Interessanter ist aber der Fall, wenn bereits eine schlüsselwortbasierte Testautomatisierung vorliegt. Dann hat man, ein geeignetes Framework vorausgesetzt, einen riesengroßen Vorteil: Dann sind nämlich wirklich dieselben Testfälle jederzeit wahlweise manuell und automatisiert durchführbar.

Bei einer klassischen skriptbasierten Automatisierung wäre das viel schwieriger. Hier ist anhand des Skripts manchmal schon nicht mehr zu erkennen, was der implementierte Testfall eigentlich überprüft.

Als Voraussetzung dafür, dass diese Wahlmöglichkeit zwischen manueller und automatisierter Durchführung bei Keyword-Driven Testing wirklich funktioniert, muss bei der Erstellung und Dokumentation der Keywords bereits berücksichtigt werden, dass die erstellten Testfälle auch für Menschen gedacht sind. In erster Linie heißt das, dass sie genug Informationen für Testerinnen bieten müssen, sodass sie diese Testfälle lesen und verstehen können.

Diese Bedingung ist tatsächlich nicht so schwer umzusetzen, und der Erfolg kann anhand von Stichproben leicht überprüft werden.

2.5 Keyword-Driven Testing im agilen Kontext

Keyword-Driven Testing ist agnostisch gegenüber dem Software-Lebenszyklusmodell. Ob »klassisch« (etwa V-Modell oder Wasserfall) oder agil (beispielsweise Scrum oder Kanban) spielt eigentlich keine Rolle. Hinsichtlich »klassisch« können wir festhalten, wir haben KDT schon vor über zwanzig Jahren erfolgreich in (damals war das keine Frage) nicht agilen Projekten eingesetzt. Heute setzen wir es ebenfalls erfolgreich in meist agilen Vorhaben ein.

Aus Sicht von KDT mag es egal sein, umgekehrt aber ist Keyword-Driven Testing *gerade* im agilen Kontext besonders notwendig.

Mitglieder agiler Teams begegnen dem oft mit Skepsis: *Wie sollen wir auch noch die Zeit haben, uns mit diesem Keyword-Overhead zu beschäftigen?* Fragen wie diese sind verständlich. Viele agile Teams haben die Erfahrung gemacht, dass Zeit für Testautomatisierung kaum zu finden ist. Und der wird KDT zugeordnet.

Warum also meinen wir, dass KDT gerade im agilen Kontext verwendet werden sollte, trotz all der Zeitnot?

Hauptsächlich aus drei Gründen:

- Gerade wegen der Zeitnot
- Wegen der häufigen Änderungen
- Wegen der besseren Lesbarkeit der Testfälle

Der Reihe nach:

Zeitnot in Iterationen: Agile Softwareentwickung wird in Iterationen gegliedert, beispielsweise in Sprints. Ein Sprint dauert typischerweise zwei Wochen (selten noch kürzer, manchmal bis zu vier Wochen). In diesem Zeitraum muss nun alles passieren, was zum nächsten Inkrement der Software benötigt wird. »DONE« heißt fertig, und fertig heißt auch, fertig getestet. Viel Zeit zum Testen ist eigentlich nie, und oft ballt sich der Testaufwand am Ende des Sprints. Was in der letzten Minute implementiert wurde, soll in der allerletzten Minute noch getestet werden.

Immer mehr Regressionstests

Natürlich reicht es nicht, die neuen Features zu testen. Es braucht auch Regressionstests, die sicherstellen, dass nicht etwas bereits Funktionierendes durch die neuen Features zerstört wurde. Diese Stichprobe an Tests durch die gesamte Bandbreite der Funktionalität wird immer größer, weil immer mehr Features umgesetzt werden. Testet man manuell, dann schluckt das irgendwann die gesamte Testkapazität. Und das ist der Grund, warum Testautomatisierung in agilen Projekten unverzichtbar ist. Es sein denn, man ist sehr mutig – Mut zur Lücke.

Testautomatisierung wiederum bringt Wartungsbedarf mit sich – vor allen Dingen, wenn sie benutzernah stattfindet. Und wenn hier nicht gegengesteuert wird, kommt man trotz und mit Testautomatisierung ins nächste Dilemma, und wieder wird jegliche Kapazität für Wartung benötigt. Die Lösung hier ist Keyword-Driven Testing: Denn damit adressieren wir das Wartungsproblem.

Die Zeit, die Testautomatisierung und Keyword-Driven Testing kostet, um sauber aufgesetzt zu werden, ist gut investiert. Denn nur so gelingt es, in engen Zyklen Software zu entwickeln, ohne Abstriche an der Testabdeckung zu machen.

Häufige Änderungen: Agil wirbt für sich damit, mit häufigen Änderungen umgehen zu können, ja sie geradezu zu lieben. Ob wir uns dem nun anschließen wollen oder nicht (wäre es nicht schön, etwas gleich richtig zu machen, statt häufig zu ändern), häufige Änderungen sind die Realität in agilen Teams. Damit können und wollen wir umgehen.

Allerdings heißt das auch, dass die Testfälle – und es sind wieder vor allen Dingen die automatisierten Tests – sehr häufig angepasst und geändert werden müssen. Also muss die Testautomatisierung optimal wartbar sein – und damit sind wir wieder beim Keyword-Driven Testing, bei dem der Wartungsaufwand einer wachsenden Anzahl von automatisierten Testfällen dennoch annähernd linear bleibt.

Lesbare Testfälle: Dieser dritte Punkt zielt nicht (nur) auf Testautomatisierung. Die fertig formulierten Testfälle sollen ja lesbar sein, für alle im Team. Idealerweise sind alle Teammitglieder gleichermaßen voll qualifiziert für alles, was an Aufgaben anstehen mag, und jeder muss also auch alle Testfälle lesen und verstehen können. Das alleine wäre schon ein Grund, mit KDT eine Domain Specific Language (DSL) aufzubauen, die das gleiche Verständnis der Testfälle sicherstellt.

Aber zurück zur Realität: Das ist ein schönes Idealbild. Die Menschen, mit denen wir arbeiten, haben alle ihre unterschiedlichen Stärken, und wir können von Glück reden, wenn das Team als Ganzes alle Fähigkeiten besitzt. Testen kann vielleicht nicht jeder, und so gibt es implizit vielleicht doch so etwas wie »Testerinnen« im Team. Dann gibt es auch eine Aufgabenteilung zwischen Menschen, die eher fachlich arbeiten, und solchen, die eher technisch unterwegs sind. Die Herausforderung, für alle lesbare Testfälle zu haben, bleibt bestehen und wird durch die Unterschiede noch größer. Gerade in diesem Zusammenhang spielt Keyword-Driven Testing seine Stärken aus: Lesbare Testfälle, die in mehreren Schichten organisiert sind, wahlweise manuell oder automatisiert ausführbar sind, und bei denen Arbeitsteiligkeit funktioniert.

An dieser Stelle wollen wir noch einen Blick auf die Testpyramide werfen, wie sie in Abbildung 2-2 gezeigt wird. Wünschenswert ist, dass die entwicklungsnahen Tests, Komponententests, den Hauptteil der Tests ausmachen, und die Tests, je näher man der Oberfläche und den Usern kommt, immer weniger werden, weil so vieles Wichtige schon vorher getestet wurde.

KDT in der Testpyramide

Der Einsatz von Keyword-Driven Testing wird gerade im agilen Umfeld meist auf Systemtestebene stattfinden. Man könnte daher annehmen, dass der Hebel, den man dort ansetzen kann, gar nicht so viel bewegt – wenn man eine »gesunde« Testpyramide vor sich hat. Das ist allerdings nur auf den ersten Blick richtig.

Wenn man genauer hinsieht, stellt man fest, dass die vielen Testfälle der unteren Ebenen, die für die Testpyramide das Fundament bilden, wenn sie erst einmal da sind, sehr stabil und verhältnismäßig pflegeleicht sind. Die Systemtests machen viel mehr Ärger. Daher lohnt es sich gerade hier, mit Keyword-Driven Testing für robuste und wartbare Systemtests zu sorgen.

Ein typischer Aspekt bei agiler Softwareentwicklung ist die Definition von Akzeptanzkriterien. Diese sind wiederum die ideale Grundlage für Konzepte wie Acceptance Test-Driven Development (siehe Abschnitt 8.4). Darauf und auf andere im Agilen verbreitete Konzepte, wie Test-

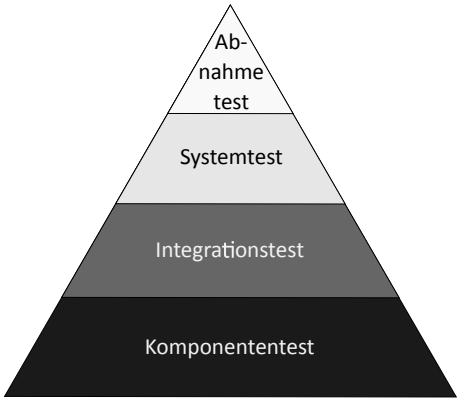

Abb. 2-2
Testpyramide (Wunschbild)

Driven Development (siehe Abschnitt 8.2) und Behavior-Driven Testing (siehe Abschnitt 8.3), gehen wir in eigenen Abschnitten ein.

2.6 Model-Based Testing und Keyword-Driven Testing

Modellbasierter Test wird seit Jahren diskutiert. Es gab bereits eigene Konferenzen dazu, beispielsweise im Jahr 2008 den »QS-Tag« [25], es gibt einen Lehrplan des ISTQB® dazu [29] und bei ISO ist ein Technical Report zu dem Thema in Vorbereitung (voraussichtlich unter der Nummer 29119-8). Auch an Literatur mangelt es nicht, mit [11] ist ein umfassendes deutschsprachiges Fachbuch zu dem Thema verfügbar.

Kurz, Model-Based Testing scheint zu den etablierten Testpraktiken zu gehören. Das trifft auch zu, bedeutet aber nicht, dass Model-Based Testing auch nur annähernd überall, wo Software getestet wird, zum Einsatz kommt. Wenn es genutzt wird, dann meist dort, wo Fehler sehr teuer oder gefährlich wären und sehr gründlich getestet werden muss, und daher die zusätzlich notwendige Beschäftigung mit dem Modell nicht gescheut wird. Häufig eingesetzt wird es bei Embedded Software, die Steueraufgaben hat, insbesondere in sicherheitskritischen Bereichen.

2.6.1 Überblick Model-Based Testing

Was kann man sich nun unter Model-Based Testing vorstellen? Dem Namen nach kommt ein »Modell« als Grundlage des Testens zum Einsatz. Das alleine wäre jetzt noch nicht sehr aussagekräftig, denn unter einem »Modell« kann man grundsätzlich eine vereinfachte Repräsentation der Wirklichkeit verstehen (vgl. Definition nach Stachowiak in

Wikipedia [40]). In diesem Sinne kann wohl jede Grundlage, die zum systematischen Testen von Software verwendet wird, als Modell gesehen werden – und dann wäre jedes Testen modellbasiert.

Im engeren Sinne verstehen wir aber unter Model-Based Testing die Art von Testen, bei dem systematisch, wiederholbar und in der Regel automatisiert aus formalen Modellbeschreibungen Testfälle, Testdaten oder Teile der Testinfrastruktur abgeleitet werden.

Das deckt sich mit der Definition von Winter und Roßner aus [11]:

> *Modellbasiertes Testen* umfasst mindestens einen der beiden folgenden Aspekte:
>
> - die Modellierung von Artefakten im Testprozess sowie
> - die Nutzung von Modellen für die Automatisierung von Testaktivitäten.

Das Format, in dem das Modell spezifiziert ist, spielt daher nur insofern eine Rolle, als der verwendete Generator in der Lage sein muss, es zu verarbeiten.

Der Vorteil beim Model-Based Testing ist, dass ein Modell für Softwaretest in der Regel gezielt einen Aspekt des Testobjektes beschreibt, zu dem mehrere Testfälle abgeleitet werden können und sollen. Durch das maschinelle Generieren der Testfälle – unter Angabe der gewünschten Abdeckung in Bezug auf das Modell – ist es sehr einfach und günstig möglich, Testfälle vollständig in Bezug auf das gewünschte Kriterium zu erhalten und auch zu pflegen: Ändern sich die Anforderungen und ändert sich daher das Modell, dann sind die Testfälle schnell aktualisiert.

Der Haken an der Sache? Da gibt es zumindest zwei:

1. Das Modell fällt leider nicht vom Himmel, es muss erstellt und gepflegt werden, und das erfordert nicht nur Aufwand, sondern auch Menschen, die modellieren können.
2. Wenn wir nun ganz einfach ganz viele Testfälle generieren können, wollen die auch durchgeführt werden. Die Versuchung ist groß, weil es ja nun so einfach ist, viel mehr Testfälle zu erzeugen. Ohne Automatisierung wird man diese Aufgabe nicht lösen können. Wir kommen in Abschnitt 2.6.3 darauf zurück.

Zum ersten Punkt ist noch zu sagen, dass die zusätzlichen Aufwände gerade in Umgebungen, in denen ohnehin viel mit Modellen gearbeitet wird, weniger ins Gewicht fallen – und von den Vorteilen, zu denen auch eine sichere Übereinstimmung der Testfälle mit der Spezifikation (in Form des Modells) gehört, aufgewogen werden können.

Harmonie birgt Gefahren

Die Harmonie zwischen Testfällen und Modell – ihre garantierte Übereinstimmung – ist nur so lange ein Pluspunkt, wie das Modell stimmt. Bei einem falschen Modell werden falsche Testfälle auch falsch negative Ergebnisse melden.

Ein Review der Modelle für den Test ist also essenziell (vgl. S. 246 in [11])!

Zum zweiten Punkt: Wenn die große Anzahl der generierten Testfälle zu einem Problem zu werden droht, sollte man sich auf risikobasiertes Testen besinnen. Es ist daher wichtig, beim Generieren darauf zu achten, dass die Testintensität (und damit die Anzahl der Testfälle) abhängig vom Risiko der betreffenden Komponente oder des jeweiligen Themas gesteuert werden kann.

2.6.2 Beispiel für Model-Based Testing

Viele Modelle und Notationen eignen sich für Model-Based Testing. Wir haben uns hier für den Typ Aktivitätsdiagramm in der Notation UML entschieden. Damit soll der Ablauf eines Kaufes in unserem Onlineshop modelliert und dann getestet werden.

Abbildung 2-3 zeigt das Modell, das wir zu diesem Zweck erstellt haben.[2]

In diesem einfachen Modell sind verschiedene Aktionen zu sehen, die zum Kauf eines unserer Kunsthandwerksartikel führen. Der Anwender hat an zwei Stellen die Wahl, weitere Artikel dem Warenkorb beizufügen oder die Anzahl der Artikel zu ändern (womit, zur Vereinfachung, auch Entfernen von Artikeln abgedeckt ist). Abgeschlossen wird der Kauf mit einer Anmeldung (Login) am System, die gültig oder ungültig sein kann, mit davon abhängigem Erfolg des Vorgangs.

Füttert man nun einen Generator mit diesem Modell, so kann dieser daraus Sequenzen von Aktionen ableiten, die als Grundlage für Testfälle dienen können. Tabelle 2-3 enthält in den Spalten einige solcher Sequenzen.

[2] Die Frage, ob bestehende Modelle, die für die Implementierung erstellt wurden, verwendet werden können, wird gerne gestellt. Das hat Vor- und Nachteile. Die Diskussion davon führt hier zu weit, daher sei auf die Literatur verwiesen, z.B. [11].

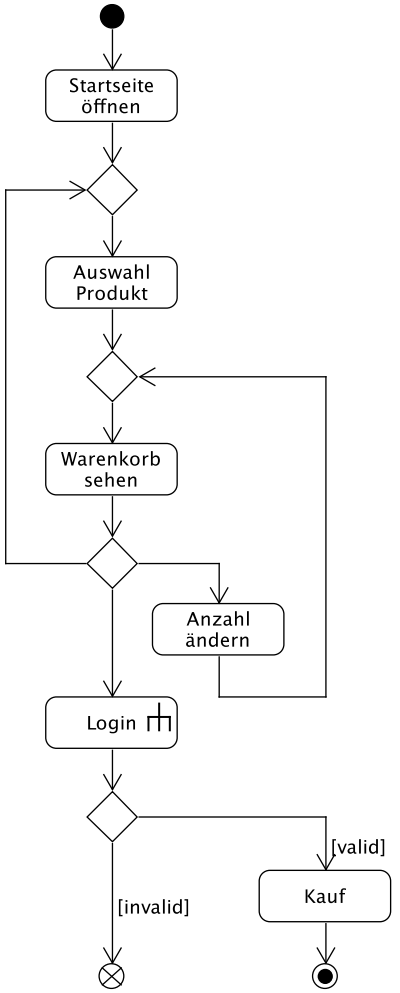

Abb. 2-3 *UML-Diagramm: Einkauf*

Tabelle 2-3 *Testsequenzen aus Modell in Abb. 2-3*

Sequenz 1	Sequenz 2	Sequenz 3
`Startseite öffnen`	`Startseite öffnen`	`Startseite öffnen`
`Auswahl Produkt`	`Auswahl Produkt`	`Auswahl Produkt`
`Warenkorb sehen`	`Warenkorb sehen`	`Warenkorb sehen`
`Login`	`Login`	`Auswahl Produkt`
`Kauf`	(Abbruch)	`Warenkorb sehen`
(Abschluss)		`Login`
		`Kauf`
		(Abschluss)

Hier werden aus Platzgründen nur drei abgeleitete Sequenzen gezeigt. Das ist nur eine kleine Auswahl: Selbst wenn man ganz bescheiden als minimales Abdeckungskriterium nur fordert, dass jede Wiederholung gar nicht oder einmal durchgeführt werden soll, sind hier noch einige Abläufe mehr enthalten. Der Generator würde sie uns alle erzeugen. Aus Testsicht sind sie auch alle wichtig.

Abdeckungskriterium

In anderen Fällen kann ein aufwendigeres Abdeckungskriterium angemessen sein, weil aufgrund eines höheren Risikos ein intensiverer Test notwendig ist. Dann werden aus demselben Modell deutlich mehr Abläufe als Vorlage für Testfälle abgeleitet.

Bis hier war noch kein Keyword-Driven Testing im Spiel. Aber drängt es sich nicht auf, wenn man auf die begrenzte Anzahl klar definierter Aktivitäten im Modell und dieselben Aktivitäten in den abgeleiteten Sequenzen sieht, hier Keywords einzusetzen? Natürlich muss man das nicht machen, aber es ist ein einfacher Schritt, zu jeder Aktion[3] ein Keyword zu definieren.

In der Motivation zu Model-Based Testing in Abschnitt 2.6.1 wurde die verbesserte Wartbarkeit beim Einsatz von Model-Based Testing gelobt, wenn sich das Modell – also, im Grunde die Spezifikation – ändert. In Abbildung 2-4 liegt uns ein geändertes Modell vor: Es wurde entschieden, dass Kunden die Wahl haben sollen, sich am Anfang, während oder am Ende des Prozesses einzuloggen.

Das Modell wurde also erweitert, und wir starten einen neuen Generierungslauf. Als Ergebnis werden bei gleichen Anweisungen an den Generator bezüglich der zu erzielenden Abdeckung mehr Sequenzen erzeugt. Das Bestechende dabei ist, dass nicht nur die Sequenzen erzeugt werden, die fehlen, sondern dass auch die, die auf der Vorversion basieren, unter Berücksichtigung der Änderungen neu erstellt werden.

Tabelle 2-4 zeigt die so aktualisierten Testfälle (wieder eine kleine Auswahl). Die Sequenzen 1 bis 3 sind die gleichen wie in Tabelle 2-3, jedoch ergänzt um die neue nun notwendige Aktion, `Login ablehnen`. Die Sequenz 4 ist neu und steht in dieser Tabelle als Vertreter der vielen möglichen Folgen von Aktionen für den Fall, dass Anwender sich für das Login am Beginn des Prozesses entscheiden. Solch eine Aktualisierung ist bei der automatischen Ableitung von Testfällen ganz einfach.

[3] In diesem Beispiel, dem Aktivitätsdiagramm, sind es Aktionen, bei anderen Modellarten ist das aber genauso möglich, denn letztlich werden immer Sequenzen abgeleitet.

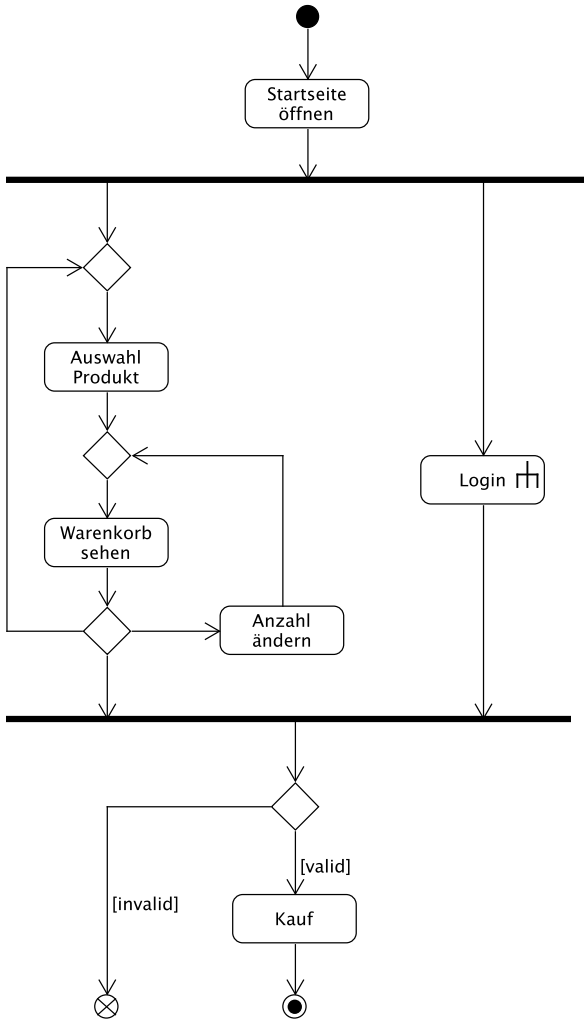

Abb. 2-4 *UML-Diagramm: Einkauf – Version 2*

Tabelle 2-4 *Testsequenzen aus Modell in Abb. 2-4*

Sequenz 1	Sequenz 2	Sequenz 3	Sequenz 4
`Startseite öffnen`	`Startseite öffnen`	`Startseite öffnen`	`Startseite öffnen`
`Login ablehnen`	`Login ablehnen`	`Login ablehnen`	`Login`
`Auswahl Produkt`	`Auswahl Produkt`	`Auswahl Produkt`	`Auswahl Produkt`
`Warenkorb sehen`	`Warenkorb sehen`	`Warenkorb sehen`	`Warenkorb sehen`
`Login`	`Login`	`Auswahl Produkt`	`Anzahl ändern`
`Kauf`	(Abbruch)	`Warenkorb sehen`	`Warenkorb sehen`
(Abschluss)		`Login`	`Kauf`
		`Kauf`	(Abschluss)
		(Abschluss)	

Beim manuellen Ableiten von Testfällen sieht das aller Erfahrung nach anders aus: Die neuen Testfälle zu erstellen, klappt noch recht zuverlässig. Zu erkennen, welche alten Testfälle wegen der Änderung anzupassen sind, ist jedoch nicht einfach, kostet Zeit und ist eine Quelle von Fehlern.

2.6.3 Von der Sequenz zur Testautomatisierung

Bis hierher ist alles schön und gut, mit Model-Based Testing bekommt man schöne Vorlagen von Testfällen generiert.

Sequenzen sind noch keine Testfälle

Bei den generierten Abläufen oder Sequenzen haben wir bewusst nicht von Testfällen gesprochen. Es sind eher Vorlagen von Testfällen, denn zu einem »richtigen« Testfall gehören wenigstens noch die Daten, die man auch zur Durchführung benötigt, und die erwarteten Ergebnisse, wenn beispielsweise der Inhalt eines erzeugten Kaufbelegs geprüft werden soll.

Mit einem geeigneten Model-Based Testing Framework lassen sich auch solche Daten ableiten und somit vollständige Testfälle generieren.

Je mehr Vorlagen das werden, umso wichtiger wird die Anbindung an eine Testautomatisierung. Mit Keyword-Driven Testing lässt sich das einfach umsetzen:

1. Zu jeder Aktion[4] wird ein Keyword angelegt und ein automatisiertes Testskript erzeugt, das das Keyword abbildet.
2. Wo Daten erforderlich sind – im Beispiel ist das für die Login-Daten, für die Produktauswahl und die Anzahl der zu kaufenden Kunstobjekte der Fall –, werden in zugeordneten Datentabellen (Stichwort Data-Driven Testing, siehe Abschnitt 2.3) geeignete Daten hinterlegt, die den Keywords über festgelegte Regeln zugeordnet werden (Beispiel: sequenziell aus der Tabelle oder randomisiert).
3. In einigen Fällen sind die Daten entscheidend für den weitere Ablauf. In unserem Fall trifft das für das Login zu, das für den Erfolgsfall gültige Daten benötigt. Im Modell ist das bereits durch Annotation an den jeweiligen Kanten (»valid«/»invalid«) vorbereitet. Der

[4] Gegebenenfalls auch zu ganzen Aktivitäten – ein Beispiel könnte »Login« sein, das in den Diagrammen hier als »call action« modelliert ist. Dahinter liegt also ein weiteres, hier nicht gezeigtes Aktivitätsdiagramm.

Generator muss diese Information beim Erzeugen der Sequenzen weitergeben.
4. Model-Based Testing Framework und Keyword-Driven Testing Framework müssen bezüglich der Datenformate aufeinander abgestimmt sein.

Automatisierung zum Quadrat

Gelingt es, diese Punkte umzusetzen, ist das Ergebnis ein Framework, in dem automatisiert erzeugte Testfälle automatisiert durchgeführt werden.

2.7 Organisatorische Randbedingungen

Bevor die eigentliche Anwendung des Keyword-Driven Testing beginnen kann, sollten die organisatorischen Randbedingungen geklärt und beschrieben sein. Betrachten Sie es wie eine Reise: Natürlich beginnt sie mit einem ersten Schritt, und den können Sie auch einfach so gehen. Auf Dauer sind Sie und Ihre Begleiter aber froh, wenn alles durchdacht wurde und die Verbindungen passen.

Als Ergebnis dieser Überlegungen entsteht ein Einsatzkonzept, in dem wir folgende Punkte erwarten:

- Festlegung der zu verwendenden Schichten und Ebenen (vgl. Abschnitt 3.1.5)
- Wird Keyword-Driven Testing manuell, automatisiert oder beides geplant?
- Basierend auf den letzten beiden Punkten und einer weiteren Bedarfsanalyse: Auswahl der Werkzeuge (eventuell mithilfe der ISO 29119-5, wie in Abschnitt 4.4 beschrieben, oder mit dem bei [24] erhältlichen Kriterienkatalog)
- Beschreibung der Verantwortlichkeiten und Prozesse: Wer darf Keywords definieren? Wer gibt sie frei? Welche Gültigkeitsbereiche haben sie? Überspannen Keywords beispielsweise Projekte? Dann gilt es, dies zu koordinieren. Gibt es ein Keyword-Review (vgl. Abschnitt 3.5)?
- Regeln für die Formulierung von Keywords (vgl. Abschnitt 3.3)
- Testebenen und Testarten, auf denen Keyword-Driven Testing einzusetzen ist
- Schulung der Mitarbeiter: intern oder möglicherweise durch Hersteller von Werkzeugen
- ... und weitere Punkte, die in *Ihrem* Umfeld wichtig sind

Bei dem entstehenden Dokument handelt es sich übrigens im Sinne der ISO 29119-3 [47] um eine Variante des »Test Plan«.

Das entstandene Einsatzkonzept sollte, wie jedes wegweisende Dokument, durch die Betroffenen und Verantwortlichen (auch als Stakeholder bekannt) geprüft und freigegeben werden und an einem solchen Ort abgelegt werden, dass es für die Nutznießer jederzeit verfügbar und vor allen Dingen wieder auffindbar ist. Hin und wieder sollte es ans Licht geholt und hinterfragt werden, ob das darin Beschriebene noch stimmt oder einer Überarbeitung bedarf.