2 Herausforderungen zur Entwicklung mobiler Apps

Zur systematischen, strukturierten und ingenieurmäßigen Entwicklung mobiler Apps sind vielfältige Herausforderungen, Besonderheiten und Restriktionen zu beachten und zu berücksichtigen. In diesem Kapitel werden diese Herausforderungen und Besonderheiten herausgearbeitet und die sich daraus ergebenden Konsequenzen für ein softwaretechnisches Mobile App Engineering erläutert.

2.1 Oberflächengetriebene Konzeption und Entwicklung

Die Entwicklung mobiler Apps ist stark oberflächengetrieben. Somit steht bei der Konzeption und dem Entwurf einer mobilen App immer die softwareergonomische Qualität, Benutzungsfreundlichkeit und Gebrauchstauglichkeit – im Englischen auch als *Usability* bezeichnet – im Vordergrund. Zudem sollte sich eine mobile App möglichst nahtlos und passgenau in die Erlebniswelt des Benutzers einfügen und hierbei zu möglichst positiven Benutzungserlebnissen, der sogenannten *User Experience (UX),* führen.

2.1.1 Usability

Usability wird in der ISO 9241-11 relativ sachlich und funktionsorientiert definiert. Hierbei werden die komplexen gegenseitigen Abhängigkeiten von Anwendungskontext, Benutzereigenschaften und der Softwareanwendung fokussiert, die eingesetzt wird, um bestimmte Ziele zu erreichen und Aufgaben des Benutzers zu erfüllen. Gemäß dieser ISO-Norm weisen Softwareanwendungen eine hohe Usability auf, wenn sie von den Benutzern einfach erlernt und effizient verwendet werden können und die Benutzer ihre beabsichtigten Ziele und Aufgaben zufriedenstellend ausführen können (vgl. [Richter & Flückiger 2016, S. 10]). Spaß, Vergnügen oder gar Begeisterung bei der Bedienung und Anwendung stehen hierbei nicht im Vordergrund. Zudem gibt es weitere Qualitätskriterien wie zum Beispiel die Anordnung von Interaktionselementen, die Anzahl notwendiger Gesten bzw. Taps, um eine Aufgabe zu erledigen, oder die Verständlichkeit der angezeigten Beschreibungen und Dialoge (vgl. [Richter & Flückiger 2016, S. 10]).

Diese eher nüchtern-sachliche und auf die Funktionalität ausgerichtete Definition von Benutzbarkeit bzw. Usability war oftmals angemessen für geschäftliche Softwareanwendungen für Desktop-Computer (vgl. [Richter & Flückiger 2016, S. 12]).

2.1.2 User Experience

Allerdings hat sich die Welt durch den großen Erfolg, die vielfach hohe Qualität der User Experience und das ansprechende Interaktionsdesign mobiler Apps sowie die starke Wettbewerbsdynamik ein gutes Stück verändert: Die zahlreichen schnell und einfach benutzbaren mobilen Apps sowie die dabei erzielten positiven Benutzungserlebnisse und -erfahrungen haben die Erwartungshaltung der Benutzer deutlich erhöht. Damit einhergehend wurde auch die Messlatte an die Qualität der grafischen Benutzungsoberfläche und das Interaktionsdesign einer mobilen App deutlich höher gelegt. In diesem Kontext spielen auf einmal subjektive Erlebnisse, positive Erfahrungen, Emotionen, Witz, Ästhetik und somit eher weiche Faktoren eine wichtige, wenn nicht zentrale Rolle, die den *entscheidenden* Unterschied zwischen einer millionenfach benutzten mobilen App und einem erfolglosen Ladenhüter im App Store ausmachen können (vgl. [Richter & Flückiger 2016, S. 12]).

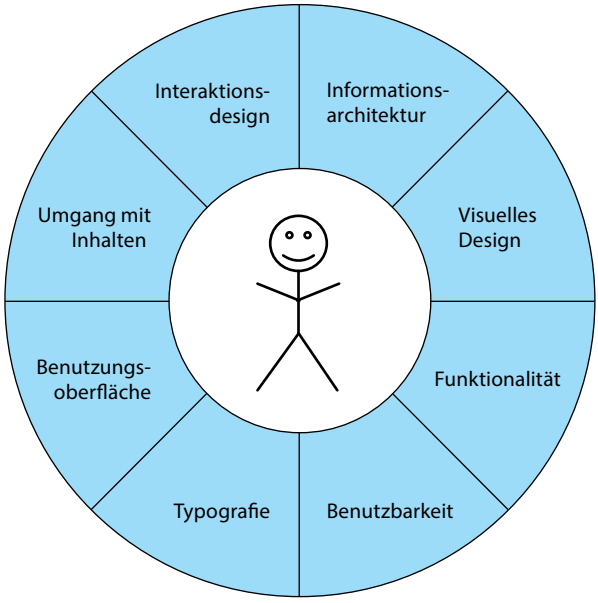

Abb. 2–1 *Bereiche und Einflussfaktoren der User Experience*[1]

1. Die Abbildung basiert auf:
http://usabilitygeek.com/wp-content/uploads/2013/07/user-experience-areas.jpg.

Der in diesem Kontext verwendete Begriff *User Experience (UX)* ist deutlich weiter gefasst als der Begriff *Usability*. Die User Experience wird im Forschungsbereich der *Human-Computer Interaction (HCI)* als eine vom Produkt – hier einer mobilen App – ausgelöste Bewertung des Benutzers verstanden. Es geht darum, wie bedeutend, wichtig, positiv und werthaltig der Benutzer die Verwendung eines Produkts *erlebt*. Auf diese Weise findet eine Verknüpfung von Handeln, Fühlen und Denken in der Interaktion mit einem Produkt bzw. einer mobilen App statt (vgl. [Hassenzahl et al. 2009]). Dabei spielen für eine gelungene, hochwertige User Experience neben der reinen Funktionalität auch Bereiche wie das visuelle Design, die grafische Benutzungsoberfläche, das Interaktionsdesign, die Informationsarchitektur und weitere Faktoren eine wichtige Rolle (siehe Abb. 2–1).

Das Konzept der User Experience betont die Orientierung auf die Benutzer und fordert somit auf, die rein funktionale Betrachtungsweise von Softwareanwendungen zu ergänzen und zu erweitern. Dabei werden vermehrt emotionale Faktoren bezüglich Design und Ästhetik einbezogen, die das Vergnügen, die Freude und die Begeisterung während der Benutzung beeinflussen und erhöhen können. Anstelle von pragmatischen, rein funktional ausgerichteten Produkten sollen möglichst positive Erfahrungen der Benutzer erzielt werden (vgl. [Hassenzahl et al. 2009]).

Vor diesem Hintergrund wird sich heutzutage kaum mehr ein Benutzer bloß mit guter Usability zufrieden geben. Vielmehr muss eine mobile App eine gute User Experience ermöglichen. Und vor diesem Hintergrund wird auch der Begriff Usability mehr und mehr durch den umfassenderen Begriff User Experience ersetzt (vgl. [Richter & Flückiger 2016, S. 12]).

2.1.3 User-Centered Design

Die Konzeption und das Design einer hochwertigen User Experience werden auch als benutzerzentriertes Design (*User-Centered Design*) bezeichnet (vgl. [Garrett 2012, S. 17]). Bei der Entwicklung mobiler Apps ist die zielgruppen- und benutzerspezifische Erhebung und Umsetzung der Anforderungen an die Gestaltung der grafischen Benutzungsoberfläche von zentraler, erfolgskritischer Bedeutung. Mobile Apps müssen schnell, einfach und intuitiv bedienbar sein und im besten Fall gute, positiv besetzte Erfahrungen beim Benutzer hinterlassen. Ein benutzerorientiertes Vorgehen bei der Konzeption umfasst Methoden, Techniken und Werkzeuge, um das angestrebte Benutzungserlebnis bei der Entwicklung einer mobilen App zu erreichen. Neben einem iterativen Vorgehen ist hierbei die Fragestellung zu beantworten, wer die genauen Ziel- und Benutzergruppen sind; und das sind bei mobilen Apps oftmals relativ heterogene Gruppen. Und ohne eine gute User Experience *und* Usability werden mobile Apps von den Benutzern nicht akzeptiert, schnell deinstalliert und durch ein Konkurrenzprodukt ersetzt.

Bei Softwareanwendungen im Allgemeinen und mobilen Apps im Speziellen ist es eine der wesentlichen Herausforderungen, unnötige Funktionen und Komplexität zu vermeiden. Daher sollte der Funktionsumfang einer mobilen App auf ein für den Benutzer ideales Minimum reduziert werden, um damit die Funktionalität der mobilen App – und somit auch das Benutzungserlebnis insgesamt – zu optimieren. Die mobile App soll den Benutzer in der Ausführung seiner Ziele und Aufgaben optimal unterstützen und wird genau dafür konzipiert und entwickelt. Die Reduktion auf das Wesentliche kommt nicht von alleine und die Entscheidung, welche Funktionen implementiert und welche weggelassen werden, erfordert in der Regel Aufwand an Kommunikation, Konzeption, Test und Evaluierung. Dieser Aufwand zahlt sich allerdings bei der Realisierung und dem Go Live der mobilen App über die Veröffentlichung im App Store aus[2].

2.1.4 Interaktionsdesign[3]

Das Interaktionsdesign definiert die Möglichkeiten zur Bedienung und Steuerung einer mobilen App, deren Verhalten sowie die Rückmeldungen an den Benutzer. Das Interaktionsdesign fokussiert hierbei die Herleitung und den Entwurf geeigneter Interaktionskonzepte für digitale Produkte und Anwendungen. Neben der Gestaltung der grafischen Benutzungsoberfläche steht dabei eine hochwertige User Experience im Vordergrund, die das Gesamterlebnis des Benutzers bei der Verwendung einer mobilen App umfasst. Usability Engineering, Interaktionsdesign und das Design der grafischen Benutzungsoberfläche sind somit eng miteinander gekoppelt und nicht trennscharf voneinander abzugrenzen.

Mittels benutzerorientierter Methoden werden die notwendigen Funktionen, Informationen und Abläufe definiert und mit Benutzern iterativ erprobt, überprüft und bewertet. Im Interaktionsdesign werden die Bedürfnisse der Benutzer in konkrete Entwürfe der grafischen Benutzungsoberfläche umgesetzt, und im *User Interface Design* werden die Konzepte auf der Zielplattform implementiert.

Die Art der grafischen Benutzungsoberfläche ist dabei von großer Bedeutung. Während die eingesetzten benutzerorientierten Methoden für unterschiedliche Technologien weitestgehend analog ablaufen, ist es für den Entwurf der grafischen Benutzungsoberfläche ein wesentlicher Unterschied, ob beispielsweise ein Datenverarbeitungssystem, ein mobiles Endgerät, ein Bedienpanel zur Maschinensteuerung oder ein Fahrkartenautomat entwickelt werden soll. Die folgenden Punkte sind hierbei unter anderem zu berücksichtigen:

---

2. Der zweite Absatz in Abschnitt 2.1.3 geht inhaltlich weitgehend auf [Richter & Flückiger 2016, S. 15] zurück.
3. Der Abschnitt geht inhaltlich weitgehend auf [Richter & Flückiger 2016, S. 191–193] zurück.

- Eingabe- und Ausgabemedien wie Stifte, Gesten und Tastatur, Bildschirme, Touchscreens, Tasten und Displays auf Geräten
- Zu verwendende Bedienelemente und Interaktionsprinzipien
- Struktur und Layout der grafischen Benutzungsoberfläche
- Aspekte wie Ergonomie, grafische Gestaltung, Industriedesign, technische Umsetzbarkeit, Corporate Design und Ästhetik

2.1.5 Multi-Touch-Displays

Die Multitouch-Displays von Smartphones und Tablets sowie deren Auflösungen sind in den letzten Jahren kontinuierlich größer geworden. Dennoch sind sie im Vergleich zu Notebook-Displays (ca. 11,6 Zoll bis 17,3 Zoll) und großen Desktop-Bildschirmen (20 Zoll bis 27 Zoll) rund vier- bis mehr als zwanzigmal kleiner.

Hersteller/Modell Technische Daten	Apple iPhone 7 Plus (6s Plus)	Apple iPhone 7 (6s)	Samsung Galaxy S7 edge	Samsung Galaxy S7
Größe	5,5"	4,7"	5,5"	5,1"
Bildschirmdiagonale	13,94 cm	11,94 cm	13,97 cm	12,92 cm
Display	Multitouch, Retina HD	Multitouch, Retina HD	Super-AMOLED-HD	Super-AMOLED-HD
Auflösung (in Pixel)	1920×1080 bei 401 ppi	1334×750 bei 326 ppi	2560×1440 bei 534 ppi	2560×1440 bei 577 ppi
Kontrastverhältnis	1300:1	1400:1	3578:1	3578:1
Länge (in mm)	158,2	138,3	150,9	142,4
Breite (in mm)	77,9	67,1	72,6	69,6
Höhe (in mm)	7,3	7,1	7,7	7,9
Gewicht (in g)	188 (192)	138 (143)	157	152

Tab. 2–1 *Technische Daten aktueller High-End-Smartphones*[4]

Zudem gibt es eine große Vielfalt von mobilen Endgeräten mit unterschiedlichen Displaygrößen. Das führt zu großen Herausforderungen bei der Gestaltung und dem Design der grafischen Benutzungsoberfläche sowie bei der Konzeption des Interaktionsdesigns, der Usability und der User Experience einer mobilen App.

4. Quellen: *www.apple.com* bzw. *www.samsung.com*.

2.1.6 Guidelines, Styleguides und Normen

Die herstellerspezifischen GUI-Guidelines und Styleguides spielen eine wichtige Rolle bei der Entwicklung mobiler Apps. Diese Richtlinien werden von Google für Android[5], von Apple für iOS[6] und von Microsoft für Windows[7] veröffentlicht. Die genaue Einhaltung dieser Gestaltungsrichtlinien ist wichtig, um die Usability und User Experience der mobilen App möglichst hochwertig und betriebssystemspezifisch zu gestalten. Zudem müssen diese Richtlinien befolgt werden, um das Review der mobilen App im Rahmen des Veröffentlichungsprozesses durch den App-Store-Betreiber erfolgreich zu bestehen.

Zudem müssen zur Gestaltung der grafischen Benutzungsoberfläche gültige Normen wie die DIN EN ISO 9241-110 »Ergonomie der Mensch-System-Interaktion« eingehalten werden. Diese Norm hat elementare Grundsätze der Dialoggestaltung definiert, die auch in Ihrem Mobile-App-Entwicklungsprojekt umgesetzt werden sollten. Dazu zählen die folgenden sieben Grundsätze, die auf mobile Apps angewendet wurden:

Aufgaben-angemessenheit	Die mobile App unterstützt die Erledigung der Aufgaben und den Arbeitsablauf der Benutzer.
Selbstbeschreibungs-fähigkeit	Die mobile App enthält Erläuterungen und ist ausreichend verständlich.
Steuerbarkeit	Der Benutzer kann den Dialogablauf beeinflussen.
Erwartungskonformität	Erwartungen, Eigenschaften und Gewohnheiten der Benutzer werden unterstützt.
Fehlertoleranz	Fehler erfordern keinen oder nur geringen Korrekturaufwand.
Individualisierbarkeit	Die mobile App kann an die individuellen Bedürfnisse angepasst werden.
Lernförderlichkeit	Die mobile App erfordert einen geringen Lernaufwand und unterstützt das Erlernen neuer Funktionen.

Tab. 2–2 *Grundsätze der Dialoggestaltung nach DIN EN ISO 9241-110*

2.1.7 Konsequenzen für das Mobile App Engineering

Um also hoch qualitative und benutzungsfreundliche mobile Apps zu entwickeln, die eine gute Usability und User Experience ermöglichen, ist es notwendig, die Benutzer zu verstehen, die die mobile App einsetzen und mit ihr interagieren. In diesem Zusammenhang sollten die Benutzerbedürfnisse, der Problem- und Einsatzbereich sowie der Anwendungskontext der mobilen App im Rahmen des Requirements Engineering frühzeitig identifiziert und analysiert werden. Zudem

5. Siehe auch *http://developer.android.com/design/index.html*.
6. Siehe auch *http://developer.apple.com/ios/human-interface-guidelines/overview/design-principles*.
7. Siehe auch *http://developer.microsoft.com/en-us/windows/apps/design*.

müssen die Tätigkeiten und Arbeitsabläufe erhoben, die idealen Funktionen festgelegt und auf dieser Basis eine adäquate, passgenaue und optisch ansprechende grafische Benutzungsoberfläche konzipiert werden. Hierbei hat es sich als äußerst zweckmäßig erwiesen, wenn die Situation und der Anwendungskontext des Benutzers beim praktischen Einsatz und der Bedienung einer mobilen App so realitätsnah wie möglich beschrieben werden.

Vor diesem Hintergrund wird zum Requirements Engineering in Kapitel 4 unter anderem eine *Contextual Inquiry* empfohlen, mit deren Hilfe der Anwendungskontext und die tatsächlichen Benutzerbedürfnisse unmittelbar erfasst werden können. Zudem werden *Personas* und *Szenarien* als praxisnahe, bewährte Methoden und Arbeitsmittel zur initialen Erhebung und Analyse der Requirements eingesetzt. Und auch zur Konzeption und zum Entwurf der grafischen Benutzungsoberfläche sowie des Interaktionsdesigns werden in Kapitel 5 praxisorientierte Arbeitsmittel wie Skizzen der grafischen Benutzungsoberfläche, Storyboards, *Low-Fidelity-Prototypen* mit *Wireframes* und *Mock-Ups* sowie *High-Fidelity-Prototypen* verwendet. Diese werden in Benutzer- und Expertentests erprobt und evaluiert, um möglichst hoch qualitative und benutzerorientierte mobile Apps entwickeln zu können.

2.2 Hardware- und softwareseitige Fragmentierung

Die starke hardware- und softwareseitige Fragmentierung stellt eine der größten Herausforderungen bei der Konzeption und Entwicklung sowie dem Test mobiler Anwendungen dar (vgl. [Wasserman 2010], [Joorabchi et al. 2013]).

2.2.1 Mobile Endgeräte und Betriebssysteme

Die verfügbaren mobilen Endgeräte mit variierenden Hardwarekomponenten, unterschiedlichen Konfigurationen und Displaygrößen sowie verschiedene mobile Betriebssysteme und Betriebssystemversionen führen zu großen Herausforderungen bei der Konzeption, dem Design, der Implementierung und dem Test mobiler Apps. Einerseits ist es nicht ausreichend, nur für *ein* mobiles Endgerät und *eine* mobile Betriebssystemversion zu entwickeln, um einen hohen Prozentsatz des Marktes zu versorgen. Andererseits stehen bei einer Entwicklung für *alle* verfügbaren mobilen Endgeräte sowie *alle* Betriebssystemversionen die exorbitanten Kosten für die Aufwände in den Bereichen Konzeption und Design, Implementierung und Test in keinem Verhältnis zum erwartbaren Nutzen.

Bei den mobilen Betriebssystemen unterscheiden sich Android und iOS im Wesentlichen dadurch, dass Android ein quelloffenes und im Gegensatz dazu iOS ein proprietäres mobiles Betriebssystem ist. Das bringt Vor- und Nachteile mit sich:

Während die Quelloffenheit von Android einerseits dafür sorgt, dass Entwickler von Android-Apps Zugriff auf alle Schnittstellen und API-Funktionalitäten bekommen und diese beliebig modifizieren können, müssen iOS-Entwickler sich mit den von Apple zur Verfügung gestellten Möglichkeiten zufrieden geben (vgl. [Joorabchi et al. 2013]), auch wenn diese in vielen Fällen sehr ausgereift sind.

Andererseits sorgt die Quelloffenheit von Android dafür, dass die Hersteller beliebige individuelle Änderungen oder Erweiterungen vornehmen können und sich die Android-Entwickler nicht darauf verlassen können, dass alle Standards und Syntaxvorgaben von Google im Hinblick auf Android auch von allen Endgeräteherstellern exakt umgesetzt werden (vgl. [Joorabchi et al. 2013]). Dieses Problem besteht bei Apple nicht, wobei die mobilen iOS-Endgeräte ohnehin von Apple hergestellt werden.

Somit stehen bei der Mobile-App-Entwicklung die Hardware sowie die Betriebssystemsoftware der mobilen Endgeräte deutlich stärker im Fokus, als dies bei konventionellen Softwareanwendungen für Desktop- und Notebook-Computer der Fall ist.

2.2.2 Gesten[8]

Ein mobiles Endgerät wird über ein Multitouch-Display bedient. Zur Bedienung sind zahlreiche Gesten verfügbar, die oftmals intuitiv einsetzbar, teilweise aber auch erst erlernt werden müssen. Hierbei sind die folgenden Gesten üblicherweise einsetzbar (vgl. [Knott 2016, S. 42]):

- **Touch**
  Der Touchscreen wird mit einem Finger kurz berührt.
- **Long Touch**
  Der Touchscreen wird mit einem Finger länger berührt.
- **Swipe**
  Der Finger wird über den Touchscreen bewegt.
- **Tap**
  Mit einem Finger wird einmal kurz auf den Touchscreen getippt.
- **Double Tap**
  Mit einem Finger wird zweimal kurz auf den Touchscreen getippt.
- **Drag**
  Der Finger wird über den Touchscreen bewegt, ohne den Kontakt mit dem Touchscreen zu verlieren.
- **Multitouch**
  Der Touchscreen wird von zwei oder mehr Fingern gleichzeitig berührt.

8. Der Abschnitt geht inhaltlich weitgehend auf [Knott 2016, S. 42–43] zurück.

- **Pinch open**
  Zwei Finger berühren den Touchscreen und werden auseinander gezogen.
- **Pinch close**
  Zwei Finger berühren den Touchscreen und werden aufeinander zubewegt.
- **Rotate**
  Zwei Finger berühren den Touchscreen und es wird eine Drehung mit den Fingern vorgenommen. Dadurch kann der Inhalt – wie zum Beispiel bei mobilen Karten-Apps – gedreht werden.

2.2.3 Konsequenzen für das Mobile App Engineering

Vor dem Hintergrund der starken hardware- und softwareseitigen Fragmentierung sollte im Rahmen des Requirements Engineering zu einem möglichst frühen Zeitpunkt entschieden und festgelegt werden, für welche mobilen Endgerätemodelle und für welche mobilen Betriebssysteme – also iOS, Android, Windows – entwickelt werden soll. Zudem sind die genauen Betriebssystemversionen festzulegen, die unterstützt werden sollen. Dabei gilt es auch, die Performanz der endgerätespezifisch eingesetzten Hauptprozessoren (CPU), die Größe des verfügbaren Arbeits- und Hauptspeichers sowie die unterschiedlichen und zeitlich limitierten Akkulaufzeiten zu berücksichtigen (vgl. [Wasserman 2010]).

Wenn es sich um ein konkretes Kundenprojekt mit einer klaren Aufteilung in Auftraggeber- und Auftragnehmerseite handelt, werden nicht funktionale Anforderungen wie die relevanten mobilen Endgerätemodelle vielfach im Gespräch mit dem Auftraggeber und späteren Benutzern ermittelt. Für den Fall, dass die mobile App für einen zu Beginn nur vage und unscharf zu definierenden Kundenkreis vorgesehen ist, sollten zunächst die Ziel- und Benutzergruppen definiert werden. Auf dieser Basis können exemplarische Benutzer ermittelt und befragt werden oder es werden *Personas* (siehe Abschnitt 4.3) und *Szenarien* (siehe Abschnitt 4.4) gebildet, mit deren Hilfe die Anforderungen an die mobilen Endgeräte ermittelt und abgeleitet werden können.

Die Festlegung der mobilen Endgerätemodelle, auf denen die mobile App voll funktionsfähig laufen soll, ist unter anderem auch wichtig, um die unterschiedlichen Testphasen (siehe Kap. 7) adäquat vorbereiten und durchführen zu können, unabhängig davon, ob der Test intern durchgeführt oder an einen externen Dienstleistungspartner abgegeben wird. Und da es bislang keine zuverlässigen Testwerkzeuge gibt, mit denen die Tests mobiler Apps für die unterschiedlichen Endgeräte und Betriebssystemversionen vollständig automatisiert werden können (vgl. [Joorabchi et al. 2013]), ist es zwingend erforderlich, dass Sie Ihre mobile App auch auf den realen mobilen Endgerätemodellen und unter realen Bedingungen manuell testen.

Zudem müssen Sie in der Testphase überprüfen, ob Ihre mobile App Multitouch-Gesten verarbeiten kann. Dazu sollten Sie mehrere Finger gleichzeitig auf dem Touchscreen benutzen. Zudem sollten Sie mehrere verschiedene Gesten schnell hintereinander auf dem Multitouch-Display ausführen, um zu sehen, ob und wie die mobile App reagiert. Achten Sie auch auf Performanzprobleme bzw. Fehler, die während der Gestenausführung potenziell entstehen können (vgl. [Knott 2016, S. 43]).

2.3 Entwicklungsparadigmen

Die starke hardware- und softwareseitige Fragmentierung durch die Vielzahl unterschiedlicher mobiler Endgeräte und Betriebssystemversionen hat wesentlich dazu beigetragen, dass mittlerweile fünf unterschiedliche Entwicklungsparadigmen für mobile Apps existieren. Neben dem *nativen* Paradigma gibt es noch weitere Entwicklungsparadigmen, deren Zielsetzung eine möglichst betriebssystemunabhängige Entwicklung ist. Auf die Paradigmen zur Entwicklung mobiler Apps wird in den folgenden Abschnitten näher eingegangen.

2.3.1 Native mobile App

Beim *nativen* Entwicklungsparadigma wird eine mobile App exakt auf ein konkretes Betriebssystem wie zum Beispiel *iOS*, *Android* oder *Windows* zugeschnitten und entwickelt. Dabei arbeitet der Mobile-App-Entwickler mit einer integrierten Entwicklungsumgebung (IDE), einem Software Development Kit (SDK) und einer konkreten Programmiersprache und entwickelt eine mobile App für eine bestimmte Betriebssystemversion (vgl. [Charland & Leroux 2011]). Dabei können auch mehrere Betriebssystemversionen unterstützt werden, wenn eine Abwärtskompatibilität bis zu einer bestimmten Version eingestellt wird. Für die beiden aktuell marktführenden Plattformen *iOS* und *Android* werden zurzeit typischerweise die Programmiersprache Swift 3.0 mit der integrierten Entwicklungsumgebung *Xcode 8* (für iOS) bzw. die Programmiersprache Java mit der integrierten Entwicklungsumgebung *Android Studio 2.3* und dem Build-Tool *Gradle* eingesetzt.

Eine native mobile App wird nach ihrer Entwicklung und Veröffentlichung aus dem entsprechenden App Store heruntergeladen, installiert und direkt von der Laufzeitumgebung des mobilen Betriebssystems des mobilen Endgeräts ausgeführt.

Vorteile

Mithilfe nativ entwickelter mobiler Apps kann die höchste Integration einer mobilen App mit dem mobilen Endgerät erzielt werden. Das native Software Development Kit (SDK) unterstützt die jeweiligen Geräteeigenschaften optimal und kann direkt auf spezifische Gerätefunktionen zugreifen (vgl. [Bredlau 2012]).

Es können *alle* Hardwareressourcen sowie *alle* gerätespezifischen Sensoren und Aktoren des mobilen Endgeräts genutzt und eingesetzt werden (siehe auch Abschnitt 2.4). Somit können nativ entwickelte mobile Apps performant, grafisch hochwertig und ressourcensparend ausgelegt werden. Zudem kann mithilfe nativ entwickelter mobiler Apps eine optimale *User Experience* und *Usability* erreicht werden, da sämtliche Möglichkeiten des mobilen Endgeräts optimal eingesetzt und genutzt werden können.

Darüber hinaus werden native mobile Apps über die jeweiligen App Stores vertrieben und nach der in der Regel einfach durchzuführenden Installation wird die mobile App mit einem eigenen Icon direkt auf einer Bildschirmseite des mobilen Endgeräts angezeigt. Und durch positive Bewertungen einer mobilen App in den App Stores können der Vertrieb, die Vermarktung sowie der daraus resultierende Umsatz und Gewinn vereinfacht und optimiert werden.

Daher ist das native Entwicklungsparadigma für komplexe, rechenintensive und performante mobile Apps mit hochwertiger User Experience am besten geeignet. Insbesondere bei kommerziell eingesetzten mobilen Apps hat sich das native Entwicklungsparadigma aktuell durchgesetzt.

Nachteile

Native mobile Apps müssen für jedes Betriebssystem separat entwickelt werden. Somit sind mehrere Entwickler bzw. Entwicklungsteams mit entsprechenden betriebssystemspezifischen Kenntnissen und Erfahrungen erforderlich. Infolgedessen sind die Entwicklungskosten für native mobile Apps deutlich höher als für betriebssystemunabhängige Web-Apps (siehe auch Abschnitt 2.3.2), bei denen es keine bzw. nur äußerst geringe betriebssystemspezifische Ausprägungen und Anforderungen gibt.

2.3.2 Mobile Web-App

Mobile Web-Apps sind mobile Webanwendungen, die in der Regel auf Basis von *HTML5*, *Cascading Style Sheets (CSS3)* und JavaScript entwickelt werden (vgl. [Charland & Leroux 2011]). Diese Web-Apps laufen vollständig in einem Webbrowser wie zum Beispiel *Safari* für *iOS* oder *Google Chrome* für *Android* bzw. in einer bildschirmfüllenden *WebView* und somit auf beliebigen mobilen Endgeräten. Dadurch ist eine Ausrichtung auf ein bestimmtes Betriebssystem, eine spezielle Betriebssystemversion oder ein bestimmtes mobiles Endgerät nicht zwingend notwendig. Vor diesem Hintergrund können Web-Apps betriebssystemunabhängig konzipiert, entwickelt und betrieben werden. Das große Problem der hardware- und softwareseitigen Fragmentierung kommt hierbei nicht zum Tragen.

Vorteile

Die für Web-Apps eingesetzten Webtechnologien HTML5, CSS und JavaScript laufen in allen gängigen Webbrowsern der verfügbaren mobilen Betriebssysteme. Dies reduziert die Entwicklungskosten deutlich. Zudem können Web-Apps mit einer Internetsuchmaschine einfach gefunden und ohne Installation genutzt werden.

Diese Alleinstellungsmerkmale bestehen allerdings nicht mehr exklusiv für Web-Apps, da über die Indexierungsfunktionen von und für iOS und Android mittlerweile auch native mobile Apps von den gängigen Internetsuchmaschinen gefunden und direkt installiert werden können (siehe dazu auch Abschnitt 8.2.3). Außerdem gibt es mittlerweile sogenannte *Android Instant Apps*[9], die auch ohne eine vorherige Installation genutzt werden können.

Reine Web-Apps werden in der Regel nicht über einen App Store vertrieben, sodass die oftmals recht zeitaufwendigen Test-, Freigabe- und Veröffentlichungsprozesse entfallen. Auf diese Weise werden die Entwicklungskosten reduziert und auch der nachfolgende *Deployment*-Prozess erfordert im Vergleich zur nativen Entwicklung weniger Aufwand (vgl. [Charland & Leroux 2011]). Zudem kann eine Web-App in einen nativen App-Container eingebettet werden, um dann als *hybride mobile App* (siehe auch Abschnitt 2.3.3) auch betriebssystemspezifische – also native – Funktionen ausführen zu können.

Nachteile

Die Verwendung einer Web-App setzt voraus, dass eine möglichst breitbandige Netzverbindung besteht. Zudem müssen die eingesetzten Webtechnologien HTML5, CSS3 und JavaScript vom Webbrowser zeitintensiv interpretiert werden. Daher ist die Performanz einer Web-App deutlich geringer als die einer nativen mobilen App. Zudem sind die Webbrowser mobiler Endgeräte üblicherweise nicht in der Lage, auf sämtliche Hardwarekomponenten eines mobilen Endgeräts zuzugreifen. Da Web-Apps üblicherweise nicht über einen App Store vertrieben werden, entfallen die hierbei durchgeführten Qualitätssicherungsprozesse. Auch kann der App Store nicht zu Marketing- und Vertriebszwecken eingesetzt werden, da jede Web-App individuell vertrieben werden muss. Es ist für die Benutzer einer Web-App somit auch nicht ohne Weiteres möglich, an einer zentralen Stelle (im App Store) Verbesserungsvorschläge bzw. Bewertungen abzugeben.

9. Weitere Informationen unter: *http://developer.android.com/topic/instant-apps/index.html*.

2.3.3 Hybride mobile App

Hybride mobile Apps sind Apps, die auf Webtechnologien (HTML5, CSS3 und JavaScript) basieren, in einer WebView ausgeführt werden und über Plug-ins auf native Funktionen und Komponenten des mobilen Endgeräts zugreifen können. Beim hybriden Entwicklungsparadigma wird die mobile App zunächst webbasiert und somit betriebssystemunabhängig entwickelt. Anschließend wird die App mit nativen Funktionen kombiniert und erweitert, um gerätespezifische Eigenschaften bzw. einzelne Sensoren wie den GPS-Sensor, die Kamera oder die Kontakte in der mobilen App nutzen zu können.

Hybride mobile Apps können über einen App Store veröffentlicht, vertrieben und installiert werden, ohne dass mehrere parallele Entwicklungsstränge existieren müssen.

Viele Unternehmen nutzen hybride mobile Apps auch als Container für bereits existierende Webseiten. Dabei besteht die Hoffnung, dass auf diese Weise eine Präsenz im App Store erzielt werden kann, ohne den höheren Aufwand zur Entwicklung einer rein nativen mobilen App aufbringen zu müssen (vgl. [Nielsen & Budiu 2013, S. 58]).

Vorteile

Eine hybride mobile App kann im Hinblick auf die Gestaltung der grafischen Benutzungsoberfläche, des Designs der User Experience und des Entwurfs des Interaktionsdesigns mittlere Qualitätskriterien erfüllen, die zwischen der hohen Qualität bei der Befolgung des nativen Paradigmas und dem etwas darunter befindlichen Niveau bei einer Entwicklung als Web-App liegen. Bei einer hybriden Entwicklung können die Entwicklungskosten im Vergleich zur nativen Entwicklung reduziert werden; allerdings ist der Aufwand und das erforderliche Know-how zur nativen Ausgestaltung der mobilen Web-App nicht zu unterschätzen.

Nachteile

Eine hybride mobile App kann aufgrund ihres Ursprungs als Web-App nicht alle geräte- und betriebssystemspezifischen Funktionen und Möglichkeiten nutzen. Zudem ist sie im Vergleich zu einer nativen mobilen App weniger performant. Auch die reinen Entwicklungskosten sind bei der hybriden Entwicklung höher als bei einer Web-App.

2.3.4 Interpretierte mobile App

Das Entwicklungsparadigma der interpretierten mobilen App wird oftmals vereinfachend dem hybriden Entwicklungsparadigma zugeordnet, auch wenn dies inhaltlich nicht präzise ist.

Bei diesem Ansatz wird der Quellcode *zur Laufzeit* auf der Zielplattform interpretiert. Voraussetzung dafür ist allerdings, dass auf der Zielplattform bereits eine spezielle Ausführungsumgebung vorhanden ist bzw. diese mit der mobilen App gemeinsam ausgeliefert wird (vgl. [Schekelmann 2016]), wie dies beispielsweise bei *Xamarin.Android* umgesetzt wird (siehe auch Abschnitt 6.4.1). Es werden betriebssystemspezifische und somit native UI-Elemente verwendet, um mit dem Benutzer zu interagieren, während andererseits die Anwendungslogik betriebssystemunabhängig realisiert wird.

Vor- und Nachteile

Der Ansatz als interpretierte mobile App hat neben dem Vorteil, betriebssystemunabhängig entwickeln zu können, einen gravierenden Nachteil: Die Entwicklung ist untrennbar mit den Grenzen, Möglichkeiten und der Aktualität des gewählten Entwicklungswerkzeugs verbunden (vgl. [Behrens 2010]). Wenn die Hardware des mobilen Endgeräts oder die darauf aufsetzende Betriebssystemsoftware neue Funktionen und Möglichkeiten bereitstellt und das ausgewählte Entwicklungswerkzeug diese Funktionen und Möglichkeiten (noch) nicht unterstützt, kann es zu Funktionseinschränkungen und Problemen kommen, die nicht ad hoc lösbar sind. Ein bekannter Vertreter für dieses Entwicklungsparadigma ist *Xamarin.Android*.

2.3.5 Cross-kompilierte mobile App

Die Grundidee dieses Ansatzes ist es, native mobile Apps für die unterschiedlichen mobilen Betriebssysteme auf Grundlage *einer* Quellcodebasis zu generieren. Dazu werden Cross-Compiler eingesetzt, die den Quellcode ohne weitere Zwischenschicht für unterschiedliche Betriebssysteme cross-kompilieren, sodass eine native mobile App entsteht. Entwicklungswerkzeuge wie *Xamarin.iOS*[10] (siehe auch Abschnitt 6.4.1) und *React Native* (siehe Abschnitt 6.4.2) nutzen dieses Entwicklungsparadigma.

10. Weitere Informationen unter:
http://developer.xamarin.com/guides/ios/getting_started/hello,_iOS.

Vor- und Nachteile

Die prinzipiellen Vorteile dieses idealtypischen Ansatzes sind vielfältig und enden nicht mit der Erweiterbarkeit der produzierten mobilen Apps (vgl. [Behrens 2010]). Allerdings ist es bislang noch nicht möglich, auf der Grundlage einer einzigen Quellcodebasis automatisch native mobile Apps für die unterschiedlichen mobilen Endgeräte und Betriebssystemversionen zu generieren, mit denen *sämtliche* betriebssystemspezifischen Funktions- und Geräteoptionen der mobilen Endgeräte vollumfänglich genutzt werden können. Insbesondere weist dieser Ansatz bei der Umsetzung der grafischen Benutzungsoberfläche Schwächen auf, sodass die GUI oftmals für jedes mobile Betriebssystem separat entwickelt werden muss.

2.3.6 Konsequenzen für das Mobile App Engineering

Die unterschiedlichen Paradigmen zur Entwicklung einer mobilen App haben zur Folge, dass in der Praxis möglichst frühzeitig evaluiert und entschieden werden sollte, welches Entwicklungsparadigma eingesetzt wird. Jedes Entwicklungsparadigma hat seine spezifischen Einsatzgebiete und Anwendungsfälle, für die es geeignet ist.

Die Entwicklung einer mobilen App als reine Web-App ist hierbei die kostengünstigste Alternative, da keinerlei endgeräte- und betriebssystemspezifische Abhängigkeiten bestehen. Allerdings sind Web-Apps im Vergleich zu hybriden, interpretierten, cross-kompilierten und nativen mobilen Apps weniger hochwertig, da nicht die volle Funktions- und Leistungsfähigkeit der mobilen Endgeräte genutzt werden kann und somit die Usability und User Experience nicht in hoher Qualität realisiert werden kann. Zudem können Web-Apps nicht über einen App Store vertrieben und beworben werden, sodass der Vertriebs- und Marketingaufwand deutlich höher als bei nativen und hybriden mobilen Apps ist.

Das hybride, interpretierte und cross-kompilierte Entwicklungsparadigma ist eine gute Option, um hochwertige mobile Apps für die führenden Betriebssysteme einigermaßen kostengünstig entwickeln und über die App Stores vertreiben zu können.

Allerdings hinkt jedes nicht native Entwicklungsparadigma grundsätzlich dem in kurzen Zeitabständen erweiterten Funktionsumfang der mobilen Betriebssysteme *Android* und *iOS* hinterher: Wenn beispielsweise Apple mit einer neuen iOS-Version ein neues Feature herausbringt, können native mobile iOS-Apps sofort – und somit meist noch vor dem offiziellen Launch der iOS-Version – damit ausgestattet werden. Jedes andere Entwicklungsparadigma ist abhängig vom jeweiligen Framework-Hersteller oder kann dies grundsätzlich nicht leisten, wie zum Beispiel Web-Apps.

Vor diesem Hintergrund hat sich in den letzten Jahren im kommerziellen Bereich das native Entwicklungsparadigma weitgehend durchgesetzt: Nur mithilfe nativer mobiler Apps kann die volle Funktions- und Leistungsfähigkeit der mobilen Endgeräte genutzt und eine optimale Usability und User Experience ermöglicht werden. Zudem ist mithilfe nativer mobiler Apps der Vertrieb und die Vermarktung in voller Breite über die App Stores von Google und Apple einfach möglich.

2.4 Hardwarekomponenten mobiler Endgeräte

In diesem Abschnitt wird ausführlich auf die zahlreichen Hardwarekomponenten mobiler Endgeräte eingegangen und welche Auswirkungen diese im Rahmen des Mobile App Engineering haben.

2.4.1 Multitouch-Display

Zur Bedienung mobiler Endgeräte wie Smartphones und Tablet-Computer hat sich das *kapazitive* Multitouch-Display durchgesetzt. Die andere der beiden Touchscreen-Technologien ist der sogenannte *resistive* oder drucksensitive Touchscreen, der für die Eingabe mit einem Eingabestift konzipiert ist und heutzutage oftmals von Paketdienstleistern eingesetzt wird. Kapazitive Multitouch-Displays reagieren sowohl auf Berührungen durch das Tippen mit einem oder mehreren Fingern als auch auf Druck.

2.4.2 Akkumulator

Mobile Endgeräte wie Smartphones und Tablets werden über einen Akkumulator (Akku) mit Strom versorgt. Die Akkus mobiler Endgeräte haben unterschiedliche Kapazitäten und Laufzeiten. Die besondere Bedeutung des Akkus für mobile Endgeräte wurde nicht zuletzt auch durch die weltweit auftretenden Ladeprobleme des Akkus beim *Samsung Galaxy Note7* und dem daraus im Oktober 2016 verkündeten endgültigen Produktionsstopp dieses Smartphone-Modells offensichtlich. Durch die immer flacher und schlanker werdenden Smartphone-Modelle, die gleichzeitig über sehr große und lichtstarke Displays verfügen, wurden die Anforderungen an die Akkus so hoch, dass – zumindest in diesem Fall – massive und nicht ad hoc behebbare Probleme auftraten. Zudem ist die Stromversorgung über einen Akku insgesamt instabiler als eine kabelbasierte Stromversorgung.