mehr zum thema:
www.xebia.github.com/Xebium
www.fitnesse.org
www.seleniumhq.com

AUFBRUCH ZU DEN FRAKTALEN KÜSTEN: MIT XEBIUM VERBORGENE POTENZIALE WECKEN

Thema dieses Artikels ist die Automatisierung von Software-Tests. Eine vollständige Testautomatisierung ist nicht möglich, heißt es oft. Zu hoch ist die Volatilität der Software während des Entwicklungsprozesses und zu groß der Anpassungsaufwand. Wir widersprechen: Eine hinreichend vollständige Testautomatisierung ist möglich. Einen Lösungsansatz dafür findet man in der Mathematik.

die autoren

Steffen Müller
(steffen.mueller@viskonz.de)
ist Senior Consultant, Scrum-Master, Testberater und Geschäftsführer der Viskonz GmbH.

Felix Stein
(stein@domus-aurea-consulting.de)
ist Projektmanager und Senior Consultant bei der DomusAurea Consulting GmbH & Co KG.
Beide Autoren eint ihre Vorliebe für praxisnahe Testautomatisierung in IT-Projekten.

Am Anfang steht eine Vision: die komplette Testabdeckung aller Funktionen einer Software durch eine andere Software. Schnell, zuverlässig, unbestechlich – und das schon während des Entwicklungsprozesses. Ein verlockender Ansatz, fiele doch die Abhängigkeit von den häufig unzuverlässigen manuellen Ressourcen weg, welche die Qualitätssicherung vieler Projekte dominieren.

Werkstudenten, Praktikanten und ausländische Service-Center scheitern zu häufig an Sprach- und Verständnisbarrieren oder an dem der Monotonie geschuldeten Aufmerksamkeitsverlust, sodass das „Durchklicken" der Testfälle mitunter selbst zum Qualitätsrisiko wird. Der Fall scheint klar zu sein, die Vision muss Wirklichkeit werden. Die reine Logik gebietet es, möchte man meinen.

Ein Blick in die Praxis zeigt allerdings ein komplett anderes Bild: Häufig finden nur Teil-Automatisierungen statt und viel zu häufig kann man sogar Rückschritte beobachten – nach hoffnungsvollen Anfängen werden bereits eingeführte Tools wieder abgeschaltet oder (schlimmer noch) einfach links liegen gelassen, sodass ihre nicht mehr aktuellen Testdurchläufe irgendwann nur noch rote Reports generieren, Dokumente des Scheiterns. Stattdessen werden wieder die bereits erwähnten manuellen Ressourcen eingesetzt. Wie kann das sein, muss man sich da fragen, wie ist dieses Bild zu erklären?

Die Antwort darauf ist eine bei vielen Projektmanagern verbreitete Fehlwahrnehmung: Um eine im Entstehen begriffene (und sich daher permanent ändernde) Software automatisiert zu testen, wäre ein so hoher Pflege- und Aktualisierungsaufwand der Testprogramme notwendig, dass das manuelle Testen im Vergleich die einfachere und wirtschaftlichere Variante wäre. Heißt es. Wir aber sehen das grundsätzlich anders: Richtig eingesetzt, können automatisierte Tests nicht nur schnell und zuverlässig sein, auch der notwendige Instandhaltungsaufwand lässt sich gering halten.

Wie wir das bewerkstelligen wollen? Indem wir uns von einem Mathematiker inspirieren lassen, der sich mit einer Frage beschäftigte, die auf den ersten Blick überhaupt nichts mit unserem Thema zu tun hat: Wie vermisst man die Gesamtlänge der Küsten von Großbritannien? Die Antwort darauf ist auch die Antwort auf unsere Frage: durch den Einsatz von Fraktalen.

Die Vermessung einer Küstenlinie

Die unter Mathematikern berühmte Frage nach der britischen Küstenlänge wurde im Jahr 1967 vom polnisch-französischen Mathematiker Benoît Mandelbrot gestellt, als er in der Zeitschrift Science seinen Aufsatz „How Long Is the Coast of Britain? Statistical Self-Similarity and Fractional Dimension" veröffentlichte. Zunächst erschien die Beantwortung dieser Frage zwar einfach – schließlich war die Küste Großbritanniens bereits auf zahllosen Karten verzeichnet, wo man nur nachmessen musste. Im Detail erwies sie sich aber als schwierig: Abhängig davon, ab welcher Größe Buchten und Ästuare mitgemessen werden sollen, kann die Gesamtlänge erheblich variieren. Eine möglichst genaue Erfassung müsste dabei möglichst viele der kleinen und kleinsten Abweichungen erfassen, was aufgrund ihrer Vielzahl kaum zu leisten war.

Als Lösung entwickelte Mandelbrot bereits bestehende Ansätze weiter und schuf den Begriff der *Fraktale* (abgeleitet von dem lateinischen Wort *fractus* – gebrochen). Ein Fraktal bezeichnet seitdem Objekte, die zu sich selbst auf unterschiedlichen Größenskalen selbstähnlich sind. Mit anderen Worten: Eine kleine Teilmenge hat eine ähnliche Beschaffenheit wie das große Ganze oder in diesem Fall – ein kleiner Küstenabschnitt ist die verkürzte Variante eines größeren. Geht man ausreichend tief ins Detail, so gelangt man hier zu kleinen, sich ständig wiederholenden Formen: kurzen Stecken gerader und gekrümmter Linien. Da diese sich ständig wiederholen, muss ein Computer nur noch

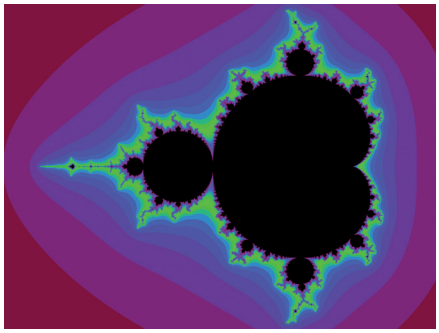

Abb. 1: Eine Mandelbrotmenge (Quelle: Wikimedia Commons/Wolfgang Beyer – CC-BY-SA).

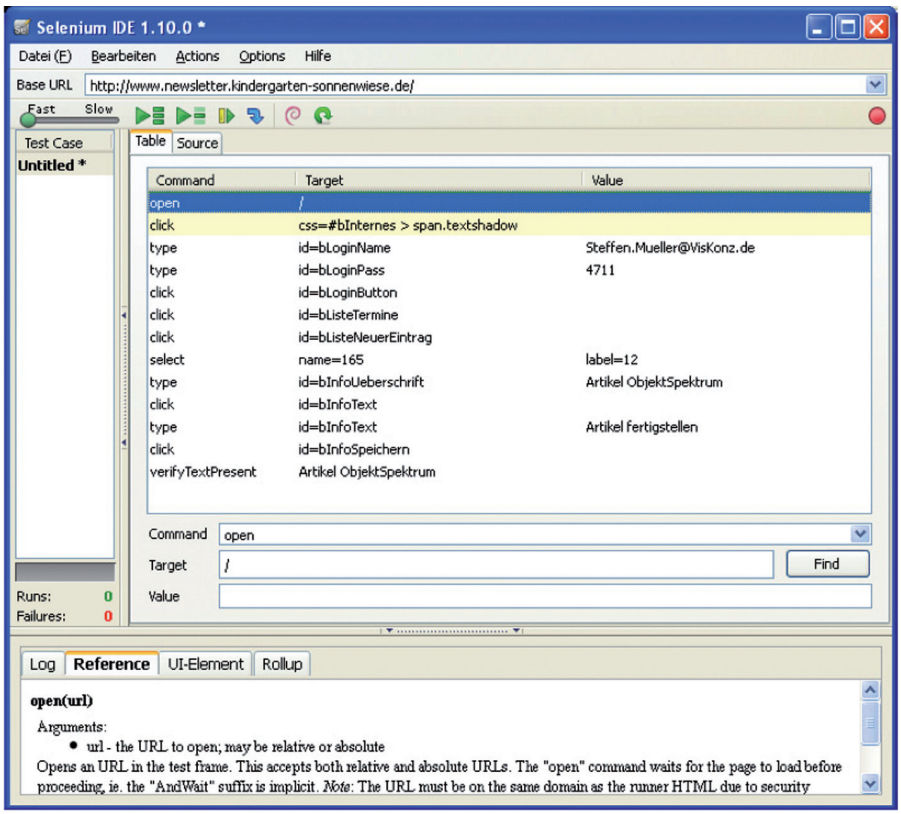

Abb. 2: Aufgezeichneter Testfall in der Selenium-IDE. Man bedient bei eingeschaltetem Rekorder einfach die Anwendung und der Testfall wird aufgezeichnet.

Das Ähnlichkeitsprinzip im Softwaretest

Komplexe Software großflächig automatisiert zu testen, gilt heute allgemein als unlösbare Aufgabe. Was Mandelbrot über die britische Küste sagte, trifft auch hier zu: Auch hier ist nahezu jeder Abschnitt so besonders, dass der „Landvermesser" an seinem zu großen Maßstab zu scheitern droht. Ein detailliertes Eingehen auf alle Besonderheiten „vor Ort" ist mit manuellen Mitteln in der Regel nicht zu schaffen, ähnlich wie beim Streichholzansatz bei der Vermessung der Küstenlinie. Je nach Größe der verfügbaren Testressourcen hält man sich darum in der Regel nur entsprechende Testfall-Entwurfsverfahren vor (z. B. nach dem ISTQB-Zertifikat), um die Software stichprobenartig zu prüfen.

Im Gegensatz dazu verwenden wir nun, ähnlich dem Ansatz von Mandelbrot, Modularisierung und Rechnerleistung, indem wir Testskripte schrittweise in kleinere, sich bei Bedarf auch mehrfach wiederholende Skripte zerlegen und mittels eines Testautomaten auch in großer Menge maschinell ausführen.

Die Testskripte werden als mathematische Funktion begreifbar und auf Korrektheit prüfbar. Diese Funktionen können Software

erfassen, wie häufig jede dieser Formen vorkommt, deren Länge addieren – und die Gesamtlänge ist bekannt. Ohne Computer hingegen wäre dieser Ansatz zum Scheitern verurteilt, es sei denn, man fände genug Freiwillige, um die Küste mit Streichhölzern nachzulegen und diese zu zählen.

Auch bei abstrakteren Problemen lässt sich diese Methode anwenden: In der Mathematik lassen sich mit einfachen Formeln faszinierend ästhetisch aussehende Fraktale generieren. Besonders bekannt ist die so genannte Mandelbrotmenge, bei der die Nähe zur Küstenlinie noch erkennbar ist (siehe Abbildung 1).

Was an dieser Grafik zu erkennen ist: Ohne Computer ist eine Erfassung nicht mehr möglich. Zu komplex sind die Teilmengen der Form, als dass man mit Augenmaß und Zollstock weiterkäme. Bereits an dieser Stelle ist damit der erste Abstraktionsschritt gemacht, weg von der noch sehr greifbaren Aufgabe der Küstenvermessung, hin zu einem höheren Level. Ein weiterer Schritt fehlt aber noch, um zum Hauptthema dieses Artikels zurückzukommen: Wie gelangen wir von Mandelbrot und seinen Formen zum Softwaretest?

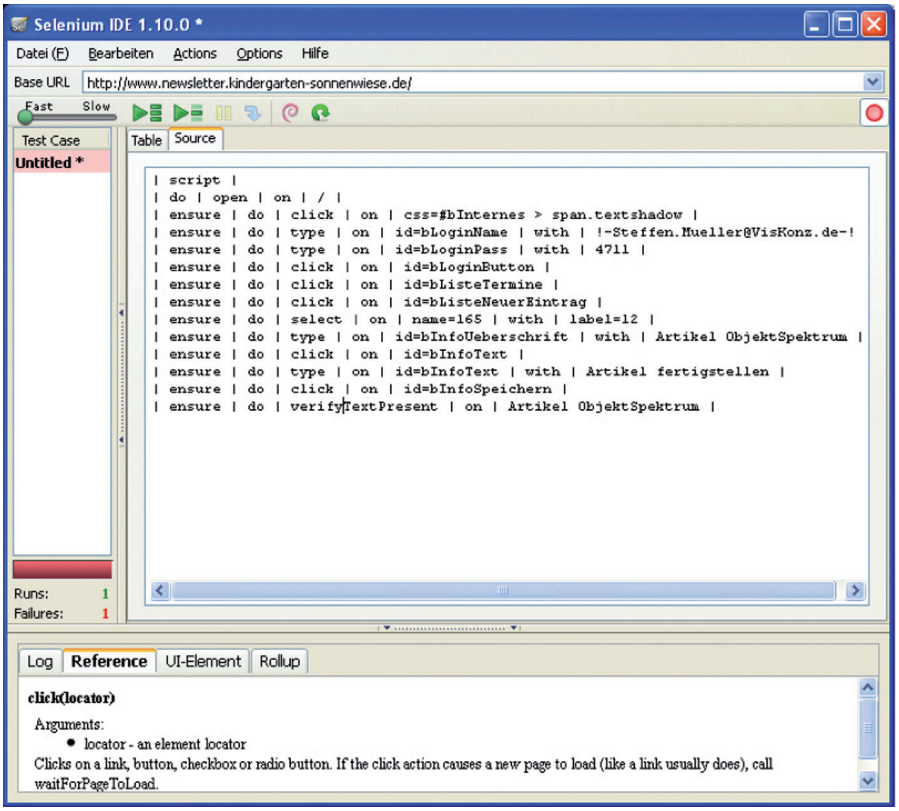

Abb. 3: Der Xebium-Formater portiert die Selenium-Befehle, die sofort in FitNesse lauffähig sind.

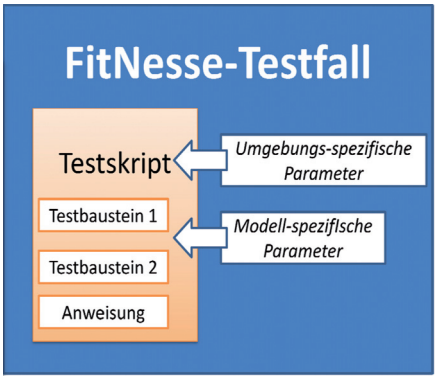

Abb. 4: Die Struktur der FitNesse-Testfälle.

als Sequenz von Eingangs-Stimuli beschreiben, wie zum Beispiel: Ein Nutzer öffnet den Browser (*S1*), gibt in ein Feld Daten ein (*S2*) oder drückt den Login-Button (*S3*). Daraus ergibt sich das Ergebnis (*R*) in folgender Form:

$$f(S1, S2, S3) = R.$$

Durch schrittweise Modularisierung können Tests wieder aus kleineren Tests zusammengesetzt werden und somit neue Funktionen bilden, die schrittweise die Software als Funktion aus Funktionen beschreiben. In diesem Artikel wollen wir ein Open-Source-Werkzeug vorstellen, das das Prinzip der schrittweisen Modularisierung technisch unterstützt und das mit Hilfe der Anwendung von Testfall-Entwurfsverfahren einen interessanten Weg der „Vermessung" von webbasierter Software aufzeigt. Hiermit hilft die Testautomatisierung, Planungssicherheit und eine höhere Testabdeckung zu erzielen.

Ein leichtgewichtiges Automatisierungswerkzeug

Xebium (vgl. [Xeb]) ist ein Open-Source-Werkzeug, das die Vorteile der beiden Vorreiter *Selenium* und *FitNesse* miteinander verbindet und mehr von beiden verspricht. Selenium ist eine bewährte und einfach zu nutzende Testumgebung für webbasierte Regressionstests. FitNesse hingegen hat seine Stärken in der Wartbarkeit von Nutzerakzeptanztests.

Werden Testfälle auf einfache Art mittels Selenium-IDE aufgezeichnet, so können diese Testfälle mittels Xebium ebenso einfach in die leichtgewichtigen Tabellenstrukturen von FitNesse importiert und ausgeführt werden. Hierbei werden die gleichen Keywords wie in Selenium verwendet und die Tests werden mittels Baustein-Library in FitNesse wartbar strukturiert und verfeinert. Durch die einfache Strukturierung (z. B. do | clickAt | id=saveButton) wird eine Testautomatisierung auch für Fachbereiche und Konzepter möglich, selbst wenn dort keine vertieften Programmierkenntnisse vorhanden sind. Auf diese Weise wird ein erhebliches Potenzial geweckt, das bisher in dieses Thema nur selten eingebunden wurde.

Werden die Testbausteine mehrfach benutzt, führt das in der Regel zur Erhöhung der Robustheit und somit auch zu einer Langlebigkeit und damit zu einer Verbesserung der Kosten-/Nutzenbetrachtung.

Über entsprechende Plug-Ins werden die Tests im Jenkins auch für nächtliche Builds, Smoketests etc. interessant. Da in FitNesse-Testfällen mühelos auch SQL-Bausteine, Web-Services oder EJB-Calls Anwendung finden, erweitert sich das Spektrum unterstützter Automatisierungsmöglichkeiten um einen weiteren Baustein – und das ohne weiteren Implementierungsaufwand.

Beispiel

Als auszumessender Küstenausschnitt (zu prüfende Software) soll der Login in einer Kommunikationsplattform eines Kindergartens dienen. Nach der Anmeldung erfolgt die Navigation zur Terminerstellung, zur Anlage eines Termins und zum abschließenden Test der Terminanlage.

Als Testwerkzeug benutzen wir Xebium, von dem seit Dezember 2012 eine neue verbesserte Version zur Verfügung steht. Folgt man der Anleitung unter [Xeb], so ist es wirklich kinderleicht, Testfälle mittels der Selenium-IDE aufzuzeichnen und diese in FitNesse zu importieren (siehe Abbildung 2 und Abbildung 3).

Wer hat diesen Satz noch nicht gehört? „Die Pflege von Testautomaten kostet mehr, als sie einbringt." Das Xebium-Gespann besitzt keine Wizards oder andere magische Elemente, um die Erkennung von Widgets zu unterstützen. Kommerzielle Tools bringen zwar deutlich mehr Komfort mit, haben aber auch einen stolzen Preis. Immerhin ist in die Selenium-IDE eine einfache Erkennung eines Widgets integriert. Jedoch kommt man um ein gewisses Maß an Handarbeit in der Generierung von XPATH-Ausdrücken nicht herum.

Struktur eines Testskripts

Um das Ähnlichkeitsprinzip anzuwenden, muss man zunächst die Struktur der FitNesse-Testfälle verstehen (siehe Abbildung 4). Ein Testfall besteht im Grunde aus einer tabellarischen Struktur, die eine Menge von Keyword-basierten Anweisungen oder auch Testbausteine in eine Reihenfolge bringt. Die Anweisungen können dabei parametrisiert sein, um umgebungs- oder fachmodellspezifische Aspekte abzubilden.

Die Verwaltung der Testbausteine erfolgt in FitNesse in einer so genannten *Szenario-Library*. Es ist empfehlenswert, innerhalb dieser Bibliothek eine Struktur zu pflegen. So könnte es Bausteine für den Zugriff auf einfache oder komplexe Widgets geben und Bausteine, die als Keyword verwendet werden und z. B. die Navigation zu einer bestimmten Stelle in der Anwendung abbilden. Testfall-Szenarios oder auch fachliche Module, die ganze Browserseiten oder Teile davon abbilden, gehören ebenso dazu. Ein Testfall ist somit eine Komposition aus eben diesen Bausteinen und kann seinerseits wieder als Baustein, z. B. als fachliches Modul oder Szenario innerhalb der Szenario-Library, abgelegt werden. Jeder Baustein für sich ist immer in tabellarischer Form abgelegt und beschreibt somit eine Sequenz der Form *S1, S2, …, Sn*. Die korrekte Ausführung dieser Sequenz beschreibt ein konkretes Verhalten (*R*) des Softwaresystems und kann als folgende Funktion interpretiert werden:

$$F(S1, S2, \ldots, Sn) = R$$

Hierbei können *S1, S2, …, Sn* bereits wieder eigene Bausteine sein, also wieder Funktionen.

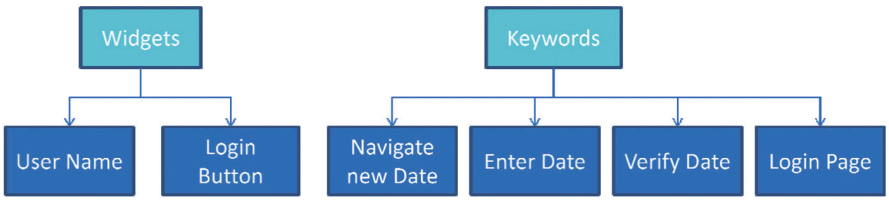

Abb. 5: Beispiel einer Szenario-Library.

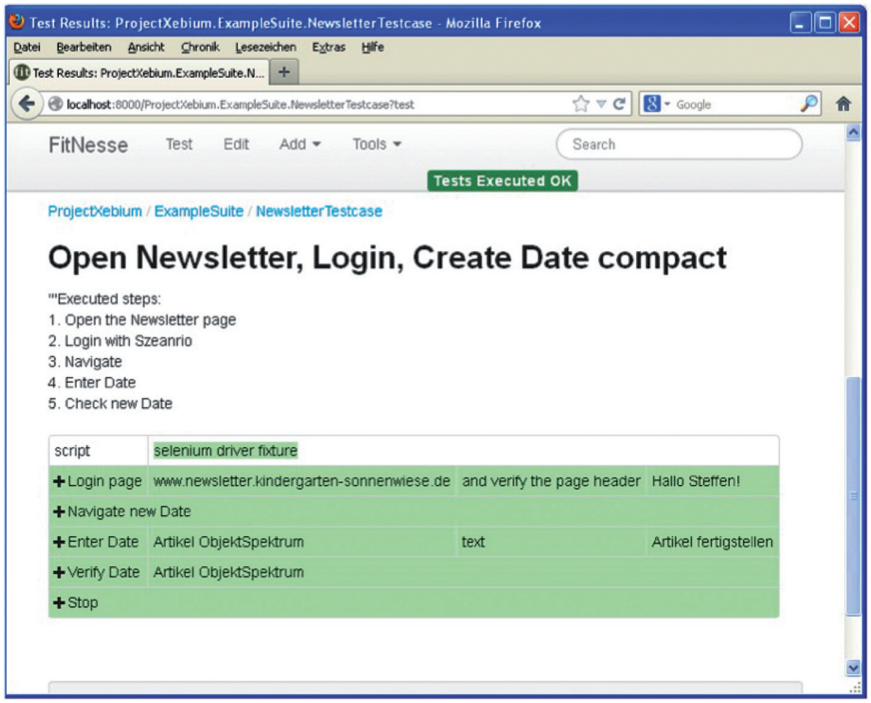

Abb. 6: Screenshot des Fitnesse-Tests.

Beispiel der Szenario-Library

Mit Hilfe der Testbausteine hat der Testfall in FitNesse die in Abbildung 6 gezeigte Struktur. Login, Navigation und Terminerstellung des Kindergartenportals sind nun automatisiert.

Testbausteine schaffen nun ein Netz von Funktionen, mit deren Hilfe leicht Testfälle und somit wieder neue Funktionen erstellt werden können. Ziel muss es sein, eine möglichst vollständige Überdeckung von Testbausteinen zu schaffen, die jedes Testszenario abbilden kann und die Mehrfachverwendung der Bausteine optimiert.

Anwendung von Testfall-Entwurfsverfahren

Mit Hilfe der schrittweisen Modularisierung kann man größere Testskripte aus kleineren ableiten, jedoch wird man mit dem beschriebenen Verfahren kaum mehr Testfälle erstellen und vor allem durchführen können als mit manuellen Methoden. Einen monetären Vorteil wird man erst nach einer größeren Anzahl von Regressionsiterationen spüren – oder ist da noch ein Vorteil?

Um entscheiden zu können, wie gut eine Testfallmenge bezüglich Testabdeckung ist, kann man den Blick auf Testfall-Entwurfsverfahren wie Äquivalenzklasse und Grenzwertanalyse richten:

- *Äquivalenzklassen* fassen Sequenzen von Eingaben der Form *S1, S2, …* zu Klassen zusammen. Ein Repräsentant dieser Klasse verhält sich im Testfall exakt wie alle anderen Elemente dieser Klasse. Somit reicht es, in der Testausführung genau diesen Repräsentant zu beachten, um so die Testfallmenge erheblich zu reduzieren.
- *Grenzwertanalysse:* Spezielle Äquivalenzklassen-Analyse, die auf der Beobachtung beruht, dass Fehler besonders häufig auf den Rändern der Äquivalenzklassen entstehen.

Im oben verwendeten Fall können wir für den Login in die Anwendung zwei fachliche Module Login und DoNotLogin herausarbeiten, mit User und Passwort als Parameter. DoNotLogin liefert immer true, wenn der Login nicht erfolgreich ist.

Mit den Parametern und den fachlichen Modulen kann nun mit Hilfe der Äquivalenzklassen-Betrachtung eine Testfall-Matrix aufgebaut werden, die wiederum als Funktion in FitNesse ausführbar ist. Und die Funktion hat es diesmal in sich (siehe Tabelle 1).

Wenn man jetzt noch aus dem Fachkonzept erkennt, dass bei Eingabe von Datum und Text das Datum nicht in der Vergangenheit liegen darf und ein Text ein Musskriterium für die Anlage ist, so kann man zwei weitere Bausteine mit den Parametern Datum und Text erstellen und eine weitere Testfall-Matrix ableiten (siehe Tabelle 2).

Diese Matrix sollte zudem den Algorithmus zur Terminlage als Funktion vollständig beschreiben. In kurzer Zeit hat man jetzt eine Menge von mehreren Testfällen erstellt. Mehr Testfälle sind an dieser Stelle nicht nötig – somit haben wir eine vollständige fachliche Testabdeckung unseres „Küstenabschnittes“, nämlich des Login-Dialoges und der Terminanlage, erreicht.

Die Ausführung all dieser Tests kostet nur Rechenzeit und keine manuellen Testressourcen je Testzyklus, was im Zeitalter knapper Ressourcen ein echter Vorteil ist.

Nicht immer lassen sich Äquivalenzklassen so einfach wie in unserem Beispiel ableiten – und auch hier wäre elektronische Unterstützung wünschenswert. Diese Testfälle dann auch noch automatisiert zu generieren, ist Teil eines Forschungsprojektes von uns.

Ausblick

Die Vision steht nicht nur am Anfang dieses Artikels, sie steht auch am Ende: die komplette Testabdeckung aller Funktionen einer Software durch eine andere Software: schnell, zuverlässig, unbestechlich und machbar. Ähnlich wie Mandelbrot durch seine Fraktale die britische Küste in Abschnitte einteilen konnte, die eine computergestützte vollständige Längenberechnung möglich machten, ist es uns gelungen, die Testläufe unserer Software so in Module zu zerschneiden, dass sie zusammengesetzt sämtliche Funktionen verifizieren können.

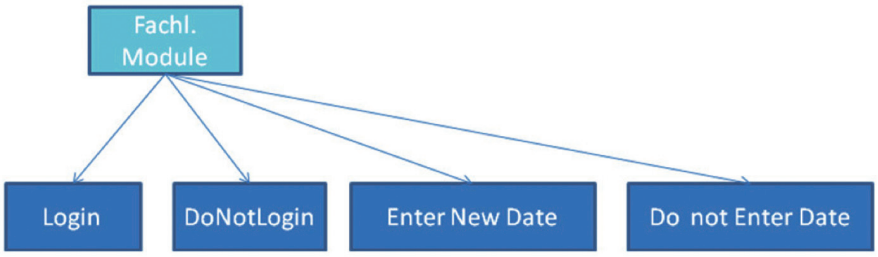

Abb. 7: Erweiterung der Szenario-Library um fachliche Module.

Modul	User	Passwort
Login	Steffen.mueller@viskonz.de	4711
DoNotLogin	xyz	4711
DoNotLogin		4711
DoNotLogin	Steffen.mueller@viskonz.de	xyz
DoNotLogin		xyz
DoNotLogin	xyz	xyz
DoNotLogin	Steffen.mueller@viskonz.de	
DoNotLogin		
DoNotLogin	xyz	

Tabelle 1: Testfall-Matrix für Login.

Modul	Date	Text
Enter New Date	Heute	Artikel fertigstellen
Enter New Date	Morgen	Artikel fertigstellen
Do not Enter Date	Gestern	Artikel fertigstellen
Do not Enter Date	Gestern	
Do not Enter Date	Heute	
Do not Enter Date	Morgen	

Tabelle 2: Testfall-Matrix für Terminanlage.

Wie in unserem Beispiel aufgezeigt, macht dieses an Mandelbrot angelehnte Vorgehen mit den Tools Selenium, Xebium und FitNesse eine vollständige Abdeckung durch automatisierte Tests nicht nur möglich, es ist auch mit vergleichbar geringem Aufwand umzusetzen. Die spannende Frage lautet jetzt: Ist dieser Ansatz wirklich auf jede Situation übertragbar? Wir sind mutig und sagen „ja“. Natürlich wird man in der Praxis immer wieder auf Hürden stoßen und es wird nicht immer notwendig sein, auch die letzte theoretisch mögliche Funktionalität permanent zu verifizieren, aber in den Bereichen, in denen eine vollständige Testabdeckung nötig und wichtig ist, dort ist sie auch machbar. Davon sind wir überzeugt.

Aber auch darüber hinaus bietet unser Verfahren Vorteile: Durch Testautomatisierung mit Xebium aktiviert man verstärkt das fachliche Potenzial in Projekten und Organisationen und schärft das technische Verständnis: Die Testaufzeichnung mit der Selenium IDE macht auch manuelle Tester oder gegebenenfalls sogar Konzepter zu Teilen der Automatisierungsteams – so nutzt man nicht nur deren Know-how, sondern verschafft ihnen auch einen besseren Einblick in den Funktionsumfang der automatisierten Tests. Aus dem in der Praxis oft zu beobachtenden Nebeneinander der Teams wird so ein Miteinander. Und dieses Miteinander kann der entscheidende Faktor dafür sein, dass die automatisierten Tests sich von Anfang an an den fachlichen Vorgaben ausrichten (auch diese scheinbare Selbstverständlichkeit wird in der Praxis oft erst nach wiederholten Refactorings erreicht).

Zuletzt ergibt sich aus den beiden letzten Absätzen noch das vielleicht entscheidendste Argument: Wenn durch unser Vorgehen eine weitgehende Testabdeckung bei vergleichsweise geringen Aufwänden möglich ist und wenn zudem die verbesserte Einbeziehung der Fachlichkeit die Zahl der nötigen Refactorings reduziert, kommt neben dem qualitativen auch ein wirtschaftlicher Aspekt ins Spiel. Wer gegenüber Projektleitung, Controlling und Auftraggeber belegen kann, dass Automatisierung nicht nur zuverlässig, sondern auch wirtschaftlich ist, der macht seine Vision zu der des Gesamtprojekts. Und das sollte auch immer das Ziel sein. ■

Literatur & Links

[Xeb] Xebium, siehe: xebia.github.com/Xebium