Logmanagement mit Graylog S. 26
Web-Apps mit Angular 2 S. 84
JavaFX auf dem Raspberry Pi S. 106

JAVA Mag
Java magazin™
Java | Architektur | Software-Innovation

AUTOMATISIERTES TESTEN

Lassen Sie die Software für sich arbeiten

www.javamagazin.de
Ausgabe 11.2016

Das richtige Testwerkzeug für JavaFX

Die Qual der Wahl

Es gibt verschiedene Softwarelösungen, die mit unterschiedlichen Ansätzen die Automatisierung von JavaFX-Oberflächen in unterschiedlicher Qualität unterstützen. Wir haben die vier kommerziellen Werkzeuge expecco, QF-Test, Squish, TestComplete und das Open-Source-Projekt TestFX auf ihre Tauglichkeit für Entwicklungsprojekte mit JavaFX untersucht und bewertet.

von Kay Grebenstein

Testautomatisierungswerkzeuge sorgen für eine schnelle und kontinuierliche Rückmeldung über den Stand der Qualität der zu testenden Software. Aber bei ihrem Einsatz müssen verschiedene Punkte beachten werden. Es gibt viele Werkzeuge auf dem Markt, die unterschiedliche Ansätze wählen, wie sie sich in den Entwicklungs- und Testprozess integrieren oder welche Technologien sie unterstützen.

Es gibt entwicklungsnahe Werkzeuge, die sich direkt in die Entwicklungswerkzeuge integrieren lassen. Mit diesen Testwerkzeugen werden keine Unit-Tests umgesetzt, da durch die Ansteuerung der grafischen Oberfläche bereits eine zusätzliche Schnittstelle eingebunden wurde. Trotzdem erfolgt ihre Bedienung oder die Beschreibung der Testskripte über das Schreiben von Code. Demgegenüber stehen Testautomatisierungswerkzeuge für den Systemtest. Sie wenden sich in ihrer Bedienung an Anwender, die nicht unbedingt Erfahrung im Programmieren mitbringen müssen. Die Testskripte werden dort „Keyword-driven“ oder „Model-based“ abgebildet. Leider verwenden die Hersteller der Werkzeuge die Bezeichnungen nicht eindeutig und standardisiert. Bei „Keyword-driven“ erfolgt die Interaktion mit den zu testenden Oberflächen über Schlüsselworte, die die Controls ansteuern, z. B. „Klick ButtonXY“ oder „Überprüfe Texteigenschaft von LabelZ“. Bei „Model-based“ werden die Testskripte in einer abstrakten Form abgebildet, z. B. in einem Flussdiagramm oder per Behavior-driven Development (BDD).

Bei der technischen Umsetzung der Ansteuerung und Interaktion mit den zu testenden Oberflächen gibt es kaum Unterschiede. Bei älteren Automatisierungstools wurde die Aufnahme der Informationen dadurch erreicht, dass die Positionsdaten des GUI-Elements, wie z. B. eines Buttons, gespeichert werden und zur Ausführungszeit ein entsprechendes Event abgesetzt wird. Anschließend erstellt die Software einen Screenshot und vergleicht ihn mit einem zuvor erstellten, um die Testergebnisse zu verifizieren. Dieses Vorgehen ist für Probleme bei Änderungen der Oberfläche anfällig. Bereits beim Ändern der Farbe der Controls würden alle Testfälle fehlschlagen. Zusätzlich setzt die Erstellung der Testfälle das Vorhandensein einer fertigen Version der zu testenden Software voraus.

Moderne Tools hingegen verfolgen einen anderen Weg. Sie verbinden sich über eine eigene Engine mit der zu testenden Applikation. Dadurch sind sie in der Lage, alle Control-Elemente der Oberfläche zu erfassen, deren Eigenschaften auszulesen und sie auch fernzusteuern. Die zugehörigen Daten der Controls werden als ein Modell der Applikation in so genannten GUI-Maps abgelegt. Die Testskripte lassen sich vorab erstellen und die anzusteuernden Controls nachträglich zuordnen (**Abb. 1**).

Dazu stellen sie Werkzeuge wie Capture and Replay oder interaktive Objekterkennung bereit. Mit der interaktiven Objekterkennung lassen sich die Controls in der GUI-Map analysieren, die Eigenschaften auslesen und den Testskripten zuordnen. Mit Capture and Replay besteht die Möglichkeit, Testskripte aufzuzeichnen, indem man die zu testenden Controls nacheinander anklickt. Somit lassen sich ohne viel Aufwand automatisierte Testfälle erstellen. Leider ist das in der Praxis nicht immer so leicht. Meist muss man an den aufgezeichneten Testskripten selbst nochmal Hand anlegen, da Controls nicht korrekt erkannt oder falsche Events für die Interaktion genutzt wurden.

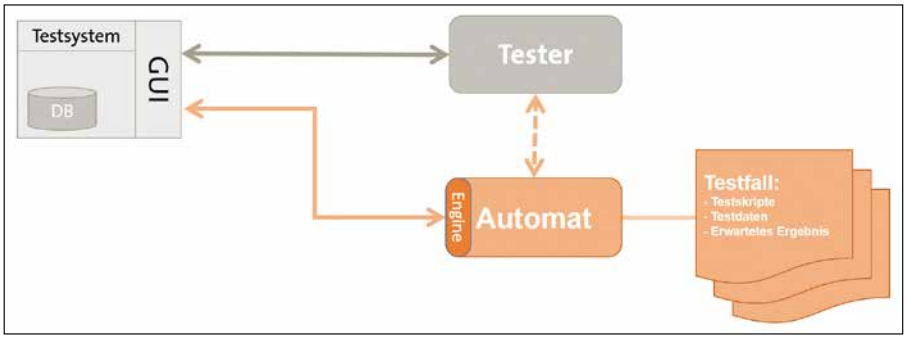

Abb. 1: Einsatz von Testautomatisierungswerkzeugen

TestFX: entwicklungsnah testen

TestFX ist ein Open-Source-Projekt, mit dem man auf Basis von JUnit Testfälle für JavaFX schreiben kann. TestFX steht als Framework auf GitHub zur Verfügung und lässt sich einfach in das bestehende Entwicklungsprojekt integrieren. Die TestFX-Testfälle (**Abb. 2**) werden analog zu den JUnit-Tests geschrieben und im Code abgelegt. Dadurch lassen sich die Tests während des Build-Laufs in der Integrationstestphase ausführen.

Das ist auch die Stärke des Frameworks: einfache Integration in die Entwicklungsumgebung und den Entwicklungsprozess. Die Erkennung und Ansteuerung der JavaFX-Controls ist sehr gut. Treten trotzdem Probleme auf, lässt sich der vorhandene Quelltext einfach auf die persönlichen Gegebenheiten anpassen. Außerdem unterliegt der Testskriptschreiber keinen Einschränkungen im Testablauf, da er vollen Zugriff auf die Möglichkeiten von Java 8 hat.

Diese Entwicklernähe bringt aber auch Einschränkungen mit sich. Die Testfälle lassen sich nur mit Entwicklerkenntnissen erstellen und anpassen. Außerdem benötigt man Wissen über die verwendeten Controls der zu testenden Software und muss diese im Code herauslesen. TestFX stellt dazu leider keine Werkzeuge wie Capture and Replay oder interaktive Objekterkennung bereit.

Aktuell liegt JavaFX in der Version 4.0.1-alpha vor. Die finale Version ist für Q1 2016 geplant [1]. Die Dokumentation in [2] ist an einigen Stellen noch lückenhaft. Das betrifft zum Beispiel die aktuellen Java Docs.

TestFX ist eine schnelle und kostengünstige Lösung für entwicklungsnahe Tests. Um effektiv arbeiten zu können, benötigen die Tester aber Entwickler-Know-how und Projektwissen. Außerdem sollte man im Hinterkopf behalten, dass es eine Open-Source-Lösung ist. Die Zukunftsfähigkeit und die Qualität der Lösung und der Support hängen von der Unterstützung der Community ab.

```
@Test
public void new_Entry() {
    // when:
    FxAssert.verifyThat("#New", NodeMatchers.isNotNull());
    final Button button_new = lookup("#New").queryFirst();
    clickOn(button_new);

    // Vorname eintragen
    final TextField test_firstNameField = lookup("#firstNameField").queryFirst();
    doubleClickOn(test_firstNameField).write("Kay");

    // Nachname nachtragen
    final TextField test_lastNameField = lookup("#lastNameField").queryFirst();
    doubleClickOn(test_lastNameField).write("Grebenstein");

    // Strasse eintragen
    final TextField test_streetField = lookup("#streetField").queryFirst();
    doubleClickOn(test_streetField).eraseText(0);
    doubleClickOn(test_streetField).write("Fritz-Foerster-Platz 2");

    final Button button_Ok = lookup("#Ok").queryFirst();
    clickOn(button_Ok);

    // then:
    FxAssert.verifyThat("Kay", LabeledMatchers.hasText("Kay"));
    FxAssert.verifyThat("Grebenstein", LabeledMatchers.hasText("Grebenstein"));

}
```

Abb. 2: Testen mit TestFX

QF-Test: automatisierte Testfälle für Java

Die Software QF-Test der Firma Quality First Software folgt dem Key-driven-Ansatz. In einer einfach gehaltenen Oberfläche werden die Testskripte, die erkannten Controls und andere Testbausteine einer Baumstruktur angeordnet. Außerdem können für die ausgewählten Baumelemente die Parameter und Einstellungen geändert werden (**Abb. 3**). Es stehen zahlreiche Tutorials und ein ausführliches Handbuch in Deutsch und Englisch bereit.

QF-Test verfügt über die beste Erkennung und Ansteuerung der getesteten Werkzeuge. Zusätzlich verfügt es über eine Capture-and-Replay-Funktion, mit der man die Testskripte und Controls aufzeichnen und nachträglich ändern kann. Dabei ist es auch möglich, Prüfpunkte in der Aufnahme zu integrieren. Nach der Testdurchführung kann ein Testbericht erstellt und in HTML exportiert werden. Die Testberichte lassen sich bei Bedarf durch Screenshots und weitere Logging-Informationen anreichern.

Mit QF-Test lassen sich ohne viel Aufwand automatisierte Testfälle für Java-Oberflächen erstellen, also JavaFX, Java/Swing, SWT, Eclipse-Plug-ins und RCP-Anwendungen, Java-Applets, Java Web Start und ULC. Auch für statische und dynamische Webseiten ergibt sich kein großer Aufwand, also für HTML und Ajax-Frameworks wie Ext JS, GWT, GXT, RAP, qooxdoo, RichFaces, Vaadin, PrimeFaces, ICEfaces und ZK. Es spricht in der Bedienung auch entwicklungsfernere Anwender an, bietet aber auch für Entwickler die Möglichkeit, die existierenden Bausteine per Jython und Groovy zu erweitern.

Das Testwerkzeug bietet einen so genannten Batch Mode an, der es erlaubt, Tests unbeaufsichtigt auszuführen. Damit lässt sich QF-Test über Kommandozeilenbefehle oder Plug-ins in den Build-Prozess integrieren. Der Batch Mode wurde erweitert, die notwendigen Laufzeitlizenzen vorausgesetzt, sodass auch die Durchführung und Auswertung von Lasttests möglich sind.

Die Bewertungskriterien

Die Bewertungskriterien berücksichtigen sowohl die Sicht der Entwickler als auch die Sicht der Tester:

- Wie gut ist die Erkennung und Ansteuerung der JavaFX-Controls?
- Wie ist die Erlernbarkeit für Entwickler und Nichtentwickler, und gibt es Tutorials und Beispiele?
- Welche Möglichkeiten der Testauswertung der durchgeführten Testläufe bieten die Programme?
- Lassen sich die Werkzeuge in die bestehenden Build-Abläufe der Projekte (CI/CD) einbinden?
- Wie gut ist die Wartbarkeit, und wie lassen sich die Testskripte nachträglich anpassen?
- Ist das Werkzeug nur auf eine Technologie beschränkt oder lässt es sich auch anderweitig einsetzen (Austauschbarkeit)?
- Gibt es einen Support in deutscher Sprache?

expecco: Erzeugung von Reports in verschiedenen Formaten

Das Werkzeug expecco der Firma Exept Software AG ist ein Vertreter der so genannten Model-based Testwerkzeuge und unterstützt ein breites Angebot an Technologien im Bereich Software, aber auch Hardware. Neben Java-, .NET-Anwendungen, HTML und Android-/iOS-Anwendungen versteht es auch Protokolle und Nachrichtenkodierungen wie ASN.1, SOAP, XML-RPC, REST, SNMP, XML, Message Queues oder Swift. Es versteht außerdem Messgeräte (SCPI, IEEE 488) und Embedded-Systeme wie CAN oder LabVIEW.

Die Testfälle werden als Aktivitätsdiagramme visualisiert (**Abb. 4**) und sind somit auch für Nichtprogrammierer verständlich. Über die Notation der Aktivitätsdiagramme lassen sich in den Testfällen klei-

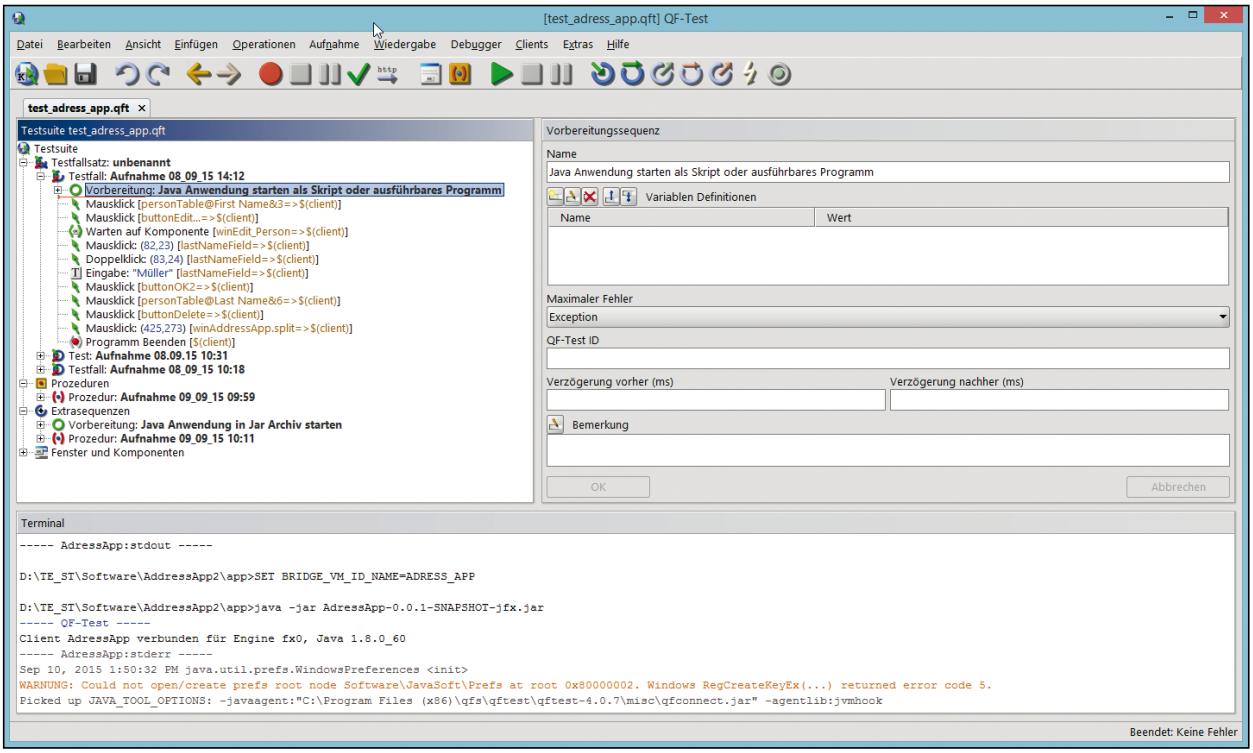

Abb. 3: QF-Test mit Testfallelementen in Baumstruktur

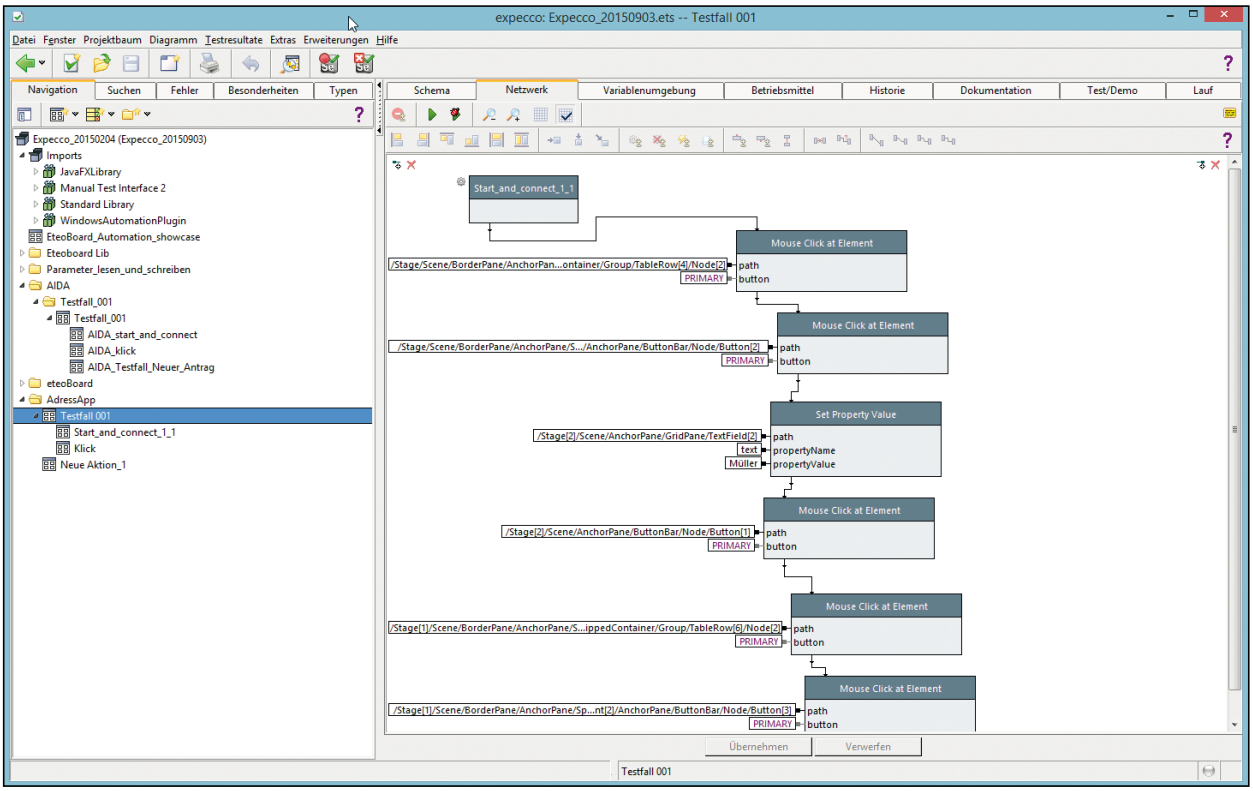

Abb. 4: expecco mit Testfall in Aktivitätsdiagramm

nere Elemente zu neuen Bausteinen zusammenfassen oder Nebenläufigkeiten abbilden.

Gegenüber den anderen Werkzeugen verfügt expecco über keine Capture-und-Replay-Funktion, mit der sich die Testskripte für JavaFX aufzeichnen lassen. Als Ersatz erkennt das Modul GUI-Browser die Controls der zu testenden Applikation und ermöglicht es, schnell und einfach aus den erkannten Controls Testbausteine zu erzeugen und diese zu Testfällen zusammenzufügen.

Bei Erkennung und Ansteuerung gibt es leider ein paar Einschränkungen. Controls wie Menübutton oder Slider lassen sich nicht über die Interaktion fernsteuern. In diesen Fällen ist die Testdurchführung über einen Umweg möglich, indem die Eigenschaften der Controls wie Slider-Position oder Auswahl des Menübuttons direkt gesetzt werden.

Auch für expecco stehen zahlreiche Tutorials und ein ausführliches Wiki in Deutsch und Englisch bereit, und es lässt sich in Jenkins/Hudson einbinden. Sehr positiv ist die Erzeugung von Reports in verschiedenen Formaten wie JUnit XML, PDF, Text und HTML. Damit lassen sich die Reports in die bestehende Testdokumentation einbinden.

Squish GUI Tester: nicht nur für JavaFX

Der Squish GUI Tester wird von froglogic entwickelt und gepflegt. Zur Beschreibung der Testskripte nutzt das

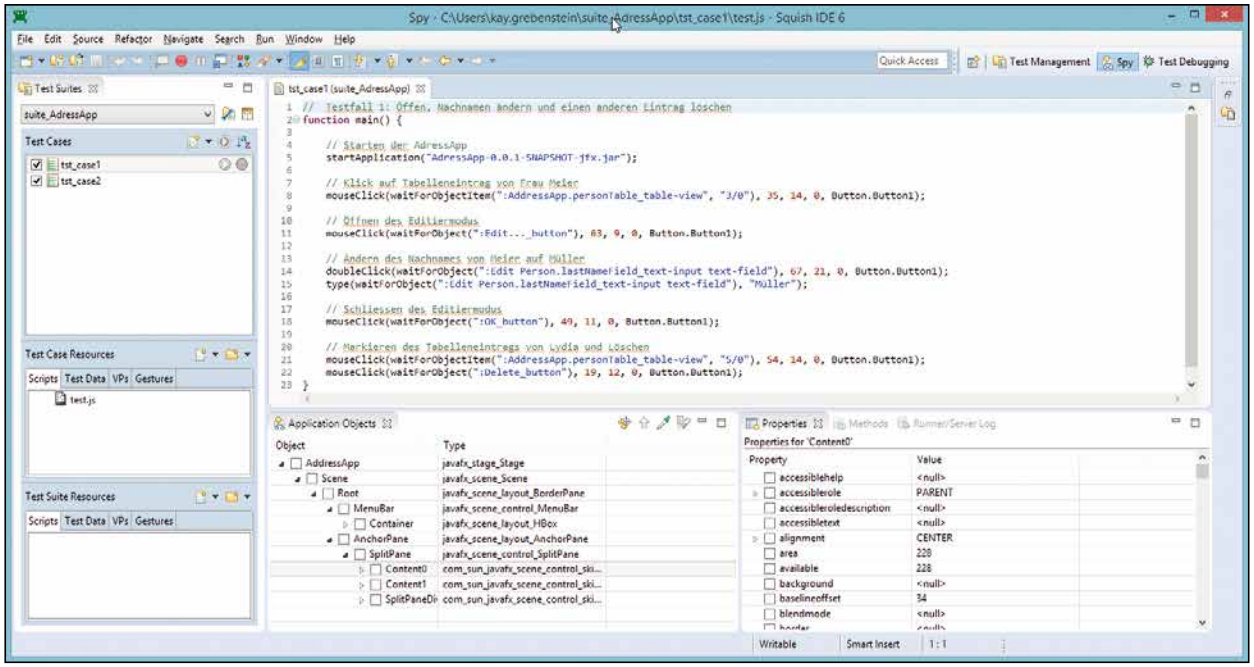

Abb. 5: Squish mit Testfall in JavaScript

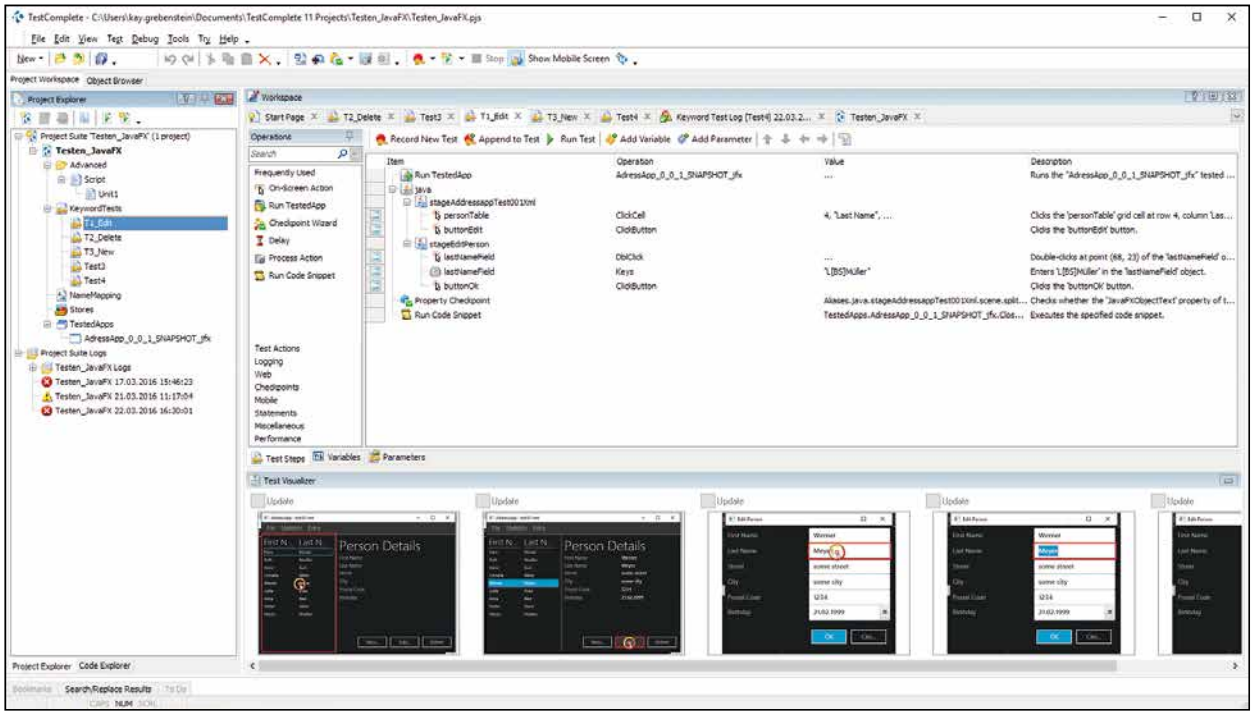

Abb. 6: TestComplete mit Testfall und aufgezeichneten Screenshots

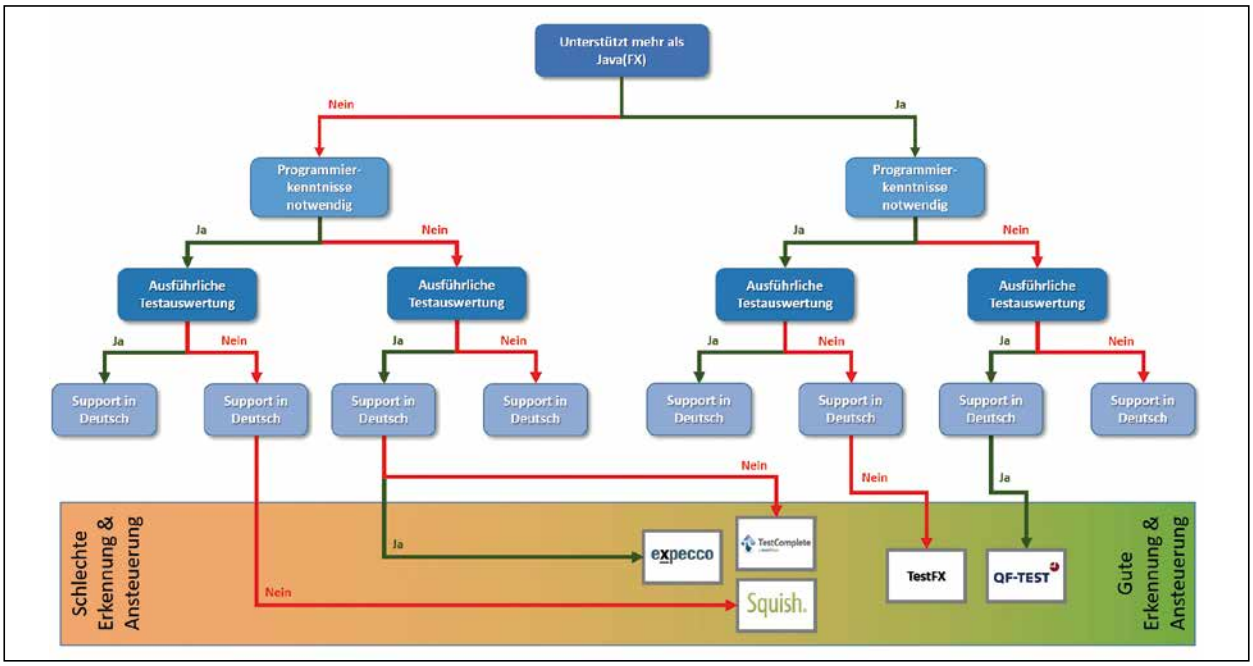

Abb. 7: Entscheidungsbaum auf Basis der Bewertungskriterien

Werkzeug Skriptsprachen. Der Anwender kann zwischen JavaScript, Perl, Python, Ruby oder Tcl wählen (**Abb. 5**). Damit fällt es in den Bereich „Keyword-driven". Ab der Version 6.0 können die Testskripte im Squish GUI Tester auch in Form von Behavior-driven-Development-(BDD-) Tests geschrieben werden. Dazu kann Gherkin als domänenspezifische Sprache genutzt werden.

Squish unterstützt neben JavaFX ein breites Spektrum an anderen Oberflächentechnologien aus den Bereichen Windows, Linux, Unix, Mac OS X, iOS und Android. Auch bei Squish gibt es ein paar Probleme bei der Ansteuerung von JavaFX Controls. So wurden in unserem Test zwar alle Elemente erkannt, aber mehrere Tab Controls nicht korrekt angesprochen. Für Squish GUI Tester steht ein Handbuch in Englisch bereit. In den Tutorials sind alle Beispiele für alle unterstützten Skriptsprachen vorhanden. Squish basiert auf Eclipse, was sich auch in der Oberfläche zeigt. Dadurch stehen neben Plug-ins für die Integration in Jenkins, Maven und Bamboo auch Plug-ins für Eclipse zur Verfügung. Die Testauswertung erfolgt direkt in der Oberfläche, und für den Export steht nur XML zur Verfügung.

TestComplete: Integration in die Entwicklungsumgebung

TestComplete wird von SmartBear Software entwickelt und gibt den Testern die Möglichkeit, automatisierte Tests für Microsoft Windows, Web, Android und iOS-Anwendungen zu erstellen (**Abb. 6**). Bei der Aufzeichnung von Tests per Capture and Replay wurden einige Controls wie Menüs nicht korrekt erkannt und mussten manuell nachgepflegt werden.

TestComplete legt hohen Wert auf eine Integration in die Entwicklungsumgebung und ermöglicht eine direkte Fehlererstellung in Axosoft OnTime und Atlassian JIRA, eine Ansteuerung von Source-Control-Systemen wie Git, Subversion und Mercurial sowie eine Integration in den Jenkins-Build-Ablauf. Auch für TestComplete gibt es umfangreiche Handbücher, Tutorials und Beispiele in englischer Sprache.

Fazit

Es gibt viele Werkzeuge für die Testautomatisierung mit JavaFX. Deren Funktionsumfang reicht von einfach bis zu sehr komfortabel. Jedes der untersuchten Werkzeuge hat seine Vor- und Nachteile sowie Besonderheiten. Darum sollte im Vorfeld geprüft werden, welche Anforderungen das Team und das Projekt an das Werkzeug stellen. In den Überlegungen sollten neben der Unterstützung der eingesetzten Technologien auch die Interessen der späteren Anwender – der Entwickler, Tester oder des Fachbereichs – oder die Frage des Supports einfließen. Als Entscheidungshilfe kann auch ein Entscheidungsbaum (**Abb. 7**) dienen, wie er auf Basis der Bewertungskriterien für diesen Artikel erstellt wurde.

Kay Grebenstein arbeitet als Tester und agiler QA-Coach für die Saxonia Systems AG, Dresden. Er hat in den letzten Jahren in Projekten unterschiedlicher fachlicher Domänen (Telekommunikation, Industrie, Versandhandel oder Energie) Qualität gesichert und Software getestet.

Links & Literatur

[1] Casall, Alexander: „TestFX – Current State and Future": http://blog.so-geht-software.de/2015/12/textfx-current-state-and-future/

[2] TestFX, GitHub: https://github.com/TestFX/TestFX

[3] GUI-Testautomatisierung für Java und Web: https://www.qfs.de/de/qftest/index.html

[4] expecco: https://www.exept.de/de/produkte/expecco.html

[5] Squish: http://www.froglogic.com/squish/gui-testing/index.php

[6] TestComplete Platform: https://smartbear.com/product/testcomplete/overview/