Business Technology

www.bt-magazin.de 2.2013 Heft 13 Deutschland: € 14,90 Österreich: € 16,80 Schweiz: sFr 29,80 Luxemburg: € 16,90

Business Technology

Architektur, Innovation & Strategie

Norm Judah: „Zukunft als Gegenwart“

ROBUST und flexibel

Baukasten für Fachleute

Agiles Architektur-management

Technische Schulden

4 191858 414906 02

Blaupause für ein fachliches Anwendungsframework

Baukasten für Fachleute

Durch Produktinnovationen und regulatorische Anforderungen sind kontinuierliche Anpassungen an den fachlichen Prozessen eines Unternehmens notwendig. Wie muss eine Fachapplikation beschaffen sein, um die Anwender mit diesem Tempo schritthalten zu lassen und nicht im Chaos zu enden?

AUTOREN: KONSTANTIN DIENER UND DANIEL KAMINSKY

Der vorliegende Artikel beschreibt zunächst die Motivation für eine flexible Fachapplikation, beginnend mit den Auslösern, hin zu den gewünschten Anforderungen. Anschließend werden die wesentlichen Bausteine vorgestellt und technische Rahmenbedingungen beleuchtet. Den Abschluss bildet die Vorstellung des fachlichen Anwendungsframeworks. Im nächsten Heft wird dieses Framework um eine prototypische Technologieauswahl ergänzt und anhand eines fachlichen Beispiels vertieft.

BESTÄNDIGER WANDEL DER UMWELT …

Die Wettbewerbsfähigkeit heutiger Unternehmen hängt von zwei wichtigen Faktoren ab. Erfolgreiche Unternehmen erfinden sich permanent neu, indem sie durch Innovationen ihre Produkte erweitern oder verbessern. Die kontinuierliche Innovation ist der wichtigste **interne** Auslöser für Veränderungen. Zusätzlich zu den internen Erfordernissen, sich zu verändern, sind die Unternehmen mit einer wachsenden Anzahl gesetzlicher Anforderungen mit festen Umsetzungsfristen konfrontiert. Hiervon ist vor allem die Finanzbranche betroffen. Aktuell befinden sich bspw. vierzehn Regulierungsvorhaben für Kapitalanlagegesellschaften im Gesetzgebungsprozess, Anfang 2011 waren es nur fünf [1] (**externe** Auslöser).

… ERFORDERT KONTINUIERLICHE ANPASSUNGEN …

Jede intern oder extern ausgelöste Änderung bedingt die Anpassung der Unternehmensprozesse. Da mittlerweile die meisten Prozesse IT-gestützt sind, führen Prozessänderungen in der Regel auch zu Änderungen an den Fachapplikationen. Diese Applikationen müssen **flexibel** sein, das heißt in erster Linie die Umsetzung von Änderungen ermöglichen. Zusätzlich ist relevant, wie schnell eine Änderung durchführbar ist: Steht beispielsweise eine Funktion erst weit nach dem gesetzlich vorgeschriebenen Stichtag zur Verfügung, ist der Nutzen gering oder gar nicht vorhanden. Verallgemeinert gesprochen, ist eine Fachapplikation erst dann flexibel, wenn mit nur minimalem **Aufwand** der durch die Änderung angestrebte **Nutzen** realisiert werden kann.

Bei agilen Vorgehensweisen sind kontinuierliche Änderungen willkommen. Allerdings sind sie in den meisten Fällen auf das Entwicklungsprojekt beschränkt, dessen Anteil am Gesamtaufwand für eine Anwendung in der Regel zwischen zehn und zwanzig Prozent darstellt. Für jede Änderung ein neues, agiles Projekt aufzusetzen, ist oft zu aufwändig.

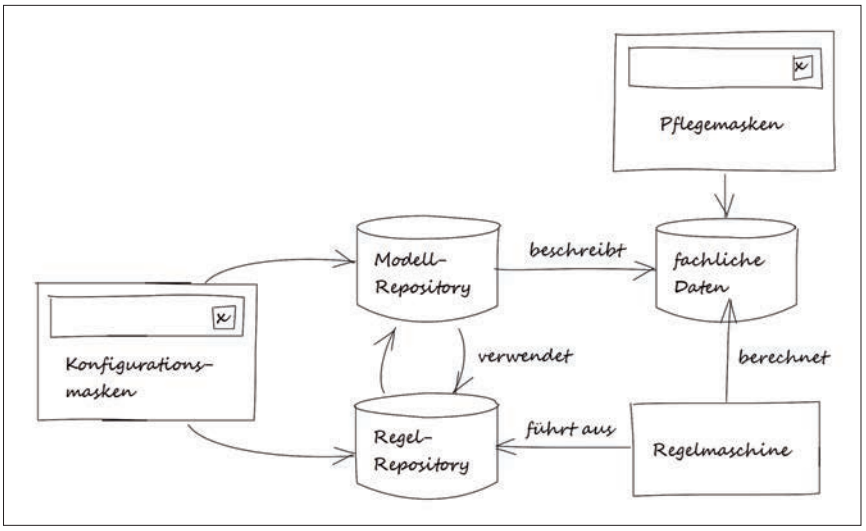

Abb. 1: Architekturblaupause

… JENSEITS KLASSISCHER UMSETZUNG

Der klassische Ablauf für eine Änderung beginnt damit, dass eine Fachabteilung einen Änderungsbedarf identifiziert (intern oder extern) und beschreibt. Die IT leitet ein technisches Konzept ab, implementiert die Änderung und nimmt sie in Betrieb.

Insgesamt führt dieses Vorgehen dazu, dass die fachliche Anforderung durch die IT in eine technische Beschreibung „übersetzt" wird. Diese Übersetzung erzeugt **Transformationskosten** und ist in der Regel sowohl fehler- als auch verlustbehaftet. Zusätzlich fällt noch die nicht fachliche Grundlast an: Planung und Durchführung von Konzeption und Implementierung sowie verschiedener Testzyklen (meist mit Kapazitäten aus der Fachabteilung). Schließlich ist die Produktivsetzung mit Betrieb und Fachbereich abzustimmen und durchzuführen. All diese Punkte lassen sich als **Transaktionskosten** zusammenfassen.

FLEXIBLE FACHAPPLIKATIONEN

Um das Ziel zu erreichen, Änderungsanforderungen in minimaler Zeit umzusetzen, müssen die Transformations- und Transaktionskosten so stark wie möglich reduziert werden. Die geringsten Kosten entstehen, wenn auf dem Weg von der Identifikation des Änderungsbedarfs beim Fachanwender bis zur Inbetriebnahme möglichst wenig Transformations- und Abstimmungsschritte passieren. Im Extremfall heißt das, dass der Fachanwender die Anwendungslogik konfiguriert und die Änderungen selbst durchführt.

Wenn die Anwendungslogik nicht fest in der Anwendung hinterlegt ist und der Anwender sie selbst konfiguriert, führt dies erst einmal dazu, dass die Anwendung weniger stabil ist. Zum Vergleich: Wären bspw. alle Elektrogeräte in einer Wohnung über eine feste Kabelverbindung mit der Wand verbunden, könnten sie nicht im Raum bewegt werden (unflexibel). Ersetzt man die festen Kabelverbindungen durch Steckverbindungen (flexibel), steigt das Risiko, dass durch ein falsches Zusammenstecken ein Kurzschluss ausgelöst und der gesamte Stromkreis unbrauchbar wird.

Flexibilität geht also mit dem Potenzial einher, Dinge „falsch" zu machen bzw. die Elemente oder Bausteine falsch zu kombinieren und damit die Stabilität des Gesamtsystems zu gefährden. Es gibt zwei Möglichkeiten, dieser Auswirkung zu begegnen:

- Fehler vermeiden: In dem Kabelbeispiel würde man die Steckverbindungen so realisieren, dass sie nur „richtig" zusammengesteckt werden können (Beispiel: USB). Syntaktische oder strukturelle Fehler können durch „Hineinkonstruieren" entsprechender Mechanismen vermieden werden.
- Fehler aufdecken: Wenn der Kunde zum Föhnen eine Lampe verwenden möchte (was nicht funktioniert), kann dieser Fehler nicht durch eine Konstruktion verhindert werden. Diese semantischen oder inhaltlichen Fehler lassen sich nur durch frühe und regelmäßige Kontrollen (Reviews/Tests) aufdecken.

Die beiden Aspekte „Fehler aufdecken" und „Fehler vermeiden" sind wesentliche Bestandteile des Toyota Production Systems (Kasten: „Poka Yoke").

ARCHITEKTURBLAUPAUSE

Für eine flexibel anpassbare Fachapplikation möchten wir gerne eine Architekturblaupause vorstellen. Dazu soll zunächst im Fokus stehen, aus welchen Bausteinen (Steckverbindungen) der Fachanwender die Anwendungskonfiguration zusammenstellt. Für eine erste grobe Idee bedienen wir uns der objektorientierten Modellierung. Danach besteht ein Softwaresystem aus Daten und Operationen. Im ersten Schritt gehen wir deshalb davon aus, dass die Blaupause aus einem Domänenmodell zur Beschreibung der fachlichen Daten und den Geschäftsregeln (Operationen) besteht (**Abb. 1**). Jede fachliche Konfiguration ist jeweils einer dieser beiden Kategorien zuordenbar.

DOMÄNENMODELL

Das Domänenmodell enthält eine Beschreibung der relevanten Elemente einer Domäne (Wertpapier, Kurs, Trans-

aktion), deren Eigenschaften bzw. Attributen (ISIN, Name, Kurswährung, Börsenplatz des Kurses) sowie den Beziehungen zwischen den Elementen („Ein Kurs gehört genau zu einem Wertpapier."). Für die Realisierung des Domänenmodells gibt es die folgenden Anforderungen, die in das abstrakte Modell in **Abbildung 2** münden:

- **Struktur konfigurierbar:** Die Domänenelemente, -attribute und Beziehungen kann der Fachanwender zur Laufzeit in der Anwendung hinzufügen, verändern oder löschen.
- **Lebenszykluskonzept:** Im Sinne der Fehlervermeidung wird sichergestellt, dass durch das Hinzufügen, Verändern oder Löschen von Elementen des Domänenmodells die auf Basis des bisherigen Modells gespeicherten Daten nicht ungültig werden.
- **Aufbau komplexer Strukturen:** Der Fachanwender kann komplexe Strukturen durch Verschachtelung in seinem Domänenmodell aufbauen (bspw. „Ein Wertpapier enthält eine Menge von Kursen.", siehe auch **Abb. 2**). Für die Beziehungen/Verschachtelungen legt der Fachanwender Kardinalitäten fest.
- **Verknüpfung von Attributen:** Die Werte für einige Attribute des Domänenmodells werden aus den Werten anderer Attribute abgeleitet (Geschäftsregeln). Das Volumen einer Wertpapiertransaktion ergibt sich bspw. aus der Anzahl (Stücke) und dem Kurs des gehandelten Wertpapiers sowie möglicherweise einem Devisenkurs.
- **Struktur und Inhalt gemeinsam abgelegt:** Sowohl die Struktur selbst als auch die darauf basierenden Daten werden gemeinsam in der Anwendung abgelegt.

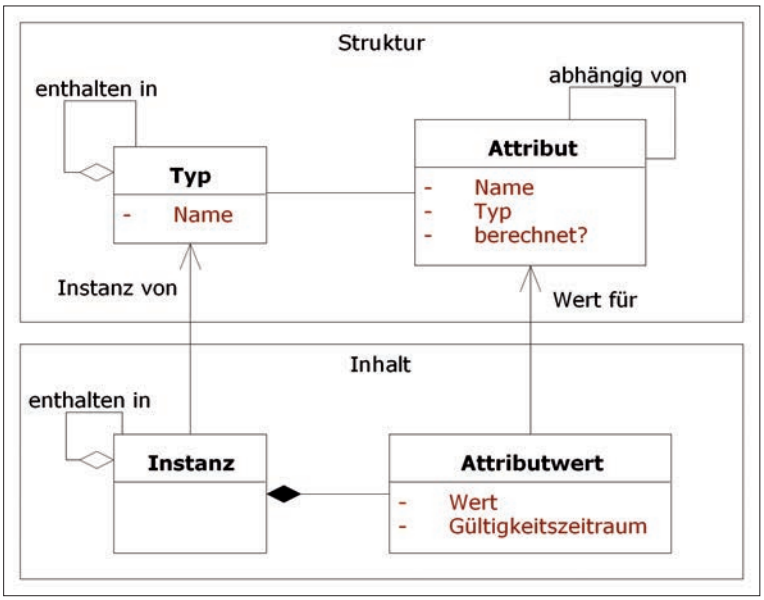

Abb. 2: Domänenmodellierung

Die Konfiguration des Domänenmodells führt der Fachanwender über ein entsprechendes Pflege-Frontend entweder textbasiert oder mittels eines grafischen Editors durch.

Für die technische Umsetzung eines den dargestellten Anforderungen genügenden Domänenmodells in der Fachapplikation eignen sich bspw. Data-Dictionary-basierte Datenmodelle, NoSQL-Datenbanken oder Eclipse Ecore [3] im Zusammenspiel mit Teneo [4]. Die letztgenannte Option hat den Vorteil, dass für Ecore im Eclipse Modelling Project bereits textuelle und grafische Editoren zur Verfügung [5] stehen.

GESCHÄFTSREGELN

Die Logik der Fachapplikation wird in Form von Geschäftsregeln konfiguriert. Diese Regeln können aus Berechnungs- und Entscheidungselementen (bspw. Wenn-Dann-Sonst) bestehen. Die Architekturblaupause in **Abbildung 1** enthält zwei Komponenten zur Ablage (Regel-Repository) und Ausführung (Regelmaschine) der Regeln. Für diese Komponenten ergeben sich u. a. die folgenden Anforderungen:

- **Regeln konfigurierbar**: Der Benutzer kann die Geschäftsregeln zur Laufzeit der Anwendung anlegen, verändern oder löschen.
- **Bereitstellung von Funktionen:** Für die Formulierung von Regeln bringt die Anwendung einen Satz von allgemeinen und domänenspezifischen Entscheidungs- und Berechnungsfunktionen in Form einer domänenspezifischen Sprache (DSL) mit (**Abb. 3**).
- **Berechnung von Attributwerten:** Der Wert eines Attributs aus dem Domänenmodell kann nicht nur durch die Anwender vergeben, sondern alternativ auch durch eine Geschäftsregel bestimmt werden.
- **Referenzierung des Domänenmodells:** Zur Formulierung von Regeln verwendet der Fachanwender Elemente und Attribute aus dem Domänenmodell. Die Regel *Transak-*

Poka Yoke

Sobald eine Möglichkeit für Modifikationen geschaffen wird, entsteht auch das Potenzial für Fehler. Diese unbeabsichtigt eingebauten Fehler dürfen nicht ins endgültige System oder Produkt gelangen, denn dort ist der Aufwand am größten, sie zu beheben.

Poka Yoke (dt. „unglückliche Fehler vermeiden", nach [2]) ist die Idee, konstruktionsbedingt diese Fehler zu verhindern. Die Maßnahmen müssen technischer Natur und leicht umzusetzen sein. Aufwändige manuelle Tests brächten die Gefahr mit sich, dass sie nicht verwendet würden.

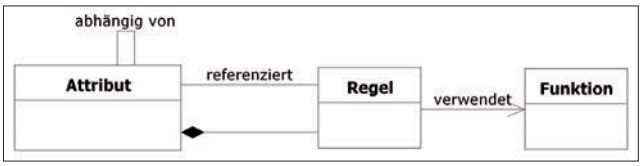

Abb. 3: Domänenmodellierung

tionsvolumen := *Stücke* * *Kurs* belegt das Attribut *Transaktionsvolumen* mit dem Produkt der Attributwerte von *Stücke* und *Kurs*.

- **Lebenszykluskonzept:** Dass die Geschäftsregeln Elemente aus dem Domänenmodell referenzieren, birgt die Gefahr, durch eine Veränderung des Modells ungültige Referenzen und damit ungültige Regeln zu erzeugen. Im Sinne der Fehlervermeidung werden die beiden relevanten Aktionen auf dem Domänenmodell wie folgt behandelt und durch die Pflegeoberflächen garantiert:
  - **Änderung:** Beim Verändern eines Attributs oder eines Elements werden die Regeln soweit möglich automatisch aktualisiert (bspw. Veränderung des Namens). Die Beziehungen zwischen Elementen kann der Fachanwender nur ändern, wenn durch die Änderung keine Regel ungültig wird.
  - **Löschen:** Das Löschen eines Attributs oder Elements aus dem Domänenmodell wird durch die Anwendung erst gestattet, sobald keine Regel das Element (mehr) verwendet.

Die Geschäftsregeln können entweder in textueller Form durch eine interne oder externe DSL [6], [7] oder in grafischer Form z. B. durch Entscheidungsbäume beschrieben werden. Für die Pflege steht ein Frontend zur Verfügung.

ÜBERGREIFENDE TECHNISCHE ANFORDERUNGEN

Neben den aus der Sicht des Fachanwenders existierenden Anforderungen kommen zusätzliche hinzu, die meist durch die IT-Abteilung aufgebracht werden, originär aber eher den Bereichen *Compliance* oder der *Revision* zugeordnet sind. Diese Anforderungen gelten gleichermaßen für Daten wie Geschäftsregeln:

- **Nachvollziehbarkeit:** Da die Fachbenutzer die Funktionalität ihrer Anwendung jederzeit verändern können, ist eine Protokollierung notwendig, um die Änderungen in ihrem fachlichen Kontext später einmal nachvollziehen zu können. Zu diesem Zweck werden die Daten oder Regeln historisiert [8]; die bisherigen Werte bei einer Änderung nicht überschrieben, sondern als nicht mehr gültig markiert (mit dem Zeitstempel der Änderung).
- **Reversible Änderungen:** Sollte sich ein semantischer Fehler in die Konfiguration eingeschlichen haben, werden die entsprechenden Änderungen zurückgenommen. Diese Anforderung ist eng mit der Anforderung Nachvollziehbarkeit verknüpft.
- **Rollenbasierter Zugriff:** Bei klassischen Fachapplikationen werden Änderungen durch die IT durchgeführt, die auch überprüft, ob der entsprechende Fachanwender die Änderungen veranlassen darf. Die Möglichkeit, das Verhalten der Fachapplikation durch Veränderungen an der Konfiguration vollständig zu verändern, erfordert eine saubere Verwaltung von Zugriffsrechten. In der Praxis bündeln einige Organisationen die Verantwortlichkeit für die Pflege des Domänenmodells und Geschäftsregeln in der Rolle des Data Owners. Oft teilen sich auch mehrere Data Owner diese Aufgabe; jeder ist für einen bestimmten Ausschnitt der Fachdomäne zuständig.
- **4-Augen-Prinzip:** Alle Änderungen am Domänenmodell oder den Geschäftsregeln müssen durch eine zweite Person freigegeben werden. Dieser Prozess dient der Fehlererkennung und erschwert den Missbrauch. Eine Unterstützung der Anwendung zur Vermeidung dieser Fehler ist nicht möglich, da es sich um semantische Fehler handelt.

FACHLICHES ANWENDUNGSFRAMEWORK

Eine auf diesen Anforderungen entworfene Fachapplikation hat andere Aufgaben, als klassische „starre“ Applikationen. Das eigentliche Domänenmodell und die Geschäftsregeln sind aus der Anwendung und der Verantwortung des Entwicklers auf den Fachanwender übergegangen. Es verbleiben zwei Verantwortlichkeiten:

Technisches Framework	Fachliches Anwendungsframework
Kein fertiges Programm	Keine fertige, Wert bringende Fachapplikation, obwohl fertiges Programm
Vorgabe der Anwendungsarchitektur	Vorgabe der Anwendungsarchitektur
Inversion of Control	Inversion of Control
Programmierer registriert konkrete Implementierungen	Fachanwender registriert konkrete Konfigurationsbausteine
Definiert den Kontrollfluss der Anwendung und die Schnittstellen für die konkreten Klassen	Definiert Lebenszyklus der Artefakte und die domänenspezifische Sprache für die Regelbeschreibung
Domänenspezifisch (meist technische Domänen)	Domänenspezifisch (immer fachlich) durch die domänenspezifische Sprache

Tabelle 1: Vergleich von technischen Frameworks und fachlichen Anwendungsframeworks

- **Domänenübergreifend:** Die Anwendung verwaltet den Lebenszyklus der einzelnen Bausteine, deren Abhängigkeiten und stellt die syntaktische Konsistenz sicher.
- **Domänenspezifisch:** Für die Formulierung von Geschäftsregeln bringt die Anwendung eine DSL mit.

Diese Aufgaben erinnern an technische Frameworks. Die beschriebene Architektur stellt ein fachliches Anwendungsframework dar. Wie das technische Pendant ist das Anwendungsframework für die Steuerung des Lebenszyklus oder Prozesses bzw. Kontrollflusses zuständig, erzeugt aber keinen fachlichen Wert. Nach dem Prinzip „Don't call us, we call you" werden die Bausteine aufgerufen, die der Fachanwender im Framework registriert. Tabelle 1 enthält einen Vergleich zwischen den beiden Arten von Frameworks [9]. Die Grundidee des fachlichen Anwendungsframeworks ist nicht neu. Es gibt zwei Ansätze, die in eine ähnliche Richtung gehen:

- **4GL-Sprachen:** Die so genannten Sprachen der vierten Generation sollten im Gegensatz zu ihren Vorgängern – den General Purpose Languages wie C++ oder Java – die Entwicklung von Anwendungen unterstützen, indem sie mit deutlich weniger Quellcode auskommen, deskriptiv sind und vom Fachanwender benutzt werden können (wie bspw. SQL).
- **General Purpose Configuration Machines (GPCM):** Eine häufig verwendete Plattform für die Umsetzung von Anwendungen mit den beschriebenen Flexibilitätsanforderungen ist Microsoft Access (oder in „einfacheren Fällen" Excel).

Den Hauptunterschied des fachlichen Anwendungsframeworks zu diesen beiden Ansätzen stellt die Konfiguration in Form einer domänenspezifischen Sprache (Domänenmodell und Geschäftsregeln) dar. Der Fokus auf eine Domäne und das Vermeiden von Abstraktion erlauben es, nah am „Sprachgebrauch" des Fachanwenders zu bleiben. Der Anwender kann die Anwendung selbst pflegen. Sobald eine Anwendungsplattform den Domänenfokus aufgibt, muss sie abstrakt sein, erfordert einen Übersetzungsprozess und kann schlechter durch den Fachanwender verändert werden.

Die GPCMs erfüllen die aufgelisteten technischen Anforderungen nicht automatisch. Sie müssen im Gegensatz zum Anwendungsframework explizit in jede Anwendung „eingebaut" werden und werden deshalb meist ausgelassen. Außerdem entstehen auf Basis von Access und Excel meist mittelfristig unwartbare Anwendungen, da ähnlich wie bei 4GL-Sprachen durch einen abstrakten Ansatz die konkrete Ausdrucksstärke in einer Domäne verloren geht. Das Anwendungsframework löst dabei noch ein Problem: Die fachliche Anforderungsbeschreibung und Dokumentation sowie die Anwendungskonfiguration sind ein Artefakt und damit inhärent konsistent. Die Vermeidung von Übersetzungsprozessen verhindert, dass Dokumentation und Konfiguration auseinanderlaufen.

FAZIT UND AUSBLICK

Eine sich rasant verändernde Umwelt und die Notwendigkeit schneller Produktinnovationen erfordern es, dass sich Fachapplikationen in kurzer Zeit anpassen lassen. Gleichzeitig sollen die vorgenommenen Änderungen nicht die Stabilität der Anwendung gefährden. Für die Realisierung solcher Applikationen wurde ein fachliches Anwendungsframework vorgestellt, das dem Fachanwender die Konfiguration des Domänenmodells und der Geschäftsregeln in einer domänenspezifischen Sprache erlaubt. Die Stabilität wird durch Mechanismen der Fehlererkennung und -vermeidung sichergestellt.

In der nächsten Ausgabe werden die Blaupause mit beispielhaften Technologien hinterlegt und die Prinzipien anhand eines fachlichen Beispiels konkretisiert.

Links & Literatur

[1] http://www.bvi.de/regulierung/
[2] http://de.wikipedia.org/wiki/Poka_Yoke
[3] http://www.eclipse.org/modeling/
[4] http://wiki.eclipse.org/Teneo
[5] http://wiki.eclipse.org/index.php/Ecore_Tools
[6] Diener, K.: „Ist das Groovy? Fachliche Domain Specific Languages mit Groovy entwickeln", in Java Magazin 8.2012
[7] Diener, K.: „DSL-Komposition – Integration einer Groovy-DSL mit einer Java-Anwendung", in Java Magazin 10.2012
[8] http://de.wikipedia.org/wiki/Temporale_Datenhaltung
[9] http://de.wikipedia.org/wiki/Framework

Konstantin Diener

ist Leading Consultant bei der Cofinpro AG. Er beschäftigt sich seit über zehn Jahren mit der Java-Plattform und sein Interesse gilt allem, was IT und Fachabteilungen flexibel und produktiv macht (Groovy, Drools, Scrum, Kanban, DevOps, fachlichen Akzeptanztestverfahren).

Daniel Kaminsky

ist als Consultant bei der Cofinpro AG beschäftigt. Er ist seit Jahren in der Java-Welt zu Hause und setzt seine Projekte mit Vorliebe nach der Scrum-Methodik um.