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

Bachelor-Thesis

zur Erlangung des akademischen Grades B.Sc.

Sebastian Bohn

2036605

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

Erstprüfer: Prof. Dr. Edmund Weitz

Zweitprüfer: Prof. Dr. Andreas Plaß

vorläufige Fassung vom 3. Januar 2016

Inhaltsverzeichnis

1 Einleitung **6**
- 1.1 Motivation . . . 6
- 1.2 Gliederung . . . 6

2 Mobile Systeme **7**
- 2.1 Marktanalyse zur Gewichtung der mobilen Systeme und der Applikationen . . . 7
  - 2.1.1 Marktanteile der mobilen Betriebssysteme . . . 7
  - 2.1.2 Verfügbare Applikationen und Kategorien der Stores . . . 7
- 2.2 Betrachtung der mobilen Systeme . . . 11
  - 2.2.1 Android . . . 11
  - 2.2.2 iOS . . . 12
  - 2.2.3 Windows Phone . . . 12

3 Native Softwareentwicklung **14**
- 3.1 Systemvoraussetzungen . . . 14
  - 3.1.1 Android . . . 14
  - 3.1.2 iOS . . . 14
  - 3.1.3 Windows Phone . . . 15
- 3.2 SDKs und Versionen . . . 15
  - 3.2.1 Android Versionen . . . 16
  - 3.2.2 iOS Versionen . . . 17
  - 3.2.3 Windows Phone Versionen . . . 17
- 3.3 Programmiersprachen . . . 18
  - 3.3.1 Android . . . 18
  - 3.3.2 iOS . . . 19
  - 3.3.3 Windows Phone . . . 19
- 3.4 Entwicklungsumgebungen . . . 19
  - 3.4.1 Android . . . 19
  - 3.4.2 iOS . . . 19
  - 3.4.3 Windows Phone . . . 20
- 3.5 Zusammengefasste Übersicht . . . 20

4 Plattformübergreifende Entwicklung **21**
- 4.1 Ziel . . . 21

4.2 Funktionsweise und Realisierungsansätze 22
4.2.1 Kompilierung 22
4.2.2 Komponentenbasiert 25
4.2.3 Interpretierung 25
4.2.4 Modellierung 28
4.2.5 Cloudbasiert 28
4.2.6 Vereinigung 29
4.3 Übersicht der Ansätze 32
4.4 Plattformübergreifende Entwicklung mobiler Applikationen ohne den Schwerpunkt Spieleentwicklung 33

5 Plattformübergreifende Frameworks zur Spieleentwicklung 34
5.1 Gamespezifische Frameworks und Engines 34
5.1.1 Cocos2D-X 34
5.1.2 Libgdx 34
5.1.3 Unity3D 34
5.1.4 Weitere Frameworks 34
5.2 Entwicklungsumgebungen 34
5.2.1 Unterstützte IDEs 34
5.2.2 Systembedingte Einschränkungen 34

6 Gegenüberstellung der Frameworks 35
6.1 Zielplattformen 35
6.2 Programmiersprachen 36
6.3 Unterstützung von 2D und 3D 37
6.4 Zugriff auf Hardware 37
6.5 Free- und Pro- Versionen 37
6.6 Einfluss auf Einstellungen 37
6.7 Zusätzlich benötigte Software 37
6.8 Aktualität - Versionen - Community 37
6.9 Zukunftsaussichten 37

7 Kosten-Nutzen Vergleich 38

8 Konzeption und Implementierung einer Test-Applikation 39
8.1 Ideen 39
8.2 Anforderungen 39
8.3 Verwendete Frameworks und Engines 39
8.4 Verwendete APIs und SDKs 39
8.5 Assets und deren Verwendung 39

9 Analyse messbarer Metriken 40

10 Vergleich der Messprotokolle 41

11 Fazit **42**

Abbildungsverzeichnis **43**

Tabellenverzeichnis **44**

Literaturverzeichnis **45**

Abstract

Zusammenfassung

1 Einleitung

1.1 Motivation

1.2 Gliederung

2 Mobile Systeme

2.1 Marktanalyse zur Gewichtung der mobilen Systeme und der Applikationen

Welche mobilen Systeme derzeit am meisten gefragt und verbreitet sind, soll in diesem Abschnitt analysiert werden. Dieses Wissen ist nötig, um vor dem Entwicklungsprozess die erfolgreichsten und erfolgversprechendsten Plattformen auszuwählen und miteinzubeziehen. Weiterhin soll geklärt werden wie viele Applikationen diese Plattformen in ihren Stores bereitstehen 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. (Abb. 2.1)
Die Grafik verdeutlicht, dass aktuell Geräte mit Android Systemen den Markt eindeutig dominieren. Darauf folgen Geräte mit iOS und Windows Phone. Laut Prognose wird sich auch in den nächsten Jahren an dieser Hierarchie nichts ändern. Schlussfolgernd sind diese drei Systeme die relevantesten auf dem globalen Markt.
Abbildung 2.2 gibt Aufschluss über die Verteilung der Systeme nach ausgewählten Ländern. Die Daten beziehen sich auf die Verkäufe von August bis Oktober 2015, welche von Kantar im Dezember 2015 veröffentlicht wurden. Bei der Internationalisierung von Applikationen ist es von Vorteil zu wissen, wie stark die Gewichtung der Systeme in den einzelnen Ländern ist.

2.1.2 Verfügbare Applikationen und Kategorien der Stores

Die Menge an verfügbaren Apps in den jeweiligen Stores ist unterschiedlich groß. Eine Analyse über die aktiven Applikationen in den einzelnen Stores und die Gewichtung der Kategorien, soll einen Überblick verschaffen was die jeweiligen Plattformen aktuell zu bieten haben. In Abbildung 2.3 wird die Menge an verfügbaren Apps im Mai 2015 dargestellt. Um eine bessere Übersicht zu gewährleisten, wurden die Werte gerundet. Der Amazon Appstore bietet wie der Google Play Store nur Android Apps an. Da es in

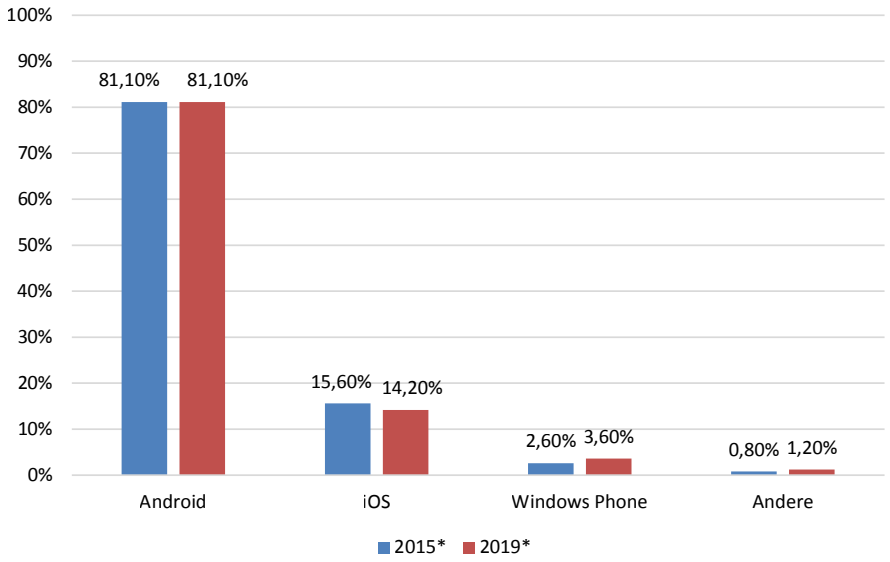

Abbildung 2.1: Prognose zu den Marktanteilen der Betriebssysteme am Absatz vom Smartphones weltweit in den Jahren 2015 und 2019 (IDC 2015)

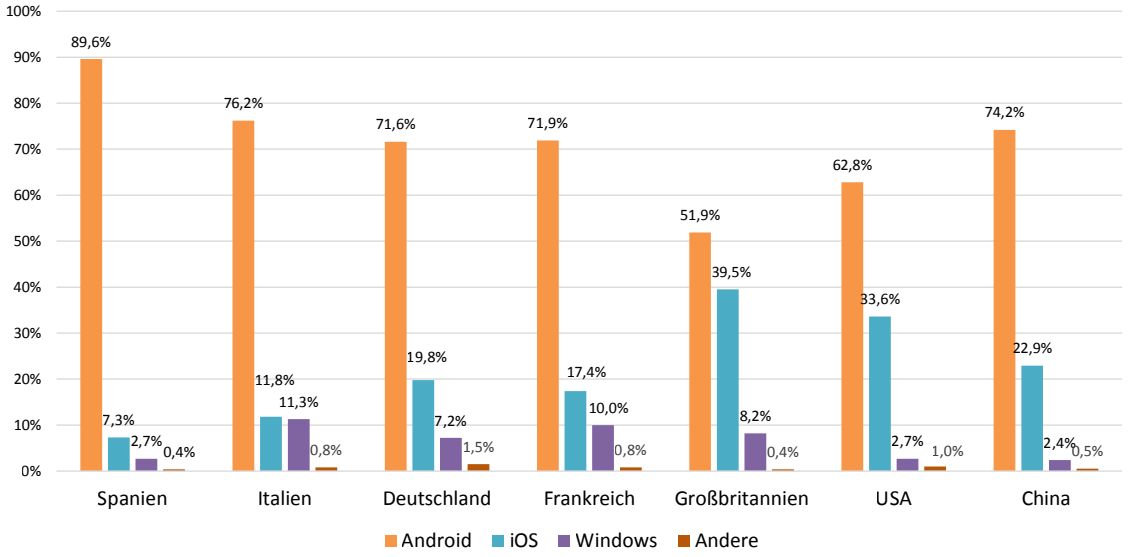

Abbildung 2.2: Marktanteile der mobilen Betriebssysteme am Absatz von Smartphones in ausgewählten Ländern von August bis Oktober 2015 (Kantar 2015)

diesen beiden Stores zum Teil zu Überschneidungen beim Angebot von Anwendungen kommt, werden diese Werte separat betrachtet und nicht summiert.

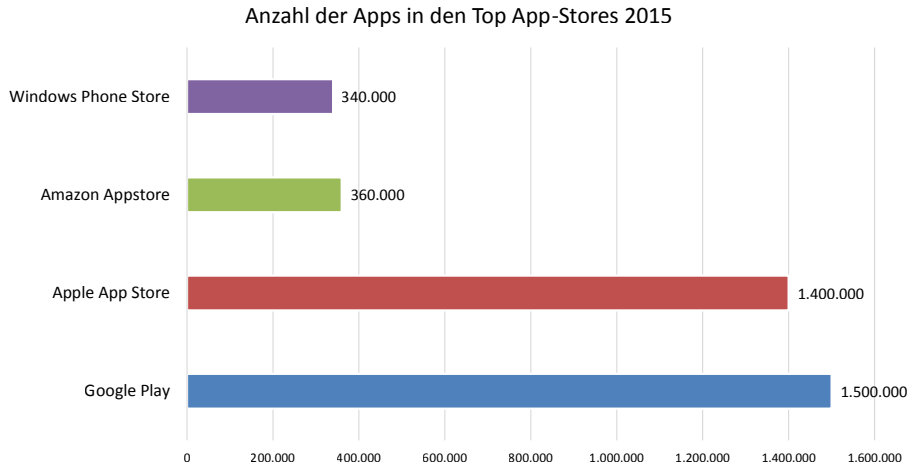

Abbildung 2.3: Anzahl der angebotenen Apps in den Top App-Stores im Mai 2015 (Statista 2015)

Der Wert für den Windows Phone Store ist der Quelle nach von September 2014 und schließt damit noch nicht die Windows 10 Universal Apps mit ein. Diese kamen erst Mitte 2015 dazu und werden in einem getrenntem Windows Store hardwareübergreifend 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 aussterbendem Version 7.8. Laut Windows wurden im September 2015 etwa 50% der Applikationen mit Windows 10 aus dem neuen Windows Store heruntergeladen. Diese Statistik gibt jedoch wenig Aufschluss wie groß dabei der Anteil an mobilen Systemen ist. Jedoch dominiert die Kategorie "Games"bei Windows 10 Apps mit fast 45% die Downloadzahlen.(Bernardo Zamora 2015)
Auch im Google Play Store werden Spiele Apps am häufigsten heruntergeladen und nehmen etwa 41% des Downloadvolumens ein.(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.(Abb.2.5)

Auch wenn jeder Store seine Applikationen auf eigene Weise kategorisiert, ist dennoch klar zu erkennen das 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 von entsprechender Entwicklungssoftware.

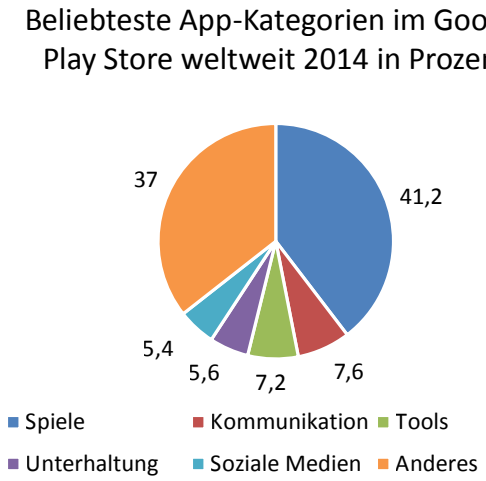

Abbildung 2.4: Anteil der im Google Play Store weltweit am häufigsten heruntergeladenen Apps nach Kategorien im Februar 2014 (Distimo 2014)

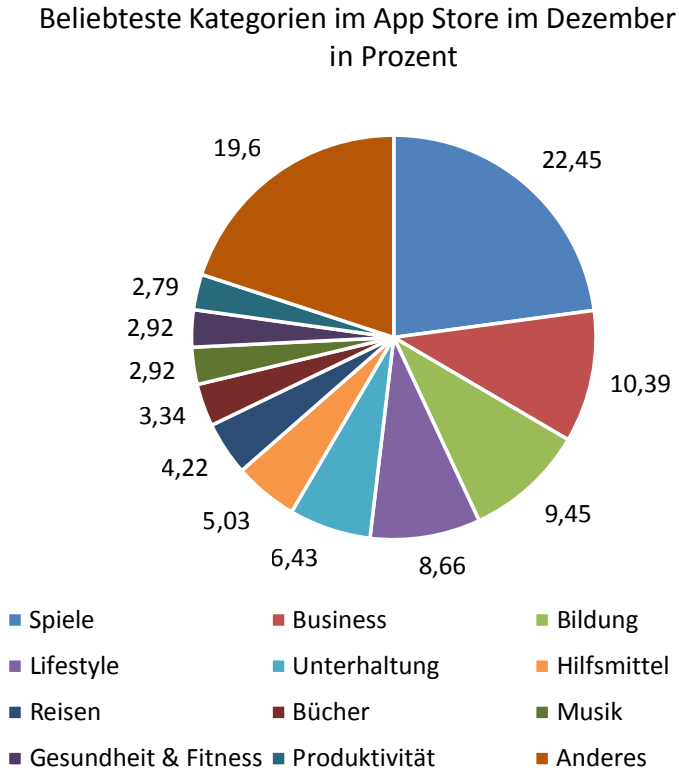

Abbildung 2.5: Ranking der Top-Kategorien im App Store im Dezember 2015 (PocketGamer.biz 2015)

2.2 Betrachtung der mobilen Systeme

Anhand der gewonnenen Erkenntnisse aus Kapitel 2.1.1, spielen derzeit die mobilen Systeme von Android, iOS und Windows Phone die größte Rolle auf dem Markt und bei den Benutzern. Folglich werden weitere Systeme nicht weiter betrachtet und der Fokus auf diese drei Systeme gerichtet.

2.2.1 Android

Android ist ein Open Source Betriebssystem und gleichzeitig eine Software-Plattform, welches stark im mobilen Bereich vertreten ist und auf dem Linux-Kernel basiert. Zu finden ist diese auf Smartphones, Tablet-Computern, Netbooks und auch auf Smart-TV Geräten. (Open Handset Alliance - Android Overview 2015) Entwickelt wird Android von der Open Handset Alliance (OHA), welche von Google gegründet wurde. Die OHA wurde im November 2007 gegründet und 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)
Durch seine Offenheit ermöglicht Android Entwicklern große Freiheit bei der Programmierung von Applikationen. Eigene Entwicklungen können auch mit Anwendungen von Google, wie zum Beispiel Google Maps, verknüpft werden.
Auch der Hardwarebereich bietet ein breites Spektrum an Geräten mit kostengünstigen, bis hochpreisigen Angeboten, sowohl mit einfacher bis qualitativ hochwertiger, technischer Ausstattung. Benutzer haben die Möglichkeit, ihre Geräte weitestgehend frei zu gestalten und einzustellen. Für die Installation von neuen Applikationen sind diese auch nicht an einen einzigen Store gebunden und können Apps auch 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 Unternehmen Apple. Dieses ist ein Derivat von Mac OS X, welches selbst auf Unix basiert. Es wird ausschließlich von Apple entwickelt und ist somit nur auf den eigenen Geräten iPhone, iPad und iPod touch zu finden. Mit der Entwicklung wurde unter externer und interner Geheimhaltung 2005 begonnen und das Resultat der Öffentlichkeit zum ersten Mal Anfang 2007 vorgestellt. Bis zur Version 4.0 wurde iOS mit dem Namen iPhoneOS betitelt. Das Konzept und Design ist schwerpunktmäßig auf hohe Benutzerfreundlichkeit und Funktionalität ausgelegt.
Durch die geschlossene Struktur des Systems sind eigene Derivate nicht möglich. Benutzer sind für den offiziellen Bezug von Applikationen auf Apples App Store angewiesen. Bei der Wahl der Hardware ist man auf die Produktpalette von Apple angewiesen, welche jährlich eine neue Generation veröffentlicht. Die Personalisierung der Geräte ist nur bedingt möglich, da Anbieter von Drittsoftware keinen Zugriff auf das System haben und Anwendungen nur offiziell über den eigenen App Store bezogen werde können. Dies bietet jedoch den Vorteil einer Qualitätssicherung durch Apple, da Applikationen vor der Veröffentlichung einer 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

Entwickler Microsoft stellt seit dem Jahr 2000 Betriebssysteme für mobile Geräte her. (Fran Berkman 2012) Seitdem hat sich die Namensgebung von Windows Mobile, über Windows Phone, bis zum aktuellsten Windows 10 Mobile vorgearbeitet. Um im allgemeinen Bezug nicht zwischen den Namen hin und her zu wechseln, wird in dieser Arbeit, wenn mobile Windows Systeme erwähnt werden, der Name Windows

Phone (oder WP) benutzt. Die frühen Versionen von Windows Phone, also Windows Mobile und Windows Phone 7 stammen noch von dem Windows CE Kernel ab, wobei die aktuellen Versionen, Windows Phone 8 und Windows 10 Mobile, Derivate des Windows NT Kernels sind. Mit dem neuesten Ableger, Windows 10 Mobile, verspricht Microsoft eine homogene Kommunikations- und Anwendungsstruktur zwischen allen Geräten die mit diesem System betrieben werden. Dazu zählen nicht nur Smartphones und Tablets, sondern auch Notebooks, Desktop PCs und die Spielkonsole Xbox One. (Microsoft 2015)
Microsoft verfolgt mit Windows Phone eine ähnlich geschlossene und proprietäre Struktur wie der Konkurrent Apple. Eigene Derivate des Systems sind also nicht offiziell möglich. Auch die Benutzer müssen für neue Anwendungen auf das Angebot des Windows Stores zurückgreifen. Jedoch will Microsoft Entwicklern die Möglichkeit bieten, zukünftige Anwendungen universell verfügbar zu machen, das diese auf allen Windows Systemen nutzbar sind. Microsoft arbeitet außerdem an einer Technik, die bestehende 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, jedoch bisher 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

(Reddit 2015)

3 Native Softwareentwicklung

Softwareentwicklung für ein bestimmtes System wird als nativ (lat.: angeboren, natürlich) bezeichnet. Hier sind Dateiformate, Programmiersprachen, Hardware, Entwicklungsumgebungen und Kompilierung genau an die Zielplattform angepasst. Nativer Code ist in der Lage alle individuellen Eigenschaften einer Zielplattform anzusprechen, ohne dabei eine eventuelle Portierbarkeit zu berücksichtigen. (John Daintith 2004) Welche Anforderungen iOS, Android und Windows Phone bezüglich nativer Entwicklung voraussetzen, soll in diesem Kapitel näher erläutert werden.

3.1 Systemvoraussetzungen

Um Applikationen für eine bestimmte Zielplattform zu entwickeln, werden unter Umständen Voraussetzungen an das Betriebssystem des Entwicklers gestellt.

3.1.1 Android

Die Entwicklung von Android Applikationen ist an kein bestimmtes System gebunden. Somit lassen sich diese unter Windows, OS X und Linux Systemen entwickeln. Dies wird unter anderem durch die Eigenschaften der Programmiersprache Java ermöglicht. Windows Benutzer sollten mindestens Windows XP nutzen. Darüber hinaus können alle aktuelleren Versionen genutzt, wobei alle 32-Bit Editionen unterstützt werden. Ab Windows 7 wird 64-Bit ebenfalls unterstützt. Mac Systeme werden ab OS X 10.5.8 von den offiziellen Entwicklungswerkzeugen unterstützt. Um auf einem Linux System zu entwickeln, kann man dies beispielsweise unter Ubuntu ab Version 8.04 tun. Bei 64-Bit Versionen ist es notwendig, dass diese fähig ist 32-Bit Anwendungen auszuführen. Da die Auswahl an Linux-Distributionen sehr umfangreich ist, wird an dieser Stelle auf diese nicht weiter eingegangen. (Sue Smith 2013)

3.1.2 iOS

Für den reinen Schreibvorgang 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 der Zielhardware zu testen. Für Entwickler die kein OS X System ihr Eigen nennen oder keinen Zugang zu solcher haben, besteht die Möglichkeit einen Cloudservice zu nutzen. *MacinCloud* bietet eine cloudbasier-

te 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 passendes Windows System vorausgesetzt. Jedoch sind manche Voraussetzungen leider etwas undurchsichtig. Das bedeutet, dass für die Entwicklung einer Windows Phone 8 App mindestens ein Windows 8 Betriebssystem benötigt wird. Außerdem wird das SDK 8.0 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 Werkzeuge, Anwendungen und bestenfalls eine aktuelle Dokumentation, um für eine bestimmte Zielplattform zu entwickeln. Auch sind sie notwendig, um geschriebenen Code zu interpretieren und kompilieren. Um die aktuellste Version eines mobilen Systems zu unterstützen, muss das SDK auf ebenso aktuellen Stand sein.

3.2.1 Android Versionen

Android Versionen sind nach süßen Leckereien benannt und dem Anfangsbuchstaben nach alphabetisch aufsteigend. (Abb.3.1) Das aktuelle SDK ist frei im Netz verfügbar. Dieses kann einzeln geladen werden oder direkt in Verbindung mit Android Studio.

Codename	Version	API Level	Erscheinungsdatum
Marshmallow	6	23	5. Oktober 2015
Lollipop	5.1.x	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.x	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 diese überwiegend intern genutzt werden. (Abb.3.2) 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 in Java entwickelt. Demnach ist es notwendig vorab eine aktuelle Java Version (JDK) zu installieren. Diese wird von dem Unternehmen Oracle, mit der aktuellen Version 8 Update 66, vertrieben. (Oracle - Java SE 2015)

3.3.2 iOS

Die primäre Programmiersprache für iOS und OS X ist derzeit die objektorientierte Erweiterung von C, Objective-C. Diese befindet sich aktuell in der Version 2.0. Objective-C wurde weiterhin von der Sprache Smalltalk beeinflusst und leitet zum Beispiel die Syntax dessen Objekteigenschaften ab. Trotzdem besteht weiterhin die Nähe zur C Syntax, in Bezug auf nicht-objektorientierter Operationen. (Mac Developer Library 2014)
Alternativ kann die relativ junge Sprache Swift eingesetzt werden. Diese ist ebenfalls objektorientiert und in der aktuellen Version 2.0. Im Dezember 2015 wurde der Quellcode als Open Source zur Verfügung gestellt. Swift soll Objective-C allerdings nicht ersetzen, sondern eine weitere Möglichkeit darstellen. Die beiden Sprachen sind miteinander kompatibel, so dass es möglich beide in demselben Projekt zu nutzen. (Apple Developer 2015)

3.3.3 Windows Phone

Für die Benutzeroberfläche wird hauptsächlich die Sprache XAML (Extensible Application Markup Language) genutzt. Um die Logik zu erstellen, hat der Entwickler die Freiheit C#, Visual Basic, JavaScript oder C++ nutzen.

3.4 Entwicklungsumgebungen

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

3.4.1 Android

Android empfiehlt das eigene Android Studio, welches die offizielle IDE für Android Entwicklung darstellt und zusätzlich das aktuelle SDK mitliefert. Android Studio basiert auf der IDE IntelliJ IDEA und ist frei verfügbar.(Android Develop Tools 2015) Alternativer Vorgänger ist die quelloffene IDE Eclipse.

3.4.2 iOS

Xcode ist die IDE von Apple, ohne die keinerlei Kompilierung von iOS Projekten möglich ist. Diese befindet sich derzeit in der stabilen Version 7 und liefert das benötigte SDK, so wie einen umfangreichen Simulator mit allen mobilen Apple Geräten. Xcode ist verpflichtend, wenn die geschriebene Anwendung auf einer realen Hardware getestet werden soll. Seit dem 3. Quartal 2015 ist dafür auch keine kostenpflichtige Entwicklerlizenz nötig, sondern fällt nur noch bei einem Release im App Store an.

(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 WP Projekte eine Visual Studio IDE voraus, welche für Entwicklungen außerhalb eines Unternehmens kostenfrei ist.

3.5 Zusammengefasste Übersicht

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

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

Tabelle 3.4: Unterschiede zwischen Android, iOS und Windows Phone

4 Plattformübergreifende Entwicklung

Die Entwicklung von Software-Produkten 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)

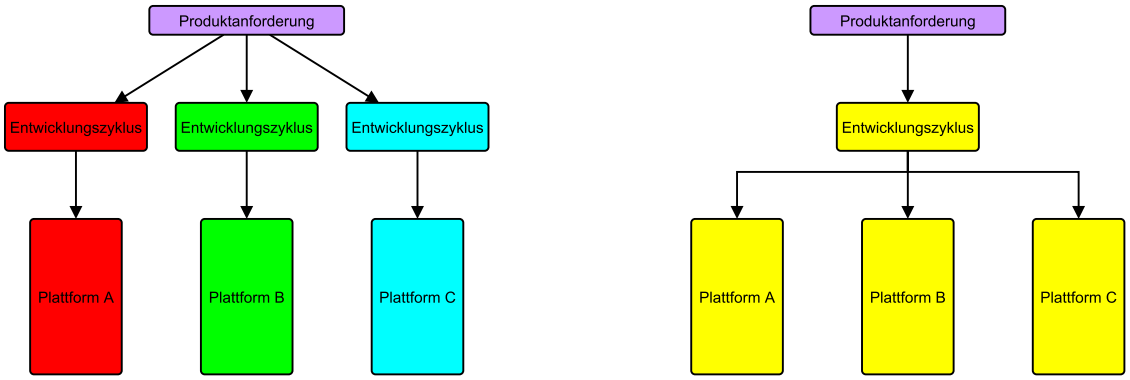

Abbildung 4.1: Traditionelle und plattformübergreifende Entwicklungsmodelle

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 es redundante Prozesse oder für jede Plattform einen eigenen

Datenspeicherordner. Die Reduzierung von Komplexitäten kann auch bis zur „Verdummung“ 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 in verschiedene spezifische Betriebssystemversionen kompiliert wird. 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 diese Verfahren anwenden, kann man als plattformunabhängig, genauer gesagt plattformübergreifend bezeichnen, da sie die unterstützten Systeme gleich wertet und keines bevorzugt.(techopedia 2015)

Die Entwicklung von plattformübergreifenden Anwendungen auf mobilen Systemen wird in sechs verschiedene Ansätze kategorisiert. (Abb.4.2)
In den folgenden Unterkapiteln werden die einzelnen Ansätze und Unteransätze näher betrachtet. Die Analyse und Betrachtung basiert auf den Informationen der Ausaurbeitung von (El-Kassas, Wafaa S. & Abdullah, Bassem A. & Yousef, Ahmed H. & Wahba, Ayman M. 2015).

4.2.1 Kompilierung

Kompilierung(Compilation) ist ein Ansatz mit zwei Unterkategorien:

- Cross-Compiler
- Trans-Compiler

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

Man spricht von einem **Cross-Compiler**, wenn das System des Compilers unterschiedlich zu dem System ist, auf dem der kompilierte Code läuft. Die Zielsysteme können Betriebssysteme, Prozessoren oder eine Kombination aus beiden sein. Abb. 4.3 stellt eine von XMLVM gebotene Lösung zu Cross-Compiling dar. Dies nutzt XML

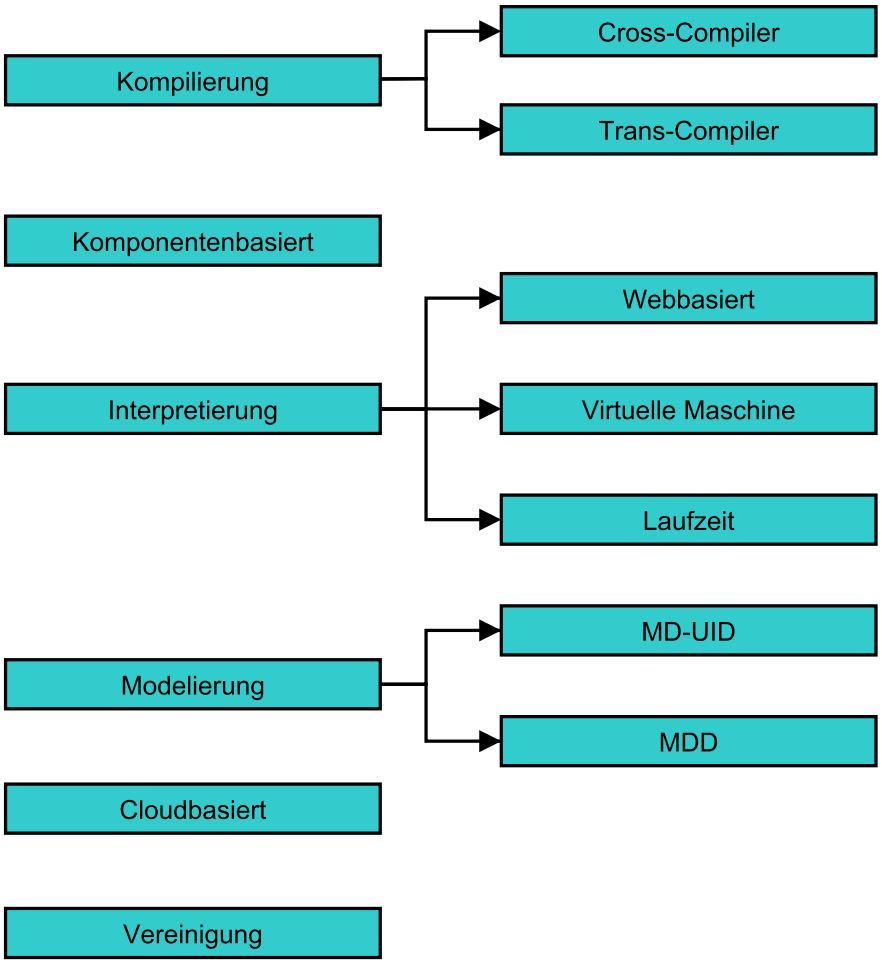

Abbildung 4.2: Haupt- und Nebenansätze zur Entwicklung von mobilen plattformübergreifenden Anwendungen

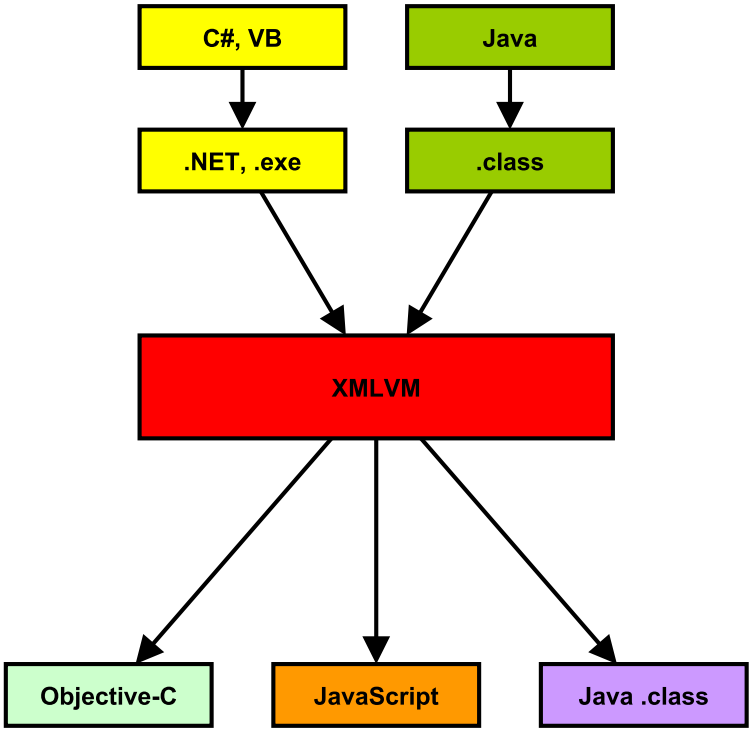

Abbildung 4.3: XMLVM Prozess mit Java oder .NET Quellcode (XMLVM 2011)

für die Frontend Darstellung und für die Verarbeitung des Bytecodes eine virtuelle Maschine(VM).

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 diese bei der Übersetzung nicht unterstützt. Auch ist der Code durch die automatisierte Erzeugung in der Regel nur schwer lesbar. Es besteht auch eine Abhängigkeit zu regelmäßigen Updates, um die Änderungen der Quell- und Zielsysteme aktuell zu halten.

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, dass die Servicedienste spezifiziert, welche von anderen Komponenten genutzt werden können. Die Kommunikation findet ausschließlich über die Schnittstellen statt, so dass eine Komponente keinerlei Informationen über den Aufbau einer anderen benötigt.

Ein beispielhafter Lösungsansatz ist der **Component-Based Mobile Web Application of Cross-Plattform**. Dieser komponentenbasierte Ansatz 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 vereinfacht 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 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 wie einer CPU, Stack, Register und ein korrespondierendes 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 Abb. 4.4 wird der Interpretierungsablauf der von Android bekannten Dalvik VM dargestellt.

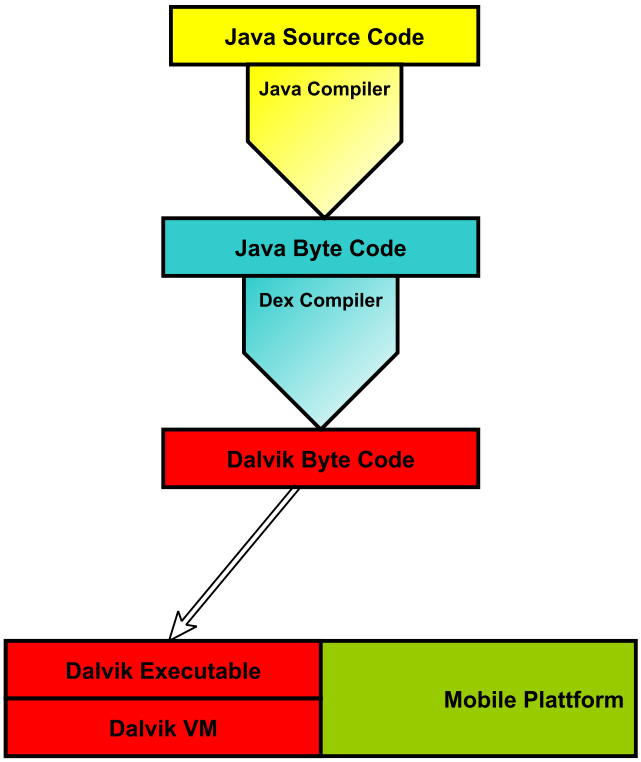

Abbildung 4.4: Ablauf des Dalvik VM Interpreter

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. Abb. 4.5 zeigt die Kommunikation und Interpretierung des Webbasierten Interpreters PhoneGap von Adobe.

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

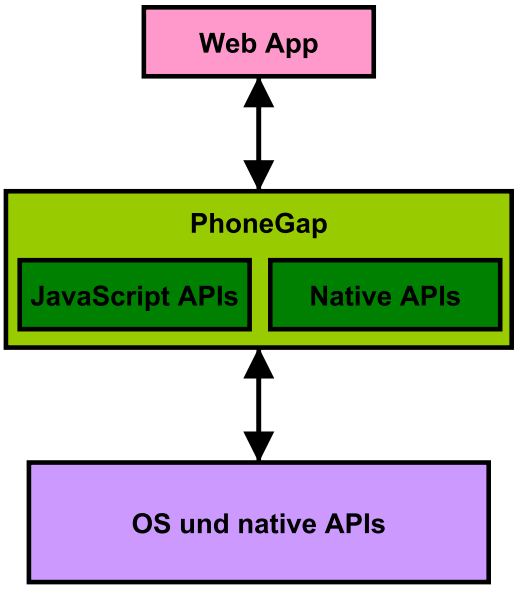

Abbildung 4.5: Vereinfachter Ablauf des PhoneGap Interpreters von Adobe

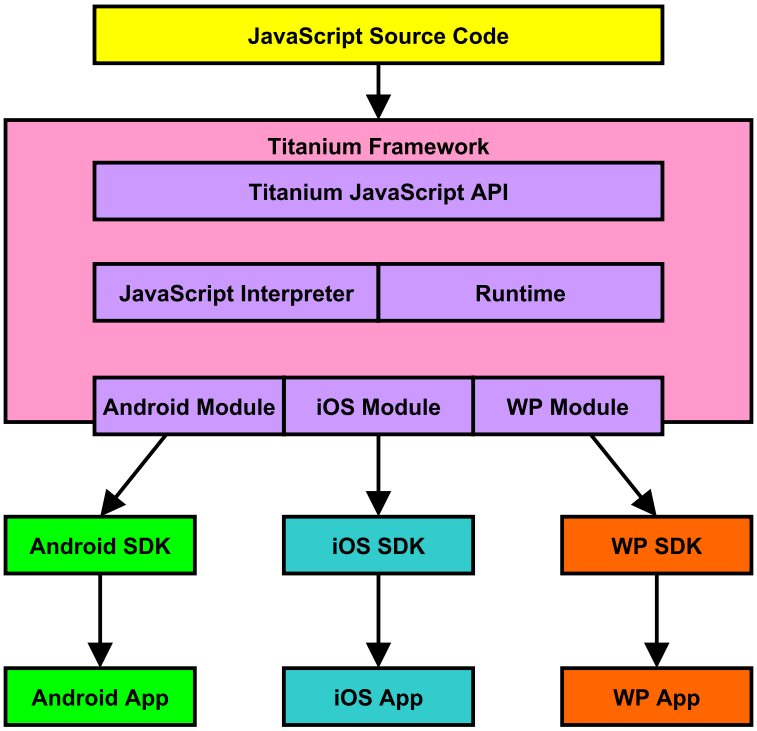

Abbildung 4.6: Ablauf des Titanium Interpreters von Appcelerator

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 App-Logik unterschieden. Für die Generierung existieren zwei Strategien.

- Eine Generierung zur Laufzeit der App, die eine Websysteme adaptiert und basiert auf Anfrage- und Antwortprotokollen (request / response). 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, also ob eine dauerhafte Verbindung bestehen soll oder zu einem selbst bestimmten Zeitpunkt synchronisiert werden soll.

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 geschieht die Verarbeitung der Anwendung nicht lokal auf dem Gerät, sondern auf einem Cloudserver. Dabei werden einige Cloudeigenschaften verwendet, wie Flexibilität, Virtualisierung, Sicherheit und dynamisches Management. Die Clientanwendung ist dabei weitest möglich reduziert, da diese nur Basisprozesse benötigt. Dies wird Thin-Client genannt, da wie in Abb. 4.7 nur Ein- und Ausgabe verarbeitet werden müssen. Cloudbasierte Anwendungen sollen dadurch besonders energieeffizient sein.
Da zur Zeit der Bearbeitung zu diesem Ansatz keine direkten Realisierungen gefunden worden, sondern überwiegend der theoretische Aufbau einer solchen Applikation, wird dieser Teil nicht weiter vertieft.

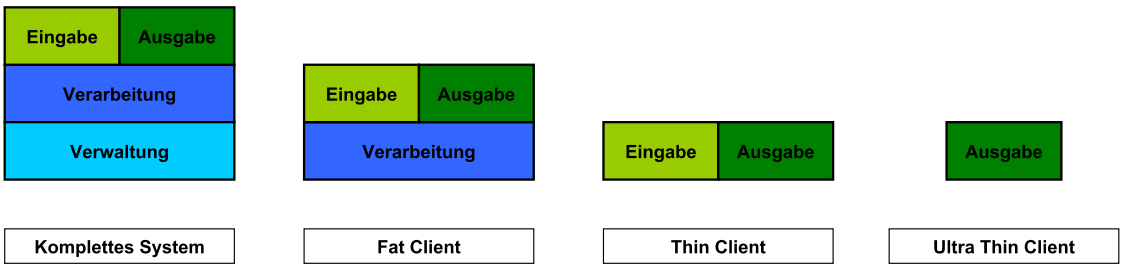

Abbildung 4.7: Aufgaben von Client-Anwendungen

4.2.6 Vereinigung

Dieser Ansatz versucht die besten Eigenschaften verschiedener Ansätze zusammenzuführen, 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 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 Universalsprache basieren. Diese Sprache wird der App als zusätzliche Kommunikationsschicht und Schnittstelle hinzugefügt, um die Komponenten und deren Methoden anzusprechen.(Abb. 4.8) Der Entwickler implementiert nur eine minimale Struktur 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 Funktionen, ohne Berücksichtigung der plattformspezifischen.

Integrated Cross-Platform Mobile Development (ICPMD) ist eine weitere Lösung die auf dem Merge-Prinzip aufbaut und drei Verwendungsszenarios unterstützt. Diese Szenarios sind, wie in Abb. 4.9 dargestellt, abhängig von dem gegebenen Input.

Der Entwickler hat...

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

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

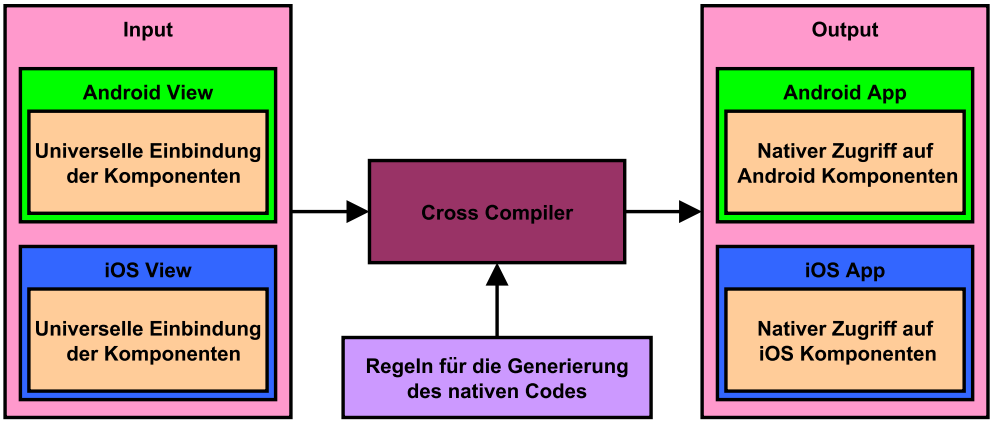

Abbildung 4.8: Funktion eines komponentenbasierten Mergeansatz

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

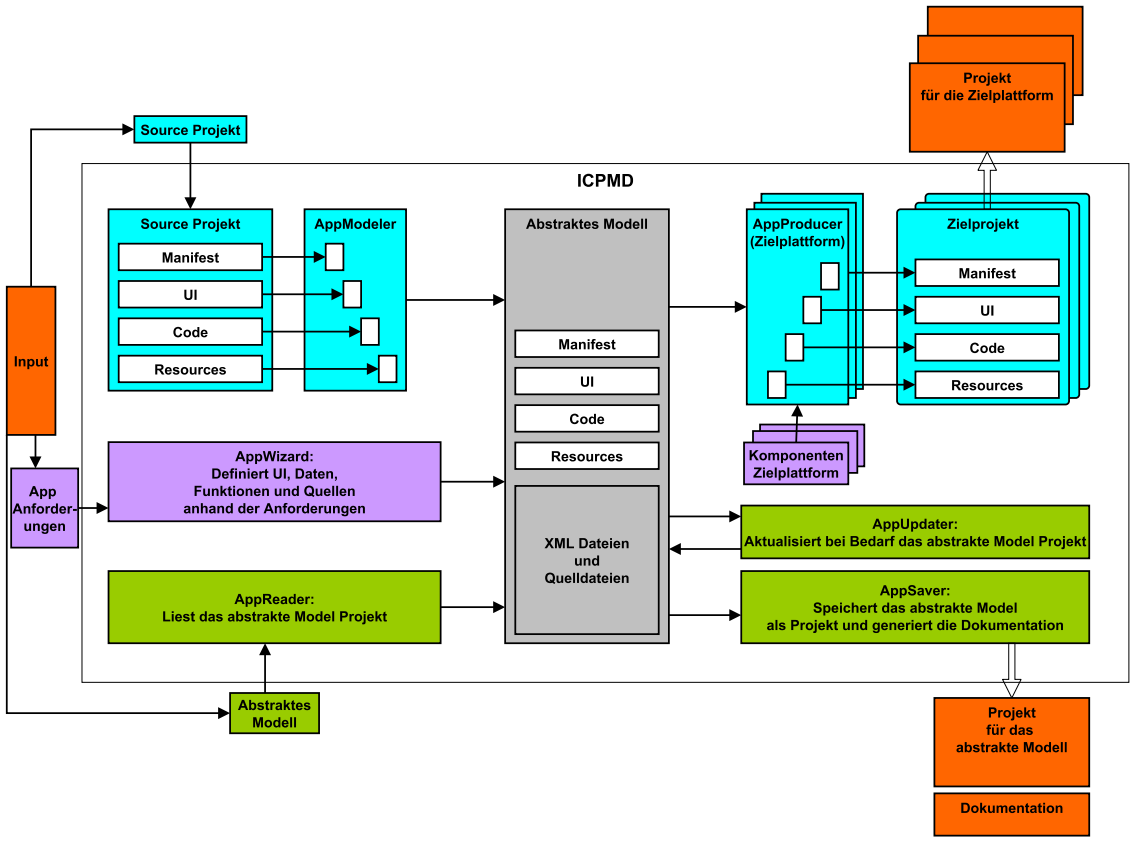

Abbildung 4.9: Drei Szenarien bei ICPMD

4.3 Übersicht der Ansätze

Tabelle 4.1 fasst die vorigen Ansätze zur plattformübergreifenden Entwicklung in Kurzform zusammen, benennt die jeweiligen Vor- und Nachteile, sowie die Namen 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 native und besitzt somit die Vorteile einer nativen App.	Die Zuordnungen zwischen zwei Sprachen ist sehr aufwendig, so dass hauptsächlich die gemeinsamen Eigenschaften berücksichtigt werden.	MoSync, Corona, Neomades, XMLVM
	Trans-Compiler	Kann genutzt werden, um veraltete Applikationen und deren veralteter Code auf eine neuere Version derselben Sprache zu übersetzen. Wiederverwertung eines existierenden Quellcodes durch Trans-Kompilierung auf eine andere Plattform.Die resultierende Applikation ist native und besitzt somit die Vorteile einer nativen App.	Konzentriert sich ausschließlich auf die gemeinsamen APIs der Quell- und der Zielsprache. Benötigt regelmäßige Aktualisierungen, um die Änderungen der APIs in Quell- und Zielsprache 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
Interpretation	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.	Die Perfomance beim Laden der App ist gering, da der Interpretierungsvorgang bei jeder einzelnen Ausführung stattfindet.	Titanium, Xamarin
Modellierung	MD-UID	Spart Entwicklungszeit durch Generierung des UI-Codes. Nützlich für Prototyping, durch schnelle UI-Entwicklung und dadurch frühe Evaluierung der Benutzerfreundlichkeit.	Muss sich bei den einzelnen Plattformen auf ähnliche Benutzerschnittstellen konzentrieren.	XMobile
	MDD	Die Modellierungssprache ist effektiv, um Anforderungen zu definieren. Der Fokus liegt auf der Funktionalität und nicht an der technischen Implementierung.	Kann existierenden, nativen Quellcode nicht verarbeiten.	JSAF, MD2, Jelly, AppliDE
Cloudbasiert		Verarbeitungsprozesse werden in eine Cloud 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

In der Thesis „*Plattformabhängige und –unabhängige Entwicklung mobiler Anwendungen am Beispiel von Geo-Wikipedia-App*“ (Vehse, Benjamin 2014) wird ebenfalls die plattformübergreifende Entwicklung analysiert, jedoch liegt hier nicht der Fokus auf Game-spezifischen Applikationen und Entwicklungswerkzeugen. Hier wird in Kapitel 2.2 auch auf die verschiedenen Herangehensweisen eingegangen und deren Resultate klassifiziert. Weiterhin wird eine Auswahl der bekannteren Cross-Plattform Entwicklungstools (PhoneGap, Xamarin, Appcelerator) analysiert. Diese Arbeit setzt den Schwerpunkt auf Frameworks und Engines zur Spieleentwicklung und geht von daher nicht weiter auf die genannten und vorangegangenen Entwicklungswerkzeuge ein.

5 Plattformübergreifende Frameworks zur Spieleentwicklung

5.1 Gamespezifische Frameworks und Engines

5.1.1 Cocos2D-X

5.1.2 Libgdx

5.1.3 Unity3D

5.1.4 Weitere Frameworks

5.2 Entwicklungsumgebungen

5.2.1 Unterstützte IDEs

5.2.2 Systembedingte Einschränkungen

6 Gegenüberstellung der Frameworks

6.1 Zielplattformen

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

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

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

6.2 Programmiersprachen

libGDX nutzt einzig und allein Java für die Entwicklung, was für reine Android Projekte ein großer Vorteil ist. Denn dadurch müsste man bei der Kompilierung kaum Kompromisse eingehen.

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

Lua ist eine schnelle, von der Syntax simpel gehaltene Skriptsprache, mit objektorientierten Eigenschaften. Die geschriebenen Skripte werden durch einen Interpreter in Bytecode übersetzt und ist kompatibel mit der Sprache C. Lua befindet sich derzeit in Version 5.3 und wird häufig in der Spieleentwicklung eingesetzt.

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

Für Entwickler die beispielsweise Erfahrungen im Webbereich besitzen, bietet die Verwendung von JavaScript einen weiteren Einstieg in Cocos2D-X.

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

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

"*When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck.*"

Beim Duck-Typing werden Objekte nicht durch ihre Klasse typisiert, sondern durch die vorhandenen Attribute und Methoden. Daher findet die Typisierung erst zur Laufzeit durch den Interpreter statt.

Oft wird UnityScript in der Unity Community, aber auch von dem Unternehmen selbst, fälschlicherweise mit JavaScript gleichgesetzt, als wären die beiden Sprachen äquivalent. Auch wenn syntaktische Ähnlichkeiten bestehen, gibt es große semantische Unterschiede. UnityScript kann, wie die meisten objektorientierten Programmiersprachen, Klassen definieren und daraus Objekte erstellen. JavaScript hingegen besitzt

zwar ebenfalls objektorientierte Eigenschaften, wobei diese aber sogenannte *Prototypes* verwendet. Dies sind Objekte die zuerst von einer Funktion definiert werden und danach mit dem Schlüsselwort *new* instanziiert werden. Nach der Instanziierung ist es möglich, das Objekt um zusätzliche Eigenschaften zu erweitern. Des Weiteren kann UnityScript im Gegensatz zu JavaScript, Klassen, Objekte, Funktionen und Variablen mit Zugriffsmodifikatoren / Sichtbarkeiten versehen. UnityScript wurde speziell für die Unity3D Engine konzipiert und ist proprietär, was es schwierig macht, genaue Spezifikationen zu finden.

6.3 Unterstützung von 2D und 3D

6.4 Zugriff auf Hardware

6.5 Free- und Pro- Versionen

6.6 Einfluss auf Einstellungen

6.7 Zusätzlich benötigte Software

6.8 Aktualität - Versionen - Community

6.9 Zukunftsaussichten

7 Kosten-Nutzen Vergleich

8 Konzeption und Implementierung einer Test-Applikation

8.1 Ideen

8.2 Anforderungen

8.3 Verwendete Frameworks und Engines

8.4 Verwendete APIs und SDKs

8.5 Assets und deren Verwendung

9 Analyse messbarer Metriken

10 Vergleich der Messprotokolle

11 Fazit

Abbildungsverzeichnis

2.1 Prognose zu den Marktanteilen der Betriebssysteme am Absatz vom Smartphones weltweit in den Jahren 2015 und 2019 8
2.2 Marktanteile der mobilen Betriebssysteme am Absatz von Smartphones in ausgewählten Ländern von August bis Oktober 2015 8
2.3 Anzahl der angebotenen Apps in den Top App-Stores im Mai 2015 . . 9
2.4 Anteil der im Google Play Store weltweit am häufigsten heruntergeladenen Apps nach Kategorien im Februar 2014 10
2.5 Ranking der Top-Kategorien im App Store im Dezember 2015 10

4.1 Traditionelle und plattformübergreifende Entwicklungsmodelle 21
4.2 Haupt- und Nebenansätze zur Entwicklung von mobilen plattformübergreifenden Anwendungen 23
4.3 XMLVM Prozess mit Java oder .NET Quellcode 24
4.4 Ablauf des Dalvik VM Interpreter 26
4.5 Vereinfachter Ablauf des PhoneGap Interpreters von Adobe 27
4.6 Ablauf des Titanium Interpreters von Appcelerator 27
4.7 Aufgaben von Client-Anwendungen 29
4.8 Funktion eines komponentenbasierten Mergeansatz 30
4.9 Drei Szenarien bei ICPMD 31

Tabellenverzeichnis

3.1 Android Versionen und ihr Erscheinungsdatum 16
3.2 iOS Versionen und ihr Erscheinungsdatum 17
3.3 Windows Phone Versionen und ihr Erscheinungsdatum 18
3.4 Unterschiede zwischen Android, iOS und Windows Phone 20

4.1 Übersicht aller Ansätze 32

6.1 Unterstützte Zielplattformen der Frameworks 35

Literaturverzeichnis

Android Studio Overview, http://developer.android.com/tools/studio/index.html, letzter Zugriff: 24.11.2015

Codenames, Tags, and Build Numbers in the history of Android, https://source.android.com/source/build-numbers.html, etzter Zugriff: 24.11.2015

Swift - Overview, https://developer.apple.com/swift/, letzter Zugriff: 28.12.2015

Blogs. Windows - Windows Store Trends - September 2015, https://blogs.windows.com/buildingapps/2015/10/12/windows-store-trends-september-2015/, letzter Zugriff: 15.12.2015

Cocos2D-X - Developers Manual, http://www.cocos2d-x.org/wiki/Cocos2d-x, letzter Zugriff: 29.12.2015

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

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

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

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

AppCode, https://www.jetbrains.com/objc/, letzter Zugriff: 28.12.2015

A Dictionary of Computing - native software, http://www.encyclopedia.com/doc/1O11-nativesoftware.html, letzter Zugriff: 24.11.2015

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

About Objective-C, https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html/, letzter Zugriff: 28.12.2015

MacinCloud, http://www.macincloud.com/, letzter Zugriff: 28.12.2015

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

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

Microsoft - Windows 10 Features, https://www.microsoft.com/de-de/windows/features, letzter Zugriff: 25.11.2015

Microsoft-Emulator für Windows 10 Mobile, https://msdn.microsoft.com/library/windows/apps/mt162269.aspx, letzter Zugriff: 28.12.2015

Members of the Open Handset Alliance, http://www.openhandsetalliance.com/oha_members.html, letzter Zugriff: 24.11.2015

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

Overview of Android by the Open Handset Alliance, http://www.openhandsetalliance.com/android_overview.html, letzter Zugriff: 24.11.2015

Java SE Development Kit 8 Downloads, http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html, letzter Zugriff: 24.11.2015

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

Reddit - Pros and cons of Windows phone, and why should I buy one instead of an android or a iPhone?, https://www.reddit.com/r/windowsphone/comments/3h21lf/pros_and_cons_of_windows_phone_and_why_should_i/, letzter Zugriff: 25.11.2015

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

Android SDK Requirements, http://code.tutsplus.com/tutorials/android-sdk-requirements--mobile-20086, letzter Zugriff: 24.11.2015

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

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

iOS Firmwares, https://www.theiphonewiki.com/wiki/Firmware, letzter Zugriff: 24.11.2015

Unity3D - Public Relations, https://unity3d.com/public-relations, letzter Zugriff: 29.12.2015

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

Übersicht von allen Android Versionen mit Veröffentlichungsdatum, https://de.wikipedia.org/wiki/Liste_von_Android-Versionen, letzter Zugriff: 24.11.2015

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

Microsoft Windows Phone 7, https://de.wikipedia.org/wiki/Microsoft_Windows_Phone_7, letzter Zugriff: 28.12.2015

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

XMLVM - Overview: Toolchain, http://xmlvm.org/toolchain/, letzter Zugriff: 25.12.2015

Requirements for Windows Phone development, http://help.yoyogames.com/entries/23355146-Requirements-for-Windows-Phone-development, letzter Zugriff: 28.12.2015

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

Ort, Datum Sebastian Bohn