Xpert.press

Die Reihe **Xpert.press** vermittelt Professionals in den Bereichen Softwareentwicklung, Internettechnologie und IT-Management aktuell und kompetent relevantes Fachwissen über Technologien und Produkte zur Entwicklung und Anwendung moderner Informationstechnologien.

Volker Gruhn · Daniel Pieper
Carsten Röttgers

MDA®

Effektives Software-Engineering mit UML 2® und Eclipse ™

Mit 293 Abbildungen

Springer

Volker Gruhn
Lehrstuhl für
Angewandte Telematik/e-Business
Universität Leipzig
Klostergasse 3
04109 Leipzig
gruhn@ebus.informatik.uni-leipzig.de

Daniel Pieper
Carsten Röttgers
adesso AG
Stockholmer Allee 24
44269 Dortmund
dp@mda-buch.info
cr@mda-buch.info

Bibliografische Information der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.de abrufbar.

ISSN 1439-5428
ISBN-10 3-540-28744-2 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-28744-5 Springer Berlin Heidelberg New York

Springer ist ein Unternehmen von Springer Science+Business Media
springer.de

Satz: Druckfertige Daten der Autoren
Herstellung: LE-TEX, Jelonek, Schmidt & Vöckler GbR, Leipzig
Umschlaggestaltung: KünkelLopka Werbeagentur, Heidelberg
Gedruckt auf säurefreiem Papier 33/3142 YL – 5 4 3 2 1 0

Vorwort

„Simple things should be simple and complex things should be possible."

Alan Kay

Informationstechnik ist über den Lauf der Jahre zum Rückgrat der Geschäftswelt geworden: kaum eine kommerzielle Transaktion läuft heute nicht IT-gestützt ab, nahezu jeder Euro, der seinen Besitzer wechselt, wird elektronisch erfasst. Die Geschäftsprozesse vieler Unternehmungen entwachsen den vier Wänden ihrer Bürogebäude oder Produktionsstätten und spielen sich in Zeiten von Globalisierung und sich öffnenden Märkten zunehmend auch auf internationalem Parkett ab. Solch internationales Treiben ist dabei nicht länger auf große Konzerne beschränkt, sondern wird immer mehr auch für kleine und mittelständische Unternehmen zum Tagesgeschäft. Die daraus resultierenden Anforderungen erfordern eine entsprechende IT mit maßgeschneiderten Lösungen, die die individuellen Bedürfnisse der Unternehmen berücksichtigen.

Das Problem

Wir können nicht wie wir wollen. Man mag es ja kaum noch hören, aber die Software-Industrie steckt immer noch in der bereits vor Jahrzehnten ausgerufenen Software-Krise. Gewiss, in puncto Effizienz, Effektivität und Produktivität sind Fortschritte erzielt worden, aber wenn wir ehrlich sind, sind es eher Schrittchen, von denen wir da sprechen und weniger die erhofften Sprünge. Die Bestandsaufnahme der Software-Projektlandschaft fällt dementsprechend fatal aus: Falls ein Projekt tatsächlich überlebt, dann selten *in Time* und noch viel seltener *in Budget*. Bei der Qualität wird es dann schließlich ganz dünn, sie fällt meist gleich als Erstes über Bord, sobald Time und Budget in Gefahr scheinen.

Dabei wird Software immer komplexer, die Ansprüche an ihre Leistungsfähigkeit, Zuverlässigkeit, Sicherheit und Skalierbarkeit bei stetig steigendem Kosten- und Zeitdruck gleichzeitig immer größer. Ein *Circulus Diaboli*, bei dem man am liebsten den Kopf in den Sand stecken möchte.

Dumm nur, wenn der Kunde dieses Spiel nicht mehr mitspielen mag und selbst aktiv nach Auswegen sucht. In letzter Zeit werden dabei immer häufiger Outsourcing und Near-/Offshoring-Maßnahmen diskutiert. Auch wenn Yourdon in [You05] attestiert, dass sich die Entwicklung von betriebswirtschaftlicher Individualsoftware weniger zu solchen Maßnahmen eignet als etwa die von wissenschaftlich oder technisch geprägter Software, fällt sein Resümee am Ende ernüchternd aus: Immer öfter gewinnen spezialisierte Offshorer auch in diesem Bereich gegen Dienstleister aus der *alten Welt* und zwar nicht nur bei den Kosten, sondern auch bei Termintreue und Qualität. Geben wir es ruhig zu: die westliche IT-Industrie ist auch hier überrumpelt worden. Über die Hälfte aller CMM5-zertifizierten Unternehmen kommt mittlerweile aus Indien.

Für die Software entwickelnden Unternehmen heißt es also endlich aktiv zu werden und nach Lösungen für eine effizientere Entwicklung und Pflege von Software zu suchen. Aussprüche wie „Das Ende der Software-Entwicklung wie wir sie kennen“ scheinen so gerade rechtzeitig den nächsten Anlauf auf die Brook´sche Silverbullet einzuläuten. Sollten wir nicht eigentlich gelernt haben, dass es selbige nicht geben kann? Nun, glauben wir zumindest Frankel [Fra04] und Greenfield et al. [GSCK04], so stehen wir allen Unkenrufen zum trotz kurz vor dem Eintritt in eine Phase zunehmender Professionalisierung der Softwareentwicklung.

Frankel wählt dazu den Begriff der *Software-Industrialisierung*, Greenfield spricht in diesem Zusammenhang gar von *Software-Fabriken*. Uneins ist man sich zurzeit noch, wie die konkrete Manifestierung dieser Ideen auszusehen hat. Einigkeit scheint hingegen darüber zu herrschen, dass aus technischer Sicht die Softwareentwicklung zukünftig architekturzentriert, aspektorientiert, domänenspezifisch und modellgetrieben erfolgen muss.

Der Vorschlag der Object Management Group

Eine ganzheitliche Vorgehensweise, die versucht durch die synergetische Verwendung bekannter Prinzipien und Methoden den neuen Anforderungen zu begegnen, ist die Model-Driven Architecture (MDA) der OMG, die bereits seit dem Jahr 2000 einen der großen Hoffnungsträger für den Weg aus der Krise darstellt.

Sie beschreibt den Rahmen für ein modellgetriebenes Vorgehen, bei dem Modelle in den Mittelpunkt des Entwicklungsprozesses

gerückt werden. Kern dieser Architektur ist ein Konzept, das zwischen plattform-unabhängigen und plattform-spezifischen Modellen unterscheidet und es so ermöglicht, die Spezifikation der Funktionalität eines Systems von der Spezifikation, wie diese Funktionalität auf eine spezifische Plattform implementiert wird, zu trennen. Der Übergang von der fachlich getriebenen Spezifikation zur ausführbaren Anwendung erfolgt dabei möglichst automatisiert durch geeignete Transformationswerkzeuge.

Versprochen wird sowohl Kostensenkung durch die generative Erzeugung von weiten Teilen der Anwendung und beschreibender Modelle sowie die vereinfachte Wiederverwendung horizontaler Querschnittskomponenten in der Entwicklung von Folgeanwendungen. Durch die Erstellung technologieunabhängiger Modelle soll zudem der Forderung der Unternehmen nach Konservierung der Fachlichkeit entsprochen werden, indem die fachlichen Modelle ihre Gültigkeit, auch bei sich ändernden unterliegenden Technologielandschaften, behalten und so die aktuelle Dominierung der Fachlichkeit durch die kurzen technologischen Änderungszyklen durchbrochen wird.

Die notwendigen Mittel zur Umsetzung

Um die eben dargestellten Prinzipien anzuwenden, bedarf es einer maßgeschneiderten unterstützenden Infrastruktur, die unter anderem die benötigten domänenspezifischen Sprachen, die geeigneten Prozesse sowie die technischen Werkzeuge zur Unterstützung des Automatisierungsansatzes liefert bzw. ermöglicht. Bislang waren diese Werkzeuge proprietär und herstellergebunden bzw. mit hohen Eigenentwicklungskosten verbunden, die gerade für kleine und mittelständische Unternehmen (KMUs) mit zu hohem Risiko und Investitionsaufwand zu Buche schlagen, um ernsthaft in Erwägung gezogen zu werden. Mittlerweile existiert jedoch eine Vielzahl offener Standards und Open-Source Frameworks, die einen Versuch der Umsetzung in Eigenregie in neuem Glanz erscheinen lassen.

Mit den Modellierungsstandards der OMG, namentlich des UML und MOF 2.x Stacks, liegt nun auch der benötigte konzeptuelle Überbau vor; domänenspezifische Erweiterungen sind in der Konzeption bzw. kurz vor der Fertigstellung. Open-Source Projekte wie Eclipse, die von vielen großen Firmen unterstützt werden, bilden ganze Ökosysteme frei verfügbarer technischer Bausteine, diese Konzepte auch auf praktischer Ebene umsetzen zu können.

Das vorliegende Buch …

… nimmt zum einen die theoretischen Konzepte unter die Lupe und versetzt den Leser so in die Lage die Eignung der MDA für die

individuelle Situation seines Unternehmens besser einordnen und bewerten zu können, liefert zum anderen jedoch auch konkrete Betrachtungen zur praktischen Umsetzung mittels frei verfügbarer Techniken und Technologien. Besonderer Fokus wird dabei auf die Verwendung der Unified Modeling Language (UML) als Modellierungssprache der Wahl sowie auf das Eclipse-Projekt als technische Basis zur Umsetzung gelegt. Im Verlaufe der Untersuchung der Randaspekte der MDA werden weiterhin viele weitere Konzepte und Technologien des modernen Software-Engineerings aufgegriffen und vertieft, sodass die Lektüre auch dann lohnt, wenn die Einführung MDA im eigenen Hause nicht kurz vor der Tür steht.

Aufgrund der Vielfalt der Themen können viele hochinteressante Aspekte der MDA jedoch trotzdem nur am Rande behandelt werden. Wir hoffen jedoch, dass bei der getroffenen Auswahl auch die für Ihre individuelle Situation passenden Themenkomplexe enthalten sind und wünschen Ihnen in diesem Sinne *Viel Spaß beim Lesen*.

Literatur

[Fra04] David S. Frankel: *Software Industrialization and the New IT*. In: David S. Frankel und John Parodi (Hrsg.): The Mda Journal – Model Driven Architecture Straight From the Masters. Meghan-Kiffer, 2004.

[GSCK04] Jack Greenfield, Keith Short, Steve Cook und Stuart Kent: *Software Factories – Assembling Applications with Patterns, Models, Frameworks, and Tools*. John Wiley & Sons, 2004.

[You04] Edward Yourdon: *Outsource – Competing in the Global Productivity Race*. Prentice Hall, 2004.

Inhaltsverzeichnis

1 Einleitung

„If all you have is a hammer, everything looks like a nail."

Bernard Baruch

1.1 An wen wendet sich dieses Buch

Das vorliegende Buch ist so aufgebaut, dass es Aspekte für diejenigen Leser enthält, die sich „nur" über die Wurzeln und theoretischen Grundlagen der Model-Driven Architecture informieren wollen, aber auch den Bogen zu deren praktischer Anwendung schlägt, sodass auch Praktiker, die Tipps zur konkreten Umsetzung erwarten, sich in diesem Buch wiederfinden werden.

Speziell werden folgende Themenkomplexe für die folgenden Interessengruppen behandelt:

1.1.1 Entscheider/Manager/Projektleiter

Für die Unentschiedenen, die nicht sicher sind, ob die Vorgehensweisen der MDA zur Bewältigung ihrer individuellen Anforderungen prinzipiell geeignet sind, bietet das vorliegende Buch erste Einblicke und die notwendige Grundlage zur fundierten Entscheidungsfindung. Für alle, die diese Entscheidung bereits getroffen haben oder kurz davor stehen, wird konkret auf die Auswirkungen eingegangen, die eine Einführung auf die bestehende Unternehmensorganisation hat bzw. haben sollte und bietet Tipps zur Adaption bestehender Entwicklungsprozesse.

Auch als „Know-how Update", um über die Themen auf dem Laufenden zu bleiben, die aktuell die Softwareentwicklung beschäftigen, ist das vorliegende Buch bestens geeignet. Die Basiskonzepte

und Ideen der MDA werden ausführlich erläutert und in den Kontext des aktuellen State-of-the-Art der Softwareentwicklung eingeordnet.

1.1.2 Berater

Für die Berater unter Ihnen liefert das Buch das notwendige Hintergrundwissen zu den, oft nur als Buzzwords missbrauchten, Konzepten eines der aktuell interessantesten Bereiche des Software-Engineerings, das zur fundierten Einschätzung und Einordnung dieser Ideen in der Praxis benötigt wird, um eine professionelle Beratungsleistung in diesem Themenkomplex bieten zu können. Anhand von Beispielen wird gezeigt, wie den Anforderungen des IT-Alltags mit den Werkzeugen der MDA begegnet werden kann. Die vorgestellten Bausteine können auch in Ansätzen, die nicht den Stempel MDA tragen, verwendet werden und bilden so eine sinnvolle Ergänzung jedes Consulting-Toolkits.

Last-but-not-least werden Antworten auf häufig – und berechtigt – gestellte Fragen zu Machbarkeit und ROI des Ansatzes gegeben, die als Argumentationshilfe in Kundengesprächen wertvolle Dienste leisten können.

1.1.3 Architekten und Entwickler

Anhand eines Fallbeispiels werden die im ersten Teil des Buches theoretisch erarbeiteten Konzepte und Forderungen nach weitestgehender Werkzeugunterstützung des MDA-Prozesses durch die Erarbeitung eines Beispielframeworks umgesetzt, das ganz ohne proprietär kommerzielle Bausteine auskommt, und ausschließlich mittels Open-Source Technologien realisiert werden kann. Der Fokus wird dabei vor allem auf das Eclipse-Projekt [Eclipse] sowie dessen untergeordnete Teilprojekte gelegt, da diese eine fast vollständige Abdeckung der notwendigen Bausteine eines solchen Frameworks liefern.

Ein mit der MDA-Infrastruktur eng verbundener Überblick über die aktuellen Softwaretools und Bibliotheken, die zur Realisierung der vertikalen und horizontalen Domänenaspekte herangezogen werden, rundet den Themenkomplex für die Praktiker unter den Lesern ab.

1.2 Ziele des Buches

Im Sinne eines *Proof-of-Concept* sollen in diesem Buch zuerst die Ziele untersucht werden, zu deren Lösung die MDA antritt und anschließend die Betrachtung nicht unterschlagen werden, ob der Ansatz diese Versprechungen mit den aktuell verfügbaren Techno-

logien prinzipiell erfüllen kann. Dabei sollen vor allem die folgenden Themenbereiche näher betrachtet werden:

1.2.1 Wie lässt sich die MDA einordnen?

Im Ansatz MDA finden sich viele Anleihen bekannter und weniger bekannter Prinzipien, Technologien und Vorgehensweisen aus dem weiten Feld des Software-Engineering. Wir wollen versuchen, diese Anleihen zu identifizieren und ihre Bedeutung innerhalb der Model-Driven Architecture näher zu beleuchten. Da einige Bestandteile bereits in der Vergangenheit große Versprechungen bezüglich der Effektivitätssteigerung der Softwareentwicklung nicht oder nur teilweise halten konnten, wollen wir außerdem versuchen zu klären, ob eine Kombination dieser Einzelteile zum jetzigen Zeitpunkt eventuell größere Aussichten auf Erfolg haben kann und die Gründe für unsere Schlussfolgerungen erläutern.

1.2.2 Darstellung der Konzepte der MDA

Hier sollen die technischen Grundlagen der MDA vorgestellt und die hinterliegenden Prinzipien erläutert werden. Modelle und (Meta-)Modellierung, spezifische Modellarten der MDA, der Plattformbegriff, Transformationen und Transformationsbeschreibungen sowie UML sind nur einige Begriffe, die verstanden und angewendet werden müssen, um die Vorstellungen der OMG umsetzen zu können. Wir wollen diese Begriffe in ihre übergeordneten Themenkomplexe einordnen und sowohl die spezifischen Einzelheiten, aber auch die großen Zusammenhänge ausführlich erläutern, da sie das Fundament des zweiten Teils der Betrachtungen bilden: der Untersuchung der Tragfähigkeit der MDA durch Konzeption und ausschnittsweise Implementierung eines Frameworks namens *genua* (vgl. Abschnitt 1.2.4) anhand eines Projekt-Fallbeispiels.

1.2.3 Koordination und Kombination

Nicht nur die Machbarkeit der technologischen Umsetzung der MDA-Prinzipien ist zu zeigen, auch die organisatorischen Aspekte müssen näher untersucht werden. So setzt die MDA a priori weder einen dezidierten Software-Entwicklungsprozess als Grundlage voraus, noch werden von der OMG selbst irgendwelche Vorschläge bezüglich des zu verwendenden Prozessmodells gemacht. Zu untersuchen ist also, was ein Prozess mindestens leisten muss, um die zusätzlichen Anforderungen, die die Vorgehensweise erfordert, erfüllen zu können. Beantwortet werden sollen also unter anderem folgende Fragen: Welche Anforderung stellt MDA an Organisation

und Prozesse? Welche Implikationen ergeben sich? Und wie können bestehende Strukturen und Vorgehensmuster so angepasst werden, dass diese Anforderungen erfüllt werden.

1.2.4 genua – prototypisches MDA Framework

Hier wird versucht die theoretisch erarbeiteten Anforderungen an eine technische Infrastruktur zur Stützung der MDA prototypisch umzusetzen. In einem Fallbeispiel wird unter dem Namen *genua* ein MDA-Framework entwickelt, das aus den geforderten Tools wie Generatoren, Transformatoren, Profilen usw. besteht, die in den vorangegangenen Kapiteln als unverzichtbar identifiziert wurden.

Die Technologien, die zur Umsetzung ausgewählt wurden, werden erklärt und das Zusammenspiel im MDA-Prozess anhand eines zusammenhängenden Fallbeispiels auszugsweise erläutert. Untersucht wird so die Tragfähigkeit und Integrierbarkeit der zur Zeit verfügbaren Technologielandschaft gegenüber Ansprüchen der MDA. Hierbei werden dezidiert nur frei verfügbare Open-Source Elemente und frei verfügbare Standards betrachtet.

1.2.5 Fazit – Ist MDA endlich die silberne Kugel?

Die in den vorigen Kapiteln gewonnenen Einsichten und Erkenntnisse werden hier in einer komprimierten Form zusammengefasst und die Tragfähigkeit der „Idee MDA“ von den Autoren anhand der gemachten Erfahrungen und theoretischen Überlegungen bewertet. Zusätzlich sollen ausgewählte Schwachstellen noch einmal explizit genannt und ein Blick in die Glaskugel auf die nähere Zukunft der MDA versucht werden.

1.3 Überblick und Leitfaden zum Lesen

Die Konzeption des Buchaufbaus folgt einer Dreiteilung: Teil I (Kapitel 2–5) beschreibt die theoretischen Grundlagen des Ansatzes (die *Architektur* der MDA). Im zweiten Teil (Kapitel 6) werden die Auswirkungen der vorgestellten Konzepte auf die Prozesse und die Organisation von Unternehmen näher beleuchtet. Teil III (Kapitel 7–9) bringt die Ideen zur Anwendung und setzt sie mit konkreten Technologien zur Umsetzung in Beziehung.

Es folgt ein Überblick über den Inhalt der einzelnen Kapitel:

Kapitel 2: MDA – Überblick und Orientierung

Hier werden die grundlegenden Konzepte und Begriffe erläutert, auf denen die MDA basiert. Außerdem wird der Ansatz in den „historischen Kontext" des Software-Engineering eingebettet und die Beziehungen zwischen „alten" Ideen und „neuer" MDA hergestellt.

Kapitel 3: Modellierung

Als einem der Hauptwerkzeuge der MDA wird der Modellierung ein eigenes Kapitel gewidmet. Hier werden die entsprechenden grundlegenden Begriffe und Konzepte eingeführt und das Werkzeug der OMG, die Unified Modeling Language (UML) sowie deren Spezifikationen unter die Lupe genommen. Weitere behandelte Punkte sind: formale Modellierung, Metamodellierung, MOF, Action Semantics, UML Repository usw.

Kapitel 4: Modelle der MDA

Hier wird der „Lebenszyklus" der Modellarten der MDA betrachtet und ein erstes durchgängiges Beispiel für konkrete Modelle vom anwendungs-unahängigen Modell (CIM) bis zur fertigen Applikation gezeigt.

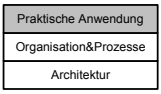
Kapitel 5: Transformation

Der Übergang von einem Modell in ein anderes wird in der MDA *Transformation* genannt. Dieses Kapitel schafft die Grundlagen zu diesem Themengebiet, die erst die Beschäftigung mit der Erstellung von Werkzeugen zur automatisierten Transformation sowie der Generierung von Quellcode als spezielle Unterart ermöglichen.

Kapitel 6: Koordination und Kombination

Nach all den „harten Fakten" spielen hier die „weichen" Aspekte der Einführung der MDA in bestehende Unternehmen eine Rolle. Die Integration des Ansatzes in bereits bestehende Software-Prozesse wird hier ebenso beleuchtet wie die Auswirkung des Ansatzes auf die Projektleiter und Entwickler bzw. deren bestehende Aufgaben und Rollen. Weiterhin werden die Anforderungen beleuchtet, die sich an die Unternehmensorganisation ergeben und die speziellen Aufgaben in der Einführungsphase (der MDA) näher betrachtet.

Kapitel 7–9: Vorstellung und Durchführung des Fallbeispiels



Anhand eines Fallbeispiels werden hier die im ersten Teil des Buches erarbeiteten Konzepte dem Lackmustest unterzogen. Konkrete

mögliche Lösungen für technische und konzeptionelle Probleme werden angegeben und auszugsweise detailliert erläutert.

Kapitel 10: /lost+found

Hier werden die Aspekte aufgegriffen, die wir Ihnen nicht vorenthalten wollten, jedoch „nirgendwo sonst so richtig gepasst haben". Weniger ein in sich geschlossenes Kapitel als eine Fundgrube interessanter Ideen und Gedankengänge finden sich hier Aspekte, die das Gesamtbild MDA weiter schärfen.

Kapitel 11: Ende gut – alles gut?

Tief durchatmend werden hier die im Laufe der vielen Kapitel erworbenen Erkenntnisse noch einmal zusammengefasst. Dabei werden wir auch einen vorsichtigen Ausblick in die Zukunft der Softwareentwicklung wagen und auch die Möglichkeit und deren Ursachen für den Fall betrachten, dass sich die MDA *nicht* durchsetzt. Natürlich darf an dieser Stelle auch ein ordentliches Schlusswort nicht fehlen.

Anhang A: UML Schnellreferenz

Im Sinne einer Referenz werden hier alle dreizehn Diagrammarten der UML 2.x aufgeführt und die wichtigsten Notationselemente vorgestellt und erläutert.

Anhang B: OOGPM

Ein Mittel, die Welt der Geschäftsprozessmodellierung mit der MDA zu verbinden, stellt die OOGPM (Objektorientierte Geschäftsprozessmodellierung, [OWS+03]) dar. Hierzu definiert sie sowohl die Methode als auch die notwendigen Profile, um die Ergebnisse als UML Modelle festhalten zu können. An dieser Stelle wird die OOGPM kurz vorgestellt und die wichtigsten Ergebnistypen samt entsprechender Beispieldiagramme erläutert.

1.4 Konventionen

Bei der Gestaltung des Buches wurden Konventionen bei der Wahl der Schrifttypen und der Verwendung von Sprachmitteln verwendet, die an dieser Stelle kurz erläutert werden:

- **Deutsch/Englisch**: Grundsätzlich wurde die im Alltag der Autoren gängige Form gewählt (also etwa *Bridge* und nicht *Brücke*). Wo erforderlich wurden diese in die entsprechende

Form der anderen Sprache übersetzt und in Klammern an den Originalbegriff angehängt. Oftmals wurden der Vollständigkeit halber beide Formen angegeben, um dem Leser eine Orientierung in den Literaturangaben, zu ermöglichen.

- **Codebeispiele**: Codebeispiele und genereller alle textlichen Artefakte, wie Auszüge aus Deskriptoren etc., sind in `Courier New` angegeben (Beispiel: Klasse `Person`).
- **Schlüsselwörter**: Begriffe, die im jeweiligen Kontext eine prominente Bedeutung besitzen sowie Schlüsselkonzepte wurden zur Hervorhebung in *Kursivschrift* gesetzt.
- **Literaturverzeichnis:** Zur besseren Übersichtlichkeit wurden neben dem Gesamtverzeichnis am Ende des Buches die kapitelrelevanten Einträge auch am Ende jedes Kapitels aufgeführt.
- **Index:** Zum besseren Auffinden relevanter Textstellen bei Benutzung des Buches als Referenz und Nachschlagewerk wurde ein ausführlicher Index erstellt, der am Ende des Buches zu finden ist und alle relevanten Schlüsselbegriffe enthält.
- **Marginalien:** Zur Erleichterung der Orientierung innerhalb der einzelnen Kapitel und Abschnitte wurden Leitbegriffe als Marginalie an den Rand der Seite aufgenommen.

1.5 Weitere Informationen

Weitere Informationen, Errata, Quelltexte sowie ein Diskussionsforum zum Austausch untereinander und direkt mit uns, finden sich unter:

```
www.mda-buch.info
```

Literatur

[Eclipse] Eclipse.org: *Eclipse.org Main Page*. http://www.eclipse.org/ (letzter Abruf Mai 2005)

[OWS+03] Bernd Oesterreich, Christian Weiss, Claudia Schröder, Tim Weilkiens und Alexander Lenhard: *Objektorientierte Geschäftsprozessmodellierung mit der UML*. dpunkt, 2003.

2 MDA – Überblick und Orientierung

„Als es noch keine Computer gab, war das Programmieren noch relativ einfach."

Edsger Wybe Dijkstra

In diesem Kapitel werden wir uns mit den Prinzipien der Model-Driven Architecture (MDA) vertraut machen sowie Grundlagen und Basiskonzepte einführen, die eine grobe Einordnung der Konzepte und Techniken ermöglichen und dann in späteren Kapiteln detaillierter beschrieben werden.

Zu Beginn wollen wir aber einen kleinen Blick zurück in die Vergangenheit werfen – auf mittlerweile 50 Jahre industrielle Software-Entwicklung. Dies soll uns dabei helfen, die Motive der Object Management Group (OMG) zu verstehen, die schließlich zur Formulierung des Model-Driven Architecture Ansatzes geführt haben.

2.1 Motivation modellgetriebener Ansätze

Software hat sich im Laufe der letzten Jahrzehnte von ihrem ursprünglichen Nischendasein zur nunmehr kritischen Rolle innerhalb von Unternehmen entwickelt. Viele betriebswirtschaftliche Basisoperationen laufen heute softwaretechnisch automatisiert oder zumindest IT-gestützt ab. Software ist damit zu einem wichtigen Erfolgsfaktor und mitunter zu einem entscheidenden Wettbewerbsvorteil vieler Unternehmen geworden (vgl. [SF03]).

Dem daraus resultierenden steigenden Bedarf an softwaretechnischen Lösungen in der betriebswirtschaftlichen Praxis stehen zunehmend gegenüber:

- eine immer schwieriger zu beherrschende Komplexität,
- stetig steigende Anforderungen an die Leistungsfähigkeit, Zuverlässigkeit und Qualität,
- kurze Technologiezyklen, die sowohl die Hardware- als auch Software-Plattformen betreffen,
- häufige Anforderungsänderungen,
- und (insbesondere in Zeiten konjunktureller Schwächeperioden) ein hoher Druck zur Kostenreduzierung.

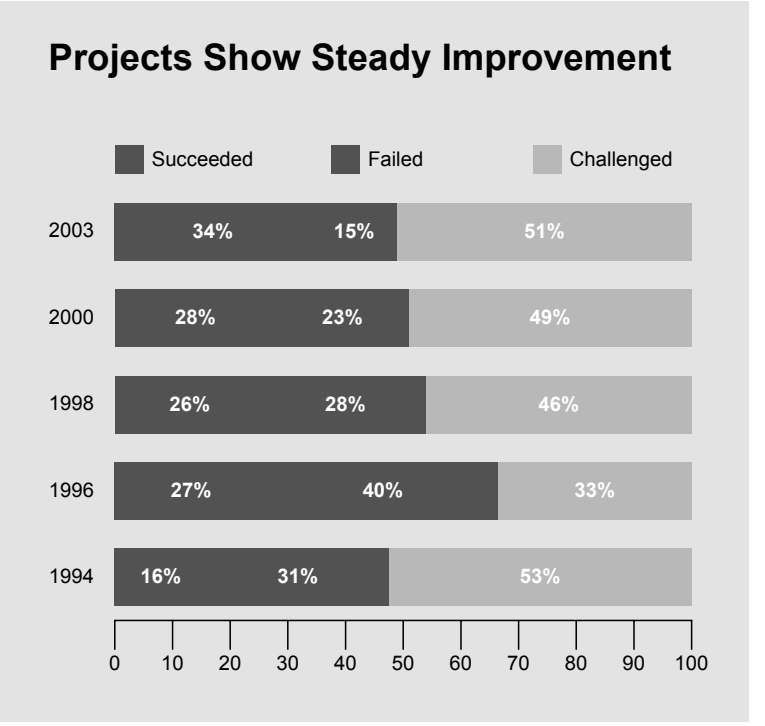


Abb. 2.1 Software-Projekte zeigen stetige Verbesserung (Quelle: [JBCR01] ergänzt durch [SG03])

Spätestens mit dem „Erkennen" der so genannten Software-Krise [NRB76] wurde daher die Forderung nach ingenieurmäßiger Software-Erstellung immer lauter. Trotz aller Erfahrungen, neuer Technologien und vielfältiger Forschung fehlen der Software-Industrie auch heute noch Entwicklungsprozesse, die an die Planbarkeit, Zuverlässigkeit, Effektivität, Effizienz und Flexibilität der „althergebrachten" Industrie auch nur annähernd herankommen.

Knowledge-Work vs. Industrielle Fertigung

Ein Versuch, unreflektiert Prozesse aus der Welt der klassischen Industrie in die Software-Welt zu übernehmen, scheitert häufig auf Grund der grundsätzlichen Verschiedenheit der materiellen Fertigung und der sich immer neu stellenden Anforderungen an die Entwicklung „geistiger" Lösungen. Die große Anzahl abgebrochener oder weit über dem ursprünglich angenommenen Budget beendeter Projekte in der Software-Industrie zeigt das.

So hat sich der relative Anteil erfolgreich durchgeführter Software-Projekte laut Studien der Standish Group von 16% im Jahre 1994 auf 34% im Jahre 2003 zwar immerhin mehr als verdoppelt [SG03], andererseits bedeutet dies aber auch, dass immer noch zwei Drittel der Projekte nicht wie geplant beendet werden (siehe auch Abb. 2.1). Projekt-Beispiele der jüngeren Vergangenheit wie *Cheops* [COWO00], *Fiscus* [HO04] oder etwa *Toll Collect* [CT03] bilden hier nur die Spitze des Eisbergs.

Immerhin lässt sich aber ein Fortschritt feststellen. Wir wollen im Folgenden einmal Revue passieren lassen, welche Anstrengungen zu diesen Erfolgen geführt haben.

2.1.1 Die Geschichte der Softwareentwicklung – ein historischer Abriss

Maschinen-sprachen

Beginnen möchten wir in den 50er-Jahren: Zu dieser Zeit bestimmten die knappen und teuren Hardware-Ressourcen die Computer-Welt. Die Erstellung von Software spielte deshalb gegenüber der Hardware-Entwicklung eine untergeordnete Rolle und erfolgte quasi „nebenbei". Software hatte noch eine beherrschbare Größe und wurde in Maschinensprache binär kodiert.

Assembler-Sprachen

Die Software-Systeme wurden komplexer und schon bald musste man erkennen, dass die für den Menschen schwer verständlichen maschinensprachlichen Binärcodes für Befehle und Operanden nicht weiter dazu geeignet waren, um Software zu entwickeln. Die Maschinensprachen wurden durch die Assembler-Sprachen (kurz Assembler) abgelöst, die mnemonische Symbole, Marken und Makros als Arbeitserleichterung für den Entwickler brachte. Mnemonische Symbole lösten die angesprochenen Binärcodierungen ab, Marken und Makros ermöglichten die Abstraktion von einzelnen Instruktionen. Der Assemblierer übersetzte den in Assembler-Sprache geschriebenen Code in ein Maschinenprogramm.

Problemorientierte/Höhere Programmiersprachen

Die Assembler-Sprachen waren nach wie vor maschinenorientiert, gefragt waren aber zunehmend Programmiersprachen, mit denen man die zu behandelnden Probleme direkter formulieren konnte. Die Programmiersprache FORTRAN (FORmula TRANslator), die 1954-1958 von Jim Backus entwickelt worden war, gilt als erste problemorientierte bzw. höhere Programmiersprache. Sie wurde für mathematisch-technische Probleme konzipiert und ist Vorläuferin vieler weiterer höherer Programmiersprachen die folgen sollten. Musste man sich bei der Assembler-Programmierung um Prozessorregister, Stack und Statusflags weitgehend selbst kümmern, konnte man mit den höheren Programmiersprachen von diesen Dingen abstrahieren. Ein Compiler erledigte stattdessen diese Arbeit und sorgte für die Übersetzung in ein ausführbares Programm.

Software-Krise und Software-Engineering

Die Hardware-Preise fielen, der Bedarf an Software wuchs, und schon bald wurden die programmierten Systeme größenmäßig so komplex, dass sie unhandhabbar wurden. Auf einer NATO-Konferenz im Jahre 1968 reagierte man auf diesen Missstand – die Software-Krise wurde ausgerufen und mit ihr wurde die Forderung nach einer angemessenen Ingenieursdisziplin immer vehementer. Bis zu diesem Zeitpunkt war die Software-Erstellung eher als künstlerische Tätigkeit aufgefasst worden, nun sollte eine strukturierte Vorgehensweise die „Kunst" verdrängen. In dieser Zeit wurde der Begriff des Software-Engineering geprägt (vgl. [Bau93] bzw. [Boe76]).

Strukturierte Programmierung

Mit komfortablen Kontrollstruktur- und Fallunterscheidungsmöglichkeiten begann man nun strukturiert – etwa in Pascal oder in C – zu programmieren [DDH72]. Die neue Disziplin Software-Engineering hatte sich aber nicht nur strukturierte Programmierung auf die Fahnen geschrieben, sondern insbesondere auch systematische Methoden. Diese Lücke wurde unter anderem durch die Strukturierte Analyse (SA) [DeM78] und das Strukturierte Design (SD) [YC79] geschlossen.

Strukturierte Analyse und Strukturiertes Design

In der *Strukturierten Analyse* verwendete man hierarchisch angeordnete Datenflussdiagramme zur abstrakten Modellierung von Prozessen. Mini-Spezifikationen (z. B. in Pseudo-Code) dienten dabei zur Beschreibung von nicht weiter verfeinerten Prozessen und Data Dictionaries stellten die einheitliche Datendefinition sicher. Die in der Strukturierten Analyse ausgemachten Funktionen wurden dann im Strukturiertem Design in hierarchisch aufgebaute Module zerlegt und in Strukturdiagrammen festgehalten. Hierbei dienten die Diagramme der Visualisierung von Programmabläufen und der Be-

schreibung der Schnittstellen zwischen den Modulen. Diagramme bzw. Modelle erhielten in den strukturierten Methoden einen hohen Stellenwert und ein Markt für Modellierungs-Werkzeuge entstand: Willkommen in der Welt des Computer Aided Software Engineering (CASE)!

Wiederverwendungskrise

Die *Wiederverwendungskrise* [Qui94] gegen Ende der 80er-Jahre machte diese Idylle zunichte. Die abermals drastisch in ihrer Komplexität gewachsenen Software-Systeme erzwangen erneut ein Umdenken bei der Software-Erstellung. Diesmal wollte man insbesondere der Wiederverwendbarkeit von Software Rechnung tragen. So hatte man zwar im Laufe der Jahre gewaltige Software-Bestände angehäuft, aber gleichzeitig das Problem, diese in geeigneter Form auch nur teilweise wieder zu verwenden.

Objektorientiertes Paradigma

Die Lösung sollte das Objektorientierte Paradigma liefern; ein Ansatz, der der natürlichen Denkweise sehr nahe kommt: Ein System besteht aus vielen Objekten und jedes dieser Objekte besitzt ein definiertes Verhalten, einen inneren Zustand und eine eindeutige Identität. Bereits in den 70er-Jahren hatte man sich diesem Ansatz in wissenschaftlichen Veröffentlichungen gewidmet, der Durchbruch sollte erst jetzt erfolgen.

Ole-Johan Dahl und Kristen Nygaard hatten bereits in den 60er-Jahren eine entsprechende Programmiersprache entwickelt: SIMULA (SIMUlation LAnguage). Ursprünglich für diskrete Ereignis-Simulation entwickelt, kamen ihre Konzepte so zu spätem Ruhm – die breite Akzeptanz wiederum, sollte später aber Sprachen wie Smalltalk, C++ und Java vorbehalten bleiben.

Daten und Funktion wurden fortan nicht mehr getrennt voneinander betrachtet, sondern zusammenhängend in Objekten. Klassen nahmen dabei die Objekte auf, die dem gleichen Verhaltensschema folgten und die gleiche innere Struktur besaßen. Vererbung und Polymorphie sollten für die oben angesprochene Wiederverwendbarkeit Sorge tragen. Um die Übersetzung des Programm-Codes in ein ausführbares Programm kümmerte sich weiterhin ein – um OO-Konstrukte aufgebohrter – Compiler.

Objektorientierte Analyse und Objektorientiertes Design

Der breiten Akzeptanz folgend schossen Anfang der 90er-Jahre adäquate objektorientierte Methoden wie Pilze aus dem Boden - Rumbaughs Object Modeling Technique (OMT) [RBP+91] und Boochs Object-Oriented Design (OOD) [Boo93] waren bzw. sind wohl die zwei Bekanntesten unter ihnen. Sie intensivierten abermals den Gebrauch von Modellen und entsprechender Modellierungs-Werkzeuge in der Software-Entwicklung. Aus der Vielzahl der

objektorientierten Methoden ging die Unified Modeling Language (UML) hervor, eine standardisierte Modellierungssprache zur Spezifikation objektorientierter Systeme.

Middleware, Komponenten und Applikationsserver

In jüngster Vergangenheit bekam der Entwickler Unterstützung durch eine Reihe verschiedener Konzepte wie Middleware [MW], Komponenten und Applikationsservern. Durch ihre Verbreitung wurden die Rechnergrenzen endgültig gesprengt und mit ihr eine Transparenz der Netzwerkkommunikation geschaffen. Komponenten brachten das Paradigma des Zusammenbaus vorgefertigter Software-Einheiten. Die beiden Konzepte wurden mithilfe von Object Request Brokern (ORBs) bzw. Applikationsservern zur verteilten Objekttechnologie bzw. Distributed Object Technology (DOT) verheiratet. Zusammengenommen verschafften sie uns insbesondere Ansätze zur Abstraktion von Verteilung, Persistenz, Transaktionalität und Sicherheit.

2.1.2 Die Gegenwart

Damit sind wir am Ende unserer kleinen Rückblicks angekommen. Gegenwärtig wird das Objektorientierte Paradigma auf breiter Basis erfolgreich um- und eingesetzt und ist aus der Entwicklungspraxis nicht mehr wegzudenken. Mittlerweile sehen wir uns hochkomplexen Zielarchitekturen gegenüber, müssen die „Jugendsünden" in Form bestehender Anwendungen integrieren und dabei die verschiedenartigsten Technologien einbeziehen. Auch hierbei unterstützen uns Modelle und das damit verbundene Prinzip der Abstraktion, indem sie helfen, komplexe Sachverhalte kompakt darzustellen.

Welche Schlussfolgerung lässt sich nun ziehen?

Betrachtet man die eben beschriebene Entwicklung der Software-Entwicklung unter dem Gesichtspunkt eines gemeinsamen „Trends" fallen folgende Punkte besonders ins Auge:

*Komplexitätsbeherrschung als **das** Problem*

- Der Fortschritt in der Software-Entwicklung ist eng verbunden mit der Bewältigung von Komplexität – mitunter spricht man deshalb bei der Komplexitätsbeherrschung auch von *dem* Problem des Software-Engineering.

Abstraktion als fortwährender Trend

- Zu beobachten ist weiterhin, dass immer, wenn die Komplexität der zu erstellenden Software-Systeme auf ein nicht mehr zu beherrschendes Maß angewachsen war, die Lösung in der Vergangenheit in einem Abstraktionssprung auf eine höhere semantische Ebene zu finden war.

Diese Beobachtungen lassen sich auch an der Evolution der Programmiersprachen festmachen: von Maschinensprachen über Assembler-Sprachen, Prozeduralen Sprachen hin zu Objektorientierten Sprachen. Insofern kann bei der Abstraktion von einem fortwährenden Trend des Software-Engineering bzw. allgemein der Informatik gesprochen werden.

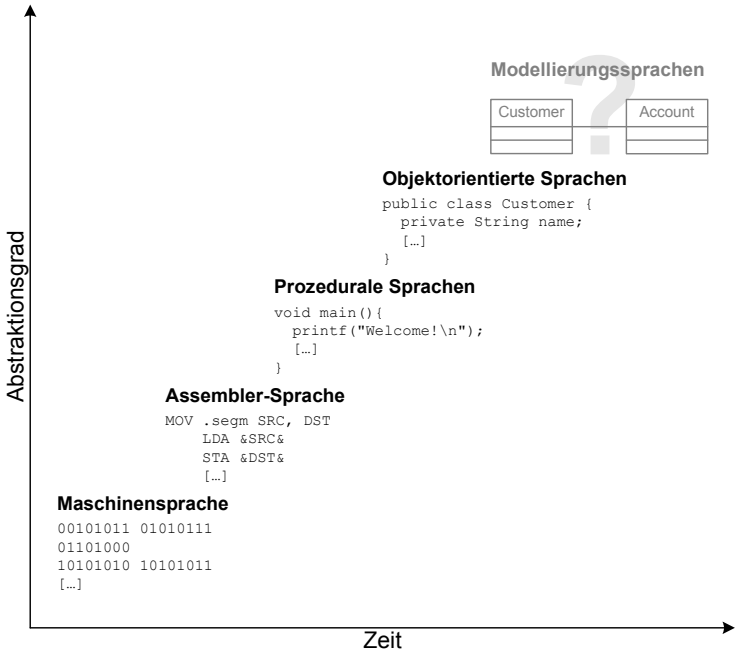


Abb. 2.2 Steigender Abstraktionsgrad

Wie bereits erwähnt sind Modelle ein weiteres Werkzeug, das uns bei der Beherrschung dieser Komplexität hilft, indem sie frei nach dem Motto *„teile und herrsche"* einen Teil der selbigen verdecken. Diese Beobachtung liefert den dritten Punkt, den wir als Trend in der Historie der betrachteten Entwicklung festmachen wollen:

- Modelle rücken mehr und mehr in den Mittelpunkt des Entwicklungsprozesses, insbesondere in Analyse und Entwurf.

Zunehmende Gewichtung von Modellen

Wie geht es weiter?

Angenommen der ausgemachte Trend wäre eine Tatsache, wäre es dann nicht konsequent zu folgern, dass Modellierungssprachen die