JAVA Mag inkl. CD

Mobile REST-iOS-Apps mit RestKit und Spring MVC ▸78

S&S

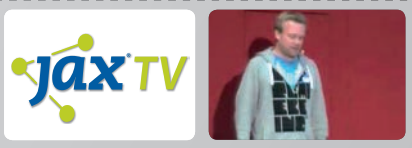
Deutschland €9,80 Österreich €10,80 Schweiz sFr 19,50 Luxemburg €11,15 5.2012

Java™ magazin

Java • Architekturen • Web • Agile

www.javamagazin.de

CD-INHALT



CoffeeScript
von Sebastian Deutsch

Video von der JAX 2011

HIGHLIGHTS


PDFOne


ExtremeFontEngine

WEITERE INHALTE

- XDEV 3
- RestKit
- Weld

Alle CD-Infos ab Seite 3

Datenträger enthält Info- und Lehrprogramme gemäß §14 JuSchG

In-Memory Data Grids
Live long and cluster! Teil 2 ▸96

Testobjekt: Geschäftsprozess
WS-BPEL und BPMN 2.0 ▸48

jax 2012
Programm ab Seite ▸51

WebSocket

Schneller als der Blitz! ▸▸30

WebSocket vs. Server-Sent Events ▸▸36, 42

Interview mit Alexey Melnikov, Koautor des WebSocket-Protokolls ▸▸44

Continuous Delivery
Das Tutorial ▸S. 58

Java EE 6 Security
„Mr. Bond, übernehmen Sie!“ ▸S. 105

Java 7
Fork-Join-Framework, Teil 2 ▸S. 22

Geschäftsagilität durch Tests sichern

Testobjekt: Geschäftsprozess

Die Automatisierung von Geschäftsprozessen mittels WS-BPEL oder BPMN 2.0 steht in der heutigen Zeit oftmals im Fokus der IT-Strategien von Unternehmen. Geschäftsprozesse werden dabei nicht mehr nur fachlich modelliert, sondern so formal beschrieben, dass sie mittels Prozess-Engines ausgeführt werden können. Diese formalen Modelle sind zwar abstrakter und näher an den fachlichen Geschäftsprozessen als normale Programme, gleichzeitig aber auch Softwareartefakte, die während der einzelnen Entwicklungszyklen und auch bei späteren Anpassungsarbeiten getestet werden müssen.

von Daniel Lübke und Tammo van Lessen

Das wichtigste Kapital eines modernen Unternehmens ist das Geschäftswissen, das sich in den Geschäftsprozessen manifestiert. Wendet man die Methoden des Geschäftsprozessmanagements an, kann man dieses Wissen optimieren und gezielter einsetzen. Ein Geschäftsprozess durchläuft dabei einen typischen Lebenszyklus, auch Business-Process-Management-(BPM-)Lifecycle genannt, der sich aus fünf Phasen zusammensetzt. Zunächst wird in der **Analysephase** der Geschäftsprozess identifiziert und anschließend in der **Modellierungsphase** formal dokumentiert. Damit hat man nun den Ist-Prozess, der in der **Ausführungsphase** zu erwarten ist, festgehalten. Im Idealfall ist er auch mit dem erwünschten Soll-Prozess identisch. Ob Soll- und Ist-Prozess tatsächlich übereinstimmen, wird in der **Überwachungsphase** festgestellt, in der nachverfolgt wird, ob sich die beteiligten Personen oder Systeme an das vorgegebene Modell halten. Die dabei gewonnen Informationen sind ein wichtiger Bestandteil für die nächste und letzte Phase des Lebenszyklus: die **Optimierungsphase**, in der das Prozessmodell verbessert wird. In dieser Phase werden außerdem mögliche Veränderungen der geschäftlichen und fachlichen Anforderungen eingearbeitet. Damit beginnt der Zyklus erneut. Das soll als kurze Einführung genügen. Es gibt natürlich verschiedene Arten, um Geschäftsprozesse umzusetzen, auch die Informationen aus der Analysephase können z. B. durch Simulation erhoben werden.

Geschäftsprozesse und IT

Interessant ist das Zusammenspiel zwischen Geschäftsprozessen und der IT. Ihre Aufgabe ist es, den Menschen bei der Abwicklung der Prozesse bestmöglich zu unterstützen. Dazu gibt es unterschiedliche Ansätze: Zum einen kann man bei der traditionellen Softwareentwicklung von der präzisen Erfassung und Dokumentation der Geschäftsprozesse profitieren. Diese Prozesse werden dann als funktionale Spezifikation interpretiert und in „Code gegossen". Das Problem ist allerdings, dass die Turnaround-Zeiten, um Änderungen in den Geschäftsprozessen wieder auf die Software abzubilden, sehr hoch sind und damit Optimierungen aufwän-

dig und teuer machen. Die Alternative stellt der Einsatz einer so genannten *Process Engine* dar. Ihre Aufgabe ist die direkte Interpretation und Ausführung von Prozessmodellen, d. h. sie arbeitet das Prozessmodell Schritt für Schritt ab. Änderungen an den Geschäftsprozessen kann die Engine direkt umsetzen, indem sie für nachfolgende Instanzen des Geschäftsprozesses das aktualisierte Prozessmodell als Ablaufbeschreibung verwendet. Voraussetzung dafür sind Standards wie WS-BPEL [1] oder BPMN 2.0, die sowohl die typische, grafische Modellierung von Prozessen erlauben, als auch eine wohldefinierte Ausführungssemantik festlegen. Zu den Aufgaben einer Process Engine gehört außerdem die Überwachung von Prozessen, d. h. jeder Ausführungsschritt wird protokolliert und kann später zur Analyse verwendet werden. Dadurch ist es möglich, die Geschäftsprozesse kontinuierlich zu verbessern und die Änderungen direkt auszurollen.

Unternehmen versprechen sich von dieser Technologie wesentlich schneller auf Marktveränderungen reagieren zu können. Daneben hat der Einsatz einer Process Engine Einfluss auf die Softwarearchitektur und die Programmierparadigmen. Die festprogrammierte Geschäftslogik weicht einer abstrakten Beschreibung der Abläufe, und man unterscheidet zwischen den Geschäftsprozessen (*Programming in the Large*) und den Geschäftsfunktionen (*Programming in the Small*) [2], also den Diensten, die durch den Prozess aufgerufen werden. Letztere werden traditionell entwickelt und können dank der modularen Struktur einer serviceorientierten Architektur auch gut und mit wenig Aufwand getestet werden. Trotz der vielen Vorteile, die der Einsatz einer Process Engine mit sich bringt, darf nicht vergessen werden, dass auch die von der Engine ausgeführten Prozessmodelle getestet werden müssen. Gerade durch die angestrebte kontinuierliche Optimierung der Prozessmodelle ist auch ein kontinuierliches Testverfahren erforderlich.

Aus einem Projekt in einer (nicht so) weit entfernten Galaxie ...

Es hatte alles so einfach angefangen. Die Prozesserhebung und -dokumentation war zwar aufgrund der vielen Stakeholder aufwändig und kräftezehrend gewesen, aber letztendlich hatte das Projektteam es geschafft, ein einheitliches Verständnis über die Prozesse zu gewinnen. Neben Workshops und Prozessmodellen wurden Prototypen erstellt, Oberflächen entworfen und die fachlichen Vorgänge immer wieder durchgespielt und nachjustiert. Doch bereits nach der Finalisierung und während die ersten ausführbaren Prozessmodelle in der Entwicklung waren, kamen neue Prozessvarianten auf. Der zunächst einfache Prozess hatte auf einmal mehrere Untervarianten und neue Attribute in den Geschäftsobjekten. In Abhängigkeit von den verschiedenen Prozessvarianten wurden Zahlungen zu unterschiedlichen Zeitpunkten ausgelöst und unterschiedliche PDFs versendet. So kämpfte das Projektteam schon vor dem ersten Release damit, neue Funktionen zu implementieren, weil Änderungen in einer Prozessvariante oftmals unerwünschte Nebeneffekte in den anderen Varianten hatten und es dadurch immer wieder zu Folgeproblemen kam. Diese entdeckte man häufig jedoch zu spät, manchmal gar erst im Produktiveinsatz. In Nachfolgeprojekten, in denen die unterschiedlichen Varianten immer weiter ausgebaut worden waren, wurde diese Situation immer prekärer. Die ursprüngliche Intention, schnell neue Prozessaktualisierungen und -änderungen entwickeln und in Betrieb nehmen zu können, verkam damit zu einem scheinbar unerfüllbaren Traum ... Doch wie kam es dazu? Natürlich würde bei einem solchen Projekt sofort BPEL oder BPMN als gescheitert erklärt werden, doch warum konnte das Projekt die eigentlich erwünschte Wartbarkeit und Flexibilität, die man von Prozess-Engines erwartet, nicht nutzen? Die Antwort ist ganz banal: Die ausführbaren Prozessmodelle waren offensichtlich nicht wartbar! Doch warum nicht? Die Antwort liegt versteckt im (Software-)Entwicklungsprozess. Auch ausführbare Prozesse sind unter dem Strich nur eine Software, die entwickelt, gepflegt und getestet werden muss. Gerade wenn hinterher Änderungen gemacht werden, sind automatisierte Tests nötig, die feststellen, ob vormals vorhandene Funktionen noch immer richtig arbeiten. Sind solche Tests nicht vorhanden, wird kein automatisierter Prozess, egal in welcher Technologie, wartbar und leicht anpassbar bzw. weiterentwickelbar bleiben.

Ein Geschäftsprozess könnte dabei beispielhaft wie in **Abbildung 1** aussehen. Der dargestellte Prozess ist stark verkürzt, aber es ist ersichtlich, dass im Laufe der Zeit verschiedene Bezahlvarianten hinzugekommen sind, die teilweise an unterschiedlichen Stellen im Prozess ausgelöst werden. Trotz der neuen Möglichkeit der Zahlung auf Rechnung muss in der Entwicklung sichergestellt werden, dass die alten Bezahlmöglichkeiten weiterhin korrekt funktionieren.

Qualitätssicherung für ausführbare Prozesse ist auch Softwarequalitätssicherung

Im Bereich des Software Engineering (SE) beschäftigt sich ein ganzes Untergebiet mit der Qualitätssicherung von Software. Auch wenn die Prozessmodellierung oftmals nicht als Teil der klassischen Softwareentwicklung gesehen wird, so gelten doch viele Prinzipien ebenfalls in der Prozessmodellierung von ausführbaren Geschäftsprozessen. Die Qualitätssicherung unterteilt sich in drei Kategorien [3]:

- Analytische Qualitätssicherung: Durch Begutachtung der Software bzw. der Prozessmodelle sollen Fehler nach der Entwicklung gefunden werden. Hierunter fallen z. B. Tests und Reviews.
- Konstruktive Qualitätssicherung: Versucht durch geeignete Mittel bereits die Entstehung von Fehlern während der Entwicklung zu vermeiden. Hierunter

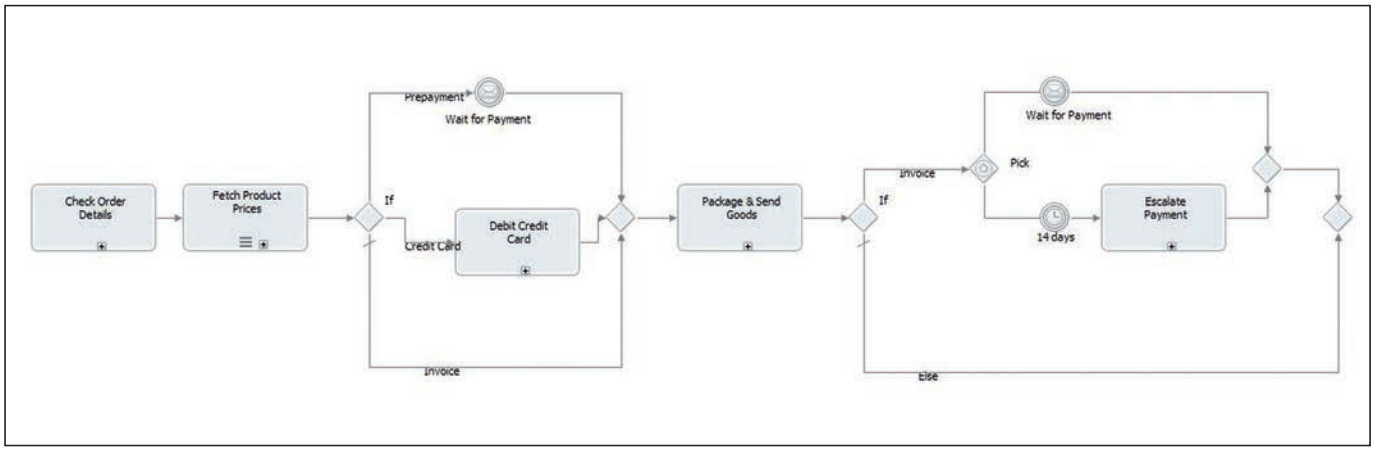


Abb. 1: Exemplarischer Bestellprozess

fallen z. B. Coding Guidelines, aber auch Test-driven Development (TDD) [4].
- Organisatorische Qualitätssicherung: Stellt den organisatorischen Rahmen für die Entwicklung her, z. B. durch Ressourcenallokation und Planung der analytischen Qualitätssicherungsmaßnahmen.

Die konstruktive und die organisatorische Qualitätssicherung sind in der Regel sehr von der Entwicklungsorganisation abhängig. Dagegen sind Verfahren der analytischen Qualitätssicherung (Reviews, Teststrategien, Testheuristiken, Testabdeckung etc.) leicht auf Geschäftsprozessautomatisierungs-Projekte übertragbar.

Was heißt das für Programming in the Large?

Prinzipiell sind alle Verfahren aus der klassischen Softwarequalitätssicherung auf die Entwicklung ausführbarer Geschäftsprozesse mit BPEL oder BPMN 2.0 übertragbar. In diesem Artikel wollen wir uns allerdings auf das Testen beschränken und die Besonderheiten beim Testen von ausführbaren Geschäftsprozessen inklusive immanenter Eigenheiten näher beleuchten.

Abb. 2: Facettenklassifikation

	Zahlmethode					Artikel			...
	Kreditkarte	Vorkasse	Rechnung	Nachnahme	nicht angegeben	1 Artikel	mehrere Artikel	kein Artikel	
TC 1	x					x			
TC 2		x					x		
TC 3			x			x			
TC 4				x			x		
TC 5					x	x			
TC 6	x							x	

Zum einen sind Prozesse typischerweise langlaufend [5]. Timer und lange Timeouts sind aber schwer zu testen. So ist es z. B. herausfordernd, eine Mahnung zu testen, die nach vier Wochen Zahlungsverzug ausgestellt werden soll. Hier muss der Prozess entweder konfigurierbar gemacht oder speziell für das Testen geändert werden. Beides birgt die Gefahr, dass sich das Verhalten des Prozesses ändert und dadurch Fehler im Originalprozess nicht gefunden werden. Insbesondere bei manuellen Änderungen und Deployments ist dies der Fall. Um solche Fehlerquellen zu minimieren, erscheint es sinnvoll, Veränderungen automatisiert und während des Testens transparent für den Entwickler vorzunehmen. So wird sichergestellt, dass keine Spuren im Prozess, die durch manuelle Änderungen verursacht wurden, quasi beim Rückbau vergessen werden. Ein weiteres Problem bei der Entwicklung und insbesondere beim Testen ist die Tatsache, dass der Prozess nur zusammen mit den ihn umgebenden Komponenten, typischerweise Web Services, funktioniert. Davon werden einige zusammen mit dem Prozess entwickelt, aber erst später fertig. Somit stellt sich das Problem, wie möglichst frühzeitig der Prozess getestet werden kann, wenn noch nicht alle einzelnen Komponenten bereit für die Testphase sind.

Hierbei kommen Mocks, d. h. simulierte Ersatzservices für den Test, zum Einsatz und stellen eine wertvolle Ergänzung dar. Dadurch, dass die Web Services über WSDL-Schnittstellen sehr gut vom Prozess entkoppelt sind, ist es leicht möglich, Mocks zu schreiben und sogar zu großen Teilen zu generieren. Über die Deployment-Deskriptoren des Prozesses können die Endpunkte leicht an die verschiedenen Test- und Produktiv-Umgebungen angepasst werden. Da BPEL- und BPMN-Prozesse über maschinenlesbare Schnittstellen mit ihrer Umwelt kommunizieren, können die Tests ebenfalls einfach automatisiert und somit häufig und effizient wiederholt werden. Das ist insbesondere bei Fortführung des BPM-Lifecycles wichtig. Wird nun der Prozess weiterentwickelt, können die Testsuites ggf. angepasst und wieder ausgeführt werden. So ist das Re-Testing deutlich einfacher und schneller. Gerade diese erhöhte Geschwindigkeit erlaubt kürzere Releasezyklen und dadurch die schnellere Umsetzung von Features. Doch wie kommt man an einen guten Satz von Tests? Auch hier gibt es aus der Softwarequalitätssicherung viele Ansätze, die man auch für Geschäftsprozesse nutzen kann. So können Äquivalenzklassen und Klassifikationsbäume [3] zur Strukturierung der Parameter verwendet werden, und es gibt für Geschäftsprozesse ebenfalls Testabdeckungsmetriken (z. B. [6]). Wie kann man den Prozessablauf nun mittels Klassifikationsbäumen strukturieren und daraus Testfälle ableiten? Zu diesem Zweck müssen die Stellen, an denen der Kontrollfluss verzweigt, lokalisiert und als Prozessvarianten aufgenommen werden. Dazu zählen insbesondere Schleifen, bei denen wir keinen, einen und mehrere Schleifendurchläufe als Varianten in den Baum aufnehmen. Für unseren Beispielprozess könnte ein solcher Baum wie in **Abbildung 2** aussehen. Mit diesen Eigenschaften bilden wir Testfälle. Dabei verknüpfen wir verschiedene Eigenschaften miteinander zu Testfällen. Maximal ein Fehler darf in einem Testfall überprüft werden. Beispielhaft sind hier verschiedene Testfälle angegeben.

Neben Tests bieten viele Entwicklungsumgebungen auch Simulationswerkzeuge an. Diese können in der Entwicklung benutzt werden, um Prozessabläufe zu verifizieren, ersetzen aber Tests nicht vollständig. Das liegt daran, dass für die Prozessausführung nicht nur der Prozess, sondern auch dessen Deployment Descriptor, das BPMS, das Betriebssystem, die benutzten Services und viele andere technische Komponenten benötigt werden, die wiederum Fehler beinhalten können. Diese Fehler können nur durch Tests auf einem BPMS, nicht aber in der Simulationsumgebung der Entwicklungsumgebung gefunden werden.

Tools nach Konzeption

Nachdem die Testfälle konzeptionell erstellt wurden, müssen diese automatisiert werden. Hierzu gibt es eine Reihe verschiedener Werkzeuge, wie z. B. soapUI (http://www.soapui.org) oder BPELUnit (www.bpel-unit.net). Diese können die SOAP-Nachrichten an den Prozess schicken und stellen auch einfache Möglichkeiten zur Verfügung, Services zu simulieren. BPELUnit und Herstellertools können daneben zusätzlich Prozesse transparent für den Test deployen, die Testabdeckung messen und ggf. alte Prozessinstanzen von den Testservern entfernen.

Als Teil der konstruktiven Qualitätssicherung sollten diese Tools in den Build-Prozess integriert werden. Automatisierte Builds, die den aktuellen Stand aus der Versionsverwaltung auschecken, die Testfälle ausführen und Deployment erstellen, zentrale Build-Server und Nightly Builds schaffen ein Umfeld, das auch bei der Geschäftsprozessentwicklung die Prozessdesigner unterstützt und Fehler früh sichtbar macht.

Prozessfluss vs. Datenfluss

Bisher haben wir uns vor allem auf den Prozessablauf konzentriert. Die Testfälle orientieren sich, egal ob aus

Blackbox- oder Whitebox-Sicht, am Prozessfluss und testen die mit der Umwelt ausgetauschten Nachrichten. Zusätzlich gibt es Artefakte, die von den Prozessen benutzt werden. Hier sind insbesondere Datentransformationen wie XSLT- und XQuery-Skripte zu nennen. Für diese Skripte ist es sinnvoll, eigene Unit Tests zu schreiben, um nicht bei jeder neuen Eingabedatenkombination den gesamten Prozess ausführen zu müssen. Dabei ist es vorteilhaft, wenn die Skripte extern vom Prozess gespeichert und nicht inline definiert sind, sodass sie wie die verwendeten Services separat getestet werden können. Für XSLT- und XQuery-Skripte empfiehlt es sich, beispielsweise JUnit zu nehmen, das einen XSLT-Prozessor aufruft und das Ergebnis mit einer Soll-XML-Datei vergleicht.

Was Sie mitnehmen sollten

Dieser Artikel soll Sie motivieren, Ihre ausführbaren Prozesse automatisiert zu testen. Automatisierte Tests helfen Ihnen, Änderungen an Ihren Prozessen agil und schnell umsetzen und trotzdem möglichst fehlerfrei ausliefern zu können. Insbesondere bei unternehmenskritischen Kernprozessen kann eine fehlerhafte Anpassung enorme Kosten verursachen. Wir selbst haben leider miterleben müssen, welche Probleme im Laufe der Zeit entstehen können, wenn Prozessmodelle nicht konsequent getestet werden. Deshalb ist es uns ein Anliegen, für dieses Thema zu sensibilisieren.

Dr. Daniel Lübke ist Senior Consultant bei der innoQ Schweiz GmbH. Er ist aktuell in SOA- und Prozessautomatisierungsprojekten für Kunden tätig, ist Maintainer des BPELUnit Frameworks und Mitautor des Buches „Geschäftsprozesse automatisieren mit BPEL“ (dpunkt.verlag).

Tammo van Lessen ist Senior Consultant bei der innoQ Deutschland GmbH und ist dort in SOA- und Prozessautomatisierungsprojekten für Kunden tätig. Er ist PMC Chair der BPEL-Engine Apache ODE, wirkte an der BPMN-2.0-Spezifikation mit und ist Mitautor des Buches „Geschäftsprozesse automatisieren mit BPEL“ (dpunkt.verlag).

Links & Literatur

[1] van Lessen, Tammo/Lübke, Daniel/Nitzsche, Jörg: „Geschäftsprozesse automatisieren mit BPEL“, dpunkt Verlag, 2011

[2] Leymann, Frank/Roller, Dieter: „Production Workflow – Concepts and Techniques“, Prentice Hall PTR, 2000

[3] Schneider, Kurt: „Abenteuer Softwarequalität: Grundlagen und Verfahren für Qualitätssicherung und Qualitätsmanagement“, dpunkt Verlag, 2007 (Neuauflage erscheint 2012)

[4] Lübke, Daniel/Moser, Simon/van Lessen, Tammo: „Open-Source-BPEL-Orchester Teil 2: Proben im BPEL Orchester“, in: Java Magazin 2.2010, Januar 2010

[5] Smith, Adam: „Wohlstand der Nationen“, Anaconda, 2009

[6] Salnikow, Alex: „Ermittlung von Testabdeckungsmetriken in BPEL“, Masterarbeit in dem Fachgebiet „Software Engineering“ der Leibniz Universität Hannover, 2007: http://www.se.uni-hannover.de/pub/File/pdfpapers/Salnikow2007.pdf