Patrick Döring
(patrick.doering@sogeti.de)
verfügt über tiefe Kenntnisse im Testmanagement und in der Testautomatisierung sowohl von webbasierten als auch mobilen Applikationen. Seit 2011 ist er für die Sogeti Deutschland GmbH als Consultant tätig.

Marcel Gaßen
(marcel.gassen@sogeti.de)
ist als Consultant für die Sogeti Deutschland GmbH tätig. Er befasst sich seit mehr als fünf Jahren mit dem Thema Mobile Testing und ist momentan im automobilnahen Umfeld als Testmanager für mobile Anwendungen im Einsatz.

Mobile Testautomatisierung
Worauf man beim Test mobiler Applikationen achten sollte

Testautomatisierung ist doch immer das Gleiche? Gerade im mobilen Bereich gilt es, die Besonderheiten und Unterschiede gegenüber der klassischen Testautomatisierung herauszufinden und die Fallstricke zu meistern. Die hohe Produktvielfalt von Smartphones im Handel stellt eine enorme Herausforderung für viele Projekte dar. Im Folgenden wird auf die Gefahren eingegangen, die zu einer enormen Kostensteigerung der Testautomatisierung führen können, und auf Methoden, wie diese vermieden werden kann.

Mobile Apps sind aus dem täglichen Leben nicht mehr wegzudenken. Angefangen von schlichten News-Apps über Dienstleistungs-Apps bis hin zu komplexen Enterprise-Apps, mit denen beispielsweise Budgetplanungen durchgeführt werden. Sie haben den Fokus von den klassischen Computern hin zu mobilen Endgeräten verschoben.

Die meisten Anwender haben vermutlich schon einmal schlechte Erfahrungen mit einer mobilen Anwendung gemacht, die nicht so funktioniert hat, wie sie es erwartet haben. Eine schlechte Performance, eine umständliche Usability oder ein plötzlicher Absturz der App führt bei Anwendern schnell zu einer Deinstallation und schlechten Bewertungen in den App Stores. Die Bewertung von Kunden kann sowohl die Akzeptanz von Apps als auch das Image der dahinter stehenden Unternehmen negativ beeinflussen.

Um das zu vermeiden, sind qualitativ hochwertige Apps notwendig, für die regelmäßige Tests ebenso unumgänglich sind. Gerade diese häufigen Wiederholungen lassen sich durch eine sinnvolle Testautomatisierung effizient gestalten. Dabei gibt es verschiedene Herausforderungen, die es zu bewältigen gilt. So ist es ratsam, Kompromisse einzugehen und Testautomatisierungstechniken und -herangehensweisen zu einem perfekten Mix miteinander zu verbinden (siehe Abbildung 1).

Ziele und Erfolgsfaktoren einer möglichen Testautomatisierung

Wenn man sich einmal die Vielfalt der auf dem Markt vorhandenen Smartphones anschaut, wird auch das nicht geschulte Auge erkennen, dass man mit einem einzelnen Gerät nicht alles abdecken kann. Neben verschiedenen Herstellern finden wir auch unterschiedliche Betriebssysteme wie auch verschiedene Softwarestände eines Gerätes vor. Da sich die Release-Zyklen der Betriebssysteme häufig ändern, werden auch die Entwicklungszyklen von Apps kürzer. Als logische Konsequenz stehen neben der Entwicklung auch erneute Tests bevor.

So zeigt sich, dass Testen von mobilen Apps auf mehreren Geräten oftmals ein

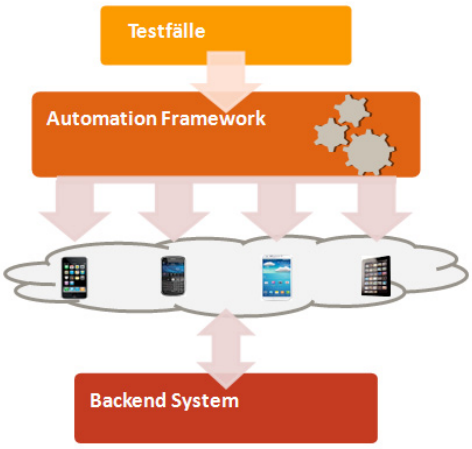

Abb. 1: Komponenten einer Testautomatisierung im mobilen Umfeld

Abb. 2: Gerätevielfalt auf dem Markt

sich wiederholender und zeitraubender Prozess ist. Gerade aus den erwähnten Gründen ist es wichtig, funktionale und nichtfunktionale Anforderungen von mobilen Apps so effektiv und effizient wie möglich zu testen. Das wichtigste Ziel bei einer erfolgreichen Testautomatisierung ist es, einen Testfall oder eine Test-Suite auf verschiedenen mobilen Geräten auszuführen.

Dieses Ziel ist von drei Faktoren abhängig: Zuverlässigkeit, Wartbarkeit und Skalierbarkeit. Unter Zuverlässigkeit versteht man die automatische Ausführung der Tests, eine exakte Prüfung der einzelnen Testschritte sowie eine automatische Reporterstellung. Der Faktor Wartbarkeit umfasst im Wesentlichen minimale Empfindlichkeit bei Anwendungsänderungen und effiziente Anpassbarkeit der Testfälle sowie eine Testfalldefinition getrennt vom Automatisierungscode. Das Ziel der Skalierbarkeit ist es, bei veränderten Anforderungen oder Steigerung der Testaktivitäten die eingesetzten Tools entsprechend aufzurüsten, um diese neue Last verarbeiten zu können.

Dieser Ansatz bringt generelle Herausforderungen mit sich. Zusätzlich zu den drei primären Faktoren muss bei einer Automatisierung der Umfang (Scope) – speziell die Device-Auswahl – im Vorfeld festgelegt werden, bevor diese in den Software Development Life Cycle (SDLC) integriert werden kann. Bei einer mobilen Testautomation müssen aber auch mobile-spezifische Herausforderungen wie Plattform- beziehungsweise Gerätevielfalt (siehe Abbildung 2) ebenso wie Toolvielfalt und Toolreife beziehungsweise -unreife beachtet werden. Hinzu kommt noch der Mangel an qualifiziertem Personal. Last but not least sollte man die hohe Rate des technologischen Wandels und die schnellen Entwicklungszyklen im Auge behalten, was eine genaue Planung unabdingbar macht.

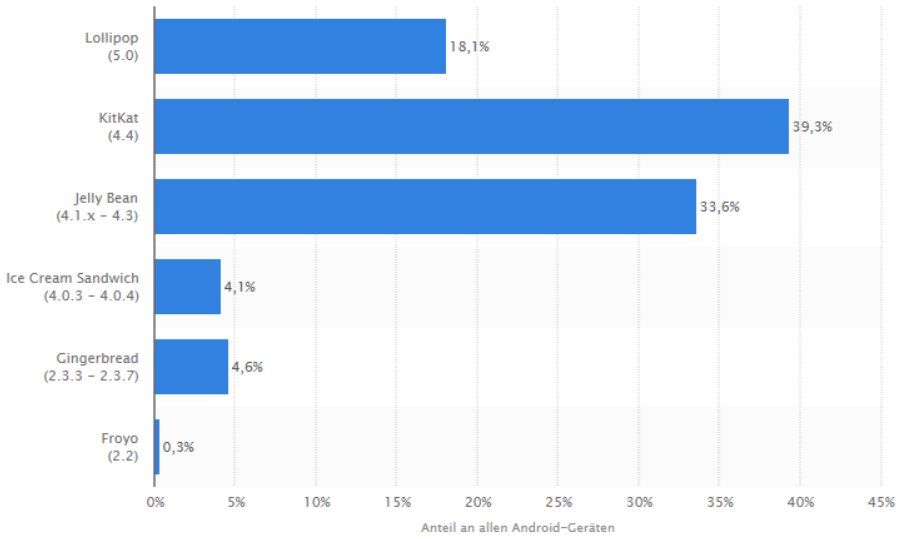

Abb. 3: Verteilung Android-Betriebssysteme

Umfang der Geräteauswahl

Bei der Planung müssen verschiedene Bereiche wie Umfang, Technik und Prozessintegration betrachtet werden. Allem voran geht natürlich die genaue Kenntnis von Markt und Kunden. Dazu können Informationen (siehe Android-Beispiel in Abbildung 3) aus den verschiedenen Stores (z. B. Apple App Store, Google Play Store oder Windows Marketplace) gesammelt werden.

Für den Umfang ist es vor allem wichtig, die richtigen Geräte und Betriebssysteme auszuwählen. So kann man mithilfe einer Device-Matrix entsprechende Merkmale priorisieren (z. B. OS-Version, Bildschirmauflösung, Speicher, CPU-Architektur) und mit wenigen Geräten trotzdem eine breite Abdeckung der marktüblichen Kombinationen erreichen. Emulatoren können bei Projekten eine sinnvolle und kostengünstige Ergänzung zu realen Devices sein, sollten jedoch niemals die einzige Testoption darstellen. Virtuelle Devices eignen sich in der Praxis vorrangig für rein funktionelle Tests, in Bereichen mit Performance- oder Stabilitätsaspekten ist es nicht ratsam, sich auf diese Optionen zu verlassen. Emulatoren können Situationen wie beispielsweise eingehende Anrufe oder Verbindungsabbrüche (WLAN/3G) bei Datenübertragungen lediglich simulieren. Weiterhin arbeitet auch nicht jedes Testautomatisierungstool einwandfrei mit Emulatoren zusammen, sodass hier möglicherweise zusätzliche Anpassungen erforderlich sein werden.

Neben diesen Punkten können auch herstellertypische Standards noch eine Herausforderung für die geplante Testautomatisierung darstellen. So befindet sich die Zurück-Schaltfläche bei Geräten von Samsung grundsätzlich auf der rechten Seite, während der Hersteller HTC diese Schaltfläche standardmäßig auf die linke Seite setzt.

Im nächsten Schritt ist es wichtig, eine Aufteilung in drei Gruppen (siehe Tabelle 1) vorzunehmen. Dieses Vorgehen eignet sich gleichermaßen für Tablets. Die Einteilung erfolgt dabei nach den folgenden Leistungsmerkmalen:

- Topaktuelle high-end Devices mit hochauflösenden Displays (z. B. Full HD 1920x1080), Multi-Core-CPUs, 1 bis 4 GB RAM und den aktuellsten Betriebs-

Device	A	B	C
iPhone 6/6 Plus mit iOS 8.4	X		
iPhone 5S/5C mit iOS 8.4	X		
Samsung Galaxy S6/ S6 Edge mit Android 5.1	X		
Samsung Galaxy S5 mit Android 5.1	X		
HTC one M8/M9 mit Android 5.1	X		
Google Nexus 5/6 mit Android 5.1	X		
Sony Xperia Z3 Compact mit Android 5.1	X		
LG G Flex2 mit Android 5.1	X		
iPhone 5 mit iOS 6		X	
Samsung Galaxy S4/S4 Mini mit Android 4.2		X	
Google Nexus 4 mit Android 4.2		X	
Sony Xperia Z1 mit Android 4.2		X	
iPhone 4S mit iOS 5			X
iPhone 4 mit iOS 4			X
Samsung Galaxy S II mit Android 2.3			X
HTC Sensation mit Android 2.3			X
Google Nexus One mit Android 2.2			X

Tabelle 1: Kategorisierung handelsüblicher Smartphones

systemversionen haben höchste Priorität (A).

- Weitverbreitete Geräte der letzten Jahre verfügen über Single- oder Dual-Core-CPUs, ca. 1 GB RAM und eine Bildschirmauflösung ab ca. 960x640 Pixel. Die Betriebssystemversionen entsprechen nicht dem aktuellen Stand, was zu einer mittleren Priorität (B) führt.
- Für ältere Geräte mit einer geringen CPU-Leistung, wenig RAM, kleinem Screen, geringer Auflösung und meist veralteter Software gilt die niedrigste Priorität (C).

Analyse der App für Plattformunterschiede

Bevor ein Tool evaluiert und ausgewählt werden kann, muss eine Analyse der zu testenden App(s) durchgeführt werden. Im ersten Schritt ist es wichtig, die Unterschiede im Aufbau der App zu kennen. Es ist in den meisten Fällen so, dass sich einzelne Schritte in den Abläufen zwischen iOS und Android oft unterscheiden. Ein Grund hierfür ist die unterschiedliche Handhabung der Betriebssysteme.

Hat man die Unterschiede im Aufbau der App herausgearbeitet, ist es weiterhin wichtig, die verschiedenen Möglichkeiten der Darstellung zu beachten. So ist bei iOS das Menü immer sichtbar oder der Abbrechen-Button ist in das Suchfeld eingebettet. Es gibt auch Unterschiede, was die einzelnen Buttons (Edit, Add, Save oder Expand) oder die Hintergrundfarbe von Listen betrifft. Im Anschluss an diese Analyse beginnt eine erste Vorauswahl der Testautomatisierungstools und eine weitere Entscheidung muss getroffen werden.

Image Based vs. Object Based

Der entscheidende Faktor bei der Auswahl von mobilen Testtools ist die Identifikationstechnik von Objekten. Um eine effiziente Automatisierung zu erreichen, ist es erforderlich, die richtige Methode zur Objekterkennung festzulegen. Die optische Erkennung verwendet im Wesentlichen eine Engine, die Bilder in maschinellen Code umwandelt, diese vergleicht und somit Darstellungsfehler identifiziert (WYSIWYG-Prinzip). Hier ist das offensichtlichste Problem die Darstellungsunterschiede auf zwei Geräten (z. B. iPhone 6 und Samsung Galaxy S6). Während die Darstellung auf einem Gerät völlig in Ordnung ist, kann diese auf dem anderen Gerät ganz anders aussehen (siehe Abbildung 4).

Das gleiche Problem tritt bei der Portrait- und Landscape-Darstellung auf. Während die Optical Character Recognition (OCR) in diesem Fall nicht funktioniert, wird die native Objekterkennung hiermit kein Problem haben. Diese Methode ist gegenüber der objektbasierten Methode deutlich langsamer und je nach Darstellung komplexer in der Verarbeitung. Die Objekt-Level-Analyse extrahiert die Objekte und deren Eigenschaften. Dies ist eine exakte und schnellere Methode, um Schaltflächen, Listen und andere Objekte, die von der Anwendung verwendet werden, zu erkennen. Der Nachteil besteht darin, dass mit diesem Ansatz keine Darstellungsfehler aufgedeckt werden. Um sich für den besten Weg zu entscheiden, wird hier noch einmal deutlich, wie wichtig eine vorangegangene Analyse der App ist.

Evaluation von Tools

Nachdem die App analysiert und die für das Projekt optimale Identifikationstechnik identifiziert wurde, gilt es nun, das richtige Tool zu finden. Es gibt verschiedene Kriterien, die bei der Auswahl von geeigneten Test-Tools berücksichtigt werden müssen (siehe Tabelle 2). In der Praxis hat es sich bewährt, mit einem Proof of Concept (PoC) für die kritischen Plattformen zu beginnen, bevor man sich (im Vorfeld) auf ein Tool festlegt. Hierbei sollte man beachten, dass möglicherweise ein Tool nicht ausreicht, um die wichtigen Ziele einer Testautomatisierung für mobile Apps zu erfüllen, daher müssen gegebenenfalls mehrere Tools für die Evaluierung herangezogen werden.

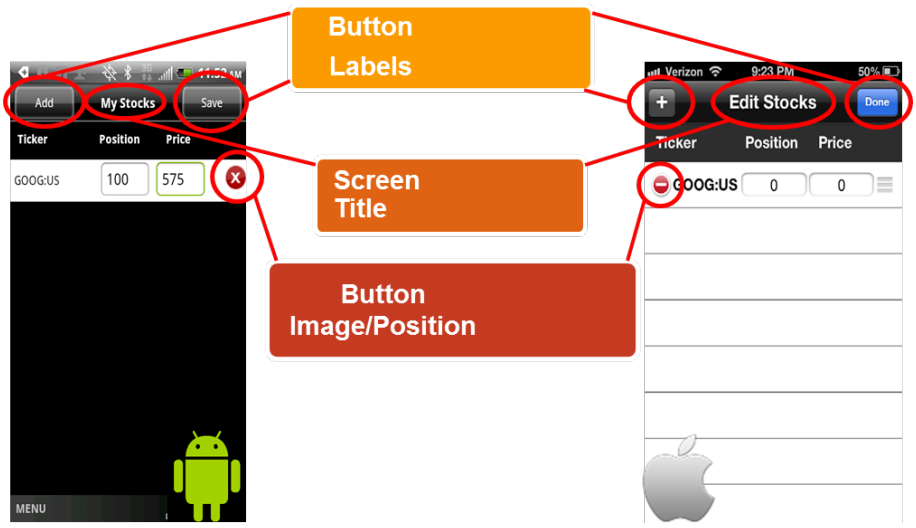

Abb. 4: Unterschiede einer App

Fazit

Der Automatisierungsansatz für mobile Apps unterscheidet sich deutlich von den traditionellen Anwendungen. Mit einer ausgeglichenen Automatisierungsstrategie und einem individuellen Konzept gelingt es, den Automatisierungsgrad und die Risikoabdeckung zu erhöhen. ■

Toolanforderungen
Ist ein Jailbreak oder Rooting der Devices erforderlich?
Ist die Objekterkennung visuelle oder object based?
Wie schnell werden neue Betriebssysteme (OS) unterstützt?
Ist es möglich, einen Testfall über verschiedene Plattformen auszuführen?
Integration mit bestehenden Integrated Development Environments (IDE)?
Wird die mobile Entwicklungsplattform unterstützt?
Kann das Tool mit Geräten remote interagieren?
Ist das Tool mit dem Testmanagementwerkzeug integrierbar?
Wie hoch sind die Lizenzkosten?
Unterstützung von physikalischen Geräten und Emulatoren?
Unterstützung von Web-Apps mit nativen Browsern?

Tabelle 2: Toolanforderungen