1

Einführung

Der wahre Zweck eines Buches ist,
den Geist hinterrücks zum
eigenen Denken zu verleiten.

Christopher Darlington Morley

Jüngste Beispiele zeigen eindrucksvoll, wie teuer falsche oder fehlerhafte Software beim Absturz einer Rakete oder bei Maut-Projekten werden können.

Um die stetig wachsende Komplexität von Software-basierten Projekten und Produkten sowohl im Bereich betrieblicher Informations- als auch eingebetterer Systeme beherrschbar zu machen, wurde in den letzten Jahren ein wirkungsvolles Portfolio an Konzepten, Techniken und Methoden entwickelt, die die Softwaretechnik zu einer erwachsenen Ingeniersdisziplin heranreifen lassen.

Das Portfolio ist zwar noch nicht vollständig ausgereift, muss aber insbesondere in dem derzeitigen industriellen Softwareentwicklungsprozess sehr viel mehr noch etabliert werden. Die Fähigkeiten moderner Programmiersprachen, Klassenbibliotheken und vorhandener Softwareentwicklungswerkzeuge erlauben uns heute den Einsatz von Vorgehensweisen, die noch vor kurzer Zeit nicht realisierbar schienen.

Als Teil dieses Portfolios behandelt dieses Buch eine auf der UML basierende Methodik, bei der vor allem Techniken zum praktischen Einsatz der UML im Vordergrund stehen. Als wichtigste Techniken werden dabei

- Generierung von Code aus Modellen,
- Modellierung von Testfällen und
- Weiterentwicklung durch Refactoring von Modellen

erkannt und in diesem Buch studiert.

Dabei basiert dieser Band 2 sehr stark auf dem ersten Band „Modellierung mit UML. Sprache, Konzepte und Methodik.“ [Rum04c], in dem das Sprachprofil UML/P detailliert erklärt ist. Es ist daher zu empfehelen, bei der Lektüre dieses Bands [Rum04b] den ersten Band [Rum04c] griffbereit zu

halten, obwohl Teile von Band 1 in kompakter Form in Kapitel 2 wiederholt werden.

1.1 Ziele und Inhalte von Band 1

Gemeinsames Mission Statement beider Bände: Es ist ein Kernziel, für das genannte Portfolio grundlegende Techniken zur modellbasierten Entwicklung zur Verfügung zu stellen. Dabei wird in Band 1 eine Variante der UML vorgestellt, die speziell zur effizienten Entwicklung qualitativ hochwertiger Software und Software-basierter Systeme geeignet ist. Darauf aufbauend enthält dieser Band Techniken zur Generierung von Code, von Testfällen und zum Refactoring der UML/P.

UML-Standard: Der UML 2.0-Standard muss sehr viele Anforderungen aus unterschiedlichen Gegebenheiten heraus erfüllen und ist daher notwendigerweise überladen. Viele Elemente des Standards sind für unsere Zwecke nicht oder nicht in der gegebenen Form sinnvoll, während andere Sprachkonzepte ganz fehlen. Deshalb wird in diesem Buch ein angepasstes und mit UML/P bezeichnetes Sprachprofil der UML vorgestellt. UML/P wird dadurch für die vorgeschlagenen Entwicklungstechniken im Entwurf, in der Implementierung und in der Wartung optimiert und so in agilen Entwicklungsmethoden besser einsetzbar.

Band 1 konzentriert sich vor allem auf die Einführung des Sprachprofils und einer allgemeinen Übersicht zur vorgeschlagenen Methodik.

Die UML/P ist als Ergebnis mehrerer Grundlagen- und Anwendungsprojekte entstanden. Insbesondere das in Anhang D, Band 1 dargestellte Anwendungsbeispiel wurde soweit möglich unter Verwendung der hier beschriebenen Prinzipien entwickelt. Das Auktionssystem ist auch deshalb zur Demonstration der in den beiden Büchern entwickelten Techniken geeignet ideal, weil Veränderungen des Geschäftsmodells oder der Unternehmensumgebung in dieser Anwendungsdomäne besonders häufig sind. Flexible und dennoch qualitativ hochwertige Softwareentwicklung ist für diesen Bereich essentiell.

Objektorientierung und Java: Für neue Geschäftsanwendungen wird heute primär Objekttechnologie eingesetzt. Die Existenz vielseitiger Klassenbibliotheken und Frameworks, die vorhandenen Werkzeuge und nicht zuletzt der weitgehend gelungene Sprachentwurf begründen den Erfolg der Programmiersprache Java. Das UML-Sprachprofil UML/P und die darauf aufbauenden Entwicklungstechniken werden daher auf Java zugeschnitten.

Brücke zwischen UML und agilen Methoden: Gleichzeitig bilden die beiden Bücher zwischen den eher als unvereinbar geltenden Ansätzen der agilen Methoden und der Modellierungssprache UML eine elegante Brücke. Agile Methoden und insbesondere Extreme Programming besitzen eine Reihe von interessanten Techniken und Prinzipien, die das Portfolio der Softwaretechnik für bestimmte Projekttypen bereichern. Merkmale dieser Tech-

niken sind der weitgehende Verzicht auf Dokumentation, die Konzentration auf Flexibilität, Optimierung der Time-To-Market und Minimierung der verbrauchten Ressourcen bei gleichzeitiger Sicherung der geforderten Qualität. Damit sind agile Methoden für die Ziele dieses Buchs als Grundlage gut geeignet.

Neue agile Vorgehensweise auf Basis der UML/P: Die UML wird als Notation für eine Reihe von Aktivitäten, wie Geschäftsfallmodellierung, Soll- und Ist-Analyse sowie Grob- und Fein-Entwurf in verschiedenen Granularitätsstufen eingesetzt. Die Artefakte der UML stellen damit einen wesentlichen Grundstein für die Planung und Kontrolle von Meilenstein-getriebenen Softwareentwicklungsprojekten dar. Deshalb wird die UML vor allem in plan-getriebenen Projekten mit relativ hoher Dokumentationsleistung und der daraus resultierenden Schwerfälligkeit eingesetzt. Nun ist die UML aber kompakter, semantisch reichhaltiger und besser geeignet, komplexe Sachverhalte darzustellen, als eine normale Programmiersprache. Sie bietet dadurch für die Modellierung von Testfällen sowie für die transformationelle Evolution von Softwaresystemen wesentliche Vorteile. Auf Basis einer Diskussion agiler Methoden und der darin enthaltenen Konzepte wird in Band 1 eine neue agile Methode skizziert, die das UML/P-Sprachprofil als Grundlage für viele Aktivitäten nutzt, ohne die Schwerfälligkeit typischer UML-basierter Methoden zu importieren.

Die beschriebenen Ziele wurden in Band 1 in folgenden Kapitel umgesetzt:

1 Einführung

2 Agile und UML-basierte Methodik
beschreibt die Vorgehensweise zum Einsatz der UML/P im Kontext vorhandener agiler Methoden.

3 Klassendiagramme
führt Form und Verwendung von Klassendiagrammen ein.

4 Object Constraint Language
diskutiert eine syntaktisch auf Java angepasste, sprachlich erweiterte und semantisch konsolidierte Fassung der textuellen Beschreibunsgsprache OCL.

5 Objektdiagramme
diskutiert Sprache und methodischen Einsatz der Objektdiagramme sowie deren Integration mit der OCL-Logik, um damit eine "Logik der Diagramme" zu ermöglichen, in der unerwünschte Situationen, Alternativen und Kombinationen beschrieben werden können.

6 Statecharts
beinhaltet neben der Einführung der Statecharts eine Sammlung von Transformationen zur deren semantikerhaltender Vereinfachung.

7 Sequenzdiagramme
beschreibt Form, Bedeutung und Verwendung von Sequenzdiagrammen.

A Sprachdarstellung durch Syntaxklassendiagramme
bietet eine Kombination aus Extended-Backus-Naur-Form (EBNF) und spezialisierten Klassendiagrammen zur Darstellung der abstrakten Syntax (Metamodell) der UML/P.

B Java
beschreibt die abstrakte Syntax des genutzten Teils von Java.

C Syntax der UML/P
beschreibt die abstrakte Syntax der UML/P.

D Anwendungsbeispiel: Internetbasiertes Auktionssystem
erläutert Hintergrundinformation zu dem in beiden Bänden verwendeten Beispiel des Auktionssystems.

1.2 Ergänzende Ziele dieses Buchs

Um die Effizienz in einem Projekt zu steigern, ist es notwendig, den Entwicklern effektive Notationen, Techniken und Methoden zur Verfügung zu stellen. Weil das primäre Ziel jeder Softwareentwicklung das lauffähige und korrekt implementierte Produktionssystem ist, sollte der Einsatz der UML nicht nur zur Dokumentation von Entwürfen dienen. Stattdessen ist die automatisierte Umsetzung in Code durch *Codegeneratoren*, die Definition von *Testfällen* mit der UML/P zur Qualitätssicherung und die Evolution von UML-Modellen mit *Refactoring*-Techniken essentiell.

Die Kombination von Codegenerierung, Testfallmodellierung und Refactoring bietet dabei wesentliche Synergie-Effekte, die zum Beispiel bei der Qualitätssicherung, bei der erhöhten Wiederverwendung und der gesteigerten Fähigkeit zur Weiterentwicklung beitragen.

Codegenerierung: Zur effizienten Erstellung eines Systems ist eine gut parametrisierte Codegenerierung aus abstrakten Modellen essentiell. Die diskutierte Form der Codegenerierung erlaubt die kompakte und von der Technik weitgehend unabhängige Entwicklung von Modellen. für spezifische Domänen und Anwendungen. Erst bei der Generierung werden technologieabhängige Aspekte wie zum Beispiel Datenbankanbindung, Kommunikation oder GUI-Darstellung hinzugefügt. Dadurch wird die UML/P als Programmiersprache einsetzbar, und es entsteht kein konzeptueller Bruch zwischen Modellierungs- und Programmiersprache. Allerdings ist es wichtig, ausführbare und abstrakte Modelle im Softwareentwicklungsprozess explizit zu unterscheiden und jeweils adäquat einzusetzen.

Modellierung automatisierbarer Tests: Die systematische und effiziente Durchführung von Tests ist ein wesentlicher Bestandteil zur Sicherung der Qualität eines Systems. Ziel ist dabei, dass Tests nach ihrer Erstellung automatisiert ablaufen können. Codegenerierung wird daher nicht nur zur Entwicklung des Produktionssystems, sondern insbesondere auch für Testfälle eingesetzt, um so die Konsistenz zwischen Spezifikation und Implementierung zu prüfen. Der Einsatz der UML/P zur Testfallmodellierung ist daher

ein wesentlicher Bestandteil einer agilen Methodik. Dabei werden insbesondere Objektdiagramme, die OCL und Sequenzdiagramme zur Modellierung von Testfällen eingesetzt. Abbildung 1.1 skizziert die ersten beiden Ziele dieses Buchs.

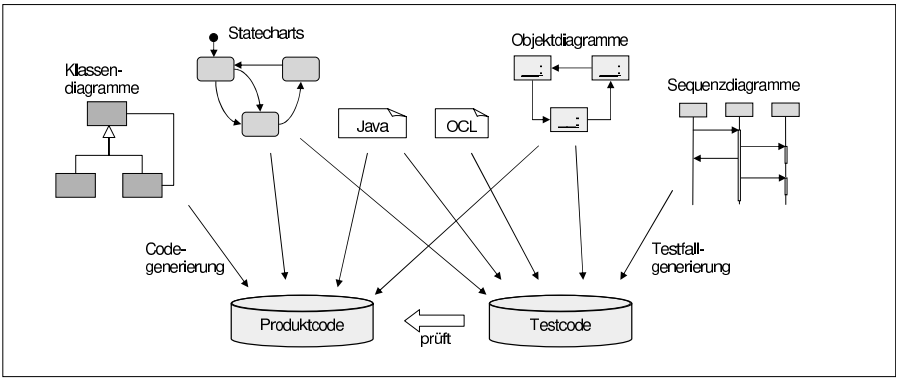

Abbildung 1.1. Notationen der UML/P

Evolution mit Refactoring: Die diskutierte Flexibilität, auf Änderungen der Anforderungen oder der Technologie schnell zu reagieren, erfordert eine Technik zur systematischen Anpassung des bereits vorhandenen Modells beziehungsweise der Implementierung. Die Evolution eines Systems in Bezug auf neue Anforderungen oder einem neuen Einsatzgebiet sowie der Behebung von Strukturdefiziten der Softwarearchitektur erfolgt idealerweise durch Refactoring-Techniken. Ein weiterer Schwerpunkt ist daher die Fundierung und Einbettung der Refactoring-Techniken in die allgemeinere Vorgehensweise zur Modelltransformation und die Diskussion, welche Arten von Refactoring-Regeln für die UML/P entwickelt oder von anderen Ansätzen übernommen werden können. Besonders betrachtet werden dabei Klassendiagramme, Statecharts und OCL.

Sowohl bei der Testfallmodellierung als auch bei den Refactoring-Techniken werden aus den fundierenden Theorien stammende Erkenntnisse dargestellt und auf die UML/P transferiert. Ziel des Buchs ist es, diese Konzepte anhand zahlreicher praktischer Beispiele zu erklären und in Form von Testmustern beziehungsweise Refactoring-Techniken für UML-Diagramme aufzubereiten.

Model Driven Architecture (MDA): Initiiert durch das Industriekonsortium OMG[1] wird derzeit an der Intensivierung des werkzeugbasierten Einsatzes der UML in der Softwareentwicklung gearbeitet. Die als MDA be-

[1] Die Object Management Group (OMG) ist für die Definition der UML in Form einer „Technical Recommendation“ zuständig.

zeichnete Technik bietet im Bereich Codegenerierung ähnliche Charakteristiken wie der hier diskutierte Ansatz. Jedoch fehlt eine geeignet angepasste Methodik. Der in diesem Buch vorgestellte Ansatz zum Refactoring bietet dazu eine adäquate Ergänzung zur „horizontalen" Transformation.

Abgrenzung: Mit den behandelten Techniken konzentriert sich dieses Buch vor allem auf die Unterstützung der Entwurfs-, Implementierungs- und Wartungsaktivitäten, allerdings ohne diese in allen Facetten abzudecken. Wichtig, aber unbehandelt sind auch Techniken und Notationen zur Erhebung und zum Management von Anforderungen, zur Projektplanung und -durchführung, zur Kontrolle sowie zum Versions- und Änderungsmanagement. Stattdessen wird an geeigneten Stellen auf entsprechende weiterführende Literatur verwiesen.

1.3 Überblick

Abbildung 1.2 erlaubt einen guten Überblick über den Matrix-artigen Aufbau weiter Teile beider Bände. Während dieser Band sich verstärkt um methodische Fragestellungen kümmert wird die UML/P im Band 1 erklärt.

	Allgemeines	Klassen-diagramme	OCL	Objekt-diagramme	Statecharts	Sequenz-diagramme
Einführung und Diskussion	*Kapitel 2 (Band 1)*	*Kapitel 3 (Band 1)*	*Kapitel 4 (Band 1)*	*Kapitel 5 (Band 1)*	*Kapitel 6 (Band 1)*	*Kapitel 7 (Band 1)*
Syntax	*Anhang A & B & C.1 (Band 1)*	*Anhang C.2 (Band 1)*	*Anhang C.3 (Band 1)*	*Anhang C.4 (Band 1)*	*Anhang C.5 (Band 1)*	*Anhang C.6 (Band 1)*
Codegenerierung	Kapitel 3	Kapitel 4.1	Kapitel 4.3	Kapitel 4.2	Kapitel 4.4	Kapitel 4.5
Testfallmodellierung	Kapitel 5	(Kapitel 7)	Abschnitte 6.2 & 6.3	Abschnitt 6.1	Abschnitt 6.5	Abschnitt 6.4
Refactoring	Kapitel 8	Abschnitte 9.1 & 9.2	Abschnitt 9.1	(Abschnitt 9.2)	*Abschnitt 6.6 (Band 1)*	(Abschnitt 9.2)

Abbildung 1.2. Übersicht über den Inhalt beider Bände

Das Kapitel 2 gibt eine kurze und kompakte Zusammenfassung von Band 1, [Rum04c]. Dabei wird vor allem das dort eingeführte Sprachprofil der UML/P sehr kompakt und daher unvollständig vorgestellt.

Die Kapitel 3 und 4 diskutieren prinzipielle und technische Problemstellungen zur Codegenerierung. Dies beinhaltet die Architektur und die Steuerung eines Codegenerators genauso wie die Frage, welche Teile der UML/P für Test- beziehungsweise für Produktionscode geeignet sind. Darauf aufbauend wird ein Mechanismus zur Beschreibung von Codegenerierung eingeführt. Ergänzend dazu wird anhand ausgewählter Teile der verschiedenen UML/P-Notationen gezeigt, wie daraus Test- und Produktionscode ge-

neriert werden kann. Dabei werden sowohl Alternativen diskutiert als auch die Kombination von Transformationen demonstriert.

Die Kapitel 5 und 6 erörtern die aus der Literatur bekannten Begriffsbildungen für Testverfahren und die beim Testen zu beachtenden Besonderheiten, die aufgrund der Nutzung der UML/P als Test- und Implementierungssprache entstehen. Es wird beschrieben, wie eine Architektur für die automatisierte Testausführung aussieht und wie UML/P-Diagramme eingesetzt werden, um Testfälle zu definieren.

Kapitel 7 zeigt anhand von Testmustern die Verwendbarkeit der UML/P-Diagramme zur Definition von Testfällen. Diese Testmuster enthalten Erfahrungswissen zur Definition von testbaren Programmen insbesondere für funktionale Tests von verteilten und nebenläufigen Softwaresystemen.

Die Kapitel 8 und 9 diskutieren Techniken zur Transformation von Modellen und Code und stellen damit Refactoring als semantikerhaltende Transformation auf eine fundierte Basis. Für Refactoring wird ein expliziter und praktisch verwendbarer Beobachtungsbegriff eingeführt und es wird diskutiert, wie vorhandene Refactoring-Techniken auf die UML/P übertragen werden können. Schließlich wird eine *additive Vorgehensweise* vorgeschlagen, die größere Refactorings unter Verwendung von OCL-Invarianten unterstützt.

1.4 Notationelle Konventionen

In diesem Buch werden mehrere Diagrammarten und textuelle Notationen genutzt. Damit sofort erkennbar ist, welches Diagramm oder welche textuelle Notation jeweils dargestellt ist, wird abweichend von der UML 2.0 rechts oben eine Marke in einer der in Abbildung 1.3 dargestellten Formen angegeben. Diese Form ist auch zur Markierung textueller Teile geeignet und flexibler als die UML 2.0-Markierung. Eine Marke wird einerseits als Orientierungshilfe und andererseits als Teil der UML/P eingesetzt, da ihr der Name des Diagramms und Diagramm-Eigenschaften in Form von Stereotypen beigefügt werden können. Vereinzelt kommen Spezialformen von Marken zum Einsatz, die weitgehend selbsterklärend sind.

Die textuellen Notationen wie Java-Code, OCL-Beschreibungen und textuelle Teile in Diagrammen basieren ausschließlich auf dem ASCII-Zeichensatz. Zur besseren Lesbarkeit werden einzelne Schlüsselwörter hervorgehoben oder unterstrichen.

Im Text werden folgende Sonderzeichen genutzt:

- Die Repräsentationsindikatoren „…“ und „©“ sind formaler Teil der UML/P und beschreiben, ob die in einem Diagramm dargestellte Repräsentation vollständig ist.
- Stereotypen werden in der Form ≪Stereotypname≫ angegeben. Merkmale haben die Form {`Merkmalsname=Wert`} oder {`Merkmalsname`}.

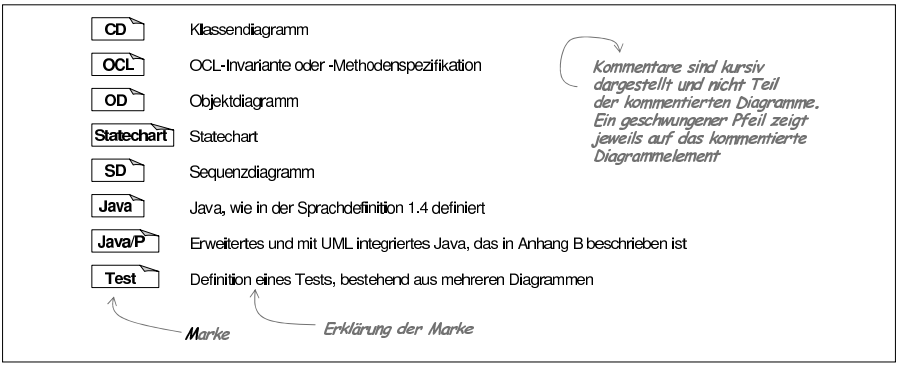

Abbildung 1.3. Marken für Diagramme und Textteile

Danksagung

Wie auch an Band 1 haben an der Erstellung dieses Buchs eine Reihe von Personen direkt oder indirekt mitgewirkt. Neben einem Verweis auf die Danksagung von Band 1 möchte ich insbesondere meinen Kolleginnen und Kollegen an der Technischen Universität Braunschweig für die freundliche Aufnahme sowie meinen Mitarbeiterinnen und Mitarbeitern für die hervorragende Unterstützung bei der Fertigstellung dieses Buchs bedanken. Im Kontext des gerade in Aufbau befindlichen neuen Instituts für Software Systems Engineering ist es durchaus ambitioniert zwei Bücher parallel zu einer Reihe neuer Vorlesungen, Seminaren und Praktika fertigzustellen.