REFAKTORISIEREN VON TESTS: ERFAHRUNGEN AUS DER SCHLÜSSELWORTGETRIEBENEN TESTENTWICKLUNG

Tests und Testautomatisierung sollen helfen, Kosten zu sparen und die Softwarequalität zu erhöhen. Damit das mittel- bis langfristig gelingt, müssen auch Tests „gehegt und gepflegt" werden. In diesem Artikel erfahren Sie anhand ausführlicher Beispiele, welche Möglichkeiten des Refaktorisierens dabei helfen, dieses Ziel zu erreichen und es nachhaltig zu sichern. Der Artikel basiert auf mehrjährigen Erfahrungen im Bereich des schlüsselwortbasierten Testens, gibt konkrete Hinweise zum erfolgreichen Refaktorisieren in diesem Umfeld und verdeutlicht diese an einer Beispielapplikation mit dazugehörigem Test.

der autor

Markus Tiede
(E-Mail: Markus.Tiede@Bredex.de)
arbeitet als Softwareentwickler und Tester für die BREDEX GmbH in Braunschweig mit den Schwerpunkten Spezifikation und Wartung automatischer Tests sowie Eclipse-RCP-Entwicklung. Er gehört zum Entwicklerteam des Testautomatisierungswerkzeugs „GUIdancer".

Die Notwendigkeit zu refaktorisieren beschränkt sich nicht auf das Themengebiet produktiven Softwarecodes. Vielmehr gilt das Vorhandensein von automatischen Tests als Voraussetzung für effektive Codeverbesserungen, die die Software im Fall einer Änderung erneut auf Herz und Nieren prüfen. Refaktorisieren soll die Lesbarkeit, Verständlichkeit, Wart- und Erweiterbarkeit von Softwarecode verbessern und letztendlich einen Fortschritt bedeuten.

Damit besitzen automatisierte Tests einen besonderen Stellenwert im Softwareentwicklungsprozess. Einerseits unterstützen sie den Entwickler während der initialen Erstellung von Softwarecode, andererseits sollen sie darüber hinaus – während des gesamten Entwicklungsprozesses einer Software – als Regressionstests dienen. Denn eben diese Rolle als Regressionstest ist für Refaktorisierungsmaßnahmen ungemein wichtig: die ständige und zeitnahe Prüfung von Soll- und Ist-Zustand. Damit Tests dieser Rolle gerecht werden können, müssen auch sie mit der Zeit gewissen Änderungen unterliegen, um – passend zum aktuellen Produktivcode – immer einen Produktivtest darstellen zu können. Auch Tests unterliegen somit wiederum diesem Refaktorisierungsgedanken und bedürfen der Anpassung und Verbesserung.

Dieser Artikel befasst sich mit genau diesem Thema – dem Refaktorisieren von schlüsselwortbasierten Tests – und beantwortet die folgenden Fragen:

- Welche Tests sollten refaktorisiert werden?
- Wie und wo können Tests refaktorisiert werden?
- Wann refaktorisiere ich Tests und wo liegen die Grenzen?

Refaktorisieren: das „Was"

Meinen Erfahrungen nach muss nicht jede Testart gleichermaßen refaktorisiert werden: Tests, die schon von Beginn an die vorgesehene Schnittstelle einer Software testen (unabhängig von der Ebene auf der sie ansetzen), müssen in der Regel seltener refaktorisiert werden. Beispiele hierfür sind Komponententests in Form von JUnit-Tests oder auch Systemtests in Form von automatisierten Oberflächentests. Weniger betroffen sind außerdem Tests, die selbst schon einen modularen Aufbau besitzen. Der Begriff „Schnittstelle" bezieht sich dabei weniger auf den augenblicklichen Ablauf und die konkrete Benutzungsschnittstelle der Software als vielmehr auf die zu Grunde liegenden Geschäftsregeln der Applikation und die damit verbundenen fachlichen Anforderungen.

Schlüsselwortbasierte Tests verwenden eben diesen modularen Aufbau von Beginn an und erlauben es, bereits bei der Erstellung wiederkehrende Aktionen und Abläufe in Modulen unter Verwendung von *Keywords* (Schlüsselwörtern) zusammenzufassen. In der Regel besteht ein solches Schlüsselwort aus drei atomaren Informationsbestandteilen:

- Welches Objekt soll getestet werden?
- Welche Aktion soll auf dem Objekt durchgeführt werden?
- Mit welchen Daten soll diese Aktion auf dem Objekt durchgeführt werden?

Auf der anderen Seite gibt es Tests, die sich nicht an zuvor definierte Schnittstellen halten, weil sie sich z. B. stark an aktuellen Implementierungsdetails orientieren und keine Mittel zur Strukturierung und Kapselung von Implementierungsdetails verwenden. Solche Tests müssen häufiger angepasst und überarbeitet werden. Dies führt einen direkt zu der Frage, warum Tests eigentlich refaktorisiert werden sollten: Refaktorisieren ist auch für automatisierte Tests kein reiner Selbstzweck, sondern ist z. B. wegen eines konkreten Mangels an Lesbarkeit und Modularität oder auch wegen zu großer Redundanz durchzuführen. Diese Notwendigkeit betrifft Produktiv- und Testcode gleichermaßen. Das bedeutet, dass sich schon die Wahl des zu Grunde liegenden Test-Frameworks und der Testmethodik nachhaltig auf den Zeitpunkt und die Dauer der Refaktorisierungsmaßnahmen auswirken kann. Meiner Erfahrung nach gehören zur letzteren Kategorie von Tests auch solche, die über einen *Capture&Replay*-Ansatz erstellt werden, da diese zwar schnell zu ersten Ergebnissen führen, jedoch langfristig oft redundante Bereiche beinhalten, die die Wartbarkeit stark beeinträchtigen.

Die Art und Weise, wie Tests refaktorisiert werden können, ähnelt den Methoden, die

auch auf gewöhnlichen Programmcode angewendet werden. Die im Folgenden vorgestellten Refaktorisierungsmaßnahmen und Beispiele stammen aus dem Umfeld automatischer, schlüsselwortbasierter Tests im Bereich graphischer Benutzungsoberflächen, lassen sich aber problemlos auf Testcode übertragen.

Refaktorisieren: das „Wie" und „Wo"

Die erste und einfachste Maßnahme, um die Lesbarkeit und Verständlichkeit des vorliegenden Testcodes zu verbessern, besteht darin, bessere Bezeichner für Testklassen, -methoden und -schlüsselwörter zu vergeben. Je aussagekräftiger solche Bezeichner sind, desto mehr fördern sie einerseits die Lesbarkeit und fordern andererseits weniger die Kapazitäten des Testers, wenn es darum geht, den vorliegenden Testcode zu verstehen. Die Verständlichkeit des Testcodes gewinnt – analog zum Softwarecode – an Bedeutung, je größer der Kreis der Personen ist, der mit ihm in Berührung kommt. Besteht das Testteam lediglich aus einer Person, wird diese Art des Refaktorisierens viel seltener durchgeführt, als wenn beispielsweise mehrere Personen an der Pflege und Erstellung der Tests beteiligt sind. Das hat einen einfachen Grund: Der einzelne Tester versteht häufig auch ohne gute Bezeichner die Bedeutung seiner Tests.

Gerade aber im Bereich des schlüsselwortbasierten Testens sind es diese Bezeichner, die noch eine weitere wichtige Rolle im Umfeld des Testens übernehmen können: Ein Test mit prägnanten Schlüsselwörtern, selbsterklärenden Testparametern und sprechenden Komponentennamen benötigt in der Regel kein zusätzliches Testdokument[1]), das den Test beschreibt – diese Funktion übernehmen der Test und der Kontext, in dem sich der Test befindet. Damit schlägt man zwei Fliegen mit einer Klappe: Die Wartbarkeit und Übersichtlichkeit des Tests werden erhöht und der Zusatzaufwand zur Erstellung und Pflege von Dokumentation wird eingespart. Das

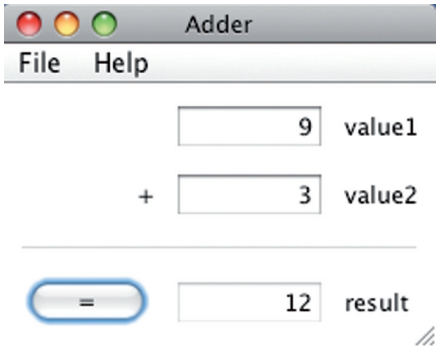

Abb. 1: Ein simpler Addierer.

ist ein nicht zu verachtender Aspekt von Redundanzvermeidung und -reduktion im Bereich automatisierter Tests.

Diese und weitere Refaktorisierungsmaßnahmen sollen anhand der folgenden überschaubaren Anwendung und der dazugehörigen Testspezifikation verdeutlicht werden. Als Beispiel dient ein trivialer Taschenrechner zum Summieren zweier Werte (siehe Abb. 1).

Getestet werden sollen die Addition der beiden eingegebenen Summanden und die korrekte Anzeige des Ergebnisses in der graphischen Benutzungsoberfläche. Der dazugehörige formale Test[2]) „Testfall 1" auf Basis schlüsselwortbasierter Testspezifikation könnte so aussehen, wie in Tabelle 1 dargestellt.

Mit nur wenigen Änderungen der im Test verwendeten Bezeichner – sowohl für die zu testenden Objekte, als auch für die durchzuführende Aktion und für das Schlüsselwort – wird der Testlauf „Einfache Addition" lesbarer und auch für externe Personen ohne Vorlage eines Testdokuments nachvollziehbar (siehe

Testfall 1		zu testendes Objekt	durchzuführende Aktion	verwendete Daten
Schritte	1.	Textfeld 1	Texteingabe	9
	2.	Textfeld 2	Texteingabe	3
	3.	Button	Klicken	–
	4.	Textfeld 3	Textüberprüfung	12

Tabelle 1: Die einfachste Form der Testfallbeschreibung.

Einfache Addition		zu testendes Objekt	durchzuführende Aktion	verwendete Daten
Schritte	1.	Feld Summand 1	Gebe ersten Summanden ein	=Summand1
	2.	Feld Summand 2	Gebe zweiten Summanden ein	=Summand2
	3.	Berechne Button	Stoße Berechnung an	–
	4.	Ergebnisfeld	Prüfe das Ergebnis	=Ergebnis

Tabelle 2: Testfallbeschreibung mit eindeutigen Bezeichnern und Parameternamen.

Einfache Addition		=Summand1	=Summand2	=Ergebnis
Datensatz	1.	1	2	3
	2.	1	-1	0
	3.	17	4	21

Tabelle 3: Trennung von Testdaten und -aktion durch die Verwendung von Parametern.

[1]) *Die Möglichkeit, ein Schlüsselwort und dessen Bestandteile mit „inline"-Kommentaren zu versehen, vorausgesetzt.*

[2]) *Die in diesem Artikel verwendete Tabellenform zur Darstellung der Testspezifikation dient der Tool-Unabhängigkeit und lässt sich leicht auf Testcode oder andere Syntaxformen übertragen.*

Tabelle 2). Gleichzeitig zeigt die geänderte Version die nächste Möglichkeit, Tests zu refaktorisieren: Durch das Um- und Auslagern von Testcode und Informationen in andere Klassen und Bereiche[3]) wird der Test übersichtlicher und die Testdaten können zentral am Schlüsselwort hinterlegt werden (siehe Tabelle 3).

Neben der Auslagerung von Testdaten an zentrale Stellen fällt in diesen Bereich des Refaktorisierens auch das eindeutige Kategorisieren von Testmodulen und Schlüsselwörtern. Nach meiner Erfahrung gibt es mindestens drei verschiedene Kategorien beziehungsweise Level, in die sich Tests einordnen lassen:

- *Setter* und *Checker:* Dies sind einzelne kleine Module und Methoden, die die Aufgabe übernehmen, Werte im Testobjekt zu setzen bzw. zu initialisieren und diese zu prüfen. Sie bilden die Basis, auf der alle weiteren Tests aufbauen. In unserem Fall sind das die Schritte 1 bis 4 des Testfalls „Einfache Addition" in Tabelle 2.
- *Fachliche Module:* Diese realisieren – unabhängig von der zu Grunde liegenden Implementierung – einen Teiltest. In unserem Fall ist dies das Modul „Einfache Addition". Es kapselt implementierungstechnische Details, z. B. die zu verwendende Aktion und das Objekt, auf dem diese angewendet wird, und reicht nur nach außen, was fachlich für den spezifizierten Test relevant ist: die Daten der Addition und die Information, dass es sich hier um eine einfache Addition handelt.
- *Ganzheitlich ausführbare Tests:* Diese besitzen keine Vor- bzw. Nachbedingungen. Ein solcher Testlauf verwendet beispielsweise das Modul „einfache Addition" und stellt zuvor sicher, dass sich das Testobjekt – in unserem Fall die Applikation – im korrekten Zustand befindet, um eine Addition durchzuführen. Ebenso gewährleistet das Modul, dass der Test in einem definierten Zustand endet.

Eine weitere Form, um Testcode umzulagern, besteht darin, den Inhalt kategorisierter Testmodule, -methoden und -klassen,

[3]) *In diesem Fall das Auslagern der verwendeten Testdaten unter Verwendung von Parametern, gekennzeichnet durch ein „=".*

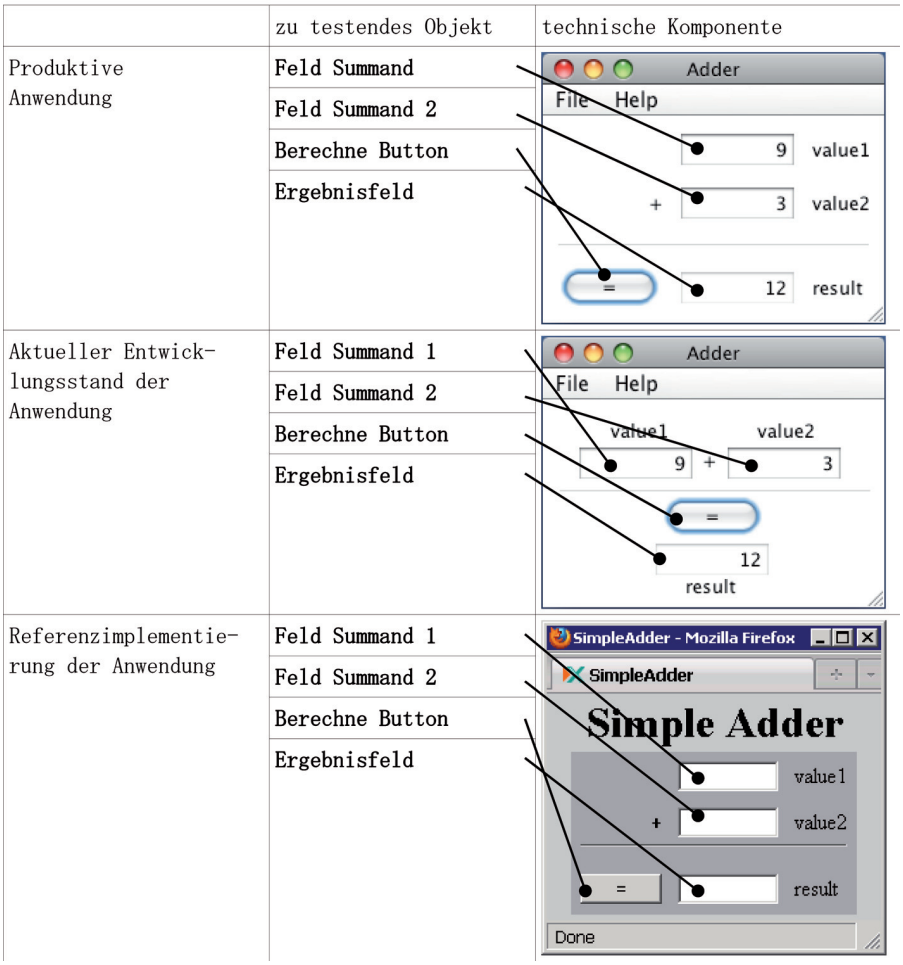

Abb. 2: Zuordnungstabelle von fachlichem zu technischem Testobjekt.

aufzuteilen und zusammenzuführen. Bei wachsenden Tests ist dieses gezielte Zusammenführen analoger Tests und Testabschnitte ein stetiger Prozess, der langfristig Redundanzen in der Testspezifikation vermindert.

Gerade beim Testen graphischer Benutzungsoberflächen lässt sich das Zusammenziehen bestimmter Testdaten und -objekte sehr gut veranschaulichen: Anstatt ein Schlüsselwort direkt an ein technisches zu testendes Objekt zu binden, z. B. das Feld für den ersten Summanden, empfiehlt es sich, solche Daten, die mit der Zeit mit hoher Wahrscheinlichkeit gewissen Änderungen unterliegen, zentral zu verwalten und innerhalb der Testspezifikation mit Platzhaltern für diese Objekte zu arbeiten. Diese Zuordnung vom fachlichen Testobjekt zum realen, technischen Objekt in der Applikation bietet gleich mehrere Vorteile: Zum einen gibt es genau eine Verbindung pro fachlich benötigtem Testobjekt, was eine leichtere Wartbarkeit gewährleistet. Zum anderen können verschiedene Ausprägungen dieser Zuordnungstabelle für ein und denselben Testfall existieren, denn der Testfall ist losgelöst von der technischen Anwendung, die getestet werden soll. In unserem Beispiel lässt sich damit der zuvor beschriebene Test auf eine andere technische Ausprägung transferieren (siehe Abb. 2).

Ein solcher entkoppelter Test ist somit auch auf verschiedenen Versionen der Anwendungen lauffähig, vorausgesetzt die fachlichen Geschäftsregeln der Anwendung bleiben gleich – in unserem Fall sind das die Struktur und Funktion eines einfachen Taschenrechners. Ein weiterer Vorteil dieses Prinzips, der jedoch für das Refaktorisieren eher vernachlässigt werden kann, bestet darin, dass die Testspezifikation und die Verfügbarkeit der realen Implementierung nicht notwendigerweise zusammenfallen müssen. Tests können durch diese strikte Entkopplung schon ohne eine direkte Verfügbarkeit der

Einfache Arithmetik	zu testendes Objekt	durchzuführende Aktion	verwendete Daten
Schritte	1. Feld Operand 1	Gebe ersten Operanden ein	=Operand1
	2. Feld Operand 2	Gebe zweiten Operanden ein	=Operand2
	3. =OperatorButton	Stoße Berechnung an	–
	4. Ergebnisfeld	Prüfe das Ergebnis	=Ergebnis

Tabelle 4: Variabilität durch parametrisierte Testobjekte, in diesem Fall des „Operators".

Software spezifiziert werden und erst später diese konkrete Bindung erhalten.

Tests, die auf diese Art und Weise strukturiert werden, lassen sich leichter überblicken und Teilstücke lassen sich gezielt wiederverwenden. Der Gedanke der Wiederverwendung von bereits vorhandenen Tests stellt die nächste Möglichkeit des Refaktorisierens dar – der Auslagerung von gemeinsam nutzbaren Teilen einer Testlogik durch Abstraktion und die Verwendung zentraler Module. Eine Anwendung dieses Prinzips ist die schon erwähnte Verwendung von parametrisierten Testfällen: Testablauf und die dafür verwendeten Testdaten werden getrennt und abstrahiert. Eine weitere Form der Entkopplung– neben der Trennung von fachlichem und technischen Testobjekt – kann beispielsweise auch die Verwendung von parametrisierten Testobjekten sein (siehe Tabelle 4). Der Test kann somit in verschiedenen weiteren Fällen eingesetzt werden, beispielsweise wenn die Anwendung um weitere Operatoren erweitert wird (siehe Abb. 3).

Mit diesem Mittel der Abstraktion lässt sich der Test entsprechend erweitern und variabler gestalten (siehe Tabelle 5).

Diese Form des Refaktorisierens – Tests so mit variablen Merkmalen zu versehen, dass sie in vielen verschiedenen Variationen eingesetzt werden können – wirkt sich stark auf die Wartbarkeit der Tests aus. Denn die erweiterten Module stellen den zentralen Anlaufpunkt dar, wenn es darum geht, den Test beispielsweise aufgrund von Änderungen am Produktivcode anzupassen. Änderungen an dieser einen zentralen Stelle wirken sich automatisch auf alle Stellen im Test aus, die auf dieses Modul verweisen. Wartungsarbeiten am Test können so effizient vorgenommen werden und führen gerade bei agilen Entwicklungsprozessen dazu, dass der Erstellungsaufwand für neue Tests reduziert werden kann, denn viele der Module können einfach wiederverwendet werden.

Eine solche Maßnahme sollte man allerdings mit Bedacht ergreifen: Das Potential, das sich im Falle einer Änderung ergibt, kann gleichzeitig eine Gefahr darstellen, wenn die vorgenommene Änderung ihrerseits fehlerhaft ist. Zudem kann man durch das mehrfache Anwenden solcher Refaktorisierungsmaßnahmen (hoch-)generische Tests erhalten (siehe Tabelle 6). Jedoch sollte man dabei in einem „gesunden Maß" refaktorisieren: Die zuletzt durchgeführte Refaktorisierungsmaßnahme ergibt das – schon auf den ersten Blick – wenig verständliche Schlüsselwort: Einfache Arithmetik [Operand1; Operand2; Summe; Differenz; Produkt; Quotient] für den vorgestellten simplen Taschenrechner (siehe Tabelle 7), was zusätzlicher Testdokumentation bedarf.

Einfache Arithmetik	=Operator Button	=Operand1 Aktion	=Operand2 Daten	Ergebnis
Datensatz 1.	Addiere	1	2	3
2.	Subtrahiere	1	1	0
3.	Multipliziere	17	4	68
4.	Dividiere	9	3	3

Tabelle 5: Parametrisierung mit verschiedenen Operatoren.

Einfache Arithmetik	=Operator Button	=Operand1 Aktion	=Operand2 Daten	Ergebnis
Datensatz 1.	Addiere	=Operand1	=Operand2	=Summe
2.	Subtrahiere	=Operand1	=Operand2	=Differenz
3.	Multipliziere	=Operand1	=Operand2	=Produkt
4.	Dividiere	=Operand1	=Operand2	=Quotient

Tabelle 6: Wiederholtes Auslagern von Daten und Testobjekten.

Einfache Arithmetik		=Operand1	=Operand2	=Summe	=Differenz	=Produkt	=Quotient1
Datensatz	1.	1	1	2	0	1	1
	2.	9	3	12	6	27	3

Tabelle 7: Erneute Trennung von Testdaten und -aktion -- zu viel des Guten?

Refaktorisieren: das „Womit"

Weitere Faktoren, die beachtet werden müssen, sind die Größe und das Ausmaß des Refaktorisierens, denn diese können eine erhebliche Menge an Testcode betreffen und einen nicht geringen, zum Großteil monotonen, Arbeitsaufwand bedeuten. Solange für die vorgestellten Methoden, mit deren Hilfe man Tests und Testcode refaktorisieren kann, wie beim Restrukturieren von Softwarecode, das Prinzip der „kleinen Schritte" eingehalten wird, können die Maßnahmen zum Refaktorisieren einfach als Sequenz atomarer Änderungen angesehen werden und bleiben damit trotzdem überschau- und in Teilaufgaben zerlegbar. Diese atomaren Änderungen besitzen zudem eine höhere Zuverlässigkeit, korrekt durchgeführt zu werden, wenn sie durch ein Werkzeug unterstützt werden: einerseits weil das Werkzeug die Korrektheit dieser atomaren Aktionen gewährleistet, andererseits weil der Tester, der das Refaktorisieren durchführt, den Fokus auf den größeren Zusammenhang legen kann, statt die vielen kleinen Änderungen am Testcode selbst durchführen zu müssen. Daher empfiehlt es sich, beim Refaktorisieren von Tests auf eine geeignete Unterstützung durch das jeweils verwendete Tool zu achten. Viele moderne IDEs bieten hier Möglichkeiten des werkzeuggestützten Refaktorisierens.

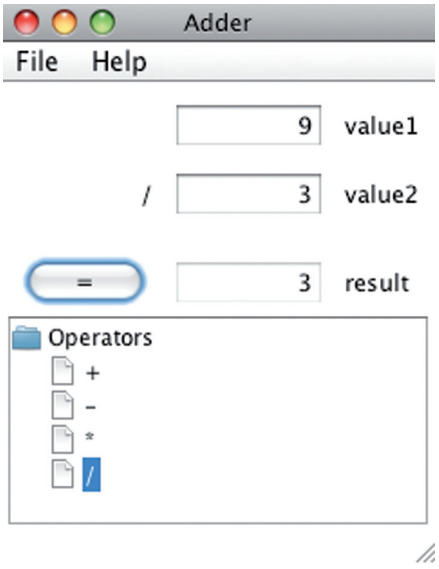

Abb. 3: Beispielapplikation mit erweitertem Funktionsumfang.

Refaktorisieren – das „Wann"

Die in diesem Artikel vorgestellten Maßnahmen sollten immer im Wechsel zum Refaktorisieren von produktivem Code angewendet werden. Das heißt, die Tests sollten nach Möglichkeit dann überarbeitet werden, wenn nicht gleichzeitig Anpassungen in den Teilbereichen des Programmcodes durchgeführt werden, die sie prüfen sollen.[4]) In diesem Fall dient der Code wiederum als Test für die Tests und muss sich dafür in einem stabilen, selbst auch getestetem Zustand befinden. Denn wenn ein Test nach durchgeführtem Refaktorisieren an selbigen fehlschlägt, ist nicht unbedingt die Software fehlerhaft, sondern mit hoher Wahrscheinlichkeit der Test selbst. Und genau hier stößt man auf den Gedanken: „Warum nicht Tests für den Test schreiben?" Diese Frage lässt sich nach Belieben erweitern und führt dazu, dass sich immer mehr Ebenen ergeben, die wiederum dem Refaktorisieren unterliegen – es handelt sich um ein Fass ohne Boden. Meiner Erfahrung nach erreicht man hier schnell die Grenze des Sinnvollen – ein Test für Tests ist die Software selbst.

Fazit

Wer langfristig, erfolgreich und automatisiert testen will, kommt um das Refaktorisieren seiner Tests nicht herum: Dieses ist ein fester Bestandteil des Software- und Testentwicklungsprozesses. Es gibt vielfältige Möglichkeiten, um diesen Erfolg und damit die Qualität der Software und Tests zu sichern. Einige davon habe ich in diesem Artikel vorgestellt. Die geschickte Wahl des verwendeten Test-Frameworks entscheidet nicht zuletzt über den Aufwand und die damit verbundenen Risiken. ■

[4]) Ausgenommen hiervon sind explizite Schnittstellenänderungen. Sie verlangen gleichzeitige Änderungen auf beiden Seiten.