Und, funktioniert's?

FitNesse: Theorie und Praxis

Oliver Rohr

Seit es 2003 das erste Mal zum Download bereitstand, erfährt das Meta-Testframework FitNesse in Testkreisen immer größere Beachtung. Dieser Artikel gibt eine kurze Einführung in das Thema und beschreibt, wie FitNesse erfolgreich in Kundenprojekten eingesetzt wurde und welche Praktiken sich dabei als nützlich erwiesen.

Überblick FitNesse

FitNesse [FN] stellt einen recht neuen und innovativen Ansatz dar, automatisierte Regressionstests für beinahe beliebige Softwaresysteme zu erstellen. Die meiste Unterstützung gibt es bei FitNesse derzeit im Java-Umfeld, aber auch andere Programmiersprachen wie C#, C++, Python, Ruby usw. werden unterstützt.

FitNesse und das zugrunde liegende FIT-Framework [FIT] sind Open-Source-Projekte. Wie später noch gezeigt werden soll, unterstützt FIT/FitNesse insbesondere die Erstellung von fachlich orientierten Tests, auch in Zusammenarbeit mit Kundenvertretern und Business-Analysten. Die FitNesse-Installation beinhaltet zu diesem Zweck einen Wiki-Server, der nach dem Entpacken der Installation sofort gestartet werden kann.

FitNesse-Tests werden nun unter Verwendung einer simplen Wiki-MarkUp-Syntax in herkömmlichen Webbrowsern als Wiki-Seiten auf dem Server angelegt, zusätzlich hierzu werden in sogenannten *Fixtures* auf Codeseite Transformationen des Wiki-Contents auf Funktionalität des zu testenden Systems (System Under Test, kurz SUT) geschaffen (s. Abb. 1).

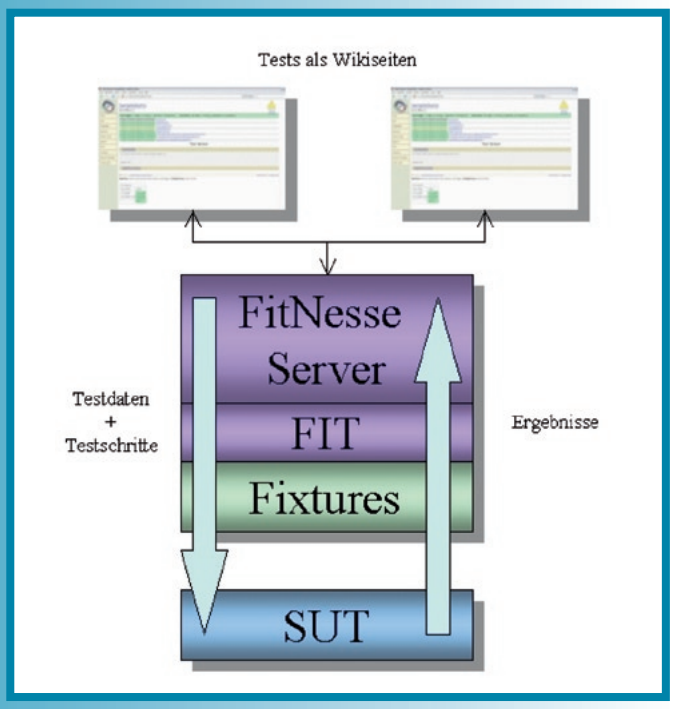

Abb. 1: FitNesse im Zusammenspiel mit dem System Under Test

Die Aufteilung in Tests und Test-Suiten ergibt sich aus der hierarchischen Strukturierung der Wiki-Seiten. Tests und Suiten werden über den entsprechenden Test- bzw. Suite-Button auf der Wiki-Seite ausgeführt. Ein Beispiel eines FitNesse-Tests vor und nach der Ausführung ist in Abbildung 2 zu sehen.

Bei der Ausführung analysiert FitNesse die Wiki-Seite und extrahiert den Inhalt aller Tabellen. Dieser wird dann über Reflection auf die zugrunde liegenden Fixtures gemappt. Hier werden Eingabewerte gesetzt und es wird Funktionalität des zu testenden Systems aufgerufen. Die Ergebnisse der einzelnen Testschritte (actual) werden dann umgekehrt durch die Fixtures mit den erwarteten Resultaten (expected) aus der Wiki-Seite verglichen. Unerwartete Fehler wie etwa Java-Exceptions werden als Stacktrace auf der Ergebnisseite angezeigt.

Im Test in Abbildung 2 wird eine sogenannte `ColumnFixture` verwendet, d. h. – vereinfacht gesagt – alle Spalten ohne ein Fragezeichen im Namen (also `article`, `quantity` und `price`) stellen Eingabewerte dar, Spalten mit einem Fragezeichen führen Prüfungen aus. Bevor die erste Spalte mit einem Fragezeichen auftaucht (in diesem Fall die `sum?`-Spalte), werden in der Fixture Testschritte ausgeführt, wie im konkreten Fall das Hinzufügen eines Artikels in einer gegebenen Menge zu einem gegebenen Preis zum Einkaufswagen (shopping cart).

Als Einstieg in das Thema FitNesse auch in Hinblick auf verfügbare Fixtures lohnt sich ein Blick in die FitNesse-Bibel [Mug05].

Rollen/Kollaboration

Die Erstellung von Testfällen unter FitNesse ist – wie bereits erwähnt – zweigeteilt:

- Fixturecode: Um die Fixtures zu schreiben, werden Programmierkenntnisse in der unterstützten Sprache benötigt.
- Wiki-Content: Um jedoch konkrete Tests im Wiki anzulegen, ist lediglich das Erlernen der FitNesse-Wiki-Syntax Voraussetzung, und das geht recht schnell.

Im Unterschied zu anderen Testtools ist keine neue Skriptsprache o. ä. zu erlernen, im Gegenteil – das Testprojekt kann auf derselben Basis wie die zu testende Applikation entwickelt werden. Wenn beispielsweise das zu testende System auf Java basiert, wird das Testprojekt ebenfalls als Java-Projekt angelegt.

Besonders, wenn Tester eingesetzt werden, die selbst keinen Fixturecode schreiben, ist eine enge Zusammenarbeit mit den „entwickelnden" Testern notwendig. Bei der initialen Erstellung der Tests kann der Wiki-Autor oftmals zusammen mit dem Fixture-Entwickler die Struktur der Testfälle vorgeben. Sehr häufig werden aber auch bereits vorhandene Fixtures in mehreren Tests wiederverwendet. In diesem Fall sollte dokumentiert sein, welche Eingaben möglich sind, welche Aktionen auf dem zu testenden System ausgeführt und welche Resultate geprüft werden können. Dies kann am besten durch entsprechende Template- oder Dokumentationsseiten im Wiki geschehen.

Idealerweise gelingt es, Kundenvertreter und Business-Analysten als weitere FitNesse-Nutzer in die Testfallerstellung einzubinden. Entsprechend der in [Mug05] vorgestellten Vision können so direkt Anforderungen und Akzeptanzkriterien an die zu testende Applikation auf eine Menge von Testfällen abgebildet werden.

Dies ist allerdings keine zwingende Voraussetzung für den Einsatz von FitNesse, auch im Rahmen herkömmlicher Systemtests lässt sich FitNesse mit Erfolg einsetzen.

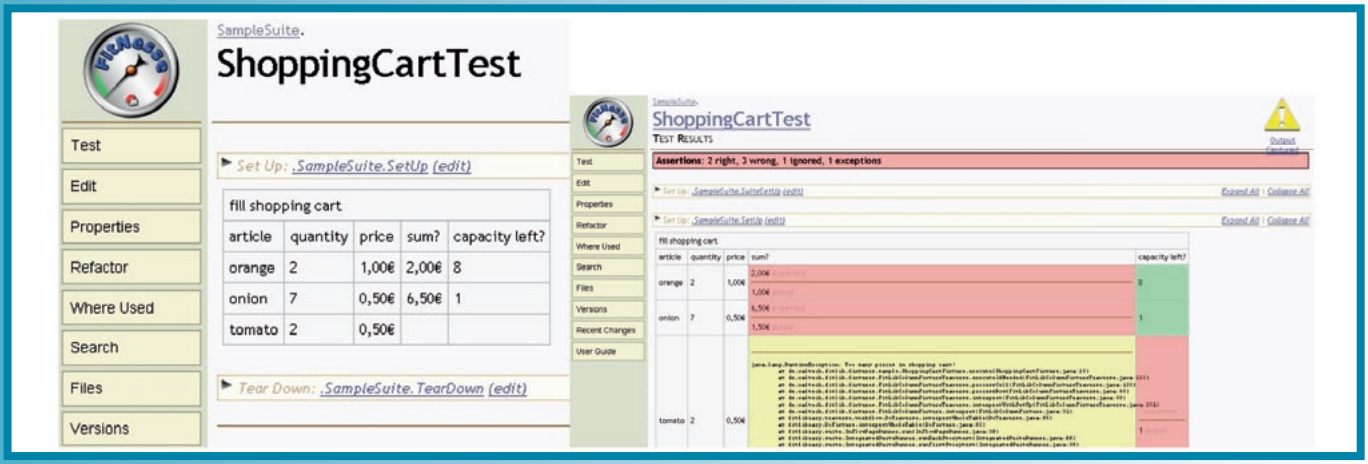

Abb. 2: Ein FitNesse-Test vor und nach der Ausführung – die Wiki-Seite wird bei der Verarbeitung im Ampelcode eingefärbt

FitNesse? Wir machen doch schon Unit-Tests ...

FitNesse kann gut eingesetzt werden, um von den konkreten Technologien des zu testenden Projekts zu abstrahieren. Dadurch ist es möglich, auch Rollen, die mit den konkreten Implementierungsdetails und den eingesetzten Technologien der zu testenden Anwendung nicht so vertraut sind, in die Lage zu versetzen, automatisierte Tests zu erstellen und diese zu nutzen.

Dies hat mehrere Vorteile. Domänenexperten, die vielleicht in einem anderen Testszenario nur indirekt ihr Wissen in die Tests transportieren, können dies so auf direktem Wege tun. Aber auch für die Projektleitung kann es interessant sein, den aktuellen Fortschritt und die Qualität der implementierten Features über den FitNesse-Server abzufragen.

Der Einsatz von FitNesse ist darüber hinaus dann sinnvoll, wenn modulübergreifend auf UseCase-, UserStory- bzw. Requirements-Ebene getestet werden soll. Dies ist z. B. oft bei Blackbox-Tests, die ausschließlich über die von der Software bereitgestellten Schnittstellen arbeiten, der Fall.

Für entwicklernahe Tests ist FitNesse bedingt auf modulübergreifender Ebene geeignet; in einem Szenario also, in dem typischerweise wenige Unit-Tests existieren. Auf Klassen- und Paketebene sind herkömmliche Unit-Tests (z. B. JUnit) völlig ausreichend und der Einsatz von FitNesse würde sich hier aufgrund des Overheads (FitNesse-Server, Wiki-Seiten und Fixturecode im Vergleich zu Unit-Tests) nicht wirklich rechtfertigen lassen.

Andererseits sind Unit-Tests selten der richtige Kandidat, um übergreifende Szenarien aus Anwendersicht zu testen. Zwar ist es technisch möglich, solche Tests zu schreiben, jedoch sind derartige Tests erst einmal nur für Entwickler verständlich. Zudem werden im Testcode viele technische Details sichtbar, die für den eigentlichen Testfall eventuell eine eher untergeordnete Rolle spielen, wohingegen FitNesse-Tests durch die Trennung in Wiki-Content und Fixturecode im Vergleich kompakter dargestellt werden können und besser lesbar sind – der eigentliche Inhalt des Tests, die zu testende Geschäftslogik, tritt in den Vordergrund. Ein weiteres Argument für die im Vergleich vielseitigeren Nutzungsmöglichkeiten von FitNesse-Tests.

Prinzipiell lässt sich mit FitNesse alles automatisieren, was sich auch mit anderen Mitteln automatisieren lässt. FitNesse ist zum Teil als eine Abstraktionsschicht bzw. eine Fassade für verschiedene, andere Testframeworks, die dann spezielle Adapter für zu testende Schnittstellen anbieten (z. B. JFCUnit für GUI-Tests oder ein Webservice-Client für Webservice-Tests usw.), zu verstehen; eine Art Meta-Testframework also.

In der Praxis findet sich in Projekten meist beides, die entwicklernahen Unit-Tests und die übergreifenden FitNesse-Tests. Während Erstere dabei helfen zu testen, dass die Software richtig entwickelt wurde, helfen Letztere dabei zu testen, dass die richtige Software entwickelt wurde. Wie man sieht, befinden sich FitNesse- und Unit-Tests nicht in Konkurrenz zueinander, sondern ergänzen sich. Jeder Testansatz hat seinen bevorzugten Anwendungsbereich, in dem er am besten wirken kann.

Blackbox-Testing mit FitNesse in der Praxis

In einem konkreten Projekt wurde ein Java-basiertes Client-Server-Shopsystem mit FitNesse getestet. Kurz zusammengefasst handelte es sich um einen Swing-Client und einen zentralen Server, der über Webservices erreichbar war und Schnittstellen zu verschiedenen Fremdsystemen bot (s. Abb. 3). In den erstellten FitNesse-Tests wurden Testdaten, Testschritte und Prüfungen fast ausschließlich gegen die von der Client- und Serveranwendung nach außen hin sichtbaren Schnittstellen (GUI, Datenbank, Logging, Batch-Anwendungen, Webservices, Dateischnittstellen) abgesetzt.

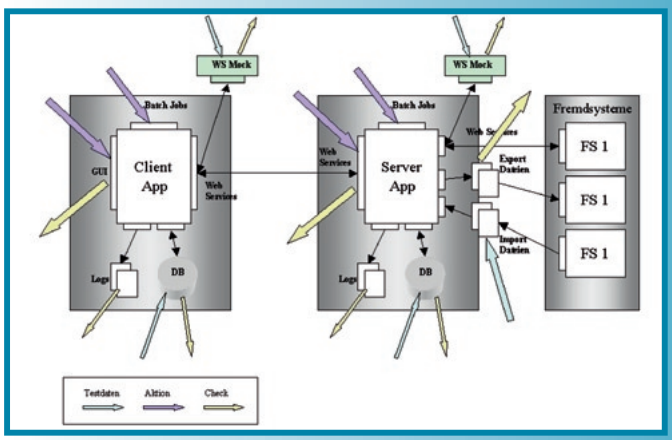

Abb. 3: Abstrakte Architektur der zu testenden Applikation aus Blackbox-Sicht. Interessant für den Systemtest sind insbesondere die vorhandenen Schnittstellen

Dabei wurde in den Wiki-Seiten eine domänenspezifische Anwendersprache verwendet, die auch größtenteils bei der Spezifikation der fachlichen Anforderungen eingesetzt wurde. Die technischen Details der einzelnen Testschritte verschwanden in den Fixtures.

Testdaten

Die verwendeten Testdaten wurden in dem erwähnten Testkontext komplett in die Wiki-Seiten integriert. So liegen z. B. beim Laden von Datenbanktestdaten nicht überall SQL-Dateien im Projekt herum, sondern es werden in den Wiki-Seiten Testobjekte befüllt, deren Inhalt über eine Fixture in die Datenbank geladen wird – ein recht primitives OR-Mapping also, das sich allerdings aus Gründen der Wartbarkeit der Testdaten recht gut bewährt hat.

Auch Import-Dateien, die als Eingabe für die Tests dienen, werden nicht im Dateisystem vorgehalten, da sich hierdurch die Verwaltung aufgrund des Medienbruchs zwischen Dateisystem und Wiki erschweren würde. Die Testdateien werden aus dem Wiki-Content mit einer einfachen Fixture generiert, gegebenenfalls wird zwischen der Domänensprache und den technischen Details der Input-Schnittstelle im Fixturecode übersetzt. Ein Beispiel einer entsprechenden Wiki-Tabelle zeigt Abbildung 4.

Beim Verarbeiten der Fixture wird eine Import-Datei im Dateisystem angelegt, die aus drei Zeilen mit Artikeldaten besteht. Anschließend wird in derselben Fixture über eine Batch-Anwendung der Import der Daten in die Datenbank angestoßen. Aus fachlicher Sicht ist an dieser Stelle nicht relevant, wie genau der Import durchgeführt wird, da in diesem Fall nur Testdaten geladen werden sollen. Details finden sich somit nicht im Wiki-Content, sondern im Fixturecode oder in den Test-Logs, also den Protokolldateien.

Importiere Artikel		
Artikelnummer	Name	Preis
11111111	Test-Artikel 1	100,00 €
22222222	Test-Artikel 2	200,00 €
33333333	Test-Artikel 3	300,00 €

Abb. 4: Zur Initialisierung des Tests wird ein Artikelimport generiert und geladen

Analyse der Protokolldateien

Gerade beim Testen von Backend-Anwendungen ergibt sich oftmals die Situation, dass aufgetretene, unerwartete Fehler nur in den Protokolldateien (engl. log files) sichtbar werden. Um hier deren manuelle Durchsuchung zu vermeiden, werden sie während des Tests mit einfachen Mitteln überwacht. Während des Testlaufs aufgetretene Exceptions oder andere Fehlerausgaben können dann an passender Stelle in der Ergebnisseite angezeigt werden.

In dem angesprochenen Fallbeispiel wurden verschiedene Protokolldateien im Client und Server parallel überwacht. Das war insbesondere bei integrativen Tests zwischen Client und Server recht hilfreich, da Fehlerursachen so recht schnell ermittelt werden konnten.

Search Freezing Compartment

Request		Response?			
best before	01.01.2010	hits	3		
		food	name	category	best before
			pizza	junk food	02.01.2010
			vegetable mix	vegetables *expected* / vegetable *actual*	01.02.2010
			ice cubes *missing*	misc	01.01.2011
			lasagne *surplus*	junk food	05.05.2010

Abb. 5: Format für den Aufruf einer Webservice-Testfixture

Webservices

Um serverseitig bereitgestellte Webservices zu testen, bieten sich verschiedene Möglichkeiten an. Die naheliegendste Möglichkeit ist es vielleicht, SOAP-XML-Schnipsel zum Server zu schicken und die Resultate mit XPath zu prüfen. Dabei ergibt sich das Problem, dass die XML-Schnipsel geeignet parametrisiert werden müssen und stark von der aktuellen WSDL-Datei und den darin referenzierten XML-Schemata abhängig sind. Eine Schnittstellenänderung schlägt in diesem Fall wieder voll in die Wiki-Seiten oder gar ins Dateisystem durch, falls die XML-Testdaten dort abgelegt werden.

Eine mögliche Alternative besteht darin, über ein Webservice-Framework, wie z. B. Apache AXIS, Webservice-Clients zu generieren und diese entsprechend in den Fixtures aufzurufen. Dieser Ansatz wurde in einem anderen Projekt zum Test einer Server-Applikation verfolgt. Dabei waren die Tests im Wiki immer gleichartig mit einer Request-/Reponse-Fixture im `ColumFixture`-Format aufgebaut (s. Abb. 5).

Die FitNesse-Erweiterung *FitLibrary* [FL] lieferte bei der Implementierung gute Unterstützung. Während der Traversierung der geschachtelten Tabellen wird von FitLibrary eine Setter-Methode im Fixturecode aufgerufen, die z. B. einen generierten Request als Parameter erwartet. Über Reflection wird nun automatisch genau so ein Request-Objekt erzeugt und mit Daten gefüllt, indem weitere Setter-Methoden auf diesem Objekt aufgerufen werden. Umgekehrt kann bei der Testausführung die vom Webservice gelieferte Response direkt geprüft werden, da FitLibrary automatisch die im Wiki vorhanden Response-Tabelle auf Aufrufe entsprechender Getter-Methoden im Response-Objekt abbildet.

Die Tests sahen in diesem Fall von der Struktur her so aus wie das Testergebnis in Abbildung 5. Im Detail wurde auf Implementierungsseite eine angepasste FitLibrary-Fixture eingesetzt, die sich nach außen hin wie eine FitNesse-`ColumnFixture` verhält.

Fazit

Bei der Testfallerstellung mit FitNesse lassen sich einige Muster ableiten, die u. a. die Nutzbarkeit, Aussagekraft und Wartbarkeit der erstellten Tests erhöhen können.

Ein nicht selten auftretendes Problem bei der Testfallerstellung ist der mangelnde Bezug der Tests zu den gestellten Anforderungen. Tests sind dann aussagekräftig, wenn sie sich an den konkreten Anforderungen des Projekts orientieren. FitNesse kann die Aussagekraft und die Bindung der Tests an die gegebenen Anforderungen verstärken, wenn eine domänen-

spezifische Sprache eingesetzt wird. Dadurch fällt es leichter zu prüfen, dass nicht nur das zu testende System richtig funktioniert, sondern auch, dass das richtige System entwickelt wird. Abweichungen der zu testenden Funktionalität zu den gestellten Anforderungen werden so schnell sichtbar.

Um die Wartbarkeit und den Nutzen der FitNesse-Tests weiter zu erhöhen, empfiehlt es sich, nur Daten in den Wiki-Seiten zu halten, die auch für den jeweiligen Test von Interesse sind. Weniger ist hier oft mehr: Die Fachlichkeit steht bei den meisten FitNesse-Tests im Vordergrund, während Implementierungsdetails, die mit der eigentlich zu testenden Anforderung nichts zu tun haben, in den Wiki-Seiten ausgeblendet werden können.

Die in diesem Artikel erwähnten Testansätze sind teilweise im Open-Source-Projekt Valtech FitLib [VFL] umgesetzt.

Literatur und Links

[FIT] Framework for Integrated Tests, `http://fit.c2.com/`
[FL] FitLibrary bei Sourceforge,
`http://sourceforge.net/projects/fitlibrary`
[FN] FitNesse-Homepage, `http://www.fitnesse.org/`
[Mug05] R. Mugridge, W. Cunningham, Fit for Developing Software: Framework for Integrated Tests, Prentice Hall, 2005
[Seif09] M. Seifert, Open-Source-Testtools für Java, in: JavaSPEKTRUM, 5/2009
[VFL] ValTech FitLib,
`http://source.valtech.com/display/vfl/Valtech+FitLib`

Oliver Rohr ist als IT-Berater bei der Valtech GmbH tätig. Seine Interessen liegen im Bereich der agilen Softwareentwicklung mit dem Technologieschwerpunkt Java. Seit mehreren Jahren beschäftigt er sich zudem mit dem Thema Qualitätssicherung, insbesondere mit der Entwicklung von automatisierten Regressionstests unter FitNesse.
E-Mail: oliver.rohr@valtech.de.