

Automatisierte Akzeptanztests mit FIT



Whitepaper

Einleitung

Dieses Whitepaper beschreibt, wie man Tests aus Anwender-/Kundensicht mit dem Open-Source-Werkzeug *FIT* beschreibt und durchführt. Das Whitepaper ist für Kunden, Anwender und Entwickler geschrieben. Lediglich der Abschnitt „Für Entwickler: FIT-Tests ans System anbinden“ ist nur für Entwickler geschrieben.

Die Bedeutung von Akzeptanztests

Mit Akzeptanztests¹ wird die Systemfunktionalität aus Sicht der Anwender/Kunden überprüft. In jedem Softwareprojekt werden Akzeptanztests durchgeführt. In den meisten Fällen setzen sich dazu ausgewählte Anwender an das System und testen. Teilweise werden die Tests ad-hoc ohne vorherige Planung durchgeführt. Mindestens bei größeren Systemen ist jedoch eine Planung notwendig, so dass ein Testkonzept erstellt wird.

Das manuelle Testen bedeutet bei den modernen iterativen (agilen) Vorgehensweisen, dass die Anwender sehr häufig testen müssen. Da mit jedem Inkrement vorhandene Funktionalität prinzipiell in Mitleidenschaft gezogen werden kann, müsste eigentlich bei jedem Inkrement das gesamte System getestet werden. Bei Releasezyklen von 1 bis 3 Monaten führt das schnell zu hohen Testaufwänden und zu Frustration bei den Anwendern – wer hat schon Lust, alle 6 Wochen dieselbe Systemfunktionalität zu testen?

Daher wäre es am Einfachsten, wenn Akzeptanztests möglichst weit automatisiert werden. Die sowieso notwendigen Testkonzepte bieten dafür eine geeignete Grundlage, so dass für die Automatisierung kaum Mehraufwand entsteht.

Was ist FIT?

FIT steht für *Framework for Integrated Tests* und ist ein Open-Source-Werkzeug, das einen ganz einfachen Ansatz zur Automatisierung von Akzeptanztests verfolgt. Demnach kann man die Funktionalität eines Softwaresystems vollständig durch Tabellen testen. Jede Zeile einer Tabelle stellt einen Test dar. Die ersten Spalten der Tabelle sind die Eingabewerte für die getestete Systemfunktion und die letzten Spalten stellen die Ausgabewerte des Systems dar.

Als ganz einfaches Beispiel mag ein Taschenrechner fungieren, der nur addieren und subtrahieren kann:

¹ Neben *Akzeptanztest* sind auch folgende Begriffe geläufig: *Funktionstest* – weil die Funktionalität des Systems getestet wird – sowie *Anwendertest/Kudentest* – weil aus Sicht der Anwender/Kunden getestet wird.



Automatisierte Akzeptanztests mit FIT
Whitepaper

Zahl1	Zahl2	Summe	Differenz
3	2	5	1
0	0	0	0

Dass die Spalten „Zahl 1“ und „Zahl 2“ die Eingabewerte und „Summe“ und „Differenz“ die Ausgabewerte der Systemfunktion sind, ist in der Tabelle nicht eindeutig erkennbar. Wir als Menschen können die Rolle der Spalten erraten, aber FIT benötigt einen kleinen Hinweis: Die Ausgabewerte enden mit ().

Zahl1	Zahl2	Summe()	Differenz()
3	2	5	1
0	0	0	0

Ein größeres System hat eine Vielzahl von Systemfunktionen, so dass wir auch noch angeben müssen, welche Systemfunktion überhaupt getestet werden soll. Das tun wir in der ersten Tabellenzelle:

Taschenrechner			
Zahl1	Zahl2	Summe()	Differenz()
3	2	5	1
0	0	0	0

FIT akzeptiert so beschriebene Tests, wenn sie als Tabellen² in einem Dokument (z.B. MS-Word oder Excel) geliefert werden. Text außerhalb von Tabellen ignoriert FIT, so dass zwischen den einzelnen Tabellen Erläuterungen zu den Tests stehen können. Als Ergebnis liefert FIT HTML-Seiten, die den Tests sehr ähnlich sehen. Lediglich die Ergebniszellen wurden eingefärbt: Die Zelle ist grün, wenn das erwartete Ergebnis mit dem berechneten Ergebnis übereinstimmt. Die Zelle ist rot, wenn das berechnete Ergebnis vom erwarteten Ergebnis abwich. Außerdem wird dann der tatsächlich berechnete Wert mit ausgegeben. So entsteht für unser Beispiel dieses Ergebnis:

Taschenrechner			
Zahl1	Zahl2	Summe()	Differenz()
3	2	5	1
0	0	0	0

² Technisch arbeitet FIT auf HTML-Seiten. Diese lassen sich heute ganz einfach mit MS-Word erstellen und pflegen. Der FIT-Aufsatz Fit-Library kann auch direkt Excel-Tabellen verarbeiten.



Automatisierte Akzeptanztests mit FIT
Whitepaper

Tabellen für neue Einsichten

Interessanterweise finden wir sehr schnell Lücken in Tests, wenn die Tests in Tabellenform vorliegen. So ist sofort klar, dass unser Beispielttest keine Aussage darüber macht, wie sich das System bei negativen Zahlen verhält. Das können wir ganz einfach herausfinden, indem wir den Test erweitern:

Taschenrechner			
Zahl1	Zahl2	Summe()	Differenz()
3	2	5	1
0	0	0	0
2	3	5	-1
-1	2	1	-3

Führen wir den Test aus, bekommen wir vielleicht dieses Ergebnis:

Taschenrechner			
Zahl1	Zahl2	Summe()	Differenz()
3	2	5	1
0	0	0	0
2	3	5	Expected: -1 --- Actual: 1
-1	2	Expected: 1 --- Actual: 3	Expected: -3 --- Actual: 1

Aus den angezeigten Fehlern beim Test kann man nicht nur ablesen, dass der Taschenrechner nicht korrekt mit negativen Zahlen umgeht. Man kann außerdem erahnen, wo das Problem liegt: Der Taschenrechner interpretiert auch negative Zahlen immer als positive Zahlen.



Für Entwickler: FIT-Tests ans System anbinden

Wir haben bisher gesehen, wie man FIT-Tests schreiben kann. Es ist aber noch nicht die Frage beantwortet, wie sie an das zu testende System angebunden werden. Woher weiß FIT, welche Methoden an welchen Klassen aufzurufen sind?

Für die Anbindung der Tests an das System werden so genannte *Fixtures* verwendet. Das sind spezielle Klassen, die von passenden Fixture-Oberklassen abgeleitet werden. Die Fixture-Klasse, die wir für oben gezeigtes Beispiel programmieren müssen, sieht so aus³:

```
public class TaschenrechnerFixture extends ColumnFixture {
    public int Zahl1, Zahl2;
    private Taschenrechner _taschenrechner;
    public int Summe() {
        return _taschenrechner.addiere(zahl1, zahl2);
    }
    public int Differenz() {
        return _taschenrechner.addiere(zahl1, zahl2);
    }
}
```

Unser Test soll mit dieser Fixture-Klasse arbeiten und nicht direkt mit der getesteten Klasse *Taschenrechner*. Daher müssen wir unseren Test entsprechend anpassen (in der ersten Zelle der Tabelle muss der Klassenname der Fixture stehen, inkl. Packagenamen):

TaschenrechnerFixture			
Zahl1	Zahl2	Summe()	Differenz()
3	2	5	1
0	0	0	0

³ FIT kann nicht nur mit Java verwendet werden, sondern auch mit .NET, Python, Smalltalk, C++ und Perl. Anbindungen für Ruby, PHP und Objective-C sind in Arbeit.



Automatisierte Akzeptanztests mit FIT Whitepaper

Der Ablauf beim Ausführen des Tests sieht folgendermaßen aus:

1. Man startet den *FileRunner* aus FIT und gibt ihm den Namen der HTML-Testdatei mit.
2. Der *FileRunner* liest die HTML-Datei ein und geht für jede Tabelle folgende Schritte durch⁴:
 - a. Erzeuge ein Objekt der Fixture-Klasse, die in der ersten Tabellenzelle angegeben ist.
 - b. Übertrage je Zeile alle Eingabewerte in die gleichnamigen public Attribute des Fixture-Objektes.
 - c. Rufe je Zeile für alle Ausgabewerte die gleichnamigen parameterlosen public Methoden des Fixture-Objektes auf.
 - d. Vergleiche die Rückgabewerte der aufgerufenen Methoden mit den erwarteten Werten im Test. Markiere die Zelle grün, wenn beide Werte übereinstimmen. Markiere die Zelle rot und schreibe erwarteten und tatsächlichen Wert in die Zelle, wenn die beiden Werte nicht übereinstimmen.

Zusammenfassung

Das Beispiel hat nur einen kleinen Ausschnitt der Möglichkeiten von FIT gezeigt. Über weitere Fixture-Klassen eröffnen sich vielfältige weitere Möglichkeiten, Tests zu formulieren. Die gezeigte Variante mit Column-Fixtures ist aber bereits sehr mächtig. Im Prinzip reicht sie aus, um komplette Systeme zu testen⁵. Das bedeutet insbesondere, dass man FIT-Tests mit sehr wenig Aufwand in sein Projekt einführen kann.

Auch für bereits laufende Projekte eignet sich FIT sehr gut. Es ist beispielsweise sehr viel einfacher, ein ungetestetes System mit FIT-Tests auszustatten als mit Unit-Tests. Mit FIT-Tests kann man mit wenig Aufwand eine sehr hohe Testabdeckung erreichen. So kann man dieses FIT-Test-Sicherheitsnetz benutzen, um notwendige Refactorings an seinem System durchzuführen. Die fehlenden Unit-Tests können dann schrittweise ergänzt werden.

Und sobald die Entwickler FIT im Griff haben, sollten sie den Kunden/Anwendern die Möglichkeit geben, ihre Anforderungen in FIT-Tests zu beschreiben.

⁴ Wir berücksichtigen hier der Einfachheit halber nur Column-Fixtures, wie wir sie in unserem Beispiel verwendet haben.

⁵ Die weiteren Fixture-Klassen erlauben für bestimmte Tests eine elegantere Aufschreibung.



Automatisierte Akzeptanztests mit FIT Whitepaper

Referenzen

1. FIT: <http://fit.c2.com>
 - Web-Site zu FIT mit Beispielen und Download-Möglichkeiten.
2. Rick Mugridge, Ward Cunningham: „Fit for Developing Software“, Prentice Hall, 2005.
 - Englischsprachiges Buch über *FIT*, das auch die FIT-Aufsätze *Fit-Library* und *Fitness* behandelt. Der erste Teil des Buches ist soweit technikkfrei, dass er auch als Einführung in das Thema für Anwender geeignet ist.
3. Frank Westphal: „Testgetriebene Entwicklung mit JUnit & FIT“, dpunkt Verlag, 2005.
 - Entwicklerorientierte Einführung in die testgetriebene Entwicklung mit Schwerpunkt auf Unit-Tests mit JUnit. Am Ende findet sich zusätzlich eine Einführung in FIT und Fitness.

Unser Angebot



Unter der Marke it-agile bieten wir Schulungen und Beratung zu agilen Themen wie testgetriebene Entwicklung, Scrum, eXtreme Programming oder Feature-Driven Development an.

Nähere Informationen finden Sie unter www.it-agile.de

akquinet auf einen Blick

- Individualentwicklung
- Standardlösungen
- Outsourcing
- agiles Vorgehen
- Beratung und Betrieb von SAP- und Navision-Systemen
- Open-Source-basierte Systeme

Weitere Informationen erhalten Sie gerne unter info@akquinet.de