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

Hamburg, 29. 02. 2016

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.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Digitale Spiele sind ein Zeitvertreib, der spätestens seit der Verbreitung von Smartphones in der modernen Bevölkerung bei allen Altersgruppen und Gesellschaftsschichten Einzug gehalten hat. Mobile Betriebssysteme bieten somit Plattformen, die eine breite Nutzergruppe erreichbar macht. Die Uneinheitlichkeit dieser Systeme erweckt das entwicklerseitige Bedürfnis nach Softwareinstrumenten, die plattformübergreifende Entwicklung ermöglicht. Der Einsatz dieser speziellen Entwicklungswerkzeuge rechtfertigt sich durch höhere Entwicklungsgeschwindigkeit und geringere Entwicklungskosten.

Gerade im Bereich der Spieleentwicklung besteht ein vielfältiges Angebot von plattformübergreifenden Frameworks und Engines, was die Wahl nach einer passenden Software nicht trivial erscheinen lässt. Demnach stellt sich die Frage, von welchen Faktoren eine geeignete Auswahl abhängt:

- Welche mobile Plattformen besitzen das höchste, wirtschaftliche Potential und bedienen die größte Zielgruppe?
- Welche Entwicklungswerkzeuge unterstützen diese erfolgreichen Plattformen gleichzeitig?
- Womit können die inhaltlichen und funktionalen Anforderungen am effektivsten erfüllt werden?

Diese grundlegenden Fragestellungen geben Anlass, ausgewählte Spieleframeworks anhand ihrer theoretischen Möglichkeiten zu analysieren und durch die Entwicklung einer Beispielapplikation zu evaluieren. Um ein besseres Verständnis von plattformübergreifender Entwicklung zu gewinnen, soll weiterhin die Arbeitsweise dieses Prin-

zips näher betrachtet und mögliche Unterschiede ausgemacht werden. Durch die Bearbeitung soll ermittelt werden, welche Software am besten zu den gestellten Anforderungen passt, welche Schwächen ausfindig gemacht werden können und ob es eine eventuelle Allzwecklösung gibt.

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 der Messresultate und des Entwicklungsprozesses sowie mit einer Empfehlung zur Benutzbarkeit der Spieleframeworks.

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 und iOS derzeit die beiden relevantesten Systeme auf dem mobilen Markt sind. Mit deutlich geringeren Nutzerzahlen steht Windows Phone an dritter Stelle.

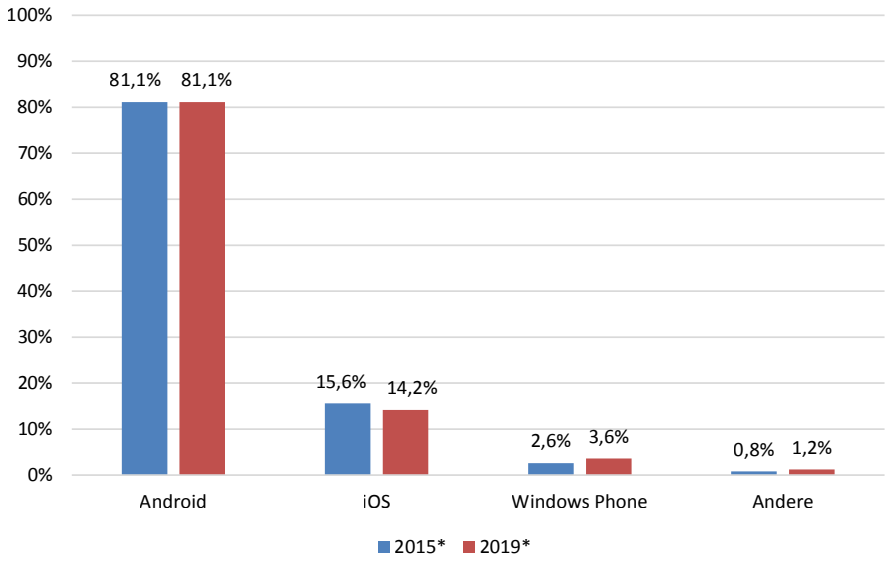

Abbildung 2.1: Prognose zu den Marktanteilen der Betriebssysteme am weltweiten Absatz von Smartphones, in den Jahren 2015 und 2019
(IDC 2015)

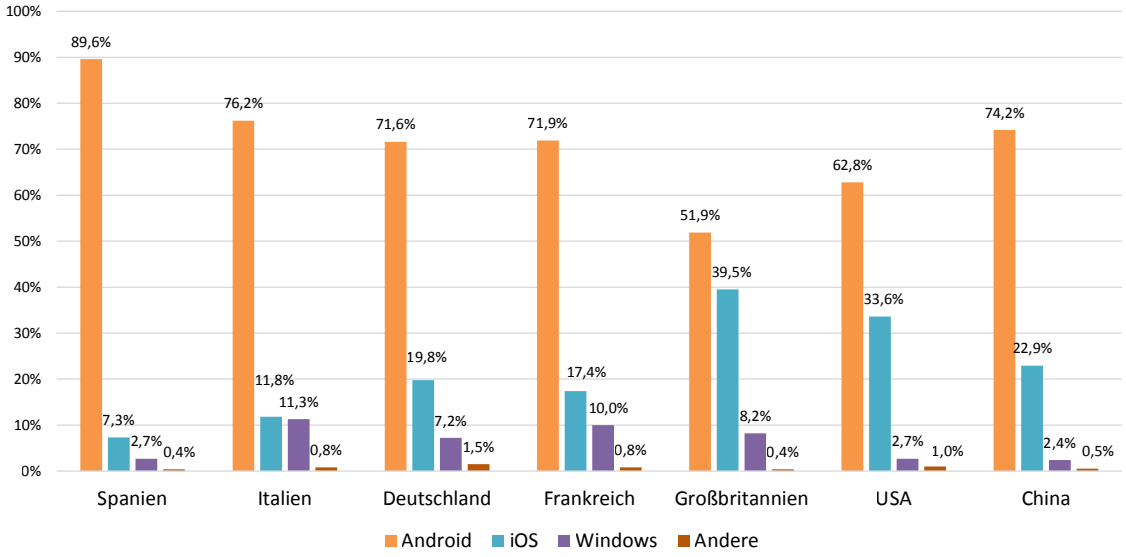

Abbildung 2.2: Marktanteile der mobilen Betriebssysteme am Absatz von Smartphones in ausgewählten Ländern, von August bis Oktober 2015
(Kantar 2015)

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.

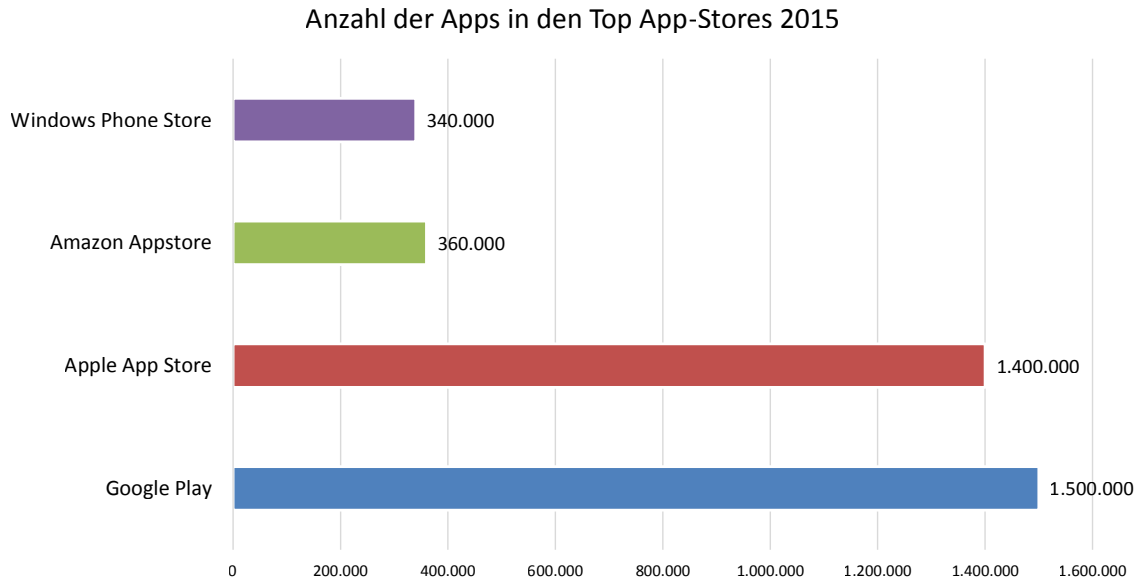

Abbildung 2.3: Anzahl der angebotenen Apps in den Top App-Stores im Mai 2015 (Statista 2015)

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% (Zamora 2015).

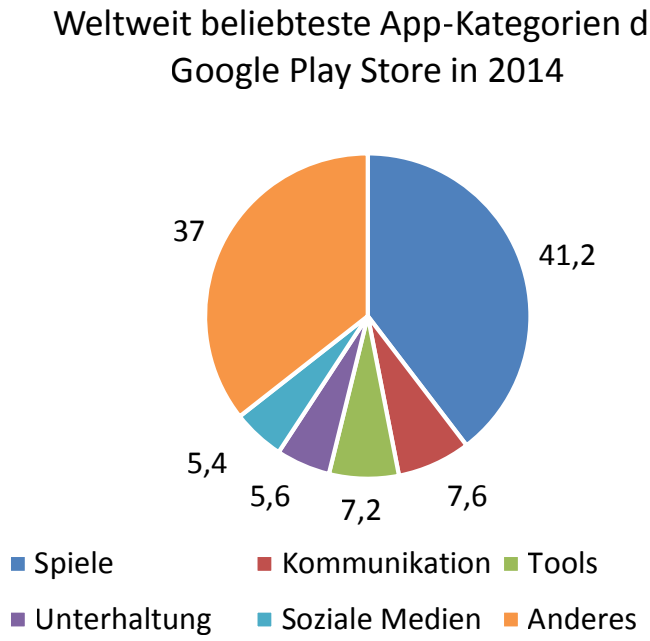

Abbildung 2.4: Aufteilung der weltweit am häufigsten heruntergeladenen Kategorien im Google Play Store, im Februar 2014 (Werte in Prozent)
(Distimo 2014)

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.

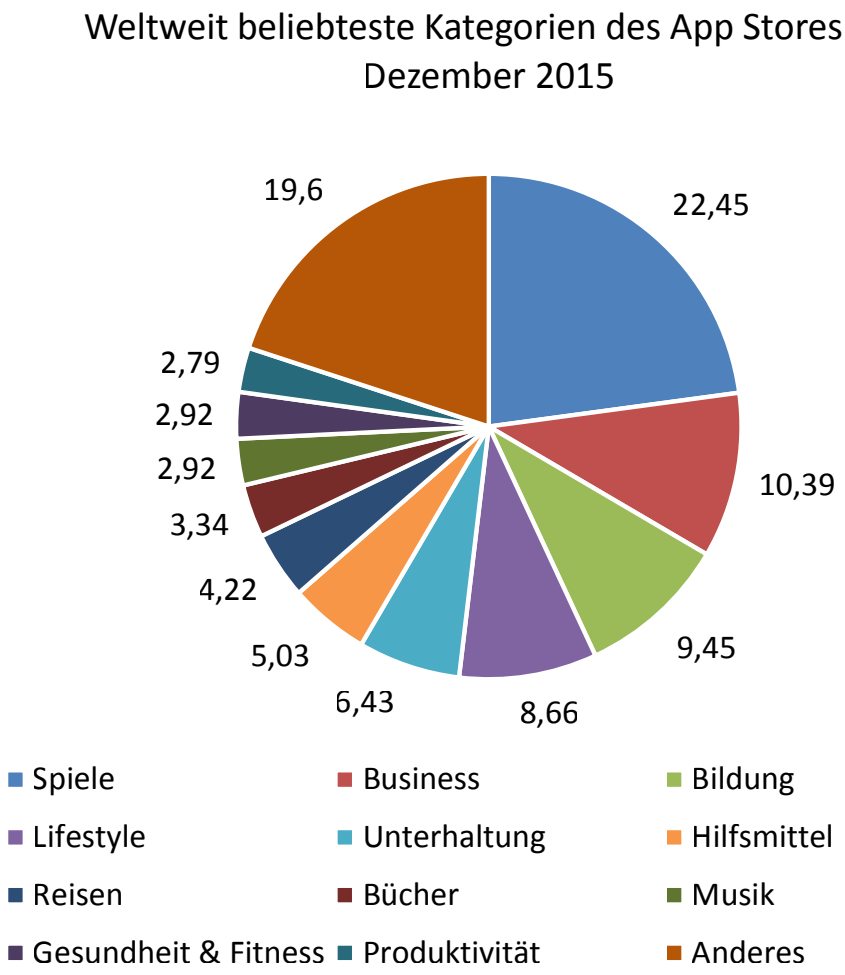

Abbildung 2.5: Downloadzahlen der Top-Kategorien des Apple App Stores, für Dezember 2015 (Werte in Prozent)
(PocketGamer.biz 2015)

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 (Wikipedia - Apple iOS 2015). 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 (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 (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 (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 (Visual Studio 2016).

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 (AndEngine 2013).

Cocos2D-Android
Eine Implementierung von Cocos2D für Android. Die Weiterentwicklung wurde allerdings eingestellt und die letzte Änderung erfolgte im Januar 2010 (Cocos2D-Android 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 (jPCT-AE 2015).

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 (SpriteKit 2015).

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 (Cocos2D-iPhone 2015).

Sparrow
Eine aktuelle Open Source Game Engine für iOS Spiele, die ausschließlich Objective-C unterstützt (Sparrow 2015).

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 (XNA 2015). 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).

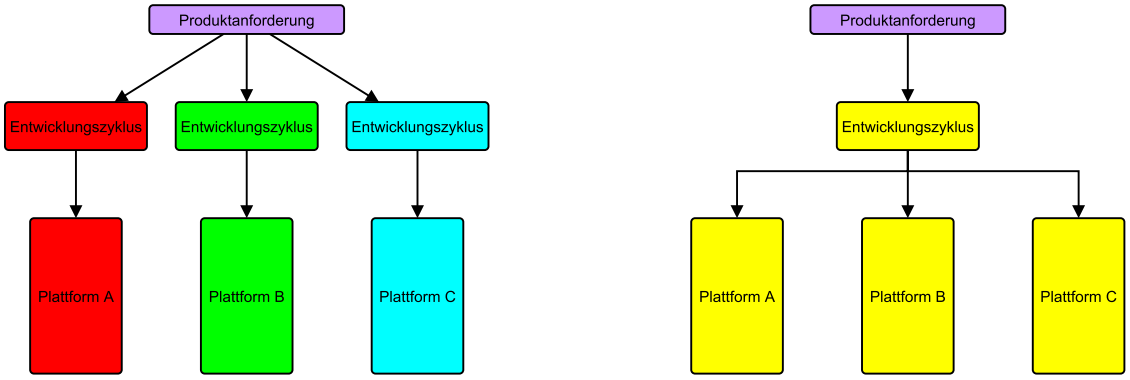

Abbildung 4.1: Traditionelle und plattformübergreifende Entwicklungsmodelle

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).

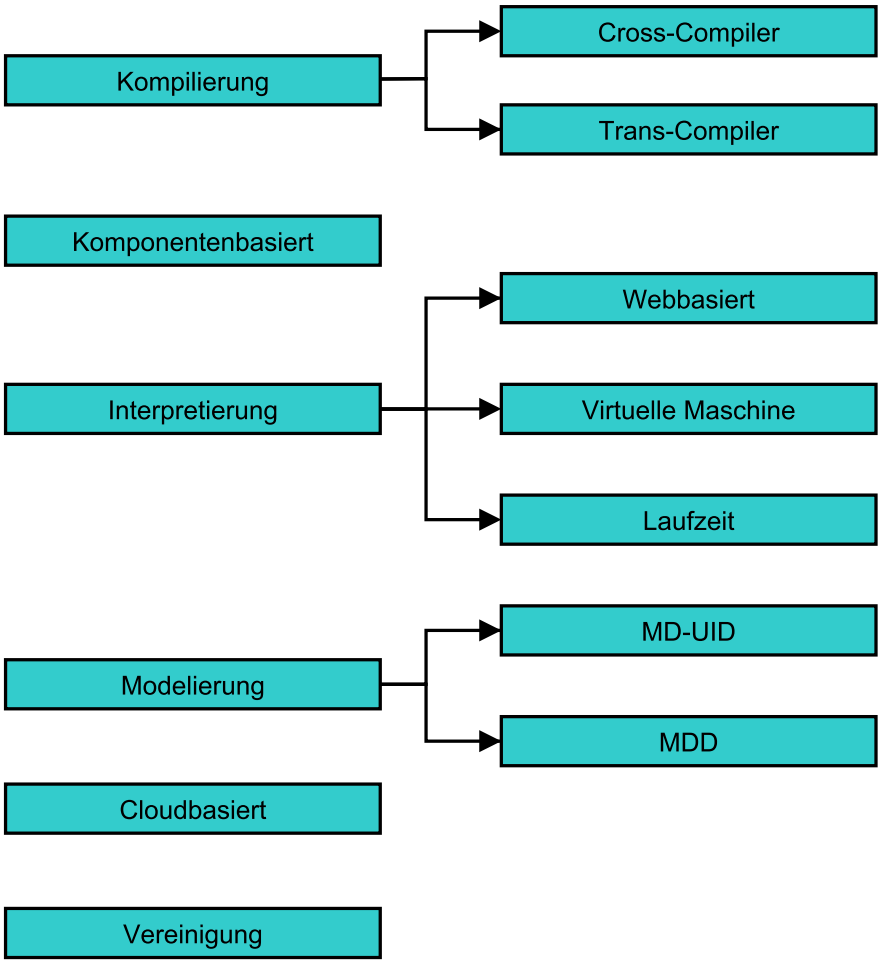

Abbildung 4.2: Haupt- und Unteransätze zur Entwicklung von mobilen, plattformübergreifenden Anwendungen

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.

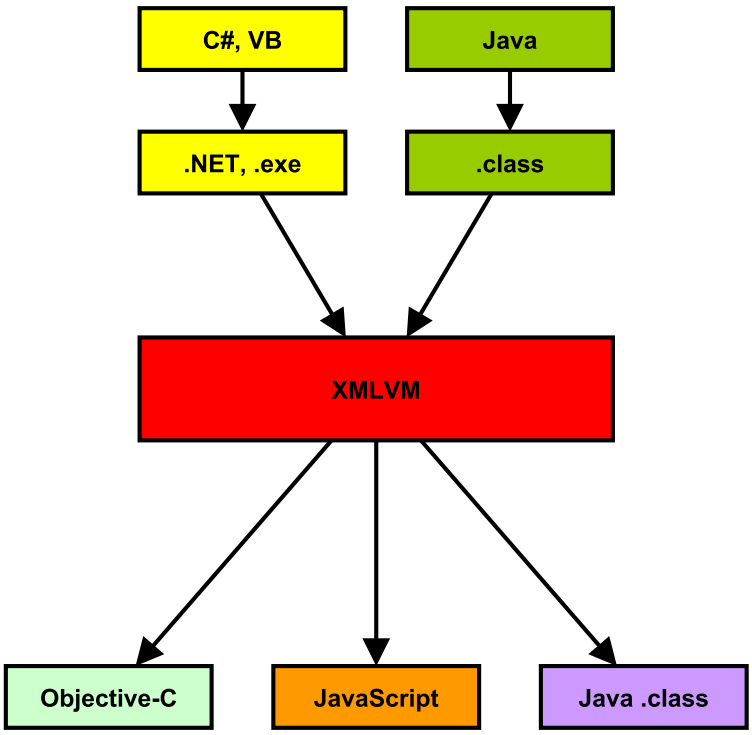

Abbildung 4.3: XMLVM Prozess mit Java oder .NET Quellcode (XMLVM 2011)

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.

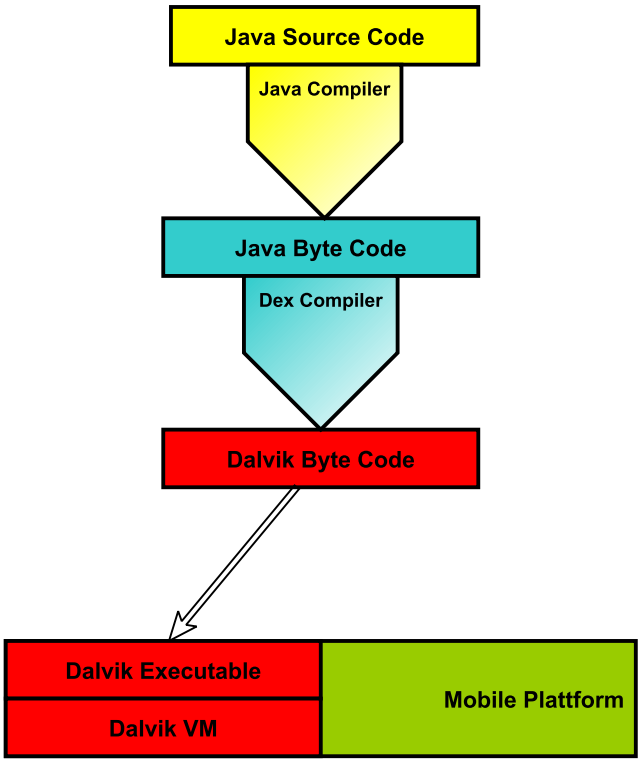

Abbildung 4.4: Ablauf des Dalvik VM Interpreter

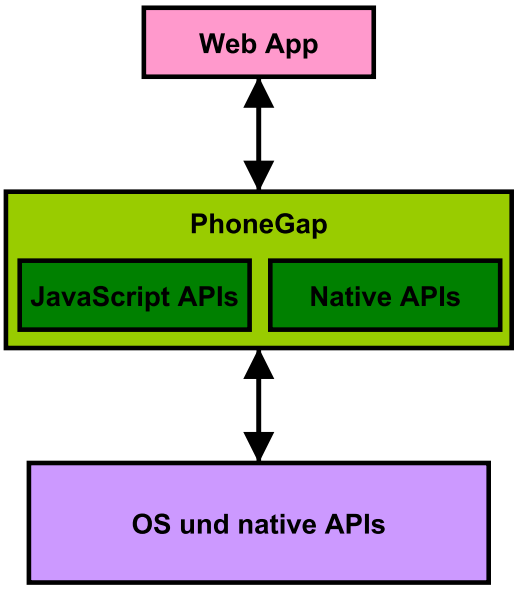

Abbildung 4.5: Vereinfachter Ablauf des PhoneGap Interpreters von Adobe

Die **Laufzeit** ist eine Schicht und Ausführungsphase, welche die mobile App auf der nativen Plattform lauffähig macht. Bei diesem Ansatz wird der Quellcode in Bytecode umgewandelt und dann zur Laufzeit ebenfalls von einer virtuellen Maschine ausgeführt. In Abbildung 4.6 wird die Verarbeitung von Appcelerators Titanium-Interpreters dargestellt.

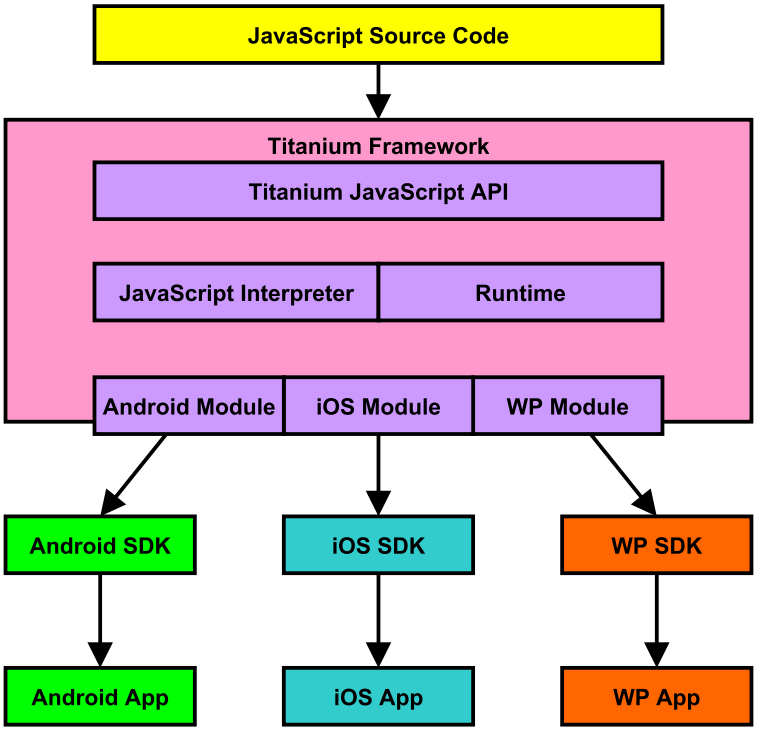

Abbildung 4.6: Ablauf des Titanium Interpreters von Appcelerator

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.

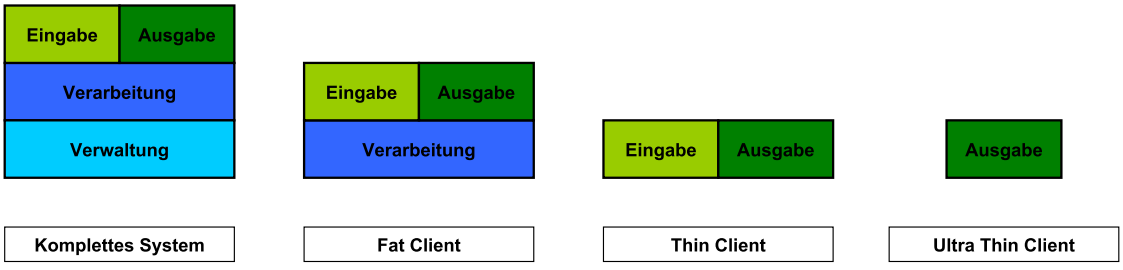

Abbildung 4.7: Aufgaben von Client-Anwendungen

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.

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.

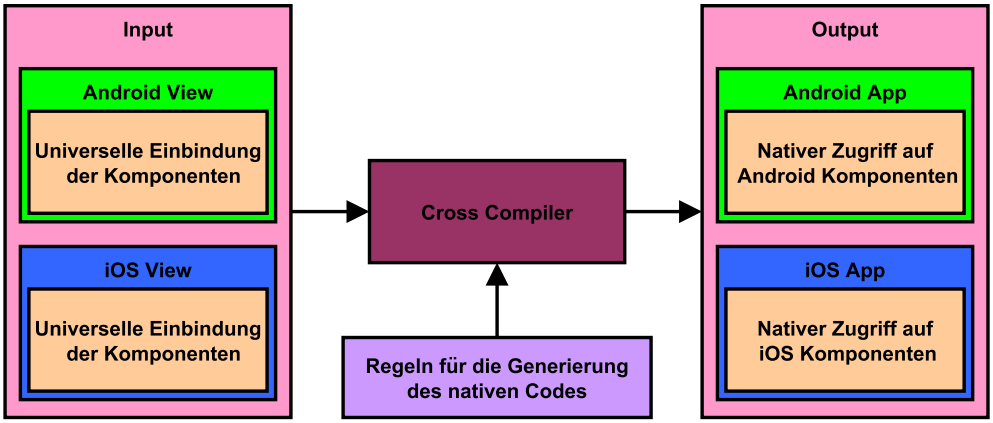

Abbildung 4.8: Funktion eines komponentenbasierten Mergeansatz

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.

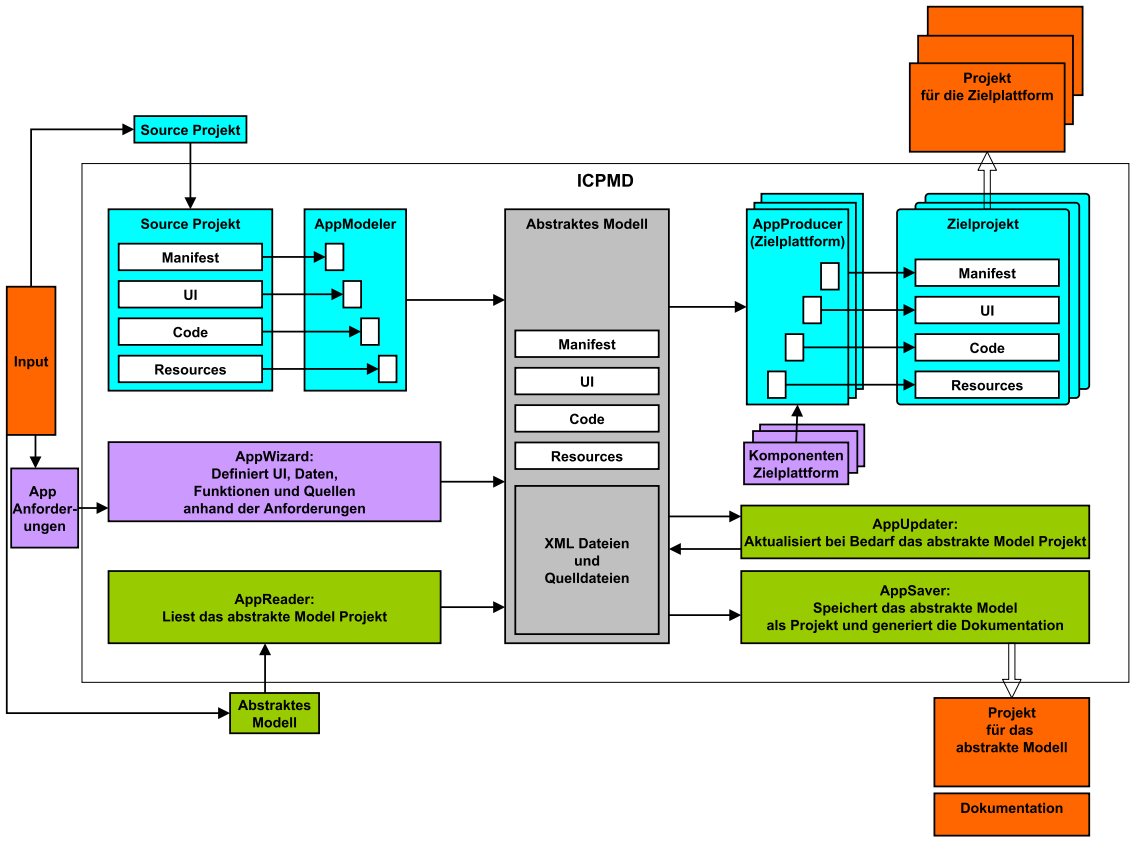

Abbildung 4.9: Drei Szenarien bei ICPMD

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 Bachelorthesis 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 Entwicklungswerkzeuge

Das Angebot an Werkzeugen für die plattformübergreifende Spieleentwicklung ist vielfältig und es bestehen Unterschiede in der Funktionalität sowie den Möglichkeiten. Zum Zeitpunkt der Bearbeitung dieser Arbeit wurden bei Wikipedia ca. 140 verschiedene, plattformübergreifende Werkzeuge zur Spieleentwicklung aufgelistet (Wikipedia - List of game engines 2016). Da ein Vergleich von allen Produkten solcher Art den Umfang dieser Arbeit drastisch überschreiten würde, werden zuerst grundlegende Anforderungen definiert, um eine spezifischere Auswahl treffen zu können.

Das Framework soll folgende Anforderungen erfüllen:

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

Die definierten Anforderungen sind nicht willkürlich gewählt und folgendermaßen gerechtfertigt:

Wie in der Marktanalyse der mobilen Systeme aus Kapitel 2.1.1 ermittelt wurde, sind Android und iOS derzeit die beiden entscheidenden mobilen Plattformen und sind daher für die spätere Beispielapplikation maßgebliche Voraussetzungen. Auf einen Vergleich der Applikation auf einem Windows Phone wird aus mehreren Gründen verzichtet. Der Marktanteil laut der Analyse ist verhältnismäßig gering und von daher vernachlässigbar. Weiterhin existieren derzeit mehr Geräte mit Windows Phone 8, das aber in naher Zukunft von Windows 10 abgelöst wird. Die Aktualität des Spieleframeworks ist wichtig, da die Entwicklungswerkzeuge den derzeitigen, technischen Stand der Zielsysteme unterstützen und sich regelmäßig weiterentwickeln sollen. Dazu gehört eine zur Version passende, aktuelle Dokumentation. Eine aktive und lebendige Community von Nutzern und Entwicklern ist wichtig, um eventuelle Problemstellungen leichter lösen zu können sowie potentielle Bugs ausfindig zu machen und diese gegebenenfalls zu melden. Diese Faktoren sind wichtig, um für die nähere Zukunft eine kalkulierbare Sicherheit der weiteren Existenz der Entwicklungssoftware zu gewährleisten. Die Möglichkeit, eine statisch typisierte Programmiersprache zu nutzen, wird vorausgesetzt, da diese, im Vergleich zu dynamischen Skriptsprachen, bei qualifiziertem Umgang in der Regel bessere Performanceleistungen bieten. Auch eventuelle Fehler werden dadurch schon bei der Kompilierung aufgedeckt und nicht erst zur Laufzeit. Durch diese Anforderung wird partiell vorausgesetzt, dass der plattformübergreifende Ansatz die Kompilierung enthält. Daraus folgt, dass Frameworks ausgeschlossen werden, die ausschließlich mit webbasierten Techniken, wie HTML, CSS und JavaScript, arbeiten. Viele aktuelle Smartphones sind von ihrer technischen Ausstattung befähigt, 3D Spiele zu unterstützen und darzustellen. Die Konzeption der Beispielapplikation soll sich deshalb die Wahl zwischen 2D und 3D vorbehalten können. Ein kostenfreier Bezug und uneingeschränkte Nutzung sichert eine größere Entwicklercommunity und vergrößert die Menge des geteilten Wissens. Weiterhin gibt dies die Möglichkeit, barrierefrei und ohne Zeitdruck mit einem Werkzeug zu arbeiten und das Resultat bei Wunsch zu veröffentlichen.

Trotz der geforderten Übereinstimmungen ist es wünschenswert, dass für den Basisquellcode der Projekte nicht dieselben Programmiersprachen genutzt werden, um den Vergleich abwechslungsreicher und kontrastvoller zu gestalten. Anhand dieser Anforderungen werden passende Werkzeuge ausgewählt.

5.2 Gamespezifische Frameworks und Engines

Aufgrund der zuvor gestellten Anforderungen wurde die Auswahl der Vergleichswerkzeuge bedeutend reduziert. Die Wahl fiel auf die beiden Frameworks libGDX und Cocos2D-X sowie die Engine Unity3D. Diese werden in den folgenden Unterkapiteln kurz 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. libGDX ist Open Source und bringt eine Menge an grundlegenden Bibliotheken für die Erstellung von 2D und 3D Spielen mit. Dabei wird allerdings nicht jede Eventualität abgedeckt, denn für spezielle Fälle sollen dem Projekt bei Bedarf entsprechend spezialisierte Module hinzugefügt werden. Dies kann bei der Erstellung eines neuen Projekts oder nachträglich geschehen. Dieses modulare Konzept soll garantieren, dass nur die Funktionalität integriert ist, die auch wirklich benötigt wird. Dafür arbeitet libGDX verstärkt mit anderen quelloffenen Frameworks und Bibliotheken zusammen. Die zur Verfügung stehende Programmiersprache ist Java, welche die Erstellung von Android Anwendungen problemlos möglich macht. Für die Generierung von iOS Apps wird auf die Fähigkeiten des Kompilierers RoboVM zurückgegriffen (libGDX 2013).

5.2.2 Cocos2D-X

Cocos2D-X ist ein Open Source Framework und unter dem MIT (Massachusetts Institute of Technology) lizensiert. Auch hier bestehen zahlreiche Schnittstellen und Kooperationen mit externen Anbietern spezieller Frameworks, wovon einige bereits fester Bestandteil der Standardbibliothek sind. Entwickler haben für die Codebasis die Wahl zwischen C++, Lua und JavaScript. Cocos2D-X ist der plattformübergreifende Ableger der vielgesichtigen Cocos Reihe. Trotz der Namensgebung können auch 3D Anwendungen erstellt werden. Das optionale Programm Cocos Studio unterstützt bei der Gestaltung von Spielszenen. Damit kann die grafische Ebene mit Texturen, Sprites und Menüobjekten angeordnet und daraufhin für die IDE exportiert werden, um die zugehörige Spiellogik zu implementieren. Projekte können aus der Kommandozeile oder einer eigenen Desktopanwendung namens Cocos heraus erstellt werden.

Cocos2D-X findet laut eigenen Angaben auch bei großen Spieleentwicklern wie Konami 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 große Anzahl an Schnittstellen für die Nutzung externer Datenobjekte und zusätzlichen Services. Des Weiteren wird durch diese Engine die größte Anzahl an aktuellen Zielplattformen unterstützt. Unity3D findet Verwendung bei Entwicklern verschiedenster Interessengebiete, wie Hobby- und Indieentwickler sowie bei professionellen Studios. Innerhalb des Editors der Engine können Spieleelemente einer Szene direkt hinzugefügt, transformiert, mit Komponenten erweitert und unmittelbar getestet werden. Die Spiellogik kann durch C#, UnityScript oder Boo Skripte in einer externen IDE umgesetzt, einem Gameobjekt als Komponente hinzugefügt werden oder als eigenständige Klassen operieren. Werte von globalen Objekten und Variablen sind im Editor durch ein entsprechend erzeugtes Formfeld veränderbar. Zudem ist es möglich, Objekte zur Laufzeit im Editor anzupassen und hinzuzufügen. 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, sollen an dieser Stelle kurz erwähnt werden.

MonoGame
Eine Open Source Implementierung von Microsofts XNA Framework, die 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 können 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 auf der CryEngine basiert (Lumberyard 2016).

6 Analyse der Frameworks

6.1 Zielplattformen

Die Anzahl der unterstützten Zielplattformen in den Kategorien Mobil, Destop und Web unterscheidet sich bei den drei Werkzeugen nur geringfügig. In Tabelle 6.1 wird gezeigt, dass Cocos2D-x und Unity3D dieselben mobilen Systeme unterstützen. In dem Bereich unterscheidet sich libGDX von den anderen Beiden. Die Möglichkeit, für das Blackberry OS Spiele zu entwickeln, ist in diesem Vergleich einzig mit libGDX möglich. Dafür muss bei der Nutzung von libGDX auf Windows Phone verzichtet werden, was aber zukünftig durch Windows 10 Universal Applikationen wieder abgedeckt wird. Alle drei Werkzeuge bieten aber die vorausgesetzte Möglichkeit, gleichzeitig die Systeme von iOS und Android zu bedienen (Unity3D 2015, Cocos2D-X 2015, libGDX 2013).

	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 Spieleframeworks (Unity3D 2015, Cocos2D-X 2015, libGDX 2013)

6.2 Programmiersprachen

Bei der Wahl der Programmiersprachen unterscheiden sich die gewählten Spieleframeworks komplett voneinander. Entwicklern mit unterschiedlichen Kenntnissen, kann durch die Wahl einer vertrauten Sprache somit grundsätzlich ein leichterer Einstieg in die Spieleentwicklung geboten werden. Auch wenn der Anforderung nach hauptsächlich die objektorientierten, statisch typisierten Sprachen für diese Arbeit entscheidend sind, werden zugunsten der Vollständigkeit alle verfügbaren Sprachen erwähnt.

6.2.1 libGDX

libGDX nutzt einzig und allein Java für die Entwicklung, was für reine Android Spiele ein großer Vorteil ist, denn dadurch müssen bei der Kompilierung kaum Kompromisse eingegangen werden. Java ist eine weitverbreitete, objektorientierte Programmiersprache, die erstmalig 1995 von dem Unternehmen Sun Microsystem vorgestellt wurde. 2010 übernahm das Unternehmen Oracle Sun Microsystem und damit auch 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 (Hölzl & Raed & Wirsing 2013). Das derzeit aktuelle JDK 8 soll 2016 auf Version 9 aufsteigen (Schmidt 2015). Die Sprache gilt als plattformübergreifend, da Programme in einer virtuellen Maschine, der JVM (Java Virtual Machine), ausgeführt werden. Diese virtuelle Maschine führt den bei der Kompilierung erstellten Bytecode aus und prüft das Programm vorweg auf Laufzeitfehler. Die Syntax wurde von C++ und C beeinflusst und besitzt eine statische Typisierung (Hölzl & Raed & Wirsing 2013).

6.2.2 Cocos2D-X

Mit Coco2D-X hat man die Wahl zwischen C++, Lua und JavaScript, welche für fast alle unterstützten Plattformen nutzbar sind. Einzige Einschränkung liegt bei JavaScript, denn diese lässt sich bei Cocos2D-X nicht mit Windows Phone verknüpfen (Cocos2D-X 2015).

Lua ist eine von der Syntax simpel gehaltene Skriptsprache, mit objektorientierten

Eigenschaften. Die geschriebenen Skripte werden durch einen Interpreter in Bytecode übersetzt und sind 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++ hat man die Möglichkeit, äußerst performante und portable Software zu schreiben, da diese direkt von C abgeleitet und kompatibel ist. C++ hat eine komplexere Syntax im Vergleich zu anderen High-Level Sprachen, zählt aber zu den schnellsten Programmiersprachen der Welt. Seit 1998 ist C++ ISO standardisiert und mit der aktuellen Version 14 verfügbar. Es werden zudem die prozeduralen, generischen und objektorientierten Programmierparadigmen unterstützt (C++ 2016).

Für Entwickler, die Erfahrungen im Webbereich besitzen, bietet die Verwendung von JavaScript, im Gegensatz zu C++, einen leichteren 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 in HTML Dokumenten clientseitige Logik zu ermöglichen. Mit der Programmiersprache Java können hingegen auch eigenständige Anwendungen aufgebaut werden. JavaScript folgt den Spezifikationen der privaten Normungsorganisation ECMA (European Computer Manufacturers Association), die Standardisierungen von Informationstechnologien entwickelt und die Richtigkeit deren Verwendung fördert, als auch eine frei zugängliche Veröffentlichung dazu liefert. Der standardisierte Kern von JavaScript wird als ECMAScript (ECMA 262) bezeichnet und beschreibt die Eigenschaften mit einer dynamischen Typisierung, die objektorientiert, aber klassenlose sind. Objekte basieren in JavaScript auf sogenannten Prototypen, die als Funktionen geschrieben werden. Nach der Instanziierung 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 verschiedene Bibliotheken und Frameworks. Mit Node.js können dann beispielsweise serverseitige Netzwerkanwendungen betrieben werden. Durch Angular.js werden bekannte Entwurfsmuster aus der Softwareentwicklung zugänglich, die der Skriptsprache normalerweise verwehrt wären. ECMAScript befindet sich seit 2015 in Version 6 (Brown 2015).

6.2.3 Unity3D

In Unity3D finden die Sprachen C#, UnityScript und Boo nutzerseitige Verwendung. Diese Sprachen entspringen Microsofts .NET Framework. Das bietet den Vorteil auch weitere .NET Sprachen benutzen zu können. Voraussetzung dafür ist, dass diese ihre Skripte in das DLL (Dynamically Linked Library) Format kompilieren können. Diese DLL Dateien können dann einem Unity Projekt hinzugefügt und verwendet werden (Unity3D 2015). Für die Kompilierung nutzt Unity3D primär die Open Source Alternative zu .NET namens Mono.

Boo ist eine von Python beeinflusste Sprache für .NET als auch für Mono und verzichtet auf Klammern und Semikolons in der Syntax. Die Typisierung ist generell statisch, kann aber trotzdem dynamische *Duck-Typing* Eigenschaften nutzen. Beim Duck-Typing werden Objekte nicht durch ihre Klasse typisiert, sondern durch die Art der vorhandenen Attribute und Methoden. Daher findet die Typisierung erst zur Laufzeit durch den Interpreter statt (Boo 2015). Der Name Duck-Typing entspringt dem sogenannten *Ententest* zur allgemeinen Typisierung. Dieser Test wurde von einem Gedicht von James Whitcomb Riley inspiriert.

"*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.*" (Whitcomb Riley 1849–1916)

UnityScript wird innerhalb der eigenen Community, oft aber auch von dem Unternehmen selbst, fälschlicherweise mit JavaScript gleichgesetzt, so, 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 aber, wie zuvor beschrieben, statt Klassen sogenannte Prototypes verwendet werden. 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 das Finden genauer Spezifikationen erschwert (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 Program-

me, die in Verbindung mit dem .NET Framework realisiert wurden. Die Syntax der objektorientierten Sprache weist große Ähnlichkeiten mit Java auf. Die Typisierung ist grundsätzlich statisch, wobei optional auch dynamische Typen genutzt werden können. Mit C# können in Unity3D auch typische Softwarearchitekturen realisiert werden (Skeet 2014). Laut einer eigenen Statistik von Unity3D sind etwa 80% der Projekte in C# geschrieben (Unity3D 2014).

6.3 Entwicklungsumgebungen

Für den Entwicklungsprozess besteht bei jedem Framework die Möglichkeit, zwischen mehreren IDEs wählen zu können. Praktisch kann jeder textbasierte Editor für die Programmierung der Anwendung verwendet werden. Aber um den Komfort von der passenden Hervorhebung der Syntax und die direkte Kompilierungsmöglichkeit des Basiscodes zu erhalten, empfiehlt es sich, auf die unterstützten Entwicklungsumgebungen zurückzugreifen. Diese sind bei den drei vorgestellten Spieleframeworks teilweise abhängig von dem verwendeten Betriebssystem des Entwicklers.

6.3.1 Systembedingte Einschränkungen

Für jedwede Erzeugung von iOS Applikationen gilt, dass die Apple IDE Xcode unumgänglich ist. Diese ist 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 der Erwerb einer Mitgliedschaft des Apple Developer Program verpflichtend. Diese wird mit aktuell 99 US Dollar jährlich berechnet. In der Mitgliedschaft sind die Rechte zur Veröffentlichung von iOS, watchOS und OS X Anwendungen enthalten sowie das Anlegen und Durchführen von Beta-Tests mit Testflight (Apple Developer 2016).

Um mit libGDX iOS Apps erstellen zu können, besteht die Bindung an einen speziellen Kompilierer, mit dem Java in Obective-C Code übersetzt werden kann. Hierfür wird mit RoboVM zusammengearbeitet. Die Einbindung dieses Kompilierers ist beispielsweise in Eclipse, Intellij IDEA und auf IDEA basierten Entwicklungsumgebungen möglich. Eine eigene IDE namens RoboVM Studio steht ebenfalls zur Wahl. Die Nutzung von RoboVM kostet derzeit 25 US Dollar bei einer Einzellizenz. Es werden aber auch reduzierte und kostenfreie Modelle für Indie Entwickler und Studenten

angeboten (RoboVM 2016).

Falls die Erstellung von Windows Phone Apps gefordert ist, wird die Verwendung von Visual Studio benötigt. Diese IDE setzt mindestens Windows 7 voraus. Um den Simulator nutzen zu können, wird Microsofts Virtualisierungstechnik Hyper-V vorausgesetzt. Hyper-V fordert wiederum eine Windows 8 Pro oder Windows 10 Pro Version (Visual Studio 2016).

6.3.2 Unterstützte IDEs

In Tabelle 6.2 werden die IDEs aufgelistet, die zur Bearbeitung des Quellcodes unterstützt und empfohlen werden. Abhängig davon, welche Programmiersprache man bevorzugt, hat man je Spieleframework mehrere IDEs zur Auswahl. Wie im vorigen Kapitel bereits erwähnt, ist Xcode mit OS X für jede iOS App verpflichtend, ebenso wie Visual Studio für Windows Phone Anwendungen nötig ist. Xcode kann bei Cocos2D-X für die Bearbeitung des Basiscodes verwendet werden, bei den anderen Frameworks nur für die Bearbeitung des kompilierten Codes. In der genannten Tabelle werden nicht alle unterstützten Programmiersprachen gelistet, sondern nur diejenigen, welche in Verbindung mit den Spieleframeworks relevant sind. Die jeweiligen Vor- und Nachteile der einzelnen Entwicklungsumgebungen werden in dieser Arbeit nicht weiter behandelt und auch nicht weiter vertieft, sondern dienen 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	Obj-C / C++	X	X	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 mit einer neuen Funktionalität belegbar sind. Spiele, die für mobile Systeme entworfen wurden, greifen daher auf alternative Steuerungsmechanismen zurück.

Überwiegende Praxis ist es, den vorhandenen Touchscreen für Eingaben und Interaktionen zu nutzen. Dies kann über Gesten geschehen oder über die Virtualisierung von Aktionsflächen, wie zum Beispiel Buttons und Schieberegler. Alle drei Spieleframeworks verfügen über Möglichkeiten, verschiedene Touchgesten zu unterscheiden. Eine weitere Möglichkeit in das Spielgeschehen eingreifen zu können bieten die geräteseitigen Sensoren. Zu den üblichen Sensoren gehören beispielsweise Gyroskop, Accelerometer und Kompass. Ein Anwendungsbeispiel für die Verwendung des Gyroskop Sensor wäre bei Rennspielen die Simulierung eines Lenkrads, wobei durch die Neigung des Gerätes gesteuert wird. Aber auch die Kamera fällt in diese Kategorie. Durch die Unterstützung von maschineller Bildverarbeitung können innerhalb von Echtzeitvideos Funktionalitäten hinterlegt werden. Ein Beispiel dafür sind Spiele mit Augmented Reality.

Unity3D liefert für fast alle gängigen Gerätefunktionen entsprechende Schnittstellen, um diese bei Bedarf mit Funktionalität zu hinterlegen (Unity3D 2015). Bei **libGDX** kann auf die meisten Sensorentypen zugegriffen werden, ausschließlich der Kamera und GPS Lokalisierung (libGDX 2013). Im Vergleich zu den Anderen kann **Cocos2D-X** bisher nur den Accelerometer ansteuern, wenn dieser vorhanden ist. Gyroskop, Kamera, GPS und weiteres werden derzeit nicht offiziell unterstützt (Cocos2D-X 2015).
Dadurch, dass libGDX und Cocos2D-X quelloffen sind, ist bei Bedarf eine eigene Implementation dennoch möglich. In Tabelle 6.3 werden die häufigsten Gerätefunktionen aufgelistet und die Unterstützung durch eine zugehörige Schnittstelle markiert. Der Punkt Netzwerkverbindung ist so zu verstehen, dass keines der drei Spieleframeworks das WLAN Modul oder das mobile Internet selbstständig ein- und ausschalten kann. Aus Gründen der Sicherheit kann dies nur der Nutzer selbst. Es kann lediglich getestet werden, ob eine Verbindung zu einem Netzwerk vorhanden oder möglich ist. Für die Prüfung einer Bluetooth Verbindung gibt es derzeit für keines der drei Spieleframeworks eine offizielle Schnittstelle.

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, können verschiedene, meist cloudbasierte, Services für optionale Spielinhalte eingebunden werden. Mit solchen Services können zum Beispiel Erfolgssysteme, Ranglisten und Mehrspielersysteme realisiert 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 umgesetzt werden. Kompetitive Elemente können mit Bestenlisten erzielt werden, in denen bestimmte, spielinterne Metriken der Spieler sortiert aufgelistet werden können. Ranglisten über die höchste Punktzahl oder die längste Spielzeit geben dem Spieler Anreiz, sich zu verbessern und mit anderen zu messen. In Mehrspielermodi, die entweder rundenbasiert oder in Echtzeit ablaufen, treten zwei oder mehr Spieler gegeneinander an, um in einem Wettbewerb den besten Spieler zu ermitteln. Eine weitere Variante sind Modi, in denen die Teilnehmer kooperieren müssen, um Spielziele gemeinsam zu erreichen. Um verbesserungswürdige Schwachstellen in der Anwendung zu ermitteln und das Verhalten von Spielern zu messen, werden Analysesysteme eingesetzt. Diese Systeme ermöglichen Einblicke auf Aktivitäten und Fortschritte der Spieler sowie Informationen zu getätigten Käufen innerhalb der Anwendung. Auch die Häufigkeit der Nutzung der Anwendung wird messbar. Diese Statistiken können zu einem besseren Verständnis der eigenen Anwendung verhelfen.

Für den Erwerb von zusätzlichen, anwendungsinternen 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 zugehörigen Stores passende Schnittstellen bereitstellt. In Tabelle 6.4 werden APIs aufgelistet, die populäre Game Services in den Spieleframeworks unterstützen. Die Tabelle zeigt hauptsächlich APIs, die plattformübergreifende Eigenschaften aufweisen und auf den beiden Zielplattformen iOS und Android funktionieren. Die Cloudservices von Google Play Game Services, App42 und NextPeer bieten die meisten Schnittstellen zu den genannten Spielinhalten.

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, plattformübergreifende Services für den mobilen Gaming-Bereich. Die Voraussetzung für den Spieler, um diese Features nutzen zu können, ist ein Account in Googles Netzwerk Google+. Obwohl dies in der Tabelle nicht markiert wurde, bietet Play Game Services Werkzeuge zur Analyse des Spielerverhaltens an. Die Angaben von Google und den Spieleframeworks sind bezüglich der Unterstützung von Analysemöglichkeiten nicht eindeutig. Die Cloud Dienste von Google sind kostenfrei nutzbar. (Play Game Services 2015).

App42
App42 von ShepHertz Technologies bietet ein umfangreiches Angebot an Cloudservices, die für Indieentwickler größtenteils kostenfrei zur Verfügung stehen. Es gibt Schnittstellen für eine Vielzahl an Plattformen und Frameworks, sowohl im nativen als auch im plattformübergreifenden Bereich (App42 2016).

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

6.6 Produktvarianten

Bei **libGDX** existieren keine weiteren, offiziellen Produktabspaltungen. Weitere populäre Derivate konnten nicht ermittelt werden, sondern nur Ableger, die aus privatem Bestrebungen entstanden sind. Die offizielle Community konzentriert sich daher hauptsächlich auf die Unterstützung und Weiterentwicklung der Hauptversion oder auf spezialisierte Module zur Erweiterung.

Cocos2D-X ist eine der offiziellen Abzweigungen aus der Cocos2D Familie. Die Basis bildet das mit Python entwickelte Cocos2D. Der erste, abgeleitete Ableger ist Cocos2D-iPhone oder auch bekannt als Cocos2D-ObjC. Diese Variante ist auf die Entwicklung mobiler Apple Geräte spezialisiert. Anwendungen können mit den plattformüblichen, nativen Sprachen Objective-C und Swift implementiert werden. Von dieser Version wurden wiederum weitere, spezialisierte Varianten abgeleitet. Dazu zählt Cocos2D-X, das die Unterstützung multipler Plattformen unterstützt. Die Ableger Cocos2D, -iPhone, -X, -HTML5 und der Editor SpriteBuilder gehören zu den offiziellen Projekten und werden durch ein koordiniertes Updateverhalten versorgt (Cocos2D-X 2015).

Unity3D bietet zwei Produktvarianten an, die Personal und die Professional Edition. Bei der Personal Edition handelt es sich um eine frei verfügbare Version, die für jeden nutzbar ist. Diese kann für Privatanwender, zu Bildungszwecken und für kommerzielle Zwecke benutzt werden. Bei der kostenfreien Version der Engine wird dem Nutzer keine der Basisfunktionalitäten vorenthalten. Sobald die grundlegenden Einnahmen eines Entwicklers oder Unternehmens mehr als 100.000 US-Dollar aus dem Vorjahr

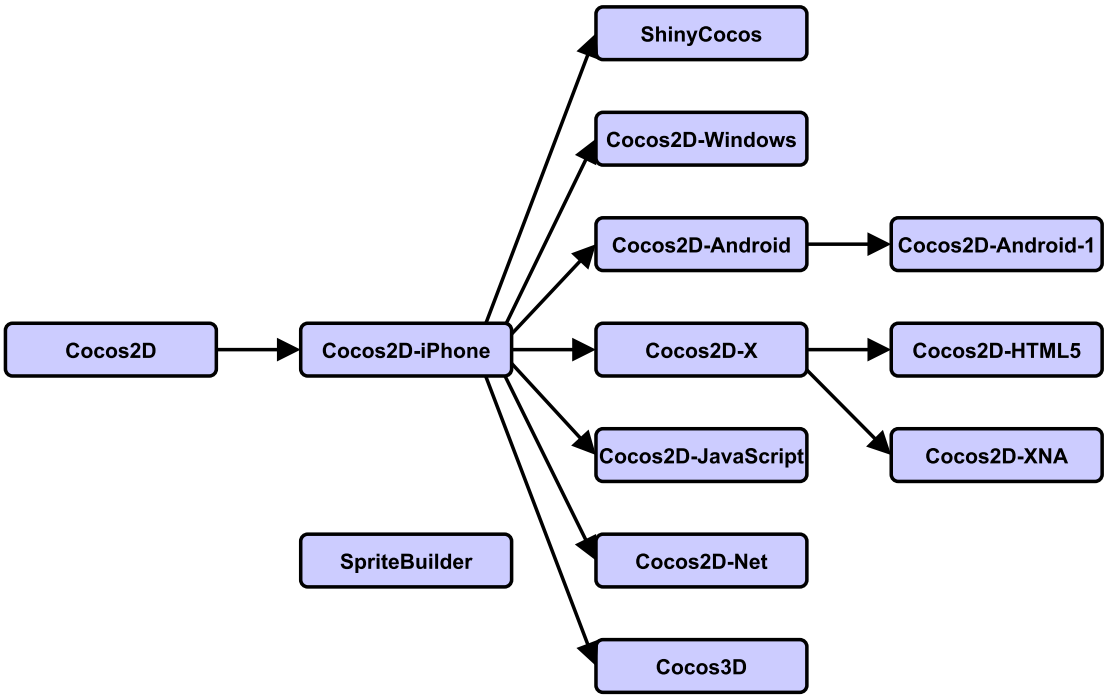

Abbildung 6.1: Ableitungen und Varianten von Cocos2D (Cocos2D-X 2015)

betragen, besteht die Verpflichtung, für den kommerziellen Gebrauch die Professional Edition zu verwenden. In dieser werden weitere Optionen auf zusätzliche Services freigeschaltet, wie Analyse Tools und cloudbasierte Projekte. In Tabelle 6.5 werden die Möglichkeiten beider Editionen aufgelistet. Das Kostenmodell der Professional Edition beginnt derzeit für ein Abonnement mit 75 US-Dollar pro Monat oder bei einer Einmalzahlung von 1.500 US-Dollar. Die mit der Einmalzahlung erworbene Dauerlizenz gilt nur für die aktuelle Version und deren Updates. Das bedeutet, dass bei dem Erwerb von Unity Professional 5 nicht automatisch die Nutzungsrechte für die darauf folgende Version 6 erhalten werden. 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 Möglichkeiten zur Personalisierung der Anwendung ermöglichen. Professionelle Projektunterstützung und der Zugang zum Quellcode der Engine können durch zusätzliche, kostenpflichtige Pakete erworben werden (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 eine mobile Beispielapplikation konzipiert. Diese wird mit den gewählten Spieleframeworks plattformunabhängig umgesetzt, um somit Versionen für verschiedene Plattformen zu erzeugen. Die Anwendungsversion soll danach anhand von vereinbarten Metriken bemessen und analysiert werden. Die gewonnenen Resultate werden daraufhin in Zusammenhang mit den theoretischen Informationen komplementiert und die Entwicklungswerkzeuge als Ganzes verglichen.

7.1 Definition von Anforderungen

Applikationen, aus Kategorien die den Schwerpunkt nicht auf Spiele legen, sondern beispielsweise auf Business und Unterhaltung, greifen in der Regel auf plattformspezifische Benutzeroberflächen zurück. Native Apps besitzen systembedingte, technische und optische Konventionen. Cross-Plattform Apps versuchen diese oft möglichst genau abzubilden und die Konventionen einzuhalten, um dem Benutzer ein natives Look-and-Feel zu bieten oder den Vorgaben der Stores nachzukommen. 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, das Forscher aus den unterschiedlichsten Fachbereichen beschäftigte und beschäftigt. Ein Teilbereich der Spielwissenschaft, genannt Ludologie (Die Lehre über das Spiel), befasst sich mit der Erforschung des Spielens und des digitalen Spielens (Junge 2015).

Die Forschungsinhalte beschäftigen 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 wird der elementare Inhalt und den Umfang eines Computerspiels definiert?

Um nicht alle konkurrierenden Theorien zu berücksichtigen, 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 werden einige wesentliche Konzepte entnommen und von Salen und Zimmerman näher erläutert:

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 eines Spiels. Dieser kann verschiedene Formen annehmen, wie kompetitive Elemente oder der Konflikt gegen das Spielsystem selbst.

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

Quantifizierbares Ziel
Das Ergebnis eines Spiels ist entweder, dass 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.

Anhand dieser Definitionen werden die Anforderungen für die Beispielapplikation konkretisiert:

- Es soll eine Spielfigur vorhanden sein, auf die der Spieler Einfluss nehmen kann.
- Die Spielfigur soll auf Hindernisse treffen, die die Aktionen des Spielers beeinflussen können.
- Das Spiel gilt als gewonnen, wenn ein definiertes Ziel erreicht oder das Spiel nicht beendet wird.
- Der Spieler verliert, wenn eine definierte Anzahl an Fehlversuchen erreicht werden, das Spiel abgebrochen wird oder andere Aktionen ausgeführt werden, 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 elementarsten 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 das 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. Ein Spiel wird verloren, wenn 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 Anlehnung an Flappy Bird entstanden zahlreiche Spiele, die auf dieser Spielidee basieren. Dieses Spielprinzip wird 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 einfach zu verstehen. Das Spiel ist auf den Portrait-Modus (Hochformat) festgelegt und für die spätere Analyse 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 wird ein Hauptmenü dargestellt, 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 zeigt einen Vogel als Spielfigur.

6. Der Vogel verliert an Höhe, wenn der Spieler nichts tut und steigt bei Berührung des Touchscreens an.

7. In einem festgelegten Intervall kommen der Figur Hindernisse in Form von grünen Röhren entgegen. Diese besitzen eine Lücke, durch die der Vogel hindurchmanövriert werden soll. Die Position der Lücke befindet sich in einer zufälligen, vertikalen Stelle innerhalb der Sichtbarkeit.

8. Wenn der Vogel erfolgreich ein Hindernis passiert hat, erhält der Spieler 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 begrenzt und kann nicht passiert werden.

11. Bei verlorenem Spiel wird in das Game Over Menü gewechselt das vom Aufbau dem Hauptmenü entspricht. In dieser Szene wird der Game Over Status angezeigt. Über den Startbutton kann ein neues Spiel gestartet werden.

Die in Kapitel 7.1 definierten Anforderungen werden durch die Spielidee und den Spielfluss somit komplett umgesetzt. Die Spielfigur wird mit Hilfe einer Spritesheet-Animation animiert und ist durch Berührung des Touchscreen 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 läuf in einer Endlosschleife. Das Spiel wird komplett in 2D gehalten. Auf weitere, optionale Spielinhalte wird verzichtet, da diese für eine Testapplikation unnötig sind.

7.4 Verwendete Werkzeuge

In diesem Kapitel werden die genutzten Softwarewerkzeuge in Tabelle 7.1 aufgelistet, die bei der Entwicklung der Testapplikation eingesetzt wurden. Für die Versionierung der Projekte wurde Git verwendet.

	Betriebssystem	Version
Betriebssysteme		
Mac OS X	Mac OS X	10.10.5
Windows	Windows	Windows 8.1 Standard Edition 64-Bit
Game-Framework/Engine		
libGDX	Mac OS X, Windows	1.9.2
Cocos2D-X	Mac OS X, Windows	3.9
Unity3D	Mac OS X, Windows	5.3.2f1
Programmiersprache		
Java	Mac OS X, Windows	1.8.0
C++	Mac OS X, Windows	11
C#	Mac OS X, Windows	3.0
Objective-C	Mac OS X	2.0
Python	Mac OS X, Windows	2.7.11
Entwicklungsumgebung		
Xcode	Mac OS X	7.2.1
Android Studio	Mac OS X, Windows	1.3.1
Visual Studio	Windows	14.0.24720
Monodevelop	Windows, Mac OS X	5.9.6
Sonstiges		
Android SDK	Mac OS X, Windows	6.0
Android NDK	Mac OS X, Windows	r10e
Apache Ant	Mac OS X, Windows	1.9.6
.NET/Mono	Mac OS X, Windows	2.0
RoboVM	Mac OS X, Windows	1.13.0

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.

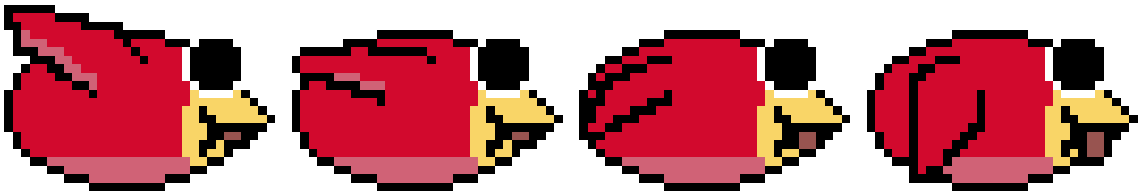

Abbildung 7.1: Spritesheet für die Animation der Spielfigur in Happy Bird

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 den Informationen zu den Grafiken, Positionen und Kollisionsboxen. Für die Erstellung der Grafiken und Animationen wurde der freie Online-Spriteeditor Piskel verwendet (Piskel 2015). Die verwendete 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 mit ihren

Eigenschaften sowie einige Framework-abhängige Dateien aufgelistet.

Dateiname	Dateityp	Objekttyp	Verwendung	Dateigröße (KB)
Red_Bird	PNG	Spritesheet	Spielfigur	1,54
bird_plist	PLIST	Property List	Spielfigur	1,63
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 den Anforderungen entsprechend C++ bei Cocos2D-X, C# bei Unity3D und Java bei libGDX für die Codebasis gewählt. Mit welcher Programmiersprache die schnellsten und effizientesten Ergebnisse erzielt werden können, steht in Verbindung mit der Vorerfahrung eines Entwicklers. Aber auch die Unterstützung durch eine qualitativ hochwertige Dokumentation und praktische Beispiele beeinflussen den Entwicklungsfortschritt. Für die Programmierung wurden die offiziellen Dokumentationen und Anleitungen zur Unterstützung verwendet. Es wurde zudem versucht, für die korrekte Darstellung alle gängigen Displaygrößen zu unterstützen. Die Darstellung konnte aber mangels Auswahl an Testgeräten nur innerhalb des Simulators getestet werden.

8 Analyse der Test-Applikationen

8.1 Definition von Metriken

Die erstellten Applikationen sollen näher analysiert und verglichen werden. Um dies möglichst objektiv und repräsentativ durchzuführen, werden vergleichbare Metriken definiert. Diese Metriken bilden anschließend den inhaltlichen Kern für das Messprotokoll in Kapitel 8.3. Weiterhin muss für jede Metrik der Ablauf der Messung sowie der Ausgangszustand der verwendeten Testgeräte normiert werden.

Ausgangsgröße einer Basisapplikation
Eine Basisapplikation ist eine Anwendung, die bei Anlegung eines neuen Projekts erzeugt wird. Dies ist eine App mit einer automatisch erzeugten Szene/View und soll dem Zweck dienen, eine Vorstellung über die Ausgangsgröße einer unveränderten, minimalen Applikation zu geben. Dieser Messwert soll das Verhältnis von Ausgangsgröße zur Gesamtgröße des erstellten Spiels anschaulicher darstellen. Für die Messung wird jeweils ein neues Projekt angelegt, die notwendigen Einstellungen für die mobile Unterstützung vorgenommen und daraus Applikationen erzeugt. Danach wird die Größe der installierten Anwendung auf dem Testgerät dokumentiert. Ein niedriger Wert gilt als positiv und ein hoher als negativ. Der Wert wird in Megabyte angegeben.

Gesamtgröße des Spiels
Dies beschreibt einen Messwert, der die Gesamtgröße des Spiels inklusive aller erstellten und hinzugefügten Inhalte beschreibt. Es werden alle genutzten Ressourcen, Bibliotheken und sonstige Dateien hinzugezählt. Für die Messung wird die Größe der installierten Applikationen auf den Testgeräten dokumentiert. Ein niedriger Wert gilt als positiv und ein hoher als negativ. Der Wert wird in Megabyte angegeben.

Größe der Ressourcen
Die Ressourcen sind die in Kapitel 7.5 deklarierten Komponenten. Hierzu zählt die

Schnittmenge der Komponenten, die einheitlich in allen Projekten verwendet werden, sowie zusätzlich jene Bestandteile, die in Abhängigkeit zu einem Spieleframework stehen. Für die Messung wird die unkomprimierte Größe der individuellen Ressourcenordner innerhalb der Projekte dokumentiert. Ein niedriger Wert gilt als positiv und ein hoher als negativ. Der Wert wird in Megabyte angegeben.

Benötigter Arbeitsspeicher
Dieser Messwert dokumentiert die Größe des belegten Arbeitsspeichers zur Laufzeit. Auf den mobilen Testgeräten von Android ist die Messung während der laufenden Anwendung nicht gleichzeitig möglich. Für die Erfassung des Werts muss die Applikation verlassen werden und aus den Systemeinstellungen abgelesen werden. Währenddessen bleibt die App im Hintergrund aktiv. Bei iOS Geräten kann aus den Laufzeitanalyse-Tools von Xcode abgelesen werden. Ein niedriger Wert gilt als positiv und ein hoher als negativ. Der Wert wird in Megabyte angegeben.

Ladegeschwindigkeit bei Neustart
Die Ladegeschwindigkeit ist die gemessene 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 Anwendung geladen wird. Hierfür wird vorausgesetzt, dass die Applikation nicht im Hintergrund aktiv ist. Ein niedriger Wert gilt als positiv und ein hoher als negativ. Der Wert wird in Sekunden angegeben.

Frames pro Sekunde
Die Berechnung und Darstellung der Framerate wurde dem Spiel manuell hinzugefügt. Dieser Wert wird innerhalb der Spielszenen dargestellt und während des Spielflusses beobachtet. Hierbei wird der Durchschnittswert dokumentiert. Falls es zu eventuellen, auffälligen Schwankungen kommen sollte und diese den Spielfluss beeinträchtigen, wird eine entsprechende Bemerkung im Abschluss dazu gegeben. Ein hoher, stabiler Wert gilt als positiv und ein niedriger, schwankender als negativ. Der Wert wird in durchschnittlichen Frames pro Sekunde angegeben.

Akkuverbrauch
Der Akkuverbrauch wird über einen Zeitraum von 15 Minuten gemessen. Ausgangswert ist ein vollständig 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 und ein hoher als negativ. Der Wert wird in Prozent angegeben.

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. Der Wert entspricht der Versionsnummer für die Mindestanforderung des Systems. Die Versionsnummer kann bei Android in Tabelle 3.1 und bei iOS in Tabelle 3.2 verglichen werden.

Codezeilen
Die Menge der Codezeilen ist nicht unbedingt eine vergleichbare Metrik, da diese in Abhängigkeit des Programmierstils, der Erfahrung des Entwicklers, der Programmiersprache, der Architektur, des Einsatzes von Kommentaren und der Verwendung von leeren Zeilen abhängig ist. Jedoch soll hierdurch ein grober Eindruck über den benötigten Code und damit verbundenen Aufwand vermittelt werden. Ein niedriger Wert gilt als positiv und ein hoher als negativ.

8.2 Testgeräte und Voreinstellungen

Die Applikationen werden auf Smartphones mit Android und iOS System getestet. Die relevanten Gerätedaten der verwendeten Testgeräte sind in Tabelle 8.1 aufgelistet. Falls zur Reproduktion ein anderes Testgerät verwendet werden soll, kann in der Tabelle 3.1 für Android und Tabelle 3.2 für iOS die Version des Betriebssystem für die Mindestanforderung verglichen werden. Um die zu dokumentierenden Messwerte 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 Töne deaktiviert.

- Alle weiteren Anwendungen und Prozesse werden weitestgehend beendet.
- Das Gerät wird in den Flug-Modus gesetzt, um die Verbindungsversuche zu einem Provider zu unterbinden.
- Das Spiel wird auf dem internen Speicher installiert.

Technische Daten	Android	iOS
Hersteller	Samsung	Apple
Modell	Galaxy S5+	iPhone 4S
Artikelname	G901F	-
Betriebssystem	Android 5.0.2	iOS 9.2
System-on-a-Chip (SoC)	Qualcomm Snapdragon 805 APQ8084	Apple A5
Prozessor	Krait 450 2500 MHz 4 Kerne	ARM Cortex A9 800 MHz 2 Kerne
Grafikprozessor	Qualcomm Adreno 420 600 MHz	PowerVR SGX 543MP 2 x 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 eingesetzten Smartphones handelt es sich um gebrauchte Geräte, was bedeutet, dass manche Leistungsmerkmale zu baugleichen 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 vorher definierten Metriken, in Zusammenhang mit den angegebenen Testgeräten, aufgelistet. In Tabelle

8.2 sind die Ergebnisse der Android Version und in Tabelle 8.3 die der iOS Version zu sehen.

Erläuterung zu den Einträgen:
Bestes Ergebnis = Hervorgehoben (fett)
Teilergebnis (unfertig) = Stern (*)
Kein Messergebnis = Strich (-)

Happy Bird - Android				
Metrik	**Wert**	**libGDX**	**Cocos2D**	**Unity3D**
Ausgangsgröße einer Basisapplikation	MB	**8,12**	14,63	29,09
Gesamtgröße des Spiels	MB	*13,4	**19,7**	32,66
Größe der Ressourcen	MB	5,9	5,9	**4,9**
Benötigter Arbeitsspeicher	MB	*22,97	**41,73**	113,63
Ladegeschwindigkeit bei Neustart	Sekunden	1.16	**0,85**	3,02
Frames pro Sekunde	Frames / Sekunde	-	**60**	**60**
Akkuverbrauch	Prozent	5	5	5
Mindestanforderung des Systems	Versionsnummer	**2.3**	**2.3**	**2.3**
Codezeilen	Zeilen	-	340	**104**

Tabelle 8.2: Messprotokoll des Spiels auf Android

Happy Bird - iOS				
Metrik	**Wert**	**libGDX**	**Cocos2D**	**Unity3D**
Ausgangsgröße einer Basisapplikation	MB	48,9	**7,2**	22,4
Gesamtgröße des Spiels	MB	*54,4	**13,9**	38,8
Größe der Ressourcen	MB	5,9	5,9	**4,9**
Benötigter Arbeitsspeicher	MB	-	**20,3**	42,7
Ladegeschwindigkeit bei Neustart	Sekunden	2,18	**1,43**	7,62
Frames pro Sekunde	Frames / Sekunde	-	**60**	30
Akkuverbrauch	Prozent	0	0	0
Mindestanforderung des Systems	Versionsnummer	**6.0**	**6.0**	**6.0**
Codezeilen	Zeilen	-	340	**104**

Tabelle 8.3: Messprotokoll des Spiels auf iOS

8.4 Auswertung der Messergebnisse

Die dokumentierten Messergebnisse aus Kapitel 8.3 sollen nun anhand der gemessenen Metriken verglichen werden. Das Ergebnis jeder Metrik wird nachfolgend erläutert, wobei einerseits die plattformabhängigen, als auch die gemeinsamen, plattformübergreifenden Daten, berücksichtigt werden. Die Beispielapplikation konnte mit Cocos2D-X und Unity3D den Anforderungen nach umgesetzt werden. Die App konnte lediglich mit libGDX nicht final abgeschlossen werden, weshalb die Messergebnisse von diesem Framework nur den erreichten Stand abbilden. Es existiert die Spielszene inklusive des steuerbaren Vogels und Hindernisse, jedoch fehlen die Kollisionsabfragen, die Punktedarstellung, Hintergrundmusik und Szenenwechsel. Diese Ergebnisse geben dennoch eine ausreichende Einschätzung, um in den Vergleich miteinbezogen werden zu können.

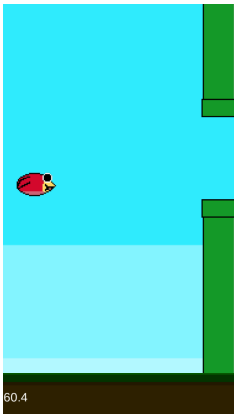

Abbildung 8.1: Mit Unity3D erstellte Spielszene unter Android

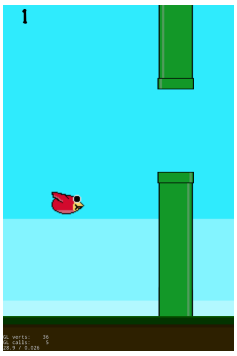

Abbildung 8.2: Mit Cocos2D-X erstellte Spielszene unter iOS

Die Abbildung 8.1 zeigt die Spielszene mit dem Android Testgerät, welche mit Unity3D erstellt wurde. Die Punktzahl ist noch nicht zu sehen, da die Spielfigur sich noch vor dem ersten Hindernis befindet. Die Framerate unten links wurde durch ein eigenes Script hinzugefügt. In Abbildung 8.2 ist die gleiche Spielszene aus der Cocos2D-X Applikation unter iOS zu sehen. Die Unterschiede bei den Spielinhalten sind kaum auszumachen. Unten links wird die Framerate angezeigt, dessen Darstellung durch eine Standardfunktion zum debuggen ermöglicht wird.

Ausgangsgröße einer Basisapplikation

Die Ausgangsgröße einer Basisapplikation ist bei Android mit 8,12 MB bei libGDX am geringsten, jedoch bei iOS im Gegensatz dazu mit 48,9 MB am größten. Die geringe Größe bei Android liegt vor allem an der verwendeten Sprache Java, dass die Erstellung nahezu nativ behandelt. Wodurch sich die überdurchschnittliche Größe der Anwendung bei iOS zusammensetzt, konnte nicht genauer analysiert werden. RoboVM nutzt zwar Xcode indirekt für die Erzeugung und Installation der Applikation auf einem Gerät, generiert aber dabei kein ausführbares Xcode Projekt. Dadurch konnten Xcodes Analysewerkzeuge nicht verwendet werden. Cocos2D-X besitzt bei beiden Plattformen einen geringen Speicherbedarf und kann sich in dieser Kategorie als der speichereffizienteste Kanditat hervortun. Unity3D benötigt bei beiden Plattformen eine ähnliche Grundgröße.

Gesamtgröße des Spiels

Da die Applikation mit libGDX nicht finalisiert wurde, wird dieser Wert nur teilweise berücksichtigt. Durch die Aufaddierung der Ressourcen auf die Basisapplikation ist jedoch anzunehmen, dass sich die Steigerung des Speicherbedarfs bei fertiggestellter Anwendung nur geringfügig erhöht. Bei libGDX sowie Cocos2D-X ergeben sich die Gesamtgrößen der Applikationen scheinbar ausschließlich durch die Erweiterung der Ressourcen. Bei Unity3D war dies nicht der Fall. Die Vergrößerung der Android-App ist kleiner als die hinzugefügten Ressourcen, wodurch anzunehmen ist, dass die Spielelemente komprimiert werden. Anders ist es bei iOS, da hier die Vergrößerung etwa um die dreifache Ressourcengröße ausfällt. Cocos2D-X kann bei dieser Messkategorie, mit 19,7 MB bei Android und 13,9 MB bei iOS, als Sieger erklärt werden.

Größe der Ressourcen

Die Größen der Ressourcen sind bei libGDX und Cocos2D-X gleich, da bei beiden Frameworks die gleichen Elemente genutzt werden. Unity3Ds Ressourcengröße ist im

Vergleich am geringsten, obwohl sich in diesem Ordner zusätzlich die geschriebenen Skripte und die gespeicherten Szenen befinden.

Benötigter Arbeitsspeicher
Die Reservierung des Arbeitsspeichers ist bei Unity3D am höchsten. Diesen Bereich kann Cocos2D-X ebenfalls für sich gewinnen, da der benötigte Arbeitsspeicher sehr gering ausfällt. Bei beiden Frameworks wird bei dem Android-Testgerät wesentlich mehr Arbeitsspeicher verwendet als bei dem iOS-Gerät. libGDX wird bei dieser Metrik nicht berücksichtigt.

Ladegeschwindigkeit bei Neustart
Der Ladevorgang ist bei den beiden Open Source Frameworks sehr kurz. Unity3D benötigt auf dem iOS-Gerät mit etwa 7,5 Sekunden am längsten, jedoch kann die Ladegeschwindigkeit bei Android mit den beiden anderen Frameworks mithalten. Die Darstellung des Ladebildschirms wird bei der Unity3D Personal Edition jedoch erzwungen, was die Geschwindigkeit beeinflusst. Cocos2D-X ist in dieser Kategorie das schnellste Framework.

Frames pro Sekunde
Bei der Framerate konnten keine Daten für libGDX gesammelt werden, da die Implementierung der Darstellung nicht umgesetzt wurde. Cocos2D-X läuft auf beiden Plattformen mit stabilen 60 Frames pro Sekunde. Dies wird codeseitig festgelegt. Unity3D forciert das Testgerät standardmäßig zu keiner fest definierten Framerate, sondern passt diese der Leistung des Gerätes an. Das iOS Testgerät lief daher nur mit 30 Frames pro Sekunde. Es ist aber ebenfalls möglich, bei Unity3D eine Framerate vorzugeben. Alle Applikationen liefen jedoch flüssig und stabil, weshalb die Ergebnisse als gleichwertig angesehen werden.

Akkuverbrauch
Der Akkuverbrauch ergab keine aussagekräftigen Ergebnisse. Bei dem iOS-Gerät war überhaupt kein Leistungsabfall zu vermerken, ähnlich der Differenz bei dem Android-Gerät. Grund hierfür kann der zu geringe Umfang der Applikation sein, da diese modernere Smartphones kaum fordert.

Mindestanforderung des Systems
Auch bei den Mindestanforderungen gab es keine Differenzen. Alle Frameworks setz-

ten die gleichen Mindestanforderungen voraus. Dies könnte ebenfalls mit der geringen technischen Anforderung einhergehen.

Codezeilen
Die Zeilenanzahl des geschriebenen Code ist bei Unity3D am geringsten, da die Funktionalität durch die umfangreiche Bibliothek kompakt ausgedrückt werden kann. Bei Cocos2D-X ist deutlich mehr Code erforderlich, was sich hauptsächlich durch die Sprache C++ und die zusätzlich benötigten Header-Dateien ergibt. libGDX wurde durch die Unvollständigkeit der App nicht mit aufgeführt. Es zeichnete sich jedoch ein Umfang ab, der ähnliche Ausmaße wie Cocos2D-X annahm.

In Tabelle 8.4 wird das Gesamtergebnis dargestellt, wobei das jeweils beste Spielefamework einer Metrik durch ein X markiert ist. Diese Bewertung ergibt sich aus den zusammengenommenen Ergbnissen beider Plattformen. Auch wenn libGDX nur teilweise berücksichtigt wurde, konnte es in der Gesamtwertung keinen eindeutigen Vorteil verzeichnen. Mit sechs gewonnenen Kategorien kann sich Cocos2D-X im Gebiet der Effizienz und Ausführungsgeschwindigkeit vor der Konkurrenz behaupten.

Gesamtergebnis			
Metrik	**libGDX**	**Cocos2D**	**Unity3D**
Ausgangsgröße einer Basisapplikation		**X**	
Gesamtgröße des Spiels		**X**	
Größe der Ressourcen			**X**
Benötigter Arbeitsspeicher	-	**X**	
Ladegeschwindigkeit bei Neustart		**X**	
Frames pro Sekunde	-	**X**	**X**
Akkuverbrauch			
Mindestanforderung des Systems		**X**	**X**
Codezeilen	-		**X**
Ergebnis		**6**	**4**

Tabelle 8.4: Gesamtergebnis anhand der Messerergebnisse

9 Vergleich zur Benutzbarkeit

In diesem Kapitel werden die Erfahrungen wiedergegeben, die während des Implementierungsprozess gewonnen wurden.

Durch die in Kapitel 8 durchgeführte Messung und darauf folgende Auswertung stellte sich heraus, dass mit Cocos2D-X sehr effiziente Resultate erzielt werden können. Bei beiden Zielplattformen konnten kurze Ladezeiten, geringer Speicherbedarf und effektive Kompilierung ohne stark anwachsende Datenmengen beobachtet werden. Das Ergebnis wird allerdings zu Lasten von einer vergleichsweise umfangreicheren Einarbeitungszeit und ebenso intensiven Testphasen beeinflusst. Bei der Einarbeitung wurden durch die offizielle Dokumentation verständliche Erklärungen und Hilfen für die beste Vorgehensweise bereitgestellt. Die Sprache C++ spielt seine Vorteile als Basis für die Spieleentwicklung in der umgesetzten Beispielapplikation voll aus. Jedoch sind für erfolgreiche und stabile Resultate längere Entwicklungszeiten und fortgeschrittene Erfahrungswerte notwendig sowie der sichere Umgang mit dieser Sprache.

Während der Entwicklungszeit wurden die zuvor hohen Erwartungen an libGDX stark getrübt. Obwohl die Entwicklung der Android Applikation vergleichsweise die effizientesten Messergebnisse ergeben hat, waren die Ergebnisse bei iOS genau komplementär. Der Vorteil der Modularität wirkt sich bei mobilen Anwendungen demnach nur bei Android aus. Auch die Kompilierung mit RoboVM zu iOS ist befremdlich, da kein Xcode Projekt erzeugt wird und das Resultat dadurch nicht ausreichend kontrolliert werden kann. Es konnten aus der IDE auch keine Paketdateien (apk/ipa) mit libGDX erstellt werden. Diese Erkenntnisse stellen den Nutzen der plattformübergreifenden Eigenschaften im Mobilbereich in Frage. Während der Entwicklung traten zudem verschiedene Hürden in Erscheinung, die dem Fortschritt unnötig im Weg standen. Die Dokumentation macht anfänglich einen guten und umfangreichen Eindruck, jedoch stieß man bei der Suche nach Umsetzungsmöglichkeiten von scheinbar einfachen Anforderungen schnell an die Grenzen. Für die Realisierung geeigneter Physik und

Kollisionsabfragen bei 2D Spielen wird in der Dokumentation das externe Framework Box2D vorgeschlagen, welches auch bei Cocos2D-X eingesetzt wurde. libGDX stellt dafür aber keine eigene Anleitung und verweist für die richtige Verwendung auf die Dokumentation von Box2D, welche allerdings nur Erklärungen für die Anwendung mit C++ bereitstellt. Auch die Einbindung von üblichen Schriftarten, wie TrueType Fonts, ist nicht möglich. Schrift innerhalb von Anwendungen wird bei libGDX durch pixelbasierte Bitmap-Schriften realisiert, die bei unterschiedlichen Bildschirmgrößen unschöne und unscharfe Skalierungen auslösen. Die Verwendung von libGDX zur Entwicklung von plattformübergreifenden, mobilen Spielen ist aufgrund der genannten Defizite nur bei gezielten Anforderungen ratsam.

Die Ergebnisse der Applikationen durch die Engine Unity3D scheinen auf den ersten Blick moderat auszufallen. Im Bereich der Speicherverwendung wird eine höhere Grundgröße beansprucht, die aber durch den Einsatz von zusätzlichen Ressourcen und Hinzufügen von Komplexitäten verhältnismäßig gering zunimmt. Die Anwendungen laufen stabil und fehlerfrei. Die vorherige Erwartung einer umfangreichen Einarbeitungszeit konnte während der Entwicklung widerlegt werden. Umgang und Arbeit mit dem Editor sind intuitiv zu verstehen und verständlich aufgebaut. Die erforderliche Logik konnte mit Hilfe der unterstützten IDEs in kurzer Zeit umgesetzt, im Editor getestet und angepasst werden. Durch die Möglichkeit, den Entwicklungsstand innerhalb des Editors zu testen und während der Laufzeit Anpassungen vorzunehmen, konnte effektiv und zielgerichtet gearbeitet werden. Bei auftretenden Fehlern während der Tests wurde zudem sofort auf die richtige Codezeile innerhalb des Scripts verwiesen. Der Gebrauch von Prefabs für die Mehrfachverwendung von wiederkehrenden Spielinhalten erwies sich als komfortabel und effizient. Durch die umfangreiche Dokumentation mit vielen Anwendungsbeispielen und Tutorials sind Momente der Ratlosigkeit von kurzer Dauer. Die Kompilierung zu den Zielplattformen wurde problemlos durchgeführt und ergab auf den Testgeräten das durch den Editor vorausgesetzte Ergebnis. Der Gebrauch von Unity3D für die Erstellung von plattformübergreifenden, mobilen Spielen ist durch die hohe Entwicklungsgeschwindigkeit und die Abdeckung aller denkbaren spielebasierten Anforderungen durchaus empfehlenswert.

10 Fazit

Cocos2D-X und Unity3D lieferten bei der Implementierung und der anschließenden Messung äußerst zufriedenstellende Ergebnisse. Cocos2D-X bietet das höchste Potential bei der Performance von Anwendungen und kann sich der Unterstützung einer umfangreichen Community erfreuen, die sich um die stetige, zeitgemäße Weiterentwicklung bemüht. Bei Spielen mit einfachen Anforderungen und geringem Inhalt kann sich Cocos2D-X gut präsentieren. Bei steigendem Umfang mit ausgedehnter Komplexität und vielfältigen Inhalten kann die effiziente Entwicklung mit diesem Framework aber nur durch erheblichen Aufwand umgesetzt werden. Dadurch sind ökonomische, plattformübergreifende Entwicklungsprozesse zwar dennoch möglich, aber eher Entwicklern mit reichhaltiger Erfahrung vorbehalten.

libGDX bietet zwar für die meisten Bedingungen eine externe Schnittstellenlösung an, stellt diese aber nicht ausreichend an das eigene Framework angepasst zur Verfügung. Dadurch verlieren viele dieser Schnittstellen ihre Zweckmäßigkeit, wenn sie nur aus Gründen der Vollständigkeit eingebunden werden, ohne aber sorgfältig verwaltet zu werden.

Unity3D ergab bei der Analyse der Applikation scheinbar durchschnittliche Ergebnisse, welche von dieser aber stabil und effizient ausgeführt wurden. Die kurze Einarbeitungszeit, die schnelle Erstellung von zweckmäßigen Resultaten sowie die vielseitigen Unterstützungen und Hilfestellungen sind ein immenser ökonomischer Vorteil, den Privatnutzer, Indieentwickler und professionelle Studios für sich nutzen können. Unity3D bietet zudem die umfangreichsten Schnittstellen zu externen Services und Gerätefunktionen sowie die größte Menge an unterschiedlichen Zielplattformen in allen Kategorien. Daher kann die Behauptung aufgestellt werden, dass es kaum Anforderungen an die Realisierung eines digitalen, mobilen Spiels gibt, die ein Hindernis für die Engine Unity3D darstellt. Somit kann sich dieses Spieleframework als vielseitiger, intuitiver und innovativer Alleskönner behaupten, dessen zunehmende Verbreitung sich durchaus rechtfertigt.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

AndEngine (2013): *Free Android 2D OpenGL Game Engine*, https://github.com/nicolasgramlich/AndEngine, letzter Zugriff: 28.12.2015

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): *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 Cloud API*, http://api.shephertz.com/, letzter Zugriff: 01.02.2016

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

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

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

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

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

C++ (2016): *C++ - A Brief Description*, http://www.cplusplus.com/info/description/, letzter Zugriff: 05.01.2016

Cocos2D-Android (2010): *cocos2d for Android: A framework for building 2D games for the Android platform*, `https://code.google.com/archive/p/cocos2d-android/`, letzter Zugriff: 28.12.2015

Cocos2D-iPhone (2015): *Cocos2D - ObjC Site*, `http://www.cocos2d-iphone.org/`, letzter Zugriff: 28.12.2015

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 - 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 - Dokumentation*, `http://www.cocos2d-x.org/reference/native-cpp/V3.9/index.html`, letzter Zugriff: 10.01.2016

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

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. : „Taxonomy of Cross-Platform Mobile Applications Development Approaches“, *Ain Shams Engineering Journal*, 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, Matthias & Raed, Allaithy & Wirsing, Martin (Hrsg.): *Java kompakt Eine Einführung in die Software-Entwicklung mit Java*, Springer 2013

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

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

jPCT-AE (2015): *jPCT 3D engine - the free 3D solution for Java and Android*, http://www.jpct.net/jpct-ae/index.html, letzter Zugriff: 28.12.2015

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): *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 - Goals and Features*, https://libgdx.badlogicgames.com/features.html, letzter Zugriff: 29.12.2015

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

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 - About*, http://www.lua.org/about.html, letzter Zugriff: 05.01.2016

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

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*, `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): *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 - Microsoft und Nokia Geräte*, `https://www.microsoft.com/de-de/nokia.aspx`, letzter Zugriff: 25.11.2015

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

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

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*, `https://www.nextpeer.com/`, letzter Zugriff: 01.02.2016

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): *Overview of the Open Handset Alliance*, `http://www.openhandsetalliance.com/oha_overview.html`, letzter Zugriff: 24.11.2015

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): *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 - Pixel Art and Animated Sprites*, `http://www.piskelapp.com/`, letzter Zugriff: 28.01.2016

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

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 - 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

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

Sparrow (2015): *Sparrow - The Open Source Game Engine for iOS*, `http://gamua.com/sparrow/`, letzter Zugriff: 28.12.2015

SpriteKit (2015): *iOS Developer Library - About SpriteKit*, `https://developer.apple.com/library/ios/documentation/GraphicsAnimation/Conceptual/SpriteKit_PG/Introduction/Introduction.html`, letzter Zugriff: 28.12.2015

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

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

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

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

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 - Public Relations*, `https://unity3d.com/public-relations`, letzter Zugriff: 29.12.2015

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

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

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

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 - Tools für alle Entwickler und alle Apps*, `https://www.visualstudio.com/de-de/dn469161`, letzter Zugriff: 15.02.2016

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

Wikipedia - Apple iOS (2015): *Apple iOS*, `https://de.wikipedia.org/wiki/Apple_iOS`, letzter Zugriff: 24.11.2015

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

Wikipedia - List of game engines (2016): *List of game engines(2016)*, `https://en.wikipedia.org/wiki/List_of_game_engines`, letzter Zugriff: 23.02.2016

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): *Microsoft Windows 10 Mobile*, `https://de.wikipedia.org/wiki/Microsoft_Windows_10_Mobile`, letzter Zugriff: 28.12.2015

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): *Microsoft Windows Phone 8*, `https://de.wikipedia.org/wiki/Microsoft_Windows_Phone_8`, letzter Zugriff: 28.12.2015

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

XNA (2015): *Microsoft Developer Network - XNA Game Studio 4.0*, `https://msdn.microsoft.com/de-de/library/bb200104(v=xnagamestudio.40).aspx`, letzter Zugriff: 28.12.2015

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

Zamora (2015): Zamora, Bernardo: *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

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