Modellbasierte Testautomatisierung: Von der Anforderungsanalyse zu automatisierten Testabläufen

Das in diesem Artikel beschriebene Vorgehen einer modellbasierten Testautomatisierung versucht, aktuelle Testansätze auf die Phase der Anforderungsanalyse zu übertragen. Wenn in den Fachabteilungen Anforderungen mit Hilfe von Modellen erstellt werden, ist es sinnvoll und machbar, diese Modelle so zu erweitern, dass auch Testabläufe erstellt werden können. Aus diesen modellierten Testabläufen können mit Hilfe von Generatoren Testskripte generiert werden, die dann automatisiert ablaufen.

In der Disziplin des Softwaretestens hat sich in den letzten Jahren einiges getan. Die Testautomatisierung hat insbesondere über schlüssel- oder aktionswort-getriebener Frameworks einen so hohen Reifegrad erreicht, dass einem größeren Einsatz in Projekten nichts mehr im Wege steht. Über agile Projektmethoden ist die testgetriebene Softwareentwicklung ins Gespräch gekommen und weiter professionalisiert worden. Neben den modellbasierten Ansätzen zur Anforderungsanalyse und Softwareentwicklung gibt es nun auch das Vorgehen des modellbasierten Testens.

Allerdings sind diese angesprochenen Methoden und Techniken sehr entwicklungslastig. Bei der Testautomatisierung liegt dies in der Natur der Sache, bei der testgetriebenen Softwareentwicklung liegt es schon im Namen. Spätestens wenn es beim modellbasierten Testen um die Generierung von Testfällen geht, bleibt ein gewisses Entwicklungswissen unabdingbar. Somit besteht immer noch eine Kluft zwischen Fach- und Entwicklungsabteilungen, die schwer zu überbrücken ist. Wenn aber in den Fachabteilungen Anforderungen mit Hilfe von Modellen erstellt werden, ist es sinnvoll und auch machbar, diese Modelle so zu erweitern, dass auch Testabläufe modelliert werden können. Im Idealfall werden dadurch Geschäftsanalysten in die Lage versetzt, Testdurchläufe zu modellieren, die – ohne weitere Programmierung bzw. Skripterstellung – automatisiert ausgeführt werden können.

Für jeden durchzuführenden Test werden zwei Informationen benötigt:

- Der Testfall, zum Beispiel als eine Beschreibung des Durchlaufes durch eine Anwendung
- Die Testdaten.

Dabei beschreiben die Testdaten zum einen die Eingabedaten und zum anderen das erwartete Ergebnis, mit dem die Korrektheit des Testes verifiziert werden kann. Dies gilt selbstverständlich auch für automatisierte Tests – dabei werden die Testdaten meistens in Dateien (Excel) ausgelagert und der Ablauf des Tests wird als Skript programmiert, das in einem Automatisierungswerkzeug mit den jeweiligen Testdaten ausgeführt wird. Dieses Vorgehen besitzt mehrere Nachteile:

- Die Pflege der Testdaten-Dateien ist unübersichtlich und fehleranfällig.
- Der Testablauf ist nicht dokumentiert, da er nur als Skript existiert.
- Änderungen und Erweiterungen am Testablauf müssen programmiert werden.
- Änderungen an den Anforderungen (Ablauflogik, Geschäftsobjekte, Oberflächen) müssen umgehend in die Testdaten und Skripte eingearbeitet werden.

Ein erster Schritt, um diese Nachteile zu umgehen, ist die Entwicklung eines Frameworks mit Schlüssel- oder auch Aktionsworten (vgl. [Sei11). Die Testabläufe werden dabei über sequenziell angeordnete Schlüsselwörter beschrieben (ähnlich wie in Listing 1). Hinter diesen Schlüsselwörtern verbergen sich dann entsprechende Skripte, die die gewünschte Funktion ausführen. Dadurch sind die Testabläufe einfacher lesbar und auch erweiterbar.

In einem nächsten Schritt bietet es sich an, die Testdaten und -abläufe zu modellieren und die entsprechenden Skripte (mit den Schlüsselwörtern) aus diesen Modellen zu generieren. Basiert auch die Anforderungsanalyse auf Modellen, so können die Testdaten mit den Geschäftsobjekten und die Testabläufe mit den Oberflächendefinitionen verknüpft werden. Dadurch können Änderungen in den Anforderungen bis in den Testablauf nachverfolgt und eingearbeitet werden. Gleichzeitig entsteht eine lesbare Dokumentation der Testabläufe mit den Testdaten. Unterstützt wird somit auch eine frühzeitige Validierung der Anforderungen, da die Testabläufe parallel mit der Anforderungsanalyse erstellt werden können. Der Hauptvorteil liegt allerdings darin, dass aus diesen Modellen die Testskripte automatisch generiert werden können.

Voraussetzung für diese modellbasierte Testautomatisierung ist neben einem Testautomatisierungswerkzeug mit einem

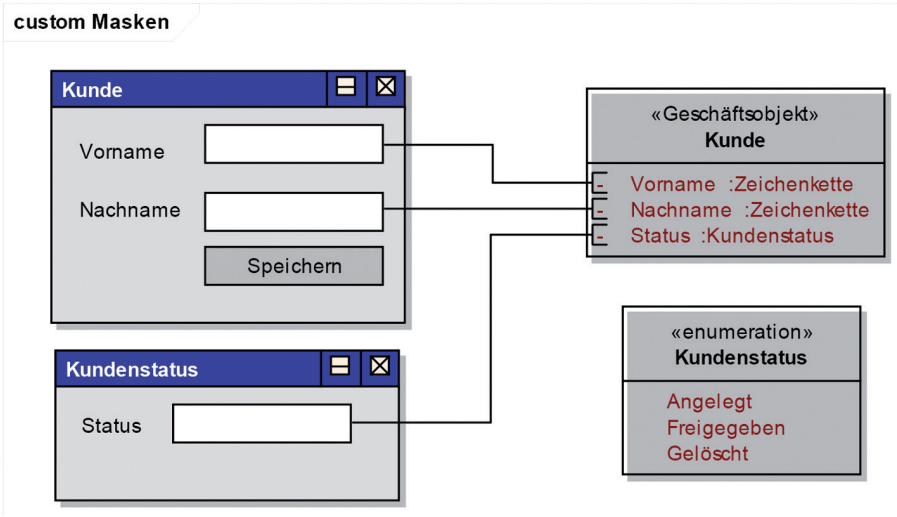

Abb. 1: Ergebnisse aus der Anforderungsanalyse.

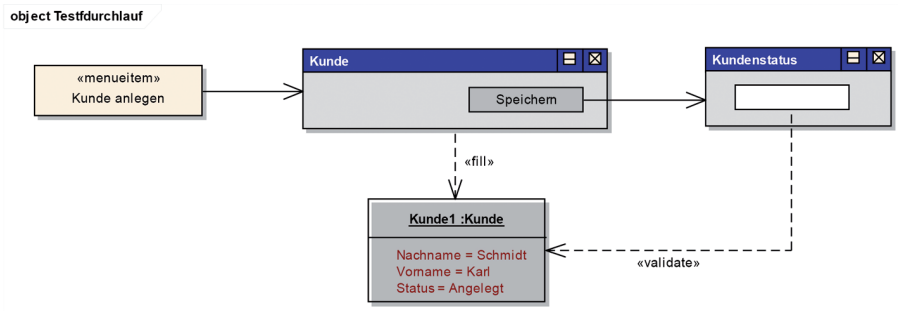

Abb. 2: Testdurchlauf.

intelligenten Framework auch ein Modellierungswerkzeug. Die Möglichkeiten, die ausgeschöpft werden können, sind grundsätzlich abhängig vom Reifegrad des eingesetzten Modellierungswerkzeuges.
Dieses Vorgehen möchten wir im Folgenden an einem Beispiel verdeutlichen. Dabei handelt es sich um eine maskenbasierte Anwendung – allerdings können die Ergebnisse ohne Weiteres auch auf Schnittstellen, Batch-Abläufe usw. übertragen werden.

Ergebnisse der Anforderungsanalyse

Die Ergebnisse einer jeglichen Anforderungsanalyse sollten zumindest die Geschäftsobjekte mit den zugehörigen Attributen sein. Bei der Definition der Oberflächen gibt es häufig einen Streit, ob dies nun zur Anforderungsanalyse oder zum IT-Design der Anwendung gehört. Für die hier angestellten Überlegungen ist dies irrelevant: Es wird davon ausgegangen, dass diese Oberflächendefinitionen existieren. Des Weiteren ist es möglich, die Attribute eines Geschäftsobjekts auf entsprechende Eingabefelder einer Oberfläche abzubilden. Im Idealfall ist das Ergebnis der Anforderungsanalyse ein Modell, wie in **Abbildung 1** dargestellt.
Für eine modellbasierte Testautomatisierung müssen die Geschäftsobjekte und die Oberflächen in der Anforderungsanalyse definiert und mit einem Modellierungswerkzeug abgebildet werden. Ist dies nicht der Fall, so bleibt die Möglichkeit, aus den bestehenden, natürlich-sprachlich formulierten Fachkonzepten die entsprechenden Geschäftsobjekte und Oberflächen zu extrahieren und sie in einem Modellierungswerkzeug, wie abgebildet, zu modellieren. Sind selbst die Geschäftsobjekte nicht zu extrahieren, so kann zu jeder Oberfläche ein Datencontainer als „Geschäftsobjekt“ definiert werden. Wichtig ist dies, weil über diese Geschäftsobjekte die Struktur der Testdaten für die Testautomatisierung definiert wird.

Beschreibung eines Testdurchlaufes

Nach der reinen Lehre sind Testfälle von den Testdaten strikt zu trennen. In einem ersten Schritt werden logische Testfälle definiert, die dann mit Testdaten in konkrete Testfälle überführt werden (vgl. [Spi12]). Im Zusammenhang mit der Testautomatisierung wird von einem Testdurchlauf gesprochen. Ein Testdurchlauf entspricht dabei einem logischen Testfall, der mit genau einem Testdatum automatisiert ausgeführt wird. Zur Vereinfachung betrachten wir im Folgenden nur Testdurchläufe, die als Modell abgebildet werden – ein modellierter Testdurchlauf beinhaltet also zum einen den logischen Testfall und zum anderen auch die Testdaten.
Abbildung 2 verdeutlicht einen beispielhaften Testdurchlauf, der die modellierten Elemente aus der Anforderungsanalyse in **Abbildung 1** wiederverwendet. Hinter dem Testdurchlauf verbirgt sich diese fachliche Anforderung: Jeder neu angelegte Kunde hat den Status „Angelegt“. Der fachliche Testdurchlauf ist: Öffne die Oberfläche „Kunde“, trage die Testdaten aus „Kunde 1“ in die Eingabefelder ein, aktiviere die Schaltfläche „Speichern“ und überprüfe in der neuen Oberfläche, ob der Status „Angelegt“ angezeigt wird.
Entscheidend ist dabei, dass durch die Modellierung sichergestellt wird, dass bei den Testdaten auch wirklich die Attribute belegt werden, die im Geschäftsobjekt definiert sind und die dadurch mit einem Eingabefeld der Oberfläche verbunden sind. (An dieser Stelle wird die UML-Notation für Klassen und Objekte verwendet, die genau diese Beziehung definiert, vgl. [Rum10].)
Mit dieser Methode können auch Fehler-Testfälle erzeugt werden, indem zum Beispiel Attribute nicht belegt werden und eine Fehlermeldung als erwartete Maske modelliert wird.

Generierung des Testskripts

Grundvoraussetzung für eine Generierung von Testskripten ist, dass ein Metamodell für die Modellierung von Testdurchläufen erstellt wird. Dieses Metamodell ist abhängig von der zu testenden Applikation, den Anforderungen an die Testautomatisierung, vom eingesetzten Testwerkzeug und vielen weiteren Parametern. Für den hier dargestellten Testdurchlauf wurde zum Beispiel in einem Metamodell definiert, wie das Eintragen von Testdaten in eine Oberfläche zu modellieren ist: über eine Abhängigkeitsbeziehung mit dem Kennzeichen `<<fill>>`.
Diese Verpflichtungen bezüglich der Notation müssen von allen Modellierern eingehalten werden, damit ein Generator die Informationen eindeutig auswerten kann. Das gilt grundsätzlich für jegliche Art der modellbasierten Methoden, da nur so sichergestellt werden kann, dass die Modelle auch von jedem Stakeholder gleichartig interpretiert und verstanden werden.
Viele Werkzeuge für die Testautomatisierung funktionieren in zwei Schritten:

- In einem ersten Schritt werden die Elemente, die eine Oberfläche beinhaltet, ausgewertet und an einer zentralen Stelle so abgelegt, dass sie eindeutig identifiziert werden können (so zum Beispiel die GUI-Map bei dem Werkzeug „Quick Test Professional“ oder UIMap bei der Automatisierung mit „CodedUI Test für Microsoft Visual Studio“).

```
Öffne Maske (Kunde.ID)
Fülle Feld (Kunde.Vorname.ID, „Karl“)
Fülle Feld (Kunde.Name.ID, „Schmidt“)
Drücke Schaltfläche (Kunde.Speichern.ID)
Validiere ist Maske offen (Kundenstatus.ID)
Validiere Feld (Kundenstatus.Status.ID, „Angelegt“)
```

Listing 1: Skript (Pseudocode).

```
Kunde.ID = M#001
Kunde.Vorname.ID = M#001F#001
Kunde.Nachname.ID = M#001F#002
Kunde.Speichern.ID = M#001#003
Kundenstatus.ID = M#002
Kundenstatus.Status.ID =
M#002F#001
```

Listing 2: Oberfläche und Elemente (Pseudocode).

- In einem zweiten Schritt werden diese Elemente dann über das Erstellen eines Skripts zu einem Testdurchlauf angesprochen und bei Bedarf aktiviert, mit Werten ausgefüllt oder der Inhalt wird ausgelesen.

Beide Schritte – die Definition der Oberflächen und die Testdurchlaufskripte – können durch einen Generator erzeugt werden. Die Oberflächenelemente werden aus der Definition der Oberflächen aus dem Modell übernommen und vom Generator in eine Datei mit den Oberflächen-Definitionen geschrieben (**siehe Listing 2**). Die eindeutigen IDs für die Elemente können beliebig aus dem Modellierungswerkzeug ermittelt werden. Dabei bietet sich die eindeutige ID an, die das Werkzeug dem Element zugeordnet hat. Wichtig ist, dass diese IDs auch von den Entwicklern der Applikation verwendet werden, damit die Elemente durch das Testautomatisierungswerkzeug gefunden werden können (siehe weiter unten).
Anhand des modellierten Testdurchlaufs und des definierten Metamodells kann nun der Generator auch automatisch das Testskript erstellen. In **Listing 1** ist das zu **Abbildung 2** gehörende Skript in Pseudocode dargestellt. Dieses Skript muss natürlich an die Skriptsprache des verwendeten Automatisierungswerkzeugs, die verwendeten Schlüsselwörter und das implementierte Framework angepasst werden. Dabei bleibt ein weiterer Vorteil der modellbasierten Testautomatisierung anzumerken: Die Testdurchläufe sind sprachunabhängig und können durch konfigurierbare Generatoren auf unterschiedliche Testautomatisierungs-Werkzeuge abgebildet und ausgeführt werden.

Anforderungen an die Entwicklung

Damit die entsprechenden Oberflächenelemente bei der Testdurchführung gefunden werden, müssen bei der Entwicklung der Oberflächen bestimmte Regeln eingehalten werden.

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
   <title id="M#001">
      Kunde
   </title>
</head>
<body>
   <form action="kundenstatus.html">
      <input type="input" id=" M#001F#001" />
      <input type="input" id=" M#001F#002" />
      <input type="submit" id="M#001F#003" />
   </form>
</body>
</html>
```

Listing 3: Beispiel HTML-Quellcode.

Die Oberflächen werden vor der Umsetzung in der Entwicklung mit den Entwicklern besprochen. Dabei werden der Typ jedes Elements und dessen Abhängigkeit zu anderen Aktionen, wie Eingaben oder das Drücken eines Buttons, eindeutig definiert. Außerdem werden die IDs der Elemente, die später für die Zuweisung und Erkennung der Elemente einer Seite für die Testautomatisierung notwendig sind, im Modell festgelegt. Die Entwickler müssen sich bei der Umsetzung an diese Vorgaben aus dem Modell halten, wobei der Aufbau der Seite und die Platzierung der einzelnen Elemente nicht relevant für die Testautomatisierung sind, da sich der Testablauf aus den Testfällen im Modellierungswerkzeug und den dahinter liegenden IDs der Elemente ergibt. Ein Beispiel für eine HTML-Oberfläche, die die in **Listing 2** definierten Felder verwendet, ist in **Listing 3** dargestellt.

Werkzeugunterstützung und der Testskript-Generator

Die Erfolge bzw. die Einsatzmöglichkeiten der hier beschriebenen modellbasierten Testautomatisierung stehen und fallen mit der Mächtigkeit der eingesetzten Werkzeuge. Das hier verwendete Beispiel wurde mit dem „Enterprise Architect" der Firma Sparx Systems erstellt. Dieses Modellierungswerkzeug unterstützt zum einen die Modellierung von Objekten mit Werten für die entsprechenden Attribute der zugeordneten Klasse, die zur UML Spezifikation gehört (vgl. [Rum10]). Zum anderen verfügt es über zusätzliche Eigenschaften, wie die Möglichkeit der Modellierung von Oberflächen und die Verknüpfung von Eingabefeldern zu Attributen einer Klasse (**siehe Abbildung 1 und Abbildung 2**). Diese zusätzlichen Eigenschaften gehören allerdings nicht zur UML-Spezifikation. Somit ist bei der Auswahl eines Modellierungswerkzeugs darauf zu achten, dass auch die für die Modellierung der Testdurchläufe benötigten Elemente und Beziehungen – die wiederum im Metamodell definiert werden – unterstützt werden.
Des Weiteren sind die Auswirkungen auf den Testskript-Generator zu berücksichtigen. Dieser muss die für das Testautomatisierungswerkzeug benötigten Dateien aus den Informationen, die im Werkzeug modelliert sind, und mit Hilfe des definierten Metamodells erzeugen. Dadurch, dass das Metamodell jeweils von den definierten Testanforderungen und dem Modellierungswerkzeug abhängt, wird es mit hoher Wahrscheinlichkeit keine Standardgeneratoren auf dem Markt geben. Der Generator muss also selbst konzeptioniert und implementiert werden. Als Eingabe benötigt der Generator die im Werkzeug modellierten Informationen, die er entweder über eine Programmierschnittstelle zu dem Werkzeug abfragen kann, oder indem er geeignete Exportfunktionen, die das Werkzeug zur Verfügung stellt, ausnutzt. Der Weg über eine Programmierschnittstelle ist robuster, einfacher zu handhaben und somit als Standardweg zu empfehlen. Der Weg über eine Exportfunktion könnte den Charme haben, dass – falls es sich um eine XMI-Schnittstelle und somit um eine standardisierte Schnittstelle für den Austausch von UML-Modellen (vgl. [Gro02]) handelt – der Generator unabhängig vom verwendeten Werkzeug programmiert werden kann. Das gelingt allerdings nur, wenn für die modellbasierte Testautomatisierung Modellierungselemente verwendet werden, die zum UML-Standard gehören bzw. die über den Erweiterungsmechanismus der UML-Profile abgebildet werden können (vgl. [Rum10]). Zusätzlich muss das Mo-

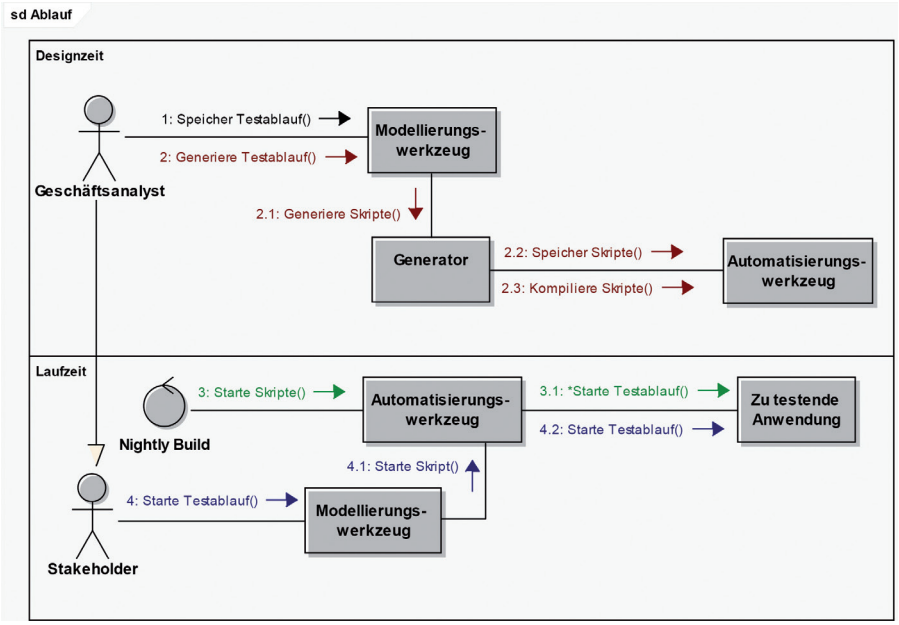

Abb. 3: Ablauf bei der modellbasierten Testautomatisierung.

dellierungswerkzeug bei diesem Vorgehen auch das Modell korrekt exportieren. Der Enterprise Architect exportiert zum Beispiel die Objekte mit den Attributwerten nicht wie in der XML-Spezifikation vorgesehen, sondern in einem proprietären Format, was dann die Übertragbarkeit sehr stark einschränkt.

Ablauf

Wie sieht nun der konkrete Ablauf bei der modellbasierten Testautomatisierung aus? Es ist zu unterscheiden zwischen dem Ablauf während der Designzeit und dem Ablauf zur Laufzeit (**siehe Abbildung 3**).

In der Designzeit können der Geschäftsanalyst oder auch Testanalyst beliebige Testabläufe erstellen und pflegen (Sequenz 1 in **Abbildung 3**). Alle Änderungen werden im Modellierungswerkzeug gespeichert, haben aber noch keine Auswirkungen auf die bereits existierenden, automatisierten Testdurchläufe. Zu einem definierten Zeitpunkt (beispielsweise „Anforderungen/Testfälle sind abgenommen“ oder „zu testende Anwendung steht bereit“) können dann alle gespeicherten Testabläufe in das Automatisierungswerkzeug übertragen werden (Sequenz 2 in **Abbildung 3**). Dies kann direkt aus dem Modellierungswerkzeug heraus angestoßen werden, z.B. über einen zusätzlichen Menüpunkt, der den Generator aktiviert, oder indem der Generator direkt gestartet wird. Die Aufgabe des Generators ist es nun, die entsprechenden Skripte zu erzeugen, diese am korrekten Ablageort abzulegen und gegebenenfalls die Kompilierung der Skripte durch das Testautomatisierungswerkzeug anzustoßen. Als Ergebnis stehen somit automatisiert ausführbare Testdurchläufe zur Verfügung.

Das Ausführen dieser Testdurchläufe geschieht zur Laufzeit. Jeder berechtigte Stakeholder kann über das Modellierungswerkzeug die Ausführung eines Testdurchlaufs starten (Sequenz 4 in **Abbildung 3**). Auch dies kann über entsprechende Erweiterungen in den Menüs des Werkzeugs realisiert werden. Eine zeitpunktgesteuerte Ausführung der Testdurchläufe ist auch möglich (Sequenz 3 in **Abbildung 3**). Existiert zum Beispiel im Projekt ein nächtlich erstellter Versionsstand (*Nightly Build*), der im nächsten Schritt automatisch auf einem Testsystem bereitgestellt wird, kann im Anschluss an diese Bereitstellung die Ausführung aller automatisierten Testdurchläufe gestartet werden.

Erfolgsfaktoren

Wie bereits angedeutet, benötigt die modellbasierte Testautomatisierung die modellierten Geschäftsobjekte und die definierten Oberflächen aus der Fachkonzeption. Erstellt der Fachbereich diese Informationen bereits mit einem Modellierungswerkzeug und ist dadurch mit dem Arbeiten eines Werkzeugs vertraut, so ist es ein Leichtes, den Fachbereich zu befähigen, auch die Testdurchläufe zu modellieren. Wenn dem nicht so ist, ist die Einarbeitung des Fachbereichs in ein Modellierungswerkzeug eine nicht zu unterschätzende Aufgabe. Insbesondere ist zu beachten, dass die Geschäftsobjekte und Oberflächen nicht annähernd so simpel sind, wie hier in den Abbildungen dargestellt. Der Aufbau von komplexen Geschäftsobjektmodellen, Oberflächendefinitionen und deren Verknüpfung über die Instanziierung von Geschäftsklassen erfordert ein grundlegendes Verständnis der UML-Notation und deren korrekte Anwendung in einem Modellierungswerkzeug.

Der Einsatz eines Modellierungswerkzeugs sollte somit nur dann angegangen werden, wenn die erstellten Modelle primär für die Fachkonzeption Verwendung finden und nicht ausschließlich für die Testautomatisierung benutzt werden. Zusätzlich profitiert allerdings auch der Entwicklungsbereich von diesen Modellen, z.B. durch die Möglichkeit der Generierung von Quellcode-Dateien oder die Möglichkeit der Verknüpfung von DV-Konstrukten mit entsprechenden Anforderungen. Eine modellbasierte Testautomatisierung ist also nur in Verbindung mit einer modellbasierten Anforderungsanalyse sinnvoll.

Ein weiterer kritischer Erfolgsfaktor ist die Implementierung des Generators, bei dem das Wissen aus mehreren Bereichen nötig ist: Das Modellierungswerkzeug und das Testautomatisierungswerkzeug müssen beherrscht und die Schnittstelle zum Modellierungswerkzeug muss verstanden werden (dies impliziert ein tiefgehendes Verständnis der UML). Der Entwickler des Generators muss somit in der Welt der Anforderungen, der Implementierung sowie der Testfaller-

Literatur

[Bec02] K. Beck, Test Driven Development, Addison-Wesley 2002
[Gro02] T. J. Grose, G.C. Doney, S.A. Brodsky, Mastering XMI, John Wiley & Sons 2002
[Roß10] T. Roßner, C. Brandes, H. Götz, M. Winter, Basiswissen modellbasierter Test, dpunkt.verlag 2010
[Rum10] J. Rumbaugh, I. Jacobson, G. Booch, The Unified Modeling Language Reference Manual, Addison Wesley 2010
[Sei11] R. Seidl, M. Baumgartner, T. Bucsics, Basiswissen Testautomatisierung, dpunkt.verlag 2011
[Spi12] A. Spillner, T. Linz, Basiswissen Softwaretest, dpunkt.verlag 2012

stellung und -durchführung zu Hause sein. Aus dem bisher Gesagten wird deutlich, dass das Vorgehen einer modellbasierten Testautomatisierung keine Aufgabe ist, die ein einzelnes Projekt für sich allein angehen sollte. Eine strategische Verankerung des Vorgehens in einem Unternehmen ist sehr zu empfehlen.

Fazit

Die modellbasierte Testautomatisierung ist ein weiterer notwendiger Baustein modellbasierter Teststrategien (zu weiteren modellbasierten Teststrategien vgl. [Roß10]). Mit entsprechenden Werkzeugen und einer durchdachten Konzeption können aus der Anforderungsanalyse mit wenig zusätzlichem Aufwand passende, automatisierte Testdurchläufe zur Verifizierung der Anforderungen erstellt werden. Das kann bei entsprechender Einweisung sogar direkt durch die Geschäftsanalysten geschehen. Die Auswirkungen von Anforderungsänderungen auf die Testdurchläufe werden im Werkzeug umgehend identifiziert, die Testdurchläufe können sofort angepasst und die Testskripte neu generiert werden. Damit sind auch die nicht zu unterschätzenden Kosten, die für die Konzeption und Realisierung eines Testskript-Generators aufzubringen sind, sinnvoll begründet und eingesetzt.

Des Weiteren unterstützt und erweitert die Methode der modellbasierten Testautomatisierung auch den Ansatz einer testgetriebenen Softwareentwicklung (vgl. [Bec02]). Dieser Ansatz – Schreibe erst die Tests (z. B. Unit-Tests), automatisiere diese Tests und entwickle dann solange, bis alle Tests erfolgreich ablaufen – beschränkt sich hauptsächlich auf Entwicklertests (Modul- und Integrationstest). Mit dem hier beschriebenen Ansatz kann dieses Vorgehen auf System- und Abnahmetests ausgeweitet werden: Modelliere erst die Testfälle (Fachbereich), generiere die automatisierten Testabläufe und entwickle dann die Anwendung, bis alle Testdurchläufe erfolgreich ablaufen. ||

Die Autoren

|| Achim Krallmann
(achim.krallmann@sqs.com)
ist als Test- und Projektmanager bei dem Unternehmen SQS tätig. Er ist graduierter systemischer Organisationsmanager und unterstützt Unternehmen bei der Einführung und Umsetzung von innovativen Testmethoden.

|| Markus Lingelbach
(markus.lingelbach@sqs.com)
ist als Testanalyst und Testautomatisierer bei der SQS beschäftigt und unterstützt Projekte im Testdesign und bei der Testdurchführung.