# Plattformabhängige und –unabhängige Entwicklung mobiler Anwendungen am Beispiel von Geo-Wikipedia-App

## Bachelor-Thesis

**zur Erlangung des akademischen Grades B.Sc.**

Benjamin Vehse

2053140

Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Design, Medien und Information
Department Medientechnik

Erstprüfer: Prof. Dr. Andreas Plaß

Zweitprüfer: Prof. Dr. Edmund Weitz

Hamburg, 19. Juli 2014

# Inhaltsverzeichnis


**1 Einleitung** **4**
- 1.1 Beschreibung der mobilen Entwicklungs-Landschaft . . . . . 4
- 1.2 Motivation für das Thema und Zielsetzung . . . . . 7
- 1.3 Gliederung der Arbeit . . . . . 8

**2 Beschreibung von Entwicklungswerkzeugen für mobile Plattformen** **9**
- 2.1 Plattformabhängige Entwicklung und beispielhafte Werkzeuge . . . . . 10
  - 2.1.1 Apple iOS SDK . . . . . 10
  - 2.1.2 Google Android SDK . . . . . 13
- 2.2 Plattformunabhängige Entwicklung und beispielhafte Werkzeuge . . . . 16
  - 2.2.1 Adobe PhoneGap . . . . . 17
  - 2.2.2 Xamarin . . . . . 19
  - 2.2.3 Appcelerator Titanium . . . . . 22
- 2.3 Übersicht der theoretischen Eigenschaften . . . . . 24

**3 Entwicklung einer mobilen Smartphone-Anwendung** **25**
- 3.1 Konzept . . . . . 25
- 3.2 Definition von Anforderungen einer üblichen mobilen App und Entwicklung einer App-Idee . . . . . 26
- 3.3 Entwicklung . . . . . 29
  - 3.3.1 Benutzeroberfläche der Anwendung . . . . . 29
  - 3.3.2 Logik der Anwendung . . . . . 36
  - 3.3.3 Entwicklungs-Ökosystem . . . . . 40
- 3.4 Erkenntnisse . . . . . 46

**4 Vergleich plattformabhängiger und -unabhängiger Entwicklungsframeworks** **49**
- 4.1 Vor- und Nachteile . . . . . 49
  - 4.1.1 für Entwickler . . . . . 49
  - 4.1.2 für Auftraggeber . . . . . 54
  - 4.1.3 für Endkunden . . . . . 55
- 4.2 Definition von Einsatzszenarien für die verschiedenen mobilen Frameworks 56

**5 Fazit und Ausblick** **57**

**Abbildungsverzeichnis** **59**

**Tabellenverzeichnis** **60**

**Codeverzeichnis** **61**

**Literaturverzeichnis** **63**

# Abstract

Motivated by the already high and steadily increasing relevancy of mobile devices, this thesis covers development methods available for mobile applications ("apps"). Specifically the differences between native and cross-platform application development are addressed. Furthermore the advantages and disadvantages of these methods are determined.

In addition to third-party opinions and literature, practical research has been carried out and is used as a basis of valuation. A concept for an example app has been developed and implemented with five different, currently relevant development tools and frameworks. It transpired that none of the candidates is an all-purpose-tool and that instead a suitable framework should be chosen based on functional and visual requirements as well as the available budget.



# Zusammenfassung

Anlässlich der hohen und weiter steigenden Relevanz von mobilen Endgeräten beschäftigt sich diese Arbeit mit den für mobile Anwendungen („Apps“) verfügbaren Entwicklungsmethoden. Hierbei werden speziell die Unterschiede zwischen plattformspezifischer und plattformübergreifender Entwicklung beleuchtet und Vor- und Nachteile dieser Methoden herausgearbeitet.

Neben externen Stimmen und Literatur wird zusätzlich auch eine praktische Untersuchung durchgeführt und als Bewertungsgrundlage herangezogen. In dieser wird ein Konzept für eine beispielhafte App erarbeitet und mit fünf verschiedenen und zum Zeitpunkt der Durchführung relevanten Entwicklungswerkzeugen und -frameworks implementiert. Es stellt sich heraus, dass keiner der Kandidaten ein Allzweckwerkzeug darstellt und ein passendes Framework stattdessen abhängig von funktionalen und visuellen Ansprüchen sowie dem verfügbaren Budget gewählt werden sollte.

# 1 Einleitung

## 1.1 Beschreibung der mobilen Entwicklungs-Landschaft

Die rasante Verbreitung von Smartphones [vgl. Nielsen 2013] im privaten und beruflichen Sektor resultiert im Bereich der Softwareentwicklung im Durchbruch einer neuen Variante von Software - den sogenannten „Apps“. Diese Art Anwendung bündelt meist eine überschaubare Anzahl Funktionen in einer hinsichtlich Bedienung und Leistung für den mobilen Gebrauch optimierten Fassung.

Im Bezug auf die verwendeten mobilen Plattformen, also Betriebssysteme und die sie umgebende Software, hat sich für die Nutzung von Apps bisher kein dominanter Standard herausgebildet [vgl. Wikipedia 2014]. Stattdessen fördern verschiedene Unternehmen jeweils unterschiedliche mobile Plattformen und Betriebssysteme. Hinsichtlich ihrer prozentualen Verbreitung sind hier maßgeblich das mobile Betriebssystem Android der Open Handset Alliance (gegründet und geleitet von Google), Apple iOS sowie Microsoft Windows Phone zu nennen. Diese stellen alle, auf unterschiedliche Programmiersprachen basierende, Entwicklungswerkzeuge (System Developer Kits, SDKs) bereit, die prinzipiell nicht interoperabel sind.

Das Entwickeln einer App auf Basis einer dieser Technologien bezeichnet man als native Entwicklung. Eine nativ für iOS entwickelte App lässt sich beispielsweise grundsätzlich nicht auf einem Telefon oder Tablet mit Android Betriebssystem installieren oder gar ausführen. Vorteilhaft ist, dass eine nativ entwickelte App den Standards der entsprechenden Plattform entspricht. So stellen Plattformbetreiber wie Apple, Google oder Microsoft Referenzdokumente bereit, in welchen die für die Plattform üblichen Gestaltungskonzepte und Interaktionsparadigmen beschrieben und empfohlen werden. Diese sollen dem Endanwender die Interaktion mit neuen Anwendungen erleichtern, indem bekannte Strukturen und vereinheitlichte Bedienkonzepte wiedererkannt und angewendet werden können.

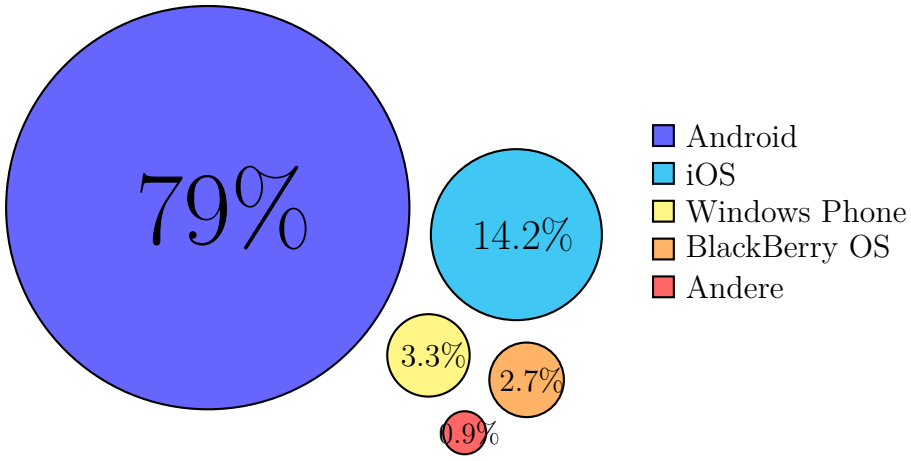


Abbildung 1.1: Prozentuale Aufteilung der weltweiten Smartphone-Verkäufe an Endnutzer nach Betriebssystem [vgl. Gartner 2013]

Aus Abbildung 1.1 geht Android im Bezug auf Marktanteil eindeutig als dominante Plattform hervor. Diese Erkenntnis lässt sich jedoch nicht direkt auf die Wahl einer geeigneten Entwicklungsplattform übertragen. Zieht man andere Indikatoren wie um ein vielfaches höhere Umsätze durch App-Verkäufe [vgl. Perez 2013] und prozentual gesehen mehr erzeugten Web-Traffic durch Geräte mit iOS-Betriebssystem [vgl. NetMarketShare 2014] in Betracht, wird deutlich, dass die Verteilung der Smartphone-Verkäufe nicht zwingend Rückschlüsse auf die Relevanz der dazugehörigen Plattformen zulässt.

Aufgrund der zunehmenden Bedeutung von Apps und der Fragmentierung des Marktes sind Firmen häufig gezwungen, ihre Apps für alle maßgeblichen mobilen Plattformen und idealerweise auch viele weitere kleine Plattformen bereitzustellen, um so ihre Kunden erreichen und zufriedenstellen zu können. Dies führt in finanzieller und zeitlicher Hinsicht zu erheblichem Mehraufwand für das Unternehmen und für die mit der Umsetzung beauftragten Entwickler, da zwar einige grundlegende Strukturen oder Gestaltungselemente zwischen den verschiedenen Plattformen ausgetauscht werden können, jedoch aufgrund verschiedener Programmiersprachen und SDKs nicht der zeitlich aufwändige Code.

Erste Abhilfe schaffen an dieser Stelle sogenannte Web-Apps, also Anwendungen, welche mit Web-Technologien (HTML, CSS, Javascript etc.) entwickelt und für die Nutzung in einem mobilen Browser als Webseite angeboten werden. Dies ermöglicht die Ausführung der Anwendung ungeachtet des Herstellers auf jedem Gerät, das über einen modernen Web-Browser verfügt. Vorteilhaft ist bei dieser Methode auch, dass gegebenenfalls Teile einer bereits existierenden, herkömmlichen Webseite übernommen werden können.

Es gelten jedoch auch viele Einschränkungen für Web-Apps. So ist durch die Nutzung der Anwendung im Browser der Zugriff auf gerätespezifische Funktionen ausgeschlossen bzw. stark eingeschränkt. Jüngste Entwicklungen im Rahmen der HTML5-Spezifikation bieten zwar mittlerweile Zugriff auf viele standardisierte Funktionen wie Ortung, das Abspielen von Audio- und Videoinhalten oder das lokale Speichern von Daten [vgl. WHATWG 2013]; deren Unterstützung hängt jedoch stark vom verwendeten Gerät, Betriebssystem und Browser ab [vgl. Firtman 2014]. Darüber hinaus reichen der Funktionsumfang und auch die Leistungsfähigkeit trotz schneller Weiterentwicklung und zunehmend leistungsfähigerer Hardware noch nicht an die von vergleichbaren nativen Lösungen heran. Letztlich sind Web-Apps zum aktuellen Zeitpunkt weniger als Apps und mehr als auf mobile Geräte angepasste Webseiten einzuordnen, welche zunehmend auch in begrenztem Maße auf Gerätefunktionen zurückgreifen. Sie sind auf den Browser beschränkt und können somit nicht im Hintergrund ausgeführt, in App-Stores angeboten werden oder Push-Nachrichten empfangen.

Hier kommen plattformübergreifende Entwicklungsmethoden ins Spiel, die es ermöglichen sollen, mit begrenztem Entwicklungsaufwand mehrere mobile Plattformen zu versorgen und zeitgleich nicht auf den Funktionsumfang und die Leistungsfähigkeit des ausführenden Geräts verzichten zu müssen. Hierfür bieten viele Drittanbieter mittlerweile kostenfreie und -pflichtige Lösungen in Form sogenannter Frameworks und Entwicklungswerkzeuge an, welche im Detail verschiedene Herangehensweisen an das Problem bieten. Gemeinsam haben diese, dass versucht wird, möglichst große Teile oder sogar den vollständigen Code der Anwendung für alle Plattformen zu verwenden und durch mitgelieferte Bibliotheken eine verallgemeinerte Schnittstelle auf die gerätespezifischen Funktionen zu bieten. Weiterhin besteht zumeist auch die Möglichkeit, bestimmte Elemente innerhalb der plattformübergreifenden App plattformspezifisch mit den nativen Sprachen und SDKs zu entwickeln - zum Beispiel aus Gründen der Funktionalität oder Performance.

Doch auch die sogenannte „Cross-Platform“-Entwicklung unterliegt je nach Herangehensweise Kompromissen. So wird häufig kritisiert, dass bei der parallelen Entwicklung für mehrere Plattformen Benutzeroberflächen (engl. user interface, UI) entstehen können, welche sich nicht an üblichen Gestaltungs- und Bedienungskonzepten der Plattform orientieren und somit dem Anwender den Einstieg erschweren. Auch in puncto Leistung und Funktionsumfang bestehen aufgrund der verwendeten Brückentechnologien Einschränkungen.

## 1.2 Motivation für das Thema und Zielsetzung

Die zunehmende Relevanz von Apps und die Notwendigkeit, diese auf verschiedenen Plattformen bereitzustellen, geben Anlass, die vielfältigen Möglichkeiten zur technischen Umsetzung dieser Anforderung genauer zu beleuchten und miteinander zu vergleichen.

Bereits die kurze Gegenüberstellung der verfügbaren Entwicklungsformen in Kapitel 1.1 macht deutlich, dass diese jeweils verschiedene Vor- und Nachteile aufweisen. Auch die innerhalb einer Rubrik angebotenen Lösungen unterscheiden sich in ihrer Herangehensweise und somit in ihrer Eignung für verschiedene Projekte und Anwendungsarten.

Dies lässt die Vermutung zu, dass das Urteil, welche Vorgehensweise für das Entwickeln von Apps geeigneter sei, möglicherweise gar nicht pauschal zu fällen ist, da es von vielen Aspekten abhängt:

- Welche Funktions- und Leistungsansprüche stellt der Endkunde oder der Anwender an die umzusetzende App?
- Welche inhaltlichen, zeitlichen und somit auch finanziellen Rahmenbedingungen stellt der Auftraggeber?
- Welche Anforderungen müssen erfüllt sein, damit der Entwickler diese einhalten und die Anwendung erfolgreich implementieren kann?

Aus diesem Grund soll ermittelt werden, für welche Parteien - Nutzer, Entwickler oder Auftragsgeber - und unter welchen Umständen sich die verschiedenen Herangehensweisen am besten eignen und wann sie eher von Nachteil sind. Um einen Vergleich der Entwicklungsformen durchführen zu können, wird deshalb im Rahmen der Arbeit eine App-Idee mit Hilfe von einigen relevanten Frameworks umgesetzt. Weiterhin werden externe Stimmen und Literatur einbezogen, um Erkenntnisse über die Eignung der beleuchteten Varianten zu gewinnen.

## 1.3 Gliederung der Arbeit

Die Arbeit “Plattformabhängige und –unabhängige Entwicklung mobiler Anwendungen am Beispiel von Geo-Wikipedia-App” ist in fünf Kapitel aufgeteilt, welche sich in folgender Weise mit dem Thema auseinandersetzen:

Das erste Kapitel beschreibt zunächst den Status Quo der modernen mobilen Entwicklung und insbesondere bestehende Hürden sowie Lösungsversuche. Im Anschluss wird weiterhin die Motivation des Autors für das Thema dargelegt und die Gliederung der Arbeit erläutert.

Im zweiten Kapitel werden beispielhaft zwei native sowie drei plattformunabhängige mobile Frameworks vorgestellt und ihr Entwicklungskonzept im Bezug auf SDKs, Programmiersprachen und Benutzeroberflächen beschrieben.

Das dritte Kapitel betrifft den praktischen Aspekt dieser Arbeit - die Entwicklung einer Smartphone App mit Hilfe der aufgeführten Frameworks. Hier werden zunächst ein für übliche Smartphone-Anwendungen repräsentativer Anforderungskatalog und darauf aufbauend ein Konzept für eine App entwickelt. Im Anschluss werden die Umsetzung dieser und dabei gemachte Erkenntnisse beschrieben.

Anschließend leitet das vierte Kapitel aus den praktischen Ergebnissen und externen Einschätzungen ab, welche Vor- und Nachteile die Entwicklungsmethoden jeweils für verschiedene Interessengruppen haben. Darüber hinaus werden für die untersuchten Entwicklungswerkzeuge geeignete Einsatzszenarien definiert.

Das fünfte Kapitel fasst gemachte Ergebnisse in Form eines Fazits zusammen und bietet einen Ausblick auf weitere für das Thema relevante Fragestellungen.

# 2 Beschreibung von Entwicklungswerkzeugen für mobile Plattformen

Im folgenden Kapitel werden einige für den aktuellen mobilen Markt repräsentative Entwicklungswerkzeuge und von ihnen angebotene Schnittstellen (engl. application programming interfaces, kurz APIs) vorgestellt. Die plattformunabhängige Entwicklung wird hierbei durch die Frameworks Adobe PhoneGap, Xamarin und Appcelerator Titanium vertreten, während im Rahmen der plattformabhängigen Entwicklung das Apple iOS-SDK und Google Android-SDK betrachtet werden. Weitere verfügbare Werkzeuge und reine Webapps werden aus Gründen des Umfangs nicht in diese Arbeit einbezogen.

Die erfolgreiche Ausführung eines Programms hängt stets von der Konfiguration des ausführenden Geräts, also der Kombination aus Prozessor, Betriebssystem und verfügbaren Dienstprogrammen ab. Aus diesem Grund gilt ein Programm auch als plattformübergreifend, wenn es auf mehr als nur einer Plattform - aber nicht zwingend „allen“ - ausgeführt werden kann [vgl. Wikipedia 2014a]. Im Themenbereich mobiler Anwendungen (Apps) impliziert Plattformunabhängigkeit meist die Möglichkeit, die Anwendung auf mehr als einer der verbreiteten Plattformen ausführen zu können.

Im Umkehrschluss zeichnen sich plattformabhängige Entwicklungsformen dadurch aus, dass die auf ihrer Basis erzeugten Programme nur auf einer auf sie zugeschnittenen, also für sie nativen (lat. nativus, angeboren, natürlich [vgl. Duden 2014]) Rechnerkonfiguration ausgeführt werden können. Hierbei ist heutzutage meist das verwendete Betriebssystem ein Indikator für die Kompatibilität, da dieses meist auch eine bestimmte zugrundeliegende Architektur und die Verfügbarkeit bestimmter Software und Hardware impliziert.

## 2.1 Plattformabhängige Entwicklung und beispielhafte Werkzeuge

Im Fall der plattformabhängigen Entwicklung stellt der Hersteller zumeist eigene Entwicklungswerkzeuge für eine von ihm bereitgestellte Plattform zur Verfügung. Zum einen können hierdurch involvierte Hardware- und Softwarekomponenten sehr gut aufeinander abgestimmt und Fehler durch ungewöhnliche Konfigurationen vermieden werden. Zum anderen ermöglicht diese Vorgehensweise auch das Erreichen einer starken Bindung von Entwicklern sowie Endkunden an die eigene Plattform.

Aus diesem Grund sind plattformabhängige Entwicklungswerkzeuge meist eng mit dem (häufig gleichnamigen) Betriebssystem verknüpft. Zum Beispiel: iOS SDK für Apple iOS, Googles Android SDK für Android und Windows Phone SDK für Microsoft Windows Phone.

### 2.1.1 Apple iOS SDK

#### Allgemeines

Das Apple iOS SDK (ursprünglich iPhone SDK) beinhaltet Entwicklungswerkzeuge zur Entwicklung mobiler Anwendungen für das seit 2007 verfügbare mobile Betriebssystem iOS. Im Gegensatz zu anderen Herstellern im gleichen Segment lizensiert Apple die eigene mobile Plattform nicht an weitere Gerätehersteller [vgl. Wikipedia 2014b], so dass das auf Apples Desktop Betriebssystem Mac OS X basierende iOS und die von Apple angebotenen mobilen Geräte (iPhone, iPad, iPod Touch, iPod) eine in sich geschlossene Plattform darstellen.

Ursprünglich nur zur Ausführung von Web-Apps vorgesehen [vgl. Gonsalves 2007] öffnete sich die Plattform 2008 im Rahmen der Eröffnung des Apple App Stores auch für nativ ausführbare Anwendungen von Drittentwicklern [vgl. Cheng 2012]. Dadurch, dass der Store - abgesehen von speziellen Enterprise-Lösungen oder dem nicht von Apple autorisierten „Jailbreak“ [vgl. Wikipedia 2014c] - die einzige Möglichkeit zur Installation von Anwendungen ist, unterliegt die Plattform auch in dieser Hinsicht der direkten Kontrolle Apples.

Während die Veröffentlichung von Anwendungen im App Store eine kostenpflichtige Mitgliedschaft erfordert [vgl. Apple 2014], ist die Nutzung der Entwicklungswerkzeuge kostenfrei. Das SDK und die Entwicklungsumgebung Xcode sind nur für das Apple Betriebssystem Mac OS X verfügbar. Das Programmieren mobiler Anwendungen auf Basis dieser Werkzeuge erfolgt hauptsächlich auf Basis der Programmiersprache Objective-C,

welche intensiv von Apple gefördert wird. Stellenweise ist es auch möglich, die verwandten Sprachen C und C++ zu verwenden. Weiterhin stellte Apple im Juni 2014 auf der Entwicklerkonferenz WWDC eine neue objektorientierte Sprache namens Swift vor, welche Objective-C ergänzt und einfacher zu nutzen sein soll [vgl. Timmer 2014].

#### Entwicklungskonzept

Für die Entwicklung mit dem iOS SDK stellt Apple eine integrierte Entwicklungsumgebung (engl. integrated development environment, IDE) namens Xcode zur Verfügung. Obwohl es mittlerweile auch Alternativen wie die IDE AppCode [vgl. Jetbrains 2014] gibt, greifen diese auch auf eine installierte Version von Xcode zurück. Weiterhin ist für einige Aufgaben, wie zum Beispiel das Einreichen der App in den AppStore, die Verwendung von Xcode unvermeidlich. Die Geschlossenheit des Systems setzt sich also auch an dieser Stelle fort.

Die bereitgestellten Entwicklungswerkzeuge fördern die Verwendung des sogenannten Model-View-Controller (MVC) Paradigmas [vgl. Apple 2013]. Dieses trennt bei der Entwicklung einer Anwendung drei Aspekte voneinander: Die Verarbeitung und Speicherung von Daten (Model), die grafische Darstellung dieser (View) und die Koordination der Komponenten und Steuerung z.B. durch Sensoren oder Nutzereingaben (Controller). Dies soll unter anderem die Wiederverwendbarkeit, Erweiterbarkeit und Übersichtlichkeit des Codes erhöhen.

So übernehmen üblicherweise Klassen vom Typ *UIViewController* die Kontrolle über eine View, also eine bestimmte Bildschirmansicht in der Anwendung. Die Views vom Typ *UIView* können entweder programmatisch oder mit Hilfe des in Xcode mitgelieferten Werkzeugs InterfaceBuilder über eine grafische Oberfläche erzeugt werden. Letztere werden als sogenannte .xib-Dateien gespeichert, welche zur Laufzeit in eine *UIView*-Instanz geladen werden, die über sogenannte *IBOutlets* und *IBActions* Interaktionen zwischen den definierten Bedienelementen und dem Controller zulässt. Den Aspekt des Models im MVC-Muster kann je nach Komplexität der Anwendung zum Beispiel ein *CoreData*-Model sein, welches den Zugriff auf eine sqlite-Datenbank verwaltet.

Auf diese Weise entwickelte Anwendungen können auf einem mit iOS betriebenen Gerät oder mit Hilfe des mitgelieferten iOS-Simulators direkt auf dem Computer ausgeführt und getestet werden. Im Simulator wird jedoch für Desktop-Prozessoren üblicher x86-Code ausgeführt und nicht etwa der für die Prozessoren von iOS-Geräten typische ARM-Code. Dies hat den Vorteil, dass die Simulation in ihrer Ausführung schnell ist und eine flüssige Darstellung ermöglicht. Nachteilig ist, dass es in Randfällen aufgrund der anderen Architektur zu Unterschieden in der Ausführung gegenüber

tatsächlichen iOS-Geräten kommen kann. So greift der iOS-Simulator auch auf alle verfügbaren Ressourcen (CPU, Arbeitsspeicher, Speicher etc.) des Computers zurück und bietet somit keinen Anhaltspunkt für die Leistung der Anwendung auf einem realen Gerät.

### API

Ein zentrales Element des iOS-SDKs sind umfangreich bereit gestellte APIs, welche den abstrahierten Zugriff auf viele Gerätefunktionen ermöglichen und die Implementation typischer Funktionen vereinfachen.

Hierarchisch an oberster Stelle und von Entwicklern am meisten verwendet ist das Framework Cocoa Touch, welches das Grundgerüst für alle Apps darstellt. Der für die Funktionen und Anforderungen mobiler Geräte angepasste Ableger der herkömmlichen Cocoa API (für Mac OS X) simplifiziert das Erstellen von iOS-typischen Benutzeroberflächen, die Nutzung von Multi-Touch-Gesten, Multitasking-Funktionen und weitere Gesamt-Lösungen für Systemfunktionen wie Kontakte oder Karten [vgl. Apple 2014b].

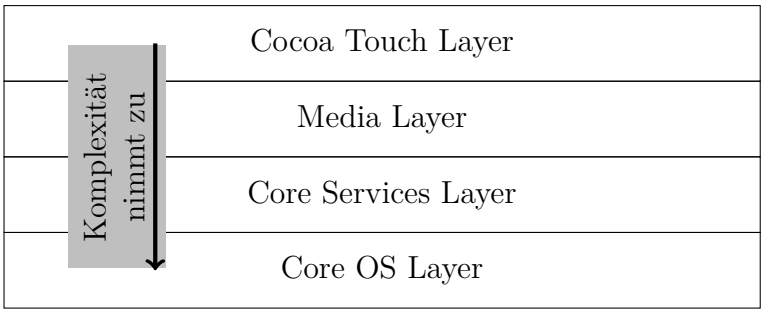


Abbildung 2.1: iOS Architekturbild [vgl. Apple 2014a]

Darunter angeordnet bietet *Media Layer* vielseitige Frameworks für Grafik-, Audio- und Videoanwendungen, die es vereinfachen, Multimedia-Funktionen in Apps umzusetzen. Die tiefer liegende Rubrik *Core Services* bietet weiterhin Dienste zur Nutzung von SQLite Datenbanken, Cloud-Speicher, Netzwerk-Funktionalität, Ortungsdiensten und Accelerometer-Daten. An unterster Stelle steht das *Core OS Layer*, auf dessen Sicherheits-, Konnektivitäts- und Peripheriedienste viele höher angesiedelte Funktionen aufbauen.

Je höher eine der Schnittstellen angesiedelt ist, desto größer fällt ihr Abstraktionsgrad gegenüber den meist in C geschriebenen Kerntechnologien aus. So sind häufig verwendete APIs auch für Neueinsteiger zugänglich und ermöglichen das schnelle Implementieren häufig benötigter App-Funktionen. Bei Bedarf kann mit entsprechend größerem Aufwand auch mit „lower-level“ Technologien gearbeitet werden.

### 2.1.2 Google Android SDK

#### Allgemeines

Für das seit 2008 verfügbare mobile Betriebssystem Android kann mit Hilfe der im Android SDK enthaltenen Werkzeuge entwickelt werden. Im Gegensatz zu Apple iOS wird Android grundsätzlich quelloffen, als freie Software, durch die von Google gegründete Open Handset Alliance entwickelt, was es unterschiedlichen Hardwareherstellern ermöglicht, das Betriebssystem auf ihren Geräten anzubieten und an diesem auch eigene Modifikationen vorzunehmen [vgl. Google 2014]. Erst seit 2010 bietet Google eigene Android-Hardware im Rahmen der Nexus Produktserie an.

Ein weiterer Unterschied zur iOS-Plattform ist die Möglichkeit, Apps aus verschiedenen Quellen zu beziehen. So bietet Google seit Ende 2008 zwar in Form des Google Play Store (ursprünglich Android Market) eine zentrale Anlaufstelle für die Beschaffung von Apps und weiteren digitale Inhalten an; diese stellt jedoch neben Angeboten der Gerätehersteller wie Samsung Apps oder Amazon Appstore for Android oder offenen AppStores wie F-Droid nur eine von vielen Möglichkeiten dar [vgl. One Platform Foundation 2013]. Weiterhin ist es auch möglich, Anwendungen aus anderen Quellen zu installieren, wenn dies in den Einstellungen des Geräts erlaubt wird [vgl. Google 2014a]. So können Anwendungen auf einer Webseite zum Herunterladen angeboten oder per Email versandt werden; dies hat aber bei Anwendungen aus unbekannten Quellen auch sicherheitsrelevante Implikationen.

Die Nutzung des Android SDKs ist kostenlos möglich. Ob Kosten für die Veröffentlichung der Anwendung anfallen, hängt von der Wahl zuvor genannter Distributionsmöglichkeiten ab. Die Entwicklungswerkzeuge und verschiedene Entwicklungsumgebungen sind für alle gängigen Desktop-Plattformen - Linux, Windows und Mac OS X - und sogar Android selbst verfügbar. Android Apps werden mehrheitlich in der verbreiteten Programmiersprache Java entwickelt. Es ist zusätzlich auch möglich, in C und C++ und anderen Sprachen geschriebene Bibliotheken mit Hilfe des Android Native Development Kits in eine Android App zu integrieren und vom Java-Code aus aufzurufen [vgl. Google 2014b].

#### Entwicklungskonzept

Die Entwicklung von Anwendungen mit dem Android SDK kann in verschiedenen IDEs durchgeführt werden. Für die langjährig von Google als offizielle Entwicklungsplattform positionierte Entwicklungsumgebung Eclipse wird ein spezielles Plugin bereitgestellt, welches die Entwicklungswerkzeuge des Android SDK in die Software einbindet. Doch auch alternative IDEs wie Jetbrains IntelliJ IDEA oder auch Oracles Netbeans unterstützen die Android Entwicklung in vollem Umfang. Derzeit noch im

Stadium eines „Early Access Preview“ arbeitet Google ergänzend auch an einer auf IntelliJ IDEA basierenden eigenen Android-zentrischen Entwicklungsumgebung mit dem Namen Android Studio.

Auch das Android SDK begünstigt eine Entwicklung nach dem in Kapitel 2.1.1 erläuterten Model-View-Controller Pattern. Hierbei übernimmt die vom SDK bereitgestellte Klasse *Activity* meist die Kontrolle über einen Bildschirminhalt, dessen Darstellung (View) programmatisch erzeugt oder in den meisten Fällen in einer separaten XML-Datei definiert werden kann. Das Anpassen dieser Layout-Dateien kann entweder über in den IDEs mitgelieferte grafische Editoren oder auch, aufgrund des von Menschen gut lesbaren Formats, per Hand vorgenommen werden. Zur Referenzierung der dort deklarierten Elemente aus dem Code können Identifikationsbezeichnungen angegeben werden. Das Model kann zum Beispiel eine der von Android angebotenen *Adapter*-Klassen sein, welche den Zugriff auf Informationen eines schlichten Arrays oder einer sqlite-Datenbank verwalten kann.

Damit erzeugte Anwendungen direkt auf dem Computer ausgeführt und getestet werden können, bietet das Android SDK die Möglichkeit, den Betrieb eines Android-Geräts zu emulieren. Um der vielfältigen Gerätelandschaft gerecht zu werden, können verschiedene Emulatoren angelegt und im Bezug auf Aspekte wie Bildschirmgröße, Menge an Arbeitsspeicher und verfügbare Sensoren konfiguriert werden. Die Emulation bietet im Gegensatz zum iOS-Simulator den Vorteil, dass der tatsächliche ARM-Code ausgeführt wird, welcher auch auf dem Gerät zum Einsatz kommt. Dies führt jedoch dazu, dass die Leistung aufgrund der Emulation auch nicht der des realen Geräts entspricht und steht im Gegensatz zum Anspruch, ein bestimmtes Gerät und die dort verfügbaren Ressourcen zu imitieren.

#### API

Das Android-SDK bietet ähnlich dem iOS-SDK umfangreiche Funktionen zur Implementation von typischen Apps. So werden Möglichkeiten zur Erzeugung von Benutzeroberflächen, Animationen und Grafikanwendungen zur Verfügung gestellt. Weiterhin werden APIs für die Nutzung von Audio-, Video- und Fotofunktionen, Datenspeicherung sowie GPS- und Beschleunigungssensoren angeboten. Leistungsorientierte Schnittstellen wie Codecs zur Medienwiedergabe oder die auf WebKit basierenden WebViews greifen hierbei häufig auf in C oder C++ geschriebene Bibliotheken zurück [vgl. Wikipedia 2014d].

Da Hersteller und Netzanbieter dafür zuständig sind, die von Google entwickelten Software-Updates für Android für ihre Geräte verfügbar zu machen, werden nicht uner-

hebliche Anteile des Android Gerätemarkts nicht mit den neuesten Android-Versionen betrieben. So verwenden zum aktuellen Zeitpunkt nur 13,6% der Geräte die aktuellste Version des Betriebssystems namens KitKat (Android 4.4, API Level 19) [vgl. Google 2014c]. Um Anwendungen, welche auf neue APIs zurückgreifen, dennoch für möglichst viele Geräte bereitstellen zu können, entwickelt Google parallel zur aktuellen API sogenannte „Android Support Libraries“. Diese ermöglichen Geräten mit einem niedrigeren API-Level die Nutzung einiger neu hinzugekommener Schnittstellen. Derzeit bieten die Support Libraries v4 (nutzbar ab API Level 4), v7 und v13 jeweils unterschiedliche Funktionen abwärtskompatibel an [vgl. Google 2014d].

## 2.2 Plattformunabhängige Entwicklung und beispielhafte Werkzeuge

Im Gegensatz zu plattformabhängigen Entwicklungswerkzeugen werden Lösungen zur plattformunabhängigen Entwicklung aus naheliegenden Gründen meist von Drittanbietern und nicht von den Geräteherstellern selbst angeboten. Ihr Anspruch ist es, das parallele Entwickeln für mehrere Plattformen zu vereinfachen. Für diese Problemstellung haben sich mittlerweile verschiedene Lösungen mit unterschiedlichen Herangehensweisen herausgebildet.

- Hybride Apps - eine Kombination aus Web-Technologien (HTML, CSS, Javascript) und nativer Funktionalität
- Cross Compiling Apps - aus einer gemeinsamen Code-Basis wird beim Kompilieren plattformspezifischer, nativer Code erzeugt
- Cross Runtime Apps - für alle Plattformen wird eine in sich geschlossene Laufzeitumgebung bereitgestellt, in der der gemeinsame Code interpretiert und ausgeführt werden kann

Hierbei wird der Begriff Cross Compiling häufig fälschlicherweise für Vertreter des Typs Cross Runtime verwendet. Die meisten Entwicklungswerkzeuge erzeugen nativen Code nur für eine dünne Schicht, welche die App beherbergt und Zugriff auf plattformspezifische Funktionen erlaubt; die eigentliche Programmlogik wird zur Laufzeit interpretiert und greift über Schnittstellen auf weitere Funktionen zu.

Gemeinsam haben diese Lösungen zumeist, dass sie zumindest in einigen Schritten auf die nativen Entwicklungswerkzeuge zurückgreifen müssen, um plattformspezifische Anwendungsformate zu erzeugen und auszuführen. Die folgenden Beispiel-Frameworks setzen für das Kompilieren und Ausführen einer Anwendung die Verfügbarkeit der jeweiligen nativen Entwicklungswerkzeuge voraus. Möchte man zum Beispiel die Android-Variante einer App auf dem Computer ausführen, müssen das Android SDK und ein entsprechender Emulator auf dem System konfiguriert sein. Dies hat auch zur Folge, dass für die Entwicklung einiger Plattformen auch die üblichen Betriebssystem-Beschränkungen gelten - so kann die iOS-App eines Cross-Platform Projekts nur auf einem Mac und die Windows Phone App nur unter Windows entwickelt werden.

Der Vollständigkeit halber ist zu erwähnen, dass eine partielle plattformübergreifende Entwicklung grundsätzlich auch ohne die Nutzung zusätzlicher Werkzeuge möglich ist. So entwickelt die Firma DropBox für ihre iOS- und Android-Anwendungen eine zentrale C++ Bibliothek, welche geteilte Funktionalität bereitstellt [vgl. Begemann 2014]. Dies

erlaubt jedoch keine plattformübergreifende Entwicklung von Benutzeroberflächen und erfordert einen großen initialen Aufwand.

### 2.2.1 Adobe PhoneGap

#### Allgemeines

Den am meisten verbreiteten Vertreter für hybride Apps stellt das 2009 vorgestellte Entwicklungsframework PhoneGap dar. Ehemals vom Unternehmen Nitobi Software entwickelt, welches 2011 von Adobe gekauft wurde [vgl. Adobe 2011], wird die Softwarelösung nun im Rahmen des Apache Cordova Project unter dem Namen Cordova durch die Apache Foundation weiter entwickelt. Doch auch PhoneGap wird auf Basis von Apache Cordova als Distribution weiterentwickelt, welche weitere Funktionalität und Integration mit weiteren Adobe Produkten beinhaltet [vgl. Leroux 2012].

PhoneGap ermöglicht die paralle Entwicklung für die mobilen Plattformen iOS, Android, Windows Phone, Blackberry OS, webOS, Amazon Fire OS, Tizen und Firefox OS [vgl. Phonegap 2014]. Die erzeugten Anwendungen können im plattformspezifischen Format über die jeweiligen Stores angeboten werden.

Die Entwicklung mit PhoneGap ist kostenlos. Da mit PhoneGap die Notwendigkeit für plattformspezifische Entwicklungswerkzeuge nicht entfällt, können dennoch auf einigen Plattformen Kosten für die Distribution anfallen. Seit 2012 bietet Adobe mit dem PhoneGap Build Service einen je nach Anforderungen kostenpflichtigen Cloud-Dienst an, welcher die eingangs aufgeführte Notwendigkeit von vorinstallierten nativen SDKs umgeht, indem die Anwendungen auf entfernten, vorkonfigurierten Systemen kompiliert werden können [vgl. Phonegap 2014d].

#### Entwicklungskonzept

Elementar für den Ansatz von Phonegap ist eine sogenannte WebView, also eine Instanz des Browsers der jeweiligen Plattform ohne Bedienelemente, in der eine mit Web-Technologien (HTML, CSS, Javascript) erzeugte Anwendung angezeigt und ausgeführt wird. Diese WebView wird als native Anwendung verpackt und kann somit als solche auf den verschiedenen Plattformen installiert werden [vgl. Phonegap 2014a]. Um jedoch mehr Funktionalität als eine reine Web-App bieten zu können, kommt zusätzlich der hybride Ansatz ins Spiel. Der von PhoneGap bereitgestellte Wrapper stellt dem in der WebView ausgeführten Code über die eingebundene *cordova.js* JavaScript-Datei diverse Schnittstellen bereit, welche den Zugriff auf plattform- und hardwarespezifische Funktionen ermöglicht. Weiterhin ist es durch PhoneGap möglich, über diverse Ereignisse des Geräts und Betriebssystems wie das Verlassen der App, informiert zu

werden und auf diese zu reagieren [vgl. Phonegap 2014b].

PhoneGap selbst stellt keine vorgefertigten Elemente für Benutzeroberflächen oder dergleichen bereit. Stattdessen ermöglicht die zentrale WebView die Verwendung jedmöglicher Web-Technologien zur Umsetzung von Navigationsstrukturen und Oberflächen der Anwendung. So ist es grundsätzlich möglich, die Anwendung als schlichte Struktur von HTML-Seiten umzusetzen oder den Quellcode einer bestehenden Webseite direkt zu übernehmen. Meistens wird jedoch auf mobile Web-Frameworks wie jQuery Mobile [vgl. jQuery Mobile 2014] oder Sencha Touch [vgl. Sencha 2014] zurückgegriffen, welche für die mobile Gestensteuerung optimierte Bedienelemente und Funktionen bieten.

Zur Erstellung und Verwaltung von PhoneGap Projekten steht eine Kommandozeilenanwendung (Command Line Interface, CLI) zur Verfügung, mit welcher Zielplattformen hinzugefügt und Apps für eine bestimmte Plattform kompiliert und ausgeführt werden können. Wird eine Zielplattform hinzugefügt, wird für diese eine native Projektstruktur - für iOS zB. ein Xcode-Projekt - angelegt, aus dem das plattformübergreifende Web-Projekt referenziert wird. Am erzeugten nativen Projekt können im Anschluss auch plattformspezifische Veränderungen vorgenommen werden. Weiterhin ist es bei Bedarf auch möglich, Ressourcen des geteilten Web-Projekts für eine bestimmte Plattform zu überschreiben.

Eine dedizierte Entwicklungsumgebung wird nicht empfohlen oder mitgeliefert. Ergänzend zur PhoneGap CLI können übliche Web-Entwicklungswerkzeuge verwendet werden. Zum Debuggen der Web-Komponente werden häufig Browser wie Google Chrome oder Apple Safari und deren integrierte JavaScript-Konsole verwendet.

Die erzeugten Anwendungen sind sich in den meisten Fällen aufgrund des zugrundeliegenden Web-Projekts recht ähnlich und verhalten sich meist nicht wie native Anwendungen der Zielplattformen.

#### API

PhoneGap setzt größtenteils auf die in der Web-Entwicklung üblichen Browser-APIs, welche über JavaScript die Manipulation und die Interaktion mit der angezeigten Seiten-Struktur (DOM, Document Object Model) sowie die Nutzung von HTML5-Funktionen ermöglichen.

Weiterhin stellt PhoneGap anspruchsvollere Funktionen als JavaScript-API bereit, welche im Hintergrund mit Hilfe von nativen Plugins auf die nativen APIs der unterstützten Plattformen zugreift. Diese werden dem Projekt mit Hilfe der PhoneGap

CLI hinzugefügt und beinhalten Implementationen für die Nutzung von Beschleunigungssensor, Kompass, Geolocation, Kamera, Kontakten, Dateisystem und das Auslösen von Benachrichtigungen (Alarm, Ton und Vibration) auf der jeweiligen Plattform [vgl. Phonegap 2014c]. Durch die Plugin-Struktur kann aus dem plattformübergreifenden Code heraus beispielsweise die Kamera angesprochen werden, ohne gerätespezifische Unterschiede beachten zu müssen - diese werden vom nativen Code der bereitgestellten Plugins bedacht. Plugins dieser Art können auch von Drittanbietern entwickelt und bereitgestellt werden [vgl. Phonegap 2014e].

Zuletzt können in einer projektweiten und in mehreren plattformspezifischen Konfigurationsdateien Aspekte wie Name und Icon der Anwendung oder unterstützte Bildschirmorientierungen festgelegt werden [vgl. Phonegap 2014f].

### 2.2.2 Xamarin

#### Allgemeines

Xamarin ist ein relativ junges Beispiel für plattformübergreifende Softwareentwicklung, welches 2011 auf Basis der plattformübergreifenden open-source Implementation des .NET-Frameworks namens Mono entstand.

Mit Xamarin können Anwendungen für Android, iOS, Windows Phone sowie Windows und Mac OS X entwickelt werden. Xamarin ist ein kostenpflichtiges Produkt, welches auf ein Abo-Modell setzt. Neben einer stark eingeschränkten kostenfreien „Starter“-Edition und einer 30-tägigen Probeversion werden unterschiedliche Preismodelle angeboten, bei denen pro Entwickler und für jede benötigte Plattform separat gezahlt werden muss [vgl. Xamarin 2014]. Die Programmierung erfolgt auf Basis der Programmiersprache C#. Mit Xamarin Studio Version 5 (bzw. 3 für Windows) kommt nun auch die Unterstützung der funktionalen Programmiersprache F# hinzu.

#### Entwicklungskonzept

Xamarin erlaubt es, auf Basis der nativen APIs der Plattformen und des von Microsoft entwickelten .NET Framework, Anwendungen für die unterstützten Plattformen parallel zu entwickeln. Mit .NET wird Code bei Kompilation in eine als Common Intermediate Language (CIL) bezeichnete „Universalsprache“ übersetzt und so unabhängig von der Computerarchitektur in einer Common Language Runtime ausgeführt und dort „just in time“ in Maschinencode transformiert.

Im Vergleich zur sogenannten „write once, run everywhere“ Mentalität herkömmlicher plattformübergreifender Lösungen verfolgt Xamarin eine andere Strategie. Im Mittelpunkt des Entwicklungsframeworks steht die Idee, möglichst viel zentrale Logik der

Anwendung (oft genannt „Business Logic“) nur einmal schreiben zu müssen und für alle Anwendungen zentral verfügbar zu machen. Die Benutzerflächen und der Code, der diese ansteuert, sollen hingegen pro Plattform entwickelt werden, um dem Nutzer die gewohnte Interaktion mit nativen Bedienelementen bieten zu können. Hierbei können zum Beispiel auch die Layout-Formate der nativen SDKs verwendet werden - .xib für iOS und das XML-basierte Format von Android. Eine beispielhafte Projektstruktur für diese Vorgehensweise kann Abbildung 2.2 entnommen werden. Es wird deutlich, dass diese Vorgehensweise besonders effektiv ist, wenn die Anwendung viel plattformübergreifende Logik beinhaltet, die zentral gebündelt werden kann. Darüber hinaus bietet Xamarin in seiner neuesten Version mit Xamarin.Forms eine Technologie, welche es ermöglicht, auf Basis von C#-Code plattformübergreifende und dennoch native Bedienoberflächen zu erzeugen [vgl. Xamarin 2014a].

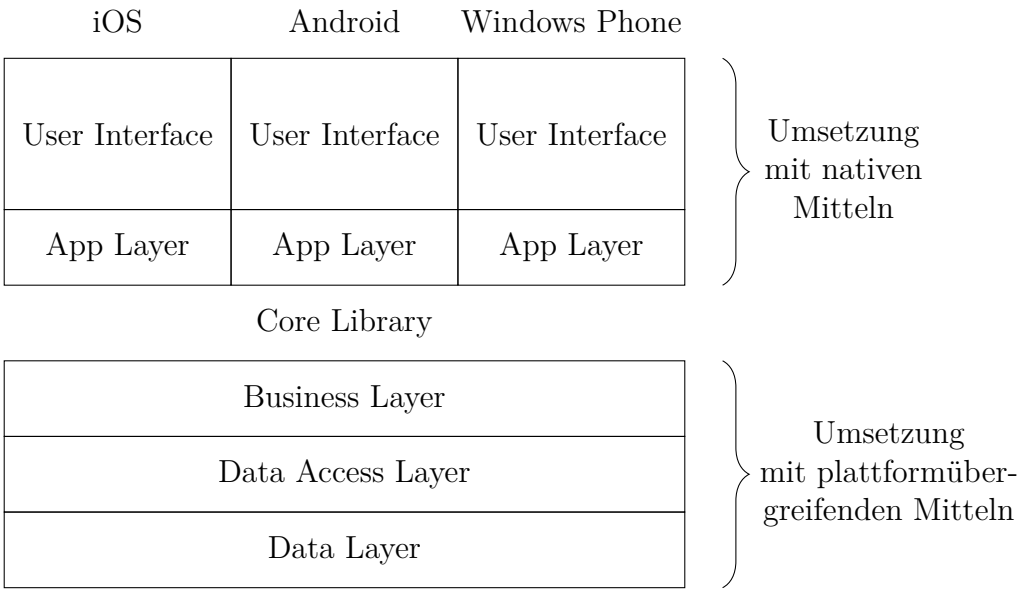


Abbildung 2.2: Beispiel für Xamarin Code-Struktur [vgl. Xamarin 2014b]

Die gleichzeitige Nutzung der plattformübergreifenden Kernlogik und der nativen APIs für Benutzeroberflächen wird ermöglicht, indem die Programmiersprache C# als anwendungsübergreifende Sprache eingeführt wird. Weiterhin stellt Xamarin die nativen APIs aller unterstützten Plattformen als C#-basierte Bibliotheken (Xamarin.Android, Xamarin.iOS) bereit, welche sogenannte Bindings zu den eigentlichen Schnittstellen herstellen. Dies bedeutet, dass auch der plattformspezifische Code in C# vorliegt und dementsprechend mit der ebenfalls in C# geschriebenen plattformübergreifenden Programmlogik interagieren kann. Die Windows Phone APIs sind unabhängig von Xamarin nutzbar, da diese ohnehin für die Nutzung mit C# vorgesehen sind.

Beim Erzeugen der Anwendung für Android oder Windows Phone wird der C#-Code in die einleitend erwähnte Zwischensprache CIL kompiliert [vgl. Xamarin 2014c]. Dazu wird die Laufzeitumgebung MonoVM der Anwendung beigefügt, sodass der CIL-Code „Just In Time“ (JIT) in Maschinensprache übersetzt und ausgeführt werden kann. Unter iOS wird der Code hingegen vorab (ahead-of-time, AOT) als Assembler-Code für ARM-Prozessoren kompiliert und die benötigten .NET-Klassen werden mitgeliefert. Dies hat den Grund, dass Apple es nicht zulässt, dass iOS-Anwendungen während der Laufzeit neuen Code generieren [vgl. Burnette 2010].

Durch die Verwendung von C# und .NET ist Microsofts IDE Visual Studio eine Option für die Entwicklung, welche von Xamarin aktiv unterstützt wird. Zusätzlich wird unter dem Namen Xamarin Studio eine eigene IDE für Mac OSX und Windows angeboten, welche die Entwicklung für alle unterstützten Plattformen erlaubt und Editoren für die nativen Benutzeroberflächen beinhaltet.

#### API

Grundsätzlich sind die APIs von iOS Cocoa Touch und des Android SDK durch Xamarins C# Bindings verfügbar. In Folge dessen können grundsätzlich auch alle plattformspezifischen Funktionen angesprochen und genutzt werden. Um dennoch zu vermeiden, dass komplexe Funktionen plattformspezifisch umgesetzt werden müssen, wird in Form von Xamarin.Mobile eine Bibliothek angeboten, welche gerätespezifische Funktionen wie z.B. Kalender, Geolocation und Foto/Video-Aufnahme als eine plattformübergreifend abstrahierte API bereitstellt [vgl. Xamarin 2014d]. Diese Bibliothek wird im Rahmen des Xamarin Components Store angeboten, in welchem Xamarin und Drittanbieter weitere kostenfreie und kostenpflichtige Bibliotheken in verschiedenen Kategorien anbieten [vgl. Xamarin 2014e].

Weiterhin ist es möglich, plattformübergreifende Aufgaben wie Netzwerkzugriffe oder Programmlogik mit APIs des .NET-Framework umzusetzen und so plattformeigene APIs zu vermeiden. Statt also z.B. die Nutzung einer sqlite-Datenbank unter iOS mit *CoreData* und unter Android mit den Klassen aus *android.database.sqlite* durchzuführen, kann stattdessen eine .NET Biblothek wie sqlite-net [vgl. sqlite-net 2014] verwendet werden und so der Anteil an von allen Plattformen geteiltem Code erhöht werden.

### 2.2.3 Appcelerator Titanium

#### Allgemeines

Ein Beispiel für Cross Runtime Apps ist das open-source Entwicklungswerkzeug Titanium der Firma Appcelerator, welches 2008 zunächst als Werkzeug zur Entwicklung von Web- und Desktop-Anwendungen veröffentlicht wurde. Nachdem im Jahr 2009 eine Betaversion mit Unterstützung von Android- und iOS-Apps folgte, veröffentlichte Appcelerator 2010 Version 1.0 ihres Cross-Platform Frameworks Titanium [vgl. Ihlenfeld 2009].

Titanium erlaubt die Entwicklung von Anwendungen für iOS, Android, BlackBerry sowie das mobile Web [vgl. Appcelerator 2014]. Die Unterstützung von Windows Phone befindet sich in Entwicklung [vgl. Feloney 2014].

Die Nutzung des Standardumfangs von Titanium ist grundsätzlich kostenlos. Ähnlich wie Adobe für PhoneGap bietet Appcelerator darüber hinaus kostenpflichtige Module und Enterprise-Lösungen wie die Appcelerator Platform an. Diese bietet zusätzliche Cloud-Dienste wie Nutzerverwaltung, Datenspeicher, Push-Benachrichtigungen sowie Tests und Analytics [vgl. Appcelerator 2014a].

#### Entwicklungskonzept

Obwohl Anwendungen auf Basis von Titanium ähnlich wie PhoneGap-Apps in Javascript programmiert werden, unterscheiden sich die Herangehensweisen der beiden Lösungen stark. Trotz der browsernahen Sprache nutzt Titanium keine WebView und erzeugt stattdessen native Benutzeroberflächen. Dies wird durch eine plattformübergreifende API ermöglicht, welche das Erzeugen und Nutzen von nativen Bedienelementen aus Javascript heraus ermöglicht. Beim Aufruf entsprechender APIs wird das native Pendant der jeweiligen Plattform angesprochen (vgl. Abbildung 2.3) - die Klasse *Titanium.UI.Button* entspricht unter iOS zB. der Klasse *UIButton* und unter Android *android.widget.Button*. Gleichermaßen interagiert das Modul für Ortung *Titanium.Geolocation* zum Beispiel unter Android mit dem Location-Framework *android.location* und unter iOS mit *CoreLocation*.

Während bei Xamarins Ansatz die nativen APIs als C#-Bindings bereitgestellt werden und native Layoutformate wie iOS .xib- oder Androids XML-Dateien verwendet werden können, stellt Titanium also eine ganz eigene API und Layout-Formate bereit, welche nur im Hintergrund auf native Funktionen aufsetzen.

Moderne Titanium Anwendungen werden zumeist auf Basis des Frameworks Alloy entwickelt [vgl. Appcelerator 2014c]. Dies stellt die Grundlage für übliche Strukturen des MVC-Patterns: In Controller-Klassen wird JavaScript-Code geschrieben,

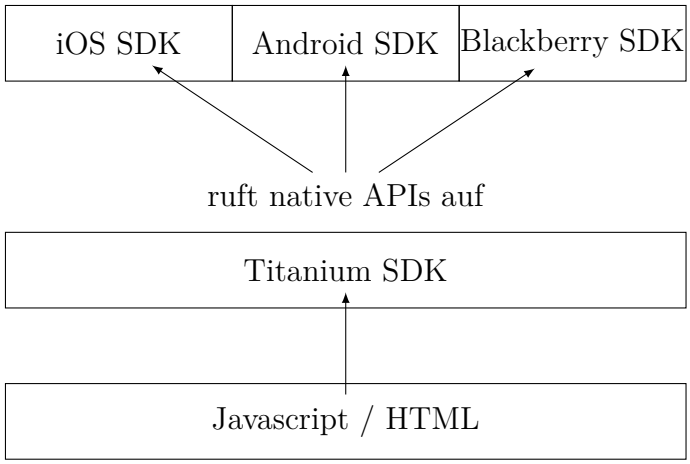


Abbildung 2.3: Titanium Projekt Architektur [vgl. Appcelerator 2014b]

welcher auf APIs von Titanium und dem Alloy-Framework zugreift. Benutzeroberflächen können programmatisch erzeugt oder in XML-Dateien deklariert werden. Die Gestaltung der Bedienelemente erfolgt in CSS-ähnlicher Syntax in .tss-Dateien (Titanium Style Sheets).

Aufgrund der verallgemeinerten API können Teile der Benutzeroberflächendeklaration auf allen Plattformen verwendet werden, während bestimmte Aspekte auch plattformspezifisch angepasst werden können. Aus diesem Grund stellt Titanium hierbei einen Kompromiss aus Xamarins plattformspezifischen UIs und PhoneGaps meist plattformübergreifenden Benutzeroberflächen dar.

Appcelerator liefert im Rahmen des SDKs mehrere Werkzeuge aus. Zum einen eine Kommandozeilenanwendung, mit welcher unter Einbeziehung der nativen SDKs aus JavaScript Quellcode, Konfigurationsdateien und weiteren Ressourcen die Anwendungsformate der verschiedenen Plattformen erzeugt und ausgeführt werden können. Weiterhin wird auch die kostenlose Entwicklungsumgebung Titanium Studio bereitgestellt, welche auf die CLI aufbaut.

Je nach plattformspezifischem Aufwand können mit Titanium erzeugte Anwendungen ihren nativ entwickelten Pendants in visueller und funktionaler Hinsicht sehr nahe kommen.

#### API

Das Titanium SDK stellt in Form verschiedener Module Funktionalität für die Entwicklung von mobilen Anwendungen bereit. Das Modul *Titanium.UI* beinhaltet Klassen für das Erzeugen von Benutzeroberflächen auf verschiedenen Plattformen. Diese Elemente sind für alle unterstützen Plattformen verfügbar und stellen eine verallge-

meinernde, übergreifende API zur Verfügung. Zusätzlich werden häufig Zusatzfunktionen angeboten, die das Nutzen von plattformspezifischen Eigenheiten ermöglichen. Ergänzend werden in *Titanium.UI.iOS* bzw. *Titanium.UI.Android* Bedienelemente bereitgestellt, welche nur für die jeweilige Plattform verfügbar sind [vgl. Appcelerator 2014d].

Zusätzlich werden unter anderem plattformübergreifende Module für die Nutzung von sqlite-Datenbanken, Kartenanzeige, Geräteortung, Dateisystemzugriff, Kalender, Kontakte, Abfrage von Systeminformationen sowie Facebook-Einbindung angeboten. Auch deren Implementation sind in bestimmten Details plattformabhängig.

Darüber hinaus bieten Appcelerator und Drittanbieter weitere kostenfreie oder -pflichtige Module im Appcelerator Marketplace an [vgl. Appcelerator 2014e].

Ergänzend kann auch die API des MVC-Frameworks Alloy angesprochen werden. Diese bietet zum Beispiel Funktionen zur Erzeugung von Animationen und Dialogen und Hilfsbibiliotheken für Datumsformate und die Manipulation von Zeichenketten [vgl. Appcelerator 2014f].

## 2.3 Übersicht der theoretischen Eigenschaften

| *Aspekt* | *iOS SDK* | *Android SDK* | *PhoneGap* | *Xamarin* | *Titanium* |
|---|---|---|---|---|---|
| unterstützte Plattformen | iOS | Android | iOS, Android, Windows Phone, Blackberry, webOS, Tizen, Amazon Fire, Firefox OS | iOS, Android, Windows Phone, Windows, Mac OS X | iOS, Android, Blackberry OS |
| Technologie | nativ, Objective-C | nativ, Java | nicht-nativ, HTML, CSS, Javascript | quasi nativ, C# | nicht-nativ, Javascript |
| Zugriff auf Geräte-API | vollständig | vollständig | begrenzt (native Plugins) | vollständig (API-Bindings) | begrenzt (eigene API) |
| Kosten | kostenfrei | kostenfrei | kostenfrei (für Basisfkt.) | kostenpflichtig (Abo-Modell) | kostenfrei (für Basisfkt.) |

Tabelle 2.1: Die theoretischen Eigenschaften der besprochenen Frameworks

# 3 Entwicklung einer mobilen Smartphone-Anwendung

## 3.1 Konzept

Um im Vergleich mobiler Entwicklungsformen in Kapitel 4 neben theoretischen Informationen auch auf praktische Erkenntnisse zurückgreifen zu können, sollen als Teil dieser Arbeit auch Anwendungen auf Basis der vorgestellten Entwicklungswerkzeuge entstehen. Um Vergleichbarkeit gewährleisten zu können, soll eine funktional klar definierte App-Idee unter Berücksichtigung der für die Frameworks typischen Vorgehensweisen umgesetzt werden.

Während bei der Nutzung plattformgebundener Entwicklungswerkzeuge jeweils nur eine Anwendung für eine Plattform erzeugt wird, entstehen bei der Verwendung von Cross-Platform Frameworks Apps für mehrere Plattformen. Um den Umfang der Arbeit zu begrenzen, werden nicht alle möglichen Zielplattformen zum Vergleich herangezogen. Die Tabelle 3.1 veranschaulicht, welche relevanten Plattformen theoretisch von den Frameworks abgedeckt werden, während in grün hervorgehoben wird, welche Apps im Rahmen dieser Arbeit entwickelt und verglichen werden.

| *Plattform* | *iOS SDK* | *Android SDK* | *PhoneGap* | *Xamarin* | *Titanium* |
|---|---|---|---|---|---|
| iOS | ✓ / ✓ | x | ✓ / ✓ | ✓ / ✓ | ✓ / ✓ |
| Android | x | ✓ / ✓ | ✓ / ✓ | ✓ / ✓ | ✓ / ✓ |
| Windows Phone | x | x | ✓ | ✓ | x |
| Blackberry OS | x | x | ✓ | x | ✓ |

Tabelle 3.1: Unterstützte Plattformen der SDKs (schwarz) und geplante Apps (grün)

## 3.2 Definition von Anforderungen einer üblichen mobilen App und Entwicklung einer App-Idee

Damit realitätsbezogene Erkenntnisse aus der Entwicklung gezogen werden können, muss die zugrunde liegende App-Idee funktional möglichst repräsentativ für die Anforderungen üblicher Apps sein. Dem Statistikportal Statista zu Folge sind Spiele im Apple App Store die beliebteste Kategorie, darauf folgen Bildung, Business, Lifestyle, Unterhaltung und Dienstprogramme [vgl. Statista 2014]. Typische Apps in der Kategorie Spiele verwenden meist keine nativen Benutzeroberflächen und Bedienelemente und erzeugen diese stattdessen mit Hilfe von Spiele-Engines (Unity, Unreal Engine etc.) oder auch direkt mit hardwarenahen Technologien wie Open-GL. Da dieser Ansatz stark von der Herangehensweise der restlichen Kategorien abweicht, wird diese Kategorie bei der Ermittlung des funktionalen Umfangs der Beispiel-App nicht berücksichtigt.

Apps in den verbleibenden Kategorien haben häufig folgende Aspekte gemeinsam:

- Sie nutzen Elemente wie Buttons, Textfelder oder Dropdown-Menüs zur Interaktion mit dem Nutzer
- Sie beziehen häufig dynamisch Daten von Web-Schnittstellen und -Diensten
- Sie stellen Daten häufig in Listenform dar
- Sie speichern Daten oder Einstellungen auf dem Gerät
- Sie verwenden Informationen wie Ort und Beschleunigung von Geräte-Sensoren
- Sie nutzen Animationen, um bestimmte Vorgänge ansprechender zu gestalten und zu verdeutlichen
- Sie sind in mehreren Sprachen verfügbar (Lokalisierung)
- Sie delegieren Aufgaben an andere Apps (Link im Browser öffnen, Email versenden)
- Sie nutzen externe Bibliotheken, welche der App über das SDK hinausgehende Funktionalität bieten

Dieser Funktionsumfang soll auch von den Beispiel-Apps abgebildet werden, welche entwickelt werden, um die Frameworks auch in praktischer Hinsicht vergleichen zu können. In Folge dessen ist die Wahl auf eine Geolocation gestützte App gefallen, welche dem Nutzer Wikipedia-Artikel anzeigen soll, die einem Ort im Umkreis seines Standorts zugeordnet sind. Der Projektname dieser App lautet ”GoodToKnow”.

Als Datenquelle der App dient die von Entwickler Ben Dodson angebotene Web-API Wikilocation [vgl. Dodson 2014]. Wöchentlich generiert diese einen Index der mit einem Ort verknüpften Wikipedia-Artikel. Unter Angabe diverser Parameter wie Längen- und Breitengrad, Radius und Sprache kann dieser Schnittstelle eine HTTP-GET Anfrage gesandt werden, die als Antwort eine Liste verfügbarer Artikel und ihrer Metadaten (Titel, Typ, Entfernung, URL etc.) im XML- oder JSON-Format zurückerhält. Der Standort des Nutzers wird über den GPS-Sensor des Geräts ermittelt.

Die App greift auf die jeweils von der Plattform oder dem gewählten Framework bereitgestellten Bedienelemente zurück. Ein Dropdown-Menü ermöglicht es dem Nutzer auszuwählen, welche Sprachversion von Wikipedia durchsucht werden soll. Ein darunter angeordneter Button startet die Suche nach Artikeln (vgl. Abbildung 3.1). Der verbleibende Teil des Bildschirms wird von einer Karte ausgefüllt, auf der sich ein Button befindet, mit welchem die eigene Position auf der Karte fokussiert werden kann. Als Kartenanbieter wurde Google Maps gewählt, da für verschiedene Plattformen SDKs zur Einbindung angeboten werden. Zudem erhöht die hohe Verbreitung dieser Lösung die Wahrscheinlichkeit, dass sie auch im Rahmen der Cross-Platform Frameworks genutzt werden kann.

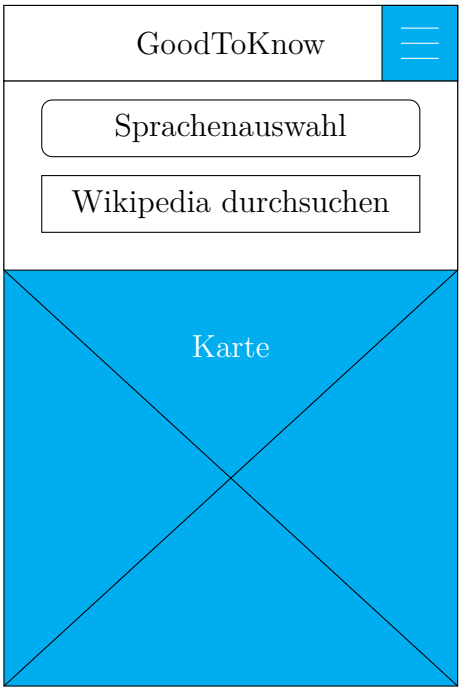


Abbildung 3.1: Geo-Wikipedia-App Mockup - Hochformat

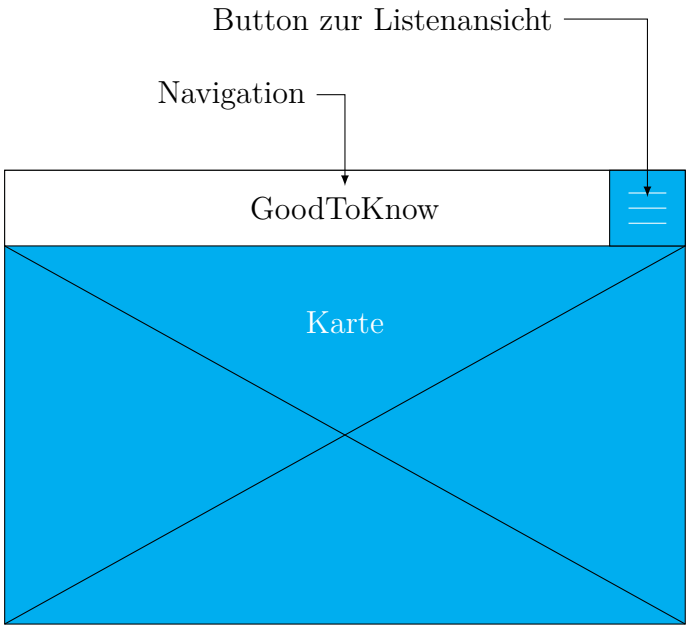


Abbildung 3.2: Geo-Wikipedia-App Mockup - Querformat

Von der Schnittstelle erfragte Artikel werden als Markierungen auf der Karte eingetragen. Zusätzlich können sie nach Berühren des Menü-Buttons in der oberen rechten Ecke auch in einer Listenansicht, sortiert nach Entfernung vom eigenen Standort, angezeigt werden. Bei Auswahl eines Eintrags über die Karte oder in der Liste wird die URL des entsprechenden Wikipedia-Artikels innerhalb der App in einer WebView geladen und angezeigt. An dieser Stelle soll es auch möglich sein, die Webseite auf Knopfdruck im externen Browser des Geräts aufzurufen. Sowohl der Fortschritt des initialen Suchvorgangs als auch der Ladevorgang der Webseite werden durch einen Fortschrittsindikator angezeigt.

Für die beschrifteten Elemente der Anwendung sind Texte in Englisch und Deutsch hinterlegt. Die Anzeigesprache orientiert sich an der Spracheinstellung des ausführenden Betriebssystems.

Weiterhin verändert sich entsprechend der Orientierung des Geräts das Layout der Benutzeroberfläche. Wird das Gerät in das Querformat versetzt, wird die aus Sprachauswahl und Suchknopf bestehende Werkzeugleiste im Rahmen einer Animation ausgeblendet, um mehr von der Karte zeigen zu können (vgl. Abbildung 3.2). Bei Rückkehr in das Hochformat wird die Werkzeugleiste ebenfalls animiert in ihren Ausgangszustand zurückversetzt.

Zuletzt wird noch der Aspekt des Speicherns von Daten auf dem Gerät abgedeckt. Da das Speichern von Daten (in diesem Fall z.B. von Wikipedia-Artikeln) in einer Datenbank den ohnehin schon umfangreichen Anforderungskatalog der Anwendung sprengen würde, wird ein schlichteres Verfahren für diese Anforderung herangezogen: Die zuletzt im Dropdown-Menü ausgewählte Wikipedia-Sprache wird gespeichert, um die Auswahl beim nächsten Start der Anwendung wieder herstellen zu können. Gespeichert wird diese Information in der jeweils verfügbaren Key-Value-Storage-Lösung des Frameworks. Key-Value-Storage bedeutet das Speichern von simplen Wertepaaren: Zum Speichern eines Wertes wird ein Schlüssel übergeben, mit welchem der zu speichernde Wert identifiziert und auch wieder abgefragt werden kann.

## 3.3 Entwicklung

Im folgenden wird der Entwicklungsablauf der Geo-Wikipedia-App mit den vorgestellten Frameworks beschrieben. Hierbei werden neben den in Kapitel 3.2 erarbeiteten Anforderungen an die App auch Aspekte wie der Funktionsumfang der Entwicklungswerkzeuge und die Verfügbarkeit von Dokumentation sowie Entwicklungs-Community betrachtet.

Zum Zeitpunkt der Entwicklung wurden die jeweils aktuellsten Versionen der folgenden Software-Produkte verwendet:

- Apple iOS: iOS SDK 7.1, Xcode 5.1.1 sowie Jetbrains AppCode 3.0
- Google Android: Android SDK 4.4.2 (API 19), Jetbrains IntelliJ IDEA 12.1.6
- Adobe PhoneGap: PhoneGap 3.4, jQuery-Mobile 1.4.2, Jetbrains IntelliJ IDEA 12.1.6
- Xamarin: Xamarin.Android 4.12, Xamarin.iOS 7.2, Xamarin Studio 4.2.5
- Appcelerator Titanium: Titanium SDK 3.2.3, Titanium Studio 3.2.3

### 3.3.1 Benutzeroberfläche der Anwendung

Bei der Entwicklung der Apps wurde als erstes stets ein Großteil des User-Interface konstruiert und dieses im Nachhinein mit der Anwendungslogik verbunden.

#### Benutzeroberfläche

Alle fünf Kandidaten bieten neben der programmatischen auch Möglichkeiten zur deklarativen Erzeugung von Benutzeroberflächen. Dies bedeutet, dass die Hierarchie von Elementen in der Benutzeroberfläche und ihre Eigenschaften nicht im Programmcode (vgl. Abbildung 3.2) sondern vorab in statischen Dateien - häufig im XML-Format - beschrieben werden (vgl. Abbildung 3.1). Dies eignet sich vor allem für relativ statische Teile der Benutzeroberfläche und erleichtert die Trennung zwischen Oberflächengestaltung und Programmlogik.

Deshalb wurde das Grundgerüst der Benutzeroberfläche in allen Fällen auf deklarative Weise erzeugt. Mit Ausnahme von Titanium und PhoneGap stellen alle Frameworks und ihre entsprechenden IDEs Editoren bereit, mit welchen die deklarativen Layout-Formate über eine grafische Oberfläche manipuliert und ohne Ausführung der Anwendung angeschaut werden können. Im Fall von PhoneGap ist das Fehlen nur bedingt

```
<LinearLayout
          android:orientation="vertical"
          android:layout_width="match_parent"
          android:layout_height="100dp"
          android:id="@+id/toolbar">

      <Button
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:text="@string/button_search"
              android:id="@+id/searchButton"
              android:layout_weight="1"/>
  </LinearLayout>
```

Code 3.1: Deklarative Layout-Erzeugung (Android SDK)

```
LinearLayout linearLayout = new LinearLayout(this);
linearLayout.setOrientation(LinearLayout.VERTICAL);
int heightInDp =  (int)
   TypedValue.applyDimension(TypedValue.COMPLEX_UNIT_DIP,
   100,getResources().getDisplayMetrics());
linearLayout.setLayoutParams(new
   LinearLayout.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT,
   heightInDp));

Button searchButton = new Button(this);
searchButton.setLayoutParams(new
   LinearLayout.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT,
   ViewGroup.LayoutParams.MATCH_PARENT, 1f));
searchButton.setText(getString(R.string.button_search));

linearLayout.addView(searchButton);
```

Code 3.2: Programmatische Layout-Erzeugung (Android SDK)

von Nachteil, da die erzeugte Anwendung relativ schnell im lokalen Browser betrachtet werden kann. Für Titanium stellt dies jedoch einen erheblichen Nachteil dar, da das manuelle Deklarieren der Elemente und ihrer Eigentschaften sowie die Notwendigkeit, die Anwendung nach jeder Änderung kompilieren und ausführen zu müssen, viel Zeit in Anspruch nimmt. Ein Nachteil des für native iOS-Apps verwendeten Layout-Formats .xib ist, dass sich dies im Gegensatz zu den Alternativen in den meisten Fällen nicht zum manuellen Editieren eignet.

Die auf diese Weise erzeugten Layouts sind im Fall von nativer iOS- und Android-Entwicklung nicht untereinander kompatibel. Dies gilt entsprechend auch für Layouts unter Xamarin, welches auch die nativen Formate verwendet. Die mit Titanium erzeugten Deklarationen sind hingegen grundsätzlich für alle Plattformen verwendbar. In der Realität muss und sollte bei der Entwicklung von UIs für Titanium jedoch auch häufig zwischen den Zielplattformen differenziert werden, da sich einige Elemen-

te je nach Plattform anders verhalten oder gar nicht verfügbar sind. In diesen Fällen können Elemente plattformabhängig deklariert werden:

```
1  <Picker id="picker" platform="android">
2  <TabbedBar id="tabbedbar" platform="ios" />
```

Code 3.3: Plattformabhängige Deklaration mit Titanium

So sollte mit Titanium ursprünglich das Bedienelement *Picker* auf beiden Plattformen verwendet werden. Die *UIPickerView*, welche dieses unter iOS repräsentiert, hat jedoch andere Dimensionen und funktioniert anders als das Android Pendant, sodass die analoge Nutzung wenig Sinn macht. Stattdessen wird nun nur unter Android das *Picker* Bedienelement verwendet, während unter iOS *TabbedBar* zum Einsatz kommt.

Die Art der Deklaration von Benutzeroberflächen mit PhoneGap hängt im Detail vom verwendeten Web-Framework ab. Naturgemäß ist sie aufgrund der genutzten Web-Technologien prinzipiell plattformunabhängig. Dennoch kann und sollte sie plattformspezifisch getestet und angepasst werden, da sich die nativen WebView-Implementationen der Plattformen in Details unterscheiden. So wird das HTML-Dokument je nach Plattform nicht zwingend identisch dargestellt, Bedienelemente verhalten sich unterschiedlich oder die Browser-API stellt bestimmte Funktionen anders oder gar nicht bereit. Für die GoodToKnow PhoneGap-App wurde jQuery Mobile, ein verbreitetes Entwicklungs-Framework für mobile Benutzeroberflächen, verwendet.

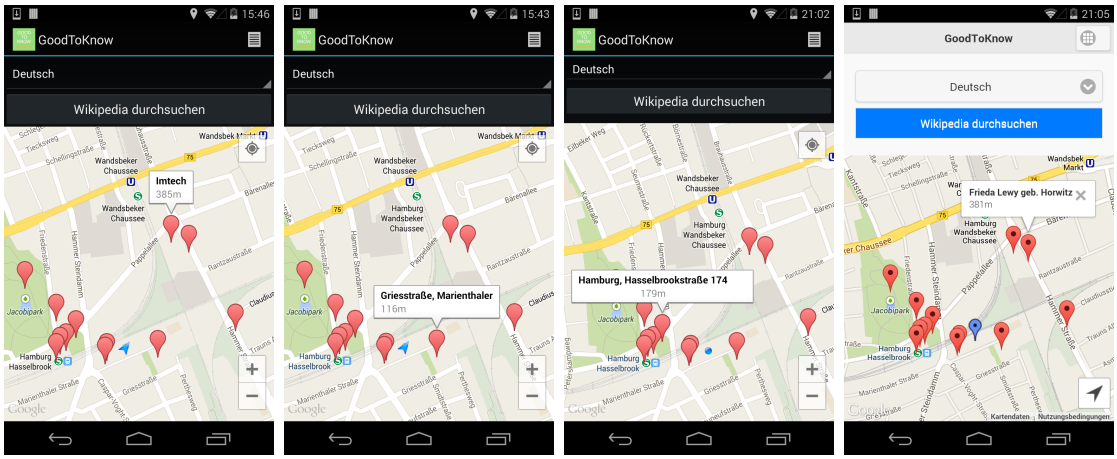


Abbildung 3.3: Hauptansicht der Android-Apps (nativ, Xamarin, Titanium, PhoneGap), deutsche Lokalisierung

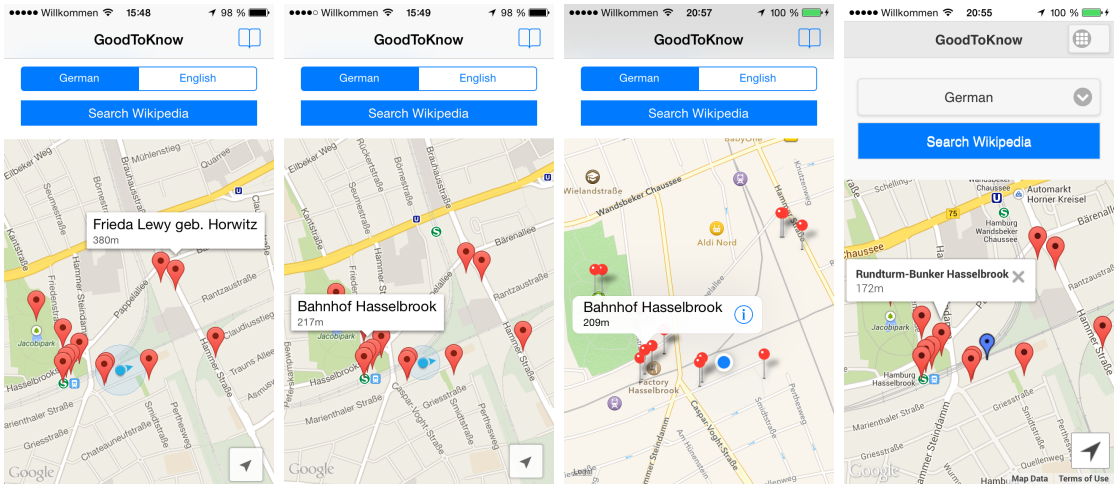


Abbildung 3.4: Hauptansicht der iOS-Apps (nativ, Xamarin, Titanium, PhoneGap), englische Lokalisierung

### Lokalisierung

Apps sollten aufgrund ihrer Verfügbarkeit und Reichweite in internationalen App-Stores häufig in mehreren Sprachen angeboten werden. Der Prozess, Text-Elemente einer Anwendung in verschiedenen Sprachversionen zu hinterlegen, nennt sich Lokalisierung. Viele Frameworks stellen Mechanismen bereit, um diesen zu vereinfachen. Meist werden Textteile für jede zu unterstützende Sprache unter Angabe eines Schlüssels hinterlegt und über diesen referenziert. Die Anzeigesprache richtet sich hierbei häufig nach der eingestellten Systemsprache des ausführenden Geräts - in selteneren Fällen ist es auch möglich, die Sprache manuell in der App zu wählen. Abbildung 3.3 zeigt die deutsche und Abbildung 3.4 die englische Lokalisierung der App.

Für eine vollständige Lokalisierung müssen in deklarierten Layouts verwendete und im Code dynamisch erzeugte Textteile sprachenabhängig generiert werden. Weiterhin können die unterschiedlichen Textlängen verschiedener Sprachen Auswirkungen auf das Layout haben, die bedacht werden müssen. Mit dem Android SDK und Titanium können lokalisierte Texte direkt in den Layout-Dateien referenziert werden (vgl. Abbildung 3.4).

```
1  <!-- Android SDK -->
2  <Button android:text="@string/key" />
3  <!-- Titanium -->
4  <Button title="L('key')"/>
```

Code 3.4: Lokalisierung in Layout-Deklaration

Zum aktuellen Zeitpunkt können mit dem InterfaceBuilder für iOS erstellte Layouts nicht so direkt lokalisiert werden. Stattdessen muss für jede Sprache eine Version des Layouts angelegt werden. In Folge dessen müssen Veränderungen am Layout stets auf

alle Versionen übertragen werden - hierbei hilft Apples Kommandozeilenanwendung ibtool [vgl. Apple 2014c]. Mit der im Juni 2014 veröffentlichten Xcode 6 Beta liefert Apple umfangreiche Verbesserungen im Bezug auf das Thema Lokalisierung nach - speziell auch das Lokalisieren direkt im InterfaceBuilder [vgl. Apple 2014d].

Xamarin nutzt wie bereits erwähnt die nativen Layoutformate von Android und iOS und unterliegt somit auch deren Restriktionen. PhoneGap liefert von Haus aus hingegen keine Werkzeuge zur Lokalisierung, sodass die Verfügbarkeit solcher Mechanismen ganz von der Wahl des als Grundlage genutzten Web-Frameworks abhängt. Für die vorliegende App wurde das *jQuery Localisation Plugin* verwendet [vgl. Wood 2013]. Mit diesem Plugin können Texte auf programmatische Weise lokalisiert werden, wie es auch bei allen anderen Frameworks möglich ist.

```
1  //iOS SDK
2  NSString *localizedString = NSLocalizedString(@"key", nil);
3  //Android SDK
4  String localizedString = getString(R.string.key);
5  //PhoneGap mit jQuery Localisation
6  var localizedString = KEY;
7  //Xamarin iOS
8  NSBundle.MainBundle.LocalizedString("key", null);
9  //Xamarin Android
10 string localizedString = GetString(Ressource.String.key);
11 //Titanium
12 var localizedString = L("key");
```

Code 3.5: Lokalisierung im Code

Aus Abbildung 3.5 ist ersichtlich, dass Xamarin von Haus aus kein einheitliches, plattformübergreifendes Lokalisierungssystem anbietet. Dies ist besonders problematisch, da ein Textelement unter iOS über einen einfachen String als Schlüssel referenziert wird, während Android eine Integer-Konstante wie *R.string.key* verwendet, welche während des Kompilierens vom Tool aapt für alle Ressourcen (Strings, Layouts, Abbildungen etc.) generiert wird [vgl. Google 2014e]. Dadurch wird erschwert, in plattformübergreifendem Code lokalisierte Strings zu referenzieren. Eine Alternative ist es, die Lokalisierungsmöglichkeiten des .NET-Frameworks in Anspruch zu nehmen [vgl. Microsoft 2014]. Dies hat jedoch zur Folge, dass dort hinterlegte Übersetzungen nicht ohne zusätzlichen Aufwand innerhalb der deklarativen Layouts verwendet werden können.

### Animation

Auch das Thema Animation hat durch Apps an Bedeutung gewonnen: Viele Vorgänge innerhalb einer Anwendung werden animiert, um diese dem Nutzer verständlicher zu machen und auch einen ansprechenden Gesamteindruck zu vermitteln. Bei der vorliegenden App wird die aus Sprachauswahl und Suchknopf bestehende Werkzeugleiste je nach Orientierung animiert ein- oder ausgeblendet.

```
1  //iOS SDK
2  [UIView animateWithDuration:0.5f animations:^{
3    mapView.frame = self.view.frame;
4      CGRect frame = controlsView.frame;
5        frame.origin.y = - frame.size.height;
6        controlsView.frame = frame;
7    }];
8
9  //Android SDK
10 ValueAnimator animator = ValueAnimator.ofInt(fromY, toY);
11   animator.addUpdateListener(new ValueAnimator.
       AnimatorUpdateListener() {
12     @Override
13       public void onAnimationUpdate(ValueAnimator valueAnimator) {
14         int val = (Integer) valueAnimator.getAnimatedValue();
15         toolbar.getLayoutParams().height = val;
16         toolbar.requestLayout();
17       }
18 });
19 animator.setDuration(500);
20 animator.start();
21
22 //Xamarin iOS und Android analog zu nativen Pendants
23
24 //PhoneGap mit jQuery
25 $("#toolbar").slideUp(500, function() {
26     scaleContentToDevice();
27   });
28
29 //Titanium
30 var animation = Titanium.UI.createAnimation();
31 animation.duration = 500;
32 animation.height = orientation.isPortrait() ? defaultheight : 0;
33 $.toolbar.animate(animation);
```

Code 3.6: Benutzeroberfläche animieren

Das iOS SDK bietet eine sehr kompakte und gleichzeitig vielfältige API für die Animation von Bildschirminhalten (Views). In einer kompakten Syntax (Objective-C *Blocks*) können Parameter wie die Koordinaten, Maße, Transformation (Skalierung, Rotation), Transparenz sowie Hintergrundfarbe der View manipuliert werden. Die innerhalb des Ausdrucks vorgenommenen Änderungen werden innerhalb des angegebenen Zeitraums anhand einer optional anzugebenden Interpolationsfunktion vom Ausgangs-

wert zum neuen Wert animiert. Für darüber hinausgehende Ansprüche steht weiterhin die *CoreAnimation* Bibliothek bereit.

Im Rahmen des Android SDKs können Animationen wahlweise programmatisch erzeugt oder wie Layouts in XML-Dateien deklariert werden. Ähnlich der iOS API können mit der Klasse *ObjectAnimator* bestimmte Eigenschaften wie die Translation, Rotation, Skalierung und Transparenz von Objekten von einem zu einem anderen Wert animiert werden. Hierbei wird jedoch nicht automatisch das vollständige Layout aktualisiert, sodass die Karte in der vorliegenden App den zuvor von der Werkzeugleiste ausgefüllten Platz nicht automatisch einnimmt. Dies erfordert eine direkte Manipulation der Layout-Parameter mit Hilfe der Klasse *ValueAnimator*, mit welcher über einen Wertebereich iteriert und in jedem Iterationsschritt der Wert im eigenen Animationscode verwendet werden kann (vgl. Code 3.6).

Wie alle anderen die Benutzeroberfläche betreffenden Aspekte ist auch Animation mit Xamarin plattformspezifisch mit den nativen APIs umzusetzen. Somit wurde die Animation in den Xamarin iOS und Android Apps genau wie in ihren nativen Pendants umgesetzt.

Mit dem Titanium SDK wird hingegen eine plattformübergreifende Animations-API bereitgestellt. Diese ermöglicht es, ähnlich dem iOS-SDK Änderungen an Eigenschaften in einer übersichtlichen Syntax zu animieren. Die Animation hiermit funktioniert in der Android-App problemlos, während die Animation unter iOS bei gleichem Code nur einmal stattfindet und danach nicht mehr ausgeführt wird. Ausführliche Recherche und Lösungsversuche führten im Rahmen des Projekts bisher zu keiner Verbesserung, sodass das Ausblenden der Werkzeugleiste in der iOS-Variante der Titanium-App derzeit ohne Animation geschieht.

Im Web-Projekt der PhoneGap-App wurde die Animation mit Hilfe der jQuery-Funktionen *slideUp* bzw. *slideDown* umgesetzt, welche exakt die benötigte Animation bereitstellen. Darüber hinaus können mit einer allgemeinen *animate*-Funktion des jQuery-Frameworks auch diverse CSS-Eigentschaften animiert verändert werden, insofern sie als numerische Werte angegeben werden (z.B. Breite, Höhe oder Schriftgröße) [vgl. jQuery 2014].

### 3.3.2 Logik der Anwendung

Nachdem die Benutzeroberfläche der Apps in groben Zügen stand, wurde die Logik der Anwendungen - also die Verarbeitung von Benutzereingaben, das Abfragen von Sensoren, das Anfordern von Schnittstellendaten sowie benötigte Navigationslogik - implementiert.

#### Web-Schnittstellen

Essentiell für die Funktionsweise der App sind die Daten der Schnittstelle WikiLocation, welche für ein gegebenes Wertepaar aus Längen- und Breitengrad umliegende Wikipedia-Artikel liefert. Hierzu muss eine übliche HTTP-GET Anfrage an den unter `http://api.wikilocation.org/` erreichbaren Dienst gesandt werden und die im JSON-Format vorliegende Antwort ausgewertet werden.

Damit die Ausführung dieser Anfrage nicht den flüssigen Betrieb der Anwendung, also zum Beispiel die animierte Darstellung eines Ladebalkens, beeinträchtigt, muss diese asynchron - also parallel zur Ausführung der eigentlichen Anwendung - stattfinden.

Hierzu stellen alle Frameworks entsprechende APIs bereit. Dennoch wurden unter iOS und Android aus Gründen des Komforts die weit verbreiteten Netzwerkbibliotheken *AFNetworking* [vgl. Thompson 2014] bzw. *Android Asynchronous Http Client* [vgl. Smith 2014] verwendet. Auch für die PhoneGap basierte App wurde nicht auf die von JavaScript mitgelieferte Klasse *XMLHttpRequest* zurückgegriffen und stattdessen die vom verwendeten Web-Framework jQuery mitgelieferten Ajax-Methoden verwendet. Unter Titanium wurde die bereitgestellte, plattformübergreifende Klasse *Titanium.Network.HTTPClient* verwendet, welche auf die bereits erwähnte JavaScript-Klasse XMLHttpRequest aufbaut. In der Xamarin-App wurde die Anfrage im Gegensatz zum Oberflächencode nicht mit den nativen Möglichkeiten des iOS- oder Android SDKs umgesetzt, da diese Funktionalität wiederverwendbar und plattformübergreifend in einer zentralen Hilfsklasse implementiert werden sollte. Aus diesem Grund wurde die Klasse *System.Net.HttpWebRequest* des .NET-Frameworks verwendet.

#### Speichern von Daten

Alle verwendeten Frameworks bieten unter anderem auch APIs zur Nutzung des unter mobilen Anwendungen verbreiteten Datenbank-Typs SQLite. Wie in Kapitel 3.2 erläutert, wird in diesem Fall jedoch zum Speichern von Daten die Nutzung simpler Key-Value-Storage-Lösungen gewählt, welche für die Anforderungen der Anwendung ausreichen.

Unter Android und iOS wurden hierfür die mitgelieferten APIs *SharedPreferences* bzw. *NSUserDefaults* verwendet. Im Rahmen des PhoneGap-Projekts wurde die durch HTML5 eingeführte Browser-API *LocalStorage* genutzt. Während Titanium in Form von *Ti.App.Properties* eine einfach zu nutzende und plattformübergreifende API für diesen Zweck mitliefert, tut dies Xamarin von Haus aus nicht. Im Fall der vorliegenden App ist das Speichern der Wertepaare Teil des plattformspezifischen Oberflächencodes, sodass an dieser Stelle einfach auf die Xamarin-Bindings der mitgelieferten nativen APIs zurückgegriffen wurde. In Fällen, wo das Speichern von Wertepaaren plattformübergreifend stattfinden muss, bietet der Xamarin Component Store hierfür in Form von *SimpleStorage* [vgl. Xamarin 2014f] eine kostenfreie und plattformübergreifende Lösung eines Drittanbieters.

Die Funktionsweise der beschriebenen APIs ähnelt sich sehr. Unter Angabe eines Schlüsselworts können Werte persistent gespeichert und über diesen zu einem späteren Zeitpunkt - auch nach Neustart der Anwendung - wieder abgerufen werden:

```
1  window.localStorage.setItem("wiki_lang",select.val());
2  var wikilang = window.localStorage.getItem("wiki_lang");
```

Code 3.7: Speichern und Abfragen eines Wertes am Beispiel der Browser-API localStorage

Die APIs des Android- und Titanium-SDKs erlauben es, hierbei einen Standardrückgabewert anzugeben, für den Fall, dass für den spezifizierten Schlüssel kein Wert hinterlegt wurde - zum Beispiel beim ersten Start der Anwendung. Dies ist ein Alleinstellungsmerkmal und wird von den anderen APIs nicht angeboten.

#### Nutzung der Bibliotheken von Drittanbietern

Im Rahmen von Softwareentwicklung wird häufig auf Bibliotheken von Dritten zurückgegriffen, welche die Umsetzung bestimmter Anforderungen vereinfachen, beschleunigen und gegebenenfalls sogar erst ermöglichen. Somit ist es für die erfolgreiche App-Entwicklung wichtig, dass die für ein Projekt benötigten Bibliotheken verfügbar sind und in das gewählte Entwicklungsframework eingebunden werden können.

In die vorliegende App sollen die Karten von Google Maps integriert werden. Für die Plattformen iOS und Android stellt Google hierfür native Bibliotheken bereit. Unter Android sind Google Maps Teil des *Google Play services SDK*s, welches auf Geräten ab Android-Version 2.2 vorinstalliert bzw. automatisch nachinstalliert wird. Für iOS wird das Google Maps SDK zur Einbindung in das native Entwicklungsprojekt angeboten. Weiterhin steht die Google Maps JavaScript API bereit, mit welcher die Kartenfunktionalität in der PhoneGap-App implementiert wird. Die API der Bibliotheken für

iOS- und Android ist relativ ähnlich und erfüllt die notwendigen Anforderungen. Die JavaScript API bedient hingegen nicht alle Anforderungen: So nutzt die Anwendung die Entfernung zwischen linkem und rechtem Kartenrand zur Ermittlung des Radius, in dessen Umkreis nach Artikeln gesucht werden soll. Während die iOS- und Android-API Längen- und Breitengrad des linken und rechten Kartenrands preisgibt und eine Methode mitliefert, welche die Entfernung in Metern zwischen diesen berechnen kann, muss diese Funktionalität bei Nutzung der JavaScript-API eigens implementiert werden. Auch ein Button, mit welchem die aktuellen Position auf der Karte zentriert werden kann, muss der JavaScript Version von Google Maps manuell hinzugefügt werden, während dieser in den iOS- und Android-Versionen leicht über Angabe eines Parameters eingeblendet werden kann.

Die nativen Bibliotheken für iOS und Android können unter Xamarin nicht einfach aus C#-Code heraus aufgerufen werden. Stattdessen stellt Xamarin in ihrem Components Store aber die Komponenten *Google Maps* für iOS und *Google Play Services* für Android bereit, welche über Bindings die Nutzung der nativen APIs in vollem Umfang ermöglichen.

Appcelerator bietet für Titanium in Form des *ti.map Moduls* auch Kartenfunktionalität an. Im Gegensatz zum Ansatz von Xamarin, wo die jeweilige API plattformspezifisch angesprochen werden muss, stellt *ti.map* eine plattformübergreifend einheitliche API bereit. Diese greift jedoch nur unter Android auf Google Maps zurück, während unter iOS die im Betriebssystem enthaltene Apple Maps-API verwendet wird. Dies macht Sinn, hat aber zur Folge, dass hiermit nicht auf allen Plattformen die gewählte Karten-Implementation verwendet werden kann und auch auf einige Funktionen verzichtet werden muss. So bietet *ti.Map* im Gegensatz zu den nativen Pendants zum Beispiel nicht die Möglichkeit, Klick-Ereignisse auf Kartenmarkierungen zu registrieren. Stattdessen können nur Ereignisse der ganzen Karte abgefangen werden, welche umständlich nach Quelle gefiltert werden müssen. Zusätzlich bieten die unter iOS verwendeten Apple Maps Funktionen, wie das Einblenden eines Buttons, über den man seinen Ort aufrufen kann, nicht an. Die Integration des Google Maps SDKs in der Titanium-basierten iOS-App wäre grundsätzlich mit dem von einem Drittanbieter im Appcelerator Marketplace angebotenen Modul möglich [vgl. Triphase 2014]. Dies ist jedoch kostenpflichtig und würde es zudem notwendig machen, die Kartenfunktionalität für iOS und Android plattformspezifisch umzusetzen.

#### Sensoren

Hardwaresensoren machen die Nutzung von Smartphones zu einer facettenreichen Erfahrung, indem sie Anwendungen mit Informationen wie dem aktuellen Standort,

der Ausrichtung des Geräts im Raum oder dem Signal der Kamera versorgen. Die Funktionalität der Geo-Wikipedia-App beruht maßgeblich auf der Nutzung des GPS-Sensors, mit Hilfe dessen der Ort des Nutzers ermittelt und als Ausgangspunkt für die Suche von Artikeln genutzt wird. Der Beschleunigungssensor der Geräte ist hingegen nicht essentiell, wird aber auch genutzt, um die Benutzeroberfläche entsprechend der Geräteausrichtung anzuzeigen (siehe Kapitel 3.3.1).

Zur Ermittlung des Orts stellen die vorliegenden Frameworks APIs bereit, mit denen einmalig oder in einem regelmäßigen Interval die aktuelle GPS-Position abgefragt werden kann. Mit Ausnahme der PhoneGap-App wurden diese jedoch gar nicht in Anspruch genommen. Stattdessen boten die verwendeten Kartenimplementationen die Möglichkeit, den aktuellen Ort automatisch auf der Karte anzuzeigen. Die eigentliche Abfrage der Location-APIs übernimmt in diesen Fällen also die verwendete Kartenbibliothek.

Die JavaScript-API von Google Maps bietet diese Funktion nicht an. Stattdessen wird die mit HTML5 eingeführte Funktion der Browser-API *navigator.geolocation.watchPosition* verwendet, welche in regelmäßigen Abständen den aktuellen Längen- und Breitengrad liefert. Entsprechend diesen Angaben wird manuell eine Markierung auf der Karte vorgenommen.

Zur Abfrage der Geräteausrichtung wird in der PhoneGap App auf API der jQuery-Bibliothek zurückgegriffen, während die Titanium App die API des Moduls *Ti.Gesture* nutzt. Wie einige andere Aspekte ist auch die Rotation des Gerätes etwas, was in der Xamarin-App plattformspezifisch im Oberflächencode der Anwendung behandelt wird, sodass jeweils die nativen APIs von iOS und Android verwendet werden.

#### Aufgabendelegation

Typisch für Smartphone-Apps ist auch eine Interaktion untereinander. Soll eine Email mit bestimmtem Inhalt versandt werden, wird diese Aufgabe zum Beispiel häufig an eine installierte Email-App delegiert und nicht innerhalb der App erledigt. In der GoodToKnow-App soll es möglich sein, einen geöffneten Wikipedia-Artikel auch in einem externen Browser öffnen zu können.

Mit den SDKs von iOS, Android und Titanium kann die URL des Wikipedia-Artikels sehr einfach mit Aufruf einer API-Funktion in einem externen Browser geöffnet werden. Die Xamarin.iOS- und Xamarin.Android-App greifen hierfür erneut auf die Bindings der nativen APIs zurück. In der PhoneGap Anwendung ist dies nicht ganz so einfach, da die Anwendung selbst auf Web-Technologien und somit Links basiert. So besteht

```
1  //iOS SDK
2  [[UIApplication sharedApplication] openURL:[NSURL URLWithString:URL
       ]];
3  //Android SDK
4  Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(URL));
5  startActivity(intent);
6  //PhoneGap mit jQuery
7  window.open(URL, "_system");
8  //Titanium
9  Ti.Platform.openURL(URL);
```

Code 3.8: Öffnen einer URL im externen Browser

leicht die Gefahr, dass der Aufruf eines Links die gesamte Ansicht der Anwendung „übernimmt“. Abhilfe schafft hier das Plugin InAppBrowser, welches es erlaubt, über den Parameter *target* die Intention der URL zu definieren: *_self* öffnet die Adresse innerhalb der PhoneGap WebView, *_blank* innerhalb des InAppBrowsers, einer weiteren, nativen WebView-Instanz innerhalb der App, und *_system* öffnet die URL im externen Browser [vgl. Cordova 2014].

### 3.3.3 Entwicklungs-Ökosystem

Für den Endanwender zählen bei Nutzung einer Anwendung letztlich nur der Funktionsumfang und die Bedienbarkeit - wie das Softwareprodukt entstanden ist und auf welchen Technologien es basiert, ist für ihn grundsätzlich irrelevant. Für den Entwickler bestehen hingegen auch andere Bewertungsgrundlagen wie die Nutzbarkeit und Dokumentation der verwendeten Tools, Frameworks und APIs, sowie die sie umgebende Entwicklergemeinschaft.

#### Entwicklungsumgebung und Werkzeuge

Im Entwicklungsprozess hat die Qualität der verwendeten Werkzeuge große Auswirkungen auf die Effektivität der Entwicklung und somit darauf, wie viel Zeit in Anspruch genommen wird.

Die im iOS-SDK beinhaltete IDE Xcode beinhaltet alles, was für eine übliche Anwendungsentwicklung und -ausführung benötigt wird. Mit ihr kann die App programmiert und die dazugehörigen Oberflächen grafisch mit dem InterfaceBuilder entwickelt werden. Weiterhin sind Code-Versionverwaltung (engl. Version Control System, VCS) mit git oder SVN und die offizielle Dokumentation direkt integriert. Schwächen sind im Vergleich zur ebenfalls für die iOS-Entwicklung von Jetbrains angebotenen IDE AppCode in den Bereichen intelligenter Code-Vervollständigung und Restrukturierung (engl. refactoring) zu finden. Im Zusammenspiel mit dem iOS-Simulator können Änderungen an der App sehr schnell getestet und anfallende Fehler schnell behoben werden.

Die vom gleichen Hersteller wie AppCode entwickelte IDE IntelliJ IDEA, auf welcher auch die von Google angebotene IDE Android Studio basiert, bietet mit Ausnahme der integrierten Dokumentation ebenfalls die gerade genannte Funktionalität. In den Punkten Code-Vervollständigung und Refactoring ist sie jedoch hilfreicher, sodass noch effektiver programmiert werden kann. Unabhängig von der IDE sind die vom Android SDK gebotenen Emulatoren jedoch im Vergleich zu den iOS-Simulatoren sehr langsam, sodass vor allem das schnelle Testen von Änderungen an der App hierdurch erheblich verzögert wird. Abhilfe schafft hier die von Intel entwickelte Virtualisierungssoftware Intel HAXM, welche es ermöglicht, einen Emulator auf Basis von Intel x86-Code auszuführen, was die Nutzung des Emulators auf ein erträgliches Maß beschleunigt [vgl. Intel 2014]. Dennoch wurde die App bei der Android-Entwicklung häufig direkt auf einem echten Gerät ausgeführt und getestet.

Xamarin Studio bietet ebenfalls Code-Vervollständigung, Refactoring und VCS-Integration, stellt aber nur für das Erstellen von Android-Layouts einen Editor bereit. Für die Erzeugung von .xib-Layouts für iOS greift Xamarin Studio auf den Xcode InterfaceBuilder zurück. In der neuesten Version beinhaltet die IDE nun auch ein Design Tool für das iOS Layout-Format Storyboard, sodass dies innerhalb der Anwendung und unter Windows editiert werden kann. Da auch die Simulatoren bzw. Emulatoren der nativen SDKs verwendet werden, unterliegt die Entwicklung mit diesen auch den vorgenannten Bedingungen.

Auch Titanium Studio greift auf die Simulatoren und Emulatoren der nativen SDKs zurück. Jedoch startet es bei jeder Ausführung der iOS-App den iOS-Simulator vollständig neu und verwendet standardmäßig einen bereits laufenden Android Emulator nicht wieder, was den Entwicklungsvorgang verlangsamt. Grundsätzlich kann ein Titanium Projekt auch im Webbrowser ausgeführt werden - da jedoch auf den mobilen Plattformen auf native Bedienelemente zurückgegriffen wird, gibt die Web-Ansicht im Gegensatz zur Entwicklung mit PhoneGap kaum brauchbare Rückschlüsse. Hinzu kommt, dass für die Erzeugung von Benutzeroberflächen keine Editoren zur Verfügung stehen, sodass die von Titanium Alloy verwendeten XML-Hierarchien und .tss-Designs manuell geschrieben werden müssen und das Resultat der Änderungen erst nach Kompilieren und Ausführen der Anwendung zu sehen ist. Rudimentäre Code-Vervollständigung und VCS-Integration sind in der IDE enthalten. Neben der IDE steht auch eine Kommandozeilenanwendung zur Verfügung, mit welchem Titanium Projekte angelegt, konfiguriert, kompiliert und ausgeführt werden können. Diese ermöglicht es, ein Projekt mit einer alternativen IDE zu entwickeln und das CLI zum Kompilieren und Ausführen der entstandenen App zu verwenden.

PhoneGap empfiehlt und liefert keine IDE im Rahmen des SDKs aus. Das der App zugrunde liegende Web-Projekt kann in jeder für die Web-Entwicklung geeigneten IDE entwickelt werden und unter Einschränkungen auch in einem Desktop-Webbrowser getestet werden. So ist die GoodToKnow App zum Beispiel nahezu in vollem Funktionsumfang in einem modernen Webbrowser ausführbar, da auch hier die API zur Ermittlung des Standorts voll funktionsfähig ist. Dies beschleunigt den Entwicklungsprozess in erheblichem Maße. Grundsätzlich ist jedoch die Nutzung von Programmteilen, welche auf native Funktionen zurückgreifen, im Webbrowser nicht möglich. Demzufolge mussten zum Beispiel Funktionen, welche die Orientierung des Gerätes betreffen, in den nativen Simulatoren bzw. Emulatoren und Geräten getestet werden. Zum Anlegen, Konfigurieren und Ausführen eines PhoneGap-Projekts dient das PhoneGap CLI.

#### Dokumentation

Auf wohl strukturierte und vollständige Dokumentation eines Frameworks im Rahmen der Entwicklung zurückgreifen zu können, ist insbesondere im frühen Stadium eines Projekts nahezu unverzichtbar. Die schnelle Weiterentwicklung moderner Softwareprodukte, speziell im mobilen Bereich, und das Hinzukommen von Funktionen und Änderungen an bestehenden Funktionen machen den Griff zur Dokumentation häufig auch im späteren Verlauf eines Projekts notwendig.

Dokumentation kann verschiedene Inhalte bieten. Fast unumgänglich ist ein Referenzdokument, in welchem verfügbare Methoden und Eigenschaften der API und ihrer Klassen aufgeführt und ggf. erläutert werden. Ergänzend bietet moderne Dokumentation häufig auch ausführliche Dokumente zu konkreten Themen, welche mit Beispiel-Code und Nutzungshinweisen der Entwickler (sog. „best-practice“) angereichert sind. Darüber hinaus bieten die meisten Softwareprodukte Anleitungen für eine erste Nutzung („first-steps“), in welchen die erste Einrichtung anhand eines simplen Beispiels erläutert wird.

Für Android stellt Google alle aufgezählten Inhalte bereit [vgl. Google 2014f]. So stehen zu verschiedenen Themen ausführliche, mit Abbildungen, Tabellen und Code-Beispielen angereicherte Dokumente bereit. Weiterhin wird eine Referenz der im Android-SDK enthaltenen Klassen angeboten, welche nach verwendeter SDK-Version gefiltert werden kann und so nur für das API-Level des Projekts verfügbare API anzeigt.

Auch Apple stellt für iOS ähnlich umfangreiche Ressourcen bereit [vgl. Apple 2014e]. Neben einer umfangreichen Referenz aller Klassen werden auch umfangreiche Dokumente zu einer Vielzahl an Themen angeboten. Negativ fällt hierbei auf, dass wenig

Wert auf die Arbeit mit älteren SDKs gelegt wird. So ist zum Beispiel ein Filtern der API-Referenz nach SDK-Version nicht möglich, und in der Dokumentation wird meist davon ausgegangen, dass für die neuesten Apple Geräte und Softwareversionen entwickelt wird. Positiv hervorzuheben ist, dass die Dokumentation automatisch in der IDE Xcode integriert ist und von einer Klasse im Code direkt zu ihrer Dokumentation gesprungen werden kann.

Xamarin teilt die Dokumentation in die Rubriken Cross-Platform, Android, iOS und Mac auf, für welche jeweils Anleitungen, Code-Beispiele und Videos angeboten werden [vgl. Xamarin 2014g]. Für die einzelnen Plattformen wird jeweils eine API-Referenz angeboten. Die Recherche in dieser bietet sich vor allem an, wenn die Implementation der Xamarin.Android- oder Xamarin.iOS-APIs sich von der nativen unterscheidet. In den meisten Fällen kann auf die reichhaltige Dokumentation der Plattformanbieter zurückgegriffen werden. Beim Schreiben des Codes muss dann nur die etwas abweichende C#-Syntax verwendet werden, welche einheitlichen Mustern folgt (vgl. Code 3.9). Zusätzlich wird auch für die Entwicklung des plattformübergreifenden C#-Codes auf Basis der .NET-Bibliothek eine Referenz der .NET Base Class Library angeboten.

```
1  //iOS SDK
2  self.navigationItem.rightBarButtonItem.enabled = NO;
3  //Xamarin iOS
4  this.NavigationItem.RightBarButtonItem.Enabled = false;
5  //Android SDK
6  LinearLayout toolbar = (LinearLayout)findViewById(R.id.toolbar);
7  //Xamarin Android
8  LinearLayout toolbar = (LinearLayout)FindViewById(Resource.Id.
       toolbar);
```

Code 3.9: Sich bis auf sprachen-typische Unterschiede ähnelnde API von Xamarin und nativen SDKs

Die Titanium Dokumentation besteht aus einer Referenz der Titanium und Alloy API, Guides zu verschiedenen Themen und einem Angebot an Videos. Für plattformübergreifende APIs und Klassen wird angegeben, auf welchen Plattformen und in welchem Umfang diese unterstützt werden [vgl. Appcelerator 2014g].

Für PhoneGap steht grundsätzlich nur Dokumentation für die Nutzung des Wrappers und die dafür verfügbaren Plugins bereit [vgl. PhoneGap 2014g]. Diese ist generell nicht so umfangreich und gut illustriert wie die der anderen Kandidaten. Problematisch ist innerhalb der Dokumentation insbesondere auch die häufig synonyme Nutzung der Begriffe PhoneGap und Cordova. Während bei der Installation von Adobe PhoneGap die PhoneGap Kommandozeilen-Anwendung installiert wird, wird in Code-Beispielen

der Dokumentation seit Version 3.3.0 die Cordova-CLI verwendet. Die PhoneGap- und Cordova-CLI unterscheiden sich stellenweise: So bietet Erstere die Möglichkeit, den Adobe PhoneGap Build Service zu verwenden, während die Cordova-CLI einige nützliche Befehle wie „cordova prepare“ exklusiv bereitstellt. Diese unscharfe Trennung zwischen dem Cordova Projekt der Apache Foundation und der darauf basierenden Distribution Apache PhoneGap führt unter Entwicklern zu Verwirrung. So schreibt Entwickler Ben Drechsel in einer Diskussion in der phonegap Google Group:

> „[...] Adobe's own 'PhoneGap'-branded docs still direct users to use Cordova CLI and commands that they removed from their own PG CLI... so incredibly unhelpful for someone just trying to learn the landscape. [...] So, if I'm running local builds *only*, and like to use commands like 'prepare', I should install and use Cordova only.“ [vgl. Drechsel 2014]

Üblicherweise findet ein Großteil der Entwicklung jedoch im Umfeld der verwendeten Web-Frameworks - im Fall der GoodToKnow-App jQuery Mobile - statt, bei der auf deren Dokumentation zurückgegriffen wird.

#### Community

Für die Entwicklung ist es auch wichtig, dass verwendete Entwicklungswerkzeuge von einer fähigen und ausreichend großen Gruppe Personen genutzt werden und ein Erfahrungsaustausch dieser auf öffentlichen Plattformen (Foren, Blogs, Entwickler-Konferenzen etc.) stattfindet. Dies ermöglicht es, auf gemeinsame Erfahrungswerte, wertvolle Hinweise und Problemlösungen zurückgreifen zu können und von diesen zu lernen. Dies führt tendenziell in kürzerer Zeit zu höherwertigen Anwendungen und Lösungen.

Die verwendeten Frameworks wurden aufgrund ihrer Relevanz ausgewählt und verfügen somit über entsprechend große Nutzerschaften. Eine für Entwickler interessante, aber nicht direkt numerisch auswertbare Metrik ist die Präsenz der Frameworks auf der verbreiteten Webseite StackOverflow, auf welcher Entwickler zu Themen der Programmierung Fragen stellen und beantworten [vgl. Stackoverflow 2014]. Betrachtet man die Anzahl von Fragen, welche einem bestimmten Framework zugeordnet sind, erhält man keine exakte Information über die Verbreitung, da eine Frage zum Beispiel gleichzeitig mit den Tags „Xamarin“ und „iOS“ markiert sein kann und somit für beide Rubriken zählt. Dennoch können grobe Rückschlüsse über die Größenordnung des Austausches über die betreffenden Frameworks gezogen werden (vgl. Abbildung 3.5).

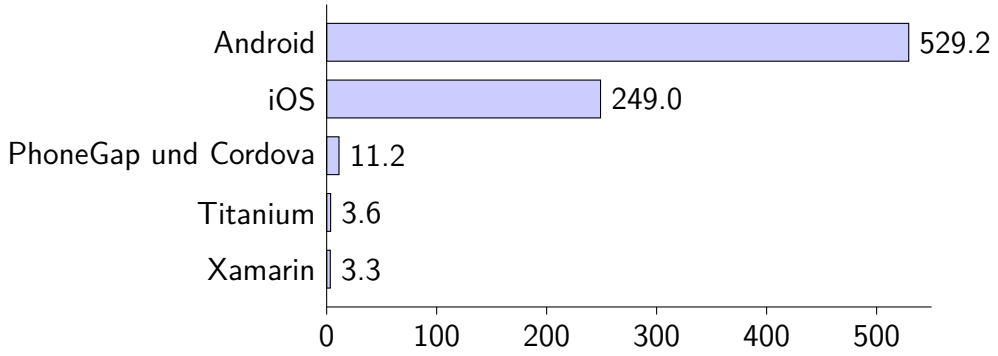


Abbildung 3.5: Anzahl (Tsd.) Fragen auf StackOverflow, die dem angegebenen Framework zugeordnet sind (Stand 27.06.2014)

Aufgrund ihrer dominanten Stellung als Plattforminhaber sind die iOS- und Android-SDKs ein viel diskutiertes Thema. Die Anteile von PhoneGap, Titanium und Xamarin fallen vergleichsweise gering aus - mitunter auch aufgrund der Vielzahl an verschiedenen Lösungen für die Cross-Plattform-Entwicklung.

> „What company has a mobile ecosystem that matches Apple, Google, or HTML5? However, it matters. When developers are 10 times more likely to find results when searching the web about an issue, it directly impacts productivity. The ecosystem of available support, services, and 3rd party components, and related tooling is, and will continue to be, significantly smaller [for Xamarin] than for native or HTML5 based apps.“ [vgl. Whitney 2013a]

Doch nicht nur der Umfang der Community-Beteiligung ist entscheidend. So betreffen Fragen zu iOS und Android inhaltlich meistens eine häufig genutzte Version des SDKs, dessen mitgelieferte Funktionen und im Fall von iOS nur die begrenzte Auswahl an Geräten von Apple. Im Gegensatz können Fragen zu PhoneGap unterschiedliche Betriebssysteme, Geräte und völlig unterschiedliche Web-Frameworks betreffen. Dies erschwert die Suche nach einer Lösung für ein Problem in der eigenen Konfiguration enorm und erhöht auch das Potenzial an seltenen konfigurationsspezifischen Konflikten. Auch Titanium unterliegt aufgrund der großen Anzahl unterstützter Plattformen und der eigenen API in gewissem Maße dieser Problematik. Bei der Entwicklung mit Xamarin kann hingegen zumeist auf die reichhaltigen Erfahrungswerte der iOS- und Android-Community zurückgegriffen werden, da die plattformspezifischen Xamarin-APIs ihren nativen Pendants sehr nahe kommen. Auch im Rahmen des Cross-Plattform-Codes kann auf den Wissensstand der .NET-Community zurückgegriffen werden.

## 3.4 Erkenntnisse


Abbildung 3.6: Icons der erzeugten Apps

Es lässt sich feststellen, dass die Kernfunktionalität der in Kapitel 3.2 entwickelten App-Idee mit allen fünf mobilen Frameworks in einem begrenzten Zeitraum umgesetzt werden konnte und die entstandenen Apps auf den Geräten der Zielplattformen ausführbar und nutzbar sind. Dies wirft ein gutes Licht auf die Zugänglichkeit der mobilen Entwicklungslandschaft, welche es mit Hilfe von umfangreichen und dennoch leicht nutzbaren APIs auch Neueinsteigern ermöglicht, recht fortgeschrittene Funktionalität für eine Vielzahl an Geräten bereitzustellen.

Im Bezug auf die native Entwicklung mit dem iOS- und Android-SDK gab es im Rahmen der Entwicklung keine völlig neuen Erkenntnisse. Beide Frameworks bieten ähnlich umfangreiche APIs, welche sowohl simple als auch komplexe Anforderungen abdecken können. Um ihre Plattform zu stärken und sich vom jeweiligen Konkurrenten abzusetzen, entwickeln Google und Apple ihre SDKs mit Nachdruck weiter. Weiterhin ist die sie umgebende Entwicklerschaft sehr groß und stellt eine große Anzahl von Drittbibliotheken und -lösungen bereit.

Der geforderte Funktionsumfang der GoodToKnow-App konnte mit beiden nativen SDKs ohne größere Hürden abgebildet werden. Der klare Nachteil beider Entwicklungsformen ist Grundlage dieser Arbeit: Die native Entwicklung erfordert Kenntnisse über die Funktionsweise und Eigenheiten der betreffenden Plattform. Darüber hinaus ist in die App investierter Entwicklungsaufwand auf eine mobile Plattform und ihre Endgeräte beschränkt und kann auch nur in geringem Maße auf eine andere übertragen werden. Im Gegenzug entsprechen die erzeugten Apps den gewohnten Bedienparadigmen der Plattformen und profitieren von nativer Funktions- und Leistungsfähigkeit.

Auch der klassische Vertreter der plattformübergreifenden Entwicklung PhoneGap überrascht nicht. Wie erwartet kann auf Basis von Web-Technologien sehr schnell und insbesondere auch ohne umfassendes Wissen über die Zielplattformen eine plattformübergreifende App entwickelt werden, welche zusätzlich je nach Funktionalität auch als Web-App angeboten werden kann. Die verwendeten Technologien sind durch ihre Nutzung in der Web-Entwicklung bereits sehr erprobt und im Repertoire vie-

ler Entwickler enthalten. Die erzeugte Anwendung verhält sich auf allen Plattformen nahezu identisch, was einerseits als Vorteil und andererseits im Bezug auf die plattformspezifischen Bedienparadigmen als Nachteil gesehen werden kann. Inwiefern die erzeugten Benutzeroberflächen und auch die wahrgenommene Performance der Anwendung den Standards der Zielplattformen entsprechen, hängt stark von der Wahl des zugrundeliegenden Web-Frameworks ab. Mit jQuery Mobile fällt der Unterschied zu nativen Apps deutlich aus. Bedienelemente sehen und verhalten sich anders und das Scrollverhalten von Listen unterscheidet sich vom nativen Pendant. Bei der vorliegenden, schlichten Anwendung entspricht die wahrgenommene Performance jedoch der von nativen Apps. Die niedrige Einstiegshürde ermöglicht es, mit wenig Aufwand relativ schnell Ergebnisse auf verschiedenen Plattformen erzielen zu können, welche jedoch dem „lowest-common-denominator“-Prinzip, also dem kleinsten funktionalen und visuellen gemeinsamen Nenner der Zielplattformen, unterliegen.

Interessant sind insbesondere die im Bezug auf Titanium und Xamarin gewonnenen Erkenntnisse. Ihrem grundsätzlichen Ansatz nach zu urteilen wirkten die beiden Frameworks zunächst sehr ähnlich: Eine zentrale Programmiersprache - Javascript bzw. C# - soll die plattformübergreifende Entwicklung ermöglichen und im Gegensatz zu PhoneGap sollen hierbei native Oberflächen entstehen. Im Detail unterschieden sich die Herangehensweisen jedoch erheblich.

Xamarin nimmt mit Ausnahme von Xamarin.Forms eine klare Trennung zwischen der plattformübergreifenden Logik einer Anwendung und ihrer plattformspezifischen Oberflächen und Logik vor. Alle die Benutzeroberfläche betreffenden Aspekte im Kapitel 3.3.1 und 3.3.2 wurden plattformspezifisch mit den entsprechenden APIs und Möglichkeiten der Zielplattform umgesetzt. Nur von der Benutzeroberfläche unabhängige Kernlogik wurde plattformübergreifend angelegt. Mit Titanium ist diese Trennung nicht ganz so strikt. Durch ein eigenes, für alle Plattformen verfügbares Layoutformat und universale Bedienelemente und -klassen kann ein Layout zunächst plattformübergreifend angelegt werden. Innerhalb dieses Layouts müssen dann aber plattformspezifische Eigenheiten berücksichtigt werden und gegebenenfalls je nach Plattform unterschiedliche Bedienelemente deklariert werden. Auch Animationen können grundsätzlich plattformübergreifend programmiert werden.

Auf den ersten Blick erscheint die Erzeugung von Oberflächen mit Xamarin also aufwändiger. Dafür kann das Endergebnis durch die Nutzung der nativen APIs auch wirklich als nativ bezeichnet werden. Titanium strebt hingegen den Spagat zwischen PhoneGaps einheitlichen und Xamarins eigenhändig erzeugten nativen Benutzeroberflächen an, indem mit plattformübergreifendem Code und Deklarationen native

Oberflächen erzeugt werden sollen. Dies spart theoretisch Zeit, führt aber nicht immer auf Anhieb zum Erfolg; häufig müssen unerwartete plattformspezifische Probleme der Titanium-API umgegangen und Anpassungen vorgenommen werden.

Die Entwicklung mit Xamarin erfordert gute Planung in der Frage, welche Bestandteile der Anwendung in plattformübergreifendem Code gekapselt werden können und welche Aspekte zu Gunsten eines „native look and feel“ mit nativen APIs umgesetzt werden. Abhängig vom Anteil der Kernlogik am Gesamtprojekt besteht auch die Gefahr, letztlich doch für jede Plattform den Aufwand einer eigenen App zu betreiben. Gelingt jedoch die Kapselung eines großen Anteils an plattformübergreifender Logik, entsteht mit überschaubarem Mehraufwand pro Plattform eine plattformübergreifende App mit wahrem nativem Erscheinungsbild. Titanium fehlt diese Flexibilität, da kein direkter Zugriff auf die nativen APIs möglich ist und die nativen Bedienelemente in eine alle Plattformen umfassende API zusammengefasst werden. In Folge dessen ist die Entwicklung mit Titanium stark von der Qualität und Verfügbarkeit der Titanium-APIs abhängig.

Ein entscheidender Nachteil von Xamarin gegenüber den anderen kostenfreien Kandidaten ist das kostenpflichtige Lizenzmodell. So reichte die kostenfreie „Starter Edition“ aufgrund von Restriktionen für Anwendungsgröße und API-Zugriff nicht für die in dieser Arbeit entwickelte Anwendung. Die deshalb verwendete 30-tägige Business-Trial-Version lief im Rahmen des Projekts aus, ließ sich aber erneuern. Ein weiterer Nachteil ist, dass mit der Trial-Version erzeugte Apps nur für 24 Stunden auf dem Gerät lauffähig sind.

# 4 Vergleich plattformabhängiger und -unabhängiger Entwicklungsframeworks

Die in Kapitel 1.2 aufgestellte Vermutung, dass der Vergleich plattformabhängiger und -unabhängiger Entwicklung und der entsprechenden Frameworks keinen klaren „Sieger“ hervorbringen würde, wurde dadurch bestätigt, dass in Kapitel 3 für alle Kandidaten spezifische Vor- und Nachteile gefunden werden konnten.

Deshalb soll im folgenden ein Vergleich unter Berücksichtigung der Anforderungen verschiedener Interessengruppen erfolgen. Ergänzend zu den in Kapitel 3 gewonnenen Erkenntnissen werden hier nun auch externe Stimmen einbezogen. Aufgrund der Aktualität des Themas gibt es wenig herkömmliche Literatur zu diesem Thema; und selbst bei dieser bestünde aufgrund der schnellen Entwicklung dieses Themenfelds die Gefahr, nicht den aktuellen Umständen gerecht zu werden. Deshalb werden Meinungen aus Entwickler-Blogs, Webseiten-Artikeln und -Diskussionen beispielhaft zitiert und zusätzlich Fragebögen ausgewertet, welche von zwei Entwicklern der akquinet AG zum Thema mobile Entwicklung beantwortet wurden ([vgl. Anonymus 2014] bzw. [vgl. Behn 2014]).

## 4.1 Vor- und Nachteile

### 4.1.1 für Entwickler

Meist sind es Entwickler, welche allein oder gemeinsam mit einer Projektleitung vor Projektbeginn die technologische Basis anhand abstrakter Anforderungen des Auftraggebers festlegen. Diese beinhalten zumeist Angaben zu den Funktionen, welche die App erfüllen muss, einen einzuhaltenden Kosten- und Zeitrahmen sowie gewünschte Zielplattformen. Anhand dieser wird vorab festgelegt, mit welchen Technologien das Projekt unter den Vorgaben erfolgreich durchgeführt werden kann.

Die einleitende Vermutung, dass es hierfür kein Allzweckwerkzeug gibt, bestätigen auch die in einem umfangreichen Blog-Artikel getätigten Aussagen des auf mobile Apps spezialisierten Entwicklerstudios MercuryDevelopment:

> „Some tools and frameworks promise to get you there through shared cross-platform code, saving you a fortune in writing separate native apps for each ecosystem. Your range of options includes touch-friendly HTML5 websites, hybrid web-apps, or nearly native apps built with cross-platform runtimes like Xamarin or Appcelerator.
>
> At Mercury, we've used them all, and have to admit that no single tool alone can match all needs and purposes." [vgl. Mercury 2013]

Ist nur eine Zielplattform erforderlich, ist die Cross-Platform-Entwicklung zunächst nicht notwendig. Es muss aber dennoch abgewägt werden, ob in naher Zukunft doch eine weitere Plattform hinzukommen und dann von einer vorausschauenden plattformübergreifenden Entwicklung profitiert werden könnte. Darüber hinaus setzt die native Entwicklung für Android oder iOS sehr spezifische Fähigkeiten voraus. Erforderlich sind zum einen umfassende Kenntnis der Besonderheiten der nativen Plattform und andererseits Erfahrung mit der dort benutzten Programmiersprache.

Die für Android verwendete Programmiersprache Java ist im Wirtschafts- und Bildungsbereich sehr verbreitet, sodass vielen Entwicklern hierdurch ein schneller Einstieg möglich ist. Die für die Entwicklung von iOS und Mac OS genutzte Sprache Objective-C hat hingegen, ausgelöst durch den Erfolg des 2008 eröffneten Apple AppStore, erst in den letzten Jahren an Verbreitung gewonnen [vgl. Tiobe 2014]. Hinzu kommt die Tatsache, dass mit Objective-C erworbene Kenntnisse außerhalb des Apple Ökosystems kaum weiterverwendet werden können, sodass Objective-C eine erhebliche Hürde zur nativen Entwicklung für iOS darstellt.

Diese Probleme versucht PhoneGap zu lösen, indem kaum plattformspezifisches Wissen gefordert wird und plattformübergreifend mit den besonders unter Webentwicklern verbreiteten Werkzeugen HTML, CSS und JavaScript gearbeitet werden kann. Vor allem Apps mit simplen Anforderungen können hiermit schnell und unter nahezu voller Wiederverwendung des Codes für mehrere Plattformen umgesetzt werden. Wenn jedoch umfassendere Anforderungen an die App gestellt werden oder ein nativer „Look & Feel" auf den Zielplattformen gefordert ist, steigt die Komplexität der Entwicklung enorm an. So schreibt das Entwicklerteam des IT-Unternehmens Chel Ramsey Productions in ihrer mehrmonatigen Untersuchung des PhoneGap Frameworks: „I was disappointed that it [PhoneGap] seemed to be adding more time to our development to try and make things work that we were used to getting working with relative ease on a native build."

und „Just trying to do integration with calendars, facebook, etc was very time consuming. I couldn't really see how this would 'save us time' considering each platform had a different plugin for these functions that had to be sourced, tested, fixed, wrestled with and may or may not work after all that." [vgl. Chel Ramsey Productions 2013]. Auch MercuryDevelopment stellt fest: „PhoneGap + Sencha Touch hybrids are packed as App Store apps and offer as much as 80–90 percent of code reuse for a second platform. Yet they are limited in terms of hardware integration and have sub-par performance when it comes to complex UI." (Sencha Touch ist ein mit jQuery Mobile vergleichbares Web-Framework).

Die praktischen Erfahrungen mit PhoneGap geben Grund zur Annahme, dass hiermit vor allem schlichte Anwendung zur Datenbeschaffung und -darstellung mit Bedienelementen wie Listen, Buttons und Dropdown-Menüs schnell und plattformübergreifend umgesetzt werden können. Dies bestätigen Chel Ramsey Productions im sechsten Teil ihrer Artikelserie: "For the mobile arena, I feel that Phonegap is the right tool for a smallish, non-complicated mobile app that needs to go multi-platform on a budget. I do not feel that its the right tool for complicated applications – yet. It may get there, but for now I definitely want to keep using it on smaller apps, probably going to look to another solution for larger projects.". Auch ein Entwickler der akquinet SLS logistics GmbH stellt fest „Für einfache Apps mit niedrigem Budget und geringem Funktionsumfang ist die Anwendung [von PhoneGap] durchaus sinnvoll." [vgl. Anonymus 2014].

Wenn an die App jedoch komplexere funktionale Anforderungen gestellt werden, ein nativer Look & Feel erwünscht ist und die App dabei auch noch plattformübergreifend verfügbar sein soll, eignen sich Frameworks wie Titanium und Xamarin. Mit beiden Werkzeugen wird versucht, den Entwicklungsaufwand für mehrere Plattformen zu reduzieren und dabei eine native Benutzeroberfläche sowie native Funktionalität zu bieten. Davon abgesehen unterscheiden sich die Philosophien aus Entwicklersicht jedoch stark:

Titanium ist durch die Verwendung der Programmiersprache JavaScript grundsätzlich recht zugänglich: „JavaScript. It's a language many developers know and enables more developers coming from a web development background to get into mobile app development." [vgl. Angelini 2012]. Neu erlernt werden muss jedoch die für Titanium eigene API, welche Funktionen der nativen Plattformen abstrahiert bereitstellt. Zudem sind im Gegensatz zur PhoneGap-Entwicklung auch Kenntnisse der nativen Plattformen notwendig, um gute Ergebnisse zu erzielen: „Titanium custom APIs also have a learning curve, so the main benefit of 'using your existing web skills' no longer applies. And because your app is now much closer to being native, you need to have some knowledge of the native platforms as well." [vgl. Mercury 2013]. Durch plattformübergreifende

APIs und Bedienelemente ist es grundsätzlich möglich, Code zu einem sehr hohen Grad wiederzuverwenden - und das obwohl native Oberflächen entstehen. Doch die scheinbar perfekte Verbindung aus Plattformunabhängigkeit und nativen Ergebnissen kommt auch nicht ohne Nachteile aus. So können plattformspezifische Probleme auftreten, bei denen eine grundsätzlich plattformübergreifende API unter einer Plattform nicht zum gewünschten Ergebnis führt oder gar nicht funktioniert (vgl. den Punkt Animation in Kapitel 3.3.1). Schwierig wird es insbesondere dann, wenn die Titanium API nicht das leistet, was zur Umsetzung einer bestimmten nativen Funktion nötig ist oder wenn die API nicht so funktioniert wie ihr natives Pendant.

> „At [the] beginning you'll love the well-defined Titanium API and you will probably love it even more every time you discover a simple property to enable behaviors that would require several lines of code on Xcode. But sooner or later you will face strange bugs and limitations. [...] So at first you will save a lot of time but as more complex the project grows you'll lose the saved time in fixing and workarounds." [vgl. Angelini 2012]

Diesem Problem begegnet Xamarin, indem die nativen APIs in vollem Umfang zur Verfügung gestellt werden und so native Funktionalität genau so wie in der nativen Entwicklung umgesetzt werden kann. Dadurch, dass nahezu alle die Oberfläche der App betreffenden Aufwände mit nativen Mitteln erfolgen, liegt der Grad an Wiederverwendung jedoch weitaus niedriger als bei Titanium.

> „Xamarin would argue that trying to abstract UI APIs across very different platforms can create unnecessary complexity or lead to a poor user experience with an LCD (lowest common denominator) design. They have a point here. Titanium tries to do this partially, and the result has made many developers unhappy with the inconsistent or unpredictable results. [...] UI problems can be some of the most time consuming aspects of developing mobile apps. Despite having a good justification, the important takeaway is that for many mobile UI problems, Xamarin will not save developers or designers time." [vgl. Whitney 2013a]

Weiterhin stellt Xamarin unter den fünf Kandidaten die höchsten Anforderungen an die Entwicklerschaft. Diese muss zum einen über umfassendes Wissen der nativen Plattformen, C# und .NET verfügen und andererseits auch in der Lage sein, durch eine gute Softwarearchitektur den Grad an Wiederverwendung von Code zu erhöhen.

> „Your Xamarin developers should have an expert knowledge of each platform that you plan to support and must work closely with each other to design a solid architecture that can achieve the maximum percentage of

> code reuse. [...] But if you master it, you'll get an honest 40-60 percent of code reuse for the second platform in a server-integrated app without any noticeable impact on application quality, performance, and user experience.“ [vgl. Mercury 2013]

Stehen diese Fähigkeiten bereit und bietet das Projekt genug Gelegenheiten, Kernlogik der Anwendung plattformübergreifend bereitzustellen, kann eine App erzeugt werden, welche zeitgleich plattformübergreifend ist und einen vollständig nativen Look & Feel bietet. Diese Behauptung wird zusätzlich auch in Sachen Performance bestärkt. Da bei Xamarin zur Laufzeit nativer Maschinencode ausgeführt wird, steht die Ansprechempfindlichkeit mit Xamarin erzeugter Oberflächen den nativen in kaum etwas nach.

> „The resulting .NET code is then compiled to native binaries for iOS and Android. [...] The binaries you get with Xamarin are slightly larger than pure native ones, and you may notice a slightly increased launch time, but when your app is fully loaded it runs much faster than JavaScript. In fact, performance of Xamarin apps is indistinguishable from native.“
> [vgl. Mercury 2013]

Trotz des nativen Erscheinungsbilds ist dies bei Titanium nicht der Fall, da zum Zeitpunkt der Ausführung einer Titanium-App tatsächlich weniger performanter JavaScript-Code ausgeführt wird:

> „It [the use of JavaScript with Titanium] makes the app's business logic slower than it can be. In complex apps, the users may see this effect as delays in screen transitions, while your JS code is loading or parsing data.“
> [vgl. Mercury 2013]

Dem hohen Aufwand von plattformspezifischen Benutzeroberflächen tritt Xamarin in einer neuen Version mit Xamarin.Forms entgegen. Einfachere Layouts können hiermit plattformübergreifend angelegt werden, während die API die einzuhaltenden Unterschiede der verschiedenen Plattformen berücksichtigt. Peter Bright von ArsTechnica zieht in seinem Review ein vorläufiges Fazit zu Xamarin.Forms: „I think Xamarin.Forms is of huge importance to the Xamarin platform, and its introduction was much-needed. [...] to those targeting two or three platforms, it's going to save a lot of time. Particularly among line of business developers, I can see Xamarin.Forms becoming the primary way of creating interfaces.“ [vgl. Bright 2014].

### 4.1.2 für Auftraggeber

Für den Auftraggeber ist die Wahl des Entwicklungsframeworks aus technischer Sicht grundsätzlich nicht relevant, sofern die gestellten Anforderungen erfüllt werden. Dennoch gibt es auch für ihn einige wichtige Aspekte zu beachten, welche im Zusammenhang mit der Wahl der technologischen Basis stehen.

So ist es wichtig, die Langlebigkeit, also langfristige Nutz- und Wartbarkeit, des Frameworks sicherzustellen. Investiert der Auftraggeber Zeit und Geld in die Entwicklung einer App auf Basis eines bestimmten Frameworks, welches daraufhin nicht mehr aktualisiert wird und somit künftigte Updates erschwert, verliert die Investition an Wert.

Alle untersuchten Frameworks sind im Markt etabliert und werden nicht von einem Tag auf den anderen verschwinden oder ihre Entwicklung einstellen. Sollte dies jedoch eintreten, sind Unterschiede in der Zukunftssicherheit der Kandidaten festzustellen. Beim Wegfall von PhoneGap können große Anteile einer erzeugten Anwendung als Webseite oder unter Verwendung eines anderen Webseiten-Wrappers weiter verwendet werden. Die PhoneGap-Plugins betreffende Funktionalität müsste jedoch ggf. neu entwickelt werden.

Bei Titanium würde der Verlust größer ausfallen, da sowohl die Titanium-API als auch die auf Titanium beschränkten UI-Deklarationen nicht außerhalb des Frameworks wiederverwendet werden können. Einzig Teile des JavaScript Codes, welche nicht auf die Titanium-API zurückgreifen, könnten übernommen werden.

Die nativen UI-Deklarationen eines Xamarin-Projektes könnten auf den jeweiligen nativen Plattformen wiederverwendet werden. Auch der plattformspezifische Code könnte nach Überarbeitung der Syntax gegebenenfalls wiederverwendet werden, da prinzipiell die gleiche API genutzt wird. Der zentrale auf .NET basierende C#-Code könnte aber nur schwer wiederverwendet werden.

> „For example, code written in Xamarin cannot be used in native or HTML5 apps. This means any code developed by a team using Xamarin cannot be shared or reused with teams using any other tooling for iOS and Android. How much this matters depends on the situation, but the problem with development is we can't predict all of our situations. So it's an uncomfortable limitation to have right out of the gate.“ [vgl. Whitney 2013a]

Die in eine native Android- oder iOS-App investierten Aufwände sind nahezu vollständig auf die Verwendung der nativen SDKs begrenzt - die Einstellung dieser Frameworks ist in naher Zukunft jedoch sehr unwahrscheinlich und würde ohnehin auch alle anderen Frameworks betreffen.

Ein weiterer Aspekt sind die mit der Entwicklung und Wartung verbundenen Kosten. So ist es einfacher und preiswerter, einen JavaScript-Entwickler für die Weiterentwicklung einer PhoneGap-Anwendung, als einen Objective-C Entwickler für eine native iOS-App oder gar jemanden mit Kenntnissen über iOS, Android, C# und .NET für eine Xamarin-App zu finden.

> „Compare trying to hire a new Objective-C developer vs hiring a JS developer. Simple economics, there are way more JS developers out there so you're going to get much a [sic] stronger developer for the same price... or the same developer for a much cheaper price." [vgl. Rust-Smith 2014]

In kleineren Projekten können zudem die Kosten der Entwicklungswerkzeuge, welche häufig vom Entwickler auf den Auftraggeber umgeschlagen werden, eine Rolle spielen. Ein häufiger Kritikpunkt an Xamarin sind deshalb die vergleichsweise hohen Lizenzgebühren, welche jährlich und pro Plattform anfallen. Alle anderen Frameworks im Testfeld sind hingegen grundsätzlich kostenlos.

### 4.1.3 für Endkunden

Der Endkunde ist an der Wahl der technologischen Basis nicht beteiligt und sich dieser in den meisten Fällen nicht bewusst. Wichtig ist ihm nur, dass die App für die eigene Plattform verfügbar ist, die angepriesene Funktion erfüllt und leicht zu verwenden ist. Unterbewusst profitiert der Nutzer von mit der Plattform übereinstimmenden Bedienungsparadigmen und nimmt einen performanten („flüssigen") Ablauf der App positiv wahr.

Hierfür eignet sich also die native Entwicklung, die quasi-native Entwicklung mit Xamarin oder mit Abstrichen auch die Entwicklung mit Titanium. Wenn die Benutzeroberfläche einer PhoneGap-Anwendung einen schlüssigen Eindruck macht und diese die gewünschte Funktionalität bietet, ist dies dem Endnutzer auch recht.

## 4.2 Definition von Einsatzszenarien für die verschiedenen mobilen Frameworks

**PhoneGap** eignet sich am besten, wenn eine relativ schlichte Anwendung schnell, kostengünstig und ohne Rücksicht auf plattformspezifische Eigenheiten als App umgesetzt werden soll. Es unterstützt im Testfeld die größte Anzahl Plattformen, ist sehr einstiegsfreundlich und erlaubt einen hohen Grad an Code-Wiederverwendung.

Die Nutzung von **Titanium** bietet sich an, wenn eine App mit überschaubarer Funktionalität und nativem Erscheinungsbild mit einem begrenzten Budget entwickelt werden soll. Hierbei kann ein großer Teil des Codes und der Benutzeroberflächen-Deklarationen für alle Plattformen verwendet werden.

Mit **Xamarin** können umfassende funktionale Ansprüche mit nativem „Look & Feel“ realisiert werden. Dies setzt aufgrund der begrenzten Wiederverwendung von Code und der vergleichsweise hohen technischen Einstiegshürde ein relativ großes Budget voraus. Die Effektivität des Xamarin-Ansatzes ist umso höher, je größer der Anteil an plattformübergreifendem Code am Projekt ist.

Die **native Entwicklung** ist angebracht, wenn mit hoher Sicherheit nur eine Plattform bedient werden muss und/oder hohe funktionale Ansprüche gestellt werden. Besonders wenn neue Funktionen der nativen SDKs kurz nach Veröffentlichung genutzt werden sollen oder die zentrale Funktion einer App nur auf einer Plattform verfügbar ist, ist die native Entwicklung die einzige Option. Naturgemäß kann hierbei meist kein Code auf anderen Plattformen wiederverwendet werden.

Tabelle 4.1 visualisiert, welches Framework sich für eine bestimmte Kombination aus gestelltem Anspruch und verfügbarem Budget am besten eignet.

| *Anspruch* | *geringes Budget* | *mittleres Budget* | *großes Budget* |
|---|---|---|---|
| funktional niedrig, visuell niedrig | PhoneGap | PhoneGap | nicht sinnvoll |
| funktional niedrig, visuell hoch | PhoneGap | Titanium | Xamarin |
| funktional hoch, visuell hoch | nicht sinnvoll | Xamarin | Xamarin |
| neueste native APIs | nicht sinnvoll | nativ | nativ |

Tabelle 4.1: Matrix Framework-Zuordnung für Anspruch und Budget

# 5 Fazit und Ausblick

Es ist erfreulich festzustellen, mit welchen vergleichsweise einfachen Mitteln mobile Apps erzeugt werden können. Leicht zu verwendende und dennoch mächtige APIs, in vielen Geräten verfügbare Sensoren und die zunehmende Verfügbarkeit von mobiler Internetkonnektivität erlauben es Entwicklern, komplexe, vernetzte und durch Sensoren kontextsensitive Anwendungen zu erzeugen. Der in der mobilen Entwicklungslandschaft zu beobachtende schnelle Fortschritt ist hierbei besonders bemerkenswert.

Alle geprüften Kandidaten haben unter unterschiedlichen Bedingungen eine Daseinsberechtigung für die Entwicklung mobiler Apps. Somit ist keines der Werkzeuge eine Ideallösung für alle Anwendungszwecke. Xamarin stellt zum Beispiel die beste Kombination für die plattformübergreifende Entwicklung anspruchsvoller Apps mit nativem Erscheinungsbild für iOS, Android und Windows Phone dar, ist jedoch keine Option, wenn geringer vertretene Plattformen wie Blackberry, Tizen oder Firefox OS plattformübergreifend erreicht werden sollen. Hierfür eignet sich im Testfeld nur PhoneGap. Die native Entwicklung hat bei komplexen Anforderungen weiterhin eine Daseinsberechtigung, ist aber durch die Verfügbarkeit von Lösungen wie Titanium oder Xamarin nicht mehr die einzige Option zur Erzeugung von Apps mit nativem Erscheinungsbild.

Aus Gründen des Umfangs wurde in dieser Arbeit die Unabhängigkeit von Gerätetypen, welche neben der Plattformunabhängigkeit eine wichtige Rolle spielt, nicht betrachtet. Durch ein zunehmend großes und vielfältiges Sortiment an mobilen Endgeräten wird es immer wichtiger, diese auf sinnvolle Weise zu unterstützen. Das bedeutet, die Oberflächen der Anwendung nicht nur entsprechend der Bildschirmgröße zu skalieren, sondern die hinzugekommene Fläche auch sinnvoll zu nutzen und auf größeren Geräten wie Tablets einen Mehrwert zu bieten. Die als „Responsive Design“ bezeichnete Herangehensweise fand seinen Ursprung in der Webentwicklung und wurde bei der Android-Entwicklung aufgrund der großen Gerätevielfalt von Anfang an verfolgt. Aufgrund des begrenzten Gerätesortiments wurde in der iOS-Entwicklung lange Zeit nur wenig Augenmerk auf „Responsive Design“ gerichtet und Oberflächen stattdessen manuell für die Gerätetypen erzeugt. Im Rahmen von iOS 8 führte Apple nun auch eine neue Technologie namens „Adaptive Design“ ein, welche die geräteunabhängige Gestaltung von Benutzeroberflächen ermöglichen soll. Die Möglichkeit, eine App mit

geringem zusätzlichem Aufwand parallel für Smartphones, Tablets und gegebenenfalls auch für den Desktopcomputer zu entwickeln, stellt einen weiteren Aspekt für Entwickler und Auftraggeber dar, durch den sich Entwicklungswerkzeuge voneinander absetzen können.

Bedauerlich ist bei der Untersuchung der Entwicklungswerkzeuge die Beobachtung, dass Drittanbieter fast ausschließlich damit beschäftigt sind, das Problem der plattformunabhängigen Entwicklung zu lösen und dadurch die Erzeugung völlig neuer Funktionalität in den Hintergrund gerät. Diese Ausarbeitung macht deutlich, dass es für Auftraggeber, Entwickler und Endkunden vorteilhaft wäre, wenn ein Plattformen und Hersteller übergreifender Standard für die App-Entwicklung entstehen würde. Vorausgesetzt der Hersteller eines Geräts hält sich an diesen Standard, könnte ein Anwender sich sicher sein, alle Anwendungen verwenden zu können, welche von Entwicklern entsprechend diesem Standard erzeugt werden. Diese Situation existiert bereits für Webseiten und Webanwendungen, welche mit Selbstverständlichkeit auf unterschiedlichen Plattformen aufgerufen werden können. Hierbei bestehen nur vergleichsweise geringe Kompatibilitätsunterschiede zwischen den verwendeten Browsern, welche die eigentliche Runtime für die Webinhalte darstellen.

Viele sehen deshalb die technische Zukunft von Apps in Webtechnologien wie HTML, CSS und Javascript. So ist es sogar das offizielle Ziel des PhoneGap-Projekts, den Betrieb einzustellen, sobald auf Webtechnologien basierende Apps in Sachen Funktionalität und Integration mit nativen Apps gleichziehen.

> „We have two high level goals with PhoneGap:
>
> - The web as a first class development platform.
> - The ultimate purpose of PhoneGap is to cease to exist.“
>
> [vgl. Leroux 2012a]

Dies ist mit erheblichen Geschwindigkeitszunahmen durch moderne mobile CPUs und Fortschritte im Bereich der JavaScript-Engines besser möglich als je zuvor. Essentiell für den Erfolg von webbasierten Apps ist es jedoch, auf APIs, welche bisher nur für native Anwendungen zur Verfügung stehen, zugreifen zu können und darüber hinaus auch wie native Apps im Betriebssystem integriert zu sein: Das heißt Installation über AppStores, Installation und Integration auf dem Homescreen, sowie Zugriff auf alle Funktionen der Plattform. Diesen Ansatz verfolgen bereits Plattformen wie Firefox OS oder Tizen.

Ein universaler Standard wird jedoch nicht nur von technischen Hürden aufgehalten. Die Betreiber der erfolgreichen Plattformen Apple, Google und Microsoft haben ein Interesse daran, einen Standard zu vermeiden und den Erfolg ihrer proprietären Plattformen auszubauen, um ihre Kundschaft an diese zu binden. Dies tun sie, indem sie ihre Plattform mit hohem Tempo mit neuen Funktionen anreichern, welche sie vorübergehend von der Konkurrenz abheben. Durch einen hohen Grad an Kontrolle über das eigene System ist ein solcher Fortschritt schnell möglich, während Gremien, welche die Standardisierung von Funktionen des WWW koordinieren, nur vergleichsweise langsam arbeiten können. So veröffentlicht das W3C (World Wide Web Consortium) erst nach Abwägung vieler Meinungen neue Funktionen als standardisierte Empfehlung, welche dann freiwillig von den Herstellern der Internetbrowser implementiert werden können. So dauert es viel länger, bis eine neue Funktion auf allen Plattformen und in allen Browsern verfügbar ist.

> „The mobile platform leaders such as Microsoft, Apple and Google are continually adding new rich features to their native app platforms to enable developers to create even better apps. For these Silicon Valley giants, the continued dominance of native apps is crucial. Whoever has the best apps available will sell the most devices. [...] If one platform vendor pushes for innovative native features, so must others. Web standards simply can't keep up. The standards bodies just aren't that flexible." [vgl. Lehtimäki 2013]

Erschwerend kommt hinzu, dass die gleichen Parteien, welche zur Zeit nicht an einem gemeinsamen Standard interessiert sind, auch die Browser- und WebView-Implementationen, von denen der Erfolg webbasierter Apps abhängt, bereitstellen und ihren Funktionsumfang kontrollieren. Somit ist nicht davon auszugehen, dass die dominanten Plattforminhaber in naher Zukunft eine gemeinsame technische Grundlage für Apps - ob mit Web- oder anderen Technologien - bilden werden.

# Abbildungsverzeichnis

1.1 Prozentuale Aufteilung der weltweiten Smartphone-Verkäufe an Endnutzer nach Betriebssystem [vgl. Gartner 2013] . . . . . 5

2.1 iOS Architekturbild [vgl. Apple 2014a] . . . . . 12
2.2 Beispiel für Xamarin Code-Struktur [vgl. Xamarin 2014b] . . . . . 20
2.3 Titanium Projekt Architektur [vgl. Appcelerator 2014b] . . . . . 23

3.1 Geo-Wikipedia-App Mockup - Hochformat . . . . . 27
3.2 Geo-Wikipedia-App Mockup - Querformat . . . . . 27
3.3 Hauptansicht der Android-Apps (nativ, Xamarin, Titanium, PhoneGap), deutsche Lokalisierung . . . . . 31
3.4 Hauptansicht der iOS-Apps (nativ, Xamarin, Titanium, PhoneGap), englische Lokalisierung . . . . . 32
3.5 Anzahl (Tsd.) Fragen auf StackOverflow, die dem angegebenen Framework zugeordnet sind (Stand 27.06.2014) . . . . . 45
3.6 Icons der erzeugten Apps . . . . . 46

# Tabellenverzeichnis

2.1 Die theoretischen Eigenschaften der besprochenen Frameworks . . . . . 24

3.1 Unterstützte Plattformen der SDKs (schwarz) und geplante Apps (grün) 25

4.1 Matrix Framework-Zuordnung für Anspruch und Budget . . . . . 56

# Codeverzeichnis


3.1 Deklarative Layout-Erzeugung (Android SDK) . . . . . 30
3.2 Programmatische Layout-Erzeugung (Android SDK) . . . . . 30
3.3 Plattformabhängige Deklaration mit Titanium . . . . . 31
3.4 Lokalisierung in Layout-Deklaration . . . . . 32
3.5 Lokalisierung im Code . . . . . 33
3.6 Benutzeroberfläche animieren . . . . . 34
3.7 Speichern und Abfragen eines Wertes am Beispiel der Browser-API localStorage . . . . . 37
3.8 Öffnen einer URL im externen Browser . . . . . 40
3.9 Sich bis auf sprachen-typische Unterschiede ähnelnde API von Xamarin und nativen SDKs . . . . . 43

# Literaturverzeichnis


[Adobe 2011] Adobe (2011): *Adobe Announces Agreement to Acquire Nitobi, Creator of PhoneGap* URL: http://www.adobe.com/aboutadobe/pressroom/pressreleases/201110/AdobeAcquiresNitobi.html Letzter Aufruf: 30.05.2014

[Angelini 2012] Enrico Angelini (2012): *5 Pros and Cons of Appcelerator's Titanium* URL: http://enricoangelini.com/2012/5-pros-and-cons-of-appcelerators-titanium/ Letzter Aufruf: 05.07.2014

[Anonymus 2014] Anonymus (2014): *Fragebogen zum Thema mobile Entwicklung 1* URL: http://benjaminvehse.de/ba/Fragebogen_Mobile_Dev_BA_1.pdf Letzter Aufruf: 11.07.2014

[Appcelerator 2014] Appcelerator (2014): *Titanium Compatibility Matrix* URL: http://docs.appcelerator.com/titanium/3.0/#!/guide/Titanium_Compatibility_Matrix Letzter Aufruf: 10.06.2014

[Appcelerator 2014a] Appcelerator (2014): *Plans & Pricing* URL: http://www.appcelerator.com/plans-pricing/ Letzter Aufruf: 10.06.2014

[Appcelerator 2014b] Appcelerator (2014): *Titanium Platform Overview* URL: http://docs.appcelerator.com/titanium/3.0/#!/guide/Titanium_Platform_Overview Letzter Aufruf: 10.06.2014

[Appcelerator 2014c] Appcelerator (2014): *Appcelerator Alloy* URL: http://www.appcelerator.com/platform/alloy/ Letzter Aufruf: 17.06.2014

[Appcelerator 2014d] Appcelerator (2014): *Titanium API Documentation* URL: http://docs.appcelerator.com/titanium/latest/#!/api Letzter Aufruf: 17.06.2014

[Appcelerator 2014e] Appcelerator (2014): *Open Mobile Marketplace — Appcelerator* URL: https://marketplace.appcelerator.com/home Letzter Aufruf: 17.06.2014

[Appcelerator 2014f] Appcelerator (2014): *Alloy API Documentation* URL: http://docs.appcelerator.com/titanium/3.0/#!/api/Alloy Letzter Aufruf: 17.06.2014

[Appcelerator 2014g] Appcelerator (2014): *Titanium 3.X - Appcelerator Docs* URL: http://docs.appcelerator.com/titanium/latest/ Letzter Aufruf: 27.06.2014

[Apple 2013] Apple (2013): *Cocoa Core Competencies: Model-View-Controller* URL: https://developer.apple.com/library/ios/documentation/general/conceptual/devpedia-cocoacore/MVC.html Letzter Aufruf: 29.05.2014

[Apple 2014] Apple (2014): *iOS Developer Program* URL: https://developer.apple.com/programs/ios/ Letzter Aufruf: 08.06.2014

[Apple 2014a] Apple (2014): *iOS Technology Overview* URL: https://developer.apple.com/library/ios/documentation/miscellaneous/conceptual/iphoneostechoverview/Introduction/Introduction.html Letzter Aufruf: 22.05.2014

[Apple 2014b] Apple (2014): *Cocoa Touch Layer* URL: https://developer.apple.com/library/ios/documentation/miscellaneous/conceptual/iphoneostechoverview/iPhoneOSTechnologies/iPhoneOSTechnologies.html#//apple_ref/doc/uid/TP40007898-CH3-SW1 Letzter Aufruf: 23.05.2014

[Apple 2014c] Apple (2014): *Preparing Your Nib Files for Localization* URL: https://developer.apple.com/library/ios/documentation/MacOSX/Conceptual/BPInternational/Articles/LocalizingInterfaces.html Letzter Aufruf: 19.06.2014

[Apple 2014d] Apple (2014): *New Features in Xcode 6 Beta* URL: https://developer.apple.com/library/prerelease/ios/documentation/DeveloperTools/Conceptual/WhatsNewXcode/Articles/xcode_6_0.html Letzter Aufruf: 28.06.2014

[Apple 2014e] Apple (2014): *iOS Dev Center* URL: https://developer.apple.com/devcenter/ios/index.action Letzter Aufruf: 27.06.2014

[Begemann 2014] Ole Begemann (2014): *How Dropbox Uses C++ for Cross-Platform iOS and Android Development* URL: http://oleb.net/blog/2014/05/how-dropbox-uses-cplusplus-cross-platform-development/ Letzter Aufruf: 05.07.2014

[Behn 2014] Ditmar Behn (2014): *Fragebogen zum Thema mobile Entwicklung 2* URL: http://benjaminvehse.de/ba/Fragebogen_Mobile_Dev_BA_2.pdf Letzter Aufruf: 11.07.2014

[Bright 2014] Peter Bright (2014): *Xamarin 3 review: Making cross-platform mobile development painless* URL: http://arstechnica.com/information-technology/2014/05/xamarin-3-review-making-cross-platform-mobile-development-painless/ Letzter Aufruf: 06.07.2014

[Burnette 2010] Ed Burnette (2010): *How MonoTouch gets around Apple's VM restrictions* URL: http://www.zdnet.com/blog/burnette/how-monotouch-gets-around-apples-vm-restrictions/1738 Letzter Aufruf: 17.06.2014

[Chel Ramsey Productions 2013] Chel Ramsey Productions (2013): *Phonegap vs Native – Month Seven of a 12 Month Review* URL: http://chelramsey.com/phonegap-vs-native-phonegap-ui-frameworks/ Letzter Aufruf: 03.07.2014

[Cheng 2012] Jacqui Cheng (2012): *"Who needs an app store?" Five years of iPhone* URL: http://arstechnica.com/apple/2012/06/who-needs-an-app-store-five-years-of-iphone/ Letzter Aufruf: 29.05.2014

[Cordova 2014] Cordova (2014): *org.apache.cordova.inappbrowser* URL: https://github.com/apache/cordova-plugin-inappbrowser/blob/master/doc/index.md Letzter Aufruf: 29.06.2014

[Dodson 2014] Ben Dodson (2014): *WikiLocation - The Geolocation Wikipedia API* URL: http://wikilocation.org Letzter Aufruf: 04.06.2014

[Drechsel 2014] Drechsel (2014): *Cordova CLI vs PhoneGap CLI* URL: https://groups.google.com/forum/#!msg/phonegap/emcseOFDNOM/ET9nBH3Gsm0J Letzter Aufruf: 29.06.2014

[Duden 2014] Duden (2014): *Duden — nativ — Rechtschreibung, Bedeutung, Definition, Synonyme, Herkunft* URL: http://www.duden.de/rechtschreibung/nativ Letzter Aufruf: 27.05.2014

[Feloney 2014] Stephen Feloney (2014): *Windows 8 Support, What's Going On?* URL: http://www.appcelerator.com/blog/2014/01/windows-8-support-whats-going-on/ Letzter Aufruf: 10.06.2014

[Firtman 2014] Maximiliano Firtman (2014): *HTML5 compatibility on mobile and tablet browsers with testing on real devices.* URL: http://mobilehtml5.org Letzter Aufruf: 04.05.2014

[Gartner 2013] Gartner (2013): *Gartner Says Smartphone Sales Grew 46.5 Percent in Second Quarter of 2013 and Exceeded Feature Phone Sales for First Time*. URL: http://www.gartner.com/newsroom/id/2573415 Letzter Aufruf: 29.05.2014

[Gonsalves 2007] Antone Gonsalves (2007): *Apple Launches iPhone Web Apps Directory* URL: http://www.informationweek.com/apple-launches-iphone-web-apps-directory/d/d-id/1060220? Letzter Aufruf: 29.05.2014

[Google 2014] Google (2014): *The Android Source Code* URL: http://source.android.com/source/index.html Letzter Aufruf: 29.05.2014

[Google 2014a] Google (2014): *Alternative Distribution Options* URL: http://developer.android.com/distribute/tools/open-distribution.html Letzter Aufruf: 29.05.2014

[Google 2014b] Google (2014): *Android NDK* URL: https://developer.android.com/tools/sdk/ndk/index.html Letzter Aufruf: 29.05.2014

[Google 2014c] Google (2014): *Dashboards: Platform Versions* URL: https://developer.android.com/about/dashboards/index.html Letzter Aufruf: 10.06.2014

[Google 2014d] Google (2014): *Support Library* URL: http://developer.android.com/tools/support-library/index.html Letzter Aufruf: 10.06.2014

[Google 2014e] Google (2014): *Accessing Resources* URL: http://developer.android.com/guide/topics/resources/accessing-resources.html Letzter Aufruf: 19.06.2014

[Google 2014f] Google (2014): *Develop Apps* URL: https://developer.android.com/develop/index.html Letzter Aufruf: 27.06.2014

[Ihlenfeld 2009] Jens Ihlenfeld (2009): *Titanium baut Applikationen für iPhone und Android* URL: http://www.golem.de/0906/67632.html Letzter Aufruf: 10.06.2014

[Intel 2014] Intel (2014): *Mit dem Intel® HAXM Android-Anwendungen schneller testen und debuggen* URL: https://software.intel.com/de-de/android/articles/speeding-up-the-android-emulator-on-intel-architecture Letzter Aufruf: 29.06.2014

[Jetbrains 2014] Jetbrains (2014): *Jetbrains Appcode* URL: http://www.jetbrains.com/objc/ Letzter Aufruf: 29.05.2014

[jQuery 2014] jQuery (2014): *.animate() — jQuery API Documentation* URL: http://api.jquery.com/animate/ Letzter Aufruf: 29.06.2014

[jQuery Mobile 2014] jQuery Mobile (2014): *jQuery Mobile* URL: http://jquerymobile.com Letzter Aufruf: 30.05.2014

[Lehtimäki 2013] Marko Lehtimäki (2013): *The Future of PhoneGap – Will it ever reach native app quality* URL: http://blog.appgyver.com/heartbeat/steroids/the-future-of-phonegap/ Letzter Aufruf: 08.07.2014

[Leroux 2012] Brian Leroux (2012): *PhoneGap, Cordova, and what's in a name?* URL: http://phonegap.com/2012/03/19/phonegap-cordova-and-what\OT1\textquoterights-in-a-name/ Letzter Aufruf: 30.05.2014

[Leroux 2012a] Brian Leroux (2014): *PhoneGap Beliefs, Goals, and Philosophy* URL: http://phonegap.com/2012/05/09/phonegap-beliefs-goals-and-philosophy/ Letzter Aufruf: 07.07.2014

[Mercury 2013] Mercury (2013): *Cross-Platform Frameworks* URL: http://blog.mercdev.com/cross-platform-frameworks/ Letzter Aufruf: 02.07.2014

[Microsoft 2014] Microsoft (2014): *Resources and Localization Using the .NET Framework SDK* URL: http://msdn.microsoft.com/en-us/library/aa309421%28v=vs.71%29.aspx Letzter Aufruf: 19.06.2014

[Nielsen 2013] Nielsen (2013): *Smartphone Switch: Three-Fourths of Recent Acquirers Chose Smartphones* URL: http://www.nielsen.com/us/en/newswire/2013/smartphone-switch--three-fourths-of-recent-acquirers-chose-smart.html Letzter Aufruf: 04.05.2014

[One Platform Foundation 2013] One Platform Foundation (2013): *List of Android Appstores* URL: http://www.onepf.org/appstores/ Letzter Aufruf: 29.05.2014

[Perez 2013] Sarah Perez (2013): *Apple And Google's App Stores Now Neck And Neck – Except On The Metric That Matters Most To Developers: Revenue.* URL: http://techcrunch.com/2013/04/17/

apple-and-googles-app-stores-now-neck-and-neck-except-on-the-metric-that-matters
Letzter Aufruf: 29.05.2014

[Phonegap 2014] Phonegap (2014): *Platform Support* URL: http://docs.phonegap.com/en/3.4.0/guide_support_index.md.html#Platform%20Support Letzter Aufruf: 30.05.2014

[Phonegap 2014a] Phonegap (2014): *Overview* URL: http://docs.phonegap.com/en/3.4.0/guide_overview_index.md.html#Overview Letzter Aufruf: 31.05.2014

[Phonegap 2014b] Phonegap (2014): *Events* URL: http://docs.phonegap.com/en/3.4.0/cordova_events_events.md.html Letzter Aufruf: 30.05.2014

[Phonegap 2014c] Phonegap (2014): *Plugin APIs* URL: http://docs.phonegap.com/en/3.4.0/cordova_plugins_pluginapis.md.html#Plugin%20APIs Letzter Aufruf: 30.05.2014

[Phonegap 2014d] Phonegap (2014): *Adobe PhoneGap Build* URL: https://build.phonegap.com Letzter Aufruf: 30.05.2014

[Phonegap 2014e] Phonegap (2014): *Plugin Development Guide* URL: http://docs.phonegap.com/en/3.4.0/guide_hybrid_plugins_index.md.html Letzter Aufruf: 10.06.2014

[Phonegap 2014f] Phonegap (2014): *Phonegap Documentation: The config.xml File* URL: http://docs.phonegap.com/en/3.4.0/config_ref_index.md.html Letzter Aufruf: 10.06.2014

[PhoneGap 2014g] PhoneGap (2014): *PhoneGap API Documentation* URL: http://docs.phonegap.com/en/3.4.0/index.html Letzter Aufruf: 27.06.2014

[Rust-Smith 2014] David Rust-Smith (2014): *Should You build Phonegap or Native?* URL: http://davidrs.com/wp/should-you-build-phonegap-or-native/ Letzter Aufruf: 06.07.2014

[Sencha 2014] Sencha (2014): *Mobile App Development Framework* URL: http://www.sencha.com/products/touch Letzter Aufruf: 30.05.2014

[Smith 2014] James Smith (2014): *Android Asynchronous Http Client* URL: http://loopj.com/android-async-http/ Letzter Aufruf: 21.06.2014

[sqlite-net 2014] sqlite-net (2014): *sqlite-net* URL: https://github.com/praeclarum/sqlite-net Letzter Aufruf: 03.06.2014

[Stackoverflow 2014] Stackoverflow (2014): *Stackoverflow* URL: http://stackoverflow.com Letzter Aufruf: 01.07.2014

[Statista 2014] Statista (2014): *Most popular Apple App Store categories in March 2014, by share of available apps* URL: http://www.statista.com/statistics/270291/popular-categories-in-the-app-store/ Letzter Aufruf: 03.06.2014

[Thompson 2014] Matt Thompson (2014): *AFNetworking* URL: http://afnetworking.com Letzter Aufruf: 21.06.2014

[Timmer 2014] John Timmer (2014): *A fast look at Swift, Apple's new programming language* URL: http://arstechnica.com/apple/2014/06/a-fast-look-at-swift-apples-new-programming-language/ Letzter Aufruf: 08.06.2014

[Tiobe 2014] Tiobe (2014): *TIOBE Index for June 2014* URL: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html Letzter Aufruf: 03.07.2014

[Triphase 2014] Triphase (2014): *Google Maps for iOS by Triphase* URL: https://marketplace.appcelerator.com/apps/5076?restoreSearch=true#!overview Letzter Aufruf: 21.06.2014

[WHATWG 2013] WHATWG (2013): *Differences from HTML4*. URL: http://html-differences.whatwg.org Letzter Aufruf: 09.07.2014

[NetMarketShare 2014] NetMarketShare (2014): *Market Share Statistics for Internet Technologies*. URL: http://marketshare.hitslink.com Letzter Aufruf: 18.07.2014

[Whitney 2013a] Lee Whitney (2013): *Why I Don't Recommend Xamarin for Mobile Development* URL: http://www.whitneyland.com/2013/05/why-i-dont-recommend-xamarin-for-mobile-development.html Letzter Aufruf: 05.07.2014

[Wikipedia 2014] Wikipedia (2014): *Mobile application development* URL: http://en.wikipedia.org/wiki/Mobile_application_development#Platform_development_environment Letzter Aufruf: 27.05.2014

[Wikipedia 2014a] Wikipedia (2014): *Plattformunabhängigkeit* URL: http://de.wikipedia.org/wiki/Plattformunabh%E4ngigkeit Letzter Aufruf: 27.05.2014

[Wikipedia 2014b] Wikipedia (2014): *Outline of iOS* URL: http://en.wikipedia.org/wiki/Outline_of_iOS Letzter Aufruf: 22.05.2014

[Wikipedia 2014c] Wikipedia (2014): *Jailbreak (iOS)* URL: http://de.wikipedia.org/wiki/Jailbreak_(iOS) Letzter Aufruf: 08.06.2014

[Wikipedia 2014d] Wikipedia (2014): *Android (Betriebssystem)* URL: http://de.wikipedia.org/wiki/Android_(Betriebssystem)#Architektur Letzter Aufruf: 10.06.2014

[Wood 2013] Keith Wood (2013): *jQuery Localisation* URL: http://keith-wood.name/localisation.html Letzter Aufruf: 19.06.2014

[Xamarin 2014] Xamarin (2014): *Store - Xamarin* URL: https://store.xamarin.com Letzter Aufruf: 03.06.2014

[Xamarin 2014a] Xamarin (2014): *Xamarin.Forms - Build native UIs from a single, shared C# codebase.* URL: https://xamarin.com/forms Letzter Aufruf: 30.06.2014

[Xamarin 2014b] Xamarin (2014): *Building Cross Platform Applications Overview* URL: http://developer.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/part_0_-_overview/ Letzter Aufruf: 03.06.2014

[Xamarin 2014c] Xamarin (2014): *Building Cross Platform Applications Overview* URL: http://developer.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/part_1_-_understanding_the_xamarin_mobile_platform/ Letzter Aufruf: 17.06.2014

[Xamarin 2014d] Xamarin (2014): *Xamarin.Mobile increases the amount of code shared across mobile platforms - Xamarin* URL: http://xamarin.com/mobileapi Letzter Aufruf: 03.06.2014

[Xamarin 2014e] Xamarin (2014): *Xamarin Components* URL: https://components.xamarin.com Letzter Aufruf: 03.06.2014

[Xamarin 2014f] Xamarin (2014): *SimpleStorage* URL: http://components.xamarin.com/view/simple-storage Letzter Aufruf: 21.06.2014

[Xamarin 2014g] Xamarin (2014): *Xamarin Developers* URL: http://developer.xamarin.com Letzter Aufruf: 27.06.2014

Ich versichere, die vorliegende Arbeit selbstständig ohne fremde Hilfe verfasst und keine anderen Quellen und Hilfsmittel als die angegebenen benutzt zu haben. Die aus anderen Werken wörtlich entnommenen Stellen oder dem Sinn nach entlehnten Passagen sind durch Quellenangaben eindeutig kenntlich gemacht.

Hamburg, den 19. Juli 2014 Benjamin Vehse