TESTAUTOMATISIERUNG MOBILER APPLIKATIONEN

VON **CARSTEN NEGRINI**

Testautomatisierung spielt gerade bei der Entwicklung von mobilen Apps eine wichtige Rolle. So gilt es in der mobilen Welt, Apps auf verschiedenen Plattformen (iOS, Android) und unter verschiedenen Betriebssystemversionen zu testen. Je nach Anwendungsfall sind auch noch spezielle Szenarien denkbar, sodass die Anzahl wiederholt durchzuführender Tests drastisch steigt und eine effiziente Testautomatisierung praktisch zwingend wird.

WARUM AUTOMATISIERT MAN TESTS?

Allem voran gilt es, die Kundenzufriedenheit stetig hochzuhalten und dabei nicht unnötig Zeit, Geld und Ressourcen aufzuwenden. Gerade im mobilen Bereich können die Kunden über die Appstores direkt öffentlich Rückmeldung zu ihren Erfahrungen mit den Apps geben. Zudem hat man meistens eine große Anzahl Installationen. Deshalb spielt die Kundenzufriedenheit im mobilen Bereich eine große Rolle.

Doch wie wird die entsprechende Kundenzufriedenheit erreicht? Mithilfe der Testautomatisierung lassen sich weitaus mehr Tests in gleichbleibender Zeit durchführen, was langfristig zu einer höheren Testabdeckung bei gleichzeitiger Kostenminimierung führt. Testfälle können reproduzierbar und auf verschiedenen Plattformen beziehungsweise auf verschiedenen Betriebssystemen durchgeführt werden. Die App reagiert unter Batteriebetrieb anders als unter Netzbetrieb? Oder im WLAN anders als im LTE-Netz? Solche Szenarien kann man effizient nur durch eine geschickte Testautomatisierung testen.

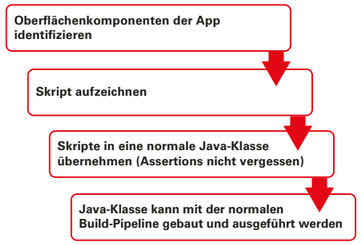
Die Tatsache, dass agiles Entwickeln und der Trend zu CI/CD häufiges Umbauen der App erfordert und damit auch einen häufigen Regressionstest, erhöht den Nutzen der Testautomatisierung in diesem Bereich.

WELCHE TOOLS KANN MAN VERWENDEN?

Es gibt viele Programme, mit denen man mobile Apps automatisiert testen kann. Bekannte Platzhirsche sind:

- Appium
- Ranorex
- UFT-Mobile

redbots nutzt häufig Appium, weshalb alle Beispiele in diesem Artikel zu Appium sind. Ranorex besitzt neben einer attraktiven und nutzerfreundlichen UI außerdem einen sehr guten Support und stammt von einem etablierten Hersteller. UFT-Mobile bietet viele Features im High-End Bereich, allerdings sind die Kosten und Einstiegshürden auch entsprechend hoch.
Appium ist ein Opensource-Programm und hat niedrige Einstiegshürden, wodurch es sich für eine Vielzahl von Anwendern eignet. Zudem ist es kompatibel mit dem Selenium Webdriver, sodass ein Investment in das Wissen einer App-Automatisierung auch übertragbar ist in die Automatisierung einer Web-Applikation und umgekehrt.
Darüber hinaus lässt sich Appium auch leicht in die üblichen Build-Pipelines integrieren und liefert daher schnell Ergebnisse. Der klassische Aufnahmeablauf wird anhand dieses Bildes etwas verdeutlicht:



Aufnahmeablauf (Quelle: redbots)

Dabei werden im ersten Schritt die Oberflächenkomponenten der App identifiziert. Im zweiten Schritt wird ein Skript aufgezeichnet, dass die Benutzerinteraktionen mit der App beinhaltet. Im dritten Schritt wird das Skript aus dem vorherigen Schritt in eine Java-Klasse übernommen und mit Assertions ergänzt, um das Sollergebnis eines jeden Schrittes abzutesten. Im vierten Schritt schließlich werden die so aufgebauten Java-Klassen in die normale Build-Pipeline integriert, sodass die Tests nahtlos und regelmäßig ausgeführt werden können.

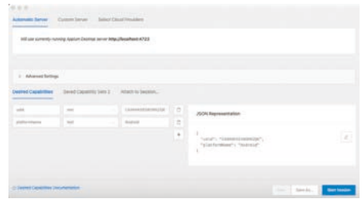
WIE SIEHT EIN TYPISCHES SET-UP AUS?

Vorweg: Der hier beschriebene Testaufbau dient dazu, eine Android-App zu automatisieren. Ähnlich wird bei der Automatisierung von iOS-Apps vorgegangen, allerdings dann anstelle Android Studio mit Xcode.
Ein typischer Aufbau einer Testautomatisierung besteht aus einer Komponente, die die Tests durchführt und steuert. Wir verwenden dazu TestNG, da diese Komponente eine sehr feingranulare Steuerung von Testreihenfolgen ermöglicht, die Gruppenbildung von Tests unterstützt und ein schönes Reporting direkt mitliefert.

Eine weitere Komponente ist die Steuerkomponente, die die Fernbedienung von auf Geräten installierten Apps unterstützt. Hierzu verwenden wir Appium, da damit sowohl eine Steuerung von Android als auch von iOS-Geräten und auch von Simulatoren möglich ist. Es können also sowohl physische Geräte, die per USB angeschlossen sind, als auch die bekannten Simulatoren gesteuert werden.
Die entsprechenden Elemente der Benutzeroberfläche werden mittels Appium-Inspektors aufgezeichnet und können dann programmatisch angesprochen werden.
Als Voraussetzung für ein lauffähiges Appium ist noch das Android SDK zu installieren, da Appium die Android Plattform-Tools als lokale Installation voraussetzt.

VERBINDUNGSAUFBAU APPIUM-TESTGERÄT

Um Appium mit dem Testgerät zu verbinden, sind innerhalb Appium die Capabilities (Verbindungen zu unterschiedlichen Geräten) entsprechend zu setzen. Um die notwendigen Informationen zu bekommen, ruft man mittels „adb devices“ die Liste der aktuell angeschlossenen Geräte auf. Die so erhaltene eindeutige Geräte-ID muss in die Appium-Capabilities als „udid“ eingetragen werden. Zusätzlich muss noch mindestens der Plattform-Name (hier: „Android“) als „platformName“ eingetragen werden.

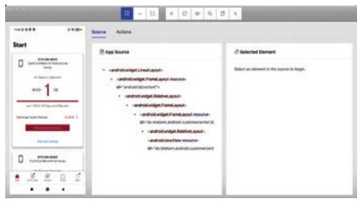


Screenshot Appium: Konfiguration der Properties, um eine Verbindung zu einem angeschlossenen Android-Testgerät zu ermöglichen.

Mit dem Klick auf „Start Session“ wird nun die Verbindung zwischen Appium und dem Testgerät etabliert, die notwendige Installation von Appium-Komponenten auf dem Endgerät wird bei der ersten Verbindung automatisch durchgeführt. Achtung: Hier sind eventuell notwendige Dialoge auf dem Endgerät zu beachten und entsprechend zu beantworten.
Das Testgerät muss sich im Entwicklermodus befinden, die hierzu notwendigen Schritte sind je nach Hersteller und Betriebssystemversion unterschiedlich.

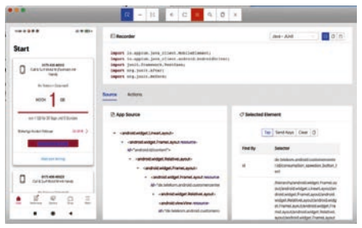
AUFZEICHNUNG DER APP-BEDIENUNG

Nun beginnt der für die Automatisierung wesentliche Schritt: Die Aufzeichnung der App-Bedienungsschritte, die der Benutzer normalerweise entsprechend macht.
Dazu werden noch mal die Capabilities so angepasst, dass die zu testende App direkt gestartet wird. Damit wird sichergestellt, dass das Testgerät zu Beginn des Tests immer im reproduzierbaren Zustand ist und die App, welche getestet werden soll, ebenfalls reproduzierbar im festgelegten Zustand ist.
Dies wird erreicht über den Eintrag „app“ sowie über den Pfad zu der entsprechenden App. Die Alternativmöglichkeit besteht darin, den Packagenamen und die Activity der App anzugeben, die gestartet werden soll.
Nun kann die Appium-Session gestartet werden und es öffnet sich die App sowohl auf dem Testgerät wie auch im Appium-Inspektor. ►



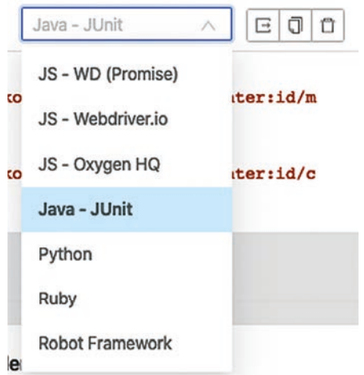
Screenshot Appium: Ansicht des Appium-Inspektors, links ein Abbild des Bildschirminhaltes eines angeschlossenen Gerätes, in der Mitte die Struktur des angezeigten Bildschirminhaltes, rechts Detailansicht des aktuell angeklickten Elements (hier nicht selektiert).

Wir verwenden hier beispielhaft eine der häufig benutzten Apps mit Versionen für iOS und Android.
Nun kann man über die Buttons oben beispielsweise einzelne Elemente der App selektieren, anklicken, über den Bildschirm mit dem Finger streichen und auch direkten Java-Code zur Testautomatisierung erzeugen.
Wir klicken also auf den „Augen"-Button, um die Aufzeichnung eines Testfalls zu starten. Der Button färbt sich nach Aktivierung rot ein. Mit dem „Select elements"-Werkzeug (in dem Dialog blau markiert) klicken wir innerhalb der App (im Dialog ganz links) auf den Button „Datenvolumen buchen".



Screenshot Appium: Hier Appium-Inspektor mit laufender Skriptaufnahme. Links die Ansicht wie auf dem angeschlossenen Gerät, rechts oben die Ansicht des aufgezeichneten Skriptes.

In der rechten unteren Ecke kann man jetzt die Detailansicht des ausgewählten Elements sehen („Selected Element").
Ein Klick auf den Button „Tap" wird jetzt sowohl in der App-Ansicht als auch auf dem angeschlossenen Testgerät die Benutzeraktion durchführen, der Button „Datenvolumen buchen" wird also angeklickt. Die Veränderung ist sowohl in der App-Ansicht wie auch auf dem Testgerät sofort sichtbar.
Der entstehende Code, um den Testfall zu reproduzieren, wird sofort oben rechts im Feld „Recorder" angezeigt. Jetzt kann noch ausgewählt werden, in welcher Sprache der Code ausgeführt werden soll.
Wir verwenden häufig Java, die Wahl der Testautomatisierungssprache ist jedoch von vielen externen Faktoren abhängig.



Screenshot Appium: Detailansicht, in welcher Skriptsprache Appium die Benutzeraktionen aufzeichnen soll. Hier ausgewählt „Java – Junit".

Der durch den Appium-Inspektor erzeugte Code kann in eine neue Java-Klasse importiert werden, diese kann dann durch weitere Features erweitert werden. So muss der Code häufig erweitert werden um dynamische Testdaten, die aus einer CSV-Datei eingelesen werden können (manchmal auch aus einer Datenbank) und dann an die App geschickt werden.

DOCH WIE HOCH IST DER AUFWAND?

Um dies zu verdeutlichen, schauen wir uns am besten einen typischen Entwicklungszyklus einer agilen Softwareentwicklung genauer an. Angenommen, diese Folge beinhaltet 3 Sprints à 3 Wochen, wobei in jeder Woche neue Features hinzugefügt werden sollen und alle 9 Wochen eine neue Version veröffentlicht werden soll. Bei manuellen Tests ist der Aufwand, um eine Regression für diese Tests durchzuführen entsprechend hoch. Automatisiert man die Tests hingegen, so ist zwar der Aufwand des eigentlichen Tests höher, dafür sinkt jedoch der Einsatz von Ressourcen für die Regressionstests signifikant, sodass der Gesamtaufwand ebenfalls stark sinkt.
Man kann also sagen, dass sich eine geschickte Testautomatisierung vor allem dann lohnt, wenn ein hoher Regressionsanteil besteht. Dieser ist natürlich wichtig, um Fehler auszumerzen und die Kundenzufriedenheit hochzuhalten.

WELCHE APPS KÖNNEN GETESTET WERDEN?

Es können alle mobilen, nativen oder hybriden Apps getestet werden, die auf iOS oder Android basieren. Dies setzt jedoch voraus, dass eine Webdriver-Implementierung vorliegt. Auf der anderen Seite ist es jedoch schwer, hardwarenahe-Tests wie etwa Bluetooth- oder eID-Tests durchzuführen, da solche Tests direkten Zugriff auf die Hardware des Testgerätes benötigen.

TIPPS ZUM ENDE

Die wichtigste Erkenntnis, die wir hier nennen können, klingt so simpel wie prägnant: Ohne Kommunikation der Entwickler und Tester wird auch die beste Automatisierung nicht die gewünschten Ergebnisse liefern. Die Rolle der Tester kann durch automatisierte Tests nicht ersetzt werden, vielmehr schafft aber eine gute Zusammenarbeit zwischen Test und Entwicklung die Grundlage erfolgreicher Testautomatisierung und damit zufriedener Kunden. ■

CARSTEN NEGRINI ist Gründer und Geschäftsführer der redbots GmbH, eines auf Java-Softwareentwicklung und Testautomatisierung spezialisierten Softwarehauses mit Sitz in Bonn und Karlsruhe. Dank zahlreicher erfolgreich umgesetzter Projekte u. a. bei DAX-Unternehmen verfügt er über eine langjährige und fundierte Erfahrung in den Bereichen Testautomatisierung, Teststrategien und Architekturfestlegungen mit Auswirkungen auf die automatisierte Testbarkeit von Java-Softwarelösungen.

„WENN DIE KI ES SICH ANDERS ÜBERLEGT"

TESTDATENMANAGEMENT IN MODERNEN ZEITEN

VON **ANNE KRAMER**

Seit Langem schon unterstützen Mustererkennungsverfahren Mediziner bei der Diagnose. Diese Verfahren basieren auf fest programmierten Algorithmen mit vorab definierten Regeln, was zwar zu guten Ergebnissen führt, in der Regel jedoch langwierig zu programmieren ist.
Nun gibt es Deep Learning-Verfahren, bei denen das System zunächst mit Trainingsdaten angelernt wird. Wir sagen der künstlichen Intelligenz (KI), welches Bild Auffälligkeiten zeigt und welches nicht. Im Produktivbetrieb lassen wir KI dann die antrainierten Auswertungsverfahren auf neue Untersuchungsergebnisse anwenden. Diese Vorgehensweise besticht durch ihre Effizienz, stellt allerdings die Qualitätssicherung vor neue Herausforderungen.

TESTDATEN FÜR KI SIND RAR

Zu den guten Praktiken im Machine Learning gehört, Trainingsdaten und Validierungsdaten sauber zu trennen. Während die Trainingsdaten dazu verwendet werden, den Algorithmus anzulernen, dienen die Validierungsdaten für den Test. Beide Datenpools müssen streng getrennt werden, denn sonst können wir das Testergebnis blind vorhersagen. Die Trainingsdaten werden immer „passed" ergeben, denn genau das hat die KI ja gelernt.
Wenn Testdaten nicht oder nur schwer künstlich erzeugt werden können, muss auf reale Daten zurückgegriffen werden. Jeder, der schon einmal Testdaten für diagnostische Software zusammenstellen musste, weiß um die damit verbundenen Schwierigkeiten. Zum einen müssen die Daten anonymisiert werden. Zum anderen sollen sie hinsichtlich der Krankheitsbilder, Patienteneigenschaften (Alter, Geschlecht, Gewicht ...) repräsentativ sein. Bei einer KI kommt noch erschwerend hinzu, dass es zum „Bias"-Effekt kommen kann.
Es ist daher unter Umständen schon schwierig, eine ausreichende Anzahl an Trainingsdaten zu bekommen. Umso glücklicher sind wir Tester, wenn schließlich sogar Validierungsdaten vorliegen.

ÄNDERUNGEN IM ALGORITHMUS DER KI

Und dann kommt es: Das Update des Algorithmus. Die Magie des Deep Learnings wird erneut angestoßen und plötzlich kommen völlig neue Ergebnisse heraus. In einem unserer Projekte hatte sich die Sensitivität des Algorithmus geändert. Krankes Gewebe wurde erst ab einem höheren Schwellwert erkannt. Systemtestfälle, die im letzten Testlauf noch vollkommen in Ordnung waren, schlugen im Regressionstest fehl. Wir mussten warten, bis neue Validierungsdaten zur Verfügung standen. Dabei ging es bei unseren Tests gar nicht um den Algorithmus an sich, sondern um die Funktionen "drumherum", also die Anzeige, das Abspeichern etc. ... Daher war es auch nicht offensichtlich, dass unsere Tests von der Änderung betroffen sein würden.

Aus Problemen soll man ja bekanntlich lernen. Folgende Ideen könnten sich als hilfreich erweisen:

- Stellen Sie einen Verantwortlichen für die Verwaltung der Testdaten ab (oder ein), der sich hauptsächlich um dieses Thema kümmert und nicht – wie der Product Owner - noch jede Menge andere Aufgaben erfüllen muss. Damit vermeiden Sie einen kritischen Flaschenhals im Projekt.
- Verankern Sie im Änderungsmanagement neben der Frage, ob die Änderungen Einfluss auf die Testfälle haben, ganz explizit auch die Frage, ob Testdaten betroffen sein könnten. Denken Sie dabei auch an die Systemtests.
- Verankern Sie die Bereitstellung der Testdaten in der „Definition of Done" des Algorithmus.

ALTERUNGSEFFEKTE

Auch ohne KI ist das Änderungsmanagement von Testdaten eine Herausforderung. Oftmals nimmt die Qualität der Testdaten über die Zeit hinweg ab. Der erste Datenpool ist noch sehr repräsentativ. Dann wird ein neuer Parameter eingeführt. Ist dieser optional, bleiben die bestehenden Testdaten weiterhin gültig. Es werden nur ein paar Datensätze geändert, um den optionalen Parameter zu prüfen. Das führt jedoch dazu, dass bei 99 % der Testdaten besagter Wert nie gesetzt ist. Ist der Parameter zwingend erforderlich, müssen alle Testdaten angepasst werden. Erneut ist der Weg des geringsten Widerstandes verlockend. Man setzt in einigen Testfällen die verschiedenen Äquivalenzklassen für den neuen Parameter ein und füllt die verbleibenden Datensätze mit einem gültigen Wert auf. Auch hier haben wir bei 99 % der Testdaten den gleichen Wert. Wir setzen an dieser Stelle auf Datenmodellierung. Die Kombinatorik der Äquivalenzklassen