

# SAG'S EINFACH MIT DIAGRAMMEN: VISUELLES MODELLIEREN

Der Artikel soll die Vorteile und den Nutzen von visuellem Modellieren darstellen. Anhand der UML werden unterschiedliche Aktivitäten betrachtet, die durch Artefakte begleitet werden können. Darüber hinaus blicken wir in eine mögliche Zukunft der UML: die „Model Driven Architecture“. Kleine Beispiele runden den Beitrag ab.

Die objektorientierte Softwareentwicklung ist heute keine Besonderheit mehr. Mag die Vermittlung der Kenntnisse dieser Konzepte noch relativ überschaubar sein, so ist die Anwendung oft viel schwieriger. Eine direkte 1:1-Abbildung von Objekten der realen Welt auf Softwareobjekte wäre zwar oft wünschenswert, ist aber in der Regel weder sinnvoll noch durchführbar.

Eine wesentliche Aufgabe des objektorientierten Designers besteht darin, die Verantwortung an Objekte zu verteilen und zu rechtfertigen (vgl. [Lar01]). Dazu bedient er sich verschiedener Mittel. Während der Design-Aktivitäten werden Lösungen für ein spezielles Problem gesucht und das beste Vorgehen wird ausgewählt. Diese Entscheidungen werden in Diagrammen festgehalten, die eine universelle Notation haben. Die *Unified Modeling Language (UML)* bietet seit dem

Jahre 1997 eine standardisierte Basis für diese Illustrationen, die objektorientierte Analytiker, Designer und Entwickler beherrschen müssen, um ihre Ideen austauschen zu können.

Ist für einige Projektmitarbeiter das Visualisieren von Ideen selbstverständlich, leiden viele Projekte noch an mangelnder visueller Begleitung von Analyse und Design. Der Bericht zeigt, welche Möglichkeiten visuelles Modellieren und die UML für objektorientierte Softwareprojekte bieten und welche Gefahren dadurch minimiert werden. Darüber hinaus betrachten wir eine mögliche Zukunft der UML: die *Model Driven Architecture (MDA)*.

Schwerpunkt dieses Artikels ist nicht die Notation der UML, sondern die Möglichkeiten, die sich für IT-Manager, objektorientierte Designer und Entwickler bieten, wenn Software-Engineering visuell

## der autor



**Jochen Krebs**  
(E-Mail: mail@jochenkrebs.com) ist seit drei Jahren als Senior Instructor für die Firma Valtech, Inc. in New York tätig. Er unterrichtet Firmen unter anderem in Projektmanagement, Anforderungsanalyse und objektorientierten Technologien.

begleitet wird. Dabei konzentriert sich der Artikel auf vier wesentliche Aktivitäten, die von der Notation unterstützt werden:

- Darstellen,
- Dokumentieren,
- Generieren und schließlich
- das Erweitern der Notation.

### Darstellen

Während eines von mir durchgeführten Workshops zum Thema Geschäftsprozessoptimierung versuchten Teilnehmer immer wieder ihre Ideen im Bild festzuhalten. Sie griffen zu Papier und Bleistift und drückten ihre Ideen in dem Geschäftsgebiet – in diesem Falle in Fabriken, Lastwagen und Straßen – aus. Jeder Teilnehmer hatte die Möglichkeit seine Vorstellungen zu äußern und zu erläutern. Viele der daraus resultierenden Fragen betrafen jedoch weniger das eigentliche Fachthema, sondern zeigten Erklärungsbedarf zum gewählten Bild selbst.

Auf der einen Seite hatten wir natürlich mehr Spaß mit den individuellen Bildern, aber jeder der Teilnehmer erkannte die Nachteile dieses Vorgehens: ein erheblich erhöhter Bedarf an Konversation, um unsere Ideen zu vermitteln. Glücklicherweise waren alle im gleichen Raum und konnten demzufolge interaktiv die Ideen diskutieren und erklären – eine Situation, die wir in Softwareprojekten nicht immer antreffen (z. B. beim „Offshore Development“). Das ist genau der Punkt, an dem die UML eingreift und versucht, redundante Diskussionen über die Sprache selbst zu minimieren. Einmal erklärt, hilft uns die vereinfachte und universell verbreitete visuelle Notation, Ideen oder Lösungen kostengünstig zu verbreiten. **Abbildung 1** zeigt ein Beispiel, in dem ein

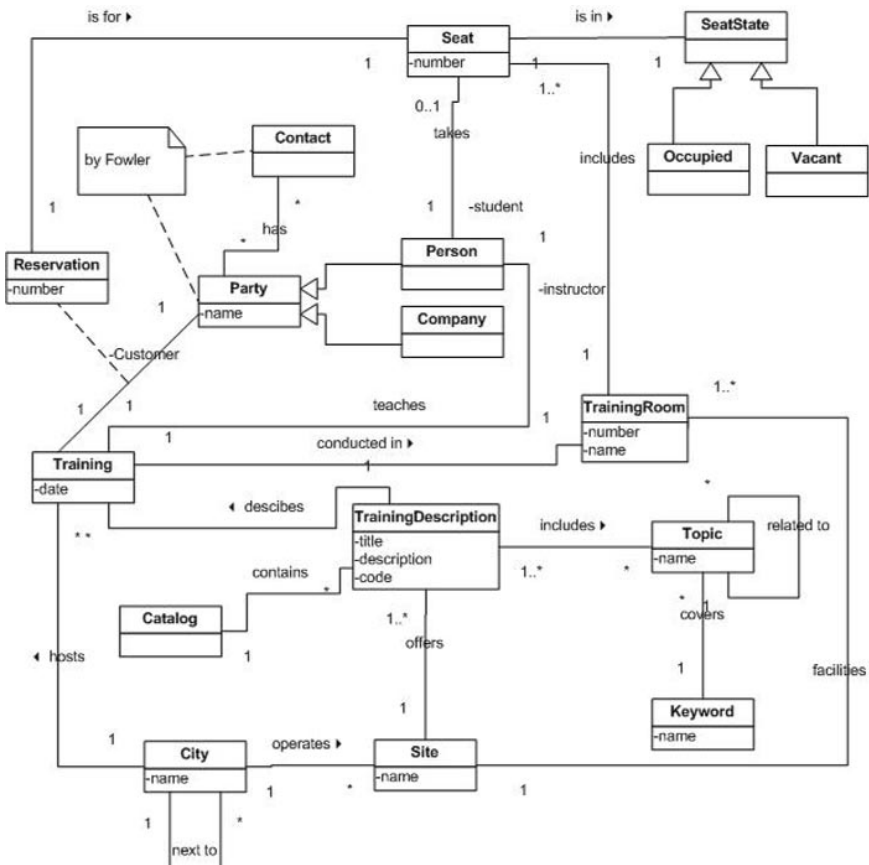


Abb. 1: UML Domänen-Modell



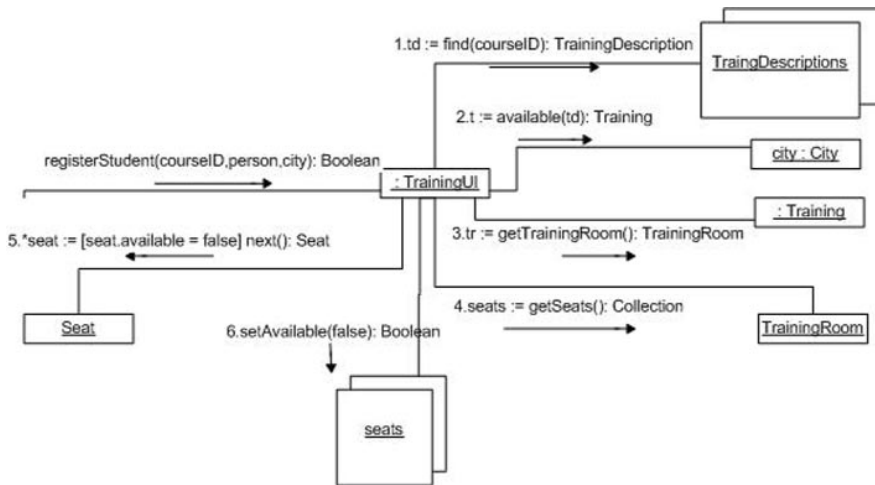


Abb. 2: UML-Kollaborationsdiagramm mit hoher Kopplung (high coupling)

Business-Analyst versucht sein Problem zu beschreiben (vgl. [Fow99]).

Das Bild verdichtet nun wichtige Informationen, die uns später bei der Entscheidungsfindung unterstützen. Wichtig ist, dass wir wahrscheinlich viel Prosa schreiben müssten, um die gleichen Informationen im Team zu verbreiten. Manchmal zeigt ein Bild mehr als 1.000 Worte. Beispielsweise kennt eine City im obigen Domänen-Modell die Nachbarorte, um bei Überbuchungen eines Trainings eine Alternative in der Nähe anzubieten. Wir können auch erkennen, dass nur eine Beschreibung eines Trainings (TrainingsDescription) benötigt wird, um alle Trainings des gleichen Typs zu spezifizieren.

Schauen wir uns etwas näher ein Diagramm aus einer Design-Aktivität an. Wir liefern eine courseID, ein Person- und ein City-Objekt, um einen Studenten in einem Kurs zu registrieren.

Abbildung 2 zeigt eine Lösung, in der das TrainingUI-Objekt viele Verbindungen hat. Diese sind auch als Sichtbarkeiten bekannt, die den Kopplungsgrad zwischen Objekten ausdrücken. Je mehr Kopplungen existieren, desto abhängiger ist ein Objekt von anderen Objekten. Der unerwünschte Nebeneffekt ist, dass die Flexibilität, Wiederverwendbarkeit und Wartbarkeit solcher Systeme geringer ist. Systeme mit hoher Kopplung erfordern mehr Aufwand in der Pflege, da bei einer einzigen Änderung eine erhöhte Abhängigkeit zu anderen Objekten besteht. Hohe Kopplungen können nun von Projektmitgliedern leichter erkannt werden, wenn diese visuell dargestellt sind (Whiteboards, CASE-Werkzeuge etc.), und entsprechende Alternativen können ausgearbeitet werden.

Das Diagramm in Abbildung 3 zeigt beispielsweise eine Lösung für das gleiche Problem. Es verbessert das Design von Abbildung 2 an einigen Stellen. Die Be-

nutzeroberfläche ist durch eine Facade zu Geschäftsobjekten entkoppelt worden. CourseLocator und Registrar sind Designentscheidungen, die zwei wesentliche Tätigkeiten voneinander entkoppeln: das Finden und das Buchen eines Kurses. Mit dieser einfachen Designentscheidung haben wir die Verantwortlichkeiten der Objekte verteilt und unabhängiger gemacht. Das wird durch das Diagramm optisch deutlich.

Ideen in Bildern und Notationen auszudrücken ist nicht neu – das wurde auch schon vor objektorientierten Methoden praktiziert. Neu ist jedoch, dass die Object Management Group (OMG) UML-Elemente bündelt und standardisiert. Das ermöglicht es Unternehmen und Mitarbeitern, ihre Ideen einheitlich austauschen.

Die UML ist hier wie jede visuelle Abbildung, zum Beispiel Notenblätter für Musiker. Niemand würde auf die Idee kommen, eine neue Notation für Musiker zu erfinden, solange die existierende allen Bedürfnissen gerecht wird. Das Wissen selbst, wie man eine Note malt oder darstellt, ist allerdings nicht ausreichend, um ein guter Musiker oder Komponist zu sein. Das ist bei

der UML nicht anders und wurde in Abbildung 2 und 3 demonstriert. Entscheidend ist, dass ein Komponist die Notenblätter an unterschiedliche Musiker verteilen kann, ohne erklären zu müssen, wie man ein A auf einer Gitarre oder auf dem Klavier erzeugt. Allein durch die Noten sind Orchester nun in der Lage, Stücke von Komponisten zu spielen, die nicht anwesend sind.

Modelle können nun in unterschiedlichen Details erstellt werden. Das kann bis in alle Details perfektioniert werden oder sehr grob auf Papier oder dem Whiteboard festgehalten werden. Zum Beispiel kann ein Team von Designern eine Lösung kollaborativ erarbeiten und lässig auf Whiteboards festhalten. Damit die wertvolle Arbeit nicht verloren geht, erweist sich beispielsweise eine digitale Kamera als nützlich. Diese Bilder können nun archiviert, ausgetauscht oder im Rahmen der technischen Dokumentation aufbereitet werden. Daher müssen nicht alle Projektmitglieder Experten in UML oder in den CASE-Werkzeugen sein.

Wenn Designer Entwurfsmuster verwenden, müssen nicht mehr alle Details visuell beschrieben werden – die UML erlaubt hier ein vereinfachtes Vorgehen. Entwurfsmuster sind wiederkehrende Problem-Lösungs-Beschreibungen (vgl. [Gam95]), die in objektorientierter Analyse und Design häufig eingesetzt werden. Die Details dieser Muster müssen aber nicht immer wieder in den eigenen Diagrammen nachgezogen werden. Ein einfacher Verweis zu dem besagten Pattern kann schon ausreichend sein. Der Modellierer kann sich demzufolge auf sein Problem konzentrieren. Ein Beispiel aus Abbildung 3 zeigt den Verweis zu einem State-Pattern, das nun nicht mehr in allen Details erläutert werden muss, da der Autor des Musters der Lösung einen Namen gab (hier „State“). In dem Beispiel wurde ein simpler UML-

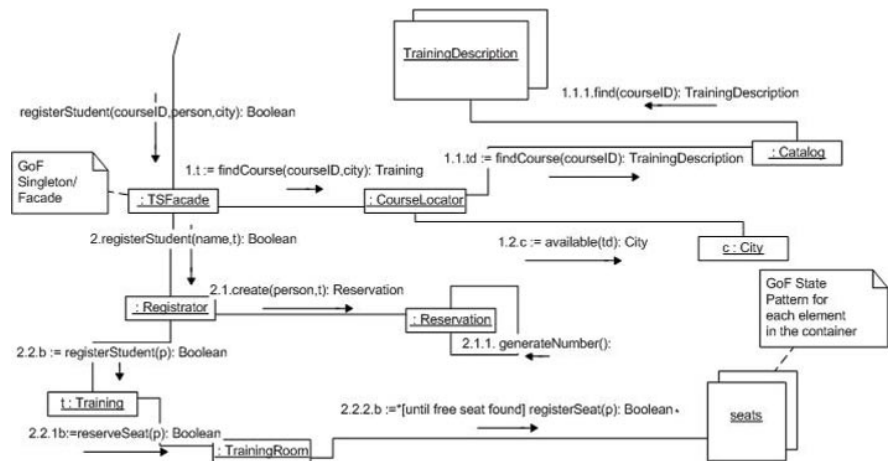


Abb. 3: UML-Kollaborationsdiagramm mit geringerer Kopplung (low coupling) ▶

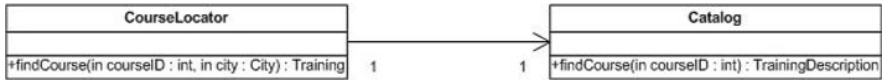


Abb. 4: UML Design-Klassendiagramm

Kommentar verwendet, der den Leser zu der Quelle verweist ([Gam95]) – dasselbe könnte jedoch auch mit Hilfe von Stereotypen oder der *Object Constraint Language (OCL)* erreicht werden (vgl. [War98]). Die Einbindung dieser Entwurfsmuster macht nicht nur das eigene Modell eleganter, es ermöglicht Projektleitern, Architekten oder Qualitätsbeauftragten, in Modellen die Verbreitung und Anwendung von Entwurfsmustern zu überblicken und letztendlich zu fördern.

Dokumentieren

Visuelle Modelle sind ebenso nützlich während Wartungsarbeiten an bestehenden objektorientierten Softwaresystemen. Zahlreiche Systeme haben wenig oder wenig hilfreiche beschreibende Dokumente. Häufig entspricht die Dokumentation nicht mehr dem aktuellen Stand der Systemfunktionalität. Ein Anzeichen dafür ist, dass Entwickler im Code navigieren, um entsprechende Funktionalität zu erklären. Spätestens wenn der Entwickler eine einzige Änderung vollzieht, ohne diese in Anforderungs-, Analyse- und Designdokumenten nachzuziehen, ist die dazugehörige Systemdokumentation veraltet und wird früher oder später wertlos.

Auf der anderen Seite kann visuelles Modellieren helfen, projektbegleitend an der Systemdokumentation zu arbeiten, und wesentlich dazu beitragen, dass die Dokumentation eines Systems aufschlussreiche Informationen für die Wartung des Systems enthält (siehe Abb. 2 und 3).

Spätestens wenn wir eine verteilte Architektur vorfinden und versuchen, die Funktionalität innerhalb des Codes aufzudecken, wird das Navigieren im Code, um Geschäftslogik aufzufinden, kompliziert. Das liegt daran, dass Objekte Systemgrenzen verlassen können. Ein Bild kann auch hier wertvolle Dienste leisten.

In iterativen Vorgehensmodellen kann Projektleitern, Architekten (oder selbst den Kunden) durch eine stetig wachsende Dokumentation der Projektfortschritt deutlich gemacht werden, ohne selbst Programmiersprachen verstehen zu müssen. Die UML ermöglicht die Einbindung aller Beteiligten, wenn diese den Sprachvorrat der Notation verstehen. Durch ein seriöses Vorgehensmodell wird mit begleitender

Dokumentation Vertrauen erzeugt, das die spätere Akzeptanz des Systems erhöht.

Generieren

Eines haben alle Softwaresysteme am Ende gemeinsam: Sie müssen ausführbar sein, denn wir werden für Software bezahlt, nicht für Papier. Da in der UML bereits Objekte, Klassen, Nachrichten und Methoden abgebildet sind, ist eine Übertragung in eine objektorientierte Programmiersprache einfach. Viel wichtiger ist, dass Entwickler weniger Zeit mit dem eigentlich Schreiben verbringen und die Zeit in frühere Aktivitäten verlagern können. **Abbildung 4 und Listing 1** zeigen ein Beispiel, wie eine Java-Klasse in Code überführt wird.

Der CourseLocator konnte nun von einem sogenannten Design-Klassendiagramm direkt in Java übertragen werden. Da sich die UML programmiersprachenunabhängig verhält, hätten wir mit Hilfe eines automatischen Generators auch anderen objektorientierten Code erzeugen können (z. B. Smalltalk, C++ oder C#).

Nachdem wir die Klasse definiert und die Nachrichten (in diesem Falle nur eine) skelettartig definiert haben, müssen wir wieder **Abbildung 3** zur Hilfe nehmen, um den dynamischen Inhalt der Nachricht zu füllen. Die Frage ist, was passiert, wenn das Objekt CourseLocator die Nachricht findCourse(courseID,city) empfängt.

Das Kollaborationsdiagramm bietet die Lösung. In Schritt 1.1 (**Abb. 2**) sendet der CourseLocator die Nachricht zum Objekt Catalog, von dem der CourseLocator eine Instanz enthält. Wir erwarten eine Beschreibung eines Trainings (Training Description) zurück. Danach möchten wir herausfinden, ob die gelieferte City das Training anbietet und geben das Training zurück (**Listing 2**).

Natürlich drängt sich nun der Gedanke auf, warum man das nicht automatisch

```

public class CourseLocator
{
    private Catalog catalog;
    public Training findCourse(int
        courseID , City city)
    {
        // wird gleich mit Leben gefüllt.
    }
    public Catalog getCatalog()
    {
        return catalog;
    }
}
    
```

Listing 1: Java-Klassen-Skelett ohne Methoden

durchführen kann. Je besser die Qualität der Diagramme ist, desto besser wird auch die Generierung von Code sein. Bei der steten Nutzung von automatischer Generierung zahlt sich die Notation nun doppelt aus.

Viele Unternehmen forcieren bereits die automatische Generierung aus Modellen direkt in ausführbaren Code. Ein beeindruckendes Beispiel stellt die Firma Lockheed Martin Aeronautics, Kennedy Carter dar. In einem Projekt zur Cockpit-Steuerung des F16-Jets wurde der komplette Code aus UML-Diagrammen generiert und kein Code manuell erzeugt. Mit einer *xUML* (einer *UML action language*) wurden die Modelle in einem Zwischenschritt umgewandelt, bevor sie letztendlich in eine objektorientierte Sprache übersetzt wurden. Die OMG hat es sich mit der MDA zum Ziel gesetzt, Codegenerierung aus Modellen mehr und mehr zu fördern. Deshalb bildet die UML eines der Herzstücke der *Model Driven Architecture (MDA)*.

Viele Firmen experimentieren mehr und mehr mit der automatischen Generierung von Code – die Gründe hierfür sind vielschichtig:

- kürzere Entwicklungszeiten,
- höhere Designqualität,
- begleitende Dokumentationen, aber auch
- eine bessere Unabhängigkeit von wiederkehrenden Veränderungen in technologischen Trends.

Viele Firmen sind bereits auf den MDA-Zug aufgesprungen und arbeiten an kommerziellen Lösungen, um den Ansatz der MDA vorwärts zu bringen. Da sich die Notation technologieneutral verhält, können Systeme mit Modellen sehr viel schneller auf veränderte Situation reagieren und leichter portiert werden. Ebenso werden Designer und Entwickler dabei unterstützt, fachliche und technische Herausforderungen besser zu trennen.

Erweitern

Grafische Darstellungen von Problem und Lösungselementen sind in der UML bereits gelöst. Nicht immer reicht die Sprache aber aus, um alle erdenkliche Anforderungen abzudecken. Die OMG machte ein Spagat zwischen Lernkurve, Lesbarkeit und Nutzen von UML. Eine Tür wurde offen gehalten, um individuellen Bedürfnissen von Projekten gerecht zu werden.

Die UML kann in allen Disziplinen eingesetzt werden. Designaktivitäten, Analysemodelle und Use-Case-Diagramme werden standardmäßig beschrieben und haben aus-



```
public class CourseLocator
{
    private Catalog catalog;

    public Training findCourse(int courseID , City city)
    {
        TrainingDescription td =
            this.getCatalog().findCourse( courseID);
        return city.available(td);
    }
    public Catalog getCatalog()
    {
        return catalog;
    }
    // und so weiter ...
}
```

**Listing 2:** Java-Klasse mit Methodenbeschreibung

reichend Elemente im Vorrat. Das bedeutet aber nicht, dass man die UML nicht in anderen Bereiche verwenden kann, zum Beispiel für Geschäftsprozessmodelle (vgl. [Eri00]).

Der UML-Sprachvorrat ist nicht statisch, er kann auf projekt-spezifische Bedürfnisse angepasst werden. Sogenannte Stereotypen oder die OCL sind hier besonders hervorzuheben. Lassen sich neue Elemente mit Stereotypen definieren oder existierende überlagern, bietet die OCL das Modellieren von Geschäftsregeln an (vgl. [War98]).

Die UML als Notation ist in einen der erfolgreichsten Entwicklungsprozesse, dem *Unified Process (UP)*, integriert. Das bedeutet aber nicht, dass man die Notation nicht ebenso für Skizzen in einem XP-Projekt oder anderen agilen Prozessen verwenden kann (vgl. [Amb01]).

### Fazit

Wenn wir die Möglichkeit hätten unser eigenes Haus zu gestalten und zu bauen, wie würden wir an dieses Projekt herangehen? Selbstverständlich würden wir uns hinsetzen und mit Zeichnungen die möglichen Problem- und Lösungsszenarien durchspielen.

Nachdem wir eine erfolversprechende Lösung vor uns haben, würden wir den Plan in die Realität umsetzen. In Softwareprojekten gehen wir oft etwas unkoordinierter vor, weil es einfacher ist, eine Zeile aus dem Quellcode zu entfernen als eine Mauer. Aber die Seiteneffekte können die gleichen sein. Mit Hilfe der einheitlichen Notation (hier die der Architekten) können wir sogar die Ausführung an Baufirmen vergeben, die den Plan interpretieren können.

Die Notation ist einer der Schlüssel zum Erfolg, wenn man sich wie im Softwaregeschäft auf die Probleme und Lösungen konzentrieren will und die Implementierungsdetails verbirgt. Nun wissen wir, dass Software anders „gebaut“ wird als Gebäude; dennoch hat selbst der UP als eines der sechs wichtigsten Konzepte „model visually!“ hervorgehoben. Jeder dieser Erfolgsfaktoren hat starken Einfluss auf die Qualität und demzufolge den Erfolg eines Projekts (vgl. [Kru00]).

Aber egal, welchen Prozess wir wählen – am Ende müssen wir Software liefern, die das tut, was der Kunde wollte. Eine aktuelle Dokumentation rundet das Paket ab und hilft eventuell dem einen oder anderen Unternehmen beim Sprung zum nächsten *CMM-Level (Capability Maturity Model)*. Wie genau es die Teams mit visuellen Modellieren nehmen, hängt von dem angestrebten Ziel ab. Will man aus den Modellen direkt Software erzeugen, muss man sehr genau alle Einzelheiten definieren. Um einige Problemstellen in Analyse oder Design zu diskutieren, mag bereits eine Skizze ausreichend sein.

Die UML wurde hier nur beispielhaft gewählt, da sie die am meisten verwendete Notation in der objektorientierten Softwareentwicklung ist. Die UML wurde seit 1997 flächenweit bekannt, ist akzep-

tiert und bietet für alle erdenklichen Anwendungssysteme die notwendigen Elemente (Echtzeit, eingebettete Systeme usw.). Die UML ist in drei grobe Kategorien eingeteilt: organisatorische, statische und dynamische. Der Vorrat ist relativ klein und rasch erlernbar, aber je nach Interessensgebiet bietet die Notation viel Gestaltungsspielraum und Offenheit.

Bewusst habe ich Namen und Versionen von kommerziellen Werkzeugen und Firmennamen aus diesem Bericht herausgelassen. Ebenso möchte ich keinen Produktvergleich vornehmen oder eine Empfehlung aussprechen. Selbstverständlich sind Werkzeuge jedoch unersetzlich, wenn wir seriöse Software von Morgen entwickeln wollen.

Egal wofür wir uns am Ende entscheiden, optische Untermauerung hat wesentliche Vorteile für alle Projektmitarbeiter. Das visuelle Modellieren von Ideen, Problemen und deren Lösung begleitet daher stets die Aktivitäten im Software-Engineering. ■

### Literatur & Links

- [Amb01] S.Ambler, *Object Primer*, Cambridge University Press, 2001
- [Eri00] H.-E. Eriksson, M. Penker, *Business Modeling with UML*, OMG Press 2000
- [Fow99] M. Fowler, K. Scott, *UML Distilled (2<sup>nd</sup> Ed.)*, Addison-Wesley 1998
- [Gam95] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns*, Addison-Wesley 1995
- [Kru00] P. Kruchten, *The Rational Unified Process (2<sup>nd</sup> Ed.)* Addison-Wesley, 2000
- [Lar02] C. Larman, *Applying UML and Patterns, (2<sup>nd</sup> Ed.)* Prentice Hall, 2002
- [War98] J. Warmer, A. Kleppe, *The Object Constraint Language*, Addison-Wesley 1998