo-Informations-System („GIS“) die Position der Schläge auf einer Flurkarte, wobei sich auch hier die farbliche Kodierung der angebauten Fruchtarten wiederfindet.

In einem weiteren Screenshot (Abbildung 5.3) wird eine Übersicht der erledigten Maßnahmen eines Schlages gezeigt, da solche Maßnahmenübersichten im weiteren

[1] z.B. HELM-Software Einzelunternehmung (Uwe Helm, `http://www.helm-software.de/`)

Verlauf der Arbeit eine wichtige Bedeutung haben. Aus der Abbildung ist ersichtlich, dass im Laufe des Erntejahres auf jedem Schlag zahlreiche Maßnahmen durchgeführt werden. Zu jeder Maßnahme werden Detailinformationen erfasst, die hier exemplarisch für die Maßnahme vom 11.05.2006 angezeigt werden.

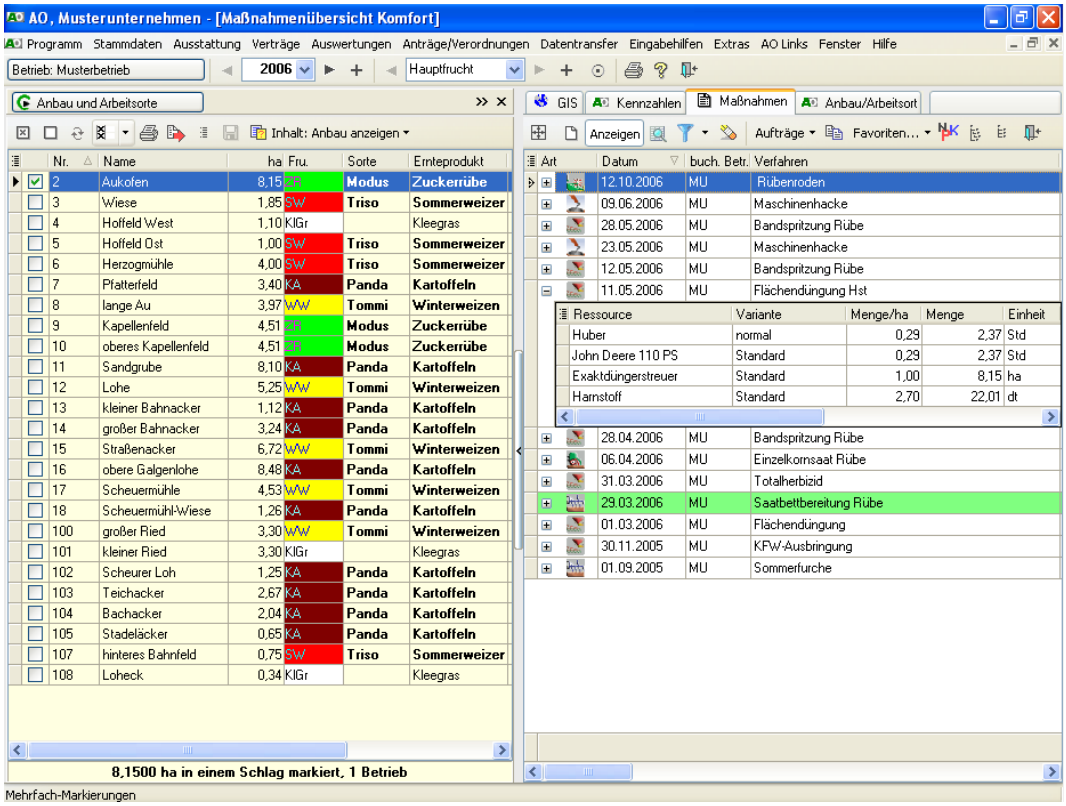


Abbildung 5.3 – Beispielhafte Maßnahmenübersicht

5.1.3 Die mobile Ackerschlagkartei *AO mobile ASK*

Da der Großteil der ackerbaulichen Wertschöpfung auf dem Schlag und damit außerhalb des Büros stattfindet, ist eine PC-gebundene Ackerschlagkartei für den Landwirt nicht zu jedem Zeitpunkt optimal einsetzbar. Oftmals werden vor Ort Informationen aus der Schlagkartei benötigt. Der zentrale Anwendungsfall ist, dass der Landwirt auf dem Schlag mit dem Wuchs der Pflanzen nicht zufrieden ist und nach möglichen Ursachen sucht. Dazu muss er wissen, ob auf diesem Schlag bereits ausreichend gedüngt wurde, wann die letzte Pflanzenschutzmaßnahme stattgefunden hat, wann die Aussaat war und dergleichen mehr. Diese Informationen sind der Maßnahmenübersicht der Schlagkartei leicht zu entnehmen. Allerdings befindet sich diese im Regelfall im Büro des Betriebs, weswegen Landwirte – für manchen Informatiker gänzlich überraschend – zu den *Early Adopters* von Mobilgeräten zählen.

Bereits 1998 vertrieb die PC Agrar GmbH ein damals Palm-basiertes Softwareprodukt, mittels dessen der Landwirt die Maßnahmenhistorie mobil abrufen konnte. Auch die Erfassung von erledigten Maßnahmen war damit direkt auf der Schlagfläche möglich. Dieses Produkt wurde kontinuierlich weiterentwickelt und nach dem Ende der Palm-Ära auf die Plattform *Windows Mobile* portiert. Mit dem Aufkommen der modernen Smartphones muss nun eine entsprechende Lösung für iOS und Android erarbeitet werden.

5.2 Grobkonzept

Neben den bereits festgelegten Mindestanforderungen an Cross-Platform-Technologien (vgl. Abschnitt 3.4) muss für das Softwareprojekt auch ein inhaltlicher Rahmen festgelegt werden. Allerdings definierte die PC Agrar GmbH das Projekt bislang nicht schriftlich. Die in Gesprächen und Meetings gewonnenen Einsichten werden in diesem Text zusammengefasst:

> „Es ist eine Software zu entwickeln, die auf allen aktuellen Smartphone-Plattformen funktioniert, mindestens auf iOS und Android. Der Entwicklungsaufwand muss dabei möglichst gering gehalten werden, weswegen der Einsatz einer Cross-Platform-Technologie notwendig ist.
>
> Inhaltliche Kernfunktion ist der mobile Zugriff auf Daten, die vom Landwirt in seiner PC-basierten Ackerschlagkartei erfasst wurden. In der ersten Ausbaustufe soll der Landwirt auf dem Schlag alle entscheidungsrelevanten Informationen über auf dem Schlag durchgeführte Maßnahmen abrufen können.
>
> Der Datenabgleich erfolgt internetbasiert über den firmeneigenen Cloud-Service *AO NetDok*; dennoch müssen die Daten zwingend auch ohne bestehende Internetverbindung angezeigt werden können. Als zentrales Arbeitswerkzeug für den Landwirt muss die mobile Schlagkartei höchsten Qualitäts- und Performance-Ansprüchen genügen.
>
> Die Software soll später schrittweise erweitert werden, z.B. um die Anzeige von Vorgaben aus der Pflanzenschutzverordnung. Auch sollen zukünftig Maßnahmen mobil erfasst werden.“

Im Rahmen dieser Arbeit wurde in Abstimmung mit dem Projektinhaber eine Einschränkung auf die beschriebene erste Ausbaustufe gewählt, da dies für ca. 70 % der Nutzer der entscheidende Anwendungsfall ist.

5.3 Feinkonzept

Das genauere Konzept des Prototypen wird anhand der gewünschten Anwendungsfunktionalität gestaltet, wobei im Sinne eines vertretbaren Entwicklungsaufwandes eine

möglichst einfache Prototyp-Lösung angestrebt wird. Der obigen Beschreibung lassen sich folgende Kernaufgaben des Prototypen entnehmen:

1. Der Prototyp muss die für den Landwirt relevanten Daten aus dem Internet laden („AO NetDok“).
2. Diese Daten müssen auf dem Smartphone persistent gespeichert werden („Offline-Fähigkeit“). Der Grund dafür ist, dass im ländlichen Raum bei Weitem keine flächendeckende Mobilfunkversorgung vorhanden ist [ATG12].
3. Der Prototyp muss die Maßnahmen, die auf einem spezifischen Schlag durchgeführt wurden, anzeigen.
4. Dazu muss der Nutzer den Schlag auswählen können, für den diese Details angezeigt werden sollen.

Um zunächst eine anschauliche Vorstellung des Prototypen zu erhalten, wird als erstes die Benutzerschnittstelle vorgestellt.

5.3.1 Skizzen der Benutzermasken

Es sind drei Bildschirme vorgesehen, deren logischer Zusammenhang und grobes Aussehen in Abbildung 5.4 dargestellt sind. Ausgehend von einer statischen Startseite kann der Nutzer eine Liste aller Schläge aufrufen, dort einen bestimmten Schlag auswählen und in einem weiteren Bildschirm die Details des Schlages ansehen. Über den Zurück-Button gelangt er jeweils auf die vorherige Seite. Durch dieses einfache Design werden alle benötigten Funktionen abgedeckt. Im Folgenden werden die einzelnen Bildschirme vorgestellt.

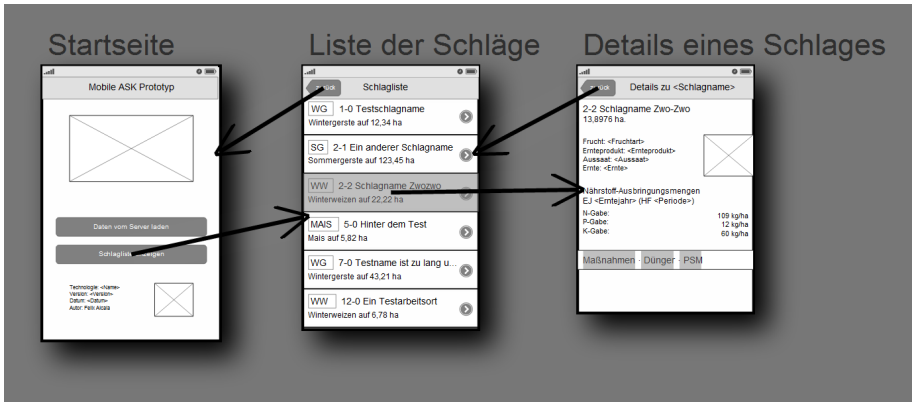
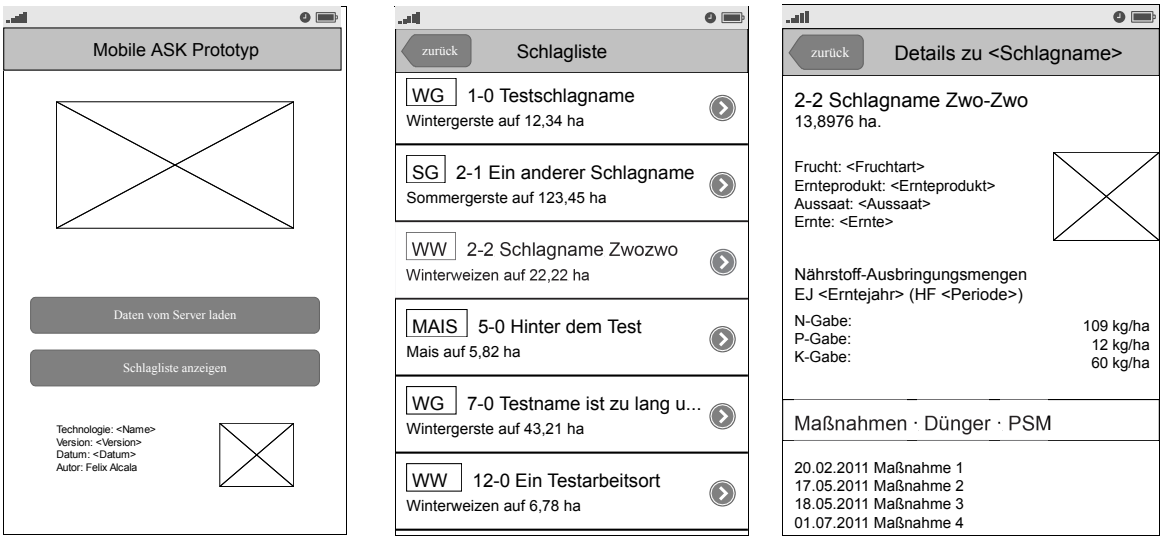


Abbildung 5.4 – Überblick über das User Interface des Prototypen

Der **Startbildschirm** ist der Einstiegspunkt in die Anwendung. Sein Design ist in Abbildung 5.5a ersichtlich. Er enthält zwei Bilder, einige Texte sowie zwei Buttons: einen zum Laden der Daten vom Server und einen zum Anzeigen der Schlagliste.

In der Evaluierung der Prototypen können mit diesem Bildschirm zwei wichtige Eigenschaften der jeweiligen Cross-Platform-Technologien getestet werden: Einerseits



(a) Startseite **(b)** Schlagliste **(c)** Schlagdetails

Abbildung 5.5 – Detaillierte Ansicht der drei Screens des User Interface

die Präzision der User-Interface-Gestaltung und der damit verbundene Aufwand. Andererseits die typische minimale Startzeit von durch die Technologie erstellten Anwendungen, denn der sehr einfach gehaltene Screen kommt ohne potenziell zeitaufwändige Datenbank- und Netzwerkzugriffe aus, womit der überwiegende Teil der Ladezeit der Cross-Platform-Technologie zuzuordnen ist.

Der Nutzer gelangt – nach dem in Abschnitt 5.3.2 erläuterten Laden der Nutzdaten – zum Bildschirm der **Schlagliste**, der in Abbildung 5.5b skizziert ist. Inhaltlich kann er dort für jeden Schlag dessen Nummer, Name und Größe ablesen, sowie das farblich kodierte Kürzel und den Langnamen der angebauten Fruchtart. Durch Tippen auf einen Schlag gelangt der Landwirt zu dessen Schlagdetails.

In der Evaluierung der Technologien spielt dieser Bildschirm eine wichtige Rolle, denn Stammdatenlisten sind in jeder betrieblichen, datenzentrischen Anwendung im wahrsten Sinne des Wortes Dreh- und Angelpunkt der Navigation. Somit kann durch diesen Bildschirm ermittelt werden, wie gut und wie schnell die Technologie Listen laden, darstellen und scrollen kann.

Der dritte und letzte Bildschirm (Abbildung 5.5c) zeigt die **Schlagdetails**, also die eigentlich für den Nutzer relevanten Daten. Im Hauptteil des Screens werden zentrale Informationen wie das Aussaatdatum und das Ernteprodukt sowie die ausgebrachten Nährstoffmengen („N/P/K-Gabe“) übersichtsweise dargestellt. Im unteren Bereich des Screens werden alle einzelnen Maßnahmen aufgeführt.

Die auf diesem Screen angezeigten Informationen werden – wie im nächsten Abschnitt erläutert wird – als HTML-Dateien aus AO AgrarOffice exportiert und dem Smartphone zur Verfügung gestellt. Die Aufgabe dieses Bildschirms besteht folglich in der Darstellung

dieser vorgefertigten, schlagspezifischen HTML-Datei. Bei der Evaluierung der Cross-Platform-Technologien kann hiermit ermittelt werden, wie gut sich Webtechniken in die Zielanwendungen integrieren lassen.

5.3.2 Datenabgleich

Die Nutzdaten müssen laut Aufgabenbeschreibung auf dem Smartphone lediglich angezeigt werden können, so dass eine mobile Datenmodifikation oder -erfassung vorerst nicht notwendig ist. Demgemäß genügt eine Synchronisierung in eine Richtung: vom PC zum Smartphone.

Diese Aufgabenstellung unterscheidet sich von typischen Cloud-Computing-Aufgaben darin, dass die Nutzerdaten *nicht* in der Cloud vorliegen, sondern auf dem PC des Landwirts. Eine Cloud-basierte Datenhaltung ist von vielen AO-Nutzern nicht erwünscht, da Bedenken über Datensicherheit und Missbrauchsmöglichkeiten bestehen. Dennoch ist eine internetbasierte Datenübertragung vorgesehen, da dies die flexibelste Möglichkeit der Synchronisation darstellt und auch das Geschäftsmodell der Anwendung maßgeblich auf die webbasierte Datenübertragung setzt. Um Bedenken über unkontrollierte Datenübertragungen entgegenzuwirken, müssen diese vom Nutzer explizit ausgelöst werden. Abbildung 5.6 beschreibt die vorgesehene Synchronisierungs-Architektur.

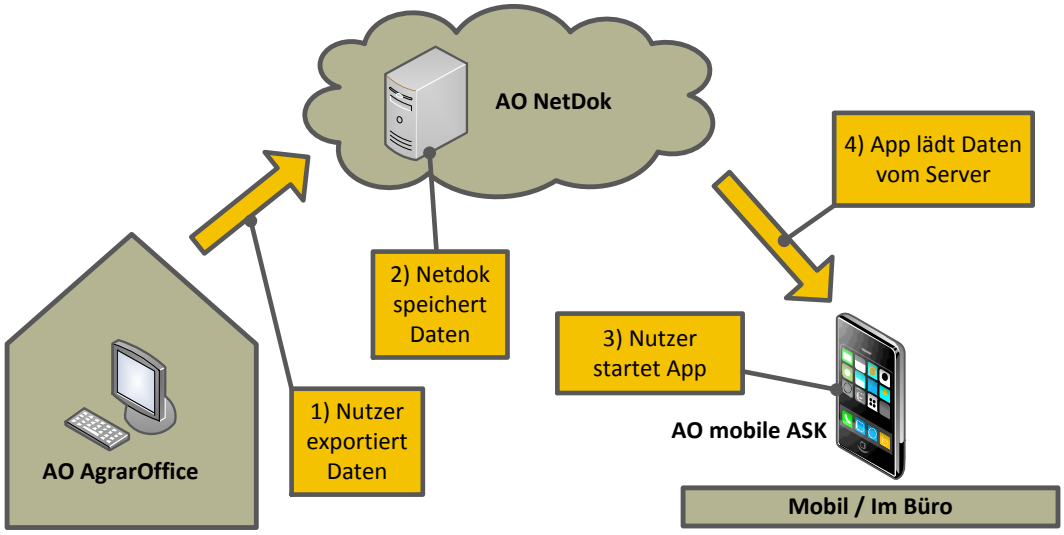


Abbildung 5.6 – Ablaufdiagramm des Datenabgleichs zwischen AO AgrarOffice und AO mobile ASK

Zwei Arten von Daten werden im Prototypen benötigt: Stammdaten-Informationen aller Schläge (Schlagstammdaten) sowie Informationen über die auf den Schlägen durchgeführten Maßnahmen (Schlagdetails).

Die **Schlagstammdaten** werden beim Export aus AO AgrarOffice in das Internet-typische Datenformat JSON[2] konvertiert. Dies erlaubt es zu prüfen, ob die jeweiligen Technologien mit diesem weitverbreiteten Datenformat umgehen können.

Die für den Landwirt relevanten **Schlagdetails** entsprechen in etwa dem Umfang der in der Vorgängerversion für Windows Mobile angezeigten Daten. Somit können diese damals als HTML-Dateien konzipierten Schlagdetails auch in der neu zu entwickelnden Android- und iOS-Anwendung genutzt werden. Die HTML-Dateien werden beim Exportvorgang von AO AgrarOffice generiert, wobei für jeden Schlag eine HTML-Datei mit den spezifischen Informationen zum Schlag erstellt wird. Nach dem Export werden diese Dateien auf den Server AO NetDok übertragen, von wo aus der Prototyp sie herunterladen kann.

Durch dieses Konzept des Prototypen ist die für den Landwirt wichtige Kernfunktionalität abgedeckt. Auf die Umsetzung einer Authentifizierung wurde verzichtet, da diese zum Beginn der Implementierungsaufgaben Server-seitig noch nicht zur Verfügung stand. Im folgenden Abschnitt kann nun untersucht werden, welche Cross-Platform-Technologien für die Umsetzung des spezifizierten Prototypen ausgewählt werden sollen.

5.4 Auswahl geeigneter Cross-Platform-Technologien

Die Auswahl der für den Prototypen in Frage kommenden Cross-Platform-Technologien basiert auf der Überlegung, dass aus jeder Gruppe von Technologien der jeweils aussichtsreichste Vertreter untersucht werden soll. Hintergrund ist, dass ex-ante nicht zu erkennen ist, welches Cross-Platform-Prinzip das meiste Potenzial hat, womit eine breite Untersuchung notwendig ist. Für Gruppen, deren Potenzial von vornherein sehr gering erscheint, wird im Sinne vertretbaren Entwicklungsaufwandes kein Prototyp implementiert. Die Gruppen werden im Folgenden untersucht.

Bei den **portierten Desktop-Runtimes** ist Java gar nicht und .NET aufgrund des geringen Cross-Platform-Potenzials kaum geeignet (vgl. Abschnitt 4.3). Somit verbleiben die Flash-Technologien, welche aber ein deutlich eingeschränktes Zukunftspotenzial besitzen (vgl. Abschnitt 4.3.2). Folglich wurde aus dieser Gruppe keine Technologie evaluiert.

Bei den **Technologien für Web Apps** offenbart ein Blick auf die Übersichtstabelle 4.1 auf Seite 42, dass alle Technologien die User-Interface-Erstellung unterstützen, aber nur zwei der vier Technologien Unterstützung für Persistenz besitzen. Von diesen beiden Technologien ist *GwtMobile* ein Hobbyprojekt, während *Sencha Touch* ein kommerzielles Produkt ist, weswegen im Sinne der Investitionssicherheit der Prototyp für diese Gruppe mittels *Sencha Touch* umgesetzt wird.

[2] JavaScript Object Notation, ein leichtgewichtiges, textbasiertes Datenaustauschformat. vgl. `http://www.json.org/`

Bei den **Technologien für Boxed Web Apps** scheidet *FeedHenry Studio* aufgrund der auf Android nicht funktionsfähigen Testanwendung aus (vgl. Abschnitt 4.5). Die verbleibenden Technologien *PhoneGap* und *QuickConnectFamily Hybrid* besitzen ähnliche Features, wobei aufgrund der Produktbeschreibung *QuickConnectFamily Hybrid* als etwas leistungsstärker angesehen werden müsste. Andererseits sind die hinter den Technologien stehenden Hersteller sehr unterschiedlich: *PhoneGap* wurde unlängst vom ressourcen- und einflussstarken Hersteller Adobe aufgekauft und soll in dessen Produktlinie Creative Suite eingehen, während *QuickConnectFamily Hybrid* im Wesentlichen von einer Einzelperson entwickelt wird (vgl. Abschnitt 4.5). Die Wahrscheinlichkeit für aktive Fortentwicklung der Technologie scheint bei *PhoneGap* deutlich höher, weswegen diese Technologie näher untersucht wird.

Bei den **Technologien für hybride Apps** stehen zwei Technologien zur Auswahl. Hier erleichtert der Blick auf die entsprechende Übersichtstabelle (Tabelle 4.3 auf Seite 45) die Entscheidung: *Rhodes* unterstützt fünf von fünf spezifizierten Features, während *Worklight* nur drei Features ganz und ein weiteres teilweise erfüllt. Die Entscheidung fällt damit klar zugunsten von Rhodes aus.

Die Gruppe **Technologien für interpretierte Apps** enthält ebenfalls zwei Technologien. Hier offenbart Tabelle 4.4 auf Seite 46, dass *Titanium Mobile* der geeignetere Kandidat ist, denn das *Corona SDK* bietet keinen Zugriff auf User-Interface-Elemente wie Buttons, Listen und dergleichen. Diese müssten von Hand entwickelt werden, was dem Ziel des geringen Entwicklungsaufwands klar entgegenläuft. Daher wird der Prototyp dieser Gruppe in *Titanium Mobile* implementiert.

Die Gruppe der **gänzlich nativen Apps** schließlich enthält Technologien, mit denen sich Apps in C/C++ programmieren lassen. Da diese Programmiersprachen für deutlich höheren Entwicklungsaufwand bekannt sind [Phi99], wurde aus dieser Gruppe kein Prototyp implementiert.

Es ergeben sich also vier zu implementierende Prototypen. Tabelle 5.1 fasst die Entscheidungen in diesem Abschnitt zusammen.

Technologie-Gruppe	Evaluierte Technologie
Klassische Laufzeitumgebungen	Keine
Web Apps	Sencha Touch
Boxed Web Apps	PhoneGap
Hybride Apps	Rhodes
Interpretierte Apps	Titanium Mobile
Native Apps	Keine

Tabelle 5.1 – Zusammenfassung der zu untersuchenden Technologien

6 Implementierung

Eine detaillierte Beschreibung der Architektur und Funktionsweise aller Technologien ist im Umfang dieser Arbeit nicht möglich und für den weiteren Verlauf der Arbeit auch nicht notwendig. An dieser Stelle wird daher lediglich auf die wesentlichen Merkmale der Technologien und umgesetzten Prototypen eingegangen. Des Weiteren werden Screenshots aller entstandenen Benutzermasken vorgestellt.

6.1 Gegenüberstellung der Technologien

Der besseren Lesbarkeit halber werden im weiteren Verlauf dieser Arbeit nicht mehr die vollständigen Bezeichnungen für die Cross-Platform-Technologien verwendet, sondern jeweils eine Kurzform:

Sencha steht für die Technologie *Sencha Touch* des Herstellers Sencha (vgl. Abschnitt 4.4 auf Seite 41).

PhoneGap steht für die Technologie *PhoneGap* des Herstellers Adobe Systems (vgl. Abschnitt 4.5 auf Seite 43).

Rhodes steht für die Technologie *Rhodes* des Herstellers Rhomobile (vgl. Abschnitt 4.6 auf Seite 44).

Titanium steht für die Technologie *Titanium Mobile* des Herstellers Appcelerator (vgl. Abschnitt 4.7 auf Seite 45).

Tabelle 6.1 fasst die für die Entwicklung wichtigen Eigenschaften der Technologien in einer Übersicht zusammen. Zunächst ist zu bemerken, dass alle Technologien *open source* sind. Bei den unterstützten Features ist zu konstatieren, dass keines der Features von allen Technologien unterstützt wird: Eine Unterstützung für User Interfaces stellen alle Technologien bis auf PhoneGap bereit, welches auf Drittanbieter verweist. Eine vollwertige Unterstützung für Persistenz bieten nur Rhodes und Titanium, während bei Sencha und PhoneGap Browser-bedingte Einschränkungen zu berücksichtigen sind (vgl. Abschnitt 2.6.3). Alle Technologien bis auf Sencha ermöglichen den Zugriff auf die Smartphone-Hardware – was im Rahmen der zu entwickelnden Anwendung allerdings keine Rolle spielt. Rhodes und Titanium stellen zudem eine eigene Entwicklungsumgebung

	Sencha	**PhoneGap**	**Rhodes**	**Titanium**
Technologiegruppe	Web Apps	Boxed Web Apps	Hybride Apps	Native Apps
Open Source	ja	ja	ja	ja
Verwendete Version	Version 1.1.0	Version 1.2.0	Version 3.1.0	Version 1.7.2
Features				
User Interface	ja	nein	ja	ja
Persistenz	bis 5 MB[1]	bis 5 MB[1]	ja	ja
Hardware-Zugriff	nein	ja	ja	ja
Entwicklungsumgebung	nein	nein	ja	ja
Programmiersprache	JavaScript	HTML/CSS/JS	Ruby und HTML/CSS/JS	JavaScript

[1] Persistenz-Unterstützung im Rahmen der Möglichkeiten des Browsers (vgl. Abschnitt 2.6.3)

Tabelle 6.1 – Überblick über entwicklungsrelevante Eigenschaften der Technologien, in denen ein Prototyp erstellt wurde

bereit, während PhoneGap auf die jeweiligen nativen Entwicklungsumgebungen der Plattformen zurückgreift und Sencha keine eigenen Entwicklungswerkzeuge bereitstellt.

Die von den Technologien unterstützten Programmiersprachen unterscheiden sich wesentlich und sind ebenfalls in der Tabelle angegeben.

6.2 Implementierungsgrundsätze

Vor der Implementierung der vier Prototypen in diesen Technologien sind einige grundsätzliche Überlegungen anzustellen:

Funktionsgleichheit Da die zu implementierenden Prototypen für eine vergleichende Technologiebewertung genutzt werden sollen, ist es im Geiste von *ceteris paribus* notwendig, die Prototypen möglichst „baugleich“ zu erstellen. Der Funktionsumfang wurde im vorigen Kapitel festgelegt und alle implementierten Prototypen setzen ihn vollständig um.

Keine Modifikation der Technologien Die Technologien sind *open source*, was bedeutet, dass es die Möglichkeit gäbe, das grundlegende Verhalten der Technologien zu modifizieren. Da jedoch die *bestehenden* Technologien verglichen werden sollten, wurde von dieser Möglichkeit kein Gebrauch gemacht.

Keine Plug-Ins von Dritten In diesem Sinne wurde bei der Implementierung ebenfalls nicht auf Plug-Ins von Dritten zurückgegriffen, es sei denn, dies war unabdingbar und explizit vom Technologiehersteller empfohlen worden. Ein Beispiel hierfür ist PhoneGap, welches keine eigene Unterstützung für User Interfaces bietet, sondern zur Verwendung von *Sencha Touch* und ähnlichen Technologien rät.[1]

[1] vgl. `http://phonegap.com/tools`

6.3 Datensynchronisierung und Persistenz

Dieser Abschnitt befasst sich mit dem Datenhandling. Zunächst muss festgestellt werden, dass die Nutzdaten, ihre Übertragung und lokale Speicherung nicht im Zentrum dieser Arbeit stehen. Das Datenhandling wird in diesem Abschnitt daher knapp umrissen.

Das Feinkonzept sieht einen Datenabgleich von der PC-Schlagkartei über einen Webserver zum Smartphone vor (vgl. Abschnitt 5.3.2). Für die Zwecke der Technologiebewertung wurde die Serverseite des Datenabgleichs nicht explizit implementiert, sondern durch statische, auf einem Webserver hinterlegte Dateien simuliert. So entsteht aus Sicht des Prototypen die Illusion des Server-Zugriffs, ohne dass der Server schon vor der Technologieauswahl hätte bereitgestellt werden müssen.

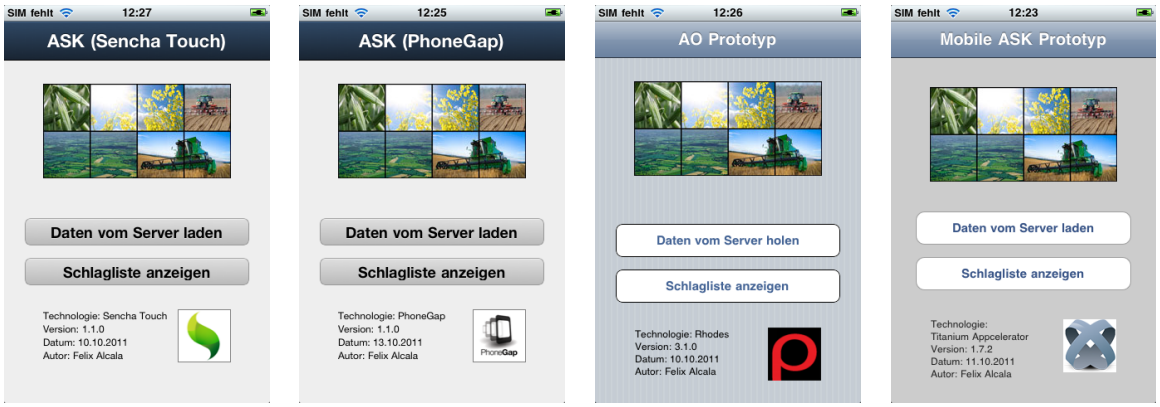
Die Nutzdaten für den Prototypen wurden entsprechend der Empfehlungen in [ATG12] in mehrere abgeschlossene Teilmengen unterteilt. Dabei ist zu beachten, dass bei der Festlegung der Anzahl und Größe der Teilmengen ein Trade-Off notwendig ist: Die im ländlichen Raum omnipräsente Möglichkeit von Verbindungsabbrüchen lässt möglichst viele, kleine Gruppen notwendig erscheinen. Andererseits ist die Latenzzeit beim Aufbauen von HTTP-Verbindungen auf dem Land oft sehr hoch (10 Sekunden sind keine Seltenheit) [ATL12][ATG12], was im Sinne einer performanten Datenübertragung zu möglichst wenigen, großen Datenteilmengen veranlasst.

Im Rahmen des Prototypen wurden die Daten in zwei Teilmengen aufgeteilt: Die Stammdaten für die Schlagliste wurden in einer Datei gespeichert, die Daten für die Schlagdetails in einer weiteren. Beide Dateien befinden sich auf der beiliegenden DVD.

Diese Daten werden vom Prototypen nach dem Herunterladen vom Server für die Offline-Verwendung lokal gespeichert. Dies geschah bei Sencha und PhoneGap mittels der Web-Storage-API (vgl. Abschnitt 2.6.3 auf Seite 23). Bei Rhodes wurde der *Rhodes Object Mapper* genutzt, der Lese- und Schreibzugriff aus der Programmiersprache Ruby ermöglicht. Bei Titanium wurde mangels Object Mapper auf das Dateisystem als Persistenzschicht zurückgegriffen: Die heruntergeladenen Daten wurden schlichtweg als lokale Dateien gespeichert.

6.4 User Interface

Das User Interface wurde entsprechend der Vorgaben aus dem Feinkonzept (vgl. Abschnitt 5.3.1) umgesetzt. Da PhoneGap keine eigene Unterstützung für User Interfaces beeinhaltet, sondern auf Webtechnologien verweist, wurde im Sinne erhöhter Vergleichbarkeit das User Interface des Sencha-Prototypen übernommen. Daher ist das Interface des Sencha-Prototypen identisch mit dem des PhoneGap-Prototypen. Technisch ge-



(a) Sencha (b) PhoneGap (c) Rhodes (d) Titanium

Abbildung 6.1 – Screenshots der Startseiten der Prototypen

(a) Sencha (b) PhoneGap (c) Rhodes (d) Titanium

Abbildung 6.2 – Screenshots der Schlagliste der Prototypen

(a) Sencha (b) PhoneGap (c) Rhodes (d) Titanium

Abbildung 6.3 – Screenshots des Detail-Screens der Prototypen

sehen ist aber PhoneGap eine native, App-Store-fähige und lokal installbare App, während der Sencha-Prototyp nur im Browser des Smartphones verwendet werden kann.

Die User Interfaces der Prototypen sind auf der gegenüberliegenden Seite als Screenshots abgedruckt. Abbildung 6.1 zeigt den Start-Screen der Prototypen, Abbildung 6.2 den Schlaglisten-Screen und Abbildung 6.3 den Detail-Screen. Offenbar ist mit allen Technologien eine optisch einwandfreie Umsetzung einer mobilen Schlagkartei möglich: Alle Screens entsprechen den Vorgaben des Feinkonzepts, was ihre große Ähnlichkeit erklärt.

Neben der offensichtlichen Tatsache, dass die entstandenen Screens sich stark ähneln, gibt es auch eine weniger offensichtliche Tatsache: Wie in den Abschnitten 4.1 und 4.6 (Seiten 37 bzw. 44) beschrieben, unterscheiden sich die Technologien in der Art und Weise, wie das User Interface erzeugt wird.

Sencha (und damit auch PhoneGap) erstellen rein webbasierte Layouts, d.h. alle angezeigten Elemente entstehen aus der webtypischen Kombination von HTML, CSS und Grafiken. Bei Rhodes wird die Titelzeile des Schlaglisten-Screens als natives User-Interface-Element gerendert, die Schlagliste per Webtechnik. Bei Titanium hingegen ist das komplette User Interface aus nativen Elementen zusammengesetzt. Abbildung 6.4 verdeutlicht die unterschiedlichen Ansätze.

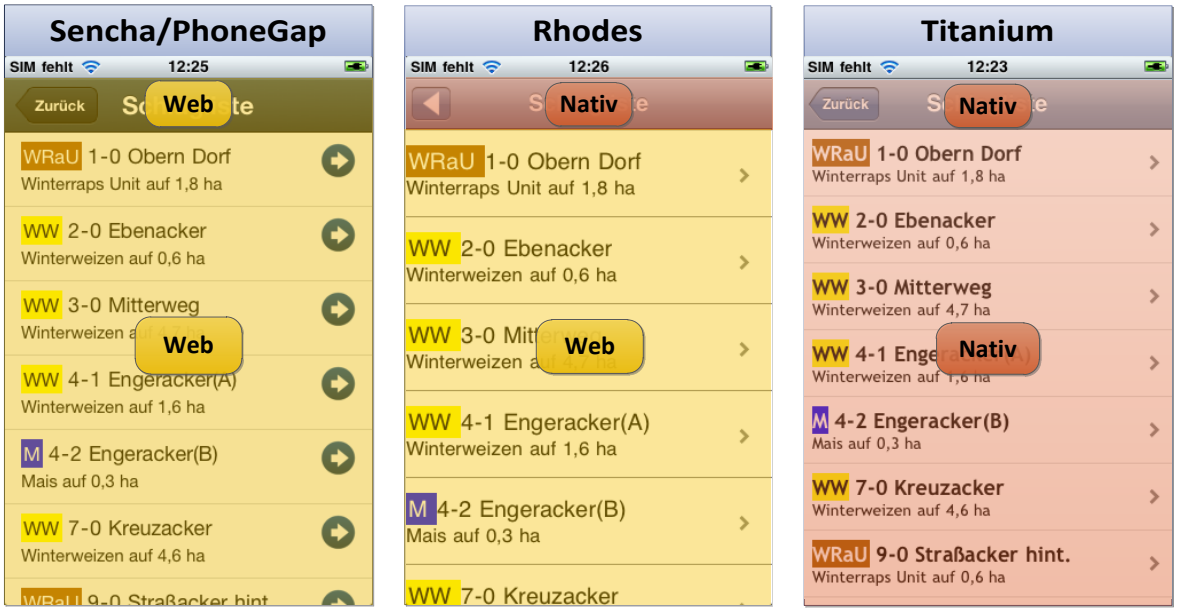


Abbildung 6.4 – Vergleich der Zusammensetzung der User Interfaces aus nativen und webbasierten Komponenten

6.5 Verlauf der Implementierung

Die Implementierung der Prototypen fand von Ende August bis Mitte Oktober 2011 statt. Als Entwicklungsrechner diente ein iMac[2]. Für jede Technologie wurde die zum Beginn der Implementierung aktuelle Softwareversion heruntergeladen und benutzt. Stellte eine Technologie eine Entwicklungsumgebung zur Verfügung, so wurde diese zur Implementierung verwendet. Der Quellcode aller Prototypen findet sich zu Referenzzwecken auf der beiliegenden Daten-DVD.

Im Ergebnis kann festgestellt werden, dass die grundsätzlichen, technischen Anforderungen von allen Technologien erfüllt werden: Alle Technologien sind in der Lage, eine plattformunabhängige App zu erzeugen, welche in nativem Design eine mobile Schlagkartei darstellt. Die Nutzung der verbreiteten Internetformate JSON und HTML ist dabei ebenso möglich wie das präzise Einbinden von Grafiken.

Durch die Implementierung ein und desselben Prototypen in den vier zu untersuchenden Technologien wurde somit die Voraussetzung für die detaillierte Prototyp-basierte Bewertung im nächsten Kapitel geschaffen.

[2] `http://www.apple.com/de/imac/`

7 Bewertung

In diesem Kapitel wird die Leistungsfähigkeit der vier ausgewählten Technologien bewertet. Zunächst müssen hierfür die zu verwendenden Bewertungskriterien festgelegt werden (Abschnitt 7.1); im weiteren Verlauf des Kapitels werden die Kriterien untersucht (Abschnitte 7.2–7.8). Wie in Abschnitt 3.6 beschrieben, müssen dabei Realbefunde ermittelt werden, die anschließend in Teilscores überführt werden. Durch gewichtete Summierung der Teilscores ergibt sich ein Gesamtscore, der in Abschnitt 7.9 errechnet wird.

7.1 Bewertungskriterien

Zur Ermittlung der Bewertungskriterien werden die vom Projektinhaber im Grobkonzept des Prototypen genannten Erfolgsaspekte zu Grunde gelegt (siehe Abschnitt 5.2) und mit der Tabelle möglicher Bewertungskriterien in Abschnitt 3.5 abgeglichen. So entsteht die Liste der hier zu untersuchenden Kriterien.

7.1.1 Unterstützte Plattformen und Offline-Fähigkeit

Im Grobkonzept wird zunächst die Lauffähigkeit auf allen Smartphone-Plattformen gefordert, „mindestens auf iOS und Android“. Daraus ist ersichtlich, dass die Reichweite der gewählten Cross-Platform-Lösung eine wichtige Rolle spielt. Der Kriterientabelle aus Abschnitt 3.5 lassen sich zwei diesbezügliche Kriterien entnehmen: *Unterstützung aktueller Smartphone-Plattformen* und *Unterstützung zukünftiger Smartphone-Plattformen*. Beide Kriterien werden in Abschnitt 7.2 untersucht.

Des Weiteren ist im Grobkonzept eine Datenanzeige „auch ohne bestehende Internetverbindung“ notwendig. Grund ist, dass die mobile Ackerschlagkartei auf dem Feld mit seiner schlechten Mobilfunk-Infrastruktur funktionieren muss. Dieses in der Kriterientabelle als *Offline-Fähigkeit* bezeichnete Technologie-Merkmal wird in Abschnitt 7.3 bewertet.

7.1.2 Entwicklungsaufwand

Auch soll laut Grobkonzept der Entwicklungsaufwand „möglichst gering“ sein. Der Wunsch nach einem geringen Entwicklungsaufwand ist dabei offenbar ein Meta-Kriterium, denn schließlich ist der Zweck von Cross-Platform-Technologien genau der, den Entwicklungsaufwand zu senken. In der Kriterientabelle finden sich folglich zahlreiche mögliche Bewertungskriterien: *Quellcode-Umfang*, *Entwicklungszeit*, *Dokumentation*, *Hersteller-Support*, *Debugging*, *Logging* und viele andere. Diese Kriterien können im Rahmen dieser Arbeit unmöglich alle in der gebotenen Tiefe untersucht werden.

Daher wurde auf ein häufig eingesetztes Instrument zur Abschätzung des Entwicklungsaufwands zurückgegriffen: Die Ermittlung der benötigten Quellcode-Zeilen [BD08], in der Kriterientabelle als *Quellcode-Umfang* aufgeführt. Dieses Instrument lässt sich gut auf die entwickelten Prototypen anwenden und wird in Abschnitt 7.4 angewendet.

7.1.3 Performance

Zuletzt spricht das Grobkonzept von „höchsten Qualitäts- und Performanceansprüchen“ an die Zielsoftware. Da der Begriff *Qualität* viele, oft auch subjektive, Aspekte beinhaltet, wurde in Absprache mit dem Projektinhaber eine Einschränkung auf die Bewertung der Performance vorgenommen.

Für die mobile Ackerschlagkartei ist Performance ein wichtiger Erfolgsfaktor, denn sie muss dem Landwirt einen möglichst schnellen Zugriff auf die benötigten Daten gewährleisten. Ungenügende Performance ist einer der häufigsten Gründe für das Versagen von Software [WV00], und mehr als die Hälfte aller Softwarehersteller sehen sich bei einem oder mehreren ihrer Produkte ernsthaften Performance-Problemen gegenüber [WFP07]. Da die Performance eine dem Softwareprodukt inhärente Eigenschaft ist, beeinflusst sie alle Aspekte der Software [WFP07]. Es wurden drei Experimente durchgeführt:

Im **ersten Experiment** wurde ermittelt, wie lange die App zum Starten benötigt. Hintergrund ist die Tatsache, dass bei web-basierten und interpretierten Cross-Platform-Technologien zunächst der Browser bzw. Interpreter geladen werden muss, bevor die eigentliche Anwendung ausgeführt werden kann. Aus dem Desktopbereich ist diese Verzögerung beim Starten von Java-Anwendungen bekannt [Kaw+07]. Mit dem Ziel, potenzielle Usability-Probleme durch überlange Startzeiten zu erkennen, wurde die Startzeit der Prototypen experimentell untersucht (Abschnitt 7.6).

Im **zweiten Experiment** wurde eine typische Nutzer-Interaktion in der Schlagkartei empirisch untersucht. Da auf Smartphones aufgrund des kleinen Bildschirms die darzustellenden Informationen häufig auf mehrere Screens verteilt werden müssen, spielt die Performance von Screenwechseln bei Smartphone-Apps eine wichtige Rolle. Daher wurde am Prototypen untersucht, wie lange ein typischer Screenwechsel mit lesendem

Datenzugriff dauert. Untersucht wurde hierfür die Zeitspanne, die die App benötigt, um vom Startscreen zur Schlagliste zu wechseln. Abschnitt 7.7 behandelt dieses Experiment.

Im **dritten Experiment** wird die Performance beim Scrollen durch die Schlagliste gemessen. Auslöser für diese Untersuchung war der subjektive Eindruck, dass sich die Schlagliste bei einigen Technologien flüssig scrollen ließ, bei anderen Technologien hingegen nicht. Da die Schlagliste das zentrale User-Interface-Element einer Schlagkartei ist, kommt ihrer Benutzbarkeit eine herausragende Bedeutung zu und wird in Abschnitt 7.8 eingehend analysiert.

In diesem Abschnitt wurden somit sieben Bewertungskriterien festgelegt, die in den nachfolgenden Abschnitten der Reihe nach untersucht werden.

7.2 Bewertung der unterstützten Plattformen

Zunächst ist zu untersuchen, auf welchen Plattformen die durch die Technologien erstellten Apps ausgeführt werden können. Tabelle 7.1 enthält die Realbefunde zu dieser Frage, zum Stichtag 20.11.2011 durch Recherche auf den Webseiten der Hersteller ermittelt. Es ist zu erkennen, dass alle Technologien iOS und Android unterstützen. BlackBerry wird von allen Technologien bis auf Titanium unterstützt, für welches die Titanium-Unterstützung aber bereits angekündigt ist. Rhodes und PhoneGap unterstützen zudem Windows Phone. Darüber hinaus können mit PhoneGap zwei weitere Smartphone-Plattformen erreicht werden.

	Sencha	**PhoneGap**	**Rhodes**	**Titanium**
iOS	ja	ja	ja	ja
Android	ja	ja	ja	ja
BlackBerry	ja	ja	ja	geplant
Windows Phone	–	ja	ja	–
Symbian	–	ja	–	–
Sonstige	–	Bada, WebOS	–	–

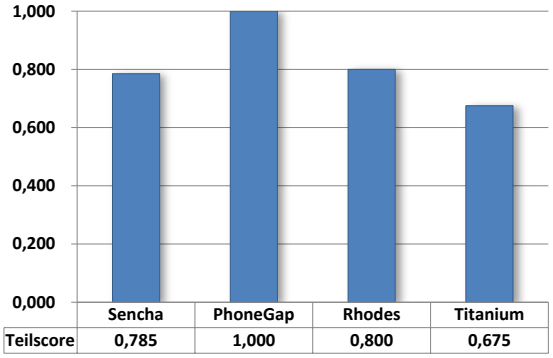
Tabelle 7.1 – Unterstützte Smartphone-Plattformen nach Technologie

Für die Anwendung der Scoring-Methode müssen diese Realbefunde in Teilscores zwischen 0 und 1 überführt werden, wobei 0 den schlechtestmöglichen und 1 den bestmöglichen Wert abbilden soll (vgl. Abschnitt 3.6). Im Falle der unterstützten Smartphone-Plattformen kann der Marktanteil der durch die Technologie unterstützten Plattformen als Teilscore verwendet werden.

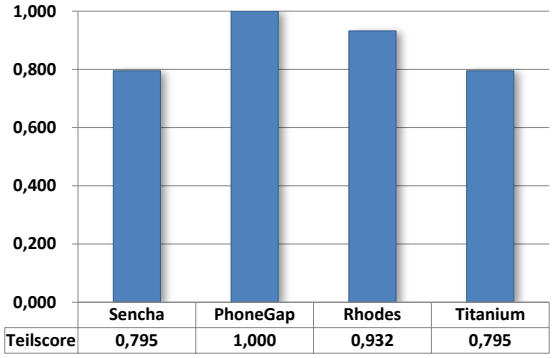
Bei der Ermittlung des Marktanteils wird auf die Zahlen in Abschnitt 2.3 zurückgegriffen: Gegenwärtige Marktanteile sind in Tabelle 2.2 auf Seite 11 zu finden, zukünftige in

der Marktprognose von Gartner in Abbildung 2.2 auf Seite 13. Methodisch sei erwähnt, dass bei den gegenwärtig unterstützten Plattformen geplante Erweiterungen nicht mit eingerechnet wurden, bei den zukünftigen hingegen schon. Da bei den Zahlen von Gartner die Zusammensetzung der sonstigen Plattformen nicht einzeln aufgeschlüsselt ist, wurde PhoneGap für die Unterstützung zweier solcher Plattformen der gesamte Prozentsatz für „Sonstige Plattformen“ zugewiesen. Für die zukünftigen Marktanteile wurden die für das Jahr 2013 prognostizierten Anteile verwendet.

Die sich ergebenden Teilscores finden sich in Abbildung 7.1. Zu erkennen ist, dass durch alle Technologien mehr als zwei Drittel des Smartphone-Marktes abgedeckt werden können. Die geringste Abdeckung bietet gegenwärtig und zukünftig Titanium mit 67,5 % bzw. 79,5 % Marktanteil, während PhoneGap jetzt und in Zukunft so gut wie alle Smartphone-Plattformen erreicht.



(a) Teilscores für *gegenwärtig unterstützte Plattformen* (Q3/2011 lt. [Gar11b])



(b) Teilscores für *zukünftig unterstützte Plattformen* (Prognose 2013 lt. [Gar11a])

Abbildung 7.1 – Teilscores der Technologien hinsichtlich der gegenwärtig und zukünftig unterstützten Plattformen

7.3 Bewertung der Offline-Fähigkeit

Als nächstes wird das Kriterium *Offline-Fähigkeit* bewertet. Zunächst kann festgestellt werden, dass alle vier untersuchten Technologien in der Lage sind, offline-fähige Apps zu erstellen, wenn auch in unterschiedlichem Ausmaß.

PhoneGap, Rhodes und Titanium erzeugen native Apps, die sich wie andere Apps auch regulär auf dem Smartphone installieren lassen. Diese Apps besitzen daher alle Möglichkeiten, die anderen Apps auch offenstehen, und sind somit uneingeschränkt offline-fähig. Diese drei Technologien erhalten daher den Teilscore 1 für die Offline-Fähigkeit.

Sencha hingegen muss als Web App die Offline-Fähigkeit durch Verwendung der Spezifikation *HTML5 Web Applications* erreichen. Wie in Abschnitt 2.6.2 beschrieben,

sind solche Apps durch die Mobilbrowser in ihrer Größe beschränkt: Maximal 5 MB dürfen alle Ressourcen beanspruchen, was möglicherweise eine zukünftige Einschränkung bedeutet. Der gegenwärtige Prototyp kommt dieser Grenze mit 713 kb gecachten Daten nicht zu nahe.

Problematisch ist allerdings die Tatsache, dass die mobile Schlagkartei unter Sencha nicht mehr offline nutzbar ist, wenn der Nutzer den Cache seines Browsers leert.[1] Insbesondere der im Bewusstsein eines regulären Anwenders vermutlich fehlende Zusammenhang (vgl. [Spo01]) zwischen „Die mobile Schlagkartei geht nicht mehr“ und „Ich habe den Browser-Cache geleert“ muss als kritisch beurteilt werden, da dies zu deutlich erhöhtem Supportaufkommen und verringerter Nutzerzufriedenheit führen könnte.

Als Teilscore für die Offline-Fähigkeit kann Sencha daher nicht die volle Punktzahl erhalten. Zwar sind Sencha-Apps prinzipiell offline nutzbar, aber nur mit den oben beschriebenen Einschränkungen. Daher wurde Sencha für die Offline-Fähigkeit der Teilscore 0,5 zugewiesen, was die Mitte zwischen „gar nicht offline-fähig“ (0) und „vollständig offline-fähig“ (1) abbildet.

Abbildung 7.2 zeigt zusammenfassend die in diesem Abschnitt vergebenen Teilscores in einem Balkendiagramm.

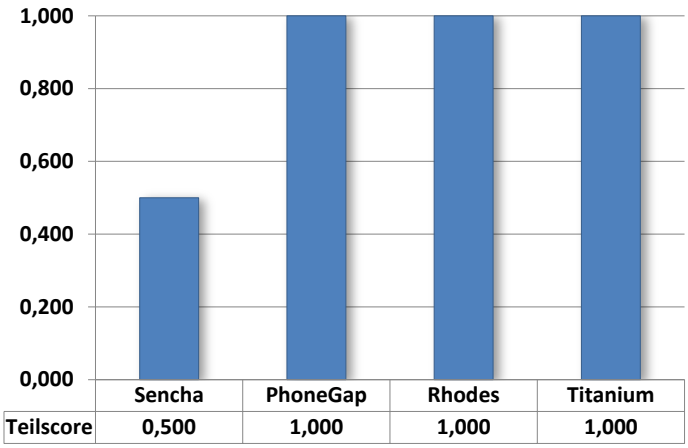


Abbildung 7.2 – Teilscores der Technologien beim Kriterium „Offline-Fähigkeit“

7.4 Bewertung des Quellcode-Umfangs

Neben der Offline-Fähigkeit der Apps muss auch der für ihre Funktionsfähigkeit notwendige Quellcode-Umfang geprüft werden, was in diesem Abschnitt geschieht. In der Literatur wird dieser Umfang typischerweise als *Lines of Code (LOC)* oder *Source Lines*

[1] Bei einem Versuch auf den beiden in Abschnitt 7.5.1 vorgestellten Testgeräten konnte die Anwendung im Offline-Modus nach Leeren des Browser-Caches nicht mehr gestartet werden.

of Code (SLOC) bezeichnet [BD08]. Einige grundlegende Informationen hierzu werden im nächsten Abschnitt gegeben.

7.4.1 Hintergrundinformationen zur SLOC-Zählung

Die Methode der SLOC-Zählung ist wissenschaftlich vielfältig untersucht worden [BD08][Ngu+07]. Das Kernproblem der Methode ist dasjenige, als wie viele logische Code-Zeilen die vorhandenen physischen Code-Zeilen gewertet werden sollen [Ngu+07]. Listing 7.1 verdeutlicht diese Problematik:

```
1 if (x > 0) {
2     alert("x ist größer Null");
3 }
4
5 // andere Schreibweise
6 if (x > 0) alert("x ist größer Null");
```

Listing 7.1 – Problem bei der SLOC-Zählung: Als wie viele Zeilen sollen die beiden Ausdrücke gewertet werden? (vgl. [Ngu+07])

Es ist offenbar, dass beide Code-Schreibweisen die gleiche Logik abbilden und daher auch den gleichen Entwicklungsaufwand benötigen. Da der SLOC-Wert als Indikator für den notwendigen Entwicklungsaufwand verwendet werden soll, sollten beide Schreibweisen einen gleich hohen SLOC-Wert erhalten. Dennoch ermittelt die SLOC-Zählung für die obere Schreibweise den Wert 3 und für die untere Schreibweise den Wert 1, was implizieren würde, dass der obere Code dreimal so aufwändig zu erstellen wäre wie der untere.

Da ähnliche Probleme auch bei der Zählung von Leerzeilen und Kommentaren auftreten, wurde der Quellcode aller Prototypen vor der SLOC-Zählung aufbereitet.

7.4.2 Aufbereitung des Quellcodes für die SLOC-Zählung

Das Ziel der Aufbereitung war, den Quellcode-Umfang der verschiedenen Technologien vergleichbar zu machen, genauer gesagt für die gleiche Logik auch eine annähernd gleiche Anzahl von Zeilen zu erhalten. Hierfür wurde der Code wie folgt umgeschrieben: `if`-Anweisungen und ähnliche Ausdrücke wurden Prototyp-übergreifend stets mit der gleichen Formatierung versehen. Bei HTML- und CSS-Elementen wurde jedes Attribut in eine eigene Zeile geschrieben. Des Weiteren wurden sämtliche für den funktionalen Programmablauf unnötigen Befehle entfernt, z.B. Log-Befehle. Durch diese Änderungen wurde der Quellcode teilweise nicht mehr funktionsfähig, weswegen sie in einer separaten Quellcode-Kopie durchgeführt wurden. Diese befindet sich auf der beiliegenden DVD.

Kommentare wurden abweichend von der gängigen SLOC-Zählweise zum Teil beibehalten. [BD08] vertritt die Auffassung, dass Kommentare nur zur besseren Lesbarkeit

des Codes dienen und daher nicht im SLOC-Wert berücksichtigt werden sollten. Dies trifft zweifelsohne auf diejenigen Kommentare zu, die den Inhalt des Quelltextes lediglich paraphrasieren [Mar09]. Solche Kommentare wurden bei der Überarbeitung für die Zählung manuell gelöscht. Andererseits gibt es Kommentare, die zur Verständlichkeit des Quelltextes unbedingt notwendig sind; diese sind Bestandteil des Quelltextes [Mar09] und wurden folglich beibehalten. Die Listings 7.2 und 7.3 stellen beide Arten von Kommentaren gegenüber.

```
1 // Zurück-Knopf anzeigen
2 @show_back_button = true
```

Listing 7.2 – Beispiel für einen unnötigen Kommentar (aus dem Rhodes-Prototypen)

```
1 // Diese Zeile sorgt dafür, dass der BACK-Button auf Android zur
2 // vorhergehenden Seite führt.
3 // vgl. http://developer.appcelerator.com/question/2731/adding-a-
     window-to-the-stack
4 window.fullscreen = false;
```

Listing 7.3 – Beispiel für einen notwendigen Kommentar (aus dem Titanium-Prototypen)

Der Kommentar in Listing 7.2 wurde gelöscht, denn er enthält keine über den eigentlichen Code hinausgehenden Informationen. Der erläuternde Kommentar in Listing 7.3 hingegen ist für das Verständnis von Code-Zeile 4 unabdingbar: Dass der Befehl `window.fullscreen = false` dazu dient, den Back-Button unter Android funktionsfähig zu machen, wäre ohne den Kommentar kaum ersichtlich. Der obere Code erhält somit den SLOC-Wert 1, während der untere, deutlich schwerer zu erstellende Code (er bedurfte immerhin einer Internetrecherche) den SLOC-Wert 4 erhält.

Nachdem durch diese Überarbeitung des Quelltextes eine Vergleichbarkeit der SLOC-Werte zu erreichen versucht wurde, konnten die Realbefunde dieses Kriteriums ermittelt werden.

7.4.3 Realbefunde

Bei der Zählung der Code-Zeilen wurden Dateien, die zur Technologie gehören, nicht berücksichtigt. Dasselbe gilt für Dateien, die zwar Teil des Prototypen-Quelltextes sind, aber nicht durch den Entwickler bearbeitet werden mussten. Leerzeilen wurden nicht gezählt.

Im Schnitt wurden auf diese Art 577 Code-Zeilen je Prototyp gezählt. Dabei kommt Titanium mit dem wenigsten Quellcode aus – der Prototyp enthält 439 Code-Zeilen. Kaum mehr Quellcode wird für den Rhodes-Prototypen benötigt, wo 469 Zeilen gezählt wurden. Die Webtechnologie Sencha benötigt mit 587 Zeilen deutlich mehr Quellcode

und liegt damit leicht über dem Durchschnitt. Am meisten Quellcode benötigt indes PhoneGap mit 733 Quellcode-Zeilen. Abbildung 7.3 visualisiert diese Realbefunde.

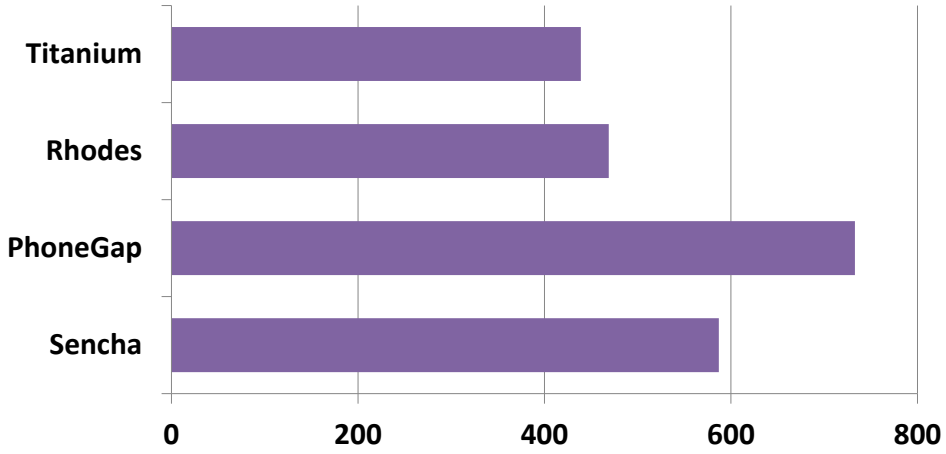


Abbildung 7.3 – Quellcodezeilen der Prototypen nach Technologie

7.4.4 Teilscoring

Bei der Umwandlung der Realbefunde in Teilscores ist zunächst festzustellen, dass weniger Code-Zeilen ein besseres Ergebnis darstellen, da sie weniger Entwicklungsaufwand bedeuten. Unter der Annahme, dass die doppelte Menge an Quellcode einem verdoppelten Entwicklungsaufwand entspricht, werden hierfür halb so viele Teilscore-Punkte vergeben. Es findet also eine lineare, indirekt proportionale Übertragung in Teilscores statt.

Der Teilscore 1, also der bestmögliche Wert, wird für 0 Zeilen Quellcode vergeben. Am anderen Ende der Scoring-Skala soll der Teilscore 0 für diejenige Menge an Quellcode vergeben werden, die unter völligem Verzicht auf Cross-Platform-Programmierung benötigt würde. Diese Zahl ist unbekannt, da keine nativen Prototypen erstellt wurden. Approximierend wird angenommen, der Aufwand entspreche dem doppelten des hier ermittelten Durchschnitts von 577 Codezeilen, womit der Wert 1154 Zeilen dem Teilscore 0 entspricht. Abbildung 7.4 gibt die Teilscores numerisch und grafisch wider.

7.5 Vorbemerkungen zur Performance-Messung

Zuletzt muss die Performance des Prototypen untersucht werden. In diesem Abschnitt werden hierfür notwendige Grundlagen erläutert.

7.5.1 Test-Hardware

Die drei im Folgenden beschriebenen Experimente wurden mittels zweier vom Projektinhaber zur Verfügung gestellter Testgeräte durchgeführt. Für die Plattform iOS war

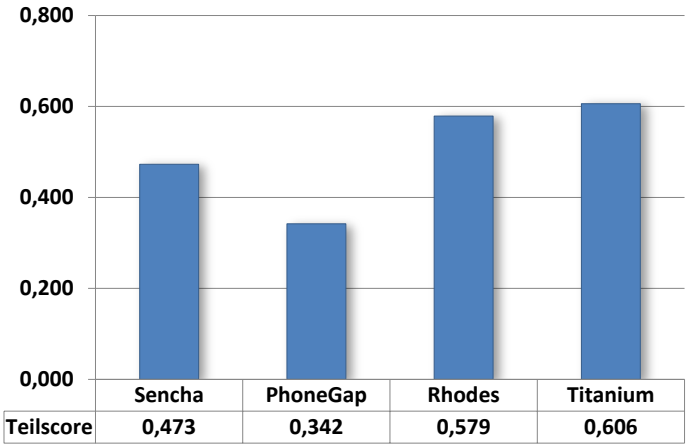


Abbildung 7.4 – Teilscores der Technologien beim Kriterium „Quellcode-Umfang“

dies ein iPhone 3G, für die Plattform Android ein Samsung Galaxy Ace. In Tabelle 7.2 sind wichtige Eckdaten der Testhardware angegeben. Es ist festzustellen, dass das im Jahr 2008 eingeführte iPhone 3G nicht mehr dem aktuellen Stand der Technik entspricht: Es arbeitet nach Angaben von Wikipedia mit 412 MHz, während das derzeit aktuelle iPhone 4S mit 2x1000 MHz getaktet ist. Da die Experimente aber nicht auf die Bestimmung von Performance-Ergebnissen in absoluten Einheiten abzielen, sondern lediglich zum Vergleich der Technologien untereinander dienen, konnte dieses Gerät nichtsdestotrotz verwendet werden.[2]

	Apple iPhone 3G[2]	**Samsung Galaxy Ace[2]**
Modellbezeichnung	MB497DN	GT-S5830
Betriebssystem	iOS 4.2.1	Android 2.2.1
Bildschirm-Auflösung	320 × 480 Pixel	320 × 480 Pixel
Bildschirmdiagonale	3,5 Zoll	3,5 Zoll
Marktstart	2008	2011
Prozessorgeschwindigkeit	412 MHz	800 MHz

Tabelle 7.2 – Technische Eckdaten der Testgeräte

7.5.2 Methodik

Die Experimente wurden unter gleichbleibenden Bedingungen durchgeführt. Die Smartphones waren bei allen Messungen im Flugzeug-Modus, d.h. WLAN, Bluetooth und Mobilfunk waren inaktiv, um Messverzerrungen durch Hintergrundprozesse zu vermeiden.

[2]Quellen der technischen Daten: Geräte-Einstellungen, iTunes, `http://en.wikipedia.org/wiki/IPhone`, `http://samsung.de/de/Privatkunden/Mobil/Mobiltelefone/Style/samsunggalaxyace/GT-S5830OKADBT/detail.aspx?atab=specifications`, `http://www.heise.de/mobil/handygalerie/apple/iphone-3g-1413/`

Da Performance-Messungen Leistungsmessungen sind und Leistung als Quotient von Arbeit und Zeit definiert ist, müssen sowohl die zu erledigende „Arbeit“ also auch die dafür benötigte Zeit exakt festgelegt bzw. gemessen werden. Die bei den Experimenten durch den Prototypen zu erbringende Aufgabe wurde daher genau bestimmt. Um die Zeiten präzise bestimmen zu können, wurde der Bildschirm der Testgeräte bei der Experiment-Durchführung per Videokamera aufgezeichnet und im Nachhinein ausgewertet. Alle Aufnahmen liegen der Arbeit auf einer DVD bei.

Bei der Auswertung wurde die Videobearbeitungssoftware[3] so eingestellt, dass sie die fortlaufende Nummer eines jeden Videobilds anzeigt. Da die verwendete Kamera[4] stets mit genau 25 Videobildern pro Sekunde filmt, kann durch die Differenzbildung zweier Videobildnummern die zwischen ihnen liegende Zeitspanne auf 0,04 Sekunden genau ermittelt werden, was für die Zwecke dieser Arbeit als hinreichend genau eingestuft wird.

Im Folgenden werden die drei durchgeführten Experimente und die ermittelten Ergebnisse beschrieben.

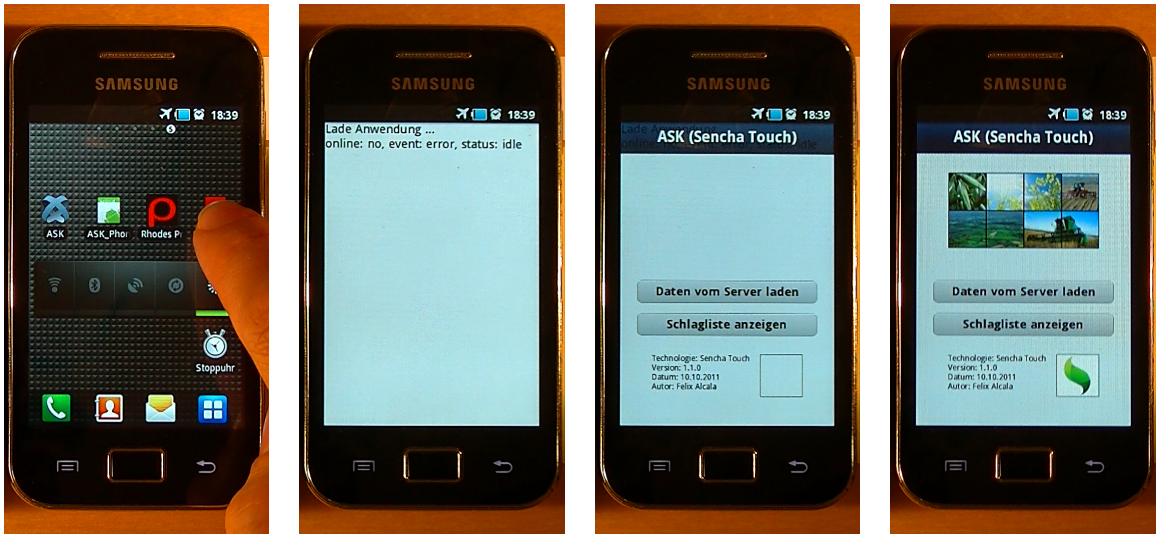
7.6 Bewertung der Performance beim Starten der App

Mit Hilfe des ersten Experiments sollte die durchschnittliche Startzeit von mit den Technologien erstellten Apps ermittelt werden. Hierfür wurden für jede Plattform und jede Technologie jeweils zehn Startzeit-Messungen durchgeführt. Vor jeder Messung wurden eventuell laufende Vorgängerprozesse der Software beendet, um die tatsächliche Startzeit der Software zu ermitteln. Bei der Auswertung des Videomaterials wurde als Startzeitpunkt einer Messung dasjenige Videobild festgelegt, auf welchem der Finger des Experimentierenden den Touchscreen erstmalig berührte. Als letztes Videobild wurde dasjenige gewertet, auf dem die Anwendung vollständig geladen war, also alle auf dem Startscreen benötigten Elemente vollständig sichtbar waren und sich nicht mehr bewegten. Abbildung 7.5 zeigt einen typischen ausgewerteten Ladevorgang.

7.6.1 Empirische Ergebnisse

Die Ergebnisse des Experiments werden als Wertetabelle sowie Boxplot in Abbildung 7.6 (iOS) und Abbildung 7.7 (Android) angegeben. Ermittelt wurde für jede Technologie und Plattform das arithmetische Mittel der zehn Messungen (blauer Balken). Als Indikator für die Streubreite wird zudem die Standardabweichung angegeben, im Boxplot durch die gelbe Box symbolisiert. Der jeweils höchste und tiefste Messwert wird ebenfalls aufgeführt und ist im Diagramm als dünne, schwarze, vertikale Linie angetragen.

[3] Adobe Premiere Pro CS 5.5 – `http://success.adobe.com/de/de/sem/products/premiere.html`
[4] JVC GZ-HM845 – `http://jdl.jvc-europe.com/product.php?id=GZ-HM845BEU&catid=100159`



(a) Start **(b)** Lädt **(c)** Lädt weiter **(d)** Fertig

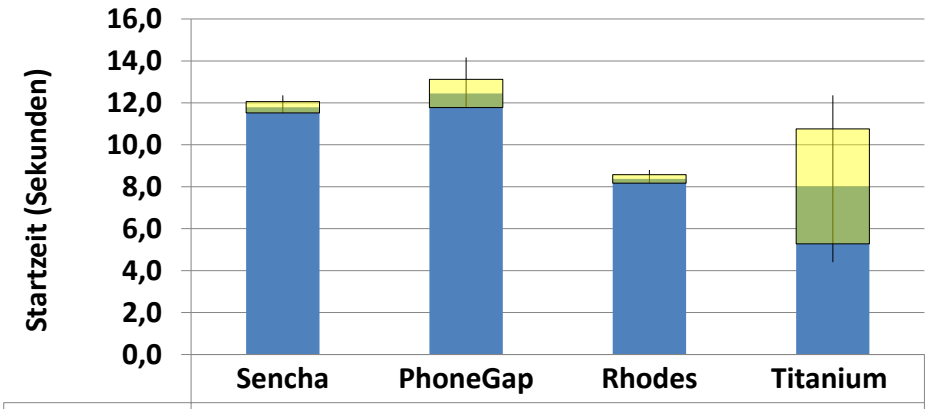
Abbildung 7.5 – Phasen des App-Startvorgangs

Die über alle Technologien gemittelte App-Startzeit ist mit 10,1 Sekunden unter iOS gut doppelt so lang wie mit 4,8 Sekunden unter Android. Dieser Unterschied ist der unterschiedlichen Leistungsfähigkeit der verwendeten Testgeräte zuzuschreiben und sollte nicht dahingehend interpretiert werden, dass die getesteten Technologien unter Android performanter wären.

Die Startzeiten je Technologie sind dabei sehr konstant, was sich in niedrigen Werten bei der Standardabweichung niederschlägt. Nur bei Titanium verhält es sich anders: Dort liegen drei der zehn Messwerte unter 4,9 Sekunden und zwei weitere über 11,3 Sekunden, was die hohe Standardabweichung zumindest mathematisch erklärt. Was allerdings zu diesem auch in späteren Versuchen bestehenden Verhalten führt, konnte nicht abschließend geklärt werden. Ein Zusammenhang mit dem veralteten Testgerät scheint indes unwahrscheinlich, da informelle Tests auf einem aktuellen iPad 2 mit iOS 5 vergleichbare Ladezeit-Schwankungen zeigten. Eine Erklärung könnte sein, dass trotz Beendens der Anwendung manchmal Teile der App im Arbeitsspeicher des Geräts verbleiben und den nachfolgenden Start beschleunigen. Da dies wiederholt auf unterschiedlichen Geräten beobachtet wurde, wird die Messung als realistisches Abbild der zu erwartenden Startzeit angesehen.

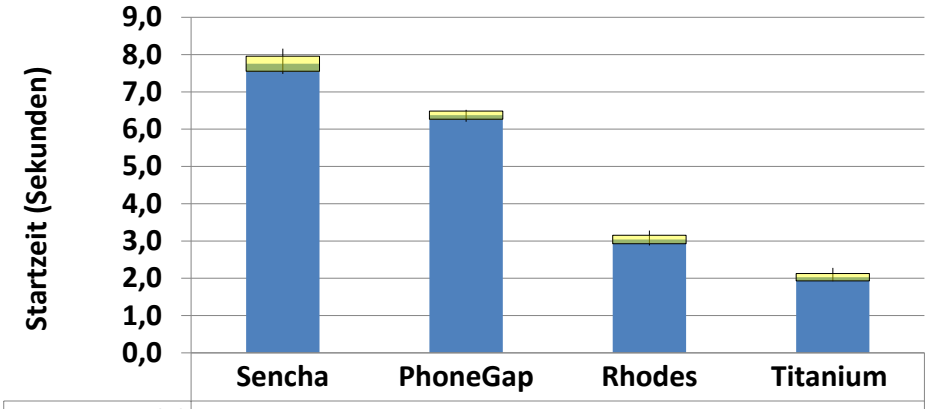
7.6.2 Scoring

Die ermittelten Messwerte müssen nun in Teilscores überführt werden. Vergleicht man eine „kurze“ und eine „lange“ Ladezeit, so entspricht die „kurze“ Ladezeit einer höheren Leistung und muss daher einen höheren Teilscore erhalten. Die kürzestmögliche Ladezeit ist offenbar 0 Sekunden und erhält den Teilscore 1. Dahingegen beginnen Software-



	Sencha	PhoneGap	Rhodes	Titanium
Mittelwert ($\bar{x}$)	11,8	12,4	8,4	8,0
$\bar{x}$ - Std. Abw.	11,5	11,8	8,2	5,3
Maximum	12,4	14,2	8,8	12,4
Minimum	11,5	12,0	8,2	4,4
$\bar{x}$ + Std. Abw.	12,1	13,1	8,6	10,8

Abbildung 7.6 – Startzeiten der Apps unter iOS



	Sencha	PhoneGap	Rhodes	Titanium
Mittelwert ($\bar{x}$)	7,76	6,38	3,04	2,03
$\bar{x}$ - Std. Abw.	7,55	6,27	2,93	1,93
Maximum	8,16	6,52	3,28	2,28
Minimum	7,48	6,20	2,88	1,92
$\bar{x}$ + Std. Abw.	7,96	6,48	3,15	2,13

Abbildung 7.7 – Startzeiten der Apps unter Android

Nutzer bei Interaktionsdauern von mehr als 10 Sekunden ihre Aufmerksamkeit von der Anwendung abzuziehen [Nie93], weswegen dem Messwert 10 Sekunden der Teilscore 0 zugeordnet wurde. Diese Marke überschreiten Sencha und PhoneGap unter iOS deutlich, was der langsamen Hardware des iPhone 3G zugeschrieben wird. Da unter iOS im Schnitt in etwa doppelt so lange Startzeiten wie unter Android gemessen wurden, wurde bei iOS der Teilscore 0 erst für Messwerte von 20 Sekunden vergeben. Die Messungen für Android und iOS wurden zu gleichen Teilen gewichtet.

In Formel 7.1 wird die eben beschriebene Bewertungsfunktion mathematisch formuliert. Es wird der Teilscore v_a für die Technologie a durch die in Sekunden angegebenen durchschnittlichen Ladezeiten t_a^{iOS} und $t_a^{android}$ ermittelt.

$$v_a = \frac{1}{2}\left(\frac{20 - t_a^{iOS}}{20} + \frac{10 - t_a^{android}}{10}\right) \tag{7.1}$$

Die Teilscores sind in Abbildung 7.8 aufgeführt. Es zeigt sich, dass Titanium und Rhodes deutlich performantere Anwendungsstarts ermöglichen als Sencha oder PhoneGap.

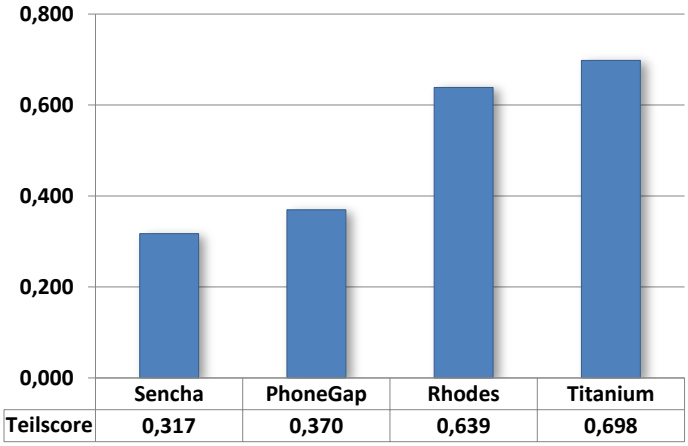
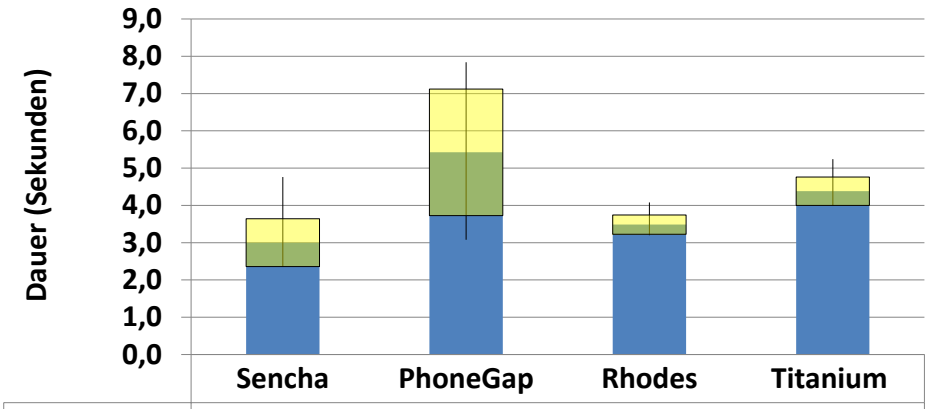


Abbildung 7.8 – Teilscores der Technologien beim Kriterium „Startzeit der Anwendung“

7.7 Bewertung der Performance beim Screen-Wechsel

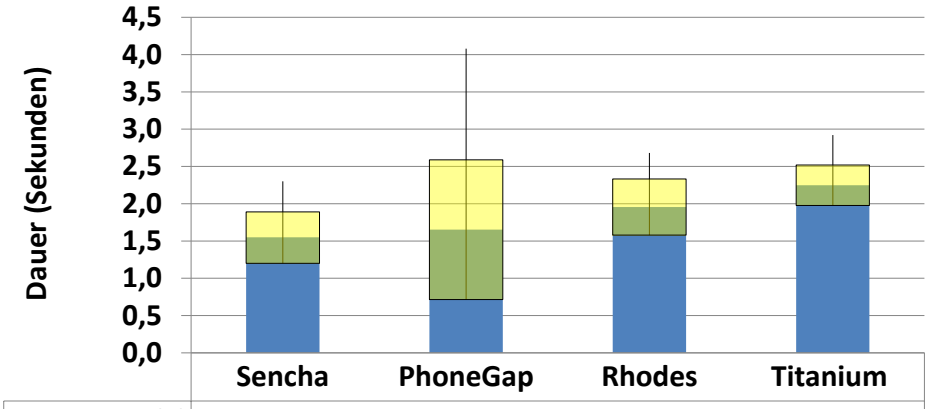
Im nächsten Experiment wurden mit der gleichen Methodik wie eben die Dauern der Screenwechsel untersucht. Genauer ausgedrückt wurde die Zeit zwischen dem Drücken des Buttons „Schlagliste anzeigen“ auf der Startseite bis zum fertigen Laden des Schlaglisten-Screens gemessen. Dieser wurde dann als fertig geladen gewertet, sobald die Schlagliste sich scrollen ließ. Dies geschah, da bei Testnutzungen der Eindruck entstanden war, dass die Schlagliste sich nicht immer sofort scrollen lässt.

Die empirischen Ergebnisse dieses Experiments sind in den Abbildungen 7.9 und 7.10 zusammengefasst. Zu bemerken ist, dass der Screenwechsel mit durchschnittlich 1,7 Se-



	Sencha	PhoneGap	Rhodes	Titanium
Mittelwert ($\bar{x}$)	3,00	5,42	3,48	4,38
$\bar{x}$ - Std. Abw.	2,36	3,73	3,23	4,00
Maximum	4,76	7,84	4,08	5,24
Minimum	2,56	3,08	3,20	4,04
$\bar{x}$ + Std. Abw.	3,64	7,12	3,74	4,76

Abbildung 7.9 – Dauer eines Screenwechsels unter iOS



	Sencha	PhoneGap	Rhodes	Titanium
Mittelwert ($\bar{x}$)	1,55	1,65	1,96	2,25
$\bar{x}$ - Std. Abw.	1,20	0,72	1,58	1,98
Maximum	2,30	4,08	2,68	2,92
Minimum	1,20	1,12	1,68	2,00
$\bar{x}$ + Std. Abw.	1,89	2,59	2,33	2,52

Abbildung 7.10 – Dauer eines Screenwechsels unter Android

kunden schneller abläuft als das Laden der Anwendung, das unter iOS im Mittel 10,1 Sekunden und unter Android durchschnittlich 4,8 Sekunden dauerte. Auch ist festzustellen, dass die Technologien relativ ähnliche Zeitspannen bei dieser Operation benötigen. Einmal geladen, kann die Anwendung also in allen Technologien in etwa gleich schnell von Screen zu Screen wechseln.

Durch Anwendung der gleichen Scoring-Funktion wie oben ergeben sich die in Abbildung 7.11 angegebenen Teilscores, die wie die empirischen Messwerte auch eine geringe Ergebnis-Streubreite aufweisen.

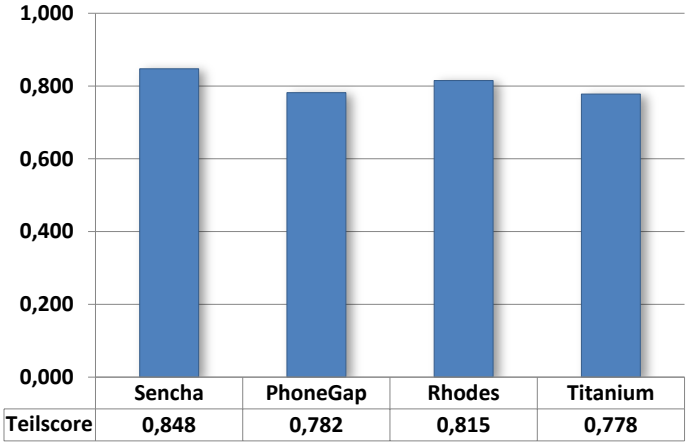


Abbildung 7.11 – Teilscores der Technologien beim Kriterium „Dauer eines Screenwechsels“

7.8 Bewertung der Performance beim Scrollen

Das dritte und letzte Experiment bezüglich der Performance versucht zu ermitteln, wie flüssig das Scrollen durch die Schlagkartei gelingt. In der Literatur wird die Performance von Bildschirmanimationen üblicherweise als so genannte *Framerate* mit der Einheit *frames per second (fps)* gemessen [DCH09][Hul+11]. Gezielte Studien über den Zusammenhang der Benutzbarkeit von betrieblicher Software und der erzielten Framerate konnten nicht ausfindig gemacht werden. Bei empirischen Untersuchungen mit Computerspielen wurde herausgefunden, dass die Höhe der Framerate in direkt proportionalem Zusammenhang mit dem Spielerfolg des Spielers steht [Cha+06]. Daher wird für diese Untersuchung vermutet, dass eine höhere Framerate förderlich für die Benutzbarkeit der Software ist.

7.8.1 Methodik

Keine der Technologien stellt einen Mechanismus zur Ermittlung der erzielten Framerate bereit, weswegen bei diesem Experiment wiederum auf die videobasierte Messung

zurückgegriffen wurde. Die verwendete Videokamera wurde hierfür in einen High-Speed-Aufnahmemodus geschaltet, der mit deutlichen Abstrichen bei der Bildqualität die Aufzeichnung von 125 Videobildern pro Sekunde ermöglicht.

Während des Experiments wurde die Schlagliste mehrfach so schnell wie möglich von oben nach unten und wieder zurück durchscrollt. In der Einzelbildanalyse wurde ermittelt, wie lange jedes gerenderte Frame angezeigt wurde. Dazu wurde gezählt, wie viele Videobilder in Folge das Frame unverändert zu sehen war. Die durchschnittlich über n Frames erreichte Rendering-Framerate f_r kann bei einer Videoaufnahme mit 125 Videobildern pro Sekunde durch die Anzahl der Videobilder x zwischen dem ersten und dem n-ten Frame wie in Formel 7.2 angegeben errechnet werden:

$$f_r = \frac{125 \cdot n}{x} \tag{7.2}$$

Um zu ermitteln, wie viele Videobilder zwischen den einzelnen Frames lagen, wurde bei jedem Framewechsel die laufende Bildnummer des neuen Videobilds notiert. Dies war nicht immer ohne Weiteres möglich, wie Abbildung 7.12 zeigt. Zu sehen sind drei Videobilder eines Frame-Übergangs beim Abwärtsscrollen durch die Schlagkartei.

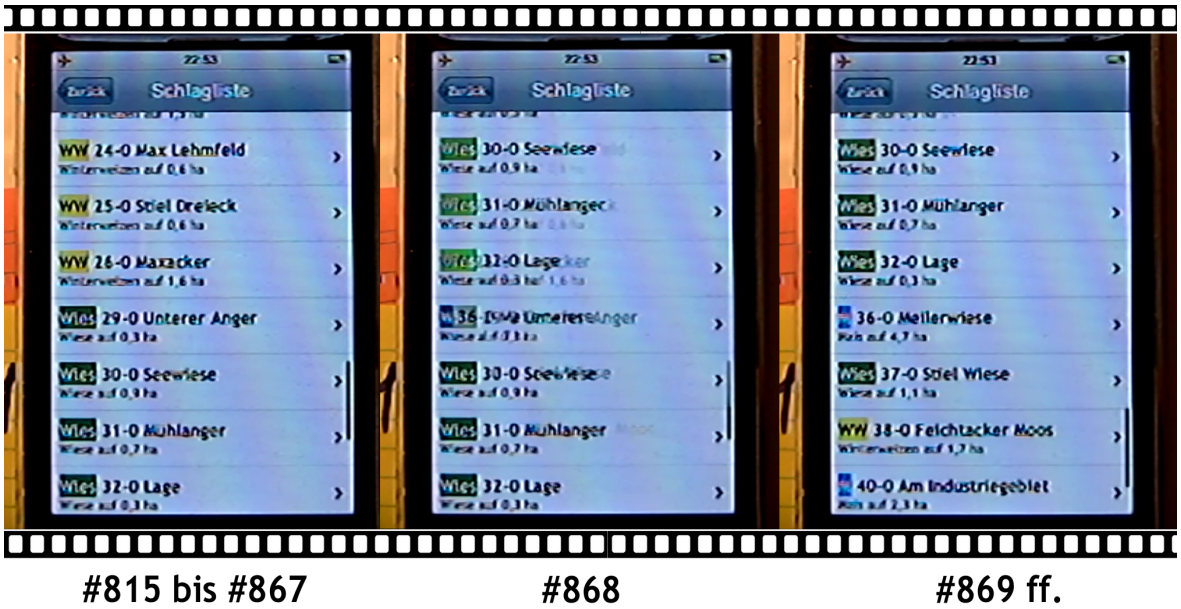


Abbildung 7.12 – High-Speed-Video-Aufnahme eines Frame-Übergangs

Im linken Bild ist ein Frame zu sehen, auf dem in der Schlagliste die Schläge 24-0 bis 32-0 angezeigt werden. Dieses Frame wurde bei Videobild 815 erstmalig gerendert und blieb bis Videobild 867 unverändert. Im rechten Bild ist das nächste durch die Technologie gerenderte Frame zu sehen, welches die Schläge 30-0 bis 40-0 anzeigt. Dieses Frame wurde ab Videobild 869 unverändert angezeigt.

Das mittlere Bild zeigt den Übergang des linken Frames in das rechte Frame: Unten im Bild befindet sich der Schlag 32-0, und zwar an der gleichen Position wie im ursprünglichen, linken Frame. Oben im Bild ist in der ersten Zeile der Schlagliste der Schlag 30-0 zu sehen, was seiner Position im rechten Frame entspricht. In der Mitte dieses Videobilds überlagern sich das linke und das rechte Frame. Diese Beobachtung wurde so gedeutet, dass die Technologie zwar das neue Frame bereits gerendert hatte, dieses aber von der Smartphone-Hardware noch nicht gänzlich dargestellt werden konnte. Das Videobild 868 wurde daher in der Auswertung nicht als eigenständiges Frame gewertet, sondern seinem Nachfolgerframe zugerechnet.

7.8.2 Empirische Ergebnisse

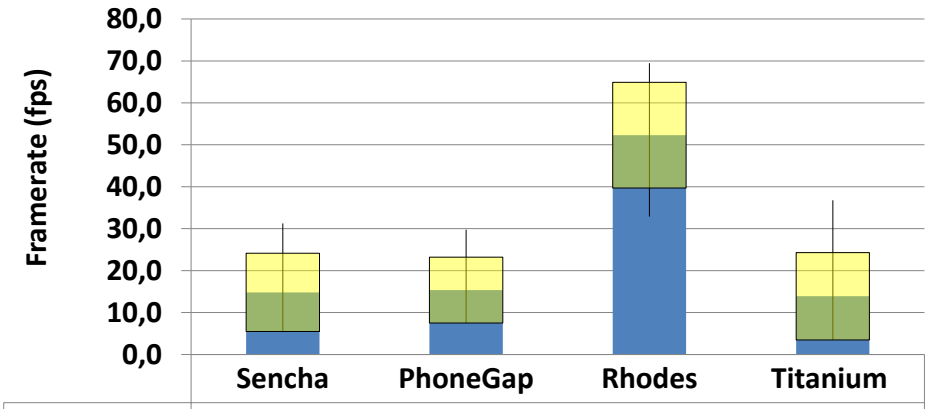
Bei der Auswertung wurden nach obigem Verfahren je Technologie und Plattform die Videobildnummern der ersten einhundert gerenderten Frames ermittelt. Dazu wurden 6220 einzelne Videobilder untersucht. Die Auswertung der Zeitspanne zwischen dem ersten und dem hundertsten Frame ergab je nach Technologie und Plattform Frameraten von 13 bis 109 fps.

Um einen Indikator für die Konstanz der zu erzielenden Framerate zu erhalten, wurden die hundert je Technologie analysierten Frames in zehn Gruppen mit je zehn Frames aufgeteilt. Für diese Gruppen wurde jeweils die erreichte Framerate ermittelt. Der Mittelwert der Gruppen-Frameraten entspricht der Framerate über alle einhundert Frames.[5] Die Abbildungen 7.13 und 7.14 fassen die ermittelten Werte grafisch und numerisch zusammen.

Legt man die im europäischen TV üblichen 25 Einzelbilder je Sekunde als Mindest-Framerate für flüssige Bilder aus, muss man feststellen, dass unter Android nur Rhodes und Titanium mit durchschnittlich 108,7 fps bzw. 49,8 fps dieser Anforderung entsprechen. Auf dem leistungsschwächeren iOS-Testgerät kann nur Rhodes den TV-Maßstab erfüllen, während Titanium dort mit durchschnittlich 13,9 fps deutliche Performance-Probleme zeigt.

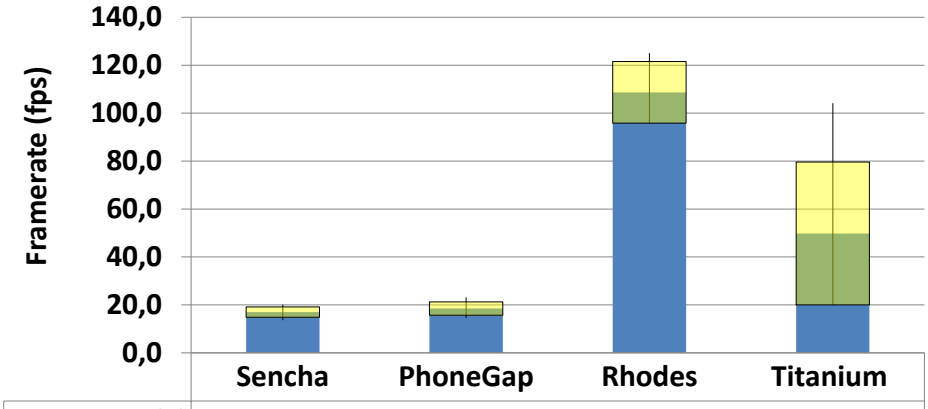
Sencha und PhoneGap können bei der Scroll-Performance nicht überzeugen: Mit durchschnittlichen Frameraten zwischen 14 und 19 fps auf beiden Plattformen gelingt kein flüssiger Scroll-Eindruck, insbesondere auf iOS, wo die Frameraten teils unter 10 fps sinken.

[5] Hierfür ist die Verwendung des *harmonischen Mittels* notwendig, da die erreichte Framerate durch Kehrwertbildung bei den gemessenen Videobildern bestimmt wird. Das harmonische Mittel ist der Kehrwert des arithmetischen Mittels der Kehrwerte der Einzelmessungen [Fah07].



	Sencha	PhoneGap	Rhodes	Titanium
Mittelwert ($\bar{x}$)	14,8	15,4	52,3	13,9
$\bar{x}$ - Std. Abw.	5,5	7,5	39,7	3,5
Minimum	7,1	9,8	32,9	4,2
Maximum	31,3	29,8	69,4	36,8
$\bar{x}$ + Std. Abw.	24,1	23,2	64,9	24,3

Abbildung 7.13 – Frameraten beim Scrollen unter iOS



	Sencha	PhoneGap	Rhodes	Titanium
Mittelwert ($\bar{x}$)	17,0	18,5	108,7	49,8
$\bar{x}$ - Std. Abw.	14,9	15,7	95,9	20,0
Minimum	13,6	14,5	96,2	25,0
Maximum	20,2	23,1	125,0	104,2
$\bar{x}$ + Std. Abw.	19,2	21,3	121,5	79,6

Abbildung 7.14 – Frameraten beim Scrollen unter Android

7.8.3 Überführung in Teilscores

Um die Messergebnisse in Teilscores zu überführen, wurde auf die Erfahrungen bei PC-Spielen in [Cha+06] zurückgegriffen. Dort wurde für Frameraten bis 60 fps empirisch belegt, dass höhere Frameraten die Benutzbarkeit verbessern. Daher wurde der Teilscore 1 für Frameraten von 60 fps und höher vergeben. Der Teilscore 0 wurde für die Framerate 0 fps verwendet.

Das Ergebnis dieses Teilscorings, in das die Messwerte von iOS und Android zu gleichen Teilen eingingen, ist in Abbildung 7.15 abgebildet. Es zeigt sich ein deutlicher Vorsprung für Rhodes, welches auf beiden Plattformen sehr flüssige Scroll-Leistungen erbrachte, gefolgt von Titanium, welches zumindest auf Android gute Ergebnisse liefern konnte. Sencha und PhoneGap hingegen erhalten für die Scroll-Performance geringe Teilscores, da sie auf keiner Plattform flüssiges Scrollen ermöglichen konnten.

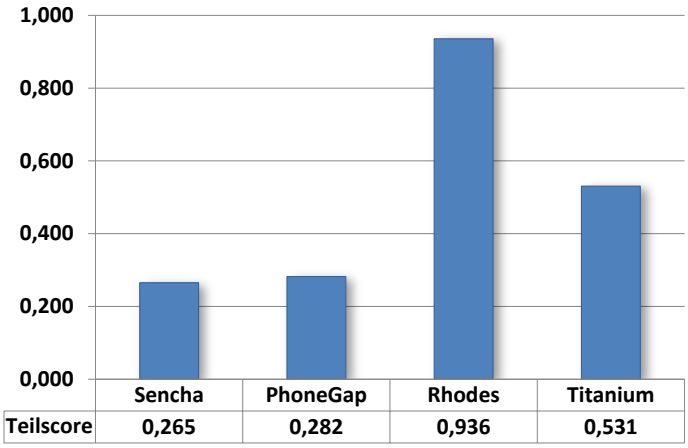


Abbildung 7.15 – Teilscores der Technologien beim Kriterium „Scrollperformance“

Mit diesem letzten Experiment ist die Ermittlung der Teilscores abgeschlossen und es kann das Gesamtergebnis ermittelt werden.

7.9 Gesamtergebnis

Da in dieser Arbeit die Methode des *gewichteten* Scorings angewendet wird, müssen dazu zunächst die zu verwendenden Kriteriengewichte ermittelt werden, wofür das Rank-Order-Centroid-Verfahren (ROC-Verfahren) genutzt wird (vgl. Abschnitt 3.6). Für das ROC-Verfahren müssen die Kriterien nach ihrer Wichtigkeit sortiert werden.

7.9.1 Ermittlung der Kriterien-Rangordnung

In Zusammenarbeit mit dem Projektinhaber wurde folgende natürlichsprachlich formulierte Reihung der Kriterien erarbeitet:

> „Das wichtigste Entscheidungskriterium ist der *Quellcode-Umfang*, der nach den vorliegenden Ergebnissen bei PhoneGap nahezu doppelt so hoch ist wie bei Titanium. Als zweitwichtigstes Kriterium ist die *Scroll-Performance* zu nennen, denn die Schlagliste muss sich unbedingt flüssig und angenehm bedienen lassen.
>
> Drittes Kriterium ist die *Offline-Fähigkeit*, denn diese muss unbedingt gewährleistet sein, und zwar ohne jegliche Abstriche.
>
> Nachdem alle Technologien die Plattformen iOS und Android unterstützen und die restlichen Plattformen keinen besonders großen Marktanteil besitzen, gehen die *gegenwärtig bzw. zukünftig unterstützten Plattformen* an vierter bzw. fünfter Stelle in die Bewertung ein.
>
> Da die App stets nur einmal gestartet werden muss, dann aber im Regelfall mehrere Screenwechsel erfolgen, kommt der *Dauer des Screenwechsels* der sechste Rang zu und der *Startdauer der App* der siebte Rang.“

Das bedeutet, der Projektinhaber reiht die Kriterien wie folgt:

1. Quellcode-Umfang
2. Scroll-Performance
3. Offline-Fähigkeit
4. Gegenwärtig unterstützte Plattformen
5. Zukünftig unterstützte Plattformen
6. Dauer des Screenwechsels
7. Startdauer der App

7.9.2 Ergebnis

„Entscheidend ist, was hinten rauskommt.“ – Helmut Kohl (deutscher Politiker, geb. 1930)[6]

Durch diese Rangordnung kann aus Tabelle 3.2 auf Seite 36 das Gewicht für jedes Kriterium abgelesen werden. Tabelle 7.3 fasst alle ermittelten Teilscores und die ihnen zugewiesenen Gewichte zusammen.

Durch Anwendung der Scoring-Formel auf Seite 35 wurden die finalen Scores ermittelt. Rhodes erhält mit 0,783 den besten Gesamt-Score, gefolgt von Titanium, welches auf 0,679 Score-Punkte kommt. Den dritten Platz belegt PhoneGap mit einem Score von 0,561 Punkten. Als am wenigsten geeignet für die gegebene Aufgabenstellung wird durch

[6] http://de.wikiquote.org/wiki/Helmut_Kohl

	Teilscores				Gewichtung	
	Sencha	**PhoneGap**	**Rhodes**	**Titanium**	**Rang**	**ROC**
Aktuelle Plattformen	0,785	1,000	0,800	0,675	4	0,109
Zukünftige Plattformen	0,795	1,000	0,932	0,795	5	0,073
Quellcode-Umfang	0,473	0,342	0,579	0,606	1	0,370
Offline-Fähigkeit	0,500	1,000	1,000	1,000	3	0,156
Dauer App-Start	0,317	0,370	0,639	0,698	6	0,044
Dauer Screenwechsel	0,848	0,782	0,815	0,778	7	0,020
Scroll-Performance	0,265	0,282	0,936	0,531	2	0,228

Tabelle 7.3 – Zusammenfassung aller Teilscores mit ROC-Gewichten

das Scoring-Verfahren die Technologie Sencha mit lediglich 0,488 Bewertungspunkte eingestuft.

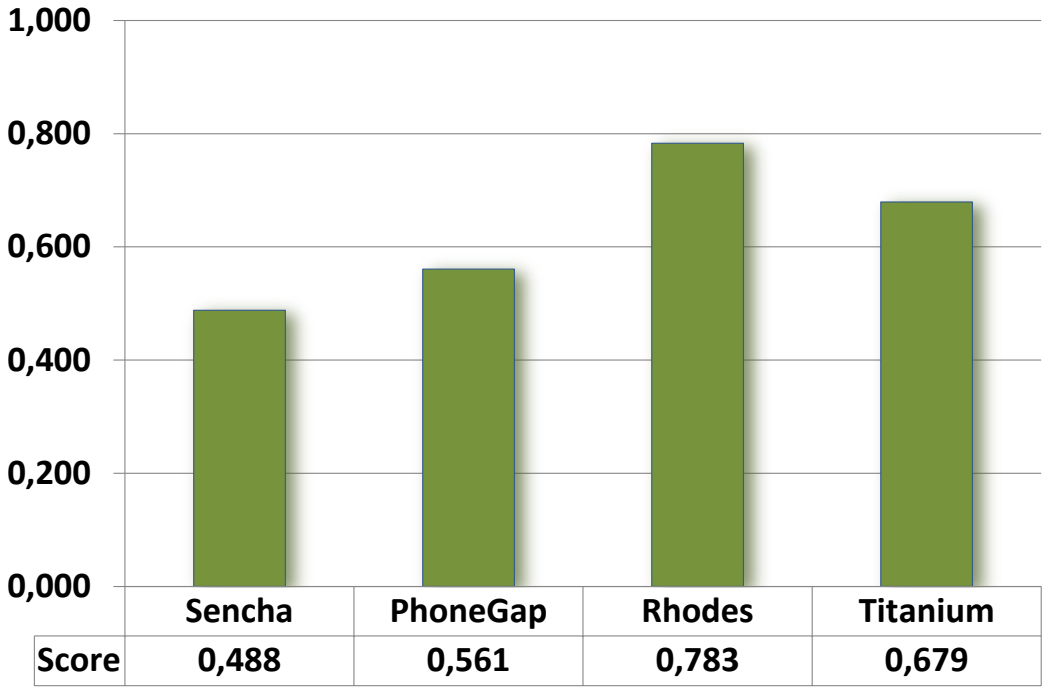


Abbildung 7.16 – Finale Scores nach Technologie

Abbildung 7.16 zeigt die oben genannten Ergebnisse grafisch und beschließt damit die Bewertung der Technologien. Im folgenden Kapitel werden die Ergebnisse interpretiert, wobei auch versucht wird, sie auf andere Aufgabenstellungen zu übertragen.

8 Diskussion der Ergebnisse

Die Aufgabe dieses Kapitels ist es, die Untersuchungsergebnisse des vorigen Kapitels in Beziehung zueinander zu setzen und daraus Schlussfolgerungen zu ziehen. Zunächst wird untersucht, was die Untersuchungsergebnisse bezüglich der Technologieauswahl für die gegebene Aufgabenstellung der PC Agrar GmbH bedeuten (Abschnitt 8.1). Im Anschluss wird versucht, die Ergebnisse auf andere Anwendungsfälle zu übertragen (Abschnitt 8.2).

8.1 Empfehlung für die vorliegende Fragestellung

Untersucht wird in diesem Abschnitt die konkrete Frage der PC Agrar GmbH nach der am besten geeigneten Cross-Platform-Technologie für die Implementierung einer mobilen Schlagkartei. Bei der im vorherigen Kapitel durchgeführten Bewertung erzielt Rhodes den höchsten Gesamt-Score, gefolgt von Titanium und PhoneGap. Dies deutet darauf hin, dass Rhodes die am besten geeignete Alternative sein könnte. Im Folgenden werden die Scoring-Ergebnisse auf ihre Plausibilität überprüft.

> *„Traue keiner Statistik, die du nicht selbst gefälscht hast."*
> – Geflügeltes Wort, gemeinhin Winston Churchill zugeschrieben.[1]

8.1.1 Analyse der Teilscores

Zunächst werden die von den Technologien je Kategorie erzielten Teilscores in einem Netzdiagramm angetragen (Abbildung 8.1), um einen Ergebnis-Überblick zu ermöglichen. Im Diagramm repräsentiert jede der sieben Achsen ein Bewertungskriterium. Der Teilscore 0 ist im Zentrum des Diagramms, der Teilscore 1 ist an der äußersten Netzlinie zu sehen. Das bedeutet: Je weiter außen sich ein Datenpunkt im Diagramm befindet, desto besser wurde die entsprechende Technologie bezüglich dieses Kriteriums bewertet.

Es ist zu sehen, dass – mit Ausnahme der beiden Kriterien bezüglich Plattform-Unterstützung – die rote Linie von Rhodes stets sehr weit außen verläuft: Entweder ist sie ganz außen oder sie ist der äußersten Linie sehr nahe. Das bedeutet, dass Rhodes

[1] http://de.wikipedia.org/wiki/Liste_gefl%C3%BCgelter_Worte/T#Traue_keiner_Statistik.2C_die_du_nicht_selbst_gef.C3.A4lscht_hast.

in allen Kriterien bis auf Plattform-Unterstützung außerordentlich gute Ergebnisse erzielen konnte. Da der Projektinhaber die beiden Kriterien hinsichtlich der Plattform-Unterstützung lediglich auf den Rängen 4 und 5 einstufte, scheint es plausibel, Rhodes zu empfehlen.

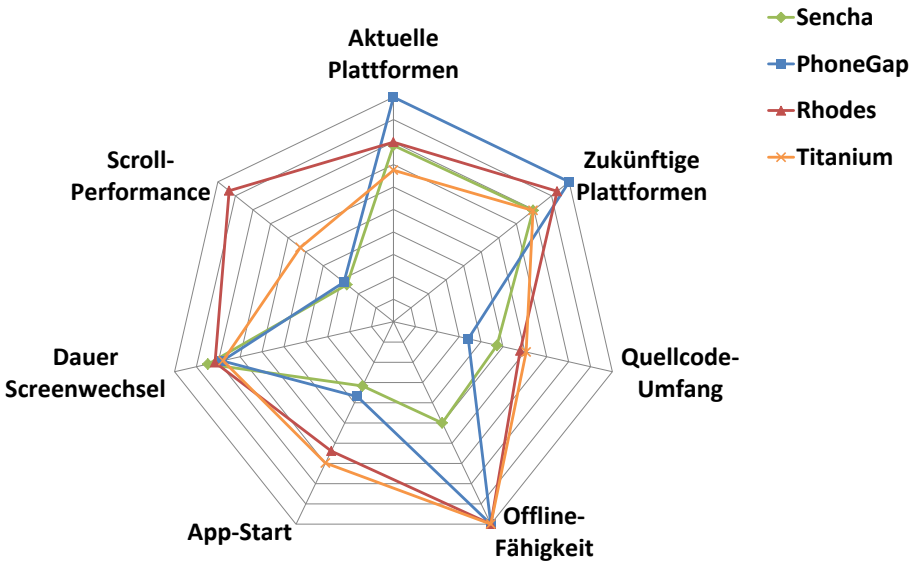


Abbildung 8.1 – Netzdiagramm der Teilscores nach Kriterium und Technologie

8.1.2 Analyse der Gewichtung

Vor einem endgültigen Ergebnis muss aber noch überprüft werden, ob die durch das ROC-Verfahren zugewiesenen Gewichte die Präferenzen des Entscheiders hinreichend widerspiegeln. Um diese Betrachtung zu vereinfachen, werden die sieben Bewertungskriterien in vier Kriteriengruppen zusammengefasst, wie in Tabelle 8.1 angegeben.

Kriterium	Kriteriengruppe
Aktuelle Plattformen	Plattformen
Zukünftige Plattformen	Plattformen
Quellcode-Umfang	Quellcode-Umfang
Offline-Fähigkeit	Offline-Fähigkeit
App-Start	Performance
Dauer Screenwechsel	Performance
Scroll-Performance	Performance

Tabelle 8.1 – Zusammenfassung der Kriterien zu Kriteriengruppen

Die den Kriterien in Abschnitt 7.9 zugewiesenen ROC-Gewichte werden im folgenden Tortendiagramm zu den vier Gruppen zusammengefasst. Das Diagramm in Abbildung 8.2

zeigt, zu welchen Anteilen die Kriterien welcher Gruppe insgesamt in die Bewertung eingehen.

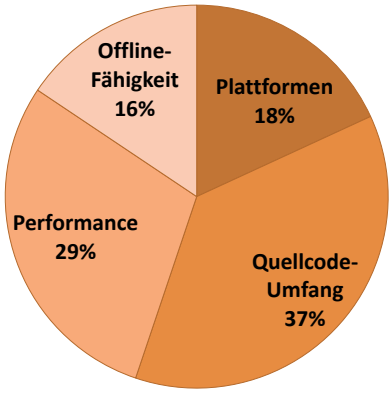


Abbildung 8.2 – Durch den Projektinhaber vorgenommene Kriteriengewichtung nach Kriteriengruppen

Es ist zu prüfen, ob diese Gewichtung plausibel ist. Dazu werden die Gewichte der einzelnen Gruppen nacheinander besprochen.

- Die unterstützten Plattformen werden mit insgesamt 18 % unterdurchschnittlich stark gewichtet. Da durch jede Technologie mehr als zwei Drittel des Smartphone-Markts abgedeckt werden können, besitzt dieses Kriterium keine allzu große Entscheidungsrelevanz, so dass die Gewichtung von 18 % als adäquat eingestuft werden kann.

- Auch die Offline-Fähigkeit ist trotz ihrer Wichtigkeit für den Projekterfolg mit 16 % unterdurchschnittlich stark gewichtet. Da aber alle Technologien offline-fähige Apps erzeugen, kommt auch diesem Kriterium bei der Entscheidung keine übergroße Bedeutung zu. Die Gewichtung von 16 % für die Offline-Fähigkeit ist also angemessen.

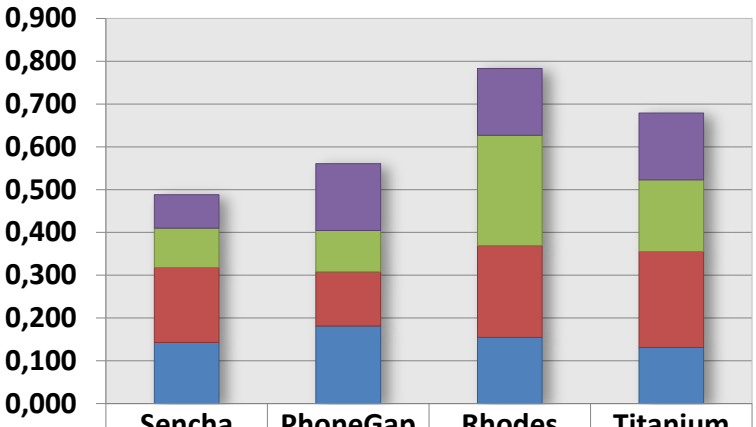
- Beim Quellcode-Umfang zeigten sich deutliche Unterschiede zwischen den Technologien. Da dies das wichtigste Kriterium für den Projektinhaber ist, wird die weit überdurchschnittliche Bewertung von 37 % als treffend angesehen.

- Auch bei der Performance-Messung traten teils gravierende Unterschiede zwischen den Technologien zutage, weswegen die Gewichtung von 29 % als geeignet erscheint.

Insgesamt kann somit festgestellt werden, dass die Gewichtung der Kriterien als plausibel angesehen werden kann.

8.1.3 Analyse der Gesamt-Scores

Abschließend wird die Zusammensetzung der von den Technologien erzielten Gesamt-Scores untersucht. Abbildung 8.3 zeigt in einem gestapelten Balkendiagramm die Gesamt-Scores der Technologien als Balken. Die Balken bestehen jeweils aus vier Blöcken, die abbilden, welcher Anteil des Gesamt-Scores auf welche Kriteriengruppe zurückzuführen ist. Diese Werte sind in der Wertetabelle numerisch angegeben.



	Sencha	PhoneGap	Rhodes	Titanium
Offline-Fähigkeit	0,078	0,156	0,156	0,156
Performance	0,092	0,097	0,258	0,168
Quellcode-Umfang	0,175	0,127	0,214	0,224
Plattformen	0,143	0,181	0,155	0,131

Abbildung 8.3 – Zusammensetzung der Gesamt-Scores durch die gewichteten Teilscores nach Kriteriengruppen

Vergleicht man die Zusammensetzung der Scores von Rhodes und PhoneGap, so ist zu erkennen, dass Rhodes bei den unterstützten Plattformen 0,155 Bewertungspunkte erreicht und PhoneGap 0,181 – was PhoneGap einen leichten Punktevorsprung vor Rhodes verschafft. Allerdings erzielt PhoneGap beim Quellcode-Umfang lediglich 0,127 Bewertungspunkte, Rhodes jedoch 0,214 Punkte, so dass nun Rhodes einen Punktevorsprung aufweist. Bei der Performance wird dieser Trend verstärkt: Hier erreicht PhoneGap 0,097 Punkte, Rhodes aber 0,258. Dies führt dazu, dass Rhodes insgesamt einen deutlich höheren Score als PhoneGap erzielen kann.

Im Vergleich von Titanium und Rhodes ist zu beobachten, dass beide Technologien annähernd gleich viele Bewertungspunkte bei Offline-Fähigkeit, Quellcode-Umfang und den unterstützten Plattformen erzielen. Allerdings kommt Rhodes bei der Performance mit 0,258 Bewertungspunkten auf ein erheblich besseres Ergebnis als Titanium mit 0,168 Punkten, wodurch sich für Rhodes auch im Vergleich zu Titanium ein deutlich höherer Gesamt-Score ergibt.

Die von Sencha erzielten Bewertungspunkte liegen in jeder Kriteriengruppe unter denen von Rhodes.

Es zeigt sich also, dass Rhodes sich gegenüber Titanium insbesondere aufgrund der höheren Performance durchsetzen kann. Gegenüber PhoneGap dominiert Rhodes ebenfalls aufgrund der Performance und zudem auch aufgrund des Quellcode-Umfangs. Gegenüber Sencha kann Rhodes in jeder Kriteriengruppe das bessere Ergebnis erzielen. Auch in dieser Untersuchung scheint es plausibel, Rhodes als die am besten geeignete Alternative anzusehen.

8.1.4 Empfehlung

In keiner der Untersuchungen kamen Zweifel an der Eignung von Rhodes für die vorliegende Aufgabenstellung auf. Daher erscheint die folgende Technologie-Empfehlung als hinreichend gesichert:

Empfehlung *Der Projektinhaber sollte zur Umsetzung seiner mobilen Ackerschlagkartei die Cross-Platform-Technologie* ***Rhodes*** *verwenden.*

8.2 Übertragbarkeit der Bewertungsergebnisse

Im Folgenden wird der Versuch unternommen, die Untersuchungsergebnisse auf andere Anwendungsfälle zu übertragen.

8.2.1 Vorüberlegungen

Für einen Entscheider, der eine Cross-Platform-Technologie auswählen soll, wäre die maximal hilfreiche Information, dass eine bestimmte Technologie den anderen stets in jeder Hinsicht überlegen ist, denn dies würde bedeuten, dass die Entscheidung trivial ist. Eine solche Erkenntnis lässt sich allerdings aus der vorliegenden Arbeit nicht ableiten. Dies ist schon deswegen nicht möglich, weil nicht alle möglichen Kriterien und Technologien untersucht wurden. Die vorliegende Arbeit hat vier Technologien bezüglich sieben Kriterien eingehend untersucht; der Technologie-Überblick zeigt aber 18 Technologien und 60 mögliche Bewertungskriterien (vgl. Kapitel 4 sowie Abschnitt 3.5).

Es stellt sich also zunächst die Frage, ob die Ergebnisse der Arbeit sich auf andere, nicht untersuchte Kriterien oder Technologien übertragen lassen. Eine Übertragbarkeit auf andere Kriterien muss dabei verneint werden. Auch nach eingehendem Studium der Kriterienübersicht ist dem Verfasser der Arbeit kein diesbezüglich gangbarer Weg aufgefallen.

Die Übertragbarkeit auf andere Technologien wird im folgenden Abschnitt untersucht.

8.2.2 Übertragbarkeit auf nicht untersuchte Technologien

Die Übertragbarkeit auf andere Technologien wird in diesem Abschnitt für alle Kriteriengruppen besprochen.

Die Ergebnisse für die *unterstützten Plattformen* lassen sich nicht übertragen, denn jede Technologie unterstützt andere Plattformen. Sie lassen sich aber mit der in Abschnitt 7.2 vorgestellten Methodik auf einfache Weise für andere Technologien ermitteln.

Ergebnis 1 *Die Plattform-Unterstützung kann auch für andere Technologien mit der in Abschnitt 7.2 vorgestellten Methodik ermittelt werden.*

Gänzlich übertragbar sind hingegen die Ergebnisse zur *Offline-Fähigkeit*: Da alle Web Apps denselben Browser-bedingten Einschränkungen unterliegen und alle nativen Apps nicht, unterliegen sämtliche Technologien diesen Einschränkungen. Daher lassen sich die in Abschnitt 7.3 ermittelten Ergebnisse vollständig auf andere Technologien übertragen:

Ergebnis 2 *Technologien, die native Apps erzeugen, sind vollständig offline-fähig.*

Ergebnis 3 *Technologien, die Web Apps erzeugen, sind eingeschränkt offline-fähig.*

Die Ergebnisse zu *Performance* und *Quellcode-Umfang* hingegen lassen sich nicht auf andere Technologien übertragen, denn sie hängen sehr stark von der jeweiligen Funktionsweise der Technologie ab.

Insgesamt ist festzustellen, dass sich die Ergebnisse dieser Arbeit zum Teil auf nicht untersuchte Technologien übertragen lassen.

8.2.3 Übertragbarkeit auf andere Gewichtungen der Kriterien

Es wird nun untersucht, ob unter Beibehalt der zu untersuchenden Kriterien und Technologien eine Verallgemeinerung der Untersuchungsergebnisse möglich ist.

Grundsätzlich kann beim Scoring durch andere Gewichtungen der Kriterien zu abweichenden Ergebnissen gelangt werden. Bei den sieben in dieser Arbeit untersuchten Kriterien können die ROC-Gewichte auf 5040 verschiedene Weisen auf die Kriterien verteilt werden.[2] Es wurde durch erschöpfende Suche ermittelt, welche Technologie durch das Scoring-Verfahren wie oft den höchsten Gesamt-Score erhält. Die Ergebnisse sind in Tabelle 8.2 angegeben.

Es zeigt sich, dass Sencha bei den gegebenen Kriterien niemals als beste Technologie angesehen werden kann. Titanium gewinnt nur bei 2 von 5040 Möglichkeiten, und zwar

[2] $7! = 5040$

	Sencha	PhoneGap	Rhodes	Titanium	Gesamt
Höchster Score	0 Mal	175 Mal	4863 Mal	2 Mal	**5040 Mal**
Anteil	0,00 %	3,50 %	96,50 %	0,00 %	**100,00 %**

Tabelle 8.2 – Scoring-Gewinner bei der erschöpfenden Suche über alle möglichen Kriteriengewichtungen

beide Male mit einem marginalen Vorsprung von 0,0004 bzw. 0,0007 Bewertungspunkten vor dem jeweils zweitplatzierten Rhodes. Daher scheint auch Titanium für die gegebene Aufgabenstellung zumindest niemals die klar bessere Alternative darzustellen. Im Wesentlichen kommen also die Alternativen PhoneGap und Rhodes in Betracht.

Ergebnis 4 *Die Technologie-Entscheidung mit den in Abschnitt 7.1 ausgewählten Bewertungskriterien kann beim gewichteten Scoring mit ROC-Verfahren nicht zur Wahl von Sencha führen.*

Ergebnis 5 *Die Technologie-Entscheidung mit den in Abschnitt 7.1 ausgewählten Bewertungskriterien kann auf die Entscheidung zwischen Rhodes und PhoneGap reduziert werden.*

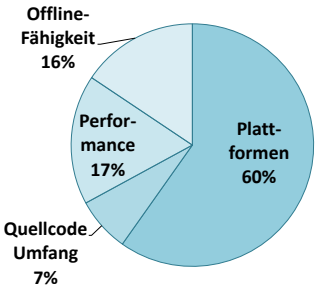
Im Ergebnisvergleich zwischen PhoneGap und Rhodes erzielt PhoneGap bei 175 der 5040 möglichen Kriteriengewichtungen mehr Bewertungspunkte als Rhodes, welches in allen 175 Fällen zweitplatziert ist.

Um eventuelle Gesetzmäßigkeiten ermitteln zu können, wurde untersucht, wie die Kriterien verteilt sein müssen, damit PhoneGap den höchsten Gesamt-Score erhält. Dazu wurde zunächst diejenige Kriteriengewichtung ermittelt, bei der PhoneGap den größten Abstand vor Rhodes erzielt. Dies ist mit 0,044 Bewertungspunkten Vorsprung die in der Fußnote angegebene Reihung.[3] Die Abbildungen 8.4a und 8.4b zeigen die sich ergebende Gewichtung der Kriteriengruppen sowie die Aufschlüsselung der Gesamt-Scores auf Kriteriengruppen.

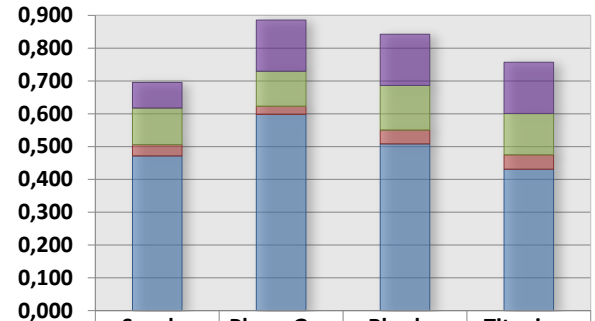
Es zeigt sich, dass die von den Technologien unterstützten Plattformen zu 60 % in das Gesamtergebnis eingehen. Angesichts dieses Ergebnisses kann der Eindruck entstehen, PhoneGap sei genau dann der Vorzug zu geben, wenn eine weitgehende Plattform-Unterstützung maximal wichtig ist. Dieser Eindruck wäre allerdings unrichtig, denn es existieren auch Kriterienreihungen, die dieser Regel entgegenlaufen und Rhodes trotz maximaler Gewichtung der Plattformen als Gewinner sehen.[4]

[3] Reihung: Aktuelle Plattformen, Zukünftige Plattformen, Offline-Fähigkeit, Dauer Screenwechsel, Quellcode-Umfang, Dauer App-Start, Scroll-Performance.
Scoring-Ergebnisse: Sencha: 0,696; PhoneGap: 0,886; Rhodes: 0,842; Titanium: 0,757

[4] Beispielsweise die Reihung: Zukünftige Plattformen, gegenwärtige Plattformen, Quellcode-Umfang, Scroll-Performance, Dauer App-Start, Offline-Fähigkeit, Dauer Screen-Wechsel.
Scoring-Ergebnisse: Sencha: 0,638, PhoneGap: 0,769, Rhodes: 0,827, Titanium: 0,711



(a) Gewichtung der Kriteriengruppen



	Sencha	PhoneGap	Rhodes	Titanium
Offline-Fähigkeit	0,078	0,156	0,156	0,156
Performance	0,111	0,107	0,136	0,126
Quellcode-Umfang	0,034	0,025	0,042	0,044
Plattformen	0,472	0,598	0,508	0,431

(b) Zusammensetzung der Gesamt-Scores

Abbildung 8.4 – Hypothetisches Scoring-Ergebnis mit PhoneGap als bestbewerteter Technologie

Feststellen lässt sich, dass PhoneGap mit maximal 0,044 Bewertungspunkten Vorsprung vor Rhodes gewinnt, Rhodes jedoch mit bis zu 0,202 Punkten deutlich höhere Punkteabstände erzielen kann. Daher ist es jedenfalls nicht völlig unrichtig, Rhodes bei unbekannter Gewichtsverteilung den Vorzug zu geben: Immerhin ist Rhodes in 96,5% der Gewichtungen als beste Alternative ermittelt worden und erzielt in den übrigen Fällen Platz zwei mit nur mäßigem Punkteabstand zur erstplatzierten Technologie.

Ergebnis 6 *Ist die Kriteriengewichtung unbekannt, ist Rhodes die beste Alternative, sofern sich die Entscheidung auf die inAbschnitt 7.1 ausgewählten Bewertungskriterien beschränkt.*

Zusammenfassend lässt sich feststellen, dass sich die anhand der konkreten Problemstellung des Projektinhabers gewonnenen Erkenntnisse teilweise auf andere Problemfälle übertragen lassen.

Damit ist die Diskussion der Untersuchungsergebnisse abgeschlossen. Im folgenden, abschließenden Kapitel wird ein Überblick über die in dieser Arbeit gewonnenen Erkenntnisse und ein Ausblick auf zukünftig notwendige Forschungen gegeben.

9 Zusammenfassung und Ausblick

Diese Arbeit wurde durch ein industrielles Realprojekt ausgelöst: Es sollte die beste Cross-Platform-Technologie für eine konkrete Aufgabenstellung ermittelt werden.

Im Rahmen einer einmonatigen Breiten- und Tiefen-Recherche wurde ein strukturierter, 18 Technologien umfassender Überblick verfügbarer Cross-Platform-Technologien erarbeitet (Kapitel 4). Es ist keine vergleichbare wissenschaftliche Zusammenstellung mobiler Cross-Platform-Technologien bekannt.

Auch wurde durch Recherche und logische Überlegung ein Katalog von 60 möglichen Bewertungskriterien für Cross-Platform-Technologien erstellt (Abschnitt 3.5), welcher in dieser Ausführlichkeit und Konsistenz in der bestehenden Literatur ebenfalls nicht gefunden werden kann.

Um die konkrete Aufgabenstellung des Projektinhabers zu beantworten, wurde in Abschnitt 3.2 eine auf der Literatur basierende 6-Schritte-Methode vorgestellt und im weiteren Verlauf der Arbeit angewendet:

1. Mindestanforderungen für Technologien festlegen (Abschnitt 3.4)
2. Technologieüberblick erstellen (Kapitel 4)
3. Prototyp definieren und die zu untersuchenden Technologien anhand seiner Anforderungen auswählen (Kapitel 5)
4. Prototyp in den verschiedenen Technologien implementieren (Kapitel 6)
5. Technologien anhand der Prototypen mittels einer Evaluierungsmethode untersuchen (Kapitel 7)
6. Ergebnisse der Evaluierung prüfen und Konsequenzen ermitteln (Kapitel 8)

Bei der durchgeführten Prototyp-basierten Bewertung erzielte Rhodes das mit Abstand beste Ergebnis. Die Untersuchungsergebnisse lassen sich teilweise auf andere Technologien und Bewertungskriterien übertragen.

Dennoch sind weitere Untersuchungen notwendig: Es wäre wünschenswert, weitere Kriterien und Technologien zu untersuchen. Zudem sollte geprüft werden, ob Cross-Platform-Entwicklung im Vergleich zur mehrfachen, nativen Implementierung tatsächlich Vorteile beim Entwicklungsaufwand bringt – was hier schlicht vorausgesetzt wurde.

Die vorliegende Arbeit stellt somit einen ersten Grundstein für die wissenschaftliche Betrachtung von Cross-Platform-Entwicklung für mobile Endgeräte dar. Getreu dem

deutschen Sprichwort „Jede Antwort bringt mindestens eine neue Frage hervor“ entstanden weitergehende Forschungsfragen, deren Beantwortung zukünftigen Untersuchungen vorbehalten bleibt.

Literatur

[Ado11a] Adobe Systems Inc. *Day 1: Nitobi joins Adobe*. Hrsg. von Adobe Systems Inc. 2011. URL: http://phonegap.com/2011/10/27/day-1-nitobi-joins-adobe/ (besucht am 20. 11. 2011).

[Ado11b] Adobe Systems Inc. *Verwenden von ADOBE® DREAMWEAVER® CS5 & CS5.5*. Hrsg. von Adobe Systems Inc. 2011.

[AL11] Mehrdad Afshari und Anthony Lambert. *How much RAM is there in an iPhone? - Stack Overflow*. 2011. URL: http://stackoverflow.com/a/371117/92756 (besucht am 11. 12. 2011).

[ATG12] Felix Alcalá Toca und Sebastian Günther. »Mobile Webanwendungen für die Landwirtschaft«. In: *Referate der 32. GIL-Jahrestagung in Weihenstephan 2012 (noch unveröffentlicht)* (2012).

[ATL12] Felix Alcalá Toca und Hans Lecker. »Mobiles Internet auf dem Ackerschlag: Analyse empirischer Langzeitdaten«. In: *Referate der 32. GIL-Jahrestagung in Weihenstephan 2012 (noch unveröffentlicht)* (2012).

[AGL10] Sarah Allen, Vidal Graupera und Lee Lundrigan. *Pro Smartphone cross-platform development: IPhone, BlackBerry, Windows Mobile, and Android Development and Distribution*. New York und NY: Apress, 2010.

[And11] Android Open Source Project. *Supporting Multiple Screens | Android Developers*. Hrsg. von Android Open Source Project. 2011. URL: http://developer.android.com/guide/practices/screens_support.html (besucht am 11. 12. 2011).

[Anu11] Anubavam LLC. *Getting Started With Cross-Platform Mobile Application Development Frameworks*. Hrsg. von Anubavam LLC. 2011. URL: http://www.anubavam.com/blogs/getting-started-cross-platform-mobile-application-development-frameworks (besucht am 01. 12. 2011).

[App10a] Apple Inc. *Java for Mac OS X 10.6 Update 3 and 10.5 Update 8 Release Notes: Cross Platform: Java*. Hrsg. von Apple Inc. 2010. URL: http://developer.apple.com/library/mac/#releasenotes/Java/JavaSnowLeopardUpdate3LeopardUpdate8RN/NewandNoteworthy/NewandNoteworthy.html#//apple_ref/doc/uid/TP40010380-CH4-DontLinkElementID_2 (besucht am 19. 11. 2011).

[App10b] Apple Inc. *Unauthorized modification of iOS has been a major source of instability, disruption of services, and other issues*. Hrsg. von Apple Inc. 2010. URL: http://support.apple.com/kb/ht3743 (besucht am 11. 12. 2011).

[App11a] Apple Inc. *Event Handling Guide for iOS: Data Management: Event Handling*. Cupertino, 2011. URL: http://developer.apple.com/library/ios/documentation/EventHandling/Conceptual/EventHandlingiPhoneOS/EventHandlingiPhoneOS.pdf (besucht am 30. 12. 2011).

[App11b] Apple Inc. *Location Awareness Programming Guide: User Experience*. Hrsg. von Apple Inc. Cupertino, 2011. URL: http://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/LocationAwarenessPG/LocationAwarenessPG.pdf (besucht am 30. 12. 2011).

[Bal08] Heide Balzert. *Basiswissen Web-Programmierung: XHTML, CSS, JavaScript, XML, PHP, JSP, ASP.NET, Ajax*. Korrig. Nachdr. Herdecke: W3L-Verl., 2008.

[BO11] D. Barrera und P. van Oorschot. »Secure Software Installation on Smartphones«. In: *Security Privacy, IEEE* 9.3 (2011), S. 42–48.

[BB96] F.H Barron und B.E Barrett. »Decision quality using ranked attribute weights«. In: *Management Science* (1996), S. 1515–1523.

[Bos11] Bert Bos. *Cascading Style Sheets*. Hrsg. von Bert Bos. 2011. URL: http://www.w3.org/Style/CSS/ (besucht am 17. 11. 2011).

[Bos+11] Bert Bos u. a. *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification: W3C Recommendation 07 June 2011. Section 9.6.1: Fixed positioning*. 2011. URL: http://www.w3.org/TR/CSS2/visuren.html#fixed-positioning (besucht am 17. 11. 2011).

[Bov07] Daniel Bovensiepen. *Das Einsteigerseminar Ruby*. 1. Aufl. Heidelberg: bhv, 2007.

[Bro06] Brockhaus, Hrsg. *Brockhaus Enzyklopädie in 30 Bänden*. 21. Aufl. Leipzig: F. A. Brockhaus, 2006.

[BD08] M. Bundschuh und C. Dekkers. *The It Measurement Compendium: Estimating and Benchmarking Success With Functional Size Measurement*. Springer, 2008.

[Cha+06] Surendar Chandra u. a., Hrsg. *The effects of frame rate and resolution on users playing first person shooter games: Multimedia Computing and Networking 2006*. Bd. 6071. SPIE, 2006.

[CL11] Andre Charland und Brian Leroux. »Mobile application development: web vs. native«. In: *Communications of the ACM* 54.5 (2011), S. 49–53.

[Cho+07] Stephen Chong u. a. »Secure web applications via automatic partitioning«. In: *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*. Stevenson (USA): ACM, 2007, S. 31–44.

[con11] connect. »Bestenliste Smartphones«. In: *connect* 12 (2011), S. 90–95.

[CY99] Michael A. Cusumano und David B. Yoffie. »What Netscape learned from cross-platform software development«. In: *Communications of the ACM* 42 (1999), S. 72–78.

[Der10] Daniel Dern. »Writing Small: Too many platforms can spoil the smartphone app«. In: *IEEE Spectrum* June 2010 (2010), S. 14–15.

[Dew08] Ryan Dewsbury. *Google Web Toolkit Applications*. Upper Saddle River: Prentice Hall, 2008.

[DD02] Reiner Doluschitz und Doluschitz-Spilke. *Agrarinformatik: 5 Tabellen*. Stuttgart (Hohenheim): Ulmer, 2002.

[DCH09] Barry Dwolatzky, Jason Cohen und Scott Hazelhurst, Hrsg. *Proceedings of the 2009 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists*. Vanderbijlpark (South Africa): ACM, 2009.

[Eck10] Christiane Eckl. *Cross-Platform Development für Smartphones: Einsatzmöglichkeiten im Einzelhandel*. Saarbrücken: VDM-Verl. Müller, 2010.

[EB94] Ward Edwards und F. Hutton Barron. »SMARTS and SMARTER: Improved Simple Methods for Multiattribute Utility Measurement«. In: *Organizational Behavior and Human Decision Processes* 60.3 (1994), S. 306–325.

[Ens11] Guilhem Ensuque. *Important Notice - Last OpenPlug Studio Release*. Hrsg. von Alcatel-Lucent. 2011. URL: http://developer.openplug.com/en/community/blog/363-cr15-important-notice (besucht am 16. 12. 2011).

[Fah07] Ludwig Fahrmeir. *Statistik: Der Weg zur Datenanlyse*. 5. Aufl. Heidelberg: Springer, 2007.

[Fet10] Ian Fette. *Gears API Blog: Hello HTML5*. 2010. URL: http://gearsblog.blogspot.com/2010/02/hello-html5.html (besucht am 18. 11. 2011).

[FM08] David Flanagan und Yukihiro Matsumoto. *The Ruby programming language*. 1. Aufl. Beijing u. a.: O'Reilly, 2008.

[FT11] Orrin I. Franko und Timothy F. Tirrell. »Smartphone App Use Among Medical Providers in ACGME Training Programs«. In: *JOURNAL OF MEDICAL SYSTEMS* (2011).

[Gar10] Gartner, Inc. *Gartner Says Android to Become No. 2 Worldwide Mobile Operating System in 2010 and Challenge Symbian for No. 1 Position by 2014*. Stamford, 2010. URL: http://www.gartner.com/it/page.jsp?id=1434613 (besucht am 18. 11. 2011).

[Gar11a] Gartner, Inc. *Gartner Says Android to Command Nearly Half of Worldwide Smartphone Operating System Market by Year-End 2012*. Egham (UK), 2011. URL: http://www.gartner.com/it/page.jsp?id=1622614 (besucht am 18. 11. 2011).

[Gar11b] Gartner, Inc. *Gartner Says Sales of Mobile Devices Grew 5.6 Percent in Third Quarter of 2011; Smartphone Sales Increased 42 Percent: Android OS Rose to Account for More Than 50 Percent of Smartphone Sales*. Egham (UK), 2011. URL: http://www.gartner.com/it/page.jsp?id=1848514 (besucht am 15. 11. 2011).

[Gar11c] Gartner, Inc. *IT Definitions and Glossary*. Hrsg. von Gartner, Inc. 2011. URL: http://www.gartner.com/technology/it-glossary/ (besucht am 12. 11. 2011).

[GE11] D. Gavalas und D. Economou. »Development Platforms for Mobile Applications: Status and Trends: Software, IEEE«. In: *Software, IEEE* 28.1 (2011), S. 77–86.

[Gil11] Zoe Mickley Gillenwater. *Stunning CSS3: A project-based guide to the latest in CSS*. Berkeley: New Riders, 2011.

[GJJ11] Tobias Gräning, Patrick Jean und Fabian Jaxt. *Einstieg in Adobe Flash CS5: Schritt für Schritt zum ersten Flash-Film; neu: Erstellen von iPhone-Apps*. 1. Aufl. Bonn: Galileo Press, 2011.

[Hae10] Chad Haefele. »One Block at a Time: Building a Mobile Site Step by Step«. In: *The Reference Librarian* 52.1-2 (2010), S. 117–127.

[HS11] Sven Haiges und Markus Spiering. *HTML5-Apps für iPhone und Android*. 2. Aufl. Poing b München: Franzis, 2011.

[Hal08] William G. J. Halfond. »Web application modeling for testing and analysis«. In: *Proceedings of the 2008 Foundations of Software Engineering Doctoral Symposium*. Atlanta: ACM, 2008, S. 13–16.

[HKR99] John S. Hammond, Ralph L. Keeney und Howard Raiffa. *Smart choices: A practical guide to making better decisions*. Boston: Harvard Business School Press, 1999.

[Har97] Elliotte Rusty Harold. *Java developer's resource: A tutorial and on-line supplement*. Upper Saddle River: Prentice Hall, 1997.

[Har11] Michael Hartl. *Ruby On Rails 3 Tutorial*. Upper Saddle River und London: Addison-Wesley, 2011.

[Hic10] Ian Hickson. *Web SQL Database: W3C Working Group Note 18 November 2010*. Hrsg. von Ian Hickson. 2010. URL: http://www.w3.org/TR/webdatabase/ (besucht am 17. 11. 2011).

[Hic11a] Ian Hickson. *HTML5: A vocabulary and associated APIs for HTML and XHTML. W3C Working Draft 25 May 2011*. Hrsg. von Ian Hickson. 2011. URL: http://www.w3.org/TR/html5/ (besucht am 13. 11. 2011).

[Hic11b] Ian Hickson. *Web Storage: W3C Working Draft 25 October 2011*. Hrsg. von Ian Hickson. 2011. URL: http://www.w3.org/TR/webstorage/ (besucht am 15. 11. 2011).

[Hog10] Brian P. Hogan. *HTML5 and CSS3: Develop with tomorrow's standards today*. Dallas: Pragmatic Bookshelf, 2010.

[Hul+11] Vedad Hulusić u. a. »Maintaining frame rate perception in interactive environments by exploiting audio-visual cross-modal interaction«. In: *The Visual Computer* 27.1 (2011), S. 57–66.

[Ica11] Miguel de Icaza. *Announcing Xamarin*. 2011. URL: http://tirania.org/blog/archive/2011/May-16.html (besucht am 19. 11. 2011).

[Ier06] Roberto Ierusalimschy. *Programming in Lua*. 2. Aufl. Rio de Janeiro: Lua.org, 2006.

[iph09] iphoneized.com. *18 Mobile Frameworks and Development Tools for Creating iPhone Apps*. Hrsg. von iphoneized.com. 2009. URL: http://iphoneized.com/2009/11/18-mobile-frameworks-development-tools-creating-iphone-apps/ (besucht am 01. 12. 2011).

[JS09] Anil S. Jadhav und Rajendra M. Sonar. »Evaluating and selecting software packages: A review«. In: *Information and Software Technology* 51.3 (2009), S. 555–563.

[Job10] Steve Jobs. *Thoughts on Flash*. 2010. URL: http://www.apple.com/hotnews/thoughts-on-flash/ (besucht am 17. 11. 2011).

[Kah11] Christian Kahle. *Nach Novell-Kauf: Attachmate feuert Mono-Team*. 2011. URL: http://winfuture.de/news,63005.html (besucht am 19. 11. 2011).

[Kaw+07] Kiyokuni Kawachiya u. a. »Cloneable JVM: a new approach to start isolated java applications faster«. In: *Proceedings of the 3rd international conference on Virtual execution environments*. San Diego: ACM, 2007, S. 1–11.

[KP10] Ulla Kirch-Prinz und Peter Prinz. *C++ lernen und professionell anwenden*. 5. Aufl. Heidelberg u. a.: mitp-Verlag, 2010.

[Kir11] George A. Kirk. »There's an app for that, but is there a market for that app? An exploration of the app market as an avenue for entrepreneurship«. In: *Issues in Information Systems* 12 (2011), S. 313–317.

[Koc11a] Stephen G. Kochan. *Programming in Objective-C*. 3. Aufl. Upper Saddle River: Addison-Wesley, 2011.

[Koc11b] Stefan Koch. *JavaScript: Einführung, Programmierung und Referenz*. 6. Aufl. Heidelberg: dpunkt, 2011.

[Kol11] Dirk Koller. *iPhone-Apps entwickeln: Applikationen für iPhone, iPad und iPod touch programmieren*. 2. Aufl. Poing: Franzis, 2011.

[Kün11] Thomas Künneth. *Android 3: Apps entwickeln mit dem Android SDK*. 1. Aufl. Bonn: Galileo Press, 2011.

[Lab11] Lutz Labs. »App-geflogen: Root-Rechte erweitern das Anwendungsangebot«. In: *heise mobil* (2011). URL: http://heise.de/-1187773 (besucht am 11. 12. 2011).

[LS11] Bruce Lawson und Remy Sharp. *Introducing HTML5*. Berkeley: New Riders, 2011.

[Mak09] Ronald Mak. *Writing Compilers and Interpreters: A Software Engineering Approach*. 3. Aufl. Indianapolis: John Wiley & Sons, 2009.

[Mar09] Robert C. Martin. *Clean Code: A handbook of agile software craftsmanship*. Upper Saddle River: Prentice Hall, 2009.

[MGL10] Lance McNearney, Tom Greene und Anthony Lambert. *Is MonoTouch now banned on the iPhone? - Stack Overflow*. Hrsg. von Stack Exchange Inc. 2010. URL: http://stackoverflow.com/questions/2604033/is-monotouch-now-banned-on-the-iphone/2604075#2604075 (besucht am 19. 11. 2011).

[Meh+11] Nikunj Mehta u. a. *Indexed Database API: W3C Working Draft 19 April 2011*. 2011. URL: http://www.w3.org/TR/IndexedDB/ (besucht am 17. 11. 2011).

[MC09] Tom Melamed und Ben Clayton. »A Comparative Evaluation of HTML5 as a Pervasive Media Platform«. In: *Mobile Computing, Applications, and Services*. Hrsg. von Thomas Phan, Rebecca Montanari und Petros Zerfos. Bd. 35. Berlin, Heidelberg und New York: Springer, 2009, S. 307–325.

[Mer11] Christoph Merte. *Marktstrategien im Mobile Banking: Smartphones als neuer Absatzkanal der Finanzindustrie*. Hamburg: Diplomica-Verl, 2011.

[Mot11] Motorola Solutions Inc. *Motorola Solutions Offers Industry's First Framework for Developing HTML5 Applications on Windows Embedded Handheld™ and Windows CE™ Devices*. 2011. URL: http://mediacenter.motorolasolutions.com/Press-Releases/Motorola-Solutions-Offers-Industry-s-First-Framework-for-Developing-HTML5-Applications-on-Windows-Embedded-Handheld-and-Windows-CE-Devices-3741.aspx (besucht am 20. 11. 2011).

[MH10] Luiz Moutinho und Graeme D. Hutcheson. *Dictionary of quantitative management research*. London: SAGE, 2010.

[Ngu+07] Vu Nguyena u. a. »A SLOC Counting Standard.« Diss. Los Angeles: University of Southern California, 2007. URL: http://sunset.usc.edu/events/2007/CIIForum/presentationByDay/fri%20nov2/2%20ToolFair/A%20SLOC%20Counting%20Standard.pdf (besucht am 27. 10. 2011).

[Nie93] Jakob Nielsen. *Usability engineering*. [Updated ed.] San Francisco: Morgan Kaufmann Publishers, 1993.

[ODe10] Jolie O'Dell. *5 Cross-Platform Mobile Development Tools You Should Try*. 2010. URL: http://mashable.com/2010/08/11/cross-platform-mobile-development-tools/ (besucht am 01. 12. 2011).

[Ora10] Oracle Corporation. *Oracle Outlines Plans for Java Platform*. San Francisco, 2010. URL: http://www.oracle.com/us/corporate/press/173712 (besucht am 13. 11. 2011).

[Paa11] T. Paananen. *Smartphone Cross-Platform Frameworks: A case study. (Bachelor's Thesis)*. Hrsg. von Jyväskylän ammattikorkeakoulu. Jyväskylä (Finnland), 2011.

[Phi99] G. Phipps. »Comparing observed bug and productivity rates for Java and C++«. In: *Software Practice and Experience* 29.4 (1999), S. 345–358.

[Pog07] David Pogue. *Ultimate iPhone FAQs List, Part 2*. 2007. URL: http://pogue.blogs.nytimes.com/2007/01/13/ultimate-iphone-faqs-list-part-2/ (besucht am 19. 11. 2011).

[Pre00] L. Prechelt. »An empirical comparison of seven programming languages: Computer«. In: *Computer* 33.10 (2000), S. 23–29.

[Rau04] Johannes Rau. *Grußwort von Bundespräsident Johannes Rau beim Festakt "125 Jahre Albert Einstein"*. Hrsg. von Bundespräsidialamt. 2004. URL: http://www.bundespraesident.de/SharedDocs/Reden/DE/Johannes-Rau/Reden/2004/03/20040314_Rede.html (besucht am 14. 11. 2011).

[Sch10] Reinhard Schiedermeier. *Programmieren mit Java*. 2. Aufl. München [u.a.]: Pearson Studium, 2010.

[SB05] Kathy Sierra und Bert Bates. *Head first Java*. 2. Aufl. Sebastopol: O'Reilly, 2005.

[Spo01] Joel Spolsky. *User interface design for programmers*. Berkeley und New York: Apress, 2001.

[Sta10a] Stack Exchange Inc. *Caching - Max size iPad / iPhone Offline Application Cache - Stack Overflow*. Hrsg. von Stack Exchange Inc. 2010. URL: http://stackoverflow.com/questions/2772908/max-size-ipad-iphone-offline-application-cache (besucht am 17. 11. 2011).

[Sta11a] Stack Exchange Inc. *Calling Objective-C method from C++ method? - Stack Overflow*. Hrsg. von Stack Exchange Inc. 2011. URL: http://stackoverflow.com/questions/1061005/calling-objective-c-method-from-c-method (besucht am 17. 11. 2011).

[Sta11b] Stack Exchange Inc. *How does Corona SDK work? Is Lua converted to Objective C?* 2011. URL: http://stackoverflow.com/questions/4767805/how-does-corona-sdk-work-is-lua-converted-to-objective-c (besucht am 20. 11. 2011).

[Sta10b] Erik Starck. *Mobile Development SDKs compared: MoSync, PhoneGap and AppWhirl*. 2010. URL: http://www.softwaresweden.com/2010/02/16/mobile-development-sdks-compared-mosync-phonegap-and-appwhirl/ (besucht am 01. 12. 2011).

[Str00] Bjarne Stroustrup. *The C++ programming language*. 3. Aufl. Boston und London: Addison-Wesley, 2000.

[Ull11] Christian Ullenboom. *Java ist auch eine Insel: Das umfassende Handbuch*. 10. Aufl. Bonn: Galileo Press, 2011.

[Wag08] Christian Wagenknecht. *Formale Sprachen, abstrakte Automaten und Compiler: Lehr- und Arbeitsbuch für Grundstudium und Fortbildung*. 1. Aufl. Wiesbaden: Vieweg + Teubner in GWV Fachverlage GmbH, 2008.

[Wen10] Christian Wenz. *JavaScript: Das umfassende Handbuch*. 10. Aufl. Bonn: Galileo Press, 2010.

[WG10] Nick Weschkalnies und Sven Gasser. *Adobe Flash CS5: Das umfassende Handbuch*. 1. Aufl. Bonn: Galileo Press, 2010.

[WV00] E. J. Weyuker und F. I. Vokolos. »Experience with performance testing of software systems: issues, an approach, and case study: Software Engineering, IEEE Transactions on«. In: *Software Engineering, IEEE Transactions on* 26.12 (2000), S. 1147–1156.

[Wik11a] Wikipedia. *Mobile application development: Stand vom 01.08.2011*. Hrsg. von Wikipedia. 2011. URL: http://en.wikipedia.org/w/index.php?title=Mobile_application_development&oldid=442367454 (besucht am 01. 12. 2011).

[Wik11b] Wikipedia. *Mobile application development: Stand vom 20.11.2011*. Hrsg. von Wikipedia. 2011. URL: http://en.wikipedia.org/w/index.php?title=Mobile_application_development&oldid=460987406 (besucht am 10. 12. 2011).

[Win11] Danny Winokur. *Flash to Focus on PC Browsing and Mobile Apps; Adobe to More Aggressively Contribute to HTML5*. 2011. URL: http://blogs.adobe.com/conversations/2011/11/flash-focus.html?PID=4003003 (besucht am 17. 11. 2011).

[Wol09] Jürgen Wolf. *C von A bis Z: Das umfassende Handbuch*. 3. Aufl. Bonn: Galileo Press, 2009.

[Won00] Becky Wong-Insley. »System And Method for Cross-Plattform Application Level Power Management«. US 6,131,166. 2000.

[Woo10] David Wood. *Choosing intermediate mobile platforms (dw2blog)*. 2010. URL: http://dw2blog.com/2010/03/04/choosing-intermediate-mobile-platforms/ (besucht am 01. 12. 2011).

[WFP07] M. Woodside, G. Franks und D. C. Petriu. »The Future of Software Performance Engineering: Future of Software Engineering, 2007. FOSE '07«. In: *Future of Software Engineering, 2007. FOSE '07* (2007), S. 171–187.

Anhang A

Nicht weiter betrachtete Technologien

Technologie	Anmerkung
3PMobile / 5o9® EZMobile)[1]	Keine iOS-Unterstützung
AlcheMo[2]	Verwaist (letzter Eintrag unter *News* am 29.06.2010)
App.Co (vormals AppWhirl)[3]	Nischenprodukt: RSS-Reader-Apps; keine Android-Unterstützung; verwaist (letzter Blog-Eintrag „1 year ago“)
Application Craft[4]	Online-Entwicklungsumgebung für jQuery/PhoneGap-Apps; bei Nachrecherche gefunden
appMobi[5]	HTML5-Web-Apps; bei Nachrecherche gefunden
Aqua[6]	Erste Beta-Version erst im September 2011 veröffentlicht
BatteryTech[7]	Spiele-Framework unter C++; bei Nachrecherche gefunden
Canappi[8]	Keine vollwertige Programmierung möglich (Baukasten-System)
Celsius[9]	Verwaist (letztes Release am 25.02.2010)
CoStore[10]	Keine Cross-Platform-Technologie, sondern Daten-Backend für mobile Webseiten; bei Nachrecherche gefunden
CrossMobs[11]	Bei Nachrecherche gefunden
dragonRAD[12]	Keine iOS-Unterstützung
Fivespark[13]	Keine vollwertige Programmierung möglich („no programming skills required“), bei Nachrecherche gefunden
Gideros Mobile[14]	Android-Unterstützung erst seit 20.09.2011

(Fortsetzung auf nächster Seite)

[1] `htttp://www.3pmobile.com/`

[2] `htttp://www.innaworks.com/alchemo-java-me-j2me-to-brew-android-iphone-flash-windows-mobile-cross-compiler`

[3] `htttp://www.app.co`

[4] `htttp://www.applicationcraft.com/`

[5] `htttp://www.appmobi.com/`

[6] `htttp://www.uxplus.com/`

[7] `htttp://www.batterypoweredgames.com/batterytech`

[8] `htttp://www.canappi.com`

[9] `htttp://www.mobile-distillery.com/celsius-2/overview-43.htm`

[10] `htttp://costore.net/`

[11] `htttp://www.crossmobs.com/`

[12] `htttp://www.dragonrad.com`

[13] `htttp://www.fivespark.com/`

[14] `htttp://www.giderosmobile.com/`

Technologie	Anmerkung
gwt-mobile-webkit[15]	Verwaist (letztes Update im April 2011)
iWebKit[16]	Keine Android-Unterstützung
JMango[17]	Nischenprodukt: Datenerfassung per SMS
July Systems Mi^{TM} Platform[18]	Keine vollwertige Programmierung möglich (nur fertige Widgets zusammensetzbar), bei Nachrecherche gefunden
JumpStart Wireless BusinessSuite[19]	Keine Cross-Platform-Technologie, sondern Dienstleistung
Kony[20]	Bei Nachrecherche gefunden
Kyte Mobile[21]	Nischenprodukt: Audio-Apps und Video-Apps
MobiAccess[22]	Bei Nachrecherche gefunden
MobiFlex / ViziApps[23]	Keine vollwertige Programmierung möglich (Baukasten-System)
MobileNationHQ[24]	Keine vollwertige Programmierung möglich, bei Nachrecherche gefunden
Moscrif[25]	Keine iOS-Unterstützung
NeoMAD[26]	Keine iOS-Unterstützung; Java-Cross-Compiler für mobile Anwendungen; bei Nachrecherche gefunden
Particle SDK[27]	Beta beendet; aufgekauft von Appcelerator: wird in Titanium Mobile integriert
Qt[28]	Nur inoffiziell für iPhone und Android verfügbar
Sproutcore[29]	Verwaist (letzte Quellcode-Modifikation am 27.09.2011)
Tiggr Mobile Apps Builder[30]	Online-Entwicklungsumgebung für jQueryMobile/PhoneGap; bei Nachrecherche gefunden
Total Cross / SuperWaba[31]	iPhone nur Jailbroken zu verwenden
WebORB Integration Server[32]	Keine Cross-Platform-Technologie, sondern Technologie zur Integration von Webservern in Apps
Whoop[33]	Keine vollwertige Programmierung möglich (Baukasten-System)
WidgetPad[34]	Online-Entwicklungsumgebung für Web Apps

(Fortsetzung auf nächster Seite)

[15] `htttp://code.google.com/p/gwt-mobile-webkit`
[16] `htttp://snippetspace.com/projects/iwebkit/`
[17] `htttp://jmango.net/`
[18] `htttp://julysystems.com/mi-platform/products/`
[19] `htttp://www.jumpstartwireless.com/`
[20] `htttp://www.kony.com/`
[21] `htttp://www.kyte.com/about`
[22] `htttp://www.mobiaccess.net/`
[23] `htttp://viziapps.com/`
[24] `htttp://www.mobilenationhq.com/`
[25] `htttp://moscrif.com/`
[26] `htttp://neomades.com/`
[27] `htttp://www.particlecode.com/`
[28] `http://qt.nokia.com/products/`
[29] `htttp://www.sproutcore.com`
[30] `htttp://gotiggr.com`
[31] `htttp://www.superwaba.com.br/en/pdas_compativeis.asp`
[32] `htttp://www.themidnightcoders.com/products.html`
[33] `htttp://www.whoop.com/`
[34] `htttp://www.widgetpad.com/`

Technologie	Anmerkung
webMethods Mobile Designer (vormals bedrock)[35]	Nur für Kunden und Partner der Software AG

[35] http://www.softwareag.com/us/products/wm/mobile_computing/mobile_designer/overview/default.asp

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur mit erlaubten Hilfsmitteln angefertigt habe.

Magdeburg, den 31. Dezember 2011

Felix Alcalá Toca