1 Einführung

Dieses Buch hat das Ziel, das Thema »Keyword-Driven Testing« möglichst umfassend zu beleuchten, unter anderem, damit Keyword-Driven Testing häufiger und intensiver eingesetzt wird. Aber was verstehen wir überhaupt unter Keyword-Driven Testing?

Keyword-Driven Testing ist eine Methode zur Spezifikation von Testfällen im Sinne einer Notation. Es macht Testen effizienter. Es entfaltet seinen Nutzen vor allen Dingen in Teams, hilft aber auch einzelnen Testerinnen und Testern. Und entgegen anderslautender Gerüchte ist »Keyword-Driven Testing« nicht nur für Testautomatisierung hilfreich – aber gerade auch dort.

Das erwartet Sie nun in den einzelnen Kapiteln:

Überblick über das Buch

Einführung – dieses Kapitel: Was »Keyword-Driven Testing« ist, warum man es nutzen sollte, Begriffe und die Einführung eines Beispiels, auf das sich das Buch im weiteren Verlauf bezieht. *Ergebnis: Sie wissen, worum es geht.*

Konzepte: Unterschiedliche (einfache und komplexe) Ansätze, mit denen »Keyword-Driven Testing« genutzt werden kann. *Ergebnis: Sie wissen, was Keyword-Driven Testing ist, und kennen Zusammenhänge.*

Umsetzung: Wie man Keywords auswählt, wie man sie strukturieren kann, was es aus sprachlicher Sicht zu beachten gilt und wie man Qualitätssicherung für Keywords betreiben kann. Und was das wirtschaftlich bedeutet. *Ergebnis: Sie wissen, wie man mit Keywords umgeht.*

Keywords und Normen: Was für Normen es im Testen und speziell zu Keywords gibt und was die Normen zu Keywords und Frameworks sagen. *Ergebnis: Sie wissen, wie Ihnen Normen zu Keyword-Driven Testing helfen können.*

Testautomatisierungsarchitektur: Was so alles gebraucht wird, wenn man Testautomatisierung und Keyword-Driven Testing betreiben möchten, und auch wie ISTQB®dazu steht. *Ergebnis: Sie kennen verschiedene Sichtweisen auf die Testautomatisierungsarchitektur und verfügen über die Grundlagen, sich Ihre Architektur im Hinblick auf Keyword-Driven Testing zu erarbeiten.*

Keyword-Driven Testing Frameworks: Praxisbeispiele, wie Keyword-Driven Testing in Projekten angewendet wird, einschließlich einer Bewertung der eingesetzten Frameworks. *Ergebnis: Sie haben einen Eindruck von unterschiedlichen Einsatzszenarien von Keyword-Driven Testing.*

Praxis mit Robot Framework: Anhand eines konkreten Frameworks zeigen wir, wie eine Umsetzung von Keyword-Driven Testing im Kontext aussehen kann. *Ergebnis: Sie können die Anwendung von Keyword-Driven Testing mit diesem Framework nachvollziehen.*

Brückenschlag: Verbindung von Keyword-Driven Testing mit verbreiteten Ansätzen im Softwaretest, die zwar nicht »Keyword-Driven Testing« im engeren Sinne sind, aber gut damit zusammenwirken. *Ergebnis: Sie können Keyword-Driven Testing im Hinblick auf diese Ansätze einordnen.*

Ausblick: Wie geht es weiter mit Keyword-Driven Testing. *Ergebnis: Sie haben das Buch geschafft und wissen nun, wie Sie mit Keyword-Driven Testing umgehen wollen!*

1.1 Wortwahl

Bevor wir mit dem Inhaltlichen beginnen, gibt es noch zwei Dinge bezüglich der Wortwahl, die uns Autoren am Herzen liegen und die wir daher hier kurz darlegen wollen:

Geschlechtsspezifische Begriffe: In diesem Buch werden wir häufig darüber reden, was eine Person – z.B. ein Tester oder eine Testerin – tut, denkt, tun kann ... Wir sind der festen Überzeugung, dass das Geschlecht einer Person grundsätzlich keine Rolle zu spielen hat. Dennoch leben wir im Bereich der Softwareentwicklung und des Softwaretests in einer stark männerdominierten Branche. Gerade deswegen ist uns eine Geschlechtergleichstellung auch besonders in sprachlicher Form wichtig.

Zugunsten einer besseren Lesbarkeit werden wir nicht jedes Mal beide sprachlichen Geschlechter aufzählen. Auch haben wir uns gegen eine Schreibweise wie »TesterIn« oder »Tester*in« entschieden.

Das generisches Maskulinum, also nur die männliche Form, zu verwenden, schied völlig aus.

Unser Weg ist es nun, für die in diesem Buch wichtigsten sechs Rollen uns jeweils zu gleichen Teilen für die weibliche und die männliche Form zu entscheiden und dann zugunsten der Lesbarkeit dabei zu bleiben – in der Hoffnung, dass jede und jeder sich so ausreichend angesprochen und wertgeschätzt fühlt.

Wir reden also von Testerinnen, Testmanagerinnen und Entwicklerinnen sowie von Testdesignern, Anwendern und Testautomatisierern.

Englisch – Deutsch: Dies ist ein deutschsprachiges Buch und im Allgemeinen streben wir an, Anglizismen zu vermeiden. Ausgerechnet für den Schlüsselbegriff dieses Buches – »Keyword-Driven Testing« – gibt es zwar mit »schlüsselwortbasiertem Test« eine deutsche Entsprechung, diese ist aber nach unserem Empfinden so wenig gebräuchlich, dass wir uns entschieden haben, innerhalb des Buches dennoch den englischen Begriff »Keyword-Driven Testing« zu bevorzugen. Gleiches gilt für die Verwendung von »Keyword« und »Schlüsselwort«. Die Germanisten mögen uns verzeihen.

1.2 Was ist Keyword-Driven Testing

Was verbirgt sich nun hinter dem Begriff »Keyword-Driven Testing«?

Es gibt dazu einige Analogien – ein Favorit: Testfälle werden aus Bausteinen zusammengesetzt. Wir alle erinnern uns gerne an unsere Kindheit. Die Vorstellung, Testfälle spielerisch zusammenzusetzen, ist daher bestimmt attraktiv.

Modularisierung

Große Dinge aus kleinen Bausteinen zusammenzusetzen ist aber jenseits vom spielerischen Aspekt eine generell bewährte Vorgehensweise, um Komplexität zu bewältigen. Wir sprechen hier von Modularisierung.

Während Modularisierung in der Softwareentwicklung im Allgemeinen selbstverständlich ist, und das vielleicht auch noch bei Testautomatisierung zutrifft, so ist sie im Zusammenhang mit Testspezifikation bei Weitem nicht so häufig anzutreffen.

Keyword-Driven Testing (KDT) ist eine Methode, um Modularisierung bei der Spezifikation von Testfällen zu erreichen.

Ein Keyword ist in diesem Sinne ein Baustein, aus dem Testfälle zusammengesetzt werden. Dabei steht das Keyword – typischerweise ein Verb zusammen mit einem Substantiv, beispielsweise `Erzeuge User` – für eine oder mehrere Aktivitäten, die bei der Testausführung später auszuführen sind.

Idealerweise stehen beim Erstellen der Testfälle alle benötigten Keywords fertig definiert zur Verfügung. Dann können alle Testfälle aus Keywords zusammengesetzt werden.

Keine Testtechnik

Unter »Testtechnik« versteht man meist »Testentwurfsverfahren«. Laut ISO/IEC 29119 [48] geht es dabei darum, auf einem definierten Weg Testfälle zu ermitteln (einem Prozess folgend).

Keyword-Driven Testing ist also keine Testtechnik in diesem Sinne. Unabhängig von Keyword-Driven Testing setzen wir für das Ableiten von Testfällen, also das Testdesign, voraus, dass eine Testtechnik (oder mehrere) eingesetzt werden.

Keyword-Driven Testing legt jedoch fest, wie die aus dem Testdesign entstandenen Testfälle aufgeschrieben oder dokumentiert werden.

1.3 Begriffe

Nur die beiden wichtigsten zentralen Begriffe – Keyword und Framework –, die in diesem Buch verwendet werden, sollen hier definiert und erläutert werden. Umfassendere Begriffssammlungen finden sich auf der Webseite von imbus [23] sowie auf der Seite des ISTQB® [32] (als Online-Anwendung), und von IEEE sind die Begriffe von IEEE, ISO, IEC und PMI veröffentlicht und als Norm ISO/IEC/IEEE 2765 [50] käuflich zu erwerben, aber auch im Web legal frei verfügbar [28].

1.3.1 Der Begriff »Keyword«

Der zentrale Begriff dieses Buches ist ohne Zweifel »Keyword« oder zu Deutsch »Schlüsselwort«.

Mehrdeutigkeit

Der Begriff »Keyword« selbst ist eigentlich ein Teekesselchen[1]: Man versteht darunter, je nach Zusammenhang, die Bezeichnung (im Beispiel oben »Erzeuge User«), die dahinter liegenden Aktivitäten (könnte hier das Auswählen eines Menüeintrags sein, dann das Ausfüllen der Felder eines Dialoges ...) oder ein vielleicht vorhandenes zugeordnetes Automatisierungsskript. Oder dies alles gemeinsam.

[1] Ein »Teekesselchen« – das Fachwort wäre Homonym – ist ein Begriff, der mehrere Bedeutungen hat. In einem Kinderspiel lässt man raten: »Mein Teekesselchen kann schwimmen oder man braucht es zum Zelten – was ist das?« – Antwort: ein Hering.

In der Norm ISO 29119-5 [49] ist dieser Begriff folgendermaßen definiert:

Keyword	Schlüsselwort
one or more words used as a reference to a specific set of actions intended to be performed during the execution of one or more test cases	ein oder mehrere Wörter, die als Verweis auf eine festgelegte Menge von Aktivitäten verwendet werden und die während der Durchführung eines oder mehrerer Testfälle ausgeführt werden sollen
(Definition laut [49])	(Übersetzung der Autoren)

Ein Keyword besteht demnach aus ein oder mehreren Wörtern, die beschreiben, was an der Stelle, an der diese Wörter in einem Testfall vorkommen, getan werden soll. Ein Beispiel für ein einzelnes Wort könnte `Login` sein, eines für mehrere Wörter `Datensatz anlegen`. Meist wird ein Keyword ein Verb enthalten (so fordert es auch die ISO 29119-5), allein schon deswegen, weil ja durch das Keyword eine auszuführende Tätigkeit beschrieben ist.

Auch wenn es rein technisch gesehen nicht notwendig ist, Keywords mit einem Verb zu bilden, so hat es sich doch bewährt. Die Lesbarkeit profitiert davon, und daher ist es empfehlenswert, sich nach dieser Regel zu richten.

Um zu der Begriffsfrage zurückzukehren: Testautomatisierer verstehen unter einem »Keyword« gerne eine aufrufbare Funktion oder ein Skript, die bzw. das im Sourcecode in einem Testautomatisierungswerkzeug implementiert ist. Testerinnen denken dagegen vielleicht eher an eine Zusammenfassung mehrerer Testschritte zu einer fachlichen Tätigkeit.

Ein Keyword kann beides sein.

Ein Keyword kann eine automatisierte Funktion widerspiegeln oder manuelle Testschritte zusammenfassen. Und: Ein Keyword kann sogar eine Zusammenfassung anderer Keywords sein.

Häufig wird unter dem Begriff »Keyword« einfach der Name des Keywords (hier: `Benutzer anmelden`) verstanden – der ja im Alltag, beim Spezifizieren von Tests, auch am meisten gebraucht wird. Im Hinterkopf sollte man aber behalten, dass dazu einiges mehr gehört.

In Abschnitt 2.1 »Verschlagwortung« werden wir auf die sprachlichen Aspekte, die hier nur angerissen wurden, noch sehr viel genauer eingehen.

1.3.2 Der Begriff »Framework«

Framework

Den Begriff »Framework« benutzen wir in diesem Buch bezogen auf Keyword-Driven Testing. Alleine stehend ist er ein zu gängiges Wort der natürlichen englischen Sprache, als dass es einer ISO oder IEEE in den Sinn kommen würde, dafür eine Definition anzugeben. Im Zusammenhang mit Testautomatisierung bietet jedoch das ISTQB®eine Definition, dankenswerterweise sowohl auf englisch als auch auf deutsch:

Test automation framework	*Testautomatisierungsframework*
A tool that provides an environment for test automation. It usually includes a test harness and test libraries.	Ein Werkzeug, das eine Umgebung zur Testautomatisierung bereitstellt. Es beinhaltet üblicherweise einen Testrahmen und Testbibliotheken.
(Definition laut [32] – englisch)	(Definition laut [32] – deutsch)

Auch in diesem Buch steht der Begriff »Framework« im Zusammenhang mit Testautomatisierung. Allerdings ist der Kontext noch etwas spezifischer. Unter »Keyword-Driven Testing Framework« verstehen wir hier die Gesamtheit der Werkzeuge, Skripte und Bibliotheken, die als Grundlagen für eine bestimmte Variante von Keyword-Driven Testing – meist mit Fokus auf Automatisierung – benötigt und verwendet werden.

Mehr zum Thema Framework, was alles dazugehört und was man davon erwarten kann, findet sich in Abschnitt 4.3 (Sichtweise der Norm ISO 29119-5) und vor allem in Kapitel 6 »Keyword-Driven Testing Frameworks«.

1.4 Keywords unter der Lupe

Elementare Eigenschaften von Keywords

Um uns Keyword-Driven Testing weiter anzunähern, wollen wir uns in den folgenden Abschnitten Keywords genauer ansehen. Erst einmal geht es um die elementaren Eigenschaften – was ist ein Keyword, was gehört auf jeden Fall dazu? Und dann geht es um einen ganz praktischen, wichtigen Aspekt, nämlich die Struktur, die ein Keyword haben kann.

Verschlagwortung: Eines der Grundkonzepte hinter Keyword-Driven Testing ist, dass Tests nicht als Prosatext verfasst werden, sondern in einzelne, klar voneinander abgegrenzte, sehr kurze Testschritte gegliedert werden. Jeder dieser Testschritte soll keine komplette Beschreibung der Tätigkeit sein, sondern nur ein kurzer Befehl, der

die entsprechende Tätigkeit zusammenfasst. Daher rührt der Begriff »Keyword«. Meist reicht es nicht aus, sich auf ein einzelnes Wort als Benennung für ein Keyword zu beschränken. Mehrere Wörter sind also in Ordnung. Als Faustformel für die Länge der Keywords kann man von 1-6 Wörtern ausgehen.

1 Keyword ≈ 1-6 Wörter

Kurze Anweisungen von bis zu sechs Wörtern können von einem Leser schnell erfasst und verstanden werden. Oft reicht einer Testerin ein einziger Blick auf eine solche Anweisung, um zu wissen, was zu tun ist.

Einige Beispiele für Keywords:

```
Webbrowser starten
Benutzerverwaltung öffnen
Benutzer an Windows anmelden
Device über WLAN verbinden
Mobilgerät entsperren
```

Beschreibung: Keyword-Driven Testing würde für manuelles Testen nicht funktionieren, wenn man sich ausschließlich auf die sprechenden Namen verlassen würde. Bestenfalls könnte man dann erwarten, dass mit den Keywords sehr vertraute Testerinnen sie mühelos und zuverlässig ausführen.

Neue Kolleginnen oder Testerinnen, die die Testspezifikation nur auszuführen haben, aber die Schritte (in Form von Keywords) vorher nicht kennen, könnten nicht wissen, was der Testdesigner von ihnen genau erwartet, und die Tests demnach auch nicht präzise ausführen. Zu der Definition eines Keywords gehört daher obligatorisch auch ein beschreibender Text.

Auch im Hinblick auf Automatisierung ist man sehr gut beraten, eine Dokumentation für das Keyword zu erfassen. Schließlich stellen Keywords hier die Schnittstelle zwischen fachlicher Sicht und technischer Sicht dar. Testautomatisierer müssen genau wissen, was sie implementieren sollen.

Bei der Keyword-Dokumentation geht es nicht nur darum, zu beschreiben, was das Keyword tun soll, sondern auch, wie es zu verwenden ist.

Parameter: Eine Interaktion zwischen einer Testerin und dem Testobjekt ist oft von Daten getrieben oder wird anhand von Daten verändert. Daher können Keywords in ihrer Verwendung ebenfalls über Testdaten variiert werden.

Die Benutzeranmeldung an einem System wird beispielsweise immer auf die gleiche Art und Weise durchgeführt. Je nach Benutzerdaten, die für die Anmeldung verwendet werden, unterscheidet sich das Verhalten des Testobjektes.

Diese Daten eines Keywords werden üblicherweise als Parameter oder Argumente bezeichnet. Parameter sind typischerweise im Keyword-Driven Testing dem eigentlichen Keyword angehängt.

Ob dieses Anhängen bzw. Trennen vom Keyword selbst wie im folgenden Beispiel mit zwei oder mehr Leerstellen oder in Tabellen mit mehreren Spalten oder wie in der Programmierung üblich in Klammern erfolgt, ist je nach Tool unterschiedlich und spielt keine Rolle.

Im Folgenden sehen wir ein Beispiel mit einigen Parametern:

```
Webbrowser starten        www.keyword-driven.de
Benutzername eingeben     admin
Passwort eingeben         123456
Anmelden klicken
Benutzerverwaltung öffnen
Device mit WLAN verbinden     Phone2     GuestWifi
Mobilgerät entsperren
```

Mehr zu diesem Thema findet sich in Abschnitt 2.3 »Data-Driven Testing«.

Keyword-Struktur: Ein wichtiger Aspekt bei Keyword-Driven Testing ist, dass Keywords »strukturiert« sein können. Wir meinen damit, dass Keywords aus kleineren Keywords zusammengebaut sein können – oder andersherum: Keywords können zu größeren Keywords zusammengefügt werden. Diese Eigenschaft von Keywords macht man sich zunutze, um Wiederverwendbarkeit und Wartbarkeit zu steigern.

Ein Beispiel: Die Anmeldung eines Benutzers am System besteht aus der Eingabe von Benutzername und Passwort und dem Klicken des »Anmelden«-Knopfes. Nehmen wir also an, dass `Benutzer anmelden` ein strukturiertes Keyword ist und aus drei Einzelschritten besteht:

```
Benutzer anmelden    Benutzer    Passwort
    Benutzername eingeben    Benutzer
    Passwort eingeben        Passwort
    Anmelden klicken
```

Ein Testdesigner braucht nun bei einem Test, der die Anmeldung benötigt, nicht immer wieder die drei Einzelschritte zu nutzen, sondern erfordert nur das eine strukturierte Keyword.

Das Thema der Struktur von Keywords greifen wir in Abschnitt 2.2 im Detail auf.

1.5 Evolution der Testautomatisierung

In einem Vortrag über Testautomatisierungsarchitekturen im Rahmen der Veranstaltung »Trends in Testing« im Jahr 2010 [13] wurde eine Evolution von Architekturen zur Testautomatisierung beschrieben.

In dieser »Evolution« hat auch Keyword-Driven Testing seinen Platz.

Vorsicht Glatteis ...

Keyword-Driven Testing wird von manchen als Königsweg der Testautomatisierung betrachtet.

An dieser Stelle über Testautomatisierungsarchitekturen zu sprechen könnte den Eindruck erwecken, dass es in diesem Buch auch so gesehen wird. Daher zur Klarstellung:

- Keyword-Driven Testing ist wunderbar geeignet für Testautomatisierung, aber:
- Keyword-Driven Testing hat auch viele Vorteile bei manuellem Test!

Eine Beschränkung des Keyword-Driven Testing auf Testautomatisierung wäre Verschwendung.

Abbildung 1-1 bringt zum Ausdruck, wie mit der Entwicklung der Testautomatisierungsarchitekturen, von Capture & Replay über datengetriebenen Test und Keyword-Driven Testing bis hin zu generierten Tests, ein Reifegrad verbunden wird.

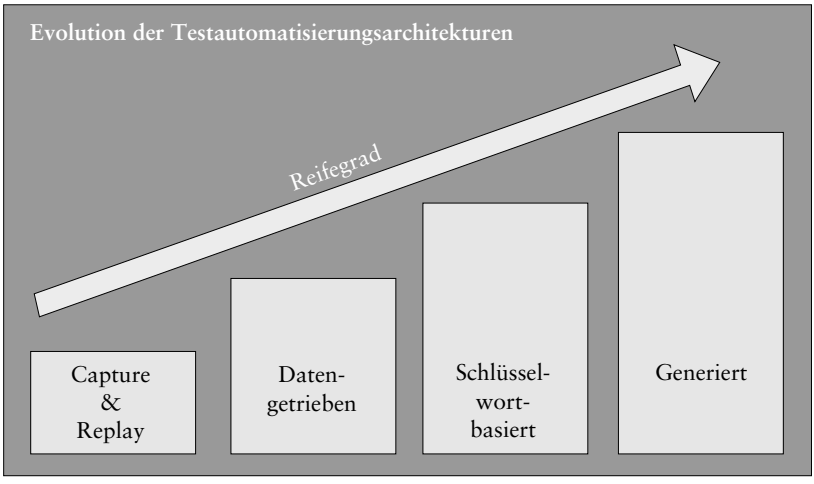

Abb. 1-1 *Evolution der Test-automatisierungs-architekturen*

Nach der Einführung von Testautomatisierung in einer Organisation entwickeln sich durch die sukzessive Gewinnung von Erfahrungen der Testerinnen sowohl die Ansprüche an die Umsetzung der Testautomatisierung als auch die Leistungsfähigkeit der Architektur typischerweise in folgenden Phasen – immer vorausgesetzt, externe Einflüsse sorgen nicht dafür, dass diese Phasen direkt übersprungen werden:

Wartungsproblem

1. Capture & Replay: Automatisierte Testfälle werden als Skripte aufgezeichnet, dafür wird eine entsprechende »Rekorder«-Funktion des Werkzeugs genutzt. Das Resultat sind automatisierte Testskripte, die einfach strukturiert sind, in der Regel mit der aktuellen Version des Testobjektes funktionieren (aber schon bei kleinen Änderungen am Test Interface Probleme bekommen können), die untereinander redundant sind (gleiche Aktionen werden in jedem Skript erneut erfasst) und, wie sich meist schnell herausstellt, einfach ein Albtraum im Hinblick auf Wartbarkeit darstellen.

   Abbildung 1-2 veranschaulicht einen solchen Fall: Eine Testspezifikation mag vorhanden sein und enthält sowohl Testlogik als auch Daten. Die Verknüpfung zur Testautomatisierung ist nur lose. Ein Testskript liegt individuell für jeden Testfall vor und wird aus einem kaum trennbaren Mix aus Code, Logik und Daten gebildet. Eine Änderung, egal aus welchem Grund, betrifft immer alles, und Fehler passieren leicht.

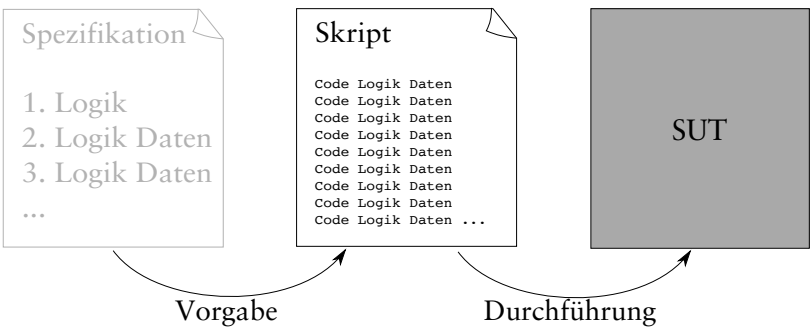

Abb. 1-2 *Monolithisches Testskript*

   Also sucht man schnell nach einem besseren Weg.

2. Datengetriebener Test: Während man durch Abkehr vom reinen Aufzeichnen der automatisierten Testskripte schon erste Vorteile in Richtung besserer Wartbarkeit erzielt, kommt die Erkenntnis, dass die Mühe, die man mit der Wartung hat, auch zwischen Testfällen geteilt werden kann. Mehrere strukturell identische Testfälle werden durch ein einziges, verallgemeinertes Skript implementiert und die Unterschiede zwischen den Testfällen, die in den Daten liegen, in

Trennung von Code und Daten

von den Testskripten getrennten Datentabellen ausgelagert (mehr dazu in Abschnitt 2.3).

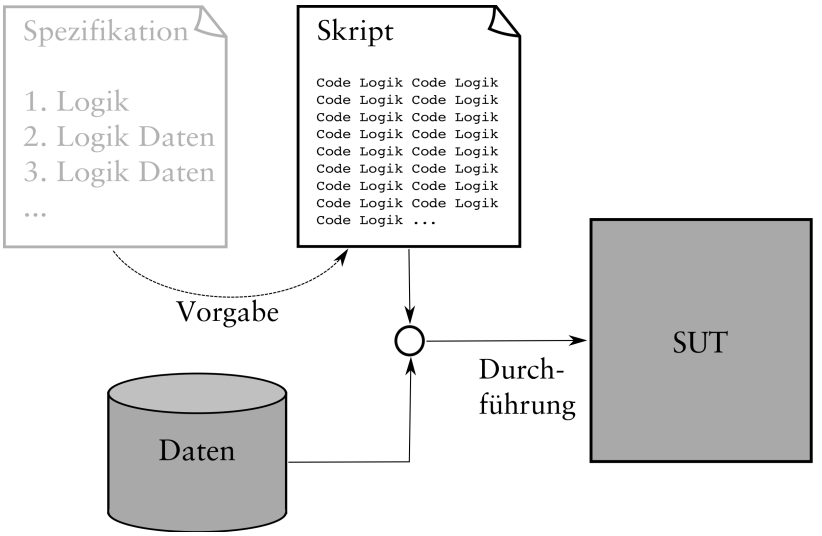

Abb. 1-3
Trennung von Skript und Daten

In Abbildung 1-3 sind strukturelle Vorteile dieses Ansatzes gegenüber den monolithischen Skripten aus Abbildung 1-2 zu erkennen: Die Kopplung zur Spezifikation ist noch lose (Anmerkung: Daten könnten an dieser Stelle bereits aus der Spezifikation ausgelagert sein), das Chaos in der Testimplementierung ist aber schon geringer, denn es sind nur noch Code und Logik im Skript miteinander vermischt. Änderungen, die nur die Daten betreffen, können unabhängig vom Rest durchgeführt werden, und das kann in vielen Fällen die Änderung sein, die benötigt wird.

Mehr noch: Es muss für eine Anzahl von fünf, zehn oder hundert Testfällen nur noch ein einziges automatisiertes Skript gewartet werden – ein großer Schritt in Richtung eines besseren Kosten-Nutzen-Verhältnisses.

3. Keyword-Driven Testing: Das, was mit datengetriebenem Test begonnen wurde, nämlich die Trennung von Code und Daten, wird beim Keyword-Driven Testing konsequent weitergeführt: Die bisher im Code verankerte Testlogik (der abstrakte Testfall) wird nun getrennt vom (Automatisierungs-)Code abgelegt.

Trennung von Logik und Code

Abbildung 1-4 zeigt, dass sich damit massive Änderungen (im Sinne von Verbesserungen) ergeben: Die Spezifikation ist ins Zentrum gerückt und enthält die reine Testlogik. Zur Spezifikation werden Keywords genutzt, denen in einem Repository entsprechende Skripte zugeordnet sind. Hier liegt also der Code. Die drei Aspekte Logik, Code und Daten werden zur Durchführung zusammengeführt.

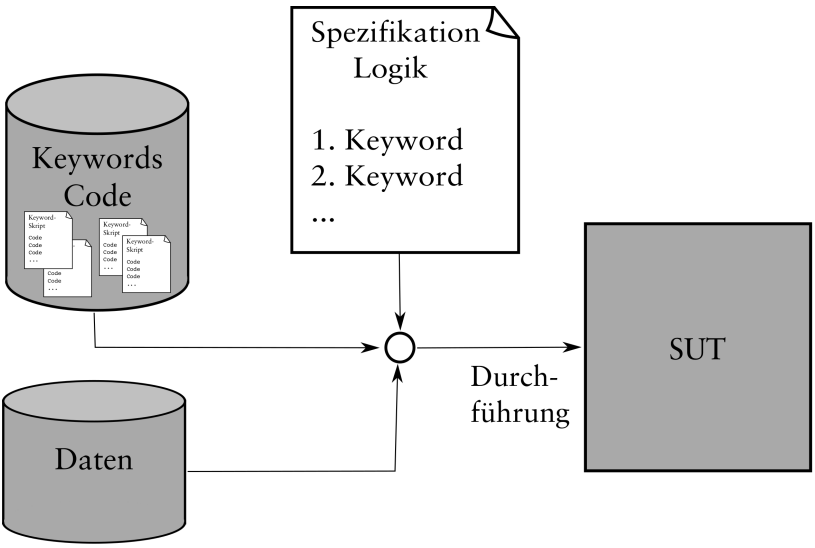

Abb. 1-4 *Trennung von Skript, Daten und Logik*

Jegliche Wartungstätigkeiten können jetzt unabhängig voneinander an Logik, Code und Daten erfolgen. Braucht man eines davon nicht zu ändern, dann muss man es auch nicht anfassen.

> Was hier passiert, die Trennung von Code und Daten, ist ein uraltes Prinzip der Informatik[2] und in der Softwareentwicklung generell ein alter Hut.
>
> Aber zu der Erkenntnis, dass die Entwicklung von Testautomatisierung die gleichen Prinzipien wie Softwareentwicklung erfordert, muss man ja erst gelangen – zunächst sieht alles so einfach aus.

4. Generierte Testfälle: Diese Entwicklungsstufe wird nach heutigem Stand nicht allerorts erreicht und muss auch nicht das Ziel sein. Hier geht es um Model-Based Testing (modellbasierten Test), also darum, Testfälle aus formal beschriebenen Modellen mechanisch abzuleiten. Keywords können dafür eine Grundlage sein.

Modellbasierter Test

Was mit diesem Ansatz vorangetrieben wird, ist der Grad der Automatisierung, die nicht mehr nur die Durchführung der Tests,

[2] Und nicht nur dort: In der Rechnerarchitektur ist dieses Prinzip unter dem Begriff »Harvard-Architektur« schon seit dem Mark II bekannt – wenn auch hier eher aus Gründen der Performanz als aus Gründen der Strukturierung und Wiederverwendung.

sondern auch die Erstellung der Testfälle beinhaltet. Die Wartung verlagert sich weg von der Automatisierung und hin zu den Modellen.

Auch wenn Keyword-Driven Testing nach diesem Modell nicht die Spitze der Evolution darstellt – es sieht doch so aus, als wäre im Zuge der Entwicklung von Praktiken zur Testautomatisierung KDT unausweichlich. Aber mehr noch: KDT passt auch ausgezeichnet zu Model-Based Testing; das greifen wir später in Abschnitt 2.6 wieder auf.

1.6 Vorteile des Keyword-Driven Testing

Mit dem bisher Gesagten haben wir eine Vorstellung davon vermittelt, was Keyword-Driven Testing ist, aber uns nicht weiter damit beschäftigt, warum es eingesetzt werden sollte. Immerhin liegt es auf der Hand, dass Keyword-Driven Testing mit einem gewissen Aufwand verbunden ist – zumindest müssen Keywords ja definiert und dokumentiert werden. Das Konzept und den damit verbundenen Aufwand können wir nicht alleine mit einem angeblich höheren Reifegrad rechtfertigen.

Im Folgenden werden wir also beleuchten, was die tatsächlichen Vorteile beim Einsatz von Keyword-Driven Testing sind.

1.6.1 Klarheit

Das Formulieren von Testfällen aus vorgefertigten Bausteinen führt auf Dauer zu mehr Klarheit, und zwar bei der Interpretation und dem Verständnis dessen, was später bei der Ausführung der Testfälle zu tun ist.

Warum ist das so?

Jede Testerin muss beim Lesen einer Testspezifikation entscheiden, was mit einer bestimmten Formulierung gemeint ist (wir unterstellen, dass sie die Testspezifikation nicht selbst geschrieben hat). Wenn die Testspezifikation herkömmlich, also in Prosa verfasst ist, so sind die Formulierungen nicht einheitlich. Derselbe Sachverhalt kann mit verschiedenen Worten beschrieben sein. Ein Beispiel: Es wird einmal der Begriff »Anmelden« und das nächste Mal der Begriff »Einloggen« verwendet. Durchführende Testerinnen müssen an dieser Stelle entscheiden, ob diese Begriffe dasselbe bedeuten - wenn ja, was, und wenn nein: Was ist der Unterschied?

Keywords verhindern Ambivalenz.

Dieselbe Interpretationsleistung muss auch bei Verwendung von Keywords erbracht werden, aber eben nur einmal. Durch die Vereinheitlichung der Begriffe und das Bausteinprinzip gibt es nur ein Keyword für diese Aktivität. Am Anfang muss man einmal verstehen, was damit

Lerneffekt

gemeint ist – aber hier findet ein Lernprozess statt und bald kennt man die Bedeutung.

Dieses Beispiel für Vorteile des Keyword-Driven Testing greift insbesondere beim manuellen Test. Also Schluss mit dem Mythos, dass Keyword-Driven Testing nur ein Thema für Testautomatisierer ist!

Klar ist: Das hier beschriebene Mehr an Klarheit wird insbesondere dann seinen Nutzen entfalten, wenn in Teams gearbeitet wird. Aber eine verbesserte Lesbarkeit ist mit Sicherheit ein Gewinn.

1.6.2 Wiederverwendbarkeit

Mehr oder weniger selbsterklärend ist die Tatsache, dass Keywords wiederverwendet werden können. Das ist natürlich bei fast allen Testautomaten in Bezug auf die automatisierten Testschritte der Fall. Jedoch können wir diese Funktionalität bei kaum einem Werkzeug für manuelles Testen oder Testmanagement finden. Hier wird in aller Regel mit rein textuellen nicht wiederverwendbaren Testschritten gearbeitet.

Die Wiederverwendbarkeit steht in direktem Zusammenhang mit dem nächsten Abschnitt über Wartbarkeit. Aber es geht nicht nur darum, sondern auch um den ersparten Aufwand bei der Erstellung von Tests und um den Vorteil bezüglich der Klarheit und Lesbarkeit.

Aufwand sparen

Ein einmalig erstelltes Keyword, das beschreibt oder implementiert, wie unser Testobjekt zu starten ist, spart bei jeder Nutzung Zeit, verglichen mit einem immer wieder erneuten Beschreiben dieser Tätigkeit.

Wiederverwendbarkeit reduziert repetitive Arbeiten also auf ein Minimum und beschleunigt das Arbeiten entsprechend.

1.6.3 Wartbarkeit

Eine verbesserte Wartbarkeit ist wahrscheinlich der am häufigsten beschworene Vorteil von Keyword-Driven Testing.

Er ergibt sich auch bei manuellem Test, ist aber besonders im Zusammenhang mit Testautomatisierung wichtig, da der Nutzen von Testautomatisierung[3] von hohen Wartungsaufwänden zunichtegemacht werden kann.

Wartbarkeit ist ein Qualitätsmerkmal, das beschreibt, mit wie kleinem oder großem Aufwand Änderungen durchgeführt werden können. Ob Änderungen an unseren Testfällen notwendig sind, können wir uns

Auslöser für Wartung

[3] Der wirtschaftliche Nutzen wird dadurch festgestellt, dass die Kostenersparnis auf Dauer die Erstellungs- und Wartungskosten aufwiegt. Es gibt noch andere Nutzenaspekte, wie Motivation, Geschwindigkeit etc.

nicht aussuchen. Die Hoffnung sagt: Es wird schon nicht so schlimm. Die Erfahrung zeigt: Änderungen werden unweigerlich kommen.

Die Änderungen können sich, je nach Ursache, auf verschiedene Aspekte beziehen:

Abstrakter Testfall: Eine Änderung der Testlogik ist dann nötig, wenn der Testfall falsch war oder sich die zugrunde liegende Anforderung geändert hat. In diesem Fall muss inhaltlich eine Änderung an der Testlogik vorgenommen werden.

Konkreter Testfall: Eine Änderung der Testdaten kann aus den gleichen Gründen notwendig sein, zusätzlich aber auch dann, wenn bei gleichem abstraktem Testfall ein weiterer konkreter Testfall hinzugefügt werden soll oder eine Änderung der Testumgebung geänderte Testdaten erfordert.

Funktionale Veränderung: Eine Änderung oder das Hinzufügen einer Funktionalität kann bedeuten, dass ein bestehender Test im fachlichen Ablauf identisch ist und von der Testlogik her unverändert bleiben kann, obgleich die Benutzerführung sich ändert.

Ein Beispiel: Stellen wir uns vor, als Neuerung würde nun bei Kundenanlage auf eine Datenschutzerklärung hingewiesen werden. Ein vorhandener abstrakter Testfall kann gleich bleiben, nur das Keyword `Neukunde anlegen` wird technisch etwas angepasst. Anmerkung: Wir sprechen hierbei nicht von dem vermutlich benötigten neuen Testfall, der die neue Datenschutzerklärung prüft!

Technologische Veränderung: Eine Änderung der Implementierung wird insbesondere dann erforderlich, wenn sich die Testschnittstelle ändert. Im Fall einer Web-GUI (Graphical User Interface) könnte ein spezifisches Element an einem anderen Ort liegen und aufgrund dessen einen anderen Selektor benötigen, um das Element eindeutig zu identifizieren. Dazu könnte ein einzelnes technisches Keyword verändert werden und alle fachlichen Keywords, die es verwenden, und die Tests, die dieses nutzen, wären wieder lauffähig.

Durch eine konsequente Trennung von Testlogik (Sequenz der Keywords) und Testdaten sowie eine Trennung von Spezifikation und Implementierung in einem Automatisierungswerkzeug können diese Änderungen unabhängig voneinander vorgenommen werden. Das macht die Änderungen überschaubar und damit weniger anfällig dafür, Seiteneffekte und Fehler auszulösen.

Umgekehrt muss im Fall einer monolithischen Implementierung von automatisierten Testfällen ohne Keywords und ohne Trennung von Daten und Logik (Worst Case) bei *jeder* Änderung, egal ob diese die Testlogik, Daten oder Implementierung betrifft, mehrere Testskripte

angepasst werden. Somit würde der Wartungsaufwand direkt proportional mit der Menge von Testfällen skalieren. Dies führt, wie in Abschnitt 3.8 noch gezeigt wird, zu dem Fall, dass ab einem gewissen Wartungsaufwand, beispielsweise einem Technologiewechsel, das Verwerfen der gesamten Tests und eine komplette Neuimplementierung günstiger ist als die Anpassung der Skripte.

Diese verbesserte Wartbarkeit bei Keyword-Driven Testing sorgt also dafür, dass Änderungen und der Wartungsaufwand nicht mit der Anzahl der Tests, sondern lediglich mit der (viel geringeren) Anzahl der implementierten Keywords oder Funktionen steigt.

Häufig ist der steigende Wartungsaufwand die natürliche Grenze dafür, wie viel automatisiert werden kann. Werden die zuständigen Schichten nicht gut designet, ist das Team ab einer gewissen Menge Tests fast vollständig mit Änderungen vorhandener Tests beschäftigt und hat keine Zeit mehr, neue Tests zu schreiben.

1.6.4 Kommunikation

Die Verwendung von Keywords zwingt zu einem gewissen Maß an geordneter Kommunikation, wenn andere Vorteile des Keyword-Driven Testing – insbesondere alles, was mit der Wiederverwendbarkeit der Keywords zusammenhängt – erreicht werden sollen. Hier ist es, wie in vielen anderen Bereichen: Klare Zuständigkeiten und Kommunikationsschnittstellen zwingen zu Disziplin. Nichteinhaltung der Kommunikationsschnittstellen andererseits führt zum Scheitern.

Ohne diese klaren Zuständigkeiten könnte im Test jede einzelne Testerin, überspitzt gesagt, tun, was sie will. Ganz für sich alleine und ohne wesentliche Kommunikation mit den anderen Kollegen entsteht Chaos. Das ist sicher einem effizienten Arbeiten nicht zuträglich. Gerade die Wiederverwendbarkeit würde immens leiden unter einem schlecht funktionierenden Team und fehlender Abstimmung.

Abgesehen davon, dass Keyword-Driven Testing alleine schon durch eine Abstimmung über die Keywords das Team zur Kommunikation zwingt und so die Zusammenarbeit fördert, erleichtert es die Kommunikation aber auch.

Nun könnte man argumentieren, dass das ja gerade kein Vorteil, sondern ein Nachteil an Keyword-Driven Testing sei. Es benötigt eine gute Kommunikation zwischen den Beteiligten, sonst scheitert es.

In gewisser Weise müssen wir hier zustimmen, aber: Wenn wir uns auf gleiche Formulierungen einigen, wenn wir von denselben Dingen in derselben Sprache sprechen und wenn alle Prozessbeteiligten sowohl die Tests als auch die Keywords lesen und verstehen können, wächst

automatisch das gemeinsame Verständnis und das Vertrauen bezogen auf den Testprozess.

Und dieser Vorteil ist in der Praxis so enorm wichtig, dass mittelfristig die Beteiligten diese gehobenen Anforderungen keineswegs mehr als Last empfinden.

Die bereits in Abschnitt 1.6.1 beschriebene verbesserte Klarheit der Testfälle wirkt sich nicht nur im Hinblick auf eine beschleunigte und fehlerärmere Ausführung der Testfälle aus, sondern hilft auch bei der Kommunikation über Testfälle, beispielsweise mit Kunden, die ansonsten in der Regel kein Verständnis der Testautomatisierung gewinnen würden. Darüber hinaus erleichtern die wohldefinierten Keywords, die ja auch Schnittstellen zwischen verschiedenen fachlichen Ebenen darstellen, die Verständigung zwischen unterschiedlich spezialisierten Teammitgliedern bei der Arbeitsteiligkeit.

1.6.5 Arbeitsteiligkeit

Insbesondere in Projekten[4] mit einer hohen fachlichen Komplexität und einer gewissen Teamgröße gibt es häufig eine gewisse Spezialisierung oder Rollen (in alphabetischer Reihenfolge): *Rollen*

Anwender oder Fachexperte, Domain-Experte: Personen, die einen tiefen Einblick in die fachlichen Zusammenhänge haben und daher die Testfälle aus logischer Sicht hervorragend gestalten können, möglicherweise aber nicht darin ausgebildet sind, in die technischen Untiefen eines Testautomatisierungswerkzeugs einzutauchen.

Entwicklerin: Implementiert die eigentliche Anwendung. Es gibt hier häufig weitere Spezialisierungen, etwa nach den Themen Frontend, Backend oder Datenbank.

Testautomatisierer: Personen, die Testautomatisierung durch und durch beherrschen, noch vor dem Frühstück Code schreiben, aber die fachlichen Zusammenhänge des Testobjektes ggf. nur sehr bedingt verstehen.

Testdesigner: Gestaltet die Testfälle basierend auf den Informationen, die von Fachexperten bereitgestellt werden.

[4] Hier und an anderen Stellen im Buch wird von »Projekt« gesprochen. Das geschieht der Einfachheit und der besseren Lesbarkeit halber, aber:

Es geht hier nicht nur um Tests in Projekten, sondern auch um Tests von Produkten, in Verfahren oder bei der Wartung von Bestandssystemen. Aus Testsicht ist jedes Vorhaben, in dem Software oder Systeme getestet werden wollen, ein »Testprojekt«.

Testerin: Eine Person, die Testfälle manuell durchführt, aber auch Testdaten vorbereitet oder Auswertungen vornimmt.

Testmanagerin: Zuständig für die Organisation des Testens, Bestimmung der Ziele und der Strategie und somit auch maßgeblich daran beteiligt, eine Entscheidung für die Verwendung von Keyword-Driven Testing herbeizuführen.

Über Rollen im Test ist schon viel geschrieben worden. Widerstehen wir der Versuchung, das hier auch zu tun. Wer aber mehr dazu lesen will, findet weitere Informationen unter anderem in [10] unter »Kompetenzen und Teamzusammensetzung« oder auch in den Lehrplänen des ISTQB®[30].

Einige dieser Rollen können von derselben Person eingenommen werden, es ist aber untypisch, dass dies für Gegenpole wie »Fachexperte« und »Testautomatisierer« zutrifft.

Sicher: Idealerweise wünschen wir uns die legendäre eierlegende Wollmilchsau – also Heldinnen, die einfach alles können. Solche mag es auch geben. Aber leider nie genug. Realistisch betrachtet, kann sich glücklich schätzen, wer ein Team hat, in dem alle erforderlichen Aspekte kompetent abgedeckt werden. Diese unterschiedlichen Ausprägungen der Teammitglieder benötigen wir, und wir sollten keinesfalls eines über das andere stellen. Der beste Testautomatisierer kann durchaus völlig unbelastet von jeglicher Vorkenntnis sein, bezogen auf den Einsatz des Testobjektes. Anders gesagt, er hat absolut kein Konzept davon, was das System tut, für das er automatisieren soll. Das kann funktionieren.

Wichtig ist, dass wir nicht nur Expertinnen haben, die verstehen, wie entwickelt wird, sondern auch hervorragend ausgebildete Fachexperten, die wissen, was entwickelt werden sollte.

Spezialisten zusammenbringen

Und hier zeigt sich eine Stärke des Keyword-Driven Testing: Mit Keywords kann man eine Arbeitsteilung zwischen fachlicher Seite und technischer Seite, zwischen Fachexperte und Testautomatisierer erreichen. Während Fachexperten die Testfälle aus Keywords zusammensetzen, können sie die Details der technischen Umsetzung in einem Automatisierungswerkzeug ignorieren. Umgekehrt können sich die Testautomatisierer auf die technische Umsetzung fokussieren und die intimen Kenntnisse über das Automatisierungswerkzeug voll ausspielen. Dabei sind die genauen fachlichen Zusammenhänge für sie nicht wichtig.

Ohne Keyword-Driven Testing ist eine solche Arbeitsteilung, wenn überhaupt, nur wesentlich schwieriger und weniger elegant zu erreichen.

1.6.6 Vereinfachte Testautomatisierung

Testautomatisierung ist zwar nicht verpflichtend, aber in vielen Bereichen nicht mehr wegzudenken. Häufig sind Regressionstests ohne Testautomatisierung personell und zeitlich nicht mehr vorstellbar. Das gilt nicht nur, aber ganz besonders im agilen Umfeld, da hier die kurzen Entwicklungszyklen keinen Spielraum für zeitlich ausgedehnte manuelle Testphasen lassen.

Mit Keyword-Driven Testing lässt sich Testautomatisierung sehr effizient umsetzen. Das passende Framework vorausgesetzt, reicht es, für jedes Keyword eine Implementierung in der gewünschten Automatisierungsumgebung bereitzustellen.

Abstrakte Tests werden aus Keywords zusammengesetzt und konkrete Testfälle werden mithilfe von konkreten Daten spezifiziert.

Das Framework übernimmt dann die Ausführung der Automatisierungssequenz basierend auf dem aus Keywords bestehenden Testfall und den zugeordneten Automatisierungsskripten bzw. -funktionen.

Vorteil: Es wird nicht mehr für jeden Testfall eine Implementierung benötigt, sondern nur für jedes Keyword. Das reduziert die Menge des benötigten Automatisierungscodes erheblich. Bei angenommen 1000 Testfällen, bestehend aus 100 Keywords, werden ohne Keyword-Driven Testing 1000 Skripte benötigt[5], mit Keyword-Driven Testing nur 100 automatisierte Funktionen alias Keywords. Kommt man mit weniger Keywords aus, wird das Verhältnis noch günstiger.

Wenige Keywords reichen aus.

Im Idealfall lassen sich ab einem gewissen Zeitpunkt Testfälle zusammenstellen, die sofort und ohne weiteres Zutun schon fix und fertig automatisiert sind, da alle benutzten Keywords bereits umgesetzt wurden.

Die bereits genannten Vorteile wie Wartbarkeit und Wiederverwendbarkeit können und sollten natürlich auch mit normalen Programmiersprachen wie JavaScript, Java, C# oder Python erreicht werden. Die Vorteile hinsichtlich Klarheit, Kommunikation und Arbeitsteilung sind jedoch mit der komplexen Syntax solcher Programmiersprachen kaum zu erreichen. Das doch zu schaffen, ist das Ziel von Methoden wie Behavior-Driven Testing (siehe Abschnitt 8.3). Selbst bei der strengen Einhaltung von Clean-Code-Prinzipien und der Optimierung des Testcodes auf Lesbarkeit liegen Welten zwischen automatisierten Tests mit Keyword-Driven Testing und in einer Programmiersprache geschriebenen Tests. Alleine die notwendigen Klammern und andere Steuerzeichen bauen für manche Menschen zu starke Barrieren auf ...

[5] Zur Vereinfachung wird der Aspekt der Testdaten hier ausgeblendet.

Schicken Sie zu diesem Zeitpunkt aber bitte nicht Ihren fähigen Testautomatisierer nach Hause! Auch wenn die Implementierungsaufwände sinken, werden Änderungen an der Implementierung und neue Keywords diese Spezialisten immer benötigen. Zusätzlich werden aufgrund der gesteigerten Geschwindigkeit bei der Testfallerstellung auch entsprechende Zuarbeiten durch die Testautomatisierer notwendig.

Vor allem schafft ein Vorgehen nach Keyword-Driven Testing und die Entlastung der Automatisierer Freiraum, damit sich diese um die Bereitstellung und Betreuung der Testumgebungen und der Test-Pipelines kümmern können.

Angesichts einer klaren Rollentrennung zwischen »Keyword-Implementierenden« und »Test-Spezifizierenden« ist es ein konsequenter Schritt, wenn die Entwicklerinnen, die ohnehin für die Implementierung des Testobjektes zuständig sind, auch die Aufgabe der Implementierung für die Keywords auf technischer Ebene übernehmen.

1.6.7 Geschwindigkeit

Zugegeben, Keyword-Driven Testing erfordert in der Startphase einen Zusatzaufwand. Initial wird eine saubere Analyse benötigt. Darauf basierend muss ein Layer-Konzept (siehe Abschnitt 2.2.2) erarbeitet werden und es müssen Keywords definiert und beschrieben werden.

Diese initialen Aufwände werden aber auf Dauer amortisiert. Die Vorarbeit macht sich bezahlt: Sowohl das Spezifizieren geht schneller, wenn man auf fertige Bausteine zurückgreifen kann, als auch später die Ausführung der Tests.

Alle zuvor beschriebenen Vorteile erhöhen die Qualität unserer Tests und Testarchitektur und führen schlussendlich zu einer höheren Geschwindigkeit. Es sei jedoch gesagt, dass das Primärziel der höheren Geschwindigkeit unserer Erfahrung nach noch nie zu einer langfristigen Verbesserung geführt hat. Erhöhung der Qualität sowohl im Testobjekt als auch in der Testspezifikation und -architektur bringt andererseits aber langfristig anhaltend mehr Geschwindigkeit als Nebeneffekt.

1.7 Werkzeuge für Keyword-Driven Testing

Welche Tools verwenden wir nun, wenn wir Keyword-Driven Testing einsetzen wollen? Und können wir uns auf die Aussage »unterstützt Keyword-Driven Testing« verlassen?

Leider eher nein – die Werbeaussagen sind oft irreführend. Im besten Fall können diese Werkzeuge als ein Teil eines Frameworks genutzt werden, vollständige Unterstützung bietet keins. Das ist erst einmal auch völlig in Ordnung. Manche Werkzeuge halten aber – vielleicht

aufgrund eines sehr stark vereinfachten Verständnisses von KDT – bei Weitem nicht, was sie versprechen. Machen Sie sich bitte Ihr eigenes Bild (vgl. Kapitel 6).

Wie zuvor in der Begriffserklärung beschrieben, wird ein Framework für Keyword-Driven Testing also aus unterschiedlichen Werkzeugen gebildet. Welche Werkzeugarten Ihnen in diesem Zusammenhang begegnen mögen und was Sie benötigen, möchten wir nachfolgend kurz erläutern.

1.7.1 Testmanagementsysteme

Testmanagementsysteme spielen häufig (leider) eine untergeordnete Rolle im Keyword-Driven Testing, da die meisten nicht darauf ausgerichtet sind, aktiv Keyword-Driven Testing zu unterstützen.

Solche Testmanagementsysteme fokussieren sich »nur« auf die Aufgabe, die manuellen Tests zu verwalten und deren Spezifikation und Ergebnisse zu beherbergen; einige werden darüber hinaus dazu verwendet, Ergebnisse externer automatisierter Tests zu archivieren. Die Testspezifikationen der automatisierten Tests sucht man in der Regel vergebens in diesen Werkzeugen.

Um das für Keyword-Driven Testing leisten zu können, müssten die Werkzeuge in der Lage sein, mit wiederverwendbaren Keywords zu arbeiten und die Testdesigner bei modularer und idealerweise datengetriebener Spezifikation der Testfälle zu unterstützen. Geht das nicht, dann können also weder manuelle noch automatisierte Testfälle aus Keywords aufgebaut werden.

Es gibt jedoch wenige Ausnahmen – Testmanagementsysteme, die entweder die Möglichkeit haben, fertig implementierte Keywords aus einem Testautomaten zu importieren, oder mit denen Testdesigner in der Lage sind, neue Keywords zu entwerfen und diese auch im Testmanagementsystem zu strukturieren.

Genau hinzusehen lohnt sich: Einige Produkte auf dem Markt unterstützen zwar die Wiederverwendung einzelner Testschritte, aber nicht Keywords, wie wir sie verstehen.

1.7.2 Full-Stack-Testautomaten

Als Full-Stack-Testautomaten bezeichnen wir eine spezifische Gruppe von Testautomaten. Der Begriff »Full-Stack« ist ein Kunstwort, mit dem wir ausdrücken wollen, dass die Werkzeuge dieser Kategorie alle Komponenten, die zu Automatisierung eines Testobjektes notwendig sind, selbst enthalten.

Dadurch grenzen sie sich von den Automatisierungstechnologien und Automatisierungsframeworks ab, die zusätzliche Komponenten zum Betrieb benötigen. Diese Komponenten sind ganz grob gesprochen der Testeditor, die Ausführungs-Engine und die Adaptierungs- oder Automatisierungstechnologie, die das Testobjekt stimuliert und dessen Reaktionen entgegennimmt. Mehr zu diesen Komponenten und den entsprechenden Aufgaben findet sich später in Kapitel 5 »Testautomatisierungsarchitektur«.

Klassische Beispiele für Full-Stack-Automaten sind UFT, Test Complete, SoapUI oder Tools wie QF-Test, Tricentis Tosca, Ranorex oder Eggplant.

All diese Werkzeuge haben eigene Editoren, die teilweise per Drag & Drop oder per Scripting Tests erstellen können. Jedes Testautomatisierungswerkzeug ist dabei auch in der Lage, wiederverwendbare Testschritte zu erstellen und in Tests zu verwenden.

Obwohl einige dieser Automaten für sich in Anspruch nehmen, Keyword-Driven Testing zu unterstützen, möchten wir bei den wenigsten Automaten von Keyword-Driven Testing sprechen. Gerade was die Lesbarkeit, Einfachheit und Strukturierung von Keywords und Tests angeht, setzen diese Tools die Konzepte von Keyword-Driven Testing unserer Auffassung nach nicht zufriedenstellend um.

1.7.3 Testautomatisierungsframeworks

In Abgrenzung zu einem Keyword-Driven Testing Framework, das die gesamte konkrete Werkzeugkette vom Editor über die Testausführungs-Engine bis zur eigentlichen Automatisierungstechnologie definiert, kümmern sich die hier genannten Werkzeuge hauptsächlich darum, Testen in einer spezifischen Programmiersprache umzusetzen.

Sie definieren dazu eine Spezifikationssyntax, ohne selbst einen Editor mitzuliefern, und geben an, wie man eine Technologie, wie beispielsweise Selenium, adaptiert, um daraus eine vollständige Automatisierungslösung zu bauen. Diese »unterste« Ebene des Automaten ist entsprechend noch zusätzlich zu implementieren oder zu adaptieren (siehe hierzu das Kapitel 5 »Testautomatisierungsarchitektur«).

Ein (Test-)Automatisierungsframework implementiert im Gegensatz zum Full-Stack-Automaten also nicht alle Komponenten eines fertigen Testautomaten, sondern konzentriert sich hauptsächlich auf die Spezifikationssprache, Ausführung von Tests und Dokumentation von Testergebnissen.

Die gängigsten Testautomatisierungsframeworks lassen sich in eine von zwei Kategorien packen:

Unit-Testing Frameworks: Frameworks dieser Kategorie, wie JUnit (Java), Pytest (Python), NUnit (C#), Google Test (C++) oder Mocha (JavaScript), verwenden alle genau die Programmiersprache zur Testspezifikation, in der sie selbst und das zu testende Testobjekt – der zu testende Code – implementiert sind.

Behaviour-Driven Development Frameworks: Diese Gruppe von Frameworks bieten dem Anwender die Möglichkeit, Tests als menschenlesbaren Text zu spezifizieren, obwohl der eigentliche Testcode in kompilierter Form vorliegt. In der Regel werden solche Tests in Gherkin-Syntax (Given/When/Then) geschrieben (siehe Abschnitt 8.3 »Behavior-Driven Testing«).

Wie steht es denn nun um Frameworks für Keyword-Driven Testing? Da Keyword-Driven Testing seine Stärken vor allem auf höheren Teststufen zeigt und Entwicklerinnen auf diesen Teststufen selten unterwegs sind, ist die Toolauswahl hier ziemlich begrenzt.

Es gbt aber ein Open Source Framework, das sowohl Behavior-Driven Testing als auch Keyword-Driven Testing hervorragend umsetzt und deswegen auch in diesem Buch besonders beleuchtet werden soll. Robot Framework ist explizit als Keyword-Driven-Automatisierungsframework entwickelt worden und kann somit viele Stärken von Keyword-Driven Testing nutzen. (Robot Framework wird in Kapitel 7 »Praxis mit Robot Framework« detailliert vorgestellt und besprochen.)

1.7.4 Testdesignwerkzeuge und Editoren

Ein Großteil von Keyword-Driven Testing spielt sich im Editor bzw. in der Benutzeroberfläche des Testautomaten ab. Leider finden wir auch gerade hier die meisten Unzulänglichkeiten.

Diese Werkzeugkategorie ist als eigenständiges Werkzeug eigentlich nur in Verbindung mit Automatisierungsframeworks relevant. Testmanagementsysteme steuern ja genau diesen Teil als eine Komponente zum Keyword-Driven Testing Framework bei. Full-Stack-Automaten haben ebenfalls einen eigenen Editor, auch wenn die Unterstützung von Keyword-Driven Testing zu wünschen übrig lässt.

Für die meisten Frameworks werden Entwicklungsumgebungen, wie Visual Studio Code, IntelliJ IDEA oder Eclipse, mit entsprechenden Erweiterungen verwendet. Ein Nachteil gegenüber dedizierten Keyword-Driven-Testing-Editoren sind die vielen für die Softwareentwicklung erstellten, jedoch für Keyword-Driven Testing unnötigen

GUI-Komponenten, die solche Tools oft überladen aussehen lassen und Nutzerinnen abschrecken.

Es ist nicht einfach, die richtigen Komponenten zu wählen, und eine »Silver Bullet«[6] gibt es nicht. Wir hoffen aber, in diesem Buch genügend handfeste Entscheidungshilfen zu liefern, damit Sie sich Ihren eigenen Werkzeugkoffer so zusammenstellen, dass es ein erfolgreiches Framework wird.

1.8 Beispiele in diesem Buch

Im Laufe des Buches werden wir zur Erläuterung auf Beispiele zurückgreifen. Manche sind sehr spezifisch; wo es aber möglich ist, werden wir ein einfaches, realistisches und für jeden nachvollziehbares Beispiel verwenden, das nicht von dem ablenkt, was erklärt werden soll. Wir haben uns gegen den ewigen Geldautomaten entschieden und nehmen statt dessen etwas anderes, von dem wir sicher sind, dass jede und jeder es kennt: einen Onlineshop.

Weil das Beispiel fiktiv ist, brauchen wir uns nicht zurückzuhalten, sondern können jederzeit Testfälle aus diesem Projekt zitieren. Es dient also als »fachliche« Referenz.

Mit echten Praxisbeispielen wäre das schwieriger, weil jeglicher Inhalt daraus natürlich vertraulich ist. In anonymisierter Form und ohne dass die betreffenden Organisationen erkennbar sind (außer vielleicht für Menschen, die damit ohnehin selbst arbeiten), können wir aber zwei Beispiele zu real in der Praxis eingesetzten Frameworks skizzieren. Das tun wir in Abschnitt 6.4.

Zurück zu unserem fachlichen Beispiel – geben wir ihm ein Gesicht:

In dem Projekt geht es also um einen Onlineshop. Stellen Sie sich vor, es werden Gegenstände verschiedener Künstler und Kunsthandwerker angeboten. Bei den Waren handelt es sich vielleicht um Drechsel- oder Töpferarbeiten, Aquarelle und Ölbilder, Mode eines kleinen Modelabels oder aber auch um handgestrickte Pullis aus Schafwolle.

Der Shop ähnelt vielen gängigen Shops. Es gibt ein Suchfeld, einen Warenkorb, verschiedene Kategorien auf der linken Seite, Bewertungsmöglichkeiten der User zu den einzelnen Produkten sowie einen persönlichen Login-Bereich.

Es gibt eine Desktop-Version für alle gängigen Browser sowie native Apps für iOS und Android. Aktuell plant das Management, für den Shop eine von den verschiedenen Plattformen unabhängige Web-App

[6] Eine »Silver Bullet« (Silberkugel) ist eine der wenigen Waffen die gegen Hexen und Werwölfe hilft und metaphorisch als eine einfache und fast magische Lösung für ein schwieriges Problem genutzt wird.

zu entwickeln. Diese Web-App soll die gleichen Funktionalitäten bieten wie die nativen Apps. Das Management verspricht sich davon eine Kosteneinsparung bei der Wartung und Pflege der einzelnen Apps.

Der Shop ist weltweit zu erreichen und die verkauften Waren werden in alle sechs Kontinente ausgeliefert.

Unser fiktiver Shop ist bereits längere Zeit auf dem Markt, sodass es viele User gibt, hoher Traffic vorhanden ist und falls etwas nicht funktioniert, werden wir schnell ein Problem haben.

Wir denken, ein sorgfältiger Test unseres Shops ist angemessen und die Herausforderungen passen zu Keyword-Driven Testing.

1.9 Ressourcen

Einige Beispiele, mit denen Sie selbst die im Buch, insbesondere in Kapitel 7 »Praxis mit Robot Framework«, vorgestellten Konzepte praktisch nachvollziehen können, haben wir für Sie zum Download bereitgestellt. Den Link hierzu haben wir im Literaturverzeichnis als [37] hinterlegt. Ganz einfach erreichen Sie die Seite aber mit diesem QR-Code:

Abb. 1-5 *Link zu Ressourcen*

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-

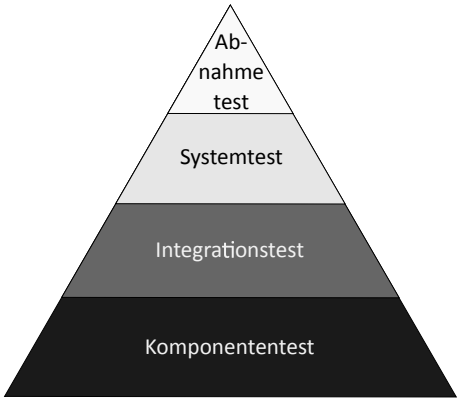

Abb. 2-2
Testpyramide (Wunschbild)

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].

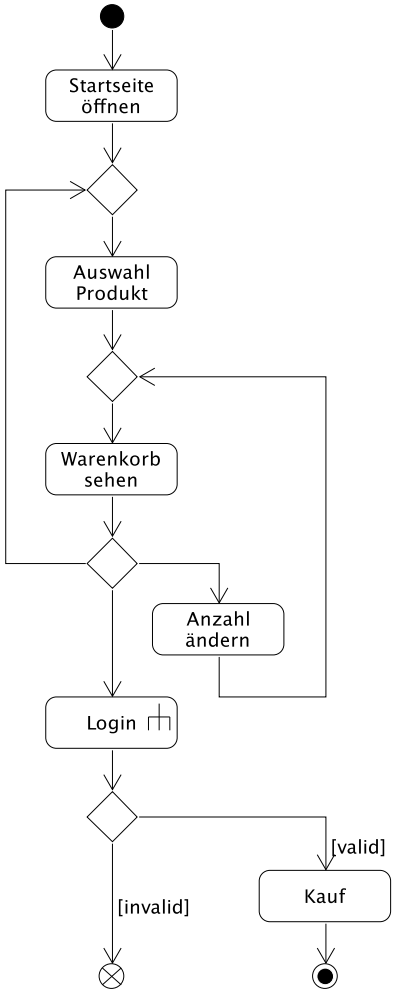

Abb. 2-3 *UML-Diagramm: Einkauf*

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)

Abdeckungskriterium

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.

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.

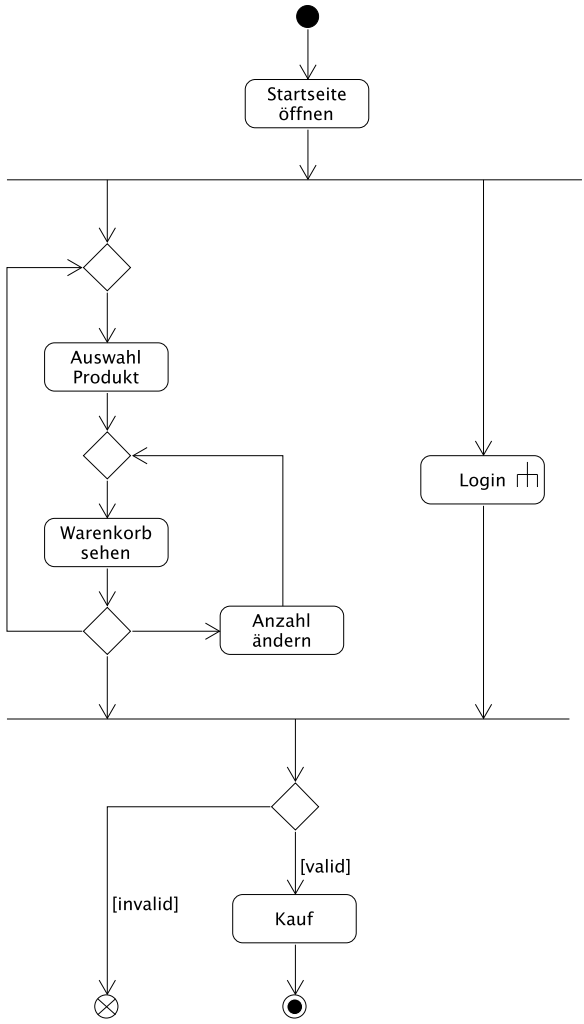

Abb. 2-4 *UML-Diagramm: Einkauf – Version 2*

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.