Best Practices für die Entwicklung mobiler Unternehmens-Apps

Schwierige Wahl

Christian Rühl, Thorsten Schenkel

Apps sind speziell für den Einsatz auf mobilen Endgeräten konzipiert. Sie lassen sich im Allgemeinen gegenüber Desktop-Anwendungen einfacher bedienen und bieten grundsätzlich andere Einsatzmöglichkeiten. Der Artikel gibt am Beispiel von Android und iOS wichtige Tipps für die Entwicklung mobiler Apps.

Die eingangs erwähnten Vorteile von Apps sowie zunehmend preiswertere und leistungsfähigere Endgeräte lassen die Nachfrage nach Apps kontinuierlich steigen. Aber auch die Anforderungen an Apps steigen in gleichem Maße. Unternehmen haben damit begonnen, Apps in unterschiedlichen Bereichen ihres Arbeitsalltags zu integrieren, etwa zur Vertriebsunterstützung, Echtzeitdarstellung relevanter Unternehmensdaten, Kundenbindung oder Optimierung der eigenen Geschäftsprozesse. Dieser Umstand wirft neue Fragen auf und stellt die Entwickler vor neue Herausforderungen, denen zur Sicherung eines längerfristigen Erfolgs zu begegnen ist.

Die Qual der Wahl

Wer für mobile Endgeräte entwickeln möchte, kann zwischen zwei grundlegend unterschiedlichen Ansätzen wählen:

- Sogenannte native Apps werden mit einem Software Development Kit (SDK) für das jeweilige Betriebssystem erstellt. Diese Apps sind ausschließlich auf der jeweilig dafür vorgesehenen Zielplattform lauffähig und lassen sich nicht auf einer anderen Umgebung ausführen.
- Webapps dagegen setzen auf HTML5 als plattformunabhängige Basistechnologie, die es diesen Apps ermöglicht, innerhalb eines Browsers auf allen modernen mobilen Endgeräten zu laufen.

Da das Schreiben nativer Apps grundlegend von der von Webapps differiert, gehört die Entscheidung über den bestmöglichen Entwicklungsansatz zu den ersten und folgenschwersten im Entwicklungsprozess. Von ihr hängt unter anderem ab, welche Gerätefunktionen sich in der App nutzen lassen, wie hoch der Migrationsaufwand bei Ausweitung der unterstützten Zielplattformen ausfällt und ob sich gegebenenfalls die Rendite infolge einer Umsatzbeteiligung des App-Store-Betreibers verringert.

Während native Apps dank des jeweiligen nativen SDKs auch auf gerätespezifische Funktionen des Hostgeräts bestens zugreifen können, sind Webapps (noch) nicht oder in nur eingeschränktem Umfang hierzu in der Lage. HTML5 befindet sich noch in der Entwicklung. Derzeit ist die quantitative und qualitative Unterstützung erweiterter HTML5-Features stark vom jeweiligen Browser des Anwenders abhängig. Das Problem zeigt sich vor allem bei der Webapp-Entwicklung für Android, da sich der Anwender hier aus einer Vielzahl unterschiedlicher – und hinsichtlich HTML5 auch unterschiedlich leistungsfähiger – Browser bedienen kann.

Unterstützt durch die rasche Evolution mobiler Browser und clientseitiger HTML5-Frameworks ist für die nähere Zukunft zu erwarten, dass HTML5 sowohl im Hinblick auf die Leistungsfähigkeit als auch hinsichtlich des Funktionsumfangs und der Homogenität der qualitativen Realisierung zu den nativen SDKs aufschließen wird. Das Versprechen von HTML5, einen „one fits all"-Ansatz bereitzustellen, können die derzeit zur Verfü-

gung stehenden Implementierungen – zumindest für komplexe Anwendungen und im direkten Vergleich zu nativen Apps – jedoch noch nicht einlösen.

Hybride Alternative

Wer zumindest einige der Unzulänglichkeiten von HTML5 kompensieren möchte, dem sei ein Blick auf „hybride Apps" empfohlen. Bei ihnen handelt es sich im Kern um Webapps, die auf dem Endgerät allerdings nicht in einem Browser bedient werden, sondern in den Rahmen einer nativen App eingebettet sind. In der Tat ist es auf den ersten Blick für einen Laien nicht ersichtlich, ob eine Anwendung nativ oder als hybride App entwickelt wurde, zumal hybride wie native Apps über die jeweiligen App-Stores der Betriebssystemhersteller verteilt werden. Frameworks zum Erstellen hybrider Apps gibt es reichlich, zum Beispiel PhoneGap [a], Titanium Appcelerator [b], Brightcove App Cloud [c] oder Adobe AIR [d].

Diese Frameworks unterstützen mittlerweile sowohl Android als auch iOS und bieten zudem eine „Bridge", mit der Webanwendungen per JavaScript API auf spezifische Funktionen des jeweiligen Zielgeräts zugreifen können. Somit kombinieren hybride Apps einige Vorteile von nativen und Webapps, nämlich vor allem die Unterstützung gerätespezifischer Funktionen und die Plattformunabhängigkeit. Zusätzlich wird die Entwicklungskomplexität durch Begrenzung auf eine einzelne HTML5-Implementierung deutlich reduziert (unter iOS und Android kommt für hybride Apps ausschließlich die WebKit-Engine zum Einsatz).

Die Entscheidung für oder gegen eine bestimmte Entwicklungstechnik ist allerdings nicht allein von technischen, sondern zunehmend auch von betriebswirtschaftlichen Kriterien bestimmt. Wird zum Beispiel eine native oder eine hybride App über den App-Store des Betriebssystemherstellers vertrieben, gehen meist 30 Prozent des erzielten Umsatzes an diesen. Gleiches gilt für den Umsatz, den der Entwickler aus dem Verkauf weiterer kostenpflichtiger Inhalte direkt aus der App heraus generiert.

Darüber hinaus besteht vor allem bei der iOS-Entwicklung ein inhärentes Risiko, dass die App im Rahmen einer Review durch den Shop-Betreiber nicht für den App-Store zugelassen wird. Daher ist es dringend erforderlich, die jeweiligen Kriterien zur Aufnahme in den Store bereits im Vorfeld des Projekts zu evaluieren. Auch dann gibt es jedoch keine Garantie für die Aufnahme in oder den dauerhaften Verbleib im App-Store: Apple ändert beispielsweise zum einen seine Review Guidelines des Öfteren, und andererseits weist die Formulierung der darin definierten Leitsätze Unschärfen auf, die teilweise Spielräume für die Auslegung durch den zuständigen Apple-Reviewer eröffnen. Um derartige Risiken für die Veröffentlichung und Renditeminderungen zu umgehen, bietet sich der Weg über Webapps an, den zum Beispiel Amazon oder die Financial Times für ihre Apps gewählt haben.

Endgerätesituation

Alle Entwicklungsansätze stehen allerdings vor derselben Herausforderung: Sie müssen die vielfältige mobile Endgerätelandschaft berücksichtigen. Sie hat es in sich und bedarf einiger Klimmzüge, vor allem wenn man für den Android-Markt entwickelt. Während Apple als alleiniger Hersteller von iOS-Geräten ein vergleichsweise kleines, konsistentes Sortiment auf den Markt gebracht hat, tummeln sich im Android-Segment zahlreiche Gerätehersteller, die mit unterschiedlicher Hardware und Geräteausstattung den Markt für Smartphones und Tablets beliefern. Die Unterschiede erstrecken sich nicht nur auf die Hardware der Geräte, sondern umfassen auch die Android-Versionen und speziellen Anpassungen der Benutzungsoberfläche durch die Gerätehersteller.

Die Hardwareunterschiede können vielfältig sein, wie eine Auflistung der zurzeit am Markt gängigen Modellvariationen zeigt:

- CPU: Single-, Dual- und Quad-Core; 600 MHz, 800 MHz, 1 GHz, 1,2 GHz
- Bildschirmgröße: 2,6 bis 4,65"
- Bildschirmauflösungen: 320 × 480 bis 1280 × 720
- verfügbarer Arbeitsspeicher: 512 MByte, 768 MByte, 1 GByte und so weiter

Anschaulich geht die Vielfalt aus einem Ende 2010 erschienenen Blog-Eintrag [e] des auf mobile Endgeräte spezialisierten Marktforschungsunternehmens PercentMobile hervor: „Since the first appearance of an Android OS Device in October 2008, PercentMobile has recorded mobile web activity for more than 100 different Android OS devices. This is roughly one new device per week over the past 2 years. Considering that it took almost 6 months for the second Android OS device to appear, the growth rate is nothing short of astonishing."

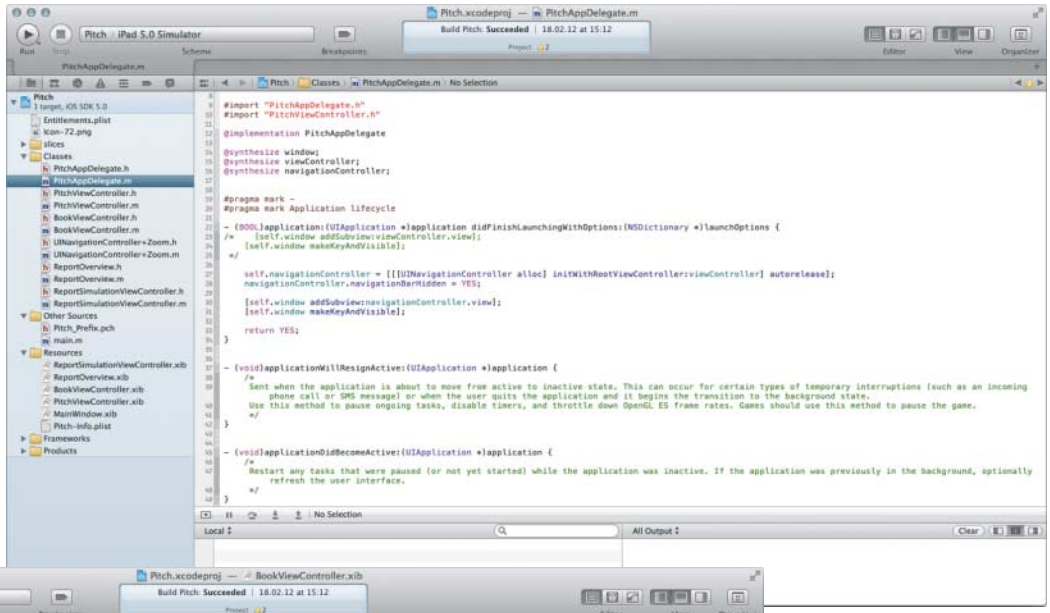
Die Vielfalt ist für den Entwickler nicht mehr überschaubar und nur mit zunehmend größer werdendem Aufwand zu bewältigen. Er muss bei der Entwicklung der Android-App versuchen, diese so flexibel wie möglich zu halten, damit sie sich auf den meisten Endgeräten vernünftig einsetzen lässt. Beispielsweise darf er beim Design der Oberfläche kein Layout verwenden, das eine bestimmte Bildschirmauflösung erwartet, da sich die einzelnen Android-Geräte hinsichtlich Auflösung und Seitenverhältnis stark unterscheiden.

Ein anderer Weg ist die Beschränkung auf Android-Geräte mit bestimmten Hardware- und/oder Softwarevoraussetzungen. Hinsichtlich dieser kann der Entwickler seine App optimal anpassen. Für beide Wege bietet das Android SDK Unterstützung. Die Flexibilität erreicht man etwa durch eine Reihe von Layout-Managern [f] und durch die ausgefeilte Ressourcenverwaltung [g]. Die Einschränkung auf bestimmte Endgeräte erfolgt durch das Festlegen von Voraussetzungen in der Manifest-Datei.

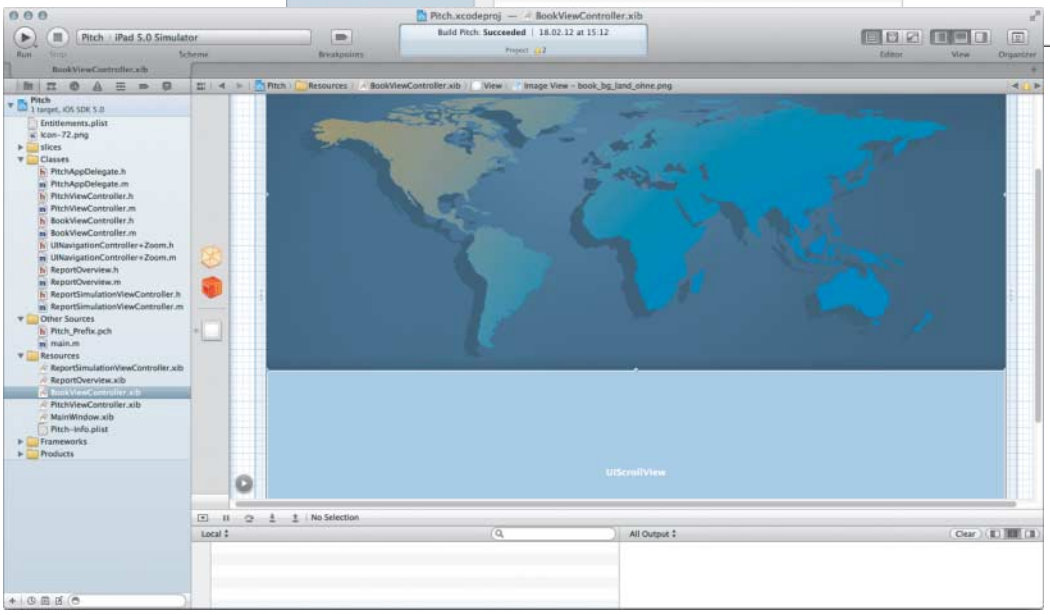
Aber auch die Vielfalt der auf dem Markt befindlichen Android-Versionen stellt den Entwickler vor Herausforderungen. Seit Oktober 2011 steht die derzeit aktuelle Android-Version 4.x bereit und mit ihr zahlreiche Änderungen, die bis in das Bedienkonzept für Apps hineinreichen. Doch Entwickler sollten genau abwägen, welche der neuen Funktionen Eingang in ihre App finden. Die Erfahrung hat gezeigt, dass es bis zu einem Jahr dauert, bis sich eine neue Betriebssystemversion auf den Geräten der Nutzer ausreichend etabliert hat. Ein schnelleres Update auf neue Android-Versionen erschweren zudem oft angepasste Oberflächen der Gerätehersteller. Demnach sollten Android-Entwickler bei der Planung und Entwicklung ihrer App daran denken, dass neue Betriebssystemversionen erst mit einiger Verspätung den Markt durchdringen.

Entwicklungsumgebungen im Vergleich

Der Einstieg in die professionelle App-Entwicklung fällt je nach Zielplattform, Entwicklungsansatz und persönlichen Skills mehr oder weniger leicht. Während sich unter Android Java-Entwickler dank Java und der umfassenden und nahtlosen Integration des Android Development Kit in Eclipse sofort heimisch fühlen dürften, wird denselben Entwicklern der Einstieg in die App-Programmierung auf Basis von iOS dank der Eclipse in puncto



Die Sicht auf das Projekt und seine Sourcen in Xcode (Abb. 1)



Quelle: mit freundlicher Genehmigung von Fünfwerken Design AG

Der nunmehr integrierte Interface Builder in Xcode (Abb. 2)

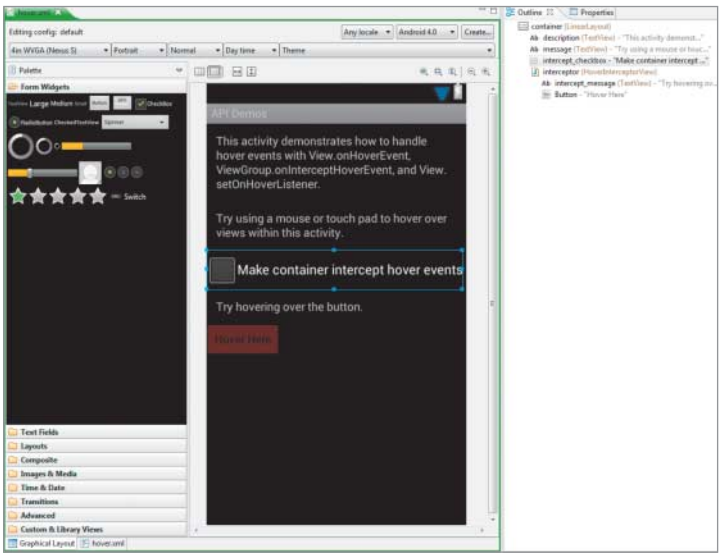
Komfort und Unterstützung unterlegenen Apple-IDE Xcode und der völlig von Java abweichenden Programmiersprache Objective-C 2.0 wie eine fremde Welt vorkommen. Letzteres dürfte gestandene C/C++-Programmierer wiederum nicht abschrecken, da Objective-C letztlich nichts anderes als eine objektorientierte Erweiterung für die Programmiersprache C ist, auch wenn es auf den ersten Blick nicht so erscheinen mag. Zu unterschiedlich sind ihre Syntax und zugrunde liegenden Konzepte.

Gleichermaßen fremd dürfte Java- und C-Entwicklern die Anwendungsentwicklung mit HTML5 sein. Sie unterliegt nämlich im Gegensatz zur Programmierung nativer Apps verhältnismäßig wenigen Richtlinien und bedingt einen bisweilen unübersichtlichen Mix aus HTML5 (zur Strukturierung der Views), CSS3 (zum Styling der View-Elemente) und JavaScript (zur clientseitigen Realisierung der Anwendungslogik). Und nicht selten ist zusätzlich ein „intelligenter" Webserver notwendig, um der App Leben einzuhauchen, was im schlimmsten Fall das Erlernen einer weiteren (serverseitigen) Programmiersprache erfordern kann.

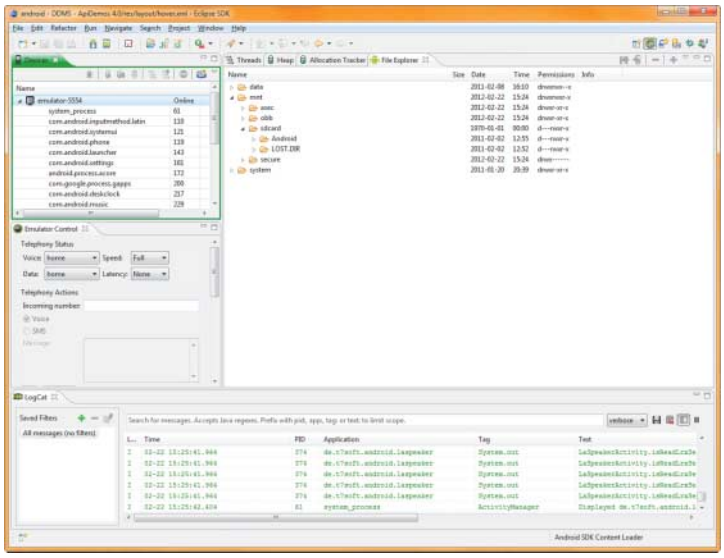
Mittlerweile können Webentwickler auf eine Vielzahl teils hochwertiger Entwicklungswerkzeuge und Frameworks zurückgreifen, die – sinnvoll eingesetzt – dabei helfen können, die Komplexität unter Kontrolle zu bringen. Eine ausführliche Evaluierung der Programmierwerkzeuge und Frameworks unter Berücksichtigung der für die Entwicklung der Webapp erforderlichen eigenen Skills und Anforderungen an die App ist allerdings dringend anzuraten. Ein Java-Entwickler beispielsweise dürfte mit Eclipse/Web Tools Platform als Entwicklungsumgebung und JavaServer Faces als serverseitigem MVC-Framework (Model View Controller) am schnellsten zu guten Ergebnissen kommen.

Besondere Aufmerksamkeit verdient weiterhin die Auswahl eines geeigneten UI-Frameworks. HTML5 bietet – bis auf die von HTML-Formularen her bekannten – vergleichsweise wenige UI-Elemente und schon gar keine komplexen Widgets, wie sie in nativen Apps zur Verfügung stehen. Wer seine Web- oder hybride App dem Erscheinungsbild nativer Apps annähern möchte, sollte sich daher die gängigen mobilen UI-Frameworks für HTML5 wie Sencha Touch [h], jQuery Mobile [i], qooxdoo [j] oder Bootstrap [k] anschauen. Sie unterscheiden sich jeweils erheblich in ihrer Programmierung und Zielsetzung: Während etwa für jQuery Mobile weitgehend HTML- und CSS-Code ausreicht, um einfache mobile Apps zu entwickeln, spricht Sencha Touch in erster Linie JavaScript-Entwickler an, die komplexere Apps schreiben möchten und entsprechende Unterstützung für MVC-Architekturen wünschen.

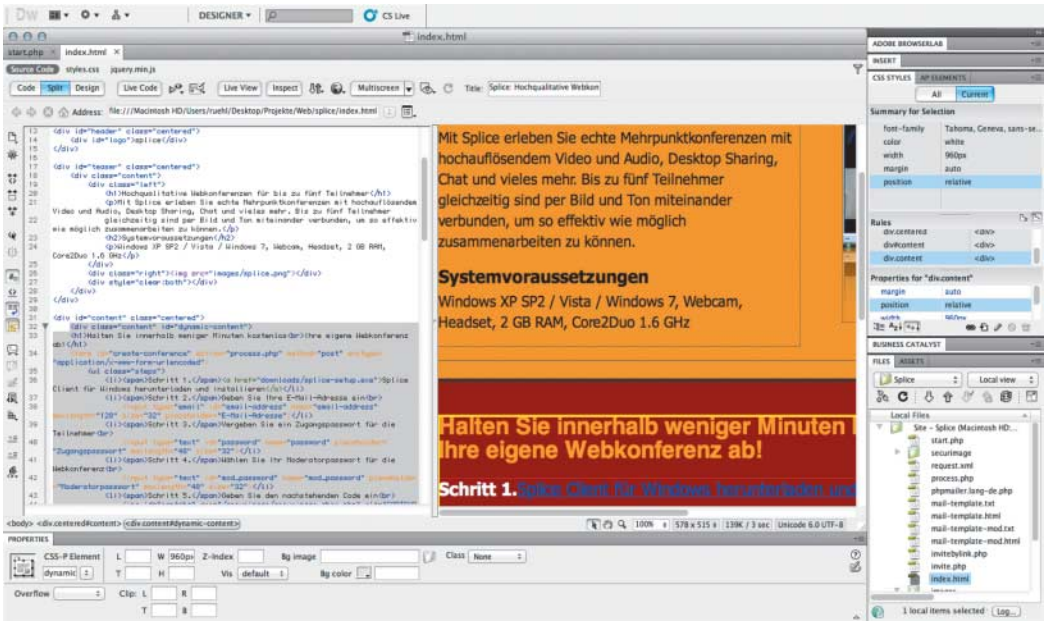
Das intensive und fortlaufende Testen einer App ist das tägliche Brot des mobilen Entwicklers. Das iOS SDK enthält einen leistungsfähigen Simulator, der relativ präzise das iOS-Betriebssystem für iPhone und iPad zu simulieren vermag. Der Simulator startet samt der App innerhalb weniger Sekunden und ermöglicht so ein zügiges Debugging. Nichtsdestoweniger ist



Sicht auf das Projekt in Eclipse mit den Android Development Tools (Abb. 3)



Der in ADT integrierte grafische Layout-Designer für Android-Apps (Abb. 4)



Ein typisches HTML5-Projekt in Adobe Dreamweaver CS 5.5: links der HTML-Sourcecode, mittig die WYSIWYG-Ansicht der HTML5-Seite und rechts die CSS-Stile (oben) und Projektdateien (unten) (Abb. 5)

das frühe und regelmäßige Testen auf einem angeschlossenen iOS-Gerät dringend empfohlen, zumal zum Testen bestimmter Funktionalitäten (z. B. dem In-App-Purchase) ein solches Gerät zwingend erforderlich ist.

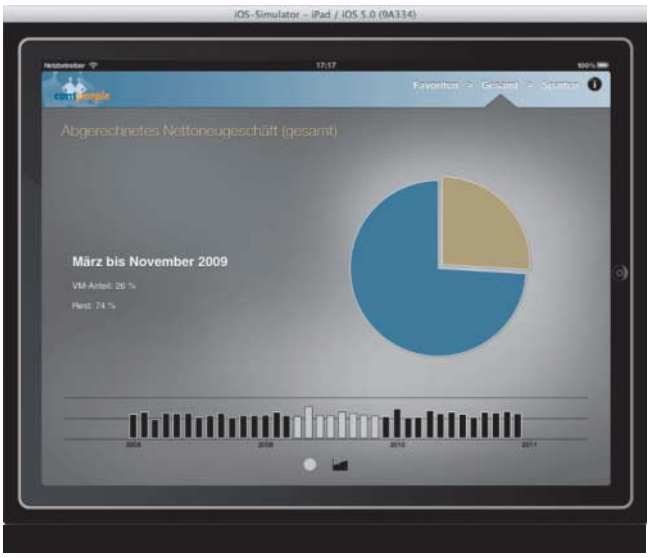
Auch Googles Android Development Tools unterstützen das Testen vor allem durch

- den Dalvik Debug Monitor Server (DDMS) zum Debuggen von Apps auf angeschlossenen Android-Geräten und zum Auslesen von Log-Daten und Gerätedaten sowie
- das Android Virtual Device (AVD), einen Emulator, mit dem sich nahezu jedes Android-Gerät samt wesentlicher Hardwareeigenschaften (z. B. Speichergröße, Bildschirmauflösung und Betriebssystemversion) emulieren lässt.

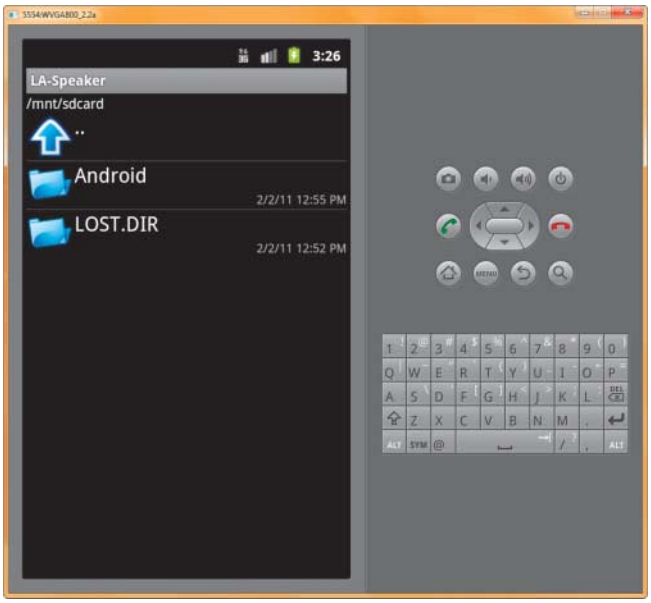
Leider ist die Performance des Emulators trotz der Snapshot-Funktion noch nicht optimal, da zum einen zunächst das Betriebssystem für die Emulation eines spezifischen AVDs zu laden ist, zum anderen der Emulator (noch) über keine Grafikbeschleunigung verfügt [l] und somit für das Debugging komplexer Bildschirminhalte unter hohen Bildschirmauflösungen nur unzureichend geeignet ist.

Unabhängig vom Zielsystem, für das entwickelt wird, empfiehlt sich dringend das frühe und regelmäßige Testen auf einem angeschlossenen Endgerät, da sich die Geräte unter realen Bedingungen anders verhalten können als in einer Simulation beziehungsweise Emulation. Das trifft vor allem auf die Speicherverfügbarkeit, Nebenläufigkeit und Internetkonnektivität zu. Darüber hinaus steht bestimmte gerätespezifische Hardware innerhalb der Simulation beziehungsweise Emulation nicht zur Verfügung. Will man solche Funktionen testen, führt kein Weg am realen Endgerät vorbei.

Aufgrund ihrer Plattformunabhängigkeit und der damit verbundenen Heterogenität der Laufzeitumgebungen gestaltet sich der Test für Webapps diffiziler und zeitaufwendiger als bei den nativen Apps. Angesichts der Tatsache, dass Android-Nutzer zwischen einer ganzen Reihe von Browsern wählen können, reicht es zudem keinesfalls aus, nur unter Androids systemeige-



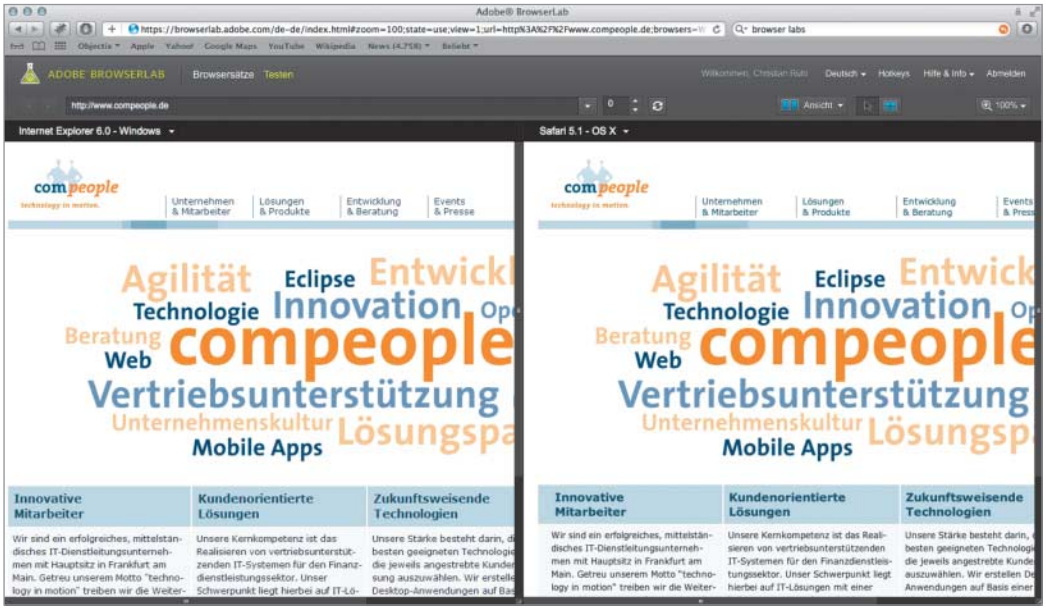
Der aus Xcode heraus gestartete iOS-Simulator (hier wird ein iPad simuliert) mit einer gestarteten App (Abb. 6)



Der Android Emulator von ADT mit einer gestarteten App (Abb. 7)

nen Webbrowser zu testen, sondern man muss alle seine Tests auch unter allen anderen populären ausführen, gegebenenfalls sogar unter Berücksichtigung der Browserversionen.

Aus der Kombination der Parameter Hardware, Browserhersteller und Browserversion ergibt sich ein ausgesprochen komplexer Testplan, der manuell nur mit größtem Aufwand zu realisieren ist. Zwar lassen sich einige Funktionen mit Unit-Tests und entsprechenden guten Test-Frameworks wie sinon.js [m] oder Googles JsTestDriver [n] automatisiert testen. Aber gerade das Testen des grafischen Frontend fällt schwer. Hier können Entwickler teilweise ebenfalls automatisierte Frontend-Testwerkzeuge wie Selenium [o] einsetzen, die aber in der Praxis einen nicht unerheblichen Aufwand zur Aufbereitung der Testfälle nach sich ziehen. Hier ist daher unbedingt das Aufwand-Nutzen-Verhältnis im Auge zu behalten. Die visuelle Integrität der Webapp unter verschiedenen Browsern, Browserversionen und Betriebssystemen lässt sich mit Dienstleistungen wie browsershots.org [p] oder Adobes BrowserLab [q] etwas einfacher si-



Side-by-Side-Vergleich einer Webpage unter zwei verschiedenen Browsern und Betriebssystemen mit Adobe BrowserLab (Abb. 8)

cherstellen. Auch hier muss der Abgleich zwischen Soll und Ist größtenteils manuell erfolgen.

Den richtigen Schlüssel finden

Hat die Entwicklung erst einmal Fahrt aufgenommen und soll das Ergebnis der eigenen Bemühungen nun auf dem Gerät getestet oder gar an den Testanwender ausgeliefert werden, ist die nächste Hürde zu nehmen: Denn das Deployment nativer und hybrider Apps zu Testzwecken kann auch für erfahrene Entwickler eine Herausforderung sein. Alle Apps sind nämlich digital zu signieren, bevor sie sich auf dem jeweiligen Endgerät installieren und ausführen lassen, selbst wenn es sich dabei nur um einen Entwicklungsprototypen handelt. iOS-Einsteiger dürften angesichts des komplex anmutenden Prozesses rund um „Unique Device Identifiers“, „App Identifiers“, „Provisioning Profiles“, „Developer Certificates“ und „Distribution Certificates“ und den damit durch Apple auferlegten Beschränkungen einige graue Haare bekommen.

Gerade während einer ohnehin anstrengenden Testphase sorgt dieser Code-Signing-Prozess in Kombination mit der in der Praxis oft anzutreffenden Volatilität der Testgeräte für einige Schweißperlen auf der Stirn, denn die Fehleranfälligkeit ist ausgesprochen hoch. Teils wenig aussagekräftige Fehlermeldungen, die Xcode oder das Gerät ausspucken, wenn etwas nicht exakt nach Apples Plan gelaufen ist, helfen nämlich nur wenig beim Auffinden und Beheben der Ursache.

Auch Android-Apps sind zu signieren, damit sich diese auf Android-Geräten oder auf dem Emulator installieren und testen lassen. Hierfür gibt es einen speziellen Debug-Key, den die Android SDK Build Tools erzeugen. Erst bei Veröffentlichung der Android-App ist sie mit einem passenden, privaten Schlüssel zu signieren. Er lässt sich mit verbreiteten Werkzeugen wie Keytool und Jarsigner erzeugen und zum Signieren der App-Datei (*.apk*) einsetzen. Dieses Modell und seine liberale Natur ist für den Entwickler etwas leichter anzuwenden als das von Apple: Es gibt nur zwei Schlüssel, einen zum Debugging, einen anderen zur Veröffentlichung in Google Play (vormals Android Market) und keine manuell zu verlängernden Zertifikate.

Onlinequellen

[a]	PhoneGap	phonegap.com
[b]	Titanium Appcelerator	www.appcelerator.com
[c]	Brightcove App Cloud	www.brightcove.com/de/content-app-plattform
[d]	Adobe AIR	www.adobe.com/de/products/air.html
[e]	PercentMobile	mobileanalyticssimplified.com/page/2
[f]	Shane Conder, Lauren Darcey; Android User Interface Design: Layout Basics	mobile.tutsplus.com/tutorials/android/android-layout/
[g]	Android: Prividing Resources	developer.android.com/guide/topics/resources/providing-resources.html
[h]	Sencha Touch	sencha.com/products/touch
[i]	jQuery Mobile	jquerymobile.com
[j]	qooxdoo	qooxdoo.org
[k]	Bootstrap	twitter.github.com/bootstrap/
[l]	Xavier Ducrohet, Tor Norbye; Android Development Tools (Präsentation von der Google I/O 2011)	www.google.com/events/io/2011/sessions/android-development-tools.html
[m]	Sinon.JS	sinonjs.org
[n]	JsTestDriver	code.google.com/p/js-test-driver/
[o]	Selenium	seleniumhq.org
[p]	Browsershots	browsershots.org
[q]	Adobe BrowserLab	browserlab.adobe.com/de-de/
[r]	Wikipedia-Artikel zur „Nutzwertanalyse“	de.wikipedia.org/wiki/Nutzwertanalyse
[s]	Marktanteile der einzelnen Android-Versionen	developer.android.com/resources/dashboard/screens.html
[t]	TestFlight	testflightapp.com
[u]	Shaun Ervine; Continuous Deployment of iOS Apps with Jenkins and TestFlight	blog.shinetech.com/2011/06/23/ci-with-jenkins-for-ios-apps-build-distribution-via-testflightapp-tutorial/

Continuous Integration für iOS

Zentrale Komponenten der Continuous Integration bei der compeople AG, einem mittelständischen IT-Dienstleistungsunternehmen und Experten für IT-Systeme zur Vertriebsunterstützung, sind:

- ein gemeinsames Versionsverwaltungssystem auf Basis von Git, das die Entwickler zur Verwahrung von Code, Grafiken und sonstigen multimedialen Inhalten verwenden.
- eine Instanz des Continuous-Integration-Servers Jenkins, der auf einem separaten Build-Rechner (intern als „Tankstelle“ bezeichnet) regelmäßig auf Änderungen im Versionsverwaltungssystem prüft, gegebenenfalls Änderungen auscheckt, das Gesamtsystem neu baut und das Ergebnis archiviert. Kunden, Qualitätssicherer oder anderweitig interessierte Mitarbeiter schließen dann einfach ihr iPhone oder iPad an die Tankstelle an und installieren per iPhone-Konfigurationsprogramm den aktuellen Stand der App auf ihr Gerät.
- TestFlight [t] zur Auslieferung der Builds auf die Endgeräte der Tester, zur Überwachung des Testverhaltens, zum Einholen von Feedback und zur Protokollierung etwaiger Crashes.

Die vollständige Automatisierung des Build-Prozesses über Jenkins ist relativ problemlos möglich [u] und entlastet die Entwickler spürbar, die sich nun besser auf die eigentliche Entwicklungsarbeit konzentrieren können. Projektverantwortliche und Kunden können gegebenenfalls täglich die Tankstelle aufsuchen und sich jederzeit einen Überblick über den aktuellen Stand der App verschaffen. Das wiederum ermöglicht ein frühzeitiges Feedback im Hinblick auf technische und fachliche Fehler, die sich schnell an das Entwicklerteam kommunizieren lassen.

Aber nicht immer können alle Projektverantwortlichen den aktuellen Build von der Tankstelle beziehen. In dem Fall erfolgt bei compeople die Auslieferung der Builds zusätzlich über den kostenlosen Webdienst TestFlight. Er protokolliert neben dem Deployment der Builds bei den Testern deren Testverhalten und Feedback. Auch das unbeliebte Einsammeln der für das Code Signing notwendigen Unique Device Identifiers (UDIDs) der Testgeräte unterstützt der Dienst automatisiert. Der Einsatz von TestFlight ist ausgesprochen einfach: Nach Aufsetzen eines iOS-Projekts auf der Webseite durch einen Entwickler lassen sich die relevanten Testpersonen einladen und in Gruppen organisieren. Entwickler können dann ihre Builds in TestFlight hochladen (entweder manuell oder als Teil des Build-Prozesses in Jenkins) und bestimmen, welche Testgruppen mit dem neuen Build versorgt werden sollen. Die so bestimmten Testpersonen bekommen daraufhin eine E-Mail, die einen Link zur automatischen Installation des Build enthält.

Entwickler können sich jederzeit informieren, welcher Tester welchen Build zurzeit installiert und wie ausführlich er diesen getestet hat sowie auf welche Probleme er bei der Bedienung gestoßen ist. Benutzt der Build das kostenfreie TestFlight SDK, lassen sich zusätzlich und mit geringem Entwicklungsaufwand detaillierte Informationen wie Checkpoints (wichtig zur Ermittlung der Test Coverage), Crash Logs oder Feedback der Tester automatisch einsammeln und durch die Entwickler jederzeit einsehen. Die Veröffentlichung eines Build per TestFlight als integraler Bestandteil des automatisierten Build-Prozesses ist durch TestFlights Upload API mit Jenkins verhältnismäßig einfach zu realisieren.

Mit den steigenden Erwartungen der Kunden an Apps gewinnt auch die enge Zusammenarbeit mit diesen an strategischer Bedeutung. Hierfür haben sich agile Entwicklungsmethoden als geeignet für das Meistern der Anforderungen an die Projekt-Teams erwiesen. Der Aspekt Continuous Integration mit seinen Grundsätzen „Check-in early and often“ und „Build often“ hilft dabei, die Entwickler frühzeitig auf technische und fachliche Probleme und Unzulänglichkeiten aufmerksam zu machen, um hierdurch ressourcenraubende „Sackgassen“-Entwicklungen zu vermeiden und eine auf die Zielgruppen ausgerichtete (Stichwort „User-Brille“), robuste Software hervorzubringen. Der Kasten „ Continuous Integration für iOS“ beschreibt, wie die compeople AG erfolgreich Continuous Integration im Rahmen ihrer iOS-Projekte realisiert hat.

Fazit

Die Entscheidung über eine gute, langfristig tragfähige Entwicklungstechnik für mobile Anwendungen ist in der Regel nicht einfach zu treffen. Eine Vielzahl teilweise miteinander konkurrierender Faktoren beeinflusst die Auswahl erheblich, und die naheliegende Wahl ist nicht immer die beste. Vor allem gilt es, verstärkt betriebswirtschaftliche Faktoren in die Entscheidungsfindung mit einzubeziehen.

Die wichtigsten Empfehlungen für die App-Entwicklung sind vor diesem Hintergrund:

- Sich über wichtige betriebswirtschaftliche und technische Anforderungen klar werden und diese gegebenenfalls anhand eines Scoring-Modells [r] bewerten, um zu einer Entscheidung bezüglich des geeigneten Entwicklungsansatzes zu kommen.
- Sich auf die Endgeräte fokussieren, die die Mehrheit der Zielgruppe verwendet (hier helfen bei der Entwicklung von Android-Apps womöglich Googles Statistiken zur Verbreitung von Gerätearten und Softwareversionen [s].
- Sich im Vorfeld des Projekts mit etwaigen Review Guidelines des Betriebssystemherstellers auseinandersetzen und diese bei der Konzeption und Entwicklung der App berücksichtigen, um böse Überraschungen bei der Review der nativen oder hybriden App zu vermeiden. Gegebenenfalls ist zu prüfen, ob es nicht besser ist, auf eine Webapp auszuweichen.
- Stehen Multi-Plattform-Fähigkeit, Vorlaufzeit und Entwicklungskosten der App zusammen im Vordergrund, bietet sich eine Web- oder hybride App an. Hierbei ist darauf zu achten, dass ein geeignetes Framework eingesetzt sowie hauptsächlich auf die durch das Framework bereitgestellten Komponenten und Funktionen gesetzt und möglichst wenig auf eigene neue Kreationen zurückgegriffen wird. Stehen maximale Performance und/oder das Ausreizen gerätespezifischer Funktionen im Vordergrund, führt derzeit kein Weg an einer nativen App vorbei.
- Möglichst früh und regelmäßig während der Entwicklung auf den Zielgeräten testen, idealerweise unter Einbeziehung geeigneter Anwender aus den Zielgruppen. Der Grundsatz lautet hier: Continuous Integration ist des Entwicklers bester Freund. (ane)

Christian Rühl
ist als IT-Berater und Softwareentwickler bei der compeople AG in Frankfurt tätig. Er ist seit 2009 bei der compeople AG für die Architektur und Entwicklung mobiler (Web-)Anwendungen für iOS-Geräte verantwortlich.

Thorsten Schenkel
ist als IT-Berater und Softwareentwickler bei der compeople AG tätig. Er beschäftigt sich seit 1999 intensiv mit Client-Server-Anwendungen sowie grafischen Benutzeroberflächen im Java-Umfeld.

Alle Links: www.ix.de/ix1217105