Cross Plattform Mobile Application Design

Björn Oltmanns
ERGOSIGN GmbH Saarbrücken
Europa-Allee 12
66113 Saarbrücken
oltmanns@ergosign.de

Abstract
Der Smartphone und Tablet Boom der vergangenen Jahre hält weiter an. Mittlerweise dominieren Apple und Google mit ihren Geräten und mobilen Betriebssystemen den Markt. In Zukunft ist jedoch von einer weiteren Diversifizierung, beispielsweise durch Microsoft Windows 8 oder neue Geräteformate, auszugehen. Konkurrierende Frameworks und Kulturen stellen Designer und Entwickler von mobilen Applikationen vor enorme Herausforderungen. Der Beitrag erläutert die konzeptionellen Unterschiede von nativen und cross-platform Applikationen und stellt Anhand einer Case-Study einige der Probleme beim Entwurf für mehrere Plattformen vor. Daneben wird auf jene technischen Einschränkungen und Fallstricke beim Entwurf von cross-platform Applikationen eingegangen, die sich direkt auf die User Experience auswirken.

Keywords:
/// Mobile
/// Cross-Plattform
/// Design
/// User Experience

1. Einleitung

Durch den Smartphone Boom der letzten Jahre haben mobile Plattformen einen enormen Aufwind erhalten. Derzeit dominieren zwei mobile Betriebssysteme ganz klar den Markt: Zum einen Googles Android mit 61% Marktanteil, sowie Apples iOS mit 20,5% Marktanteil. Microsoft liegt mit seinem Windows Phone Betriebssystem derzeit abgeschlagen noch hinter Blackberry mit 5,2% Marktanteil. [1]

Der Marktforscher IDC sieht teilweise jedoch ein starkes Wachstum der von Microsoft entwickelten Windows Phone Plattform und des für Tablets optimierten Betriebssystems Windows 8. Bis 2016 könnte sich im Smartphone Markt das Kräfteverhältnis auf 52,9% für Android, 19,0% für iOS und 19,2% für Windows Phone verschieben. [1] Insgesamt ist jedoch von einer weiteren Diversifizierung des Marktes auszugehen. So plant z. B. Facebook ein eigenes Smartphone samt Betriebssystem auf den Markt zu bringen. Intel arbeitet nach dem erfolglosen MeeGo nun an dessen Nachfolger Tizen.

Auf dem Tablett Markt dominiert weiterhin Apples iPad mit einem Marktanteil von 65%, dahinter liegen Samsung und andere Hersteller von Android getriebenen Geräten.

Die native Entwicklung von Applikationen für alle Plattformen stellt aufgrund der wachsenden Diversifizierung eine enorme Herausforderung dar, denn jede der Plattformen basiert auf unterschiedlichen Technologien und Frameworks.

Dieses Problem wird vermehrt durch cross-platform Lösungen angegangen, welche die Entwicklung von Anwendungen für unterschiedlichste Plattformen auf einer gemeinsamen Technologiebasis erlauben. Vor allem HTML5-basierte Lösungen treten hier in den Vordergrund. Die Entwickler werfen dabei gerne Versprechen wie „Code-Once, Deploy-Everywhere" in den Raum.

2. Nativ oder nicht?

Der Begriff der Nativität einer Anwendung hat unterschiedliche Dimensionen. Zum einen kann dies bedeuten, dass eine Anwendung die vom Hersteller vorgesehenen Programmiersprachen und Frameworks einer Plattform nutzt. Dies stellt in den meisten Fällen auch die Grundlage für die Verwendung von nativen UI-Komponenten dar. Entscheidend für eine positive User Experience einer App ist jedoch die Konformität zur Kultur der Plattform und weniger deren technische Grundlage. Neben nativen UI Controls gehören hierzu auch das Layout, das Informations- und Interaktionsdesign, im Speziellen auch der Einsatz von Gesten.

3. Problematik der Fragmentierung

3.1. Konzeptuelle Fragmentierung und Plattformkultur

Jede der heute am Markt anzutreffenden Plattformen verwendet eine eigene Design-Sprache und etabliert eine eigene Kultur, was Aussehen, Interaktion und Verhalten von nativen Anwendungen angeht.

iOS revolutionierte im Jahr 2007 den Markt und stellte etablierte Bedienkonzepte nicht

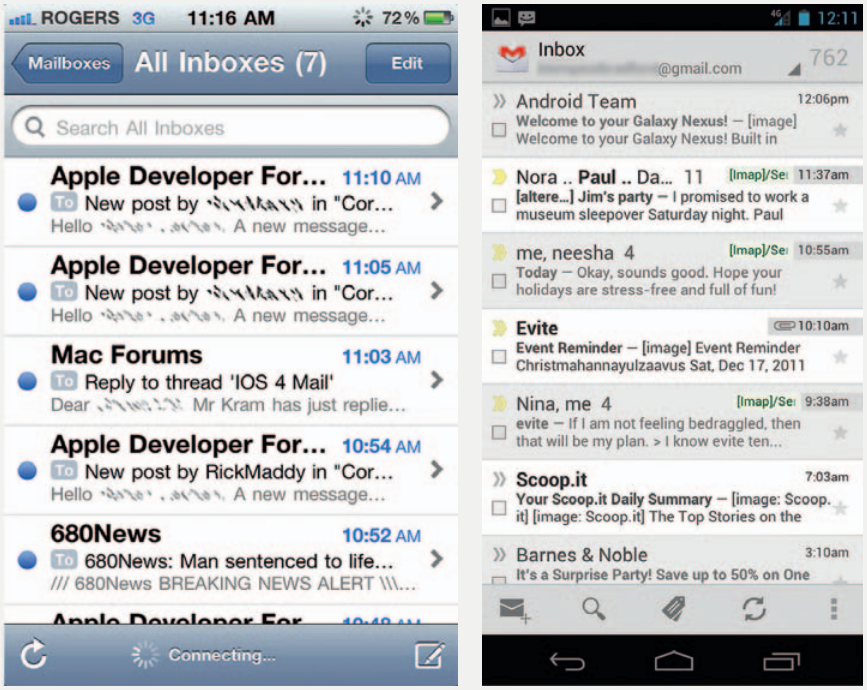

Abb. 1.
Visuelle und konzeptuelle Unterschiede von iOS 5 und Android 4.0 am Beispiel der Email Apps

nur in Frage, sondern löste einen wahren „Touch Boom" aus. Während die Entwicklung von iOS von ihren Anfängen bis heute eine kontinuierliche Evolution darstellt, folgten bei Android mit einigen Versionssprüngen teils massive Änderungen am UI. Google hat im Zuge von Android 4.0 das gesamte Interface von Android überarbeitet und eine visuell reduzierte Design-Sprache etabliert.

Größtes Problem aus Designersicht war zuvor neben fehlenden Design Guidelines auch die Fragmentierung der Android Plattform durch Herstellererweiterungen und deren Custom-UIs.

Apple setzt hingegen auf ein geschlossenes Ökosystem und stellte sehr früh umfangreiche Richtlinien in Form der „iOS Human Interface Guidelines" zur Verfügung. In diesen Dokumenten wird detailliert die Kultur der Plattform beschrieben, auf deren Konzepte und Paradigmen eingegangen und die korrekte Verwendung von UI Controls erläutert. Die Einhaltung dieser Richtlinien stellt zumindest im Groben auch eine Voraussetzung für die Zulassung der eigenen Produkte in Apple's App Store dar. [2]

Für die ersten Android Versionen konnte man sich beim Design lediglich an Anwendungen von Google oder jenen von Drittparteien orientieren. Problematisch ist, dass HTC oder Samsung wie andere Hersteller auf eigene Oberflächenerweiterungen und Look&Feels setzen, welche die Erscheinung von Standardkomponenten beeinflussen. Erst mit Version 4.0 stellt Google selbst ein umfangreiches Design-Dokument zur Verfügung. Zudem schreibt Google die Auslieferung des Standard Look&Feels als Fallback vor. So kann eine Anwendung entscheiden, ob sie Google konform aussehen möchte oder sich dem Herstellerdiktat unterwirft, mit teils undurchschaubaren Folgen. [3]

Grundsätzlich stellt die Einhaltung der Android Design Guidelines jedoch keine Voraussetzung für eine erfolgreiche Aufnahme in den Play Store von Google dar. **[Abb. 1]**

3.2. Technische Fragmentierung

Neben den konzeptuellen Unterschieden der Plattformen, wirken sich auch technische Aspekte auf das plattformübergreifende Design aus. Diese bedürfen ebenfalls einer individuellen Berücksichtigung, um ein optimales Benutzererlebnis zu ermöglichen.

3.2.1. Hardware Buttons

Jede Plattform verwendet Tasten oder Softkeys zur direkten Interaktion mit ihren Anwendungen. Anzahl und Verwendung unterscheiden sich jedoch stark. Das Spektrum reicht von einer einzelnen Taste bei iOS Geräten bis hin zu vier Tasten bei Android Geräten älterer Generation.

Bei iOS ist die Taste lediglich dem Betriebssystem zugeordnet, eine Nutzung innerhalb einer App ist nur zum Verlassen dieser vorgesehen.

Bei Android 2.3 Geräten wird eine Taste zum Verlassen der App und zur Rückkehr zum Home Screen verwendet, eine weitere für die Suche, eine dritte für das Aufrufen des in-App Menüs und eine vierte zur Rückkehr zum vorherigen Screen. Bei Samsung Geräten ist jedoch oft keine Suche Taste vorhanden.

Android 4.0 führt das Konzept von virtuellen Tasten auf der „Navigation Bar" am unteren Ende des Screens ein. Diese haben die Funktionen „Zurück zum letzten Screen", „Applikation verlassen und zum Homescreen" und „App Switcher öffnen" für das Multitasking. Der Menü Button wird durch das neue Konzept der „ActionBar" ersetzt. Legacy Anwendungen die innerhalb von 4.0 ausgeführt werden, erhalten einen zusätzlichen Menü Button in der

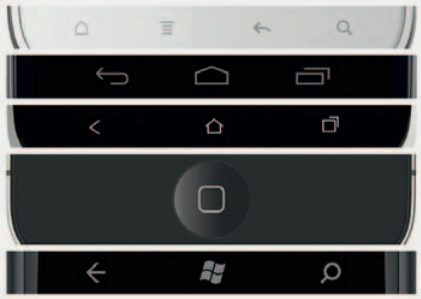

Abb. 2.
Android 2.3 Hardkeys (HTC Sensation XL),
Android 4.0 Virtual Keys (Samsung Galaxy Nexus),
Android 4.0 Hardkeys (HTC One X),
iOS Hardkeys (Apple iPhone 4S),
Windows Phone 7 Hardkeys (Nokia Lumia 800)

„Navigation Bar", der das aus Android 2.x vertraute Menü öffnet. **[Abb. 2]**

3.2.2. Formfaktoren und Auflösungen

Neben unterschiedlichen Betriebssysteme und Versionen, müssen Applikationen auch verschiedene Formfaktoren und Auflösungen beherrschen.

Das iOS Ökosystem stellt hierbei noch die geringste Herausforderung dar. Eine Anwendung muss zwei Formfaktoren, iPhone und iPad, unterstützten sowie deren korrespondierende Auflösungen, 480x320 und 1024x768 auf 3,5 und 9,7 Zoll Bildschirmdiagonale, sowie im Idealfall deren Rotationsformate. „Retina" Auflösungen stellen lediglich eine Vervielfachung der Pixelanzahl dar, jedoch keine Änderung in den physikalischen Größen. Der Designer kann hier pixelgenaue Entwürfe erstellen. Für die Verwendung im Control-Styling und für Icons müssen dann jedoch sowohl Standard als auch Retina Auflösung zur Verfügung gestellt werden.

Das Android Ökosystem hingegen ist stark fragmentiert. Smartphones rangieren von 3 Zoll Screens mit Auflösungen von 320x240 bis hin zu 4,7 Zoll Screens mit bis zu 1280x720 Pixeln. Android Tablets erstrecken sich von 7,0 bis 10,1 Zoll und Auflösungen von 800x480 über 1024x600 bis 1280x800 Pixel. Das native Android Framework führt deshalb eine Matrix aus DPI (Auflösung im Verhältnis zur physikalischen Größe) und Displaygröße ein. Eine native Anwendung kann so bestimmen, welche Auflösungen von UI Grafiken verwendet werden müssen und wie das Layout auf dem Gerät aufgebaut werden muss. Der Designer muss zumindest die gängigen Kombinationen explizit vorsehen und andere durch ein flexibles Layout unterstützen. Ein pixelgenaues Design ist somit kaum möglich. Insgesamt müssen für eine optimale Darstellung Grafiken von Controls und Icons für alle DPI Bereiche vorliegen. Im Idealfall heißt dies vier Auflösungsvarianten pro Element. Eine Verdopplung gegenüber iOS.

4. Case Study: Portierung einer bestehenden Anwendung auf vier Plattformen

Die in den ersten Kapiteln erläuterten Probleme gehörten zu den Vorüberlegungen eines konkreten Kundenprojektes von ERGOSIGN.

4.1. Ausgangspunkt

Ausgangspunkt für unsere Entwicklung war sowohl ein Desktop Client auf Microsoft Silverlight Basis, als auch eine mobile Applikation auf Microsoft Windows Mobile 6.5 Basis. Neu hinzukommen sollten Versionen für Android Smartphones und Tablets und iOS Geräte, die durch uns zu entwerfen waren.

Idee des Kunden war es, konzeptuell auf jeder Zielplattform so nativ wie möglich aufzutreten. Hierzu zählte auch der Einsatz von Standard Controls, wenn immer möglich. Im Hintergrund sollte eine plattformübergreifende Engine abstrakte Konzepte in jeweils native UI Komponenten übersetzen. Wo nötig sollten für einzelne Plattformen Speziallösungen gefunden werden.

Da die neuen Applikationen von Funktionsumfang mindestens der derzeitigen Mobilanwendung entsprechen sollten, erfolgte zunächst eine sorgfältige Analyse derselben. Hierzu wurde anhand eines Papierprototypen der Interaktionsfluss der mobilen Anwendung nachgestellt. **[Abb. 3]**

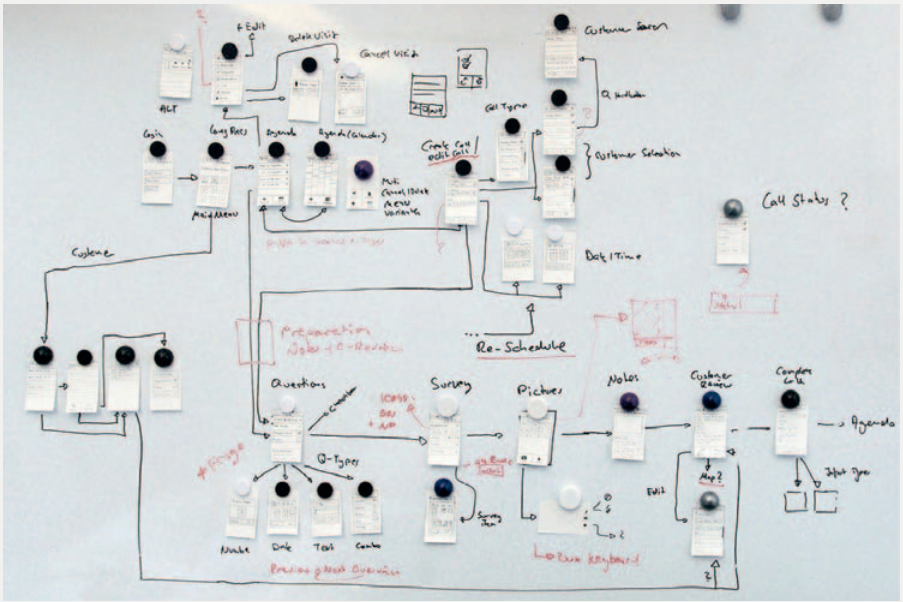

Abb. 3.
Papierprototyp der mobilen Applikation

4.2.
Abstrakter Tablet Master

Um den Umfang der Applikation für die mobilen Plattformen im Gesamten festzulegen, wurden generische Tablet Wireframes angefertigt. Diese legten das plattformübergreifende, grundlegende Layout und den Funktionsumfang fest. Die Entwürfe sollen dabei die Szenarien und Prozesse der bestehenden Anwendung benutzergerecht in einem mobilen Tablet Kontext abbilden.

Jedes Modul der Anwendung bildet in der Regel jeweils einen Prozess ab, der aus mehreren Schritten und Unterschritten bestehen kann. Die Reihenfolge muss jedoch nicht streng eingehalten werden, so dass man nicht von einem Wizard sprechen kann.

Innerhalb eines Prozesses gliedert sich das Layout dann in drei Ebenen:

Zum einen die Anzeige und Auswahl der Prozessschritte, eine Liste mit Elementen oder Optionen zum aktuellen Prozessschritt (Master) und eine Liste mit Optionen zum im Master ausgewählten Element (Detail). Teilweise können in einem Detail noch einmal Unteroptionen auftreten, welche dann in einem Overlay oder Flyout zur Verfügung gestellt werden. Das Design ist hauptsächlich durch Listen getrieben, die je nach Kontext zum Betrachten von Informationen oder zum Bearbeiten von Werten verwendet werden, um den zugrundeliegenden Prozess erfolgreich durchzuführen. **[Abb. 4]**

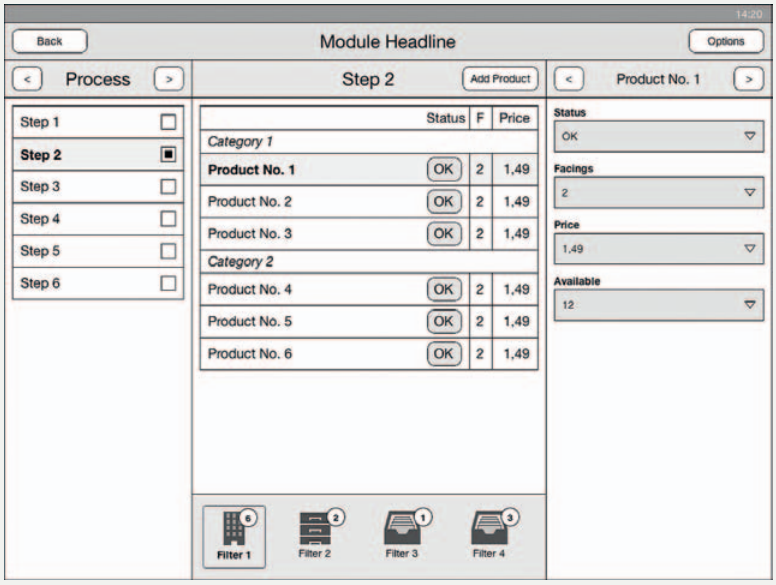

Abb. 4.
Abstraktes Tablet Wireframe mit Prozessnavigation, Prozessschnitt-Master und Prozessschritt-Detail.

4.3.
Überführung in konkrete Plattform Entwürfe

Der abstrakte Master bildet den gesamten Funktionsumfang und die grundlegende Informationsstruktur der Anwendung ab. Die beschriebenen Probleme der ersten Kapitel bedurften einer individuellen Betrachtung jeder Plattform.

Im Folgenden musste deshalb der abstrakte Entwurf in individuelle Entwürfe für alle anvisierten Plattformen übersetzt werden. Für Android wurden sowohl Entwürfe für Version 2.3 als auch 4.0 angestrebt.

Zunächst sollten dabei alle plattformspezifischen Designmuster und Best-Practices so berücksichtigt werden, als erfolge die Entwicklung nur für eine einzelne Plattform. Danach sollten die Entwürfe konsolidiert werden, indem Gemeinsamkeiten identifiziert und zu plattformübergreifenden Konzepten überführt werden. Unvereinbare Unterschiede sollten in Individuallösungen festgehalten werden.

Für den Kunden war es wichtig, eigene Designrichtlinien zu definieren, die es ihm erlauben, bestehende Datenstrukturen jeweils in die für die Plattformen geeigneten Controls zu überführen, so dass diese wiederum den Design Richtlinien der Plattformen und deren Kriterien der Benutzbarkeit unterliegen. Im technischen Sinne würden die Richtlinien als Input für die plattformübergreifende Engine des Kunden herangezogen.

4.3.1.
Zerlegung für Formfaktoren

Sowohl für iPhone als auch Android mussten die komplexen Layouts des Tablet Masters zunächst in einzelne Views zerlegt werden. Während der Tablet Master bis zu drei Ebenen gleichzeitig anzeigt (Prozess, Master, Detail), kann der Smartphone Formfaktor zu jeder Zeit nur eine Ebene anzeigen oder zumindest nur modal zur Verfügung stellen.

Für Android 2.3 und iPhone führten die Entwürfe zu ähnlichen Ergebnissen. Das Fehlen von Design Guidelines für Android 2.3 verlangte nach eigenen Lösungen, die sich an bestehenden Anwendungen orientierten.

Die Hierarchien der Informationsarchitektur werden in den meisten Fällen durch Listen mit drill-down umgesetzt. Generell wird nur eine Ebene gleichzeitig angezeigt. Die anderen Ebenen sind über Navigationselemente in den Titelleisten, Tab oder Toolbars erreichbar.

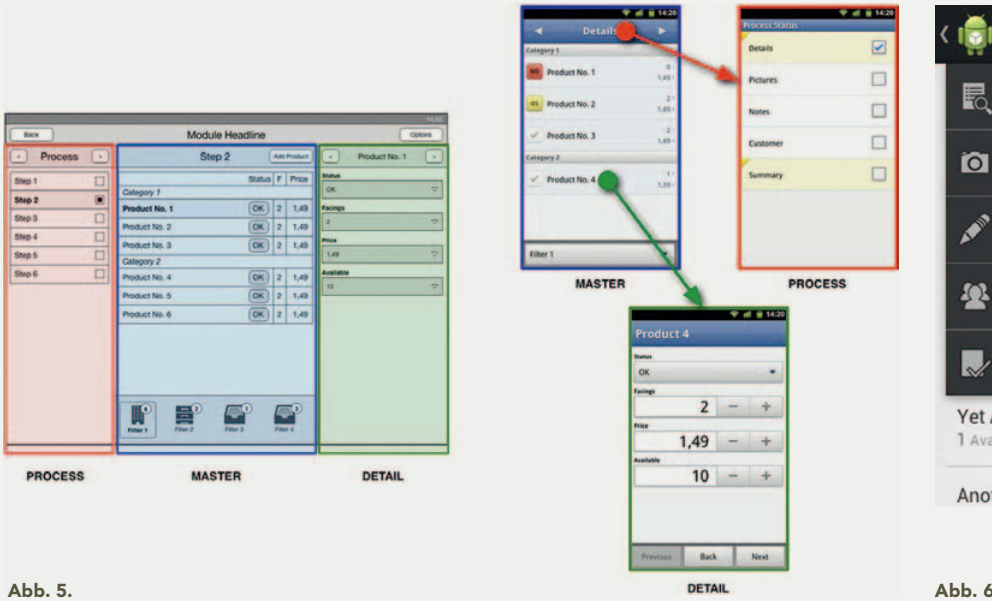

Abb. 5.
Zerlegung des Tablet-Wireframes in „Process", „Master" und „Detail" für Smartphones am Beispiel von Android 2.3.

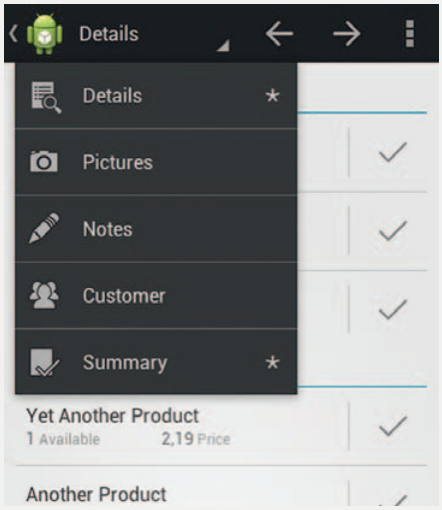

Abb. 6.
Action Bar zur Prozessnavigation in Android 4.0.

Vom Master aus ist die Navigation über Navigationselemente im Header erreichbar. Ein Custom Control erlaubt sowohl den direkten Wechsel zum vorherigen oder nächsten Prozessschritt, als auch den Sprung zur Prozessübersicht.

Die Darstellung von Details zu einem einzelnen Element des Masters erfolgt im Drill-Down auf einen eigenen Screen. **[Abb. 5]**

Für die Rückkehr zum Master muss auf dem iPhone ein „Zurück" Button vorgesehen werden, während auf Android auf die entsprechende Hardwaretaste zurückgegriffen werden kann.

In vielerlei Hinsicht stellt die mit Android 4.0 eingeführte „Action Bar" einen Fortschritt für eine vereinheitliche Navigation innerhalb von Applikationen dar. Sie beinhaltet ein „Nach oben" in der Informationsarchitektur, ein Dropdown um schnell zwischen Bereichen der Applikation zu wechseln, Kontext Aktionen über Icons und einen Menü Button als Ersatz für das Applikationsmenü vor Android 4.0. Das Konzept konnte von uns sehr sinnvoll zur Optimierung der Android Entwürfe eingesetzt werden. Die Prozessnavigation wurde in das Dropdown gelegt und bedarf nun keines eigenen Screens mehr. Zusätzlich wurden Icons in der Bar zum direkten Wechsel auf den vorherigen oder nächsten Prozessschritt gelegt. Das Applikationsmenü wird vollständig im Menü Control der Action Bar verankert. Konzeptuell haben sich damit iPhone und Android Applikation voneinander entfernt, jedoch profitiert der Android Benutzer sehr stark vom eigenständigen Konzept. **[Abb. 6]**

Wenig problematisch stelle sich die Überführung der abstrakten Entwürfe in konkrete Designs für das iPad und Android Tablets dar. Unterschiede ergaben sich hier vor allem in der Verwendung von Toolbars auf dem iPad und der Action Bar auf den Android Tablets. Die grundsätzliche Struktur und das dreispaltige Layout sind auf beiden Plattformen gut umsetzbar.

4.3.2. Konzeptuelle Eigenheiten anhand von Listen und Gesten

Das Anzeigen und Ausführen von Aktionen auf Listenelementen in Android 2.3 erfolgt standardmäßig über eine „Tap-And-Hold" Geste, z. B. für „Löschen" oder „Abbrechen". Die Menü Hardwaretaste wird eingesetzt, um elementübergreifende Aktionen durchzuführen, wie z. B. „Hinzufügen". Sie ruft das Applikationsmenü am unteren Bildschirmrand auf. Um Aktionen auf mehreren Elementen durchzuführen, muss in einen Bearbeiten oder Mehrfachselektionsmodus gewechselt werden. Der Wechsel erfolgt bei uns explizit über das Applikationsmenü. Elemente können dann mittels Checkboxen ausgewählt werden und die Aktion wird über das Menü für alle selektierten Elemente ausgeführt.

„Tab-And-Hold" ist eine auf iOS selten anzutreffende Interaktionsform. Für Elementoptionen werden z. B. Toolbars, Bearbeiten-Modi oder Disclosure verwendet. Da auf den Elementen weiterhin ein Drill-Down möglich sein soll, haben wir uns für einen Disclosure Button entschieden. Ein

„Options" Button wird für elementübergreifende Aktionen vorgesehen. Für die Mehrfachselektion ist das Verhalten analog zu Android umgesetzt. Der Moduswechsel erfolgt über das Optionsmenü. Die Aktion wird über eine Toolbar ausgewählt.

Für iOS müssen also wieder mangels Hardwaretasten einige zusätzliche Interface Elemente eingesetzt werden, die dann auch sinnvoll in Navigation- und Titelleisten untergebracht werden müssen, oder eigene Leisten benötigen.

Während die zuvor beschriebenen Optionen zwar jeweils anders ausgelöst und im UI anders dargestellt werden, sind sie konzeptuell ähnlich und es lassen sich für beide Regeln definieren, um sie aus einer gemeinsamen Datenbasis herzuleiten. Android 4.0 führt jedoch ein eigenes Konzept für Selektion und Kontextoptionen ein. Die „Tab-And-Hold" Geste wird nicht mehr im Android 2.3 Sinne verwendet, sondern um in einen Selektions-Modus zu wechseln, welcher dann für alle selektierten Elemente eine eigene Toolbar zur Verfügung stellt. Möchte man sofort zugängliche Elementoptionen anbieten, so muss bei Drill-Down auslösenden Elementen auch bei Android 4.0 ein Disclosure Button vorgesehen werden. Ein solches Listenelement erhält dann einen Dropdown Marker in Form eines Rechteckes am rechten Rand. Elementübergreifende Aktionen werden über Icons in der „Action Bar" und deren Menü ausgelöst. **[Abb. 7]**

5. Cross-Plattform

Das in der Case-Study beschriebene Konzept einer nativen Anwendung für jede Plattform, mit einer übergreifenden Logik-Schicht sollte im Folgenden mit einer auf HTML5 basierenden Cross-Plattform Lösung verglichen werden.

Der Kunde erwog hierzu die Umsetzung mittels RhoMobile. Die übergreifende Logik würde damit in einem Ruby Backend umgesetzt. Das Frontend würde ausschließlich mit Web-Technologien auf HTML5 und JavaScript Basis implementiert. RhoMobile sieht hierzu standardmäßig den Einsatz des jQuery Mobile Frameworks vor. [4]

Um eine möglichst native User Experience zu generieren, sollen die im Entwurf identifizierten konzeptionellen Unterschiede weiterhin Berücksichtigung finden.

5.1. Cross-Plattform Ansätze

Während native Ansätze der individuellen Entwicklung mit den Sprachen der Plattformen bedürfen, verfolgen die meisten Cross-Plattform Ansätze eine Entwicklung mit Web-Technologien.

Das Spektrum reicht von Cross-Compiling aus JavaScript in native Sprachen, über Back-End Systeme mit Python oder Ruby bis zu reinen Web-Apps, bei denen das Layout über HTML erfolgt, das Styling über CSS und die Logik mit JavaScript definiert wird.

Weiterhin sind hybride Apps inzwischen verbreitet, die native Komponenten mit WebViews mischen, um dokumentennahe Teilbereiche der Applikation durch HTML zu generieren. Ein bekannter Vertreter ist hier die Facebook App.

Cross-Compiling Lösungen wie Titanium erlauben es zudem, die nativen Programmiersprachen und Frameworks durch plattformfremde Technologien zu ersetzen, jedoch ein natives UI zu generieren und damit im Fortgang auch die Kultur der Plattform zu ehren. Die Entwicklung macht sich hier jedoch abhängig von der Qualität und den Fähigkeiten der Drittanbieterlösung. Insbesondere ist es für Designer ohne Kenntnisse des Frameworks nicht transparent, wie anpassbar die generierten Controls sind und welche UI Komponenten sich tatsächlich nativ umsetzen lassen. [5]

Sencha Touch und jQuery Mobile sind rein browserbasierte Ansätze, bei denen sowohl GUI als auch Logik in einer WebView laufen. Lösungen wie Phonegap schließen die Brücke der WebView zur eigentlichen Device-Logik, um über HTML5 und JavaScript auf Gerätehardware und native Schnittstellen zurückzugreifen. RhoMobile verwendet als Brücke zur nativen Welt eine eigene Serverkomponente, die auf dem Gerät läuft und Daten an eine WebView ausliefert. Das Front-End kann mit HTML und JavaScript oder gar mit jQuery Mobile oder Sencha Touch gestaltet werden.

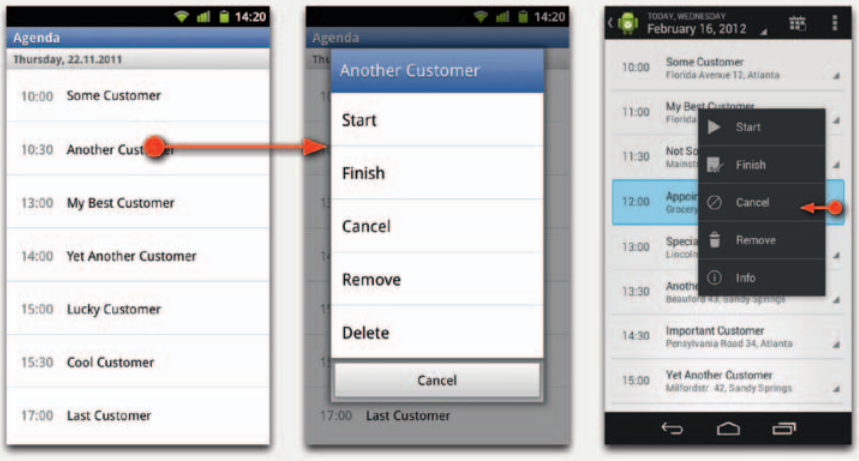

Abb. 7.
„Tab-and-Hold" (Links) öffnet ein Vollbild-Overlay (Mitte) auf Android 2.3 mit Kontext-Aktionen. Das Kontext-Menü wird über ein Disclosure Element in Android 4.0 geöffnet (Rechts).

HTML5 basierte Lösungen haben nur begrenzte Möglichkeiten Formfaktoren, Displaygrößen, Auflösungen und Pixeldichten zu berücksichtigen. JavaScript und auslösungsabhängige Stylesheets können analog zum Response Webdesign herangezogen werden, um die Anwendung anzupassen. Einzig der Android Browser bietet zusätzliche Möglichkeiten zur Evaluation der Bildschirmdichte.

Auf keiner der beschriebenen Plattformen können die Hardware Tasten in einer reinen Web-App verwendet werden. Im Android Browser ist lediglich die Navigation in der URL Historie mittels „Zurück" Taste möglich. Bei einer AJAX-getriebene Anwendung ist ein Einsatz in diesem Sinne nicht unmöglich. Erst die Kapselung als Native App durch beispielsweise PhoneGap oder RhoMobile erlaubt den Zugriff. Um beim Benutzer keine Verwirrung zu stiften, müssen die Tasten in jedem Fall plattform- und betriebssystemabhängig korrekt unterstützt werden. Hingegen müssen „Menü" und „Zurück" bei Fehlen der korrespondierenden Taste über Controls im UI zugänglich sein. Im Sinne einer nativen Anwendung hat iOS zumindest für das „Zurück" das feste Konzept des „Back Buttons" im Header. [7]

Der ausschließliche Einsatz von nativen Frameworks und Design-Richtlinien kann zumindest eine grundlegend Benutzbarkeit und das schnelle Zurechtfinden des Benutzers begünstigen.

Hingegen erlaubt PhoneGap es beispielsweise nur, Webtechnologien in einen nativen Container zu verpacken, ohne Zugriff zu nativen UI Komponenten zu erhalten. Das User Interface muss hier mit HTML, CSS und JavaScript implementiert werden und kann höchsten die Kultur der Plattform nachahmen, jedoch niemals vollständig abbilden.

Der cross-platform Ansatz erlaubt im Prinzip das Ausrollen einer einheitlichen Applikation auf allen Plattformen. So ist eine auf Basis der iOS Richtlinien entwickelte Web-App ohne weiteres auf Android Geräten lauffähig. Jedoch werden die Anmutung, die Anordnung von Tabbars oder die Anwesenheit eines „Back" Buttons bei Android Benutzern eine Fremdkörper-Wahrnehmung hervorrufen.

Kein Problem stellt dies sicherlich bei immersiven Applikationen, wie z. B. Spielen dar, da solche eigene Welten erschaffen in die der Benutzer eintaucht. Standard Controls oder Plattform Paradigmen werden hier nur in den seltensten fällen eingesetzt. So besteht von Benutzerseite auch keine Erwartungshaltung diesbezüglich.

6.1. jQuery Mobile

Das unter anderem von RhoMobile nahe gelegte JQuery Mobile Framework verfolgt aus Entwicklersicht einen klassischen Ansatz des Webs. Ein beliebiges Back-End generiert HTML Code, der die Inhalte der Applikation definiert. Das JavaScript Framework generiert aus dem HTML Gerüst eine touch-freundliche Oberfläche. Das Styling der Oberfläche erfolgt mit CSS.

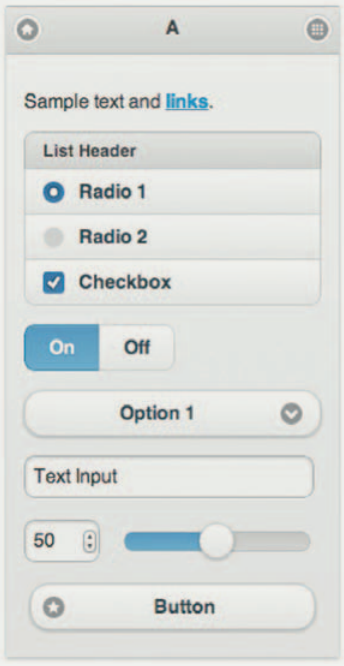

Abb. 8.
Einige jQuery Mobile Controls

jQuery Mobile orientiert sich dabei kaum an nativen UIs. Es bietet eigene Controls mit eigenem Stil (Abb. 8). Weder iOS noch Android Elemente werden nachgeahmt. Hieraus ergibt sich ein eigenes Kulturverständnis für „native" jQuery Mobile Anwendungen. Solche Applikationen werden zwangsläufig als plattformfremd wahrgenommen. Positiv daran ist, dass keine falsche Erwartungshaltung beim Benutzer hervorgerufen wird.

Die Umsetzung von Prototypen oder gar Anwendungen ist für viele Designer mit wenigen HTML und JavaScript Kenntnissen nach kurzem Studium der Dokumentation möglich. Zusätzlich existieren am Markt bereits visuelle UI Builder wie z. B. Codiqa (http://www.codiqa.com/), mit welchen sich jQuery Mobile UIs innerhalb kürzester Zeit definieren lassen. **[Abb. 8]**

6.2. Sencha Touch

Sencha Touch geht im Vergleich zu jQuery Mobile einen anderen Weg: die Definition des UIs erfolgt über deklaratives JavaScript. Das Framework generiert dann zur Laufzeit passend dazu ein HTML Gerüst. Das Styling erfolgt jedoch auch hier mit CSS.

Sencha Touch bringt dabei teils eigene und teils nachgeahmte Controls mit (Abb. 9). Diese sind stark an die nativen Komponenten von iOS angelehnt. Die Interaktionskonzepte sind ebenfalls denen von iOS ähnlich. Am augenscheinlichsten sind hier Navigationsbars, Tabbars, Toolbars, Action Sheets und Picker. Die Controls haben standardmäßig jedoch „mattes" Styling. Dadurch entsteht der Eindruck eines eigenen Look&Feels für native iOS Standardcontrols. Insgesamt wirkt Sencha Touch auf iOS vertraut, auf Android damit jedoch fremd. Es kann der falsche Eindruck einer unangepassten Portierung von iOS entstehen. Vor allem Navigationbars mit „Back" Button und die Anmutung und Anordnung von Tabbars stehen im Widerspruch zu Android Designprinzipien, falls die Controls unangepasst verwendet werden.

Abb. 9.
Sencha Touch Controls erinnern stark an die nativen Controls von iOS

Die Entwicklung eines interaktiven und animierten Prototypen für die Case-Study mit Hilfe von Sencha Touch hat sich als sehr produktiv herausgestellt. Die Arbeit mit JavaScript und HTML sollte für die meisten Web-Designer keine fremde Welt darstellen. Das Styling mit CSS3 führt zu schnellen Ergebnissen. Inzwischen bietet Sencha Touch mit Sencha Architect ebenfalls einen eigenen GUI Builder. **[Abb. 9]**

6.3.1. Performance

Am Beispiel von Sencha Touch zeigt sich jedoch auch, dass vor allem auf Android Devices noch massiver Nachholbedarf seitens der Browserperformance für HTML5 Anwendungen besteht. Komplexe Anwendungen können hier derzeit unmöglich eine native Performance und Responsivität erreichen. In unseren eigenen Tests, vor allem mit Geräten der Mittelklasse, ergaben sich selbst beim Einsatz von lediglich lokalen Daten bereits spürbare Verzögerungen des UIs. Auf iOS Geräten neuerer Generation (iPhone 4, iPad 2) konnten wir eine subjektiv akzeptable Responsivität feststellen. Beim Scrollen von langen Listen mit vielen Einträgen ergeben sich leider dennoch kleinere Verzögerungen.

Selbst cross-platform Framworks, die ein teilweise natives UI generieren, sind nicht vor Performance Problemen gefeit. So hat die bekannte und mit Titanium entwickelte „Wunderlist" App ihre Android Version Ende 2011 von Grund auf neu als native Android App entwickelt. Gründe hierfür waren unter anderen eine ungenügende Performanz und Responsivität. [8] [9]

6.3.2. Konzeptuelle Fallstricke

Da in reinen HTML5 Lösungen die Verwendung von nativen UI Komponenten ausgeschlossen ist, besteht die Möglichkeit diese zumindest visuell nachzuahmen.

Eine HTML basierte Toolbar oder ein Button sieht so z. B. auch wie die entsprechenden nativen Komponenten des iOS AppKit Frameworks von Apple aus. Eingebettet in Listen und Tabellen, die alle jeweils einen nativen Anstrich erhalten haben, wird dem Benutzer praktisch ein natives iOS UI vorgeschwindelt.

Eine solche Erscheinung erweckt automatisch eine auf den Erfahrungen mit nativen Applikationen beruhende Erwartungshaltung: Wie responsiv die App sein wird, welche Elemente welches Feedback liefern, wie das UI animiert ist, wie das Scroll-Verhalten definiert ist, wie Daten manipuliert werden, welche Gesten zur Verfügung stehen und so weiter.

Insgesamt können diese Erwartungshaltungen höchstens Ansatzweise erfüllt werden. Alleine visuelle Ungenauigkeiten, die beim Nachbilden der nativen Controls unvermeidbar sind, werden zwangsläufig den Qualitätseindruck der Anwendung verschlechtern.

Martin Fowler beschreibt dies als das aus der Robotik und Animation bekannte Phänomen des „Uncanny Valley". Eine Applikation, die nahezu so aussieht wie eine native und sich ähnlich verhält, jedoch ein letzter Zweifel übrig bleibt, ruft beim Betrachter oder Benutzer eine überproportionale negative Reaktion hervor. [10] **[Abb. 10]**

Als Design-Prozess zur Entwicklung hybrider Applikationen wäre ein Weg des kleinsten gemeinsamen Nenners denkbar. Dabei werden das UI und die Benutzererfahrung auf Controls und Konzepte beschränkt, die auf allen angestrebten Plattformen vertreten sind. Eventuell reduziert man sogar auf zwei Apps, jeweils für eine Smartphones und für Tablets. Die jeweiligen Entwürfe werden dann konsolidiert. Das Konzept einer portablen Applikationen ist heute faktisch die Grundlage jeder reinen Web-App. Im mobilen Kontext wird letztendlich damit jedoch die User Experience auf allen Plattformen eingeschränkt. Im Prinzip wird eine Applikation entwickelt, die es versucht, es allen recht zu machen aber niemanden vollends befriedigt.

Es stellt sich die Frage, ob überhaupt ein angemessener Mittelweg zu finden ist. Vorstellbar ist es, ein eigenes Design für jede Plattform zu entwickeln, welches die plattformspezifischen Gegebenheiten und Design Muster berücksichtig, ohne als „Copycat" aufzutreten. Hierzu gehören die Informationsarchitektur der App, das Interaktionsdesign und die grundsätzliche Verwendung von Controls sowie der

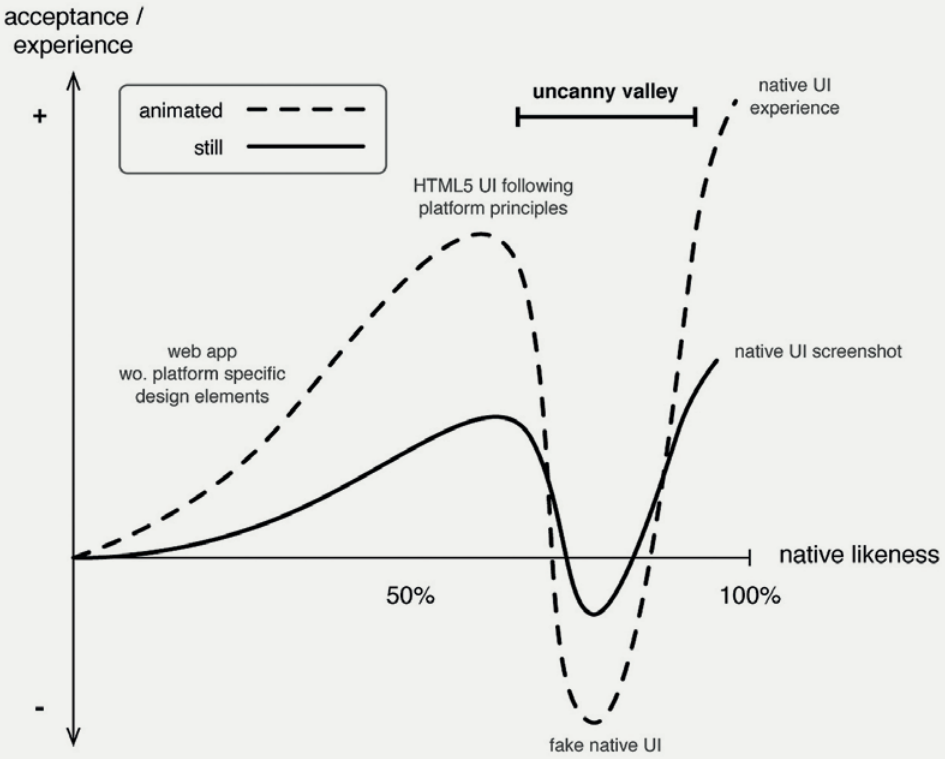

Abb. 10.
Wie das „uncanny valley" im Bereich cross-platform App Design aussehen könnte.

generelle visuelle Stil der Applikation und der Einsatz von Hardware Tasten zum Aufruf von App Funktionen.

Optimiert man eine Applikation stark auf den Use-Case und fügt den nativen Mustern dafür eine ganze Reihe eigener Konzepte hinzu, entstehen zwangsläufig plattformübergreifende Konzepte. Gepaart mit einem eigenen, jeweils plattformspezifischen visuellen Design, das die entsprechende Designsprache nicht ignoriert, sondern weiter entwickelt, kann durchaus eine robuste User Experience entstehen. Die Frage ist, ob eine solche Lösung kostengünstiger als eine rein native multi-platform Entwicklung zu implementieren ist. Und wenn nicht, wie dann der cross-platform Ansatz noch zu begründen wäre.

7. Fazit

Sowohl die aufgeführte Case-Study aus Design Sicht als auch die beschriebenen technischen Rahmenbedingungen und Limitierungen machen klar, dass Cross-Plattform Design und Entwicklung heute eine enorme Herausforderung darstellen.

Die Diversifizierung der mobilen Landschaft hat viele miteinander konkurrierende native Technologien, Design Richtlinien und Plattformkulturen hervorgebracht. Das Ziel einer bestmöglichen User Experience kann nur erreicht werden, indem jede gewünschte Zielplattform im Einzelnen berücksichtigt wird.

Cross-Plattform „Lösungen" gibt es nicht. „Code-Once, Deploy-Everywhere" ist heute technisch kein Problem mehr, dank der umfangreichen Fähigkeiten modernen Browser und technologischen Brücken zwischen HTML5, Javascript und nativen Schnittstellen. Jedoch befreit dies nicht von einer sorgfältigen Analyse der einzelnen Plattformkulturen, Paradigmen und Muster.

Trotz aller Abstraktionsmöglichkeiten in der Informationsarchitektur und im Interaktionsdesign müssen für jede Plattform die passenden Layouts, Controls und Gesten verwendet werden. Alleine die Unterschiede zwischen iOS und Android verbieten die Vorstellung einer einfachen Anpassung mittels Stylesheets oder ähnlichem, denn es ist nicht nur die oberflächliche Erscheinung, die dem Benutzer das Gefühl gibt zuhause zu sein.

Im Idealfall steht hinter jeder Version einer Applikation ein eigenes sorgfältig durchdachtes Design und plattformspezifischer Code.

Literatur

1. [1] IDC. (2012, Juni) Android Expected to Reach Its Peak This Year as Mobile Phone Shipments Slow. [Online]. http://www.idc.com/getdoc.jsp?containerId=prUS23523812
2. [2] Apple Inc. (2012, März) iOS Human Interface Guidelines. [Online]. http://developer.apple.com/library/ios/#documentation/userexperience/conceptual/mobilehig/Introduction/Introduction.html
3. [3] Google Inc. Android Design Guidelines. [Online]. http://developer.android.com/design/index.html
4. [4] Motorola Inc. (2012) RhoMobile Suite. [Online]. http://www.rhomobile.com
5. [5] Appcelerator Inc. (2012) Titanium SDK. [Online]. http://www.appcelerator.com/platform/titanium-sdk
6. [6] Sencha Inc. (2012) Sencha Touch. [Online]. http://www.sencha.com/products/touch
7. [7] jQuery Foundation. (2012) jQuery Mobile. [Online]. http://jquerymobile.com/
8. [8] 6 Wunderkinder. (2011, September) Wunderlist for Android? Rebuild, relaunched and really awesome! [Online]. http://www.6wunderkinder.com/blog/2011/09/05/wunderlist-for-android-rebuilt-relaunched-and-really-awesome/
9. [9] Manuel Gomes. (2011, September) Wunderlis for Android going native, drops Appcelerator Titanium Mobile. [Online]. http://www.tryexcept.com/articles/2011/09/12/wunderlist-for-android-is-going-native-drops-appcelerator-titanium-mobile.html
10. [10] Martin Fowler. (2012, Juni) Developing Software for Multiple Mobile Devices. [Online]. http://martinfowler.com/articles/multiMobile/