Analyse und Evaluierung von plattformübergreifenden Spiel-Engines und Frameworks, anhand der Implementierung einer mobilen Beispielapplikation

Bachelor-Thesis

zur Erlangung des akademischen Grades B.Sc.

Sebastian Bohn

2036605

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

Erstprüfer: Prof. Dr. Edmund Weitz

Zweitprüfer: Prof. Dr. Andreas Plaß

vorläufige Fassung vom 22. Februar 2016

Inhaltsverzeichnis

Abstract

The market of mobile devices such as smartphones and tablets regularly improves its technological standards and experiences a continuous growth for years. Due to high usability, mobile devices enjoy great popularity among all kind of consumers. Through the broad quantity of users with high-performance mobile computers, digital games are earning greater popularity than ever. The category Games dominates the amount of downloads of the application stores in all leading systems. In order to meet the demands of users with different systems in an economical way, developers require efficient tools for game development. At first, these special frameworks and engines for cross-platform game development are reviewed by their specifications, followed by the implementation of an example application. The theoretical and practical gained knowledge and the conclusive comparison reveal, that the choice of the right development tool still largely depends on the requirements. Nevertheless, Unity3D still clearly dominates its competitors in most disciplines and stands out as an all-rounder in game development.

Zusammenfassung

Der Markt für mobile Geräte wie Smartphones und Tablets erfährt seit Jahren kontinuierliches Wachstum und übertrifft regelmäßig seine technologischen Standards. Durch die hohe Benutzerfreundlichkeit erfreuen sich mobile Geräte großer Beliebtheit innerhalb aller Verbraucherschichten. Aufgrund der breiten Masse an Nutzern von performanten, mobilen Computern gewinnen auch digitale Spiele immer größere Popularität. Die Spielekategorie dominiert die Downloadzahlen der Stores für Applikationen in allen führenden Systemen. Um die Nachfrage für Nutzer unterschiedlicher Systeme auf ökonomische Weise decken zu können, bedarf es entwicklerseitig effizienter Werkzeuge zur Spieleentwicklung. Diese besonderen Frameworks und Engines für die plattformübergreifende Spieleentwicklung werden zunächst aufgrund ihrer Spezifikation begutachtet und daraufhin durch die Implementierung einer Beispielapplikation praktisch angewandt. Durch die gewonnenen theoretischen und praktischen Erkenntnisse und den abschließenden Vergleich kann evaluiert werden, dass die Wahl des passenden Entwicklungstools primär in Abhängigkeit zu der gestellten Anforderung steht. Jedoch kann Unity3D in den meisten Disziplinen seine Konkurrenz klar dominieren und nahezu als Alleskönner der Spieleentwicklung hervorgehoben werden.

1 Einleitung

1.1 Motivation

1.2 Gliederung

Die Thesis *„Analyse und Evaluierung von plattformübergreifenden Spiel-Engines und Frameworks, anhand der Implementierung einer mobilen Beispielapplikation“* ist in zehn Kapitel unterteilt, die das Thema systematisch aufbauen und analysieren.

Im ersten Kapitel wird die Motivation für diese Arbeit begründet sowie die Gliederung für die Vorgehensweise beschrieben.

Das zweite Kapitel analysiert die aktuelle Marktsituation des mobilen Sektors, um die meist genutzten Systeme zu ermitteln. Daraufhin wird durch Statistiken das Verhältnis der Downloads an mobilen Spielen gegenüber anderen Kategorien ermittelt. Weiterhin werden die Spezifikationen der mobilen Betriebssysteme betrachtet und verglichen.

Das dritte Kapitel befasst sich mit der nativen Softwareentwicklung von Android, iOS und Windows Phone. Dabei werden die Voraussetzungen, Notwendigkeiten und Möglichkeiten dargelegt.

Die Aspekte der plattformübergreifenden Entwicklung werden im vierten Kapitel behandelt. Dabei werden die generellen Ziele genannt und die verschiedenen Ansätze zur Umsetzung erläutert.

Im fünften Kapitel werden plattformübergreifende Werkzeuge zur Spieleentwicklung betrachtet. Dafür werden Anforderungen definiert, um Tools zu bestimmen, die vergleichbare Eigenschaften für die Entwicklung mobiler Spiele besitzen. Die ausgewählten Werkzeuge werden daraufhin vorgestellt.

Das sechste Kapitel analysiert die spezifischen Eigenschaften der Spieleframeworks. Es zeigt die erreichbaren Zielplattformen, die nutzbaren Programmiersprachen, Entwicklungsumgebungen sowie die Schnittstellen für native Gerätefunktionen und Game Services auf. Als letztes werden eventuelle Produktvarianten aufgelistet.

Die Konzeption und darauf folgende Implementierung der Beispielapplikation sind Thema des siebten Kapitels. Dafür werden zuerst Anforderungen an ein beispielhaftes, mobiles Spiel definiert. Danach wird die gewählte Spielidee und der Spielfluss erklärt. Weiterhin werden die verwendeten Betriebssysteme, Werkzeuge und Versionen aufgelistet. Danach werden die eingesetzten Spiellemente und ihre Verwendung erklärt.

Im achten Kapitel werden die erzeugten Applikationen analysiert. Dafür werden als Erstes messbare Metriken definiert, um die Anwendungen vergleichbar zu machen. Danach werden die genutzten Testgeräte und die vorgenommenen Grundeinstellungen aufgezeigt. Die Messprotokolle mit den dokumentierten Ergebnissen werden im letzten Teil aufgelistet.

Das neunte Kapitel befasst sich mit dem Vergleich der gewonnenen Erkenntnisse aus den Analysen und der Messresultate. Dafür werden verschiedene Szenarien vorgestellt, welche Ausgangssituationen zeigen und zu Entscheidungen für ein zweckmäßiges Entwicklungstool führen.

Das finale Kapitel gibt ein abschließendes Fazit zu der gesamten Arbeit ab.

2 Mobile Systeme

2.1 Marktanalyse zur Gewichtung der mobilen Systeme und der Applikationen

Welche mobilen Systeme derzeit am Gefragtesten und Verbreitetsten sind, soll in diesem Abschnitt analysiert werden. Dieses Wissen ist nötig, um vor dem Entwicklungsprozess die aktuell erfolgreichsten und zukünftig erfolgversprechendsten Plattformen auszuwählen und miteinzubeziehen. Weiterhin soll geklärt werden, wie viele Applikationen diese Plattformen in ihren Stores bereitstellen und wie die Kategorien gewichtet sind.

2.1.1 Marktanteile der mobilen Betriebssysteme

Eine Statistik über die Marktanteile der mobilen Betriebssysteme bei Smartphones soll veranschaulichen, welche Systeme aktuell zu den Führenden gehören. Zusätzlich wird eine zukünftige Verteilung prognostiziert. Die Darstellungen beziehen sich auf Daten der International Data Corporation (IDC) über den globalen Absatz von Smartphones und wurde im August 2015 veröffentlicht (vgl. Abb. 2.1).
Die Grafik verdeutlicht, dass Geräte mit Android Systemen den Markt eindeutig dominieren, direkt gefolgt von iOS und Windows Phone. Laut Prognose wird sich auch in den nächsten Jahren an dieser Hierarchie nichts ändern. Abbildung 2.2 gibt weiteren Aufschluss über die Verteilung der Systeme nach ausgewählten Ländern. Die Daten beziehen sich auf die Verkäufe von August bis Oktober 2015, welche von Kantar (Kantar 2015) im Dezember 2015 veröffentlicht wurden. Man kann anhand der Daten schlussfolgern, dass Android, iOS und Windows Phone derzeit die drei relevantesten Systeme auf dem mobilen Markt sind.

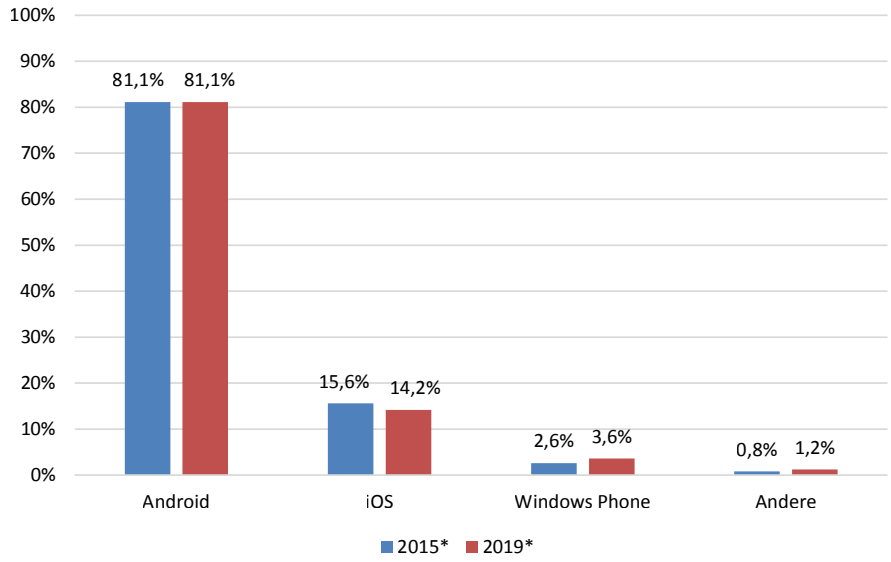

Abbildung 2.1: Prognose zu den Marktanteilen der Betriebssysteme am weltweiten Absatz von Smartphones, in den Jahren 2015 und 2019
(IDC 2015)

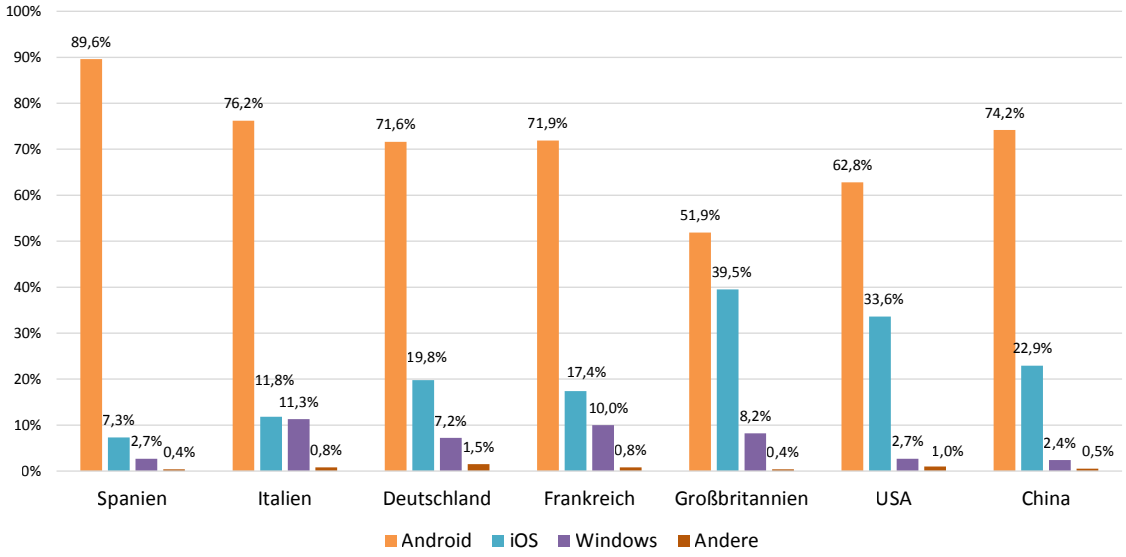

Abbildung 2.2: Marktanteile der mobilen Betriebssysteme am Absatz von Smartphones in ausgewählten Ländern, von August bis Oktober 2015
(Kantar 2015)

2.1.2 Verfügbare Applikationen und Kategorien der Stores

Die Menge an verfügbaren Apps in den jeweiligen Stores ist unterschiedlich groß. Eine Analyse der aktiven Applikationen in den einzelnen Stores und die Gewichtung der Kategorien soll einen Überblick darüber geben, was die jeweiligen Plattformen aktuell zu bieten haben. In Abbildung 2.3 wird die Gesamtquantität der Apps für Mai 2015 dargestellt. Um eine bessere Übersicht zu gewährleisten, wurden die Werte auf Zehntausend gerundet. Zu erwähnen ist, dass der Amazon Appstore, genau wie der Google Play Store, nur Android Apps anbietet. Da es in diesen beiden Stores zu Überschneidungen des Angebots von Anwendungen kommt, werden diese Werte separat betrachtet und nicht summiert.

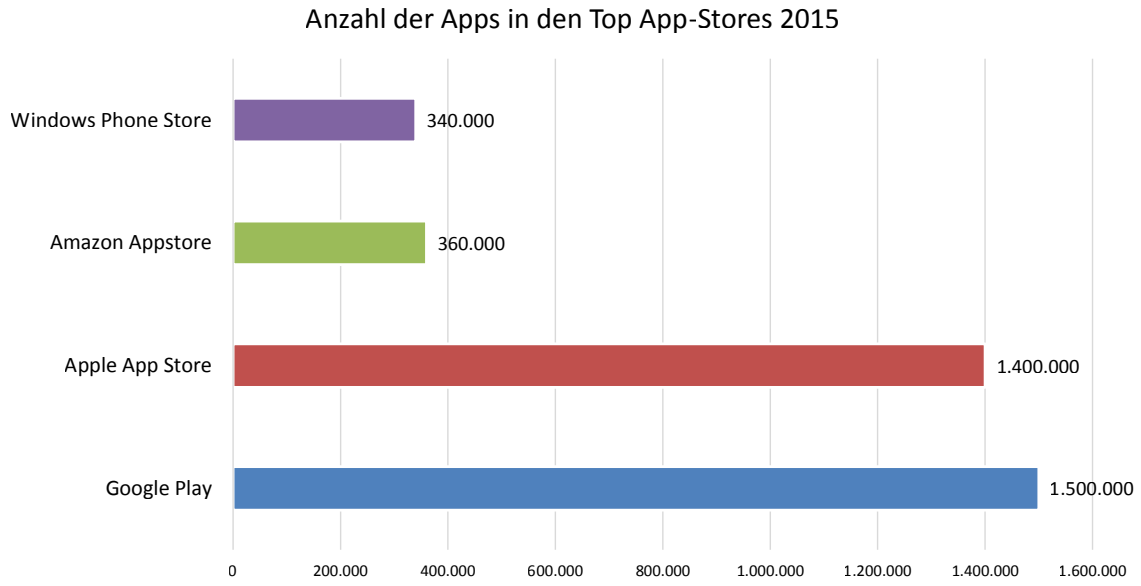

Abbildung 2.3: Anzahl der angebotenen Apps in den Top App-Stores im Mai 2015 (Statista 2015)

Der angegebene Wert für den Windows Phone Store ist laut der Quelle von September 2014 und schließt damit noch nicht die Windows 10 Universal Apps mit ein. Diese erschienen erst Mitte 2015, sollen für jedwede Windows 10 Hardware verfügbar sein und werden in einem separaten Windows Store angeboten. Im September 2015 waren rund 80% der Downloads aus dem Windows Phone Store von Geräten mit der Version 8.1, etwa 15% von 8.0 Benutzern und etwa 5% von der alternden Version 7.8. Laut Microsoft wurden im September 2015 etwa 50% der Applikationen mit Windows 10 aus dem neuen Windows Store heruntergeladen. Diese Statistik gibt jedoch

wenig Aufschluss darüber, wie groß dabei der Anteil an mobilen Systemen ist. Jedoch dominiert die Kategorie Games bei Windows 10 Apps die Downloadzahlen mit fast 45%.(Bernardo Zamora 2015)

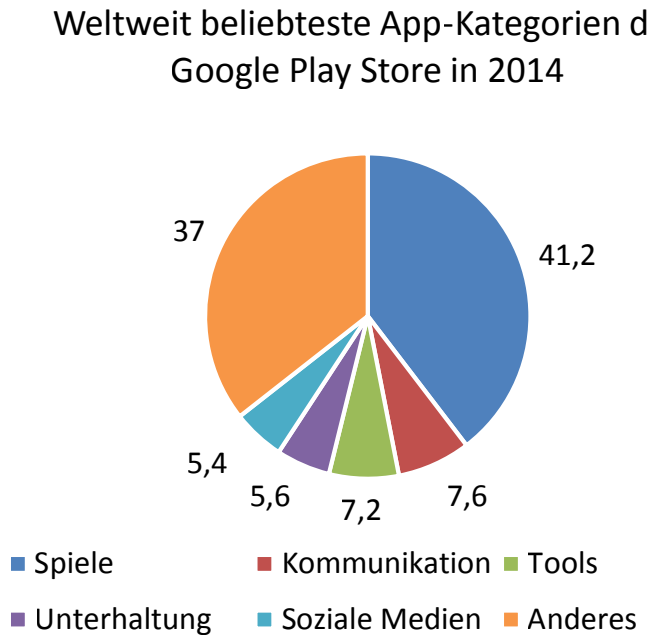

Abbildung 2.4: Aufteilung der weltweit am häufigsten heruntergeladenen Kategorien im Google Play Store, im Februar 2014 (Werte in Prozent) (Distimo 2014)

Auch im Google Play Store werden Spiele am häufigsten heruntergeladen und nehmen etwa 41% des Downloadvolumens ein(vgl. Abb. 2.4).
Die beliebtesten Kategorien des Apple App Store werden ebenfalls deutlich von den Spielen angeführt. Auch wenn der Abstand zur zweithäufigsten Kategorie geringer ist als bei den anderen Stores, macht der Spielebereich trotzdem etwa ein Viertel aller Downloads aus (vgl. Abb. 2.5).

Obwohl jeder digitale Marktplatz seine Applikationen auf eigene Weise kategorisiert, ist dennoch klar zu erkennen, dass Spiele bei jedem Anbieter das höchste Downloadvolumen ausmachen und sich stetig wachsender Beliebtheit erfreuen. Die Nachfrage nach mobilen Spielen ist demnach plattformübergreifend und berechtigt die Evaluierung entsprechender Entwicklungssoftware.

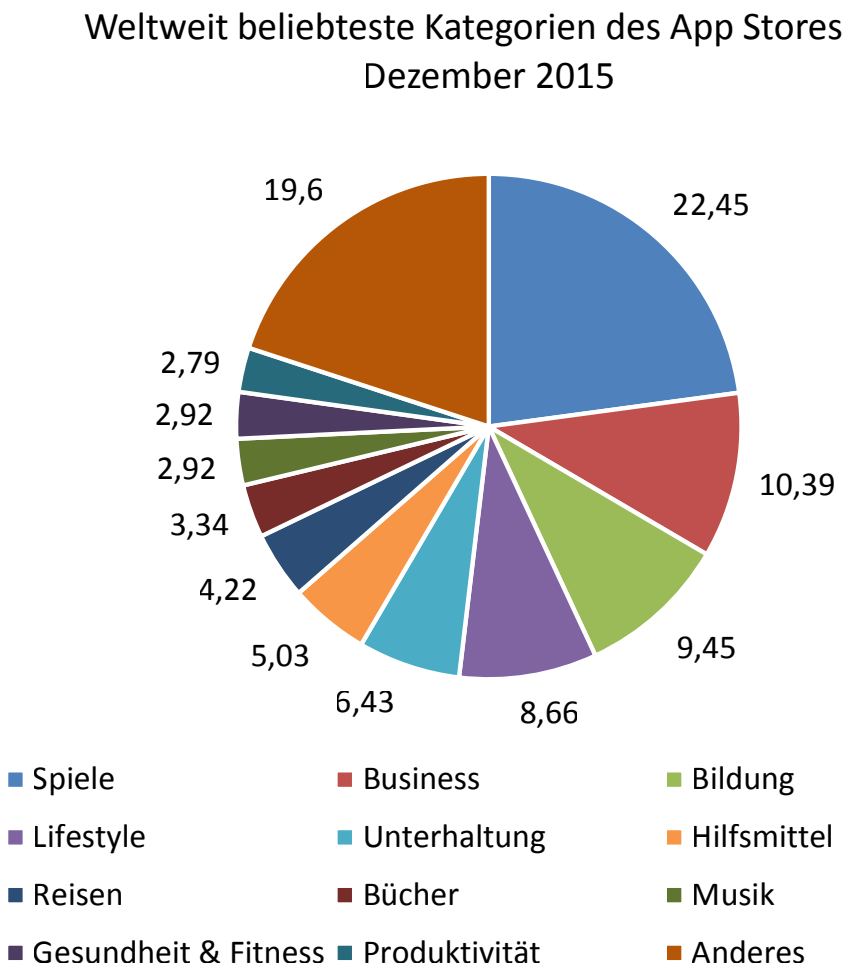

Abbildung 2.5: Downloadzahlen der Top-Kategorien des Apple App Stores, für Dezember 2015 (Werte in Prozent)
(PocketGamer.biz 2015)

2.2 Betrachtung der mobilen Systeme

Die gewonnenen Erkenntnisse aus Kapitel 2.1.1 zeigen, dass derzeit die mobilen Systeme von Android, iOS und Windows Phone die größte Rolle bei den Verbrauchern spielen. Folglich werden im weiteren Verlauf andere Systeme nicht weiter berücksichtigt, so dass der Fokus auf diese drei Systeme gerichtet wird.

2.2.1 Android

Android ist ein Open Source Betriebssystem und gleichzeitig eine Software-Plattform, welche stark im mobilen Bereich vertreten ist und auf dem Linux-Kernel basiert. Zu finden ist Android auf Smartphones, Tablets, Netbooks und auf Smart-TV Geräten. (Open Handset Alliance - Android Overview 2015) Entwickelt wird Android von der Open Handset Alliance (OHA), welche im November 2007 von Google gegründet wurde. Die OHA ist ein Konsortium von mehr als 80 Unternehmen aus den Bereichen Mobilfunknetz, Geräteherstellung, Halbleiterindustrie, Marketing und Software. (Open Handset Alliance - Alliance Members 2015) Der Grund für die Entwicklung von Android war und ist es, einen offenen Standard für mobile Geräte zu schaffen. (Open Handset Alliance - Alliance Overview 2015)
Dadurch gewährt Android den Entwicklern große Freiheit bei der Programmierung von Systemeigenschaften und Applikationen. Schnittstellen zu anderen Anwendungen und den Services von Google, wie zum Beispiel Google Maps, können somit problemlos genutzt und mit der eigenen App verknüpft werden.
Auch der Hardwarebereich bietet ein breites Spektrum an Geräten, deren Angebot von kostengünstig bis hochpreisig reicht und jede Qualitätsstufe bei der technischen Ausstattung abdeckt. Benutzer haben die Möglichkeit, die Benutzeroberfläche und Systemeigenschaften ihrer Geräte weitestgehend frei zu gestalten und zu personalisieren. Für die Beschaffung und Installation von neuen Applikationen sind sie auch nicht an einen offiziellen Store gebunden, sondern können diese aus verschiedensten Quellen beziehen.

Vorteile:

- Open Source
- Unabhängigkeit von Anbietern

- Personalisierung
- Hardwareangebot

Nachteile:

- Hohe Verbreitung von Schadsoftware
- Aktualität der Version ist abhängig vom Gerätehersteller

2.2.2 iOS

iOS ist das mobile Betriebssystem des Unternehmens Apple und ist ein Derivat von Mac OS X, welches wiederum auf Unix basiert. Es wird ausschließlich von Apple entwickelt und ist somit nur auf den eigenen Geräten iPhone, iPad und iPod touch zu finden. Mit der Entwicklung wurde unter externer und interner Geheimhaltung 2005 begonnen und das Resultat der Öffentlichkeit zum ersten Mal Anfang 2007 vorgestellt. Bis zur Version 4.0, wurde iOS mit dem Namen iPhoneOS betitelt. Das Konzept und Design ist schwerpunktmäßig auf hohe Benutzerfreundlichkeit und Funktionalität ausgelegt.
Durch die proprietäre Struktur des Systems sind eigene Derivate nicht möglich. Benutzer sind für den Bezug von Applikationen auf den offiziellen App Store angewiesen. Ebenso bei der Hardwarewahl an die Produktpalette von Apple gebunden, welche jährlich eine neue Generation veröffentlicht. Die Personalisierung der Geräte ist dabei nur bedingt möglich, da Anbieter von Drittsoftware keinen Zugriff auf das System haben. Apple bietet jedoch den Vorteil einer Qualitätssicherung, da Applikationen vor der Veröffentlichung einer recht strengen Prüfung unterzogen werden.

Vorteile:

- Kompatibilität von Software und Hardware
- Benutzerfreundlichkeit
- Geräteübergreifende Kommunikation
- Kontrollen bei Veröffentlichung von Anwendungen

Nachteile:

- Restriktive Firmenpolitik
- Proprietäres System
- Hardwareauswahl
- Anwendungen nur über den App Store

2.2.3 Windows Phone

Microsoft stellt seit dem Jahr 2000 Betriebssysteme für mobile Geräte her (Fran Berkman 2012). Seitdem hat sich die Namensgebung von Windows Mobile, über Windows Phone, bis zum aktuellsten Windows 10 Mobile vorgearbeitet. Um im allgemeinen Bezug nicht zwischen den Namen hin und her zu wechseln, wird in dieser Arbeit, wenn mobile Windows Systeme erwähnt werden, der Name Windows Phone (oder WP) benutzt. Die frühen Versionen von Windows Phone, also Windows Mobile und Windows Phone 7 stammen noch von dem Windows CE Kernel ab, wobei die aktuellen Versionen, Windows Phone 8 und Windows 10 Mobile, Derivate des Windows NT Kernels sind. Mit dem neuesten Ableger, Windows 10 Mobile, verspricht Microsoft eine homogene Kommunikations- und Anwendungsstruktur zwischen allen Geräten die mit diesem System betrieben werden. Dazu zählen nicht nur Smartphones und Tablets, sondern auch Notebooks, Desktop PCs und die Spielkonsole Xbox One (Microsoft 2015).
Microsoft verfolgt mit Windows Phone eine ähnliche proprietäre Struktur wie der Konkurrent Apple. Eigene Derivate des Systems sind also offiziell nicht möglich. Für neue Anwendungen müssen die Benutzer auf das Angebot des Windows Stores zurückgreifen. Jedoch will Microsoft Entwicklern die Möglichkeit bieten, zukünftige Anwendungen universell verfügbar zu machen, so dass diese auf allen Windows Systemen nutzbar sind. Microsoft arbeitet außerdem an einer Technik, die bestehende Projekte von Android und iOS Anwendungen auf die Windows Plattform überführen kann (Golem 2015).
Die aktuellen Windows Phone Versionen sind, durch eine Allianz von Windows und Nokia, hauptsächlich auf mobilen Geräten von Nokia zu finden (Microsoft 2014). Aber auch andere Hersteller bieten Geräte mit Windows Phone an, bisher jedoch in einem überschaubaren Umfang.

Vorteile:

- Kompatibilität von Software und Hardware
- Universelle Anwendungen
- Benutzerfreundlichkeit

Nachteile:

- Proprietäres System
- Anwendungen nur aus dem Windows Store
- Geringeres Angebot an Anwendungen

3 Native Softwareentwicklung

3.1 Systemvoraussetzungen

Entwicklung von Software für ein einzelnes, bestimmtes System wird als nativ (lat.: angeboren, natürlich) bezeichnet. Hier sind Dateiformate, Programmiersprachen, Hardware, Entwicklungsumgebungen und Kompilierung den Anforderungen entsprechend angepasst. Alle individuellen Eigenschaften einer Zielplattform werden unterstützt, ohne dass dabei eine eventuelle Portierbarkeit berücksichtigt wird (John Daintith 2004). Die Entwicklung von Applikationen für eine bestimmte Zielplattform, stellt in der Regel gewisse Mindestvoraussetzungen an das Betriebssystem des Entwicklers.

3.1.1 Android

Android Applikationen sind an kein bestimmtes System gebunden und lassen sich somit unter Windows, OS X und Linux entwickeln. Dies wird unter anderem durch die Eigenschaften der Programmiersprache Java und deren virtueller Maschine ermöglicht. Windows Systeme sollten mindestens Windows XP oder aktuellere Versionen nutzen. Grundsätzlich werden alle 32-Bit Editionen unterstützt und ab Windows 7 auch 64-Bit. Mac Systeme werden ab OS X 10.5.8 von den offiziellen Entwicklungswerkzeugen unterstützt. Entwicklungen auf einem Linux System können beispielsweise unter Ubuntu ab Version 8.04 erfolgen. Bei 64-Bit Versionen ist es notwendig, dass diese fähig sind, 32-Bit Anwendungen auszuführen (Sue Smith 2013). Da die Auswahl an Linux-Distributionen sehr umfangreich und komplex ist, wird in dieser Arbeit auf Linux nicht weiter eingegangen.

3.1.2 iOS

Für den reinen Programmiervorgang von iOS Anwendungen ist prinzipiell jedes System geeignet. Jedoch ist es offiziell nur auf einem Apple OS X System möglich,

die geschriebene Software zu kompilieren und auf ein iOS Gerät aufzuspielen. Für Entwickler, die kein OS X System ihr Eigen nennen oder keinen Zugang zu einem solchen haben, besteht die Möglichkeit, einen Cloudservice zu nutzen. MacinCloud bietet eine cloudbasierte Vermietung von OS X Systemen, inklusive der benötigten Entwicklungssoftware (MacinCloud 2015).

3.1.3 Windows Phone

Ähnlich wie bei iOS wird für die Kompilierung von Windows Phone Applikationen ein Windows Betriebssystem vorausgesetzt. Für die Entwicklung einer Windows Phone 8 App wird mindestens ein Windows 8 Betriebssystem benötigt. Das SDK 8.0 wird aber nur von der 64-bit Version unterstützt. Weiter wird beispielsweise für die Nutzung des Simulators eine Windows 8 Pro Version und die Virtualisierungstechnologie Hyper-V benötigt. Das Windows 10 SDK erwartet minimal Windows 7. Der Simulator benötigt die gleichen Mindestanforderungen wie bei Windows 8 (YoYo Games 2013, Microsoft Developer 2015, MSDN 2015).

3.2 SDKs und Versionen

Software Development Kits, kurz SDKs, liefern dem Entwickler die Softwarebibliotheken, Anwendungen und bestenfalls eine aktuelle Dokumentation, um für eine bestimmte Zielplattform entwickeln zu können. Auch sind sie notwendig, um den geschriebenen Code je nach Art zu interpretieren oder zu kompilieren. Um die aktuellste Version eines mobilen Systems zu unterstützen, muss das SDK auf ebenso aktuellem Stand sein.

3.2.1 Android Versionen

Android Versionen sind nach süßen Leckereien benannt und den Anfangsbuchstaben nach alphabetisch aufsteigend, wie in Tabelle 3.1 zu sehen ist. Das aktuelle SDK kann direkt in Verbindung mit der Entwicklungsumgebung Android Studio oder auch einzeln bezogen werden.

Codename	Version	API Level	Erscheinungsdatum
Marshmallow	6	23	5. Oktober 2015
Lollipop	5.1	22	9. März 2015
Lollipop	5.0.x	21	3. November 2014 - 19. Dezember 2014
Wear	4.4W	20	Juni 2014
KitKat	4.4.x	19	31. Oktober 2013 - 19. Juni 2014
Jelly Bean	4.3.x	18	24. Juli 2013 - 4. Oktober 2013
Jelly Bean	4.2.x	17	13. November 2012 - 12. Februar 2013
Jelly Bean	4.1.x	16	27. Juni 2012 - 10. Oktober 2012
Ice Cream Sandwich	4.0.3 - 4.0.4	15, NDK 8	16. Dezember 2011 - 4. Februar 2012
Ice Cream Sandwich	4.0 - 4.0.2	14, NDK 7	19. Oktober 2011 - 15. Dezember 2011
Honeycomb	3.2	13	16. Juli 2011
Honeycomb	3.1	12, NDK 6	10. Mai 2011
Honeycomb	3	11	23. Februar 2011
Gingerbread	2.3.3 - 2.3.7	10	23. Februar 2011 - 20. September 2011
Gingerbread	2.3 - 2.3.2	9, NDK 5	6. Dezember 2010 - Januar 2011
Froyo	2.2 - 2.2.2	8, NDK 4	20. Mai 2010 - Januar 2011
Eclair	2.1	7, NDK 3	12. Dezember 2010
Eclair	2.0.1	6	3. Dezember 2009
Eclair	2	5	26. Oktober 2009
Donut	1.6	4, NDK 2	15. September 2009
Cupcake	1.5	3, NDK 1	30. April 2009
ohne Codename	1.1	2	10. Februar 2009
ohne Codename	1	1	23. September 2008

Tabelle 3.1: Android Versionen und ihr Erscheinungsdatum
(Android Source - Codenames, Tags, and Build Numbers 2015, Wikipedia - Liste von Android-Versionen 2015)

3.2.2 iOS Versionen

Apple nutzt für seine Produkte Codenamen, die keinem bestimmten Muster folgen. Verbrauchern sind diese meist unbekannt, da die Namen überwiegend intern genutzt werden. In Tabelle 3.2 sind die derzeitigen Versionen aufgelistet. Das SDK wird offiziell ausschließlich in Verbindung mit XCode bezogen.

Codename	Version	Erscheinungsdatum
Monarch	9.2 Beta	3. November 2015
Monarch	9.1	21. Oktober 2015
Monarch	9.0.x	16. September 2015
Copper	8.4.x	30. Juni 2015
Stowe	8.3	8. April 2015
OkemoZurs	8.2	9. März 2015
OkemoTaos	8.1.x	9. Dezember 2015
Okemo	8.0.x	17. September 2014
Sochi	7.1.x	10. März 2014
Innsbruck	7.0.x	18. September 2013
Brighton	6.1.x	21. Februar 2013
Sundance	6.0.x	19. September 2012
Hoodoo	5.1.x	7. März 2012
Telluride	5.0.x	12. Oktober 2011
Durango	4.3.x	9. März 2011
Jasper	4.2.x	22. November 2010
Baker	4.1	8. September 2010
Apex	4.0.x	21. Juni 2010
Wildcat	3.2.x	3. April 2010
Northstar	3.1.x	9. September 2009
Kirkwood	3.0.x	17. Juni 2009
Timberline	2.2.x	21. November 2008
Sugarbowl	2.1.x	9. September 2008
Big Bear	2.x	11. Juli 2008
Little Bear	1.1.x	14. September 2007
Alpine	1.0.x	29. Juni 2007

Tabelle 3.2: iOS Versionen und ihr Erscheinungsdatum (the iphone wiki 2015)

3.2.3 Windows Phone Versionen

Tabelle 3.3 berücksichtigt alle Versionen ab Windows Phone 7. Die Unterstützung seitens Microsoft wurde bereits eingestellt. Version 8 soll laut Angabe bis etwa 2017 weitergeführt werden, bis die Portierungen der Nutzer zu Windows 10 abgeschlossen sind.

Codename	Version	Erscheinungsdatum
Windows Phone 7	7.0.7004.0	21. Oktober 2010
PreNoDo	7.0.7008.0	21. Februar 2011
NoDo	7.0.7390.0	22. März 2011
	7.0.7392.0	3. Mai 2011
	7.0.7403.0	September 2011
7.5 / Mango	7.10.7720.68	27. September 2011
	7.10.7740.16	17. November 2011
	7.10.8107.79	4. Januar 2012
	7.10.8112.7	Juni 2012
7.5 Refresh / Tango	7.10.8773.98	27. Juni 2012
	7.10.8779.8	15. August 2012
	7.10.8783.12	30. Januar 2013
7.8	7.10.8858.136	30. Januar 2013
	7.10.8860.142	14. März 2013
	7.10.8862.144	15. März 2013
8.0 / Apollo	8.0.9903.10	29.Oktober 2012
Portico	8.0.10211.204	29. Januar 2013
Apollo+	8.0.10327.77	19. Juli 2013
	8.0.10512.142	14. Oktober 2013
Blue	8.10.12397.895	16. Juli 2014
	8.10.14234.375	5. Dezember 2014
	8.10.15148.160	11. April 2015
Windows 10	10.0.10586.0	20. November 2015
	10.0.10586.29	8. Dezember 2015

Tabelle 3.3: Windows Phone Versionen und ihr Erscheinungsdatum (Wikipedia - Windows Phone 7 2015, Wikipedia - Windows Phone 8 2015, Wikipedia - Windows 10 2015)

3.3 Programmiersprachen

In der nativen Entwicklung werden für jede Zielplattform bestimmte Programmiersprachen unterstützt.

3.3.1 Android

Android Applikationen werden grundsätzlich in Java entwickelt. Demnach ist es notwendig, zusätzlich eine aktuelle Java Version (JDK) zu installieren. Diese wird von dem Unternehmen Oracle in der aktuellen Version 8 vertrieben (Oracle - Java SE 2015). In Kapitel 6.2.1 wird noch etwas detaillierter auf Java eingegangen.

3.3.2 iOS

Die primäre Programmiersprache für iOS und OS X ist derzeit die objektorientierte Sprache Objective-C, die eine Erweiterung von C ist und Einflüsse von Smalltalk aufweist. Von Smalltalk werden beispielsweise die Syntax deren Objekteigenschaften abgeleitet. Trotzdem besteht bei nicht-objektorientierten Operationen weiterhin die Nähe zur C Syntax. Objective-C befindet sich aktuell in der Version 2.0 (Mac Developer Library 2014).
Alternativ kann die relativ junge Sprache Swift verwendet werden. Diese ist ebenfalls objektorientiert und in der aktuellen Version 2.0. Im Dezember 2015 wurde der Quellcode Open Source zur Verfügung gestellt. Swift soll Objective-C allerdings nicht ersetzen, sondern eine zusätzliche Möglichkeit darstellen (Apple Developer 2015). Die beiden Sprachen sind miteinander kompatibel, so dass es möglich ist, beide innerhalb eines Projekts zu nutzen.

3.3.3 Windows Phone

Für die Programmierung der Logik hat ein Entwickler die Freiheit C#, Visual Basic, JavaScript oder C++ zu nutzen. Um die Benutzeroberflächen zu gestalten, wird hauptsächlich die Sprache XAML (Extensible Application Markup Language) eingesetzt.

3.4 Entwicklungsumgebungen

Für die Entwicklung werden seitens der Betreiber jeweils verschiedene IDEs (Integrated Development Environment) unterstützt und empfohlen. Eine Besonderheit bei IDEs für mobile Systeme ist die Unterstützung eines Simulators. Dieser virtualisiert ein spezifiziertes Gerät, um die Anwendung direkt testen zu können.

3.4.1 Android

Android empfiehlt die eigene, offizielle IDE Android Studio. Diese ist kostenfrei erhältlich und basiert auf der IDE IntelliJ IDEA von JetBrains (Android Develop Tools 2015). Als Alternative gilt der Open Source Vorgänger Eclipse.

3.4.2 iOS

Xcode ist die IDE von Apple, ohne die eine Kompilierung von iOS Projekten nicht möglich ist. Sie befindet sich derzeit in der stabilen Version 7 und liefert das benötigte SDK sowie einen umfangreichen Simulator der alle mobilen Apple Geräte virtualisieren kann. Xcode ist ebenfalls verpflichtend, wenn die kompilierte Anwendung auf ein Mobilgerät installiert und getestet werden soll. Seit dem 3. Quartal 2015 ist dafür keine kostenpflichtige Entwicklerlizenz mehr nötig, denn diese wird nur noch bei einer Veröffentlichung im App Store verlangt (t3n 2015). Eine Alternative bietet JetBrains IDE Appcode, welches eine kompatible, aber kostenpflichtige Erweiterung zu Xcode ist (JetBrains 2015).

3.4.3 Windows Phone

Microsoft setzt für die Kompilierung und Realisierung für Windows Phone Projekte eine Visual Studio IDE voraus, welche für unkommerzielle Entwicklungen kostenfrei ist.

3.5 Native Spieleentwicklung

Bei der nativen Entwicklung basieren die mobilen Applikationen auf den jeweiligen SDKs. Auch mobile Spiele können auf nativem Weg realisiert werden. Die Verwendung der mitgelieferten Grafikfunktionen für die Visualisierung und selbst definierte Spielweltlogik, kann schon eine Basis für einfache 2D Spiele verkörpern. Durch die freie Grafikbibliothek OpenGL können Spielinhalte in 2D oder sogar 3D dargestellt werden. Allerdings ist OpenGL nicht für Windows Phone verfügbar, denn hier wird das eigene Pendant, DirectX, verwendet. Um die Spieleentwicklung zu erleichtern, können spezialisierte Frameworks und Engines genutzt werden, die eigens für die native Entwicklung von Spielen konzipiert wurden und auf die gewünschte Plattform

abgestimmt sind. In den folgenden Unterkapiteln werden beispielhafte Werkzeuge für die jeweiligen Plattformen aufgelistet.

3.5.1 Android

Für Android existieren einige Frameworks, wovon jedoch viele auf einem veralteten Stand sind. Das ist häufig damit verbunden, dass diese aus privatem Interesse entstanden sind und ein unkommerzielles, kostenloses Vertriebsmodell betreiben. Daher sind regelmäßige Aktualisierungen oft nicht gewährleistet und manche Arbeiten wurden auch ganz eingestellt.

AndEngine
Eine Game Engine, die auf OpenGL basiert und für 2D Spiele ausgelegt ist. Die letzte Aktualisierung erfolgte im Dezember 2013.

Cocos2D-Android
Eine Implementierung von Cocos2D für Android. Die Weiterentwicklung wurde allerdings eingestellt und die letzte Änderung erfolgte im Januar 2010.

jPCT-AE
Eine 3D Engine, die auf jPCT für Desktopanwendungen mit Java basiert und für Android portiert wurde. Die Grafik wird von OpenGL 1.x und 2.0 unterstützt. Das Projekt wird aktuell weiterentwickelt und ist frei verfügbar.

3.5.2 iOS

Auch für iOS sind diverse Frameworks erhältlich. Die Suche nach einer aktuellen Entwicklungsunterstützung gestaltet sich hier ein wenig einfacher, wobei man aber, genau wie bei Android, auch auf aufgegebene Projekte stößt. Ein großer Vorteil besteht aber durch SpriteKit, da dies von Apple selbst entwickelt wurde und eine gewisse Sicherheit bei zukünftiger Unterstützung gewährleistet.

SpriteKit
Eine Eigenentwicklung von Apple für 2D Spiele auf iOS und OS X. Es kann Objective-C oder Swift genutzt werden.

Cocos2D-iPhone
Ein Abkömmling von Cocos2D für iOS mit den Implementierungen für Objective-C

und zukünftig auch Swift. Dies ist einer der offiziellen Hauptzweige von Cocos2D, der stetig aktualisiert wird.

Sparrow
Eine aktuelle Open Source Game Engine für iOS Spiele, die ausschließlich Objective-C unterstützt.

3.5.3 Windows Phone

Frameworks, die einzig und allein für Windows Phone Spiele ausgelegt sind, waren nicht direkt ausfindig zu machen. Microsoft selbst rät zu dem eigenen XNA Framework, womit auch die meisten Entertainment Plattformen von Microsoft bedient werden können. Hauptsächlich werden hier Cross Plattform Tools genutzt, da die Verbreitung von entsprechender Hardware noch relativ gering ausfällt. Auch durch das neue Windows 10 und den Wunsch nach universellen Anwendungslösungen, besteht derzeit ein noch nicht abgeschlossener Änderungsprozess. Dadurch werden spezialisierte Lösungen für eine Windows Gerätekategorie zukünftig vermutlich eher uninteressant.

3.6 Zusammengefasste Übersicht

Die Tabelle 3.4 soll eine kompaktere Übersicht der einzelnen Systeme bieten.

Eigenschaft	Android	iOS	Windows Phone
Virtuelle Maschine	Dalvik VM	-	CLR
Programmiersprache	Java	Objective-C, Swift	C#, C++, Visual Basic, JavaScript, .NET
User Interface	XML	Cocoa Touch	XAML
Speicher Management	Garbage collector	Reference counting	Garbage collector
IDE	Eclipse, Android Studio	Xcode	Visual Studio
Entwicklungsplattform	Multi-Plattform	Mac OS X	Windows
Geräte	Heterogen	Homogen	Homogen
App Markt	Google Play Store	Apple App Store	Windows Phone Store

Tabelle 3.4: Unterschiede zwischen Android, iOS und Windows Phone

4 Plattformübergreifende Entwicklung

Die Entwicklung von Softwareprodukten und Services, welche auf mehreren Systemen oder Laufzeitumgebungen funktioniert, wird als plattformübergreifende oder auch Cross-Plattform Entwicklung definiert. Um dies zu gewährleisten, nutzen Entwickler unterschiedliche Methoden und Techniken, um verschiedene Systeme mit einer Projektstruktur zu erreichen (techopedia 2015).

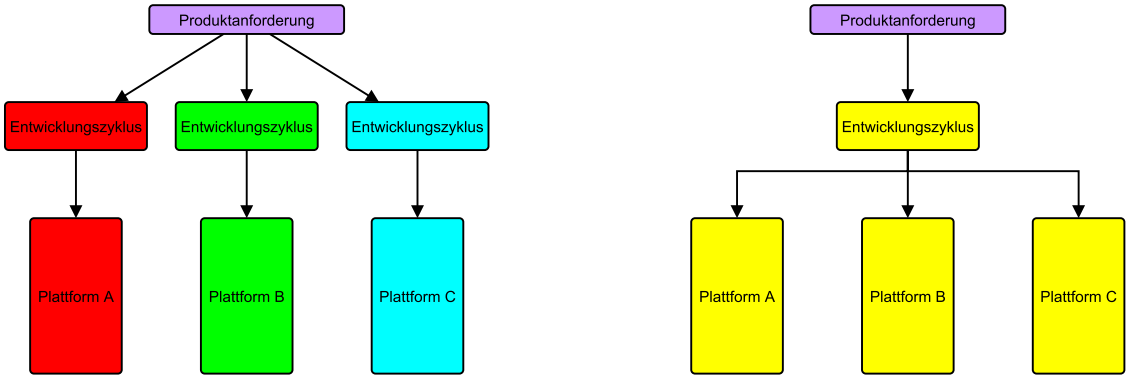

Abbildung 4.1: Traditionelle und plattformübergreifende Entwicklungsmodelle

4.1 Ziel

Die Idee und das Ziel von plattformübergreifender Entwicklung sind, dass eine Softwareanwendung auf mehr als einer spezifischen digitalen Umgebung zufriedenstellend funktioniert. Diese Vorgehensweise wird angewandt, um ein Softwareprodukt auf mehreren proprietären Betriebssystemen zu vertreiben. Dies soll die Entwicklungszeit und sich daraus ergebene Kosten einsparen. Durch die Entwicklung von

mobilen Geräten sowie die zunehmende Verbreitung von Open Source Technologien entstanden sukzessiv unterschiedliche Ansätze zur Realisierung.
Die Nutzung dieser Arbeitsweisen hat aber nicht nur Vorteile. Als nachteilig gilt die potentiell geringere Effizienz der Anwendung gegenüber der nativen Entwicklung. Beispielsweise enthält das Programm redundante Prozesse oder für jede Plattform einen eigenen Datenspeicherordner. Die Reduzierung von Komplexitäten kann auch bis zur „Verdummung“ des Programms ausarten, um das Programm für weniger anspruchsvolle Softwareumgebungen anzugleichen.
Trotz mancher momentanen Grenzen bietet die plattformübergreifende Entwicklung ausreichende Möglichkeiten, die eine derartige Projektstruktur befürworten (techopedia 2015).

4.2 Funktionsweise und Realisierungsansätze

Zu den grundlegenden Strategien gehört, dass ein Projekt oder Programm auf einen allgemein verständlichen Zwischencode (z.B. Bytecode) reduziert wird, um daraufhin zu verschiedenen Zielbetriebssystemen kompiliert zu werden. Weitere Methoden beinhalten die Verwendung von Teilbäumen in der Projektstruktur, um die Anwendung bestmöglich an die Eigenheiten der entsprechenden Zielplattform anzupassen. Ein anderer Ansatz ist die Abstraktion des Codes auf unterschiedlichen Ebenen, um sich mehreren Softwareumgebungen anzunähern. Softwareprojekte, die solche Verfahren anwenden, kann man als plattformunabhängig, genauer gesagt plattformübergreifend, bezeichnen, da sie die unterstützten Systeme gleich werten und keines bevorzugen (techopedia 2015).

Die Entwicklung von plattformübergreifenden Anwendungen auf mobilen Systemen wird in sechs verschiedene Ansätze kategorisiert. Diese Ansätze werden zum Teil in Unteransätze aufgeteilt, wie in Abbildung 4.2 zu sehen ist.
In den folgenden Unterkapiteln werden die einzelnen Ansätze und Unteransätze näher betrachtet. Die Analyse und Betrachtung der folgenden Abschnitte basieren auf den Informationen der Ausaurbeitung von *„Taxonomy of Cross-Platform Mobile Applications Development Approaches“* (El-Kassas, Wafaa S. & Abdullah, Bassem A. & Yousef, Ahmed H. & Wahba, Ayman M. 2015).

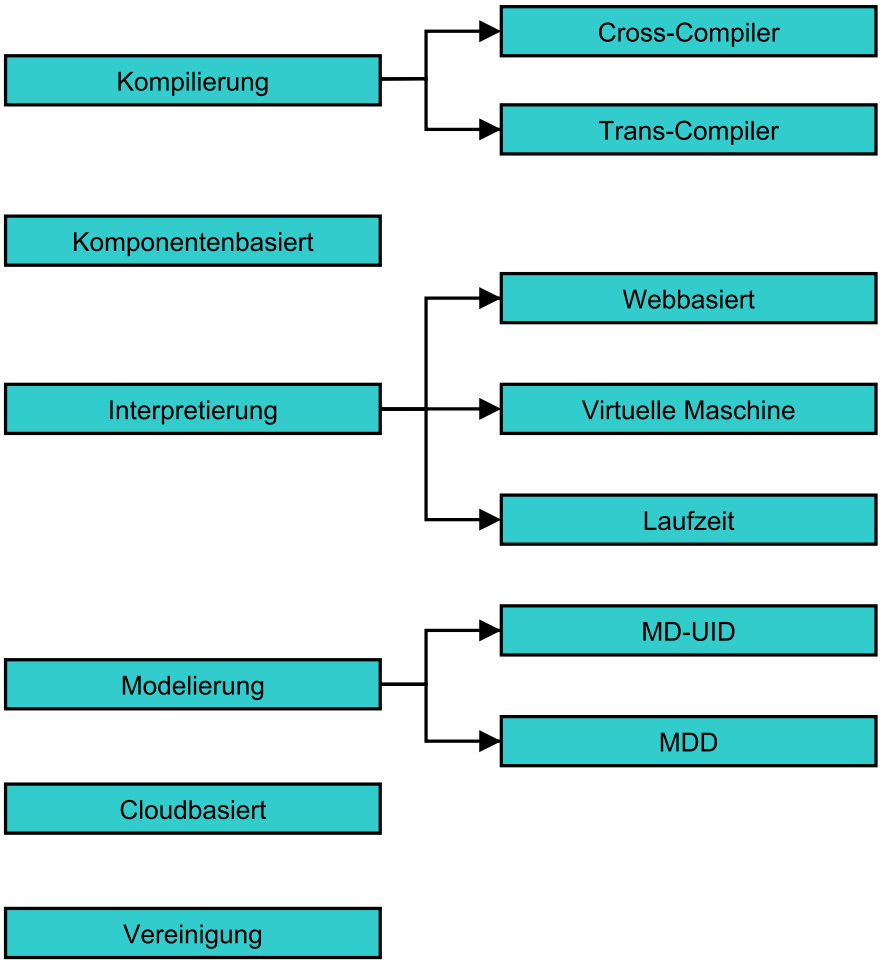

Abbildung 4.2: Haupt- und Unteransätze zur Entwicklung von mobilen, plattformübergreifenden Anwendungen

4.2.1 Kompilierung

Der Kompilierungsansatz wird in zwei Unterkategorien aufgeteilt:

- Cross-Compiler
- Trans-Compiler

Der Compiler ist ein Programm, welches den Quellcode einer High-Level Programmiersprache in einen Low-Level Code übersetzt. Dieser Low-Level Code ist ein Binärcode in Maschinensprache, der von Prozessoren verstanden wird. Dieser Konvertierungsprozess wird als Kompilierung bezeichnet.

Man spricht von einem **Cross-Compiler**, wenn das System auf dem der Compiler sich befindet unterschiedlich zu dem System ist, auf dem der kompilierte Code ausgeführt werden soll. Die Zielsysteme können Betriebssysteme, Prozessoren oder eine Kombination aus beiden sein. Abbildung 4.3 stellt eine von XMLVM gebotene Lösung eines Cross-Compilers dar. Dieser Compiler nutzt XML für die Darstellung des Frontends und eine virtuelle Maschine (VM) für die Verarbeitung des Bytecodes.

Ein **Trans-Compiler** kompiliert eine High-Level Programmiersprache in eine andere High-Level Programmiersprache. Da viele Sprachen jedoch unterschiedliche Eigenschaften und Leistungsmerkmale besitzen, muss der generierte Code unter Umständen nachbearbeitet werden, wenn der Compiler die Quelleigenschaften nicht für die Zielsprache übersetzen kann. Zudem ist der Code durch die automatisierte Erzeugung in der Regel nur schwer von Menschen lesbar. Es besteht außerdem eine Abhängigkeit zu regelmäßigen Updates, um die Änderungen der Quell- und Zielsysteme aktuell zu halten und aufeinander abzustimmen.

4.2.2 Komponentenbasiert

Die Komponente besteht aus einem Paket oder einem Modul, dessen Funktionen und Daten untereinander in Relation stehen. Jede Komponente besitzt eine Schnittstelle, welche die Servicedienste spezifiziert, die von anderen Komponenten genutzt werden können. Die Kommunikation findet ausschließlich über die Schnittstellen statt, so dass eine Komponente keinerlei Informationen über den internen Aufbau einer anderen benötigt.

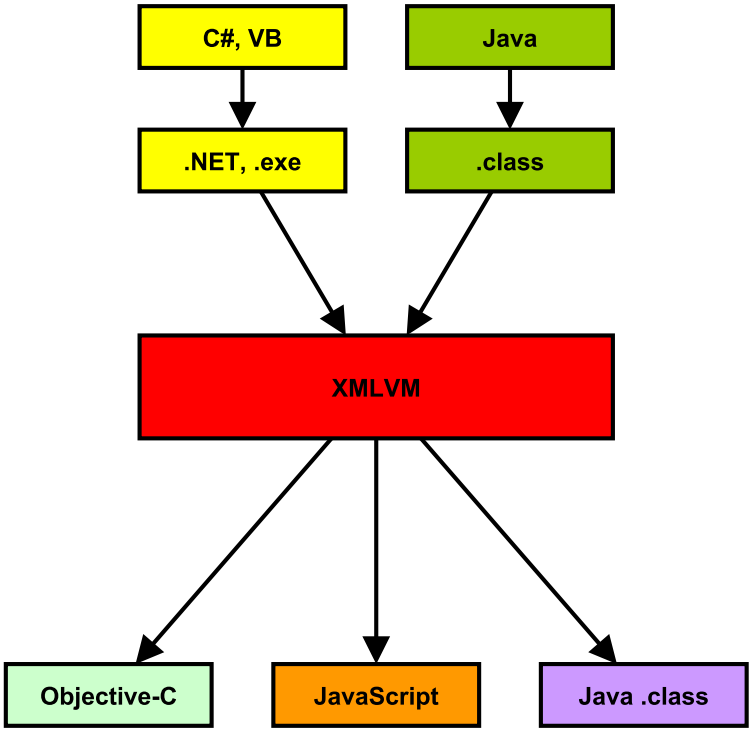

Abbildung 4.3: XMLVM Prozess mit Java oder .NET Quellcode (XMLVM 2011)

Ein theoretischer, komponentenbasierter Lösungsansatz der nicht offiziell betitelt ist, versucht die Entwicklung mobiler Web-Apps dahingehend zu vereinfachen, dass durch das Konzept von Softwarekomponenten, die Kernfunktionalitäten modular aufgeteilt werden. Diese Module beinhalten Speichermanagement, Netzwerkkommunikation, Grafik, Dateisystem und die Systemdienstkomponenten. Dadurch erhalten die Komponenten eine Wiederverwertbarkeit und vereinfachen die Migration auf andere Plattformen. Jede Plattform kann dieselben Schnittstellen nutzen, benötigt jedoch eine eigene innere Implementierung für die Unterstützung.

4.2.3 Interpretierung

Bei der Interpretierung übersetzt ein Interpreter (Dolmetscher) den Quellcode, meist in Form von Skriptsprachen, in ausführbare Anweisungen. Dies geschieht in Echtzeit mit Hilfe einer dedizierten Maschine. Hierbei existieren drei Unteransätze:

- Virtuelle Maschine (VM)
- Webbasiert (Web-based)
- Laufzeit Interpretation (Runtime Interpretation)

Die bekannteste **virtuelle Maschine** ist die Java Virtual Machine (JVM). Diese verfügt über eine eigene, komplette Hardwarearchitektur mit CPU, Stack, Register und einem korrespondierenden Befehlssystem. Die Grundidee hierbei ist es, die mobile App mit einer plattformübergreifenden Sprache zu entwickeln, die auf der dedizierten, virtuellen Maschine läuft und auf entsprechenden Plattformen installiert ist. In Abbildung 4.4 wird der Interpretierungsablauf der von Android bekannten Dalvik VM dargestellt.

Webbasierte Tools verwenden Technologien wie HTML(5), Javascript und CSS, die auf verschiedenen Plattformen ausführbar sind. Der Zugriff auf Hardwarekomponenten wie Kamera und Sensoren erfolgt durch Wrapper. Wrapper sind Adapter oder Schnittstellen, um auf die nativen APIs zugreifen zu können. Abbildung 4.5 zeigt die Kommunikation und Interpretierung des webbasierten Interpreters PhoneGap von Adobe.

Die **Laufzeit** ist eine Schicht und Ausführungsphase, welche die mobile App auf der nativen Plattform lauffähig macht. Bei diesem Ansatz wird der Quellcode in

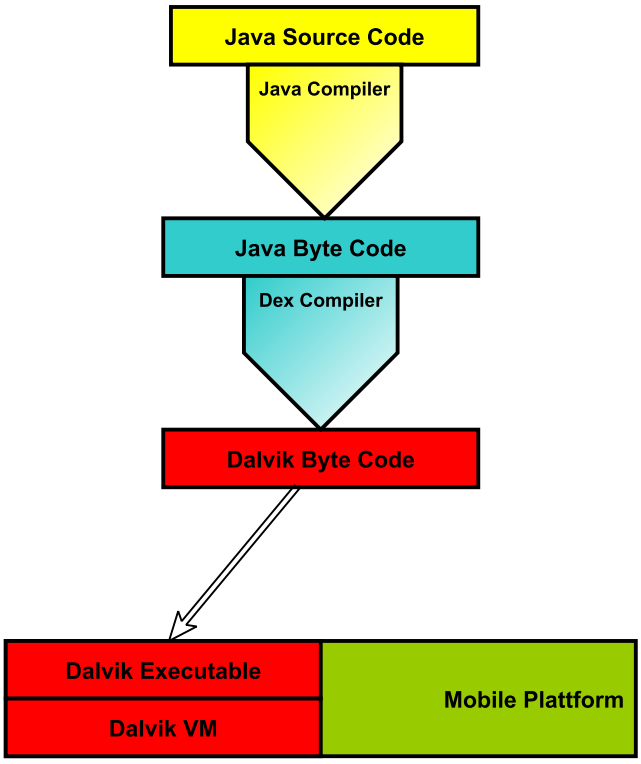

Abbildung 4.4: Ablauf des Dalvik VM Interpreter

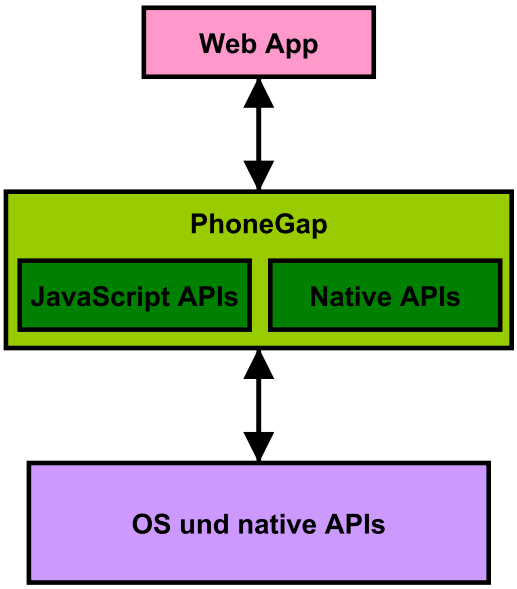

Abbildung 4.5: Vereinfachter Ablauf des PhoneGap Interpreters von Adobe

Bytecode umgewandelt und dann zur Laufzeit ebenfalls von einer virtuellen Maschine ausgeführt. In Abbildung 4.6 wird die Verarbeitung von Appcelerators Titanium-Interpreters dargestellt.

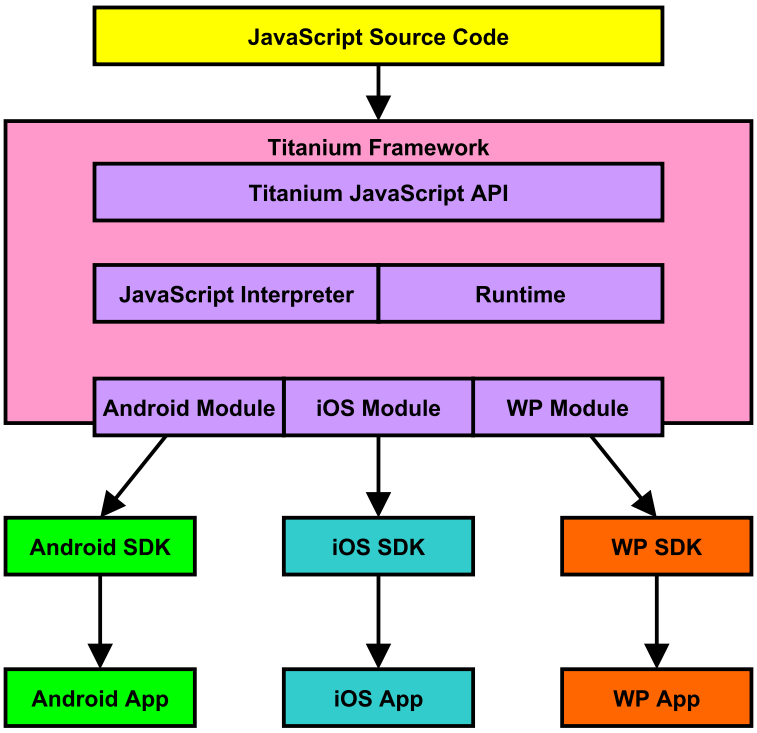

Abbildung 4.6: Ablauf des Titanium Interpreters von Appcelerator

4.2.4 Modellierung

Bei der Modellierung verwenden Entwickler abstrakte Modelle, um die Funktionen und/oder die Benutzeroberfläche der Anwendungen zu beschreiben. Diese Modelle werden für jede Zielplattform in entsprechenden Quellcode transformiert. Hierbei gibt es die Ansätze des Model-Based User Interface Development (MB-UID) und des Model-Driven Development (MDD).

MB-UID wird genutzt, um die Benutzeroberfläche durch die formale Beschreibung von Aufgaben, Daten und Benutzern einer App automatisch zu generieren. Hierbei wird zwischen der Benutzeroberfläche und der Logik unterschieden. Für die Generierung existieren zwei Strategien:

- Die Generierung zur Laufzeit der App, die Websysteme adaptiert und auf Anfrage- und Antwortprotokollen (request/response) basiert. Eingeschränkt wird dies durch die Voraussetzung einer dauerhaften Verbindung zu einem Server.
- Die Generierung während der Entwicklungszeit, also vor Ausführung der Anwendung. Hier kann der Entwickler das generierte Interface überprüfen und zu jeder Plattform spezifische Funktionalitäten hinzufügen. Dabei kann die Funktion zur Verbindungsart festgelegt werden, ob eine dauerhafte Verbindung bestehen soll oder zu einem selbst bestimmten Zeitpunkt synchronisiert wird.

Das Hauptkonzept von **MDD** ist die Generierung von plattformspezifischen Versionen, basierend auf dem plattformunabhängigen, abstrakten Modell. Das Modell wird zum Beispiel durch Domain-Specific Language (DSL) beschrieben.

4.2.5 Cloudbasiert

In diesem Ansatz wird die Logik der Anwendung nicht lokal auf dem Gerät verarbeitet, sondern auf einem Cloudserver. Dabei werden einige Cloudeigenschaften genutzt, wie Flexibilität, Virtualisierung, Sicherheit und dynamisches Management. Die Clientanwendung ist dabei weitestmöglich reduziert, da diese nur Basisprozesse zur Kommunikation benötigt. Dies wird Thin-Client genannt, da, wie in Abbildung 4.7, nur Ein- und Ausgabe verarbeitet werden müssen. Cloudbasierte Anwendungen sollen dadurch besonders energieeffizient sein.

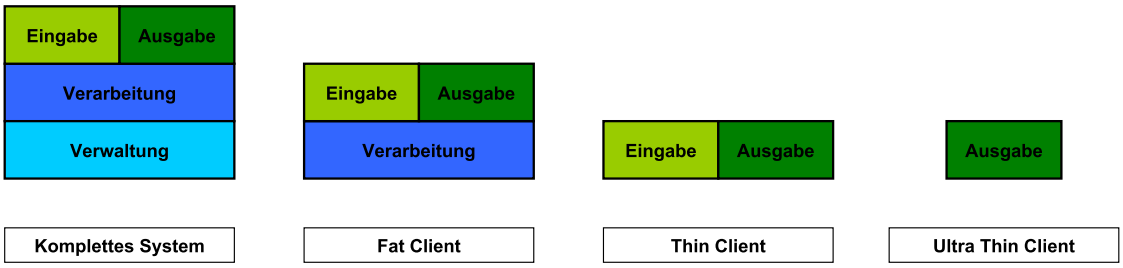

Abbildung 4.7: Aufgaben von Client-Anwendungen

Da zur Zeit der Bearbeitung zu diesem Ansatz keine praktischen Umsetzungen gefunden wurden, sondern nur der theoretische Aufbau einer solchen Applikation, wird dieser Teil mangels Beispielen nicht weiter vertieft.

4.2.6 Vereinigung

Dieser Ansatz versucht die besten Eigenschaften verschiedener Ansätze zusammenzuführen (Merge), von den jeweiligen Vorteilen zu profitieren und Nachteile zu minimieren.

Ein unbetitelter Lösungsansatz vereinigt den komponentenbasierten Ansatz mit dem Cross-Compiler und einer darauf angepassten Universalsprache. Um die nativen Hardwarefunktionen wie Kamera und GPS sowie native Softwareeigenschaften wie Buttons und andere Interaktionsfelder anzusprechen, wird eine Sammlung an spezialisierten Komponenten erstellt. Implementierungen dieser Komponenten können durch gemeinsame Schnittstellen für jede Zielplattform erfolgen. Dieses Framework soll dem Entwickler ermöglichen Applikationen zu entwickeln, die auf nativen Code und der definierten Universalsprache basieren. Diese Sprache wird der App als zusätzliche Kommunikationsschicht und Schnittstelle hinzugefügt, um die Komponenten und deren Methoden anzusprechen (vgl. Abb. 4.8). Der Entwickler implementiert nur eine minimale Grundstruktur der App auf nativer Basis, welche die Benutzerschnittstelle und Navigation beinhaltet. An welcher Stelle und auf welche Weise die Komponenten integriert werden, wird durch die Universalsprache definiert. Das Framework regelt die Codeintegrierung innerhalb des nativen Codes. Bei diesem Lösungsansatz ist es erforderlich, die Benutzerschnittstelle für jede Plattform manuell zu definieren. Dabei liegt der funktionale Fokus auf allgemeingültigen Methoden.

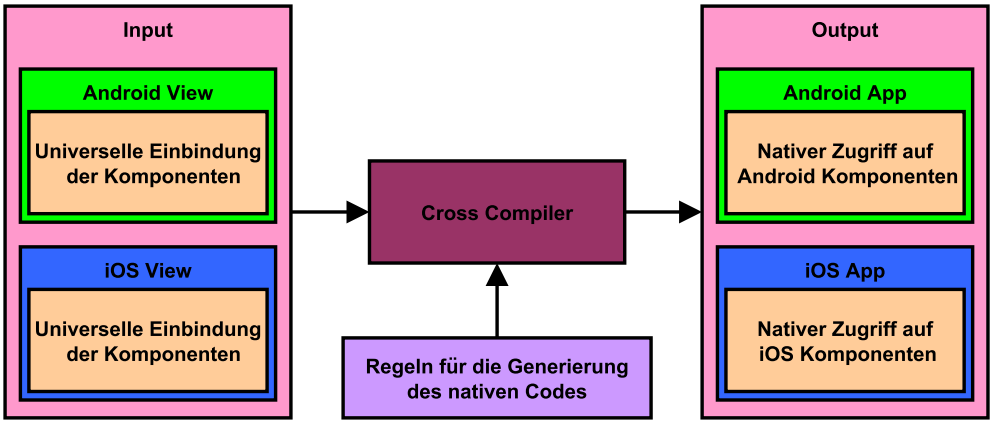

Abbildung 4.8: Funktion eines komponentenbasierten Mergeansatz

Integrated Cross-Platform Mobile Development (ICPMD) ist eine weitere

Lösung, die auf dem Vereinigungsprinzip aufbaut und drei Verwendungsszenarios unterstützt. Diese Szenarios sind, wie in Abbildung 4.9 dargestellt, abhängig von dem gegebenen Input.

Der Entwickler hat...

1. ...bereits ein bestehendes Projekt (z.B. Windows Phone) und möchte dies auf weitere Plattformen (z.B. iOS und Android) ausweiten.

2. ...definierte Anforderungen und möchte daraus, auf bestimmten Zielplattformen, eine mobile App erzeugen.

3. ...ein Projekt basierend auf dem abstrakten Modell und möchte dies aktualisieren und speichern oder daraus, auf bestimmten Zielplattformen, eine mobile App erzeugen.

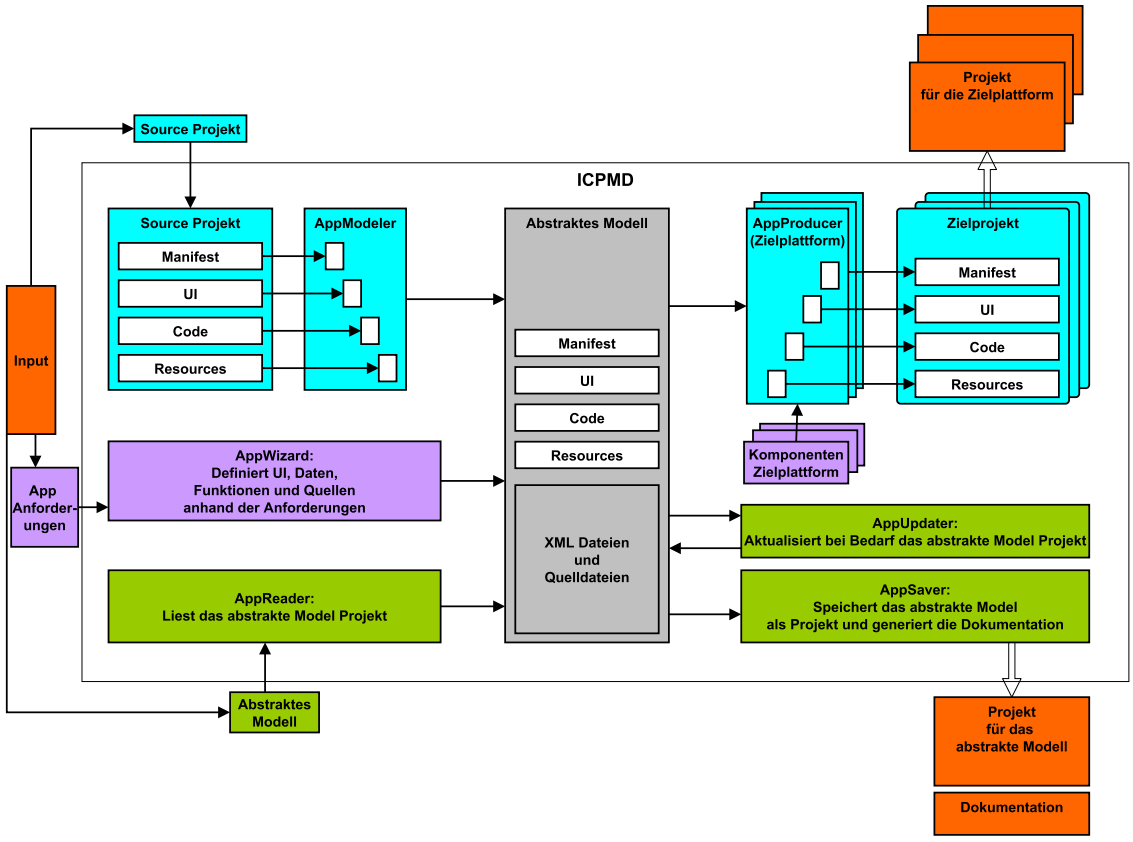

Abbildung 4.9: Drei Szenarien bei ICPMD

4.3 Übersicht der Ansätze

Tabelle 4.1 fasst die Ansätze zur plattformübergreifenden Entwicklung in Kurzform zusammen und benennt die jeweiligen Vor- und Nachteile, sowie die Projekttitel bekannter Lösungen.

Ansatz	Unteransatz	Pro	Contra	Beispielhafte Lösungsansätze
Kompilierung	Cross-Compiler	Wiederverwertung eines existierenden Quellcodes durch Cross-Kompilierung auf eine andere Plattform. Die resultierende Applikation ist nativ und besitzt somit derartige Vorteile.	Die Zuordnungen zwischen zwei Sprachen sind sehr aufwendig, so dass hauptsächlich die gemeinsamen Eigenschaften berücksichtigt werden.	MoSync, Corona, Neomades, XMLVM
	Trans-Compiler	Kann genutzt werden, um veraltete Applikationen und deren veralteten Code auf eine neuere Version derselben Sprache zu übersetzen. Wiederverwertung eines existierenden Quellcodes durch Trans-Kompilierung auf eine andere Plattform. Die resultierende Applikation ist nativ und besitzt somit derartige Vorteile.	Konzentriert sich ausschließlich auf die gemeinsamen APIs von Quell- und Zielsprache. Benötigt regelmäßige Aktualisierungen, um jeweilige Änderungen zu unterstützen.	JUniversal
Komponentenbasiert		Vereinfacht die Unterstützung neuer Plattformen durch definierte Schnittstellen bei implementierten Komponenten.	Konzentriert sich nur auf die Gemeinsamkeiten der unterstützten Plattformen.	Theoretisch
Interpretierung	Webbasiert	Leicht zu erlernen und zu benutzen, da es auf bekannten Web-Technologien aufbaut.	Die Benutzerschnittstelle von webbasierten Apps besitzen nicht das gleiche Aussehen und Handhabung wie bei nativen Apps. Geringere Perfomance gegenüber nativen Apps.	PhoneGap, Rhomobile, xFace
	Virtuelle Maschine	Geringere Gesamtgröße und schnellerer Download der Apps, da alle benötigten Bibliotheken und Funktionen in der VM gespeichert sind.	Langsame Ausführung der Applikation auf der VM. Die VM muss erst aus einem Store geladen werden, was auf iOS nicht unterstützt wird.	MobDSL
	Laufzeit	Der Quellcode muss nur einmal geschrieben werden.	Geringe Perfomance beim Laden der App, da der Interpretierungsvorgang bei jeder einzelnen Ausführung stattfindet.	Titanium
Modellierung	MD-UID	Spart Entwicklungszeit durch Generierung des UI-Codes. Nützlich für Prototyping, durch schnelle UI-Entwicklung und frühe Evaluierung der Benutzerfreundlichkeit.	Muss sich bei den einzelnen Plattformen auf verallgemeinerte Benutzerschnittstellen konzentrieren.	XMobile
	MDD	Die Modellierungssprache ist effektiv, um Anforderungen zu definieren. Der Fokus liegt auf der Funktionalität und nicht auf der technischen Implementierung.	Kann existierenden, nativen Quellcode nicht verarbeiten.	JSAF, MD2, Jelly, AppliDE
Cloudbasiert		Verarbeitungsprozesse werden auf einen Cloudserver ausgelagert.	Das Endgerät und die App benötigen eine schnelle, permanente Netzwerkverbindung.	Theoretisch
Vereinigung		Vorteile aus den Stärken anderer Ansätze. Bietet dem Entwickler vielseitige Möglichkeiten.	Benötigt hohen Entwicklungsaufwand.	ICPMD

Tabelle 4.1: Übersicht aller Ansätze

4.4 Plattformübergreifende Entwicklung mobiler Applikationen ohne den Schwerpunkt Spieleentwicklung

Plattformübergreifende Entwicklung für mobile Plattformen ist nicht nur für die Spieleentwicklung interessant. In der Bachelorthesis „*Plattformabhängige und –unabhängige Entwicklung mobiler Anwendungen am Beispiel von Geo-Wikipedia-App*“ (Vehse 2014) wird ebenfalls die plattformübergreifende Entwicklung analysiert, jedoch liegt hier der Fokus nicht auf mobilen Spieleapplikationen und zugehörigen Entwicklungswerkzeugen. Vehse geht in Kapitel 2.2 auch auf die verschiedenen Herangehensweisen ein und klassifiziert deren Resultate. Weiterhin wird eine Auswahl der bekannteren Cross-Plattform Entwicklungstools (PhoneGap, Xamarin, Appcelerator) analysiert. Im Gegensatz zu der Arbeit von Benjamin Vehse, setzt diese Thesis den Schwerpunkt auf Frameworks und Engines zur Spieleentwicklung und geht daher nicht weiter auf die in seiner genannten und analysierten Entwicklungswerkzeuge ein.

5 Plattformübergreifende Spieleentwicklung

5.1 Definition von Anforderungen für vergleichbare Entwicklungstools

Das Angebot an Tools für die plattformübergreifende Spieleentwicklung ist vielfältig und es bestehen Unterschiede in der Funktionalität und den Möglichkeiten. Da ein Vergleich aller angebotenen Produkte den Umfang dieser Arbeit drastisch überschreiten würde, werden zuerst grundlegende Anforderungen definiert.

Das Framework soll folgende Anforderungen erfüllen:

- Mobile Geräte gehören zu den Zielplattformen, wobei mindestens Android und iOS enthalten sein müssen.
- Sowohl 2D, als auch 3D Spiele können entwickelt werden.
- Es steht mindestens eine objektorientierte, statisch typisierte Programmiersprache zur Wahl.
- Es handelt sich um ein aktuelles Framework, mit regelmäßigen Updates und einer aktiven Entwicklergemeinschaft.
- Ein kostenfrei und kommerziell nutzbarer Produkttyp steht zur Auswahl.

Trotz der geforderten Übereinstimmungen ist es wünschenswert, dass für die Codebasis nicht dieselben Programmiersprachen genutzt werden, um den Vergleich abwechslungsreicher und kontrastvoller zu gestalten. Android und iOS sind derzeit die beiden entscheidenden mobilen Plattformen und sollen als Testsysteme für die Beispielapplikation genutzt werden. Zudem ist es wichtig, dass die Entwicklungswerkzeuge den

aktuellen Stand der Zielsysteme unterstützen und sich regelmäßig weiterentwickeln. Dazu gehört eine dem Zustand entsprechende Dokumentation, als auch eine lebendige Community. Dies soll für die nähere Zukunft eine gewisse Sicherheit der weiteren Existenz gewährleisten. Statisch typisierte Programmiersprachen bieten bei qualifiziertem Umgang oft bessere Performanceleistungen, im Vergleich zu dynamischen Skriptsprachen. Durch diese Anforderung wird partiell vorausgesetzt, dass der plattformübergreifende Ansatz durch Kompilierung umgesetzt wird. Daraus folgt, dass Frameworks ausgeschlossen werden, die hybride Applikationen erzeugen. Dies sind Apps die auf webbasierten Techniken, wie HTML, CSS und JavaScript basieren. Für Smartphones ist es heutzutage technisch keine besondere Eigenschaft mehr, 3D Spiele zu unterstützen. Die Konzeption der Beispielapplikation soll sich deshalb die Wahl zwischen 2D und 3D vorbehalten können. Ein kostenfreier Bezug sichert eine größere Entwicklercommunity und die Möglichkeit, sich barrierefrei und ohne Zeitdruck mit einem Werkzeug zu arbeiten. Anhand dieser Anforderungen werden passende Werkzeuge ausgewählt.

5.2 Gamespezifische Frameworks und Engines

Aufgrund der zuvor gestellten Anforderungen, wurde die Auswahl der Vergleichswerkzeuge stark reduziert. Die Wahl fiel auf die beiden Frameworks libGDX und Cocos2D-X und die Engine Unity3D, welche zu den populärsten Tools in der Spieleentwicklung zählen. Diese werden in den folgenden Unterkapiteln vorgestellt und darauf in Kapitel 6 anhand der theoretischen Angaben und Möglichkeiten analysiert.

5.2.1 libGDX

Das auf Java basierende Entwicklungsframework libGDX ist unter Apache 2.0 lizensiert und Open Source. libGDX bringt eine Menge an grundlegenden Bibliotheken für 2D und 3D Spiele mit. Dabei wird aber nicht jede Eventualität abgedeckt, was aber auch gar nicht gewollt ist. Für spezielle Fälle sollen bei Bedarf dem Projekt entsprechend spezialisierte Module hinzugefügt werden. Dies kann bei der Erstellung eines neuen Projekts oder nachträglich geschehen. Dies soll garantieren, dass nur die Funktionalität integriert ist, die auch wirklich benötigt wird. Dafür arbeitet libGDX stark mit anderen quelloffenen Frameworks und Bibliotheken zusammen. Durch die Programmiersprache Java, ist die Erstellung von Android Anwendungen problemlos

möglich. Für die Generierung von iOS Apps werden die Fähigkeiten des Kompilierer RoboVM verwendet. (libGDX 2013)

5.2.2 Cocos2D-X

Cocos2D-X ist ein Open Source Framework, unter MIT lizensiert. Auch hier bestehen zahlreiche Schnittstellen und Kooperationen mit externen Anbietern, wovon einige bereits schon in die Bibliothek integriert wurden. Entwickler haben für die Codebasis die Wahl zwischen C++, Lua und JavaScript. Cocos2D-X ist der plattformübergreifende Ableger der vielseitigen Cocos Reihe. Trotz der Namensgebung können 3D Anwendungen erstellt werden. Das Programm Cocos Studio unterstützt bei der Gestaltung von Spielszenen, wobei die grafische Oberfläche mit Texturen, Sprites und Menüobjekten angeordnet und daraufhin für die IDE exportiert werden kann, um die Spiellogik zu implementieren. Projekte können aus der Kommandozeile oder einer eigenen Desktopanwendung namens Cocos heraus erstellt werden. Cocos2D-X findet laut eigenen Angaben auch unter größeren Spieleentwicklern Anklang und wird häufig in ostasiatischen Ländern wie China, Japan und Südkorea verwendet. (Cocos2D-X 2015)

5.2.3 Unity3D

Unity3D ist derzeit die international führende Game-Engine. Das System ist proprietär, besitzt aber eine riesige Menge an Schnittstellen für die Nutzung von Objekten externer Anwendungen. Auch wird hier die größte Anzahl an aktuellen Zielplattformen unterstützt. Unity3D findet Verwendung bei Entwicklern verschiedenster Bereiche, wie Hobby, Indie und sogar bei professionellen Studios. In dem Editor der Engine können Spieleelemente direkt in der Szene hinzugefügt, transformiert, mit Komponenten erweitert und direkt getestet werden. Spiellogik kann durch C# Klassen, UnityScript oder Boo Skripte in einer IDE umgesetzt werden. Diese können auch einem Gameobjekt als Komponente hinzugefügt werden. Werte von globalen Variablen oder Objekte sind im Editor direkt veränderbar. Zudem ist es möglich, Objekte innerhalb eines Tests zur Laufzeit anzupassen. Mit sogenannten Prefabs hat man die Möglichkeit, Kompositionen von Spielobjekten zur Wieder- und Mehrfachverwendung zu speichern. Der zugehörige Assetstore stellt einen vielfältigen Marktplatz dar, der verschiedenste Spielinhalte oder komplette Projekte anbietet. (Unity3D 2015)

5.2.4 Weitere Frameworks

Weitere Werkzeuge zur Spieleentwicklung, die den Anforderungen entsprechen, aber nicht für den Vergleich weiter analysiert werden konnten, sollen an dieser Stelle kurz erwähnt werden.

MonoGame
Eine Open Source Implementierung von Microsofts XNA Framework, das eine Vielzahl an Zielplattformen aus verschiedenen Bereichen unterstützt. Als Programmiersprache wird C# genutzt. (MonoGame 2016)

AppGameKit
Eine Lösung zur Spielentwicklung die zwar offiziell nicht kostenfrei, aber für kleines Geld erhältlich ist. Es kann C++ oder eine eigens kreierte Skriptsprache namens BASIC genutzt werden. (AppGameKit 2015)

Lumberyard
Dies ist eine kostenlose, Open Source Spiele-Engine von Amazon, die zur Zeit der Bearbeitung als Betaversion veröffentlicht wurde und basiert auf der CryEngine. (Lumberyard 2016)

6 Analyse der Frameworks

6.1 Zielplattformen

Die Menge an Zielplattformen unterscheidet sich in den Bereichen Mobil, Destop und Web nur geringfügig. In Tabelle 6.1 wird deutlich, dass sich die Menge an unterstützten Plattformen in diesen Kategorien, bei Cocos2D-x und Unity3D kaum unterscheidet. Das Blackberry OS wurde von beiden, hauptsächlich mangels Nachfrage, aus dem Programm genommen. Einzig libGDX stellt noch die Möglichkeit, Spiele hierfür zu entwickeln. Dafür befinden sich in dem Katalog von libGDX die meisten Lücken, wovon die gravierendste, aber derzeit noch verzeihbare, das Fehlen von Windows Phone und Universal Windows ist. Dennoch bieten alle drei Frameworks die Möglichkeit, gleichzeitige Entwicklung bei iOS und Android, als die wichtigsten mobilen Plattformen zuzulassen.

	Cocos2D-X	LibGDX	Unity3D
Mobil			
iOS	X	X	X
Android	X	X	X
Windows Phone 8	X		X
Tizen	X		X
Blackberry		X	
Desktop			
Mac	X	X	X
Windows	X	X	X
Universal Windows Platform	X		X
Linux / Steam OS	X	X	X
Web			
Web GL	X	X	X
Web Player			X
Java Applet		X	
Gesamt	9	8	10

Tabelle 6.1: Unterstützte Zielplattformen der Frameworks (Unity3D 2015, Cocos2D-X 2015, libGDX 2013)

6.2 Programmiersprachen

Die gewählten Spieleframeworks unterscheiden sich bei den verfügbaren Programmiersprachen komplett untereinander. Das verschafft Entwicklern mit unterschiedlichen Kenntnissen, einen leichteren Einstieg in die Spieleentwicklung, durch die Wahl einer vertrauten Sprache. Da manche Frameworks auch mehr als eine Sprache unterstützen, werden diese zugunsten der Übersichtlichkeit in eigenen Unterkapiteln behandelt.

6.2.1 libGDX

libGDX nutzt einzig und allein Java für die Entwicklung, was für reine Android Spiele ein großer Vorteil ist. Denn dadurch werden bei der Kompilierung kaum Kompromisse eingegangen. Java ist eine weitverbreitete, objektorientierte Programmiersprache, die erstmalig 1995 von dem Unternehmen Sun Microsystem vorgestellt wurde. 2010 übernahm das Unternehmen Oracle Sun Microsystem und die Weiterentwicklung von Java. Es entstanden verschiedene spezialisierte Technologielösungen. Die bekanntesten und meistgenutzten stellen dabei Java SE (Standard Edition) und Java EE (Enterprise Edition) dar. Java SE beinhaltet die komplette Standardbibliothek, wobei Java EE um Bibliotheken für die Entwicklung von Server-, Netzwerk- und Webanwendungen erweitert wurde. Das derzeit aktuelle JDK 8 soll 2016 auf Version 9 aufsteigen. (Schmidt 2015) Die Sprache gilt als plattformübergreifend, da Programme in einer virtuelle Maschine ausgeführt werden, der JVM(Java Virtual Machine). Diese führt den bei der Kompilierung erstellten Bytecode aus und prüft das Programm auf Laufzeitfehler. Die Syntax lehnt sich an C++ und C. Die Typisierung ist statisch und gilt allgemein als sicher im Speicherbereich. (Hölzl & Raed & Wirsing 2013)

6.2.2 Cocos2D-X

Mit **Coco2D-X** hat man die Wahl zwischen C++, Lua und JavaScript, welche auf allen unterstützten Plattformen funktionieren. Einzige Einschränkung liegt bei JavaScript, da diese sich bei Cocos2D-X nicht mit Windows Phone verknüpfen lässt. (Cocos2D-X 2015)

Lua ist eine schnelle, von der Syntax simpel gehaltene Skriptsprache, mit objektorientierten Eigenschaften. Die geschriebenen Skripte werden durch einen Interpreter in

Bytecode übersetzt und ist kompatibel mit der Sprache C. Lua befindet sich derzeit in Version 5.3 und wird häufig in der Spieleentwicklung eingesetzt. (Lua 2015)

Mit der Unterstützung von C++, welches direkt von C abgeleitet ist, hat man die Möglichkeit äußerst performante und portable Software zu schreiben. Voraussetzung ist dabei ein sicherer Umgang, da C++ im Vergleich mit anderen High-Level Sprachen, sich weniger um Logikfehler kümmert und entsprechend auch keine Warnungen ausgibt. Die aktuelle Version ist C++14.

Für Entwickler die beispielsweise Erfahrungen im Webbereich besitzen, bietet die Verwendung von JavaScript einen weiteren Einstieg in Cocos2D-X. JavaScript ist eine Skriptsprache die es seit 1995 gibt und für dynamisches HTML in Webbrowsern entwickelt wurde. Trotz der Namensähnlichkeit zu Java, unterscheiden sich die beiden Sprachen grundlegend voneinander. JavaScript wird in der Regel genutzt um Clientseitige Logik innerhalb von HTML Dokumenten zu ermöglichen. Mit der Programmiersprache Java hingegen, können eigenständige Anwendungen aufgebaut werden. Die Sprache folgt Spezifikationen der privaten Normungsorganisation ECMA (European Computer Manufacturers Association), die Standardisierungen von Informationstechnologien entwickelt, die Richtigkeit deren Verwendung fördert und eine frei zugängliche Veröffentlichung dazu liefert. Der standardisierte Kern von JavaScript, wird als ECMAScript (ECMA 262) bezeichnet und beschreibt eine dynamische Typisierung, mit objektorientierten, aber klassenlosen Eigenschaften. Objekte basieren auf sogenannten Prototypen, die als Funktionen beschrieben werden. Nach der Instanzierung ist es möglich, das Objekt um zusätzliche Eigenschaften zu erweitern. Skripte können imperativ, aber auch funktional aufgebaut werden. Geschriebene Skripte werden durch einen Interpreter übersetzt. Basierend auf JavaScript, entstanden viele Bibliotheken und Frameworks, um beispielsweise serverseitige Netzwerkanwendungen zu betreiben (Node.js) oder Entwurfsmuster aus der Softwareentwicklung zu nutzen (Angular.js). ECMAScript befindet sich seit 2015 in Version 6. (Brown 2015)

6.2.3 Unity3D

In **Unity3D** finden die Sprachen C#, UnityScript und Boo Verwendung. Alle für Unity3D nutzbaren Sprachen entspringen Microsofts .NET Framework. Das bedeutet auch, dass es möglich ist weitere .NET Sprachen zu benutzen, wenn diese ihre Skripte in das DLL (Dynamically Linked Library) Format kompilieren können. Diese

DLL Dateien können dann dem Unity Projekt hinzugefügt und verwendet werden. (Unity3D 2015)

Boo ist eine von Python beeinflusste Sprache für .NET und Mono, welche ohne Klammern und Semikolons auskommt. Die Typisierung ist generell statisch, kann aber trotzdem dynamische *Duck-Typing* Eigenschaften nutzen. (Boo 2015) Der Name Duck-Typing entsprang dem sogenannten *Ententest* zur Typisierung, welcher wiederum auf einem Gedicht von James Whitcomb Riley zurückzuführen ist.

"*When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck.*" (James Whitcomb Riley 1849–1916)

Beim Duck-Typing werden Objekte nicht durch ihre Klasse typisiert, sondern durch die vorhandenen Attribute und Methoden. Daher findet die Typisierung erst zur Laufzeit durch den Interpreter statt. Aufgrund des seltenen Gebrauchs, wird Boo seit Unity 5.0 in der Dokumentation nicht mehr unterstützt.

Oft wird UnityScript in der Unity Community, aber auch von dem Unternehmen selbst, fälschlicherweise mit JavaScript gleichgesetzt, als wären die beiden Sprachen äquivalent. Auch wenn syntaktische Ähnlichkeiten bestehen, gibt es große semantische Unterschiede. UnityScript kann, wie die meisten objektorientierten Programmiersprachen, Klassen definieren und daraus Objekte erstellen. JavaScript hingegen besitzt zwar ebenfalls objektorientierte Eigenschaften, wobei diese anstatt Klassen aber sogenannte Prototypes verwendet. Des Weiteren kann UnityScript im Gegensatz zu JavaScript, Klassen, Objekte, Funktionen und Variablen mit Zugriffsmodifikatoren/Sichtbarkeiten versehen. UnityScript wurde speziell für die Unity3D Engine konzipiert und ist proprietär, was es schwierig macht, genaue Spezifikationen zu finden. (Unify Community Wiki 2014)

C# oder auch Visual C# ist eine von Microsoft entwickelte Programmiersprache, die durch ECMA-334 standardisiert ist und sich aktuell in Version 6.0 befindet. Die Standardisierung bezieht sich allerdings nur auf die Sprache selbst und nicht auf Programme die in Verbindung mit dem .NET Framework realisiert worden. Alternativ zu .NET bietet Mono ein Open Source Framework inklusive Compiler für C# an. Die Syntax der objektorientierten Sprache weißt große Ähnlichkeiten mit Java auf und wird oft mit dieser verglichen. Die Typisierung ist grundsätzlich statisch, wobei

optional auch dynamische Typen genutzt werden können. (Skeet 2014) Laut Statistik, sind etwa 80% der Unity Projekte in C# geschrieben. (Unity3D 2014)

6.3 Entwicklungsumgebungen

Für den Prozess der Entwicklung besteht bei jedem Framework die Wahlmöglichkeit zwischen mehreren IDEs. Praktisch kann selbstverständlich jeder textbasierte Editor zur Programmierung verwendet werden, aber um den Komfort von passendem Code-Highlighting und die direkte Projekterstellung zu nutzen, empfiehlt es sich die unterstützten Entwicklungsumgebungen zu verwenden. Diese stehen bei den drei vorgestellten Spieleframeworks in direkter Abhängigkeit zu dem verwendeten Betriebssystem.

6.3.1 Systembedingte Einschränkungen

Es gilt für jedwede Erzeugung von iOS Applikationen, das die Apple IDE Xcode unumgänglich ist. Diese ist auch ausschließlich unter OS X Systemen erhältlich und einsetzbar. Um die iOS Applikation testen zu können, benötigt man seit Xcode 7 nur noch eine gültige Apple ID. Um die App zu veröffentlichen, ist man letztlich aber dennoch verpflichtet eine Mitgliedschaft des Apple Developer Program zu erwerben. Diese geht mit aktuellen 99 US Dollar jährlich einher. In der Mitgliedschaft sind die Rechte zur Veröffentlichung von iOS, watchOS und OS X Anwendungen, sowie das Anlegen und Durchführen von Beta-Tests mit Testflight und diverse andere Features. (Apple Developer 2016)

Um iOS Apps mit libGDX zu erstellen, ist man zudem an einen speziellen Kompilierer gebunden, der Java in Obective-C Code übersetzen kann. Hierfür wird mit RoboVM zusammengearbeitet. Die Einbindung dieses Tools ist beispielsweise in Eclipse oder in auf Intellij IDEA basierten Entwicklungsumgebungen möglich. Eine eigene IDE mit Namen RoboVM Studio steht ebenfalls zur Wahl. Für die Nutzung von RoboVM fallen im Normalfall kosten ab derzeit 25 US Dollar für eine Einzellizenz an. Es werden jedoch auch reduzierte bis kostenfreie Modelle für Indie Entwickler oder Studenten angeboten. (RoboVM 2016)

Die Erstellung von Windows Phone Apps ist mit der Verwendung von Visual Studio verbunden. Diese benötigt mindestens Windows 7. Um den Simulator nutzen zu

können, wird Microsofts Virtualisierungstechnik Hyper-V vorausgesetzt, wofür eine Windows 8 Pro oder Windows 10 Pro Version notwendig ist. (Visual Studio 2016)

6.3.2 Unterstützte IDEs

In Tabelle 6.2 werden die Editoren aufgelistet, die für die Bearbeitung des Quellcodes unterstützt und empfohlen werden. Abhängig welche Programmiersprache man bevorzugt, hat man pro Gameframework eine oder mehrere IDEs zur Auswahl. Wie im vorigen Kapitel bereits erwähnt, ist für jede iOS App Xcode unter OS X verpflichtend, ebenso wie Visual Studio für Windows Phone Anwendungen. Daher werden diese nicht zusätzlich für jedes Framework markiert. Die jeweiligen Vor- und Nachteile der einzelnen Entwicklungsumgebungen werden in dieser Arbeit nicht weiter behandelt und dieser Teil soll auch nicht weiter vertieft werden. Es dient lediglich der Vollständigkeit.

IDE	Betriebssystem	Sprache	libGDX	Cocos2D-X	Unity3D
Eclipse	Windows / OS X	Java	X		
IDEA	Windows / OS X	Java	X		
Android Studio	Windows / OS X	Java	X		
NetBeans	Windows / OS X	Java	X		
RoboVM Studio	Windows / OS X	Java	X		
Xcode	OS X	C++		X	
Cocos Code	Windows / OS X	JavaScript / Lua		X	
Visual Studio	Windows	C# / C++		X	X
MonoDevelop	Windows / OS X	C# / UnityScript			X

Tabelle 6.2: Unterstützte Entwicklungsumgebungen zur Bearbeitung der Codebasis (libGDX 2015, Cocos2D-X 2015, Unity3D 2016)

6.4 Native Gerätefunktionen und Schnittstellen

Smartphones und Tablets besitzen in der Regel kaum oder gar keine Hardwarebuttons, die für Anwendungen neu belegbar sind. Spiele die für solche mobilen Systeme entworfen worden, greifen daher auf eine oder zwei der beiden populärsten Steuerungsmechanismen zurück.

Überwiegende Praxis ist es, den vorhandenen Touchscreen für Eingaben und Interaktionen zu nutzen. Dies kann über bekannte Gesten geschehen oder über die Virtualisierung von Aktionsflächen, wie zum Beispiel Buttons und Regler. Alle drei Fra-

meworks verfügen über Möglichkeiten, verschiedene Touchgesten zu unterscheiden. Eine andere Möglichkeit in das Spielgeschehen einzugreifen, bieten die geräteseitigen Hardwaresensoren. Ein Anwendungsbeispiel für den Gyroskop Sensor, wäre bei Rennspielen die Simulierung eines Lenkrades, bei dem durch die Neigung des Gerätes gesteuert wird. Weiterhin gehören Accelerometer, GPS und sowohl eine Kamera zu den üblichen Gerätefunktionen.

Unity3D liefert für fast alle gängigen Gerätefunktionen entsprechende Schnittstellen, um diese zu nutzen oder mit Funktionen zu hinterlegen. Bei **libGDX** kann man auf die meisten Sensorentypen zugreifen, muss aber auf Kamera und GPS verzichten. Hingegen kann **Cocos2D-X** bisher nur den Accelerometer ansteuern, wenn dieser vorhanden ist. Gyroskop, Kamera, GPS Lokalisierung und weiteres werden derzeit nicht unterstützt. Dadurch dass die beiden Frameworks quelloffen sind, ist bei Bedarf eine eigene Implementation theoretisch dennoch möglich. Eine offizielle Erweiterung ist demnach auch abhängig von dem Streben der Entwickler-Community. In Tabelle 6.3 werden die häufigsten Gerätefunktionen und die Unterstützung durch eine Schnittstelle, der einzelnen Frameworks gelistet. Der Vollständigkeit halber ist zu dem Punkt Netzwerkverbindung zu sagen, dass keines der drei Frameworks ein WLAN Modul oder mobiles Internet ein- und ausschalten kann. Aus Gründen der Sicherheit, kann dies nur der Nutzer selbst. Es kann lediglich getestet werden, ob eine aktive Verbindung existiert. Für die Prüfung einer Bluetooth Verbindung, gibt es derzeit für keines eine offizielle Schnittstelle. Wenn ein Projekt die Anforderung einer bestimmten Schnittstelle besitzt, die aber in dem gewählten Framework nicht vorhanden ist, hat man dennoch die Möglichkeit diese plattformabhängig zu implementieren.

Schnittstelle	libGDX	Cocos2D-X	Unity3D
Touchgesten	X	X	X
Netzwerkverbindung	X	X	X
Accelerometer	X	X	X
Gyroskop	X		X
Vibration	X		X
Kompass	X		X
Kamera			X
Geoposition			X
Bluetooth			

Tabelle 6.3: Verfügbare Schnittstellen zu den Gerätefunktionen (libGDX 2013, Cocos2D-X 2015, Unity3D 2015)

6.5 Game Services

Um die Motivation und die Wiederspielbarkeit zu erhöhen, sowie neue Inhalte zur Verfügung zu stellen, können verschiedene Funktionalitäten eingebunden werden. Ein Erfolgssystem basiert auf dem Belohnungsprinzip und kann den Benutzer auf unterschiedliche Weise in seinem Spielverhalten motivieren. Durch definierte Herausforderungen werden verschiedene Aufgaben gestellt, wobei dem Spieler durch das Erreichen dieser, ein visuelles Feedback gegeben werden kann. Das kann beispielsweise in Form von Medaillen, Erhöhung des Spielerlevels oder einer Prozessleiste auftreten. Kompetitive Elemente können mit Bestenlisten erzielt werden, in denen bestimmte Metriken von den Spielern sortiert aufgelistet werden können. Ranglisten über die höchste Punktzahl oder die längste Spielzeit geben Anreiz sich mit anderen zu messen. In rundenbasierten und Echtzeit Mehrspielermodi treten zwei oder mehr Spieler gegeneinander an, um in einem Wettbewerb den besten Spieler zu ermitteln. Eine andere beliebte Variante sind Modi, in denen die Teilnehmer kooperieren, um Spielziele gemeinsam zu erreichen oder einfach unterhaltsame Erfahrungen zu schaffen. Um verbesserungswürdige Schwachstellen in der Anwendung zu ermitteln und das Verhalten von Spielern zu messen, werden Analysesysteme verwendet. Diese ermöglichen Einblicke auf Aktivitäten und Fortschritte der Spieler und Informationen zu getätigten Käufen. Auch wie häufig ein Benutzer die Anwendung gebraucht kann gemessen werden. Diese Statistiken verhelfen zu einem besseren Verständnis der eigenen Anwendung.

Für den Erwerb von zusätzlichen, applikationsinternen Inhalten und die Abwicklung von Zahlungen sind Shop Systeme notwendig. Dies wird bei plattformübergreifenden Applikationen durch die Implementierung einer Bibliothek ermöglicht, die zu den gewünschten Zielplattformen und deren Stores, passende Schnittstellen bereitstellt. In Tabelle 6.4 werden APIs aufgelistet, die populäre Game Services in den Frameworks unterstützen. Die Tabelle zeigt ausschließlich APIs, die plattformübergreifende Eigenschaften aufweisen und auf den beiden wichtigsten, mobilen Zielplattformen iOS und Android funktionieren.

Game Service	libGDX	Cocos2D-X	Unity3D
Erfolge	Play Game Services App42	Play Game Services App42	Play Game Services App42
Bestenlisten	Play Game Services App42	Play Game Services App42	Play Services App42
Multiplayer	Play Game Services App42 Nextpeer	Play Game Services App42 Nextpeer	Play Game Services App42 Nextpeer
Spielstand sichern	Play Game Services App42	Play Game Services App42	Play Game Services App42
Analyse	App42	App42	Unity Analytics App42 Soomla
Shop System	gdx-pay	SDKBOX-IAP	Soomla
Soziale Netzwerke **Freundesliste**	App42 Nextpeer	App42 Nextpeer	Social API App42 Nextpeer Soomla

Tabelle 6.4: Game Services und plattformübergreifende Schnittstellen

Play Game Services

Das Framework von Google bietet zahlreiche Services für den mobilen Gaming-Bereich. Es werden die meisten in der Tabelle genannten Services für die plattformübergreifende Entwicklung unterstützt. Die Voraussetzung für den Spieler, um diese Features nutzen zu können, ist ein Account in dem Netzwerk Google+. Play Game Services bietet auch Services zur Analyse des Spielerverhaltens, jedoch sind die Angaben von Google und den Spieleentwicklungstools bezüglich der Unterstützung nicht eindeutig. Die Cloud Dienste von Google sind ohne Kosten implementierbar. (Play Game Services 2015)

App42
App42 von ShepHertz Technologies bietet ein umfangreiches Angebot an Cloud Diensten, welches für Indie-Entwickler größtenteils kostenfrei zur Verfügung steht. Es gibt Schnittstellen für eine Vielzahl an Plattformen und Frameworks, sowohl im nativen, als auch im plattformübergreifenden Bereich. Die wichtigsten Game Services werden alle unterstützt. (App42 2016)

NextPeer
NextPeer bietet für die drei Game Frameworks APIs für die Unterstützung von Multiplayer Spielen und Anbindungen an soziale Netzwerke. Für die Nutzung steht ebenfalls eine kostenfreie Variante zur Auswahl. (Nextpeer 2016)

6.6 Produktvarianten

Das Java Framework **libGDX** ist das offizielle Kernprojekt. Eigene Varianten sind durchaus möglich, aber derzeit sind keine weiteren populären Ableger bekannt. Die Community konzentriert sich demnach hauptsächlich auf die Weiterentwicklung der Hauptversion oder auf eigene Erweiterungen in Form von Plugins.

Cocos2D-X ist eine von mehreren Abzweigungen aus der Cocos2D Familie. Stamm der Entwicklung bildet das mit Python entwickelte Cocos2D, welches selbige Sprache auch für die Entwicklung daraus resultierender Desktopanwendungen benutzt. Der erste daraus abgeleitete Ableger Cocos2D-iPhone oder auch bekannt als Cocos2D-ObjC, erlaubt die Implementierung auf mobile Apple Geräte. Diese können mit den plattformüblichen nativen Sprachen Objective-C und Swift konstruiert werden. Aus dieser Version entsprangen direkt wiederum viele weitere Varianten für gezielte Sprachen und Anwendungen, als auch das in dieser Arbeit behandelte Cocos2D-X für die Unterstützung multipler Plattformen. Die Cocos2D Frameworks iPhone, X, HTML5 und dem Editor SpriteBuilder sind kooperierende Projekte und folgen demselben Leitplan. Daher haben sie ein koordiniertes Updateverhalten. (Cocos2D-X 2015)

Unity3D bietet zwei verschiedene Hauptprodukte an, die Personal und die Professional Edition. Bei der Personal Edition handelt es sich um eine frei verfügbare Version, die für jeden zugängig ist. Diese kann für Privatanwender, zu Bildungszwecken und sogar für die kommerzielle Entwicklung benutzt werden. Die kostenfreie Version der

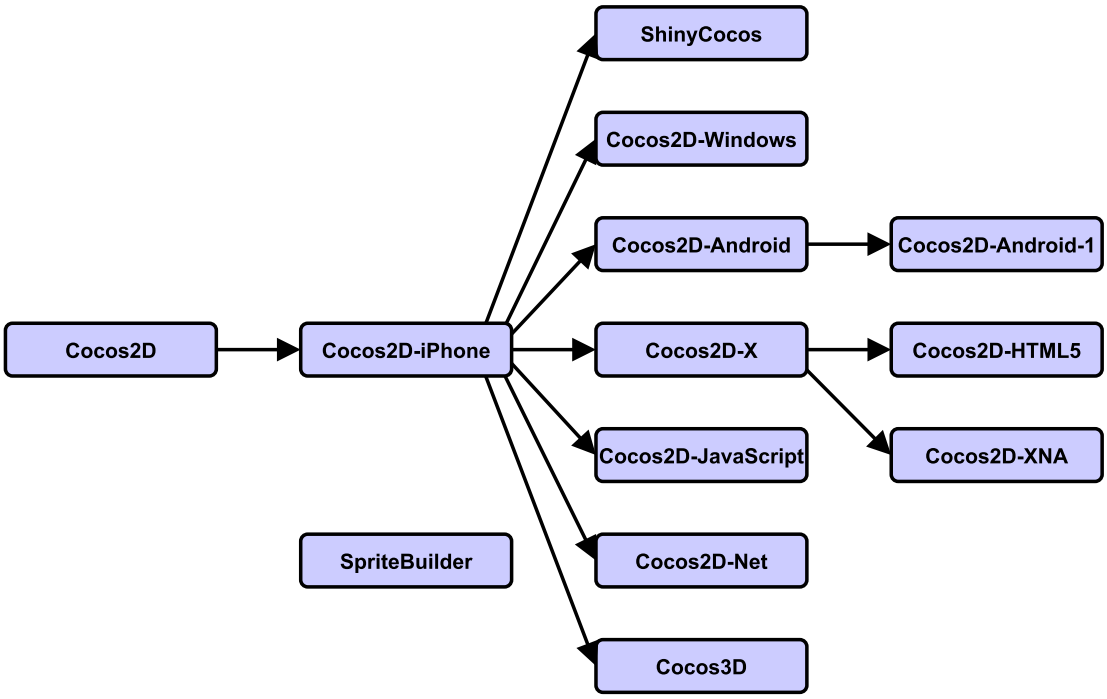

Abbildung 6.1: Ableitungen und Varianten von Cocos2D (Cocos2D-X 2015)

Engine verfügt auch über alle essentiellen Funktionen. Sobald die allgemeinen Einnahmen einer Person oder eines Unternehmens mehr als 100.000 US-Dollar aus dem Vorjahr betragen, besteht die Verpflichtung auf die Professional Edition zu wechseln. In dieser bekommt man dann auch die Option auf zusätzliche Funktionen, wie Analyse Tools und die Möglichkeit Projekte in einer speziellen Cloud zu veröffentlichen. In Tabelle 6.5 werden die Möglichkeiten beider Editionen gegenübergestellt. Das Kostenmodell beginnt derzeit mit einem Abonnement von 75 US-Dollar pro Monat oder einer Einmalzahlung von 1.500 US-Dollar. Die mit der Einmalzahlung erworbene Dauerlizenz gilt allerdings nur für die aktuelle Version und deren Updates. Das bedeutet bei dem Erwerb von Unity Pro 5 hat man nicht automatisch die Nutzungsrechte für die darauf folgende Version 6. Diese müssten dann in Form eines kostenpflichtigen Upgrades erfolgen. Weitere optionale Erweiterungen können mit den iOS Pro und Android Pro Add-Ons erworben werden, die weitere Funktionen zur Personalisierung ermöglichen. Zusätzliche kostenpflichtige Pakete sind die professionelle Unterstützung bei Projekten oder der Zugang zum Quellcode der Engine. (Unity3D 2015)

Edition	Feature
Professional / Personal	Game Engine inklusive aller Funktionen
Professional / Personal	Unterstützung aller Plattformen (ohne Zusatzlizenz Unity Splash Screen bei iOS und Android)
Professional / Personal	Lizenzfreiheit
Professional	Unity Cloud Building
Professional	Team Lizenz
Professional	Game Performance Analyse
Professional	Keine Einkommensbeschränkung
Professional	Personalisierbarer Splash Screen
Professional	Unity Analytics zur Analyse des Spielerverhaltens
Professional	Priorisierte Behandlung bei eingereichten Bugs
Professional	Zugang zu Beta-Versionen
Professional	Unterstützung für zukunftige Plattformen
Professional	Vergünstigungen im Asset Store

Tabelle 6.5: Gegenüberstellung von Personal und Professional Edition (Unity3D 2015)

7 Konzeption und Implementierung einer Beispielapplikation

Um die Möglichkeiten der gewählten Entwicklungstools tiefgehender analysieren zu können, sollten die theoretischen Informationen um praktische Erkenntnisse ergänzt werden. Demnach wird als Teil dieser Arbeit ein Beispiel-Spiel konzipiert, mit den jeweiligen Tools umgesetzt und daraus Applikationen für verschiedene Plattformen erzeugt. Die Anwendung soll dann Anhand von vereinbarten Metriken bemessen und analysiert werden. Die gewonnenen Resultate werden daraufhin in Zusammenhang mit den theoretischen Informationen komplementiert und die Entwicklungstools als Ganzes verglichen.

7.1 Definition von Anforderungen

Applikationen aus den Kategorien Business, Unterhaltung und die meisten weiteren ohne den Fokus auf Spiele, greifen in der Regel auf plattformspezifische Benutzeroberflächen zurück. Native Apps besitzen jeweils technische und optische Konventionen. Cross-Plattform Apps versuchen diese möglichst genau zu generieren und einzuhalten, um dem Benutzer ein natives Look-and-Feel zu bieten. Meist folgen Spiele diesen Regeln nicht, da sie in Abhängigkeit des Spielprinzips auf unterschiedliche Eingabeelemente und UIs zurückgreifen. Das Spiel als solches ist ein komplexes Phänomen, mit dem sich bereits Philosophen, Psychologen, Philanthropen und Forscher aus anderen Fachgebieten auseinandergesetzt haben. Ein Teilbereich der Spielwissenschaft, genannt Ludologie (Die Lehre über das Spiel), befasst sich unter anderem mit der Erforschung des digitalen Spielens. (Junge 2015) Die Forschungsinhalte befassen sich unter anderem mit Fragestellungen aus den Bereichen:

- Geschichte des digitalen Spielens

- Elemente des digitalen Spiels
- Begrifflichkeit zur Klassifizierung von Spielen und Genres
- Regeln und Spielmechanik
- u.v.m.

Wie definiert man aber nun den Inhalt und den Umfang eines Computerspiels? Um sich in den konkurrierenden Theorien zu verlieren, wird hierfür beispielhaft die Gebrauchsdefinitionen von Salen und Zimmerman herangezogen:

„A game is a system in which players engage in an artificial conflict, defined by rules, that results in a quantifiable outcome." (Salen & Zimmerman 2004: 80)

Aus dieser Definition lassen sich einige wesentliche Konzepte entnehmen.

Spieler
Ein Spiel ist etwas, das ein oder mehrere Spieler aktiv spielen. Die Spieler interagieren mit dem System und Methoden des Spiels.

Konflikt
Der Konflikt bezeichnet den Wettkampfcharakter jedes Spiels. Dieser kann verschiedene Formen annehmen, wie kompetitive Elemente oder der Konflikt gegen das Spielsystem.

Regeln
Die Regeln liefern die Struktur des Spiels, in dem sie festlegen was der Spieler machen kann und was nicht.

Quantifizierbares Ziel
Das Ergebnis eines Spiels ist entweder das der Spieler gewonnen oder verloren hat, oder eine Form eines Erfolgsfaktors erhält. Dieser Erfolgsfaktor kann sich beispielsweise durch ein höheres Level oder eine Art Punktzahl bemerkbar machen. (Salen & Zimmerman 2004: 80)

Anhand dieser Definitionen werden die Anforderungen für die Beispielapplikation konkretisiert:

- Es soll eine Spielfigur geben, auf die der Spieler Einfluss nehmen kann.
- Die Spielfigur soll auf Hindernisse treffen, welche die Aktionen des Spielers beeinflussen können.
- Der Spieler gewinnt, wenn er ein definiertes Ziel erreicht oder das Spiel nicht beendet wird.
- Der Spieler verliert, wenn er eine definierte Anzahl an Fehlversuchen erreicht hat, das Spiel abbricht oder andere Aktionen ausführt, die gegensätzlich zum gewinnen des Spiels stehen.
- Der Spieler soll einen Erfolgsfaktor erhalten, um seine Spielweise zu messen.

Darüber hinaus werden weitere allgemeine Anforderungen gestellt, die häufig in Spielen enthalten sind:

- Das Spiel soll einen visuellen Charakter haben und Grafiken verwenden.
- In dem Spiel sollen Animationen vorkommen, wie Bewegungen.
- Das Spiel soll über Audioelemente verfügen, wie Musik oder Soundeffekte.
- Es sollen mehrere Spielszenen verwendet werden.
- Ein Spielobjekt soll in allen Szenen verfügbar sein und seinen Status beibehalten.

Die Applikation soll zudem möglichst simpel gehalten werden, um sie Anhand der einfachsten und elementaren Eigenschaften zu bemessen.

7.2 Spielidee

Ein mobiles Spiel das mitunter durch seine Einfachheit, in kürzester Zeit großen Erfolg erlangte, ist das Spiel Flappy Bird. (Wikipedia - Flappy Bird 2015) In diesem Spiel steuert der Spieler durch Antippen des Touchscreens die Flughöhe eines Vogels, um ihn vor dem Aufprall auf den Boden zu bewahren und durch entgegenkommende Hindernisse zu manövrieren. Hierbei handelt es sich um eine Art Endlos-Spiel, das sich dadurch auszeichnet, dass der Spieler solange spielt, bis er verliert. Verlieren kann

man, indem die Spielfigur mit einem anderen Objekt kollidiert. Es verfügt zudem über eine Punktzahl, die sich erhöht, wenn der Vogel erfolgreich die Hindernisse passiert hat.

In der Anlehnung zu Flappy Bird entstanden zahlreiche Spiele die große Ähnlichkeit aufweisen. Auch wird dieses Spielprinzip häufig als Vorlage verwendet, um beispielhaft den Einstieg in die Spielentwicklung mit einer bestimmten Engine oder einem Framework zu erlernen. Da alle vorher genannten Anforderungen in dieses Spielprinzip hineinpassen, wurde für diese Arbeit ein Spiel konzipiert, das sich an dieser Vorlage orientiert. Das Spiel trägt den Titel: **Happy Bird**.

7.3 Spielfluss

Das Spielprinzip und der Spielfluss bei Happy Bird sind denkbar einfach. Das Spiel wurde zudem auf den Portrait-Modus (Hochformat) festgelegt und es wird durchgängig eine Anzeige mit der aktuellen Framerate dargestellt.

1. Das Spiel wird über das App Icon gestartet.
2. Direkt nach dem Start wird ein Logo des jeweiligen Frameworks oder Engine angezeigt.
3. Daraufhin gelangt man in ein Hauptmenü, das den Titel des Spiels, Hintergrundgrafiken und einen Startbutton anzeigt. Die Hintergrundmusik wird abgespielt.
4. Bei betätigen des Startbuttons wird in die Spielszene gewechselt.
5. Die Spielszene beginnt mit den Hintergrundgrafiken und einem Vogel als Spielfigur.
6. Der Vogel verliert an Höhe, wenn der Spieler nichts tut und steigt bei antippen des Touchscreens an.
7. In einem festgelegten Intervall kommen der Figur nun Hindernisse in Form von grünen Röhren entgegen. Diese besitzen eine Lücke in der Mitte, durch die der Vogel hindurchmanövriert werden muss. Die Position der Lücke befindet sich in einer zufälligen, vertikalen Stelle innerhalb der Sichtbarkeit.

8. Wenn der Vogel erfolgreich das Hindernis passiert hat, erhält man einen Punkt, der oben links im Spiel zu der Gesamtpunktzahl hinzugefügt und dargestellt wird.

9. Das Spiel endet und gilt als verloren, wenn der Vogel mit einem der entgegenkommenden Hindernisse oder dem Boden kollidiert.

10. Der obere Rand der Spielszene wird durch eine unsichtbare Wand blockiert, die nicht passiert werden kann.

11. Bei verlorenem Spiel, wird in das Game Over Menü gewechselt, das wie das Hauptmenü aufgebaut ist. Dies zeigt den Game Over Status an und gibt die Möglichkeit, über den Startbutton ein neues Spiel zu starten.

Die in Kapitel 7.1 definierten Anforderungen, werden somit alle umgesetzt. Die Spielfigur wird durch einen Spritesheet animiert und ist durch Touchscreen-Input steuerbar. Es werden Hindernisse erzeugt, die bei Kollision das Spielende hervorrufen. Der Spieler hat die Möglichkeit Punkte zu erhalten. Die Hintergrundmusik wird in allen drei Spielszenen gespielt. Sie startet bei Wechsel der Szene nicht von vorne, behält einen globalen Status bei und wird in einer Endlosschleife gespielt. Das Spiel wird komplett in 2D gehalten. Auf weitere diverse, für eine Testapplikation unnötige Benutzerannehmlichkeiten wurde verzichtet. Für die Versionierung der Projekte wird Git verwendet.

7.4 Verwendete Werkzeuge

In diesem Kapitel werden die genutzten Softwarewerkzeuge in Tabell 7.1 aufgelistet, die bei der Entwicklung der Testapplikation eingesetzt wurden.

	Betriebssystem	Version
Betriebssysteme		
Mac OS X	Mac OS X	
Windows	Windows	Windows 8.1 Standard Edition 64-Bit
Game-Framework/Engine		
libGDX	Mac OS X, Windows	
Cocos2D-X	Mac OS X, Windows	3.9
Unity3D	Mac OS X, Windows	5.3.2f1
Programmiersprache		
Java	Mac OS X, Windows	
C++	Mac OS X, Windows	
C#	Mac OS X, Windows	
Objective-C	Mac OS X	
Python	Mac OS X, Windows	
Entwicklungsumgebung		
Xcode	Mac OS X	
Android Studio	Mac OS X, Windows	
Visual Studio	Windows	
Monodevelop	Windows, Mac OS X	
Sonstiges		
Android SDK	Mac OS X, Windows	
Android NDK	Mac OS X, Windows	
Android ANT	Mac OS X, Windows	
Mono	Mac OS X, Windows	
RoboVM	Mac OS X, Windows	

Tabelle 7.1: Eingesetzte Software auf den jeweiligen Betriebssystemen und Versionen

7.5 Eingesetzte Komponenten

Bei den eingesetzten und benötigten Komponenten existieren geringe Unterschiede. Um eine Sprite-Animation in Cocos2D und libGDX zu ermöglichen, empfehlen die Frameworks die Nutzung von Spritesheets in Kombination mit einer Property List-Datei (.plist). Bei einem Spritesheet handelt es sich um eine Sammlung von Grafiken in einer Datei. Diese Grafiken können allein oder in einer zusammenhängenden Abfolge stehen. Diese werden dann der Größe entsprechend ausgeschnitten.

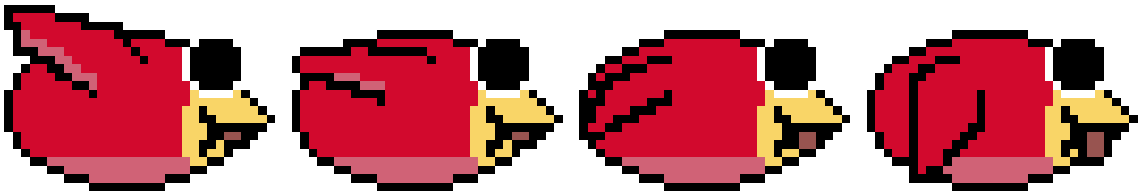

Abbildung 7.1: Spritesheet für die Animation der Spielfigur in Happy Bird

Durch Zuhilfenahme der Property List (Liste von Eigenschaften), die in Zusammenhang mit dem Spritesheet erstellt werden kann, können Informationen über die einzelnen Sprites ausgelesen werden. Diese geben Auskunft über Namen, Größe, Offset, Skalierung und Rotation der einzelnen Grafiken auf dem Spritesheet und sind auf Basis von XML-Dateien gespeichert oder binär kodiert. Innerhalb der Anwendung können die einzelnen Grafiken dann in einer einstellbaren Geschwindigkeit und Dauer nacheinander angezeigt werden. Diese Art von Animation ähnelt einem Daumenkino. In Unity3D können ebenfalls Spritesheets genutzt werden, jedoch werden die Property Lists für die Teilung und den Informationshintergrund nicht benötigt. Komfortable Möglichkeiten bieten die sogenannten Prefabs. Ein Prefab kann spezifisch definierte Informationen und Eigenschaften über ein oder mehrere Gameobjekte tragen und diese in einem Objekt abrufbereit verfügbar machen. In Happy Bird werden die Pipe-Hindernisse beispielsweise als Prefab definiert, mit dem Inhalt der Grafiken, der Positionen und der Kollisionsboxen. Für die Erstellung der Grafiken und Animationen wurde der freie Online-Spriteeditor Piskel verwendet. (Piskel 2015) Die genutzte Hintergrundmusik in Happy Bird, stammt von dem amerikanischen Komponisten und Musikproduzenten Kevin MacLeod. Das Musikstück mit dem Titel *"Monkeys Spinning Monkeys"* ist unter der Creative Commons lizensiert und frei verfügbar. (MacLeod 2014) In Tabelle 7.2 werden die Hauptkomponenten und ihre

Eigenschaften, sowie einige Framework-abhängige Dateien aufgelistet.

Dateiname	Dateityp	Objekttyp	Verwendung	Dateigröße (KB)
Red_Bird	PNG	Spritesheet	Spielfigur	1,54
bird	PLIST	Property List	Spielfigur	2,94
sprite_1	PNG	Sprite	Spielfigur	0,44
sprite_2	PNG	Sprite	Spielfigur	0,4
sprite_3	PNG	Sprite	Spielfigur	0,39
sprite_4	PNG	Sprite	Spielfigur	0,4
Sky	PNG	Sprite	Hintergrund	11,8
Ground	PNG	Sprite	Hindernis	3,05
Pipe	PNG	Sprite	Hindernis	1,99
StartButton	PNG	Sprite	Button	6,98
Marker Felt	TTF	Font	Schrift	26
Arial	TTF	Font	Schrift	761
BGMusic	MP3	Audio	Hintergrundmusik	4770

Tabelle 7.2: Liste von eingesetzten Spielelementen und deren Eigenschaften

7.6 Programmierung

Für die Entwicklung wurde C++ bei Cocos2D-X, C# bei Unity3D und Java bei libGDX für die Codebasis gewählt. Dies liegt darin begründet, dass diese objektorientierten, statisch typisierten Sprachen im Gegensatz zu den Skriptsprachen eine bessere Performance versprechen. Jedoch mit welcher Programmiersprache man die schnellsten und effizientesten Ergebnisse erzielen kann, steht zusätzlich in Verbindung mit der subjektiven Vorerfahrung. Während des Entwicklungsprozesses bestätigte sich ein weiterer wichtiger Faktor. Die Qualität der Dokumentation und Codebeispiele der Spieleframeworks.

Für libGDX finden sich viele Beispiele zu verschiedenen Bereichen innerhalb des offiziellen Wiki, die funktional sind und zudem mit Texten zur Erklärung hinterlegt sind. Auch die Suche nach individuelleren Lösungen wird befriedigend gesättigt. Die Entwicklungsgeschwindigkeit ist für jemanden ohne Vorerfahrung von daher durchaus annehmbar.

In diesem Punkt enttäuschte Cocos2D-X leider mehr als zuvor angenommen. Die offizielle Dokumentation schwächelte zur Zeit der Entwicklung an vielen grundlegenden

Stellen bezüglich der Aktualität. Dies muss nicht unbedingt tragisch sein, jedoch sind manche Einträge zu Standardlösungen nicht mehr ganz aktuell und lassen sich nicht mehr wie beschrieben anwenden. Das gleiche gilt für allgemeine Suchanfragen für Lösungen im Internet. Es ist als Anfänger von daher nicht trivial, für manche scheinbar einfache Problemstellungen herauszufinden, was die beste, empfohlene Vorgehensweise sein soll.

Im Gegensatz dazu, macht Unity3D es einem leichter. Einen Neuling mag dieser umfangreiche Editor zunächst mehr fordern, hingegen werden für jede Erfahrungsstufe Hilfestellungen geboten. Die Vielfalt an Unterstützung ist immens, fast immer mit praktischen Beispielen vertieft und gut sortiert. Wenn es passiert, dass überholte Methoden verwendet werden, hilft einem die unterstützte IDE oder die Dokumentation sofort weiter. Ein weiterer Vorteil und Zeitersparnis ist das direkte testen des Spiels innerhalb des Unity Editors. Das Testspiel konnte somit ohne Hindernisse umgesetzt werden.

8 Analyse der Test-Applikationen

8.1 Definition von Metriken

Die erstellten Applikationen sollen näher analysiert werden. Um dies möglichst objektiv und repräsentativ durchzuführen, müssen vergleichbare Metriken definiert werden. Diese Metriken werden anschließend für das Messprotokoll in Kapitel 8.3 verwendet. Gleichfalls muss für jede Metrik der Ablauf der Messung, sowie der Ausgangszustand der Testgeräte normiert werden.

Gesamtgröße einer Basisapplikation
Mit Basisapplikation ist die Anwendung gemeint, die bei Anlegung eines neuen Projekts erzeugt wird. Dies ist in der Regel eine App mit einer Szene / View und wird oft als *Hello World* deklariert. Dies soll eine Vorstellung über die Grundgröße einer minimalen Applikation geben und die Gesamtgröße des Spiels verständlicher darstellen. Für die Messung wird jeweils ein neues Projekt angelegt, daraus Applikationen erzeugt und die Dateigröße auf dem Testgerät dokumentiert. Ein niedriger Wert gilt als positiv, ein hoher als negativ.

Gesamtgröße des Spiels
Hiermit ist die Größe der Applikation insgesamt gemeint. Hier werden alle genutzten Ressourcen, Bibliotheken und sonstige Dateien eingerechnet. Für die Messung werden die Dateigrößen der erzeugten Applikationen auf den Testgeräten dokumentiert. Ein niedriger Wert gilt als positiv, ein hoher als negativ.

Größe der Ressourcen
Die Ressourcen für das Spiel sind die in Kapitel 7.5 deklarierten Komponenten. Hierzu zählen einerseits die Komponenten, die als Schnittmenge in allen Projekten vorkommen, sowie diejenigen die für ein jeweiliges Framework und Engine zusätzlich benötigt oder erzeugt wurden. Für die Messung wird die Größe des Ressourcenordners inner-

halb des jeweiligen Projekts dokumentiert. Ein niedriger Wert gilt als positiv, ein hoher als negativ.

Benötigter Arbeitsspeicher
Der genutzte Arbeitsspeicher, den das Spiel belegt soll ebenfalls dokumentiert werden. Dies kann auf den Testgeräten allerdings nur schwerlich in Echtzeit geschehen, da hierfür die Applikation verlassen werden muss und die Werte aus den Systemeinstellungen abgelesen werden, während die App nur im Hintergrund aktiv ist. Ein niedriger Wert gilt als positiv, ein hoher als negativ.

Ladegeschwindigkeit bei Neustart
Die Ladegeschwindigkeit ist die Zeit zwischen der Betätigung des Starticons und der Bereitschaft der Menüszene. Die Dauer der Anzeige des Splashscreens wurde gemäß der Grundeinstellungen der Projekte übernommen und bildet die Zeit ab, in der die benötigten Dateien geladen werden. Hierfür wird vorausgesetzt, dass die Applikation nicht im Hintergrund aktiv ist. Ein niedriger Wert gilt als positiv, ein hoher als negativ.

Frames pro Sekunde
Die Framerate wurde in das Spiel sichtbar in allen Szenen implementiert und innerhalb eines Zeitraums von 15 Minuten beobachtet. Hierbei wird der Durchschnittswert angegeben und eventuelle, auffällige Schwankungen zu bestimmten Zeitpunkten dokumentiert. Ein hoher und zugleich stabiler Wert gilt als positiv, ein niedriger oder stark schwankender als negativ.

Akkuverbrauch
Der Akkuverbrauch wird über einen Zeitraum von 15 Minuten gemessen. Ausgangswert ist ein frisch geladenes Testgerät mit 100% Ladezustand. Das Spiel wird über den Messzeitraum durchgehend genutzt. Dokumentiert wird anschließend der prozentuale Akkuverbrauch. Ein niedriger Wert gilt als positiv, ein hoher als negativ.

Mindestanforderung des Systems
Dies gibt die minimalen Anforderungen der mobilen Betriebssystemversion an. Die Unterstützung von älteren Versionen wird hierbei als vorteilhaft beurteilt, weil dadurch eine größere Zielgruppe erreicht werden kann.

Codezeilen
Die Menge der Codezeilen ist nur bedingt eine vergleichbare Metrik, da diese in Abhängigkeit des Programmierstils und der Erfahrung des Entwicklers, der Programmiersprache, der Architektur, des Einsatzes von Kommentaren und der Verwendung von leeren Zeilen abhängt. Jedoch soll hierdurch ein grober Eindruck über den benötigten Code vermittelt und dokumentiert werden. Für die Zählung werden Kommentare und leere Zeilen ignoriert. Ein niedriger Wert gilt als positiv, ein hoher als negativ.

8.2 Testgeräte und Voreinstellungen

Die Applikationen werden auf Smartphones mit Android und iOS System getestet. Auf eine Analyse des Spiels auf einem Windows Phone wird aus mehreren Gründen verzichtet. libGDX unterstützt die mobilen Windows Systeme derzeit nicht, die aktuellen Betriebssysteme sind nicht kompatibel zueinander, Windows Phone 8 wird nach und nach von Windows 10 abgelöst, weiterhin ist der Marktanteil laut der Analyse relativ gering und von daher vernachlässigbar. Die relevanten Daten der genutzten Testgeräte sind in Tabelle 8.1 aufgelistet. Falls zur Reproduktion ein anderes Testgerät verwendet werden soll, kann in der Tabellen 3.1 für Android und Tabelle 3.2 für iOS die Version des Betriebssystem für die Mindestanforderung verglichen werden. Um die zu dokumentierenden Messungen möglichst unverfälscht aufnehmen zu können, werden die Geräte vor der Durchführung der Messung einheitlich konfiguriert. Dies ist notwendig, um Daten wie Akkuverbrauch, Arbeitsspeicher oder Framerate möglichst wenig zu beeinflussen und die Ergebnisse reproduzierbarer zu machen.

- Alle Verbindungsoptionen (WLAN, mobiler Datenverkehr, Bluetooth, GPS, ...etc.) werden deaktiviert.
- Die Bildschirmhelligkeit wird auf 50% gestellt.
- Die Lautstärke für Medien und Anwendungen wird auf 50% gestellt und alle anderen deaktiviert.
- Alle weiteren Anwendungen und Prozesse werden weitestgehend beendet.
- Das Gerät wird in den Flug-Modus gesetzt, um die Verbindung zu einem Provider zu unterbinden.

- Das Spiel wird auf dem internen Speicher installiert.

Technische Daten	Android	iOS
Hersteller	Samsung	Apple
Modell	Galaxy S5+	iPhone 4
Artikelname	G901F	
Betriebssystem	Android 5.0.2	
System-on-a-Chip (SoC)	Qualcomm Snapdragon 805 APQ8084	Apple A4
Prozessor	Krait 450 2500 MHz 4 Kerne	ARM Cortex A8 800 MHz 1 Kern
Grafikprozessor	Qualcomm Adreno 420 600 MHz	PowerVR SGX 535 200 MHz
Arbeitsspeicher	2 GB	512 MB
Displaygröße	5,1 Zoll	3,5 Zoll
Auflösung	1080 x 1920 Pixel	640 x 960 Pixel
Akku	2800 mAh Li-Ion	1420 mAh Li-Ion

Tabelle 8.1: Datenblatt der Android und iOS Testgeräte

Bei den eigesetzten Smartphones handelt es sich um gebrauchte Geräte, was bedeutet, dass manche Leistungsmerkmale zu gleichen Modellen abweichen können. Da es bei Smartphones mit Android und iOS eine sehr große Vielfalt an verschiedensten Geräten, Versionen und Ausstattungen gibt, kann die fehlerfreie Funktion des Spiels nur theoretisch garantiert werden. Um die Darstellung in den unterschiedlichen Auflösungen zu testen, wurden zusätzlich die Simulatoren von Xcode und Android Studio zu Hilfe gezogen.

8.3 Messprotokoll

In diesem Teil der Arbeit werden die Messergebnisse zu den zuvor definierten Metriken, an den angegebenen Testgeräten aufgelistet. In Tabelle 8.2 sind die Ergebnisse der Android Version und in 8.3 die der iOS Version zu sehen.

Happy Bird - Android				
Metrik	**Wert**	**libGDX**	**Cocos2D**	**Unity3D**
Gesamtgröße eines leeren Projekts	MB			
Gesamtgröße des Spiels	MB			
Größe der Ressourcen	MB			
Benötigter Arbeitsspeicher	MB			
Ladegeschwindigkeit bei Neustart	Sekunden			
Frames pro Sekunde	Frames / Sekunde			
Akkuverbrauch	Prozent / Minute			
Mindestanforderung des Systems	Versionsnummer			
Codezeilen	Zeilen			

Tabelle 8.2: Messprotokoll des Spiels auf Android

Happy Bird - iOS				
Metrik	**Wert**	**libGDX**	**Cocos2D**	**Unity3D**
Gesamtgröße eines leeren Projekts	MB			
Gesamtgröße des Spiels	MB			
Größe der Ressourcen	MB			
Benötigter Arbeitsspeicher	MB			
Ladegeschwindigkeit bei Neustart	Sekunden			
Frames pro Sekunde	Frames / Sekunde			
Akkuverbrauch	Prozent / Minute			
Mindestanforderung des Systems	Versionsnummer			
Codezeilen	Zeilen			

Tabelle 8.3: Messprotokoll des Spiels auf iOS

9 Vergleich zur Benutzbarkeit

10 Fazit

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Android Develop Tools(2015)] Android Develop Tools(2015): *Android Studio Overview*, http://developer.android.com/tools/studio/index.html, letzter Zugriff: 24.11.2015

[Android Source - Codenames, Tags, and Build Numbers(2015)] Android Source - Codenames, Tags, and Build Numbers(2015): *Codenames, Tags, and Build Numbers in the history of Android*, https://source.android.com/source/build-numbers.html, letzter Zugriff: 24.11.2015

[App42(2016)] App42(2016): *App42 Cloud API*, http://api.shephertz.com/, letzter Zugriff: 01.02.2016

[AppGameKit(2015)] AppGameKit(2015): *AppGameKit - Platforms & Features*, http://www.appgamekit.com/platforms-and-features.php, letzter Zugriff: 18.02.2016

[Apple Developer(2015)a] Apple Developer(2015)a: *Swift - Overview*, https://developer.apple.com/swift/, letzter Zugriff: 28.12.2015

[Apple Developer(2016)b] Apple Developer(2016)b: *How the Program Works*, https://developer.apple.com/programs/how-it-works/, letzter Zugriff: 15.02.2016

[Bernardo Zamora(2015)] Bernardo Zamora(2015): *Blogs.Windows - Windows Store Trends - September 2015*, https://blogs.windows.com/buildingapps/2015/10/12/windows-store-trends-september-2015/, letzter Zugriff: 15.12.2015

[Boo(2015)] Boo(2015): *Boo - A scarily powerful language for .NET*, http://boo-language.github.io/, letzter Zugriff: 05.01.2016

[Brown(2015)] Brown, Ethan (Hrsg.): *Learning Javascript*, O'Reilly 2015

[Cocos2D-X(2015)a] Cocos2D-X(2015)a: *Cocos2D-X - Developers Manual*, http://www.cocos2d-x.org/wiki/Cocos2d-x, letzter Zugriff: 29.12.2015

[Cocos2D-X(2015)b] Cocos2D-X(2015)b: *Cocos2D-X - Relationships in Cocos2D Family*, http://www.cocos2d-x.org/wiki/Relationships_in_Cocos2d_Family, letzter Zugriff: 05.01.2016

[Cocos2D-X(2015)c] Cocos2D-X(2015)c: *Cocos2D-X - Dokumentation*, http://www.cocos2d-x.org/reference/native-cpp/V3.9/index.html, letzter Zugriff: 10.01.2016

[Distimo(2014)] Distimo(2014): *Anteil der im Google Play Store weltweit am häufigsten heruntergeladenen Apps nach Kategorien im Februar 2014. In Statista - Das Statistik-Portal.*, http://de.statista.com/statistik/daten/studie/321703/umfrage/beliebteste-app-kategorien-im-google-play-store-weltweit/, letzter Zugriff: 14.12.2015

[El-Kassas, Wafaa S. & Abdullah, Bassem A. & Yousef, Ahmed H. & Wahba, Ayman M.(2015)] El-Kassas, Wafaa S. & Abdullah, Bassem A. & Yousef, Ahmed H. & Wahba, Ayman M. : „Taxonomy of Cross-Platform Mobile Applications Development Approaches“, *Ain Shams Engineering Journal*, 2015

[Fran Berkman(2012)] Fran Berkman(2012): *Microsoft Mobile: From Pocket PC to Windows Phone 8*, http://mashable.com/2012/10/29/microsoft-mobile-history/#DYxZxZ7wTuqD, letzter Zugriff: 25.11.2015

[Golem(2015)] Golem(2015): *Microsoft demonstriert Android- und iOS-Apps unter Windows*, http://www.golem.de/news/windows-10-microsoft-demonstriert-android-und-ios-apps-unter-windows-1504-113812.html, letzter Zugriff: 25.11.2015

[Hölzl & Raed & Wirsing(2013)] Hölzl, Matthias & Raed, Allaithy & Wirsing, Martin (Hrsg.): *Java kompakt Eine Einführung in die Software-Entwicklung mit Java*, Springer 2013

[IDC(2015)] IDC(2015): *Prognose zu den Marktanteilen der Betriebssysteme am Absatz vom Smartphones weltweit in den Jahren 2015 und 2019. In Statista - Das Statistik-Portal.*, http://de.statista.com/statistik/daten/

studie/182363/umfrage/prognostizierte-marktanteile-bei-smartphone-betriebssystemen/, letzter Zugriff: 14.12.2015

[James Whitcomb Riley(1849–1916)] James Whitcomb Riley(1849–1916): *Ententest - Begriffsentstehung*, https://de.wikipedia.org/wiki/Ententest, letzter Zugriff: 05.01.2016

[JetBrains(2015)] JetBrains(2015): *JetBrains AppCode*, https://www.jetbrains.com/objc/, letzter Zugriff: 28.12.2015

[John Daintith(2004)] John Daintith(2004): *A Dictionary of Computing - native software*, http://www.encyclopedia.com/doc/1O11-nativesoftware.html, letzter Zugriff: 24.11.2015

[Junge(2015)] Junge, Prof. Dr. Jens: *Zur Begriffsklärung von Ludologie und Spielwissenschaft*, http://www.ludologie.de/neues-spiel/detailansicht/news/detail/News/zur-begriffsklaerung-von-ludologie-und-spielwissenschaft/, letzter Zugriff: 28.01.2016

[Kantar(2015)] Kantar(2015): *Marktanteile der mobilen Betriebssysteme am Absatz von Smartphones in ausgewählten Ländern von August bis Oktober 2015. In Statista - Das Statistik-Portal.*, http://de.statista.com/statistik/daten/studie/198453/umfrage/marktanteile-der-smartphone-betriebssysteme-am-absatz-in-ausgewaehlten-laendern/, letzter Zugriff: 14.12.2015

[libGDX(2013)a] libGDX(2013)a: *libGDX - Goals and Features*, https://libgdx.badlogicgames.com/features.html, letzter Zugriff: 29.12.2015

[libGDX(2013)b] libGDX(2013)b: *libGDX API*, https://libgdx.badlogicgames.com/nightlies/docs/api/, letzter Zugriff: 10.01.2016

[libGDX(2015)c] libGDX(2015)c: *libGDX - Setting up your Development Environment (Eclipse, Intellij IDEA, NetBeans)*, https://github.com/libgdx/libgdx/wiki/Setting-up-your-Development-Environment-(Eclipse,-Intellij-IDEA,-NetBeans), letzter Zugriff: 18.02.2016

[Lua(2015)] Lua(2015): *Lua - About*, http://www.lua.org/about.html, letzter Zugriff: 05.01.2016

[Lumberyard(2016)] Lumberyard(2016): *Lumberyard Beta - Details*, https://aws.amazon.com/de/lumberyard/details/, letzter Zugriff: 18.02.2016

[Mac Developer Library(2014)] Mac Developer Library(2014): *About Objective-C*, https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html/, letzter Zugriff: 28.12.2015

[MacinCloud(2015)] MacinCloud(2015): *MacinCloud*, http://www.macincloud.com/, letzter Zugriff: 28.12.2015

[MacLeod(2014)] MacLeod, Kevin: *Monkeys Spinning Monkeys*, Licensed under Creative Commons: By Attribution 3.0 http://creativecommons.org/licenses/by/3.0/ http://incompetech.com/music/royalty-free, letzter Zugriff: 28.01.2016

[Microsoft Developer(2015)] Microsoft Developer(2015): *Windows Software Development Kit (SDK) for Windows 10*, https://dev.windows.com/en-us/downloads/windows-10-sdk, letzter Zugriff: 28.12.2015

[Microsoft(2014)a] Microsoft(2014)a: *Microsoft - Microsoft und Nokia Geräte*, https://www.microsoft.com/de-de/nokia.aspx, letzter Zugriff: 25.11.2015

[Microsoft(2015)b] Microsoft(2015)b: *Microsoft - Windows 10 Features*, https://www.microsoft.com/de-de/windows/features, letzter Zugriff: 25.11.2015

[MonoGame(2016)] MonoGame(2016): *MonoGame - About*, http://www.monogame.net/about/, letzter Zugriff: 18.02.2016

[MSDN(2015)] MSDN(2015): *Microsoft-Emulator für Windows 10 Mobile*, https://msdn.microsoft.com/library/windows/apps/mt162269.aspx, letzter Zugriff: 28.12.2015

[Nextpeer(2016)] Nextpeer(2016): *Nextpeer*, https://www.nextpeer.com/, letzter Zugriff: 01.02.2016

[Open Handset Alliance - Alliance Members(2015)] Open Handset Alliance - Alliance Members(2015): *Members of the Open Handset Alliance*, http://www.openhandsetalliance.com/oha_members.html, letzter Zugriff: 24.11.2015

[Open Handset Alliance - Alliance Overview(2015)] Open Handset Alliance - Alliance Overview(2015): *Overview of the Open Handset Alliance*, http://www.openhandsetalliance.com/oha_overview.html, letzter Zugriff: 24.11.2015

[Open Handset Alliance - Android Overview(2015)] Open Handset Alliance - Android Overview(2015): *Overview of Android by the Open Handset Alliance*, http://www.openhandsetalliance.com/android_overview.html, letzter Zugriff: 24.11.2015

[Oracle - Java SE(2015)] Oracle - Java SE(2015): *Java SE Development Kit 8 Downloads*, http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html, letzter Zugriff: 24.11.2015

[Piskel(2015)] Piskel(2015): *Piskel - Pixel Art and Animated Sprites*, http://www.piskelapp.com/, letzter Zugriff: 28.01.2016

[Play Game Services(2015)] Play Game Services(2015): *Play Games Services*, https://developers.google.com/games/services/, letzter Zugriff: 01.02.2016

[PocketGamer.biz(2015)] PocketGamer.biz(2015): *Ranking der Top-20-Kategorien im App Store im Dezember 2015. In Statista - Das Statistik-Portal.*, http://de.statista.com/statistik/daten/studie/166976/umfrage/beliebteste-kategorien-im-app-store/, letzter Zugriff: 15.12.2015

[RoboVM(2016)] RoboVM(2016): *RoboVM - Create truly native iOS apps in Java*, https://robovm.com/, letzter Zugriff: 15.02.2016

[Salen & Zimmerman(2004)] Salen, Katie & Zimmerman, Eric: „Unit 1: Core Concepts: Defining Games“, (Hrsg.): *Rules of Play: Game Design Fundamentals*, The MIT Press 2004

[Schmidt(2015)] Schmidt, Julia: *heise Developer - Java 9: Pläne für Release im Herbst 2016*, http://www.heise.de/developer/meldung/Java-9-Plaene-fuer-Release-im-Herbst-2016-2635526.html, letzter Zugriff: 06.01.2016

[Skeet(2014)] Skeet, Jon (Hrsg.): *C# in Depth*, Manning 2014

[Statista(2015)] Statista(2015): *Anzahl der angebotenen Apps in den Top App-Stores im Mai 2015. In Statista - Das Statistik-Portal.*, http://de.statista.com/statistik/daten/studie/208599/umfrage/anzahl-der-apps-in-den-top-app-stores/, letzter Zugriff: 14.12.2015

[Sue Smith(2013)] Sue Smith(2013): *Android SDK Requirements*, http://code.tutsplus.com/tutorials/android-sdk-requirements--mobile-20086, letzter Zugriff: 24.11.2015

[t3n(2015)] t3n(2015): *Xcode 7: Apps auch ohne Entwickler-Account auf dem iPhone testen*, http://t3n.de/news/xcode-7-apps-ohne-615214/, letzter Zugriff: 28.12.2015

[techopedia(2015)] techopedia(2015): *Cross-Platform Development*, https://www.techopedia.com/definition/30026/cross-platform-development, letzter Zugriff: 21.12.2015

[the iphone wiki(2015)] the iphone wiki(2015): *iOS Firmwares*, https://www.theiphonewiki.com/wiki/Firmware, letzter Zugriff: 24.11.2015

[Unify Community Wiki(2014)] Unify Community Wiki(2014): *Unify Community Wiki - UnityScript versus JavaScript*, http://wiki.unity3d.com/index.php/UnityScript_versus_JavaScript, letzter Zugriff: 05.01.2016

[Unity3D(2014)d] Unity3D(2014)d: *Unity3D - Documentation, Unity scripting languages and you*, http://blogs.unity3d.com/2014/09/03/documentation-unity-scripting-languages-and-you/, letzter Zugriff: 05.01.2016

[Unity3D(2015)a] Unity3D(2015)a: *Unity3D - Public Relations*, https://unity3d.com/public-relations, letzter Zugriff: 29.12.2015

[Unity3D(2015)b] Unity3D(2015)b: *Unity3D Documentation - Managed Plugins*, http://docs.unity3d.com/Manual/UsingDLL.html, letzter Zugriff: 05.01.2016

[Unity3D(2015)c] Unity3D(2015)c: *Unity3D - Get Unity*, http://unity3d.com/get-unity, letzter Zugriff: 05.01.2016

[Unity3D(2015)d] Unity3D(2015)d: *Unity3D - Scripting API*, http://docs.unity3d.com/ScriptReference/index.html, letzter Zugriff: 10.01.2016

[Unity3D(2016)e] Unity3D(2016)e: *Unity3D - Tutorials: Scripting*, https://unity3d.com/learn/tutorials/topics/scripting, letzter Zugriff: 10.02.2016

[Vehse(2014)] Vehse, Benjamin: „Plattformabhängige und –unabhängige Entwicklung mobiler Anwendungen am Beispiel von Geo-Wikipedia-App“, *Bachelor-Thesis*, 2014

[Visual Studio(2016)] Visual Studio(2016): *Visual Studio - Tools für alle Entwickler und alle Apps*, https://www.visualstudio.com/de-de/dn469161, letzter Zugriff: 15.02.2016

[Wikipedia - Flappy Bird(2015)] Wikipedia - Flappy Bird(2015): *Flappy Bird*, https://de.wikipedia.org/wiki/Flappy_Bird, letzter Zugriff: 28.01.2016

[Wikipedia - Liste von Android-Versionen(2015)] Wikipedia - Liste von Android-Versionen(2015): *Übersicht von allen Android Versionen mit Veröffentlichungsdatum*, https://de.wikipedia.org/wiki/Liste_von_Android-Versionen, letzter Zugriff: 24.11.2015

[Wikipedia - Windows 10(2015)] Wikipedia - Windows 10(2015): *Microsoft Windows 10 Mobile*, https://de.wikipedia.org/wiki/Microsoft_Windows_10_Mobile, letzter Zugriff: 28.12.2015

[Wikipedia - Windows Phone 7(2015)] Wikipedia - Windows Phone 7(2015): *Microsoft Windows Phone 7*, https://de.wikipedia.org/wiki/Microsoft_Windows_Phone_7, letzter Zugriff: 28.12.2015

[Wikipedia - Windows Phone 8(2015)] Wikipedia - Windows Phone 8(2015): *Microsoft Windows Phone 8*, https://de.wikipedia.org/wiki/Microsoft_Windows_Phone_8, letzter Zugriff: 28.12.2015

[XMLVM(2011)] XMLVM(2011): *XMLVM - Overview: Toolchain*, http://xmlvm.org/toolchain/, letzter Zugriff: 25.12.2015

[YoYo Games(2013)] YoYo Games(2013): *Requirements for Windows Phone development*, http://help.yoyogames.com/entries/23355146-Requirements-for-Windows-Phone-development, letzter Zugriff: 28.12.2015

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.

Ort, Datum Sebastian Bohn