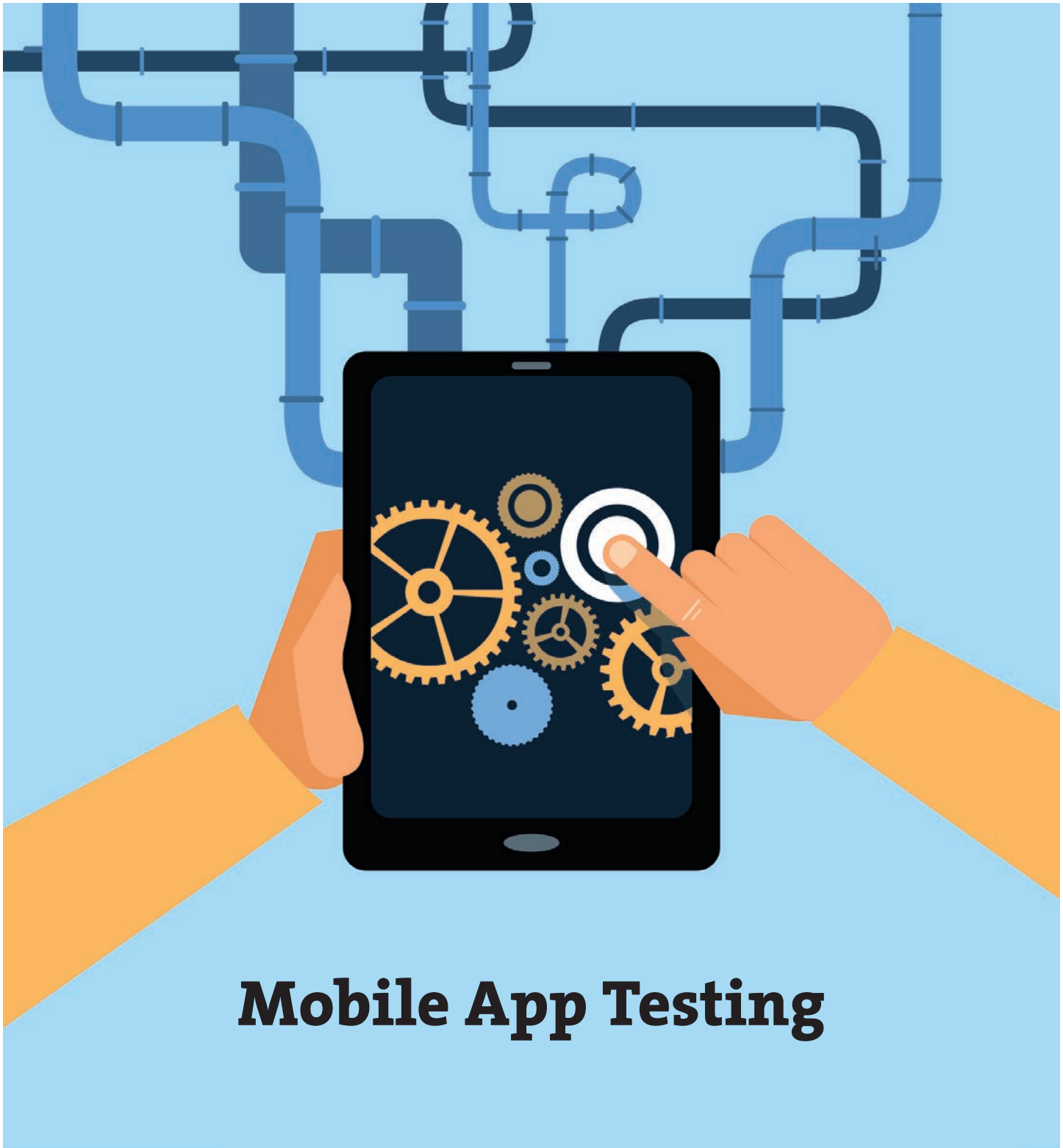
testing experience de

Das Magazin für Tester, Entwickler und Manager

Nr. 3 | Oktober 2013
kostenlose Ausgabe



# Mobile App Testing



Díaz Hilterscheid

# Automatische Akzeptanztests für mobile Anwendungen: Überlegungen zur Teststrategie


Die Autoren

**Alexandra Schladebeck**

Alexandra Schladebeck arbeitet seit 2005 bei der BREDEX GmbH. Ihre Rolle lässt sich am besten als „Übersetzerin" zwischen dem Entwicklungsteam und der Kundschaft beschreiben. Sie kommuniziert Erfahrungen aus dem Entwicklungsprozess mit Kunden und auf Konferenzen und arbeitet als Product Owner für das GUIdancer/ Jubula-Projekt.

**Markus Tiede**

Markus Tiede arbeitet als Softwareentwickler und Testberater bei der BREDEX GmbH (*www.bredexsw.com*) mit den Schwerpunkten Eclipse-RCP-Entwicklung und Entwurf und Entwicklung automatischer Tests und gehört zum GUIdancer-Entwicklungsteam. Markus ist darüber hinaus Eclipse Committer im UI-Testautomatisierungsprojekt Jubula (*www.eclipse.org/jubula*), Package Maintainer für „Eclipse for Testers" und hat einen Abschluss als Diplom-Informatiker von der FH Braunschweig-Wolfenbüttel.


Ob individuell entwickelte Enterprise-Apps oder Apps aus dem Store – die Meinung zu einer App wird schnell gefällt. Noch nie war es so einfach, eine Anwendung auszuprobieren und bei Nichtgefallen zu deinstallieren. Die entsprechenden öffentlichen Bewertungen oder das Feedback vom Kunden können den Ruf und das Image der App und auch der dahinter stehenden Firma schnell verbessern oder verschlechtern.

Die Benutzerperspektive anhand von Akzeptanztests zu vertreten ist deshalb für mobile Anwendungen besonders wichtig. So kann man frühzeitig feststellen, ob eine App einerseits das gewünschte Verhalten besitzt und andererseits keinerlei unerwünschten Nebeneffekte bei der Bedienung auftreten. Wie für jedes Projekt muss dafür eine geeignete und angemessene Teststrategie einschließlich Testarten und -methoden, Automatisierungsgrad und Zielplattformen definiert werden. Besondere Herausforderungen lauern allerdings beim automatischen Testen mobiler Apps, die – so viel sei schon vorweg gesagt – Auswirkungen auf Ihre Teststrategie haben werden.

## Vorüberlegung: Gehören automatisierte Akzeptanztests überhaupt zu meiner Strategie?

Provokativ gesagt: Wenn man weniger als eine Stunde benötigt, um eine neue Version der App „durchzuklicken", darf sich dieser Aufwand auch für verschiedene Geräte oder Plattformen verfünf- oder verzehnfachen, ohne in „zu viel manuelles Testen" auszuufern. Bei solchen Voraussetzungen könnte man sich fragen, ob eine Testautomatisierung sich überhaupt lohnt.

Die Versuchung, manuelle Testaufwände auf diese Art und Weise herunterzuspielen und zu pauschalisieren, ist gerade im mobilen Umfeld groß. Größe und Komplexität der App dürfen jedoch nicht alleine für die Teststrategie ausschlaggebend sein. Vielmehr sind es die geplanten Zyklen, die zu erreichende Qualität, Time-to-Market Wünsche und die Zielplattformen sowie -versionen, die eine Entscheidung für oder gegen eine Automatisierung der Akzeptanztests beeinflussen.

## Argumente für eine Testautomatisierung

### Und wenn sich etwas ändert ...

In dieser Hinsicht ist das Testen von mobilen Apps genau wie das Testen jeder anderen Anwendung zu bewerten. Wenn es wichtig ist, die App nach Änderungen kontinuierlich und in kurzen Zyklen überprüfen zu können, kommt man um eine Automatisierung dieser Schritte nicht herum. Denn auch einstündige Tests können nur begrenzt tagein, tagaus ausgeführt werden – sowohl aus Sicht der Teamkapazitäten als auch aufgrund der mentalen Unterforderung und der damit verbundenen Unzuverlässigkeit der Ergebnisse.

### Mehr, als man ahnt

Je nach Projekt kann die obige Aussage über den minimalen Aufwand auch komplett daneben liegen: Die Kombinationsvielfalt der unterschiedlichen Geräte, Betriebssysteme und Versionen einzelner Plattformen macht es klar: Das kann man nicht alles schaffen! Dass Testen in vielen Fällen nur eine risikobewertete Stichprobe sein kann, wissen wir bereits. Jedoch sind die Varianten von mobilen Geräten so zahlreich und so verteilt, dass Stichproben von zwei bis drei Systemen (ein durchaus erstrebenswertes Ergebnis für Desktopanwendungen) keinesfalls eine repräsentative Ergebnismenge darstellen. Automatisierung stellt hier den entscheidenden Schlüssel zur Effizienzsteigerung bereit, um verschiedenste Varianten und Versionen mit geringem zusätzlichem Aufwand in kurzer Zeit bewältigen zu können.

### „Ich war es nicht"

Das Stichwort „Version" ist besonders im mobilen Umfeld sehr heterogen zu sehen. Neue Betriebssystem-Versionen erscheinen sehr häufig und die Unterschiede zwischen diesen können groß sein. Und daher ist es sehr wichtig, diese neuen Versionen frühzeitig in die Qualitätssicherung zu integrieren. Denn auch eine nicht geänderte Software kann alleine durch einen Versionswechsel des Anwenders

fehlerhaftes Verhalten aufgrund von geändertem API aufzeigen. Erschwerend kommt hinzu, dass der App-Hersteller nie die Kontrolle darüber hat, wann der Anwender seiner Software auf eine neue Version umsteigt, und dabei unter Umständen auch verschiedene Versionsstände überspringt. Die Testautomatisierung kann hier helfen, peinliche Fehler, wie beispielsweise den Absturz der App, zu vermeiden, indem man komplette Test-Suiten frühzeitig, schnell und damit effizient ausführt und auswertet.

### Die Zeit tickt

Apps unterliegen generell kürzeren Release-Zyklen mit häufigen Änderungen. Schnell sein ist wichtig, um mit der Konkurrenz Schritt zu halten. Geschwindigkeit auf Kosten der Qualität ist allerdings sehr kontraproduktiv – vor allem wenn eine App einem längeren Review-Prozess unterworfen ist, wie zum Beispiel im App-Store von Apple. Aber auch wenn die Auslieferungsphase kurz ist, muss die Qualität auf Anhieb stimmen – 4 bis 5 Stern-Bewertungen sind ein Muss, um auf lange Sicht eine breite Akzeptanz zu finden. Interne Kunden-Apps haben zwar keinen öffentlichen Review-Prozess, aber „5*" sind trotzdem anzustreben, gerade weil dann eine sehr hohe Kundennähe vorhanden ist.

## Argumente gegen die Testautomatisierung

Trotz aller Vorteile der Testautomatisierung kann es durchaus Situationen geben, in denen es nicht wichtig genug ist, den benötigten Minimalaufwand zu investieren.

### Eintagsfliegen

Eine solche Situation liegt zum Beispiel vor, wenn eine App einen sehr kurzen Lebenszyklus (weniger als drei Monate) besitzen wird. Dazu zählen Anwendungen, die beispielsweise lediglich für einen bestimmten Zeitraum benötigt werden (man denke an Konferenz-Apps, WM-Apps usw.). Analog für viele Bereiche des Softwareentwicklungsprozesses für eine solch kurzlebige Applikation wird man auf weitgehende Automatisierung der Prozesse, was auch die Testautomatisierung betrifft, verzichten. Wenn zusätzlich nur wenige vereinzelte Releases vorgesehen sind und keine Updates des Betriebssystems während der Anwendungszeit anstehen, dann reicht höchstwahrscheinlich eine manuelle Testphase vor den Auslieferungen.

Aber Vorsicht! Gerade solche Apps erfordern eine exakte Punktlandung für ihr Release. Eine WM-App, die erst nach dem Turnier erscheint, wird keiner mehr benutzen. Für solche Situationen gilt wiederum, dass man laufend Information über die Qualität benötigt. Wie das Team diese Informationen zeitnah und häufig beschaffen kann, muss individuell entschieden werden.

### Bekannte Eingrenzungen

Wenn die zu entwickelnde App in ihrem Umfang, ihrem Nutzerkreis und ihrer Zielumgebung eingegrenzt ist, entfällt ein Großteil der Variationsvielfalt. Wenn für einen speziellen Kundenkreis entwickelt wird, können Zielplattformen festgelegt beziehungsweise priorisiert werden. In solchen Fällen reduzieren sich die Testaufwände drastisch. Allerdings muss man sich auch hier Gedanken über die Lebenserwartung der App und die gewünschte Qualität machen. Wenn mehrere Zyklen mit jeweils neuen Funktionen über Betriebssystemversionen hinweg geplant sind, dann ändert sich gegebenenfalls die Teststrategie.

### Jemand anders macht es

Eine andere Alternative, die gerade für das Testen mobiler Applikationen attraktiv sein kann, ist, die Tests in die Crowd zu geben. Über Crowd-Sourcing kann eine viel größere Fülle an Systemen getestet werden, ohne selber jede Gerät-Betriebssystemversion-Kombination vorrätig zu haben. So kann Crowd-Sourcing ein Teil der Teststrategie darstellen, wenn die notwendige Infrastruktur für Koordination und Kommunikation der Tests und ihrer Ergebnisse vorliegt. Allerdings eignet sich dieser Ansatz nur für Apps, deren Sicherheitsrichtlinien dies erlauben: Denn gerade auf mobilen (*bring your own device* – BYOD) Geräten ist der adäquate Umgang mit sensiblen Kundendaten und Anwenderinformationen das A und O, um eine langfristige Vertrauensbasis aufzubauen. Herausfordernde Probleme in diesem Bereich sind außerdem die Reproduzierbarkeit von Fehlerzuständen, deren Analyse auf Seite der Entwickler und der nachfolgende Regressionstest: Denn wenn nur die Crowd entsprechende Geräte und Szenarien besitzt, um das Problem nachzustellen beziehungsweise den Test durchzuführen – wie soll der Entwickler eine solche Fehlersituation selbst nachstellen, debuggen, beheben und den Fix nachhaltig verifizieren, wenn er keinen Zugang zum Gerät, der Version und der Umgebung hat?

## Überlegung: vollautomatisch oder teilautomatisch?

Nehmen wir Folgendes an: Die Teststrategie sieht vor, dass manche Akzeptanztests automatisiert werden. Automatisiert heißt aber nicht gleich vollautomatisiert. Es gibt verschiedene Möglichkeiten, automatisierte Tests laufen zu lassen. Aus der Erfahrung mit Desktop-Anwendungen kennt man die komplette „Continuous Integration and Testing"-Kette als das Herzstück, das alle Elemente zusammenhält: aus„checken", bauen, deployen, die Testumgebung aufsetzen, konfigurieren, Tests durchführen, Ergebnisse sammeln und aufräumen. Das Gleiche ist auch für das Testen mobiler Anwendungen möglich, allerdings gibt es hierfür teilweise bewusst Restriktionen namhafter Hersteller und daraus resultierend eine gestiegene Komplexität, die das Aufsetzen und die Pflege eines solchen CI-Prozesses aufwändiger machen.

### Gründe gegen die Vollautomatisierung – die Herausforderungen

Schon das Hinzufügen weiterer Systeme zur CI-Toolkette führt zu einer komplexeren Koordination des gesamten Prozesses. Für die Continuous Integration plus Tests im Desktop-Bereich braucht man einen Build-Server und (virtualisierte oder physikalische) Testmaschinen, die vom Build-Server aus erreichbar sind. Will man Akzeptanztests auf einem realen mobilen Gerät durchführen, so muss das Gerät über eine Schnittstelle angebunden sein, die es erlaubt, die Software darauf zu deployen und Tests laufen zu lassen.

Erschwerend kommen hier die Restriktionen im Bereich Software und Hardware hinzu. Für iOS-Tests sind ein Mac sowie XCode und ein iOS-Gerät ein Muss. Für Android und Windows sehen die Voraussetzungen und Abhängigkeiten wieder ganz anders aus, und jede Plattform bringt darüber hinaus seine Eigenheiten mit. Kurzum: Um Tests auf den Geräten der drei größten Anbieter laufen lassen zu können, muss eine Vielzahl unterschiedlicher Hard- und Softwarekomponenten in die Automatisierungskette eingebunden werden.

Es muss auch dafür gesorgt werden, dass eine installierte App gestartet und nach dem Test wieder gestoppt wird. Aufgrund des Sandbox-Prinzips auf vielen mobilen Systemen besitzen Apps jedoch wenig Einfluss auf ihren eigenen Lebenszyklus und die Kopplung zu systemnahen Funktionen. Die Schritte zum Starten und Stoppen müssen deshalb extern vom Testtool

plattformabhängig gelöst werden. Einen Einfluss auf die verwendete Laufzeitumgebung hat man ebenfalls nur sehr begrenzt.

Dabei ist aber der Verzicht auf reale Geräte keinesfalls eine Lösung. Simulatoren sind zwar von vielen Netzwerk- und Kommunikationsproblemen befreit, aber sie setzen häufig genauso ihre bevorzugte Software voraus. Außerdem gibt es eine Reihe bekannter Unterschiede zwischen Simulatoren und physikalischen Geräten, zum Beispiel in den Bereichen Performance und UI-Rendering.

Nach der Lösung aller Beschaffungs- und Prozesskommunikationsprobleme bleiben weiterhin viele Aspekte offen, die gerade für die Continuous Integration herausfordernd sind. Dazu gehören: automatischer Orientierungswechsel im Test, GPS-Abfragen, Auswirkung der durchgeführten Aktionen auf den Akku, Umgang mit fehlender/schlechter/vorhandener Netzverbindung sowie die Anbindung an externe Apps und Dienste. Hierfür muss man sich fragen, welche Informationen eine Teil- oder Vollautomatisierung liefern soll und wie viel diese Informationen wert sind.

### Warum es sich trotzdem lohnen kann – die Vorteile

Trotz der langen Liste an Herausforderungen gibt es gute Gründe für eine weitreichende Automatisierung. Teilweise handelt es sich dabei um genau dieselben Argumente, die für automatisierte Tests sprechen: häufig zeitnahe Informationen zur Qualität der Software zu bekommen. Wer einmal am Tag oder sogar mehrmals täglich bauen und Akzeptanztests durchführen will, wird schnell den Wert eines durchgehend automatisierten Prozesses schätzen. Auch bei großen oder verteilten Teams können sich die manuellen Aufwände – gerade auch zur schnellen und zeitnahen Fehlerbehebung – deutlich verringern.

### Der Mittelweg

Glücklicherweise ist die Prozessautomatisierung nicht binär. Man kann (und soll) den Automatisierungsgrad iterativ wachsen lassen. Ein automatisierter Build ist häufig der erste sinnvolle Schritt. Sobald das Deployment der Anwendung danach „angehängt" wird, können zeitnah manuelle oder manuell gestartete automatisierte Tests durchgeführt werden. Der Schritt, auch automatische Tests direkt nach dem Deployment anzustarten, kann zunächst für ein ausgewähltes System oder Gerät erfolgen. Nach und nach und bei Bedarf lassen sich dann andere Plattformen der Kette anschließen. Hier ist der Leitgedanke: Den nächsten Schritt wählen, sodass aktuell getriebener Aufwand reduziert wird, Fehlerquellen geringer werden oder um schneller an wichtige Informationen zu gelangen.

## Überlegung: Wie viele gleiche Tests möchte ich schreiben?

Ein weiterer Aspekt der Teststrategie ist das Thema Plattform(un)abhängigkeit der Tests. Der Markt für mobile Geräte, Systeme und Versionen ist stark fragmentiert und die Unterschiede zwischen Herstellern und ihren Konzepten sind bedeutsam. Inwiefern die Entwicklung möglichst Cross-Plattform getrieben wird, muss an anderer Stelle entschieden werden. Aber „Cross-Plattform" ist genauso ein Diskussionsthema für die Testautomatisierung.

### One test to rule them all?

„Cross-Plattform-Testautomatisierung“ ist längst kein Neuland für Desktop-Anwendungen. Ein Test, der auf Windows, Linux und MacOS läuft (jeweils auf verschiedenen Versionen) ist nicht überraschend. Ganz anders sieht es für mobile Tests aus. Auch wenn ein Testautomatisierungstool eine Abstraktionsebene zwischen dem Treiber und der Testspezifikationssprache bietet, heißt das längst nicht, dass sich die Anwendungen auf den unterschiedlichen Systemen gleich bedienen lassen oder sie gar die gleichen Funktionen aufweisen. Diese Unterschiede sind sowohl in Projekten zu finden, die auf Cross-Plattform-Entwicklung setzen, als auch in Projekten, die jede Plattform für sich entwickeln. Den Benutzerkreisen sind die Unterschiede sogar lieb und wichtig: Der Versuch, eine App auf verschiedenen Systemen komplett gleich bedienbar zu machen, stößt auf heftige Inakzeptanz. Die Erwartungskonformität innerhalb der Plattform eines Herstellers ist dabei entscheidend.

Kurz gesagt: Es ist zwar technisch möglich, einen einzigen Test für verschiedene Plattformen zu nutzen. Allerdings zeigt die Realität, dass Navigation, Bedienung, Workflows und Funktionalitäten von Multi-Plattform-Apps zu unterschiedlich sind, um die Unterschiede sinnvoll fachlich abstrahieren zu können.

### Redundanzen vermeiden

Aus der Perspektive eines Testers für Desktop-Anwendungen ist diese Feststellung irritierend. Man kennt die kleinen Unterschiede zwischen Windows und Mac (zum Beispiel in der Menü-Bedienung), aber die Vorstellung, dass komplette Navigationskonzepte und Workflows zwischen Desktop-Varianten einer Anwendung ersetzt werden oder gänzlich fehlen, ist unvorstellbar.

Es kann natürlich sehr aufwändig werden, für jede Plattform (eventuell für jede Version einer Plattform) einen kompletten Satz an separaten Tests zu schreiben. Hier gilt: In enger Kommunikation zwischen Entwicklungsteam(s), Testern und Kunden (falls vorhanden) sollten Kandidaten für „One Workflow – One Test“ identifiziert werden. Jeder dieser Bereiche hilft dem Team, den Automatisierungsaufwand zu verringern. Bei zu heterogenen Anwendungsfällen, die auf jedem System unterschiedlich getestet werden müssen, muss die Wiederholbarkeit der Tests für den Return-Of-Investment sorgen.

## Die eine passende Strategie?

Was bedeuten diese Überlegungen für die Teststrategie? Leider kann hier keine gemeingültige Antwort gegeben werden. Jedes Projekt muss eine eigene Strategie anhand des ihm gegebenen Kontexts entwickeln. Die Informationen aus den vorherigen Absätzen sollen dazu dienen, die Wahl der Strategie besser einschätzen und planen zu können. Fazit: Bestimmte Projektmerkmale können benannt werden, die die Wahl der Strategie in die eine oder andere Richtung lenken könnten.

### Merkmal 1: Kundenkreis/Markt kennen

Der erste Hinweis lautet: know your user. Für Individualentwicklungen ist dies sicher einfacher, denn man kann den Kunden direkt nach Benutzerkreisen, Prioritäten und Plattformen fragen. Für öffentliche Apps muss man sich mit dem Markt sehr genau auseinandersetzen: Aktuelle Verteilung und Trends der Plattformen und Versionen, Zielgruppe und Prioritätsgruppe. Wie bereits zuvor erwähnt: Es kann nicht alles getestet werden, und die Teststrategie soll vorsehen, dass die wichtigsten Komponenten auf den Systemen mit dem größten Verbreitungsgrad zuerst und am umfassendsten getestet werden.

### Merkmal 2: Individualsoftware/ öffentliche Software

Wie im vorherigen Absatz schon angedeutet hat der Unterschied zwischen einer Individualentwicklung und einer allgemeinen Lösung große Auswirkungen auf die Teststrategie. Bei einer Individualsoftware ist ein Kunde anwesend, mit dem man sich abstimmen kann. Auf der anderen Seite fallen die Möglichkeiten wie Crowd-Testing komplett weg: Mit geheimen Kundendaten kann man in den wenigsten Fällen aus den eigenen vier Wänden heraus. Bei einem Kunden, dessen Mitarbeiter die neue App benutzen müssen, hat man weder den 5-Sterne-Druck noch die längeren Review-Prozesse (es sei denn, der Kunde hat selbst ähnliche Einrichtungen). Nichtsdestotrotz möchte man keinesfalls einen Image-Schaden für eine ungelungene Entwicklung riskieren.

Für öffentliche Apps wiederum ist der Druck, einen guten Ruf zu bekommen (und zu behalten), sehr präsent. Die Teststrategie muss sowohl diesen Aspekt als auch die Abdeckung vieler Systeme miteinbeziehen. Dafür ist die Option, Unterstützung über externe Dienste zu holen, in diesem Fall einfacher als bei einer Individuallösung. Anders als bei Software für einen Kunden ist hier aber der Auslieferungszeitpunkt eventuell wichtiger: schnelles Erscheinen mit guter Qualität und User Experience, um Benutzer nachhaltig zu begeistern.

### Merkmal 3: Wann rechnet es sich?

Der Vorteil der Automatisierung liegt in der Wiederholbarkeit. Es lohnt sich (wie in jedem Projekt), ein Rechenbeispiel zu erstellen: über den Projektzeitraum, die geplanten Releases, eventuelle Updates des Betriebssystems sowie die Menge an zu testenden, automatisierbaren Anwendungsfällen. Somit kann das Team einsehen, welche Automatisierungen sich am ehesten lohnen werden, und kann sich als Erstes darauf konzentrieren.

### In allen Fällen

Die mobile Welt ist gleichzeitig durch ihre Diversität und Schnelllebigkeit gekennzeichnet. Trotz der im Artikel erwähnten Herausforderungen gibt es sehr gute Gründe, warum man auf die aktuellen und zeitnahen Informationen, die automatisierte Akzeptanztests liefern, nicht komplett verzichten kann. Wofür und an welche Stellen diese Tests einzusetzen sind, bleibt schlussendlich Teil der individuellen Teststrategie. ■