Keyword-driven Testing mit GUIdancer, einem kommerziellen Werkzeug für automatisiertes GUI-Testen

Automatische GUI-Tests

>> ALEXANDRA IMRIE

Denkt man an den möglichen Aufbau eines GUI-Tests, liegt zunächst die Idee nahe, zu testen, ob alle Komponenten da sind und ob diese funktionieren. Oder man stellt sich vor, der Test prüft einfach die Schriftart und Schriftgröße angezeigter Texte. Ganz so „oberflächlich" – wenn man das so ausdrücken darf – sollte GUI-Testen allerdings nicht sein. Vielmehr ist es das Ziel, die Funktionalität einer Anwendung über die Oberfläche zu testen. Wie das für eine RCP-Anwendung mit einem kommerziellen Tool funktioniert, wird in diesem Artikel erläutert.

GUI gut, alles gut? Sicher kann und muss man auch im Rahmen eines GUI-Tests überprüfen, ob beispielsweise Pflichtfelder bei Fehleingaben rot markiert werden. Das Spannende beim GUI-Testen liegt aber eher darin, für den Endanwender wichtige Bedienungen (z. B. verschiedene mögliche Workflows) in einem Black-Box-Test abzubilden. Sinn und Zweck eines funktionalen GUI-Tests ist daher, die Qualität der entwickelnden Software mit vertretbaren Kosten zu verfolgen. Ein erfolgreicher Test deutet darauf hin, dass die Anwendung sich den Anforderungen entsprechend verhält. Abweichungen (fehlgeschlagene Tests) zeigen in der Regel Fehler in der Software auf. Nach der Behebung eines Fehlers sollte der Test erneut durchgeführt werden. Dieser Regressionstest bringt zwei Vorteile: Erstens wird sichergestellt, dass der Fehler tatsächlich behoben wurde, und zweitens, dass die Beseitigung keine anderen Fehler verursacht hat. Beim Testen grafischer Oberflächen sind Regressionstests also besonders wichtig. Das Schlüsselwort dabei ist „Wartbarkeit", denn die Software erfährt in der Regel bereits während der Entwicklung erste Änderungen. Je weniger Aufwand benötigt wird, um die Regressionstests ablauffähig zu halten, desto höher ist im Endeffekt der Return of Investment (ROI) und desto besser ist die Qualität der Software. Ein zweites Ziel des automatischen GUI-Testens ist, die Zeit zwischen Entwicklung und Test zu verringern. Wird ein Fehler erst Wochen oder Monate nach dem „Einchecken" eines Codestücks gefunden, entstehen deutlich mehr Kosten durch erhöhten Aufwand, den Fehler zu beheben, als wenn dieser früh gefunden worden wäre.

Die Anforderungen an einen Test beinhalten also auf der einen Seite das zeitnahe Schreiben und Ausführen von Tests. Tests sollten möglichst während der Implementierung geschrieben und direkt ausgeführt werden können, sobald die zu testende Anwendung (AUT) verfügbar ist. Auf der anderen Seite ist gutes Design und eine wohlüberlegte Struktur unabdingbar für die Wartbarkeit der Tests. Keyword-driven Testing bietet einen möglichen Weg, diesen Anforderungen gerecht zu werden.

Stichwort „Keywords"

Keywords kann man sich als Testbausteine (Module) vorstellen, die mit einem für den Tester griffigen und verständlichen Schlüsselwort versehen werden. GUI-Tests bestehen aus vielen, sich wiederholenden Sequenzen von Aktionen (Text eingeben/prüfen, Schaltfläche klicken, Reiter auswählen usw.). Für solche Aktionen können vorgefertigte Module erstellt werden. Ein Tester muss nur noch die notwendigen Keywords zusammenstellen und gegebenenfalls mit Daten versehen, um seine Testabläufe zu erstellen. Aus einfachen Keywords können wiederum komplexere Keywords zusammengesetzt und wiederverwendet werden (z. B. „Öffne Suchdialog aus dem Menü", „Fülle Suchmaske aus", „Schließe Suchdialog mit O.K.", Abb. 1). Damit bleibt die Arbeit erspart, eine Sequenz von Aktionen mehr als einmal definieren zu müssen.

Keyword-driven Tests haben also den Vorteil, dass sie meist gut strukturiert sind. Wiederverwendbare Module sorgen für ein leichtes Pflegen der Tests, denn Veränderungen an einer zentralen Stelle halten den gesamten Test aktuell. Das Erstellen der Keywords wird häufig von Automation Experts durchgeführt. Keywords werden anhand einer Programmiersprache für eine Anwendung definiert und anschließend an das Testteam zur Verwendung weitergegeben. Eine zeitnahe Testerstellung ist dann möglich, wenn die Implementierung der Keywords rechtzeitig zur Verfügung gestellt wird. Im Folgenden beschreibt dieser Artikel den Ansatz des kommerziellen

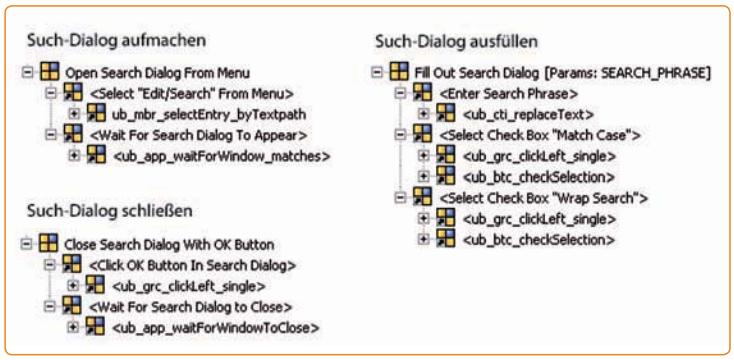

Abb. 1: Beispiel-Keywords für einen Suchdialog

Tools *GUIdancer* zur Durchführung von Keyword-driven Testing.

GUIdancer

GUIdancer [1] ist ein Werkzeug zum automatischen Testen von Java-Oberflächen. Das Tool soll es ermöglichen, Tests frühzeitig und ohne Programmierkenntnisse durchzuführen und so den Testprozess synchron mit der Entwicklung zu halten. In der ersten Version wurden nur Swing-Oberflächen unterstützt. Die Erweiterung der Testmöglichkeiten auf SWT/RCP-Oberflächen und HTML war ein wichtiger Meilenstein für das auf Eclipse basierende Tool: Ab Dezember 2007 (Version 2.0) konnte sich GUIdancer endlich selbst testen. Seit Sommer 2008 ist Version 2.2 verfügbar.

Die Testerstellung mit GUIdancer basiert auf zwei elementaren Prinzipien. Erstens kann man aus den Anforderungen, und damit vor der Verfügbarkeit der AUT, Tests erstellen. Somit kann der automatisierte Test quasi zeitgleich mit der Entwicklung aufgebaut werden. Das zweite Prinzip ist die Wiederverwendung von Testfällen, die darauf abzielt, den Wartungsaufwand möglichst gering zu halten. Zu diesem Zweck setzt GUIdancer Keywords ein. Im Unterschied zu anderen Keyword-driven-Ansätzen erfolgt die Test- und Keyword-Erstellung allerdings ohne Programmierung – der Aufwand, Code zu schreiben und zu pflegen, entfällt.

Der Keyword-Ansatz erfordert es, sich schon am Anfang Gedanken über eine gute Struktur und ein gutes Testdesign zu machen. Analog zu den Prinzipien vom Softwaredesign gehören solche Gedanken früher oder später ohnehin zu jedem Testprozess. Stellt man erst im Nachhinein fest, dass ein Test eine hohe Modularität benötigt, bietet GUIdancer dem Tester eine Refactor-Funktion.

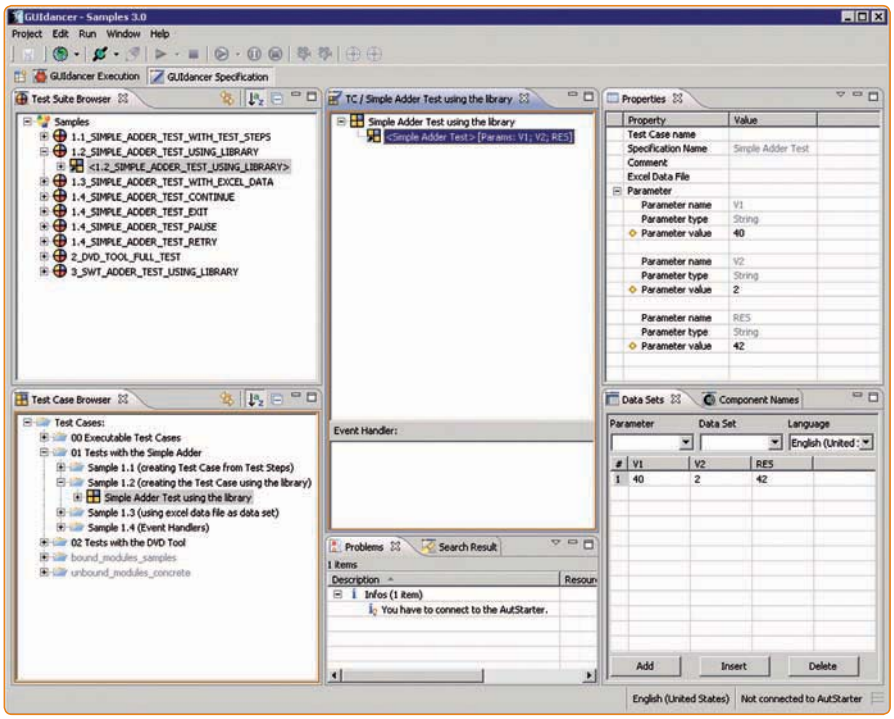

Abb. 2: Die GUIdancer-Oberfläche

Erste Schritte

Wie genau die Erstellung von Keywords und Testläufen funktioniert, kann jeder mit einer Demolizenz herausfinden. Nach dem Herunterladen und Installieren der Software muss zunächst eine Demolizenz im GUIdancer Shop angefordert werden. Die dafür notwendige *ServerID* erhält man im Lizenzadministrator, der automatisch nach der Installation aufgerufen wird. Die Demolizenz wird dann im Lizenzadministrator aktiviert und GUIdancer kann gestartet werden. Für Eclipse-Benutzer sieht die Oberfläche vertraut aus: Perspektiven, Editoren und Views, einschließlich Properties View und Problems View sind vorhanden (Abb. 2). Preferences werden im Workspace gespeichert. Ein großer Unterschied zu Eclipse besteht darin, dass Projekte in einer Datenbank (entweder in der mitgelieferten HSQL-Datenbank oder einer separaten Oracle-Datenbank) gespeichert werden. Durch die Verwendung einer Datenbank können mehrere Benutzer gleichzeitig an einem Projekt arbeiten. Um den Einstieg in das Tool zu erleichtern, zeigen Cheat Sheets (schrittweise Anleitungen), wie man einen ersten Test zum Laufen bringt. Folgende Schritte werden erklärt:

1. Das Anlegen eines neuen Projekts (Namen und Sprachen definieren).
2. Die Konfiguration einer AUT (sofern diese schon bekannt bzw. vorhanden ist). In einer AUT-Konfiguration wird festgelegt, wo und wie die AUT zu starten ist – wahlweise über ein JAR oder ein Executable (*.exe*). Für RCP-Anwendungen gibt es weitere Details bei der Konfiguration zu beachten (Kasten: „RCP-GUI-Tests: Was muss man wissen?“).
3. Das Erstellen und Wiederverwenden von Testfällen (Keywords).
4. Das Anlegen von Test-Suiten.
5. Die Durchführung des Object Mappings (spätestens hier muss die AUT vorliegen).
6. Die Ausführung des Tests.

Bei der täglichen Arbeit mit GUIdancer hat man es am häufigsten mit dem Design und der Erstellung von Testfällen zu

tun. Diese Arbeit wird in den nächsten Abschnitten näher erläutert.

Spezifizieren statt Programmieren

Drei Dinge sind stets für einen Test vonnöten: Komponente, Aktion und Parameter (Daten). Aus diesen drei Informationen besteht jeder Testschritt (als kleinstes Element) in GUIdancer, z. B. auch das einmalige Klicken (Aktion) auf den *O.K.*-Button (Komponente) mit der linken Maustaste (Parameter). Zwar kann man solche Testschritte mit einem einfachen Dialog im GUIdancer-Client anlegen, aber eine andere Arbeitsweise wird bevorzugt. Um das Neuerstellen gleicher oder ähnlicher Testschritte zu vermeiden, stehen in jedem GUIdancer- Projekt wiederverwendbare, schon parametrisierte Keywords in einer Bibliothek zur Verfügung. Dort werden alle Aktionen auf allen unterstützten Komponenten (auch RCP-spezifische Komponenten) abgebildet. Das Erstellen von Keywords erfolgt per Drag-and-Drop.

Testerstellung per Drag-and-Drop

Um beispielsweise einen Testfall zu erstellen, der einen Benutzernamen eingibt,

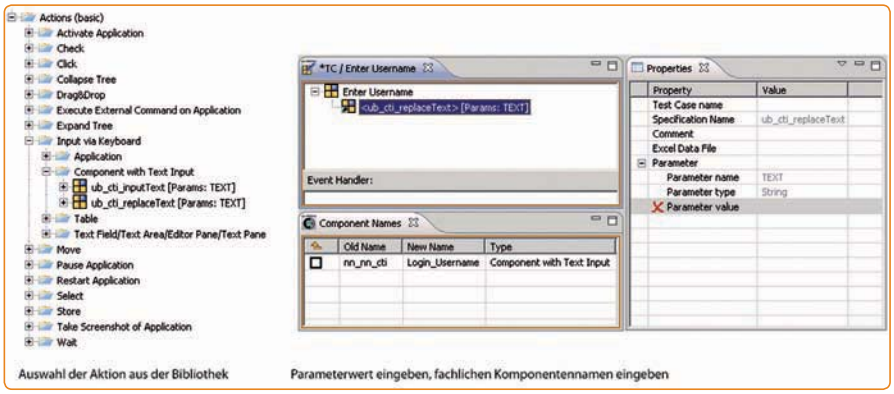

Abb. 3: Testfall: Enter Username

legt man zunächst einen leeren Testfall an, z. B. unter dem Namen *Enter Username*. Aus der Bibliothek sucht man nach einem treffenden Baustein, etwa aus der Kategorie *Input via Keyboard/Component with Text Input*. Das Keyword *replace text* ist die passende Wahl. Per Drag-and-Drop wird dieser Testfall zum leeren *Enter-Username*-Testfall hinzugefügt (Abb. 3). Dieser Baustein lässt sich so für jede mögliche Komponente verwenden, egal, welchen konkreten Typ die GUI-Komponente besitzt (editierbare Combo-Box, JTextfield usw.). Im *Component Names View* muss man einen sprechenden fachlichen Namen für diese Komponente vergeben (was später für das Object Mapping wichtig ist). Im Properties View sieht man, welche Parameter diese Aktion benötigt. An dieser Stelle kann man entweder konkrete Daten (z. B. *admin*) angeben oder eine Referenz eintragen *(=USERNAME)* und dadurch den Testfall parametrisieren. Damit steigt die Wiederverwendbarkeit eines Testfalls erheblich. Referenzierte Daten lassen sich auch über Excel importieren, damit ein Test einen datengetriebenen Ansatz verwenden kann.

Abb. 4: Object-Mapping-Editor

Nach dem Speichern kann man diesen *Enter-Username*-Testfall beliebig in anderen Testfällen wiederverwenden und dabei den verwendeten Benutzernamen jedes Mal neu übergeben. Muss etwas an diesem Testfall verändert werden, genügt es, nur eine einzige Stelle anzupassen, um alle Wiederverwendungsstellen aktuell zu halten. Mit diesem hierarchischen Verfahren können Tests exponentiell wachsen. Ein *Do-Login*-Testfall besteht beispielsweise aus folgenden Keywords:

- *Enter Username* (enthält einen Replace Text)
- *Enter Password* (enthält einen Replace Text)
- *Click OK In Login Dialog* (enthält einen click left once)

RCP-GUI-Tests: Was muss man wissen?

Der RCP-Accessor
Um das Testen von RCP-Anwendungen zu ermöglichen, muss man den so genannten RCP-Accessor aus der GUIdancer-Installation ins Verzeichnis *Plugins* oder *Dropins* der AUT extrahieren. Dieses Plug-in wird mit der Anwendung gestartet.

Keyboard-Layout
Gerade bei SWT- und RCP-Anwendungen ist es wichtig, in der AUT-Konfiguration das Tastaturlayout einzutragen, das vom System verwendet wird, auf dem die AUT läuft. Das in den Systemeigenschaften stehende Tastaturlayout wird eingegeben. Die tatsächlich angeschlossene Tastatur hat keine Auswirkung.

Workspace Chooser
Wenn man den Test von der Kommandozeile aus startet, empfiehlt es sich, über den Befehl *-data* den gewünschten Workspace einzugeben, damit der Workspace Chooser nicht erscheint.

- *Wait For Window To Close* (Login-Dialog) bzw. *Wait For Window* (Main Screen)

Jeder im *Do Login* verwendete Testfall kann auch an jeder anderen Stelle im Test benutzt werden, bei Bedarf auch mit veränderten Daten.

Ab zum Test

Hat man eine Anzahl an fertigen Testfällen zum Ausführen erstellt, werden diese zu einer Test-Suite zusammengefügt. Spätestens jetzt müssen fehlende Daten eingetragen und das Object Mapping durchgeführt werden, denn bevor ein Test ausgeführt werden kann, muss er vollständig und mit der Anwendung verbunden sein. Zum Mapping wird die konfigurierte AUT aus dem GUIdancer heraus gestartet. Der Mapping-Modus wird über die GUIdancer Toolbar eingeschaltet. Das „Einsammeln" der benötigten Oberflächenkomponenten erfolgt per Tastenkombination. Im Object-Mapping-Editor sieht man sowohl die eingesammelten Objekte (*technical names*) als auch die vom Tester im Component Names View vergebenen fachlichen Namen aus der Test-Suite (*component names*) (Abb. 4). Mittels Drag-and-Drop werden sie untereinander verknüpft. Der Test erhält so seine Bindung an die Anwendung. Nach dem Speichern des Object-Mapping-Editors kann der Test direkt ausgeführt werden.

Gedanken über den Testprozess

Zur Integration im automatischen Build- und Testprozess kann die AUT auch vom Command Line Client oder über eine Ant <*exec*> Task gestartet werden. Die GUIdancer-Serverkomponente (der so genannte AUTStarter) kann auf Windows- oder Linux-Rechnern im gesamten Netzwerk installiert werden, um verteiltes Testen zu unterstützen. HTML- und XML-Berichte sowie Screenshots von Fehlersituationen können abgelegt werden, um den Testfortschritt zu verfolgen und die eventuelle Fehlersuche zu erleichtern. Fehler im Testablauf werden mit so genannten Event Handlern behandelt, die auf bestimmte Fehler einzeln oder zentral reagieren können. Ist ein auftretender Fehler nicht schwerwiegend, kann der Test weiterlaufen. Bei anderen Fehlern können Neustarts der AUT oder verschiedene Aufräumaktionen angesetzt werden.

Ausblick

Für die kommende Version 3.0 (voraussichtlich ab März 2009 verfügbar [2]) sind zwei wichtige Erweiterungen angekündigt: zum einen eine modellgetriebene Entwicklung (Modeling Perspective) von Testfällen aus Anwendungsfällen zum frühen Strukturieren von Tests, zum anderen ein leichterer Einstieg in das automatische Testen bei schon existierenden Softwareprojekten durch einen überarbeiteten Aufzeichnungsmodus. Aufgezeichnete Testschritte sollen sich dann kaum noch von spezifizierten Tests unterscheiden und über die Refactor-Funktion beliebig modularisiert werden können.

Zusammengefasst

Mit dem Keyword-driven-Ansatz erlaubt GUIdancer eine AUT-unabhängige Testspezifikation, die sich sowohl in agile als auch in traditionelle Prozesse integrieren lässt. Das programmatische Erstellen von Keywords entfällt aufgrund der umfangreichen Bibliothek. Sollte die Funktionalität nicht ausreichen, kann man über eine offene Schnittstelle neue Aktionen und Komponenten zu den schon unterstützten Implementierungen hinzufügen. Der Fokus auf die Wiederverwendung von Keywords hat den Effekt, dass aus der Entwicklung bekannte Standards auch im codefreien Bereich gefördert werden. Für RCP-Projekte, die automatische GUI-Tests einsetzen wollen, empfiehlt sich das Anfordern der 14-tägigen Demolizenz. Das GUIdancer Support Team [3] hilft beim Einstieg – von technischen Fragen über Best-Practices bis hin zu Schulungsmöglichkeiten. Bei der Gelegenheit kann ein Tester auch sehen, wie man GUI-Tests in den Build- und Test-Prozess integrieren kann.

Alexandra Imrie hat einen Abschluss als Master of Phonetik/Phonologie. Sie beschäftigt sich bei der BREDEX GmbH mit den Konzepten und der Planung für GUIdancer sowie mit Dokumentation, Kundensupport und Demos.

>> Links & Literatur

[1] GUIdancer Website: www.guidancer.de

[2] GUIdancer Version 3.0 Details: www.bredex.de/de/news/pdf/Version3.0_Announcement_de.pdf

[3] GUIdancer Support: gdsupport@bredex.de