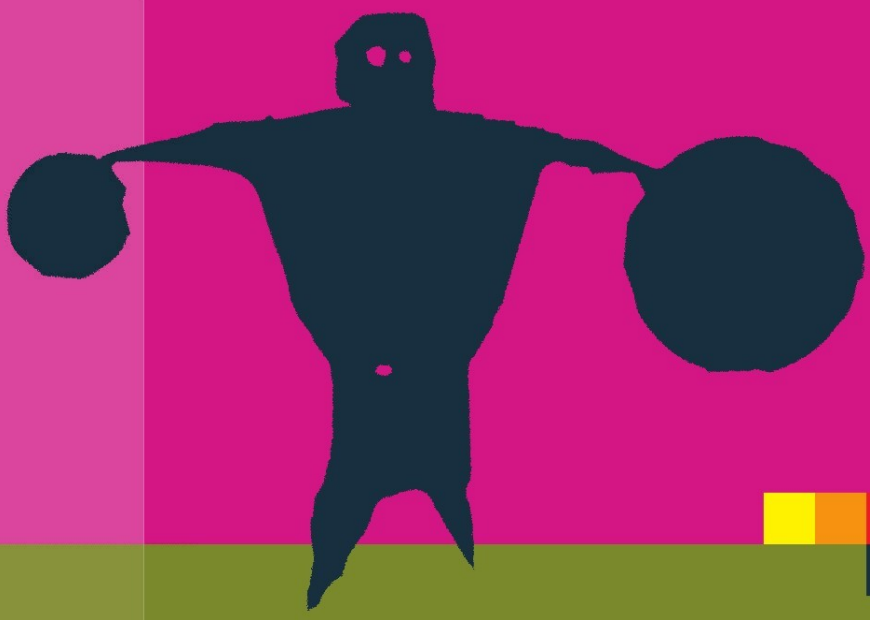

Xpert.press

Bernhard Rumpe

Agile Modellierung mit UML

2. Auflage

Springer Vieweg

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.

Bernhard Rumpe

Agile Modellierung mit UML

Codegenerierung, Testfälle, Refactoring

2. Auflage

Springer Vieweg

Prof. Dr. Bernhard Rumpe
Aachen, Deutschland

ISSN 1439-5428
ISBN 978-3-642-22429-4 e-ISBN 978-3-642-22430-0
DOI 10.1007/978-3-642-22430-0

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Springer Vieweg

Gedruckt auf säurefreiem und chlorfrei gebleichtem Papier

Springer Vieweg ist eine Marke von Springer DE.
Springer DE ist Teil der Fachverlagsgruppe Springer Science+Business Media
www.springer-vieweg.de

Vorwort

Vorwort zur 2ten Auflage

Da dieses das zweite Buch zur agilen Softwareentwicklung mit der UML ist, der geneigte Leser Buch 1 [Rum11] daher wahrscheinlich kennt und das Vorwort in [Rum11] für beide Bücher gilt, erlaube ich mir an dieser Stelle auf Buch 1 zu verweisen. Darin ist Folgendes dargelegt:

- Agile Methoden und modellbasierte Methoden sind beide erfolgreiche Vorgehensweisen.
- Aber eine Annäherung beider Vorgehensweisen hat bisher nicht stattgefunden.
- Auf Basis der Grundidee, Modelle statt Programmiersprachen zu verwenden besteht aber genau dazu die Möglichkeit.
- Das vorliegende Buch liefert dazu einen Beitrag in Form der UML/P.
- In der zweiten Auflage wurde die UML/P aktualisiert und auf die UML 2.3 sowie Java Version 6 angepasst.

Viel Spaß bei der Nutzung des Buchs beziehungsweise seiner Inhalte.

Bernhard Rumpe

Aachen im März 2012

Weiterführendes Material:

http://mbse.se-rwth.de

Vorwort zur 1ten Auflage

Softwaresysteme sind heutzutage in der Regel komplexe Produkte, für deren erfolgreiche Produktion der Einsatz ingenieurmäßiger Techniken unerlässlich ist. Diese nun mittlerweile mehr als 30 Jahre alte und häufig zitierte Erkenntnis hat dazu geführt, dass in den letzten drei Jahrzehnten innerhalb der Informatik im Gebiet des Software Engineering intensiv an Sprachen, Methoden und Werkzeugen zur Unterstützung des Softwareerstellungsprozesses gearbeitet wird. Trotz großer Fortschritte hierbei muss allerdings festgestellt werden, dass im Vergleich zu anderen, durchgängig viel älteren Ingenieursdisziplinen noch viele Fragen unbeantwortet sind und immer neue Fragestellungen auftauchen.

So macht auch ein oberflächlicher Vergleich zum Beispiel mit dem Gebiet des Bauwesens schnell deutlich, dass dort internationale Standards eingesetzt werden, um Modelle von Gebäuden zu erstellen, die Modelle zu analysieren und anschließend die Modelle in Bauten zu realisieren. Hierbei sind dann auch die Rollen- und Aufgabenverteilungen allgemein akzeptiert, so dass etwa Berufsgruppen wie Architekten, Statiker sowie Spezialisten für den Tief- und Hochbau existieren.

Eine derartige modellbasierte Vorgehensweise wird zunehmend auch in der Softwareentwicklung favorisiert. Dies bedeutet insbesondere, dass in den letzten Jahren international versucht wird, eine allgemein akzeptierte Modellierungssprache festzulegen, so dass etwa wie im Bauwesen, von einem Software-Architekten erstellte Modelle von einem „Software-Statiker" analysiert werden können, bevor sie von Spezialisten für die Realisierung, also Programmierern in ausführbare Programme umgesetzt werden.

Diese standardisierte Modellierungssprache ist die Unified Modeling Language, die in einem schrittweisen Prozess durch ein international besetztes Konsortium stetig weiterentwickelt wird. Aufgrund der vielfältigen Interessenlagen im Standardisierungsprozess ist mit der aktuellen Version 2.0 der UML eine Sprachfamilie entstanden, deren Umfang, semantische Fundierung und methodische Verwendung noch viele Fragen offen lässt.

Diesem Problem hat sich Herr Rumpe in den letzten Jahren in seinen wissenschaftlichen und praktischen Arbeiten gewidmet, deren Ergebnisse er nun in zwei Büchern einer breiten Leserschaft zugänglich macht. Hierbei hat Herr Rumpe das methodische Vorgehen in den Vordergrund gestellt. Im Einklang mit der heutigen Erkenntnis, dass leichtgewichtige, agile Entwicklungsprozesse insbesondere in kleineren und mittleren Entwicklungsprojekten große Vorteile bieten, hat Herr Rumpe Techniken für einen agilen Entwicklungsprozess entwickelt. Auf dieser Basis hat er dann eine geeignete Modellierungssprache definiert, in dem er ein so genanntes Sprachprofil für die UML definiert hat. In diesem Sprachprofil UML/P hat er die UML geeignet abgespeckt und an einigen Stellen so abgerundet, dass nun eine handhabbare Version der UML insbesondere für einen agilen Entwicklungsprozess vorliegt.

Herr Rumpe hat diese Sprache UML/P ausführlich in dem diesem Buch vorangehenden Buch „Modellierung mit UML“ erläutert. Das Buch bietet eine wesentliche Grundlage für das hier vorliegende Buch, deren Inhalt allerdings auch in diesem Buch noch einmal kurz zusammengefasst wird. Das hier vorliegende Buch mit dem Titel „Agile Modellierung mit UML“ widmet sich nun in erster Linie dem methodischen Umgang mit der UML/P.

Hierbei behandelt Herr Rumpe drei Kernthemen einer modellbasierten Softwareentwicklung. Dies sind

- die Codegenerierung, also der automatisierte Übergang vom Modell zu einem ausführbaren Programm,
- das systematische Testen von Programmen mithilfe einer modellbasierten, strukturierten Festlegung von Testfällen sowie
- die Weiterentwicklung von Modellen durch den Einsatz von Transformations- und Refactoring-Techniken.

Alle drei Kernthemen werden von Herrn Rumpe zunächst systematisch aufgearbeitet und die zugrunde liegenden Begriffe und Techniken werden eingeführt. Darauf aufbauend stellt er dann jeweils seinen Ansatz auf der Basis der Sprache UML/P vor. Diese Zweiteilung und klare Trennung zwischen Grundlagen und Anwendungen machen die Darstellung außerordentlich gut verständlich und bieten dem Leser auch die Möglichkeit, diese Erkenntnisse unmittelbar auf andere modellbasierte Ansätze und Sprachen zu übertragen.

Insgesamt hat dieses Buch einen großen Nutzen sowohl für den Praktiker in der Softwareentwicklung, für die akademische Ausbildung im Fachgebiet Softwaretechnik als auch für die Forschung im Bereich der modellbasierten Entwicklung der Software. Der Praktiker lernt, wie er mit modernen modellbasierten Techniken die Produktion von Code verbessern und damit die Qualität erheblich steigern kann. Dem Studierenden werden sowohl wichtige wissenschaftliche Grundlagen als auch unmittelbare Anwendungen der dargestellten grundlegenden Techniken vermittelt. Dem Wissenschaftler bietet das Buch einen umfassenden Überblick über den heutigen Stand der Forschung in den drei Kernthemen des Buchs.

Das Buch stellt somit einen wichtigen Meilenstein in der Entwicklung von Konzepten und Techniken für eine modellbasierte und ingenieurmäßige Softwareentwicklung dar und bietet somit auch die Grundlage für weitere Arbeiten in der Zukunft. So werden praktische Erfahrungen mit dem Umgang der Konzepte ihre Tragbarkeit validieren. Wissenschaftliche, konzeptionelle Arbeiten werden insbesondere das Thema der Modelltransformation etwa auf der Basis von Graphtransformationen genauer erforschen und darüber hinaus das Gebiet der Modellanalyse im Sinne einer Modellstatik vertiefen.

Ein derartiges vertieftes Verständnis der Informatik-Methoden im Bereich der modellbasierten Softwareentwicklung ist eine entscheidende Voraussetzung für eine erfolgreiche Kopplung mit anderen ingenieurmäßigen Methoden etwa im Bereich von eingebetteten Systemen oder im Bereich von intelligenten, benutzungsfreundlichen Produkten. Die Domänenunabhängigkeit der Sprache UML/P bietet auch hier noch viele Möglichkeiten.

Gregor Engels

Paderborn im September 2004

Inhaltsverzeichnis

1

Einführung

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

Christopher Darlington Morley

Viele Projekte zeigen mittlerweile eindrucksvoll, wie teuer falsche oder fehlerhafte Software werden kann.

Um die stetig wachsende Komplexität von Software-basierten Projekten und Produkten sowohl in den Bereichen betrieblicher oder administrativer Informations- und Websysteme als auch bei Cyber-Physischen Systemen wie Auto, Flugzeug, Produktionsanlagen, E-Health- und mobilen Systemen beherrschbar zu machen, wurde in den letzten Jahren ein wirkungsvolles Portfolio an Konzepten, Techniken und Methoden entwickelt, die die Softwaretechnik zu einer erwachsenen Ingenieursdisziplin heranreifen lassen.

Das Portfolio ist noch keineswegs ausgereift, muss aber insbesondere in dem derzeitigen industriellen Softwareentwicklungsprozess noch sehr viel mehr 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.

Weiterführendes Material:

http://mbse.se-rwth.de

B.Rumpe, *Agile Modellierung mit UML*, 2. Aufl., Xpert.press
DOI 10.1007/978-3-642-22430-0_1, © Springer-Verlag Berlin Heidelberg 2012

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.“ [Rum11], in dem das Sprachprofil UML/P detailliert erklärt ist. Es ist daher zu empfehlen, bei der Lektüre dieses Bands [Rum12] den ersten Band [Rum11] griffbereit zu halten, obwohl Teile von Band 1 in kompakter Form in Kapitel 3 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-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 Techniken 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.

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 Klassendiagramme
führt Form und Verwendung von Klassendiagrammen ein.

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

4 Objektdiagramme
diskutiert Sprache und methodischen Einsatz der Objektdiagramme so-

wie deren Integration mit der OCL-Logik, um damit eine „Logik der Diagramme“ zu ermöglichen, in der unerwünschte Situationen, Alternativen und Kombinationen beschrieben werden können.

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

6 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.

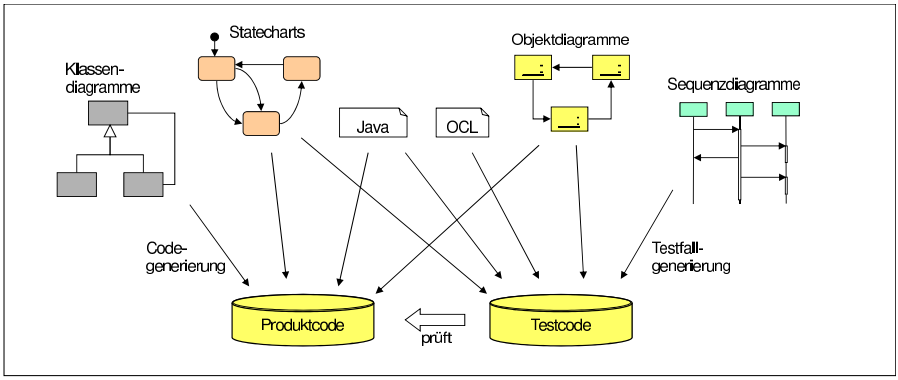

Abbildung 1.1. Notationen der UML/P

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 dar-

gestellt 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 bezeichnete 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 hier nicht behandelt 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	*Kapitel 2 (Band 1)*	*Kapitel 3 (Band 1)*	*Kapitel 4 (Band 1)*	*Kapitel 5 (Band 1)*	*Kapitel 6 (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 4	Kapitel 5.1	Kapitel 5.3	Kapitel 5.2	Kapitel 5.4	Kapitel 5.5
Testfallmodellierung	Kapitel 6	(Kapitel 8)	Abschnitte 7.2 & 7.3	Abschnitt 7.1	Abschnitt 7.5	Abschnitt 7.4
Refactoring	Kapitel 9	Abschnitte 10.1 & 10.2	Abschnitt 10.1	(Abschnitt 10.2)	*Abschnitt 5.6 (Band 1)*	(Abschnitt 10.2)

Abbildung 1.2. Übersicht über den Inhalt beider Bände

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

Kapitel 2 skizziert eine agile UML-basierte Vorgehensweise. Sie nutzt die UML/P als primäre Entwicklungssprache, um ausführbare Modelle zu erstellen, daraus Code zu generieren, Testfälle zu entwerfen und Architekturanpassungen (Refactorings) zu planen.

Kapitel 3 gibt eine kurze und kompakte Zusammenfassung von Band 1, [Rum11]. Dabei wird vor allem das dort eingeführte Sprachprofil der UML/P sehr kompakt und daher unvollständig vorgestellt.

Die Kapitel 4 und 5 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 generiert werden kann. Dabei werden sowohl Alternativen diskutiert als auch die Kombination von Transformationen demonstriert.

Die Kapitel 6 und 7 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 8 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 9 und 10 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 Na-

me des Diagramms und Diagramm-Eigenschaften in Form von Stereotypen beigefügt werden können. Vereinzelt kommen Spezialformen von Marken zum Einsatz, die weitgehend selbsterklärend sind.

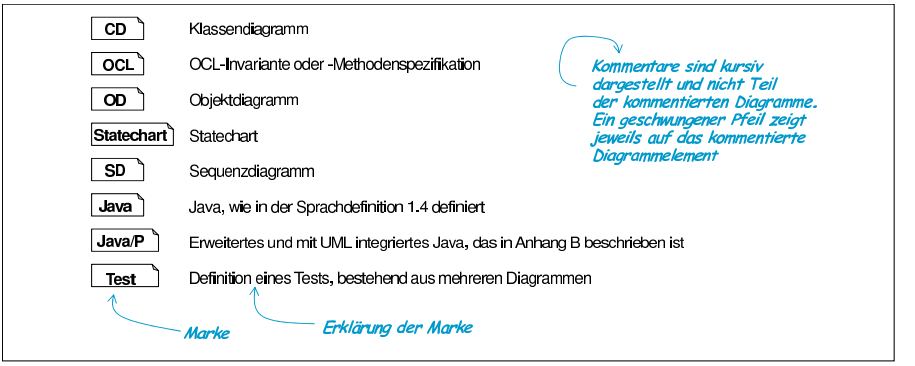

Abbildung 1.3. Marken für Diagramme und Textteile

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`}.

2

Agile und UML-basierte Methodik

Die wertvollsten Einsichten sind die Methoden.
Friedrich Wilhelm Nietzsche

Der zielgerichtete, in eine Methodik eingebettete Einsatz ist für die erfolgreiche Verwendung einer Modellierungssprache unverzichtbar. In diesem Kapitel werden Charakteristika agiler Methoden, insbesondere des *Extreme Programming (XP)* Prozesses [Bec04, Rum01] herausgearbeitet. Gemeinsam mit weiteren Elementen wird daraus ein Vorschlag für eine agile, auf der UML basierende Methodik eingeführt.

B.Rumpe, *Agile Modellierung mit UML*, 2. Aufl., Xpert.press
DOI 10.1007/978-3-642-22430-0_2, © Springer-Verlag Berlin Heidelberg 2012

Die Verbesserungen in der Softwaretechnik sind seit Jahren verantwortlich für kontinuierliche Effizienzsteigerungen in Softwareentwicklungsprojekten. Software mit hoher Qualität ist mit immer weniger personellen Ressourcen in immer kürzeren Zeiten zu erstellen. Prozesse wie der Rational Unified Process (RUP) [Kru03] oder das V-Modell XT [HH08] sind eher für große Projekte mit vielen Teammitgliedern geeignet. Sie sind aber speziell für kleinere Projektgrößen überfrachtet mit Tätigkeiten, die für das Endergebnis nicht unbedingt essentiell sind, und deshalb auch zu schwerfällig, um auf die sich schnell ändernde Umgebung (Technik, Anforderungen, Konkurrenz) eines Softwareentwicklungsprojekts reagieren zu können.

Flexibilität, schnelles Feedback, Konzentration auf die wesentlichen Arbeitsergebnisse und die Fokussierung auf die im Projekt beteiligten Personen und insbesondere Kunden sind wesentliche Charakteristika der neu entstandenden Prozesse, denen unter dem Begriff der „Agilen Prozesse“ ein gemeinsames Label gegeben wurde.

Der Standish Report [Gro09b] beschreibt, dass wesentliche Ursachen des Scheiterns von Projekten im schlechten Projektmanagement zu suchen sind: Es wird nicht adäquat kommuniziert, zu viel, zu wenig oder das Falsche dokumentiert, Risiken nicht rechtzeitig entgegengesteuert oder zu spät Rückkopplung von den Anwendern eingefordert.

Speziell die menschliche Komponente, also die projektinterne Kommunikation und Zusammenarbeit zwischen Entwicklern untereinander und der Entwickler mit den Anwendern, wird als wesentliche Ursache für das Scheitern von Softwareentwicklungsprojekten erkannt. Auch laut [Coc06] scheitern Projekte selten aus technischen Gründen. Das spricht einerseits für die gute Beherrschung auch von innovativen Techniken, ist andererseits aber zumindest teilweise zu hinterfragen. Denn wenn die Technik Schwierigkeiten bereitet, dann verursachen diese Probleme oft zwischenmenschliche Kommunikationsstörungen, die dann im weiteren Projektverlauf aus emotionalen Gründen in den Vordergrund rücken und letztlich als „gefühlte“ Gründe für das Scheitern des Projekts in Erinnerung bleiben.

Die richtige Technik ist in Form von Sprachen und Werkzeugen die wesentliche Grundlage für die „Leichtgewichtigkeit“ der neuen Prozessgeneration. Je kompakter die Sprache und je besser die Analyse- und Generierungswerkzeuge sind, um so effizienter sind die Entwickler und um so weniger Zusatzaufwand für Management oder Dokumentationsleistungen fällt an.

Softwaretechnik bietet mittlerweile ein großes *Portfolio* an Vorgehensweisen, Prinzipien, Entwicklungspraktiken, Werkzeugen und Notationen, die zur Entwicklung von Softwaresystemen unterschiedlichster Form, Größe und Qualität verwendet werden können. Diese Elemente des Portfolios ergänzen sich teilweise, können aber auch alternativ eingesetzt werden, so dass in einem Projekt eine Bandbreite an Auswahlmöglichkeiten zur Verfügung steht, die das Management, die Kontrolle und die Durchführung des Projekts betrifft.

Dieses Kapitel untersucht zunächst den aktuellen Stand des Portfolios der Softwaretechnik. In Abschnitt 2.2 wird „Extreme Programming“ (XP) diskutiert und in Abschnitt 2.3 drei essentielle Praktiken daraus vorgestellt. Abschnitt 2.4 beinhaltet einen Vorschlag für eine einfache Methode, die sich als Referenz für die detailliert diskutierten Notationen der UML/P und Techniken dieses Buchs eignet.

In Kapitel 3 folgt eine kompakte Übersicht über das für die vorgeschlagene Methode verwendbare Sprachprofil UML/P der Unified Modeling Language, die die wesentlichsten Eigenschaften des Sprachprofils erläutert und in [Sch12] mit einem geeigneten Werkzeugt unterstützt wird. Das Sprachprofil UML/P ist wie die UML selbst weitgehend *methodenunabhängig*. Das bedeutet, es ist möglich und sinnvoll, UML/P als Notation in Kombination mit anderen Methoden zu verwenden. Allerdings ist durch den Fokus auf Generierbarkeit von Code und Tests die UML/P besonders gut geeignet für den Einsatz in agilen Methoden.

2.1 Das Portfolio der Softwaretechnik

In [AMB+04, BDA+99] wurde ein Anlauf unternommen, das mittlerweile seit über 40 Jahren gewachsene Wissen der Softwaretechnik in einem „Software Engineering Body of Knowledge“ (SWEBOK) zusammenzufassen. Dabei werden Begriffsbildungen vereinheitlicht, die wesentlichen Kernelemente der Softwaretechnik als Ingenieursdisziplin dargestellt und insbesondere versucht, einen allgemein akzeptierten Konsens über die Inhalte und Konzepte der Softwaretechnik herzustellen.

Ein für unsere Überlegungen wesentlicher Teil der für Softwareentwicklungsprozesse verwendeten Terminologie ist in Abbildung 2.1 dargestellt.

Aus der in den letzten Jahren gesammelten Erfahrung bei der Durchführung von Softwareentwicklungsprojekten kristallisiert sich heraus, dass es *den* Prozess für Softwareentwicklung nicht geben kann. Es scheint heute noch nicht einmal möglich, ein einigermaßen aussagekräftiges Prozessframework angeben zu können, das für die nach Wichtigkeit, Größe, Anwendungsbereich und Projektumfeld sehr unterschiedlichen Softwareentwicklungsprojekte Allgemeingültigkeit besitzt. Stattdessen ist man dabei, eine Sammlung an Konzepten, Best Practices und Werkzeugen aufzustellen, die es erlaubt, projektspezifischen Erfordernissen in einem individuellen Prozess Rechnung zu tragen. Dabei werden Detaillierungsgrade und Präzision der Dokumente, Meilensteine und die abzuliefernden Ergebnisse in Abhängigkeit von der Größe des Projekts und der gewünschten Qualität des Ergebnisses festgelegt. Vorhandene, als Muster zu verstehende Prozessbeschreibungen können dabei Hilfestellung geben. Jedoch werden projektspezifische Anpassungen als grundsätzlich notwendig erachtet. Für Projektbeteiligte ist

Vorgehensmethode. Eine Vorgehensmethode (syn. *Vorgehensmodell*) beschreibt das Vorgehen „für die Abwicklung der Softwareerstellung in einem Projekt“ [Pae00]. Es kann zwischen technischem, sozialem und organisatorischem Vorgehensmodell unterschieden werden.

Softwareentwicklungsprozess. Der Begriff Softwareentwicklungsprozess wird gelegentlich als Synonym zu „Vorgehensmethode“ verwendet, oft aber als detaillierte Form verstanden. So definiert [Som10] einen Prozess als Menge von Aktivitäten und Ergebnissen, mit denen ein Softwareprodukt erstellt wird. Meist werden auch die zeitliche Reihenfolge oder die Abhängigkeiten der Aktivitäten herausgestellt.

Entwicklungsaufgabe. Ein Prozess wird in eine Reihe von Entwicklungsaufgaben unterteilt, die jeweils bestimmte Ergebnisse in Form von *Artefakten* erbringen. Die Projektbeteiligten führen *Aktivitäten* aus, um diese Aufgaben zu erledigen.

Prinzip. „Prinzipien sind Grundsätze, die man seinem Handeln zugrunde legt. Prinzipien sind allgemein gültig, abstrakt, allgemeinster Art. Sie bilden eine theoretische Grundlage. Prinzipien werden aus der Erfahrung und der Erkenntnis hergeleitet [...].“ [Bal00]

Best Practices beschreiben erfolgreich erprobte *Entwicklungspraktiken* in Entwicklungsprozessen. Eine Entwicklungspraktik kann als konkretes, operationalisiertes Prozessmuster aufgefasst werden, das ein allgemeines Prinzip umsetzt (vgl. RUP [Kru03] oder XP [Bec04]).

Artefakt. Entwicklungsergebnisse werden in einer konkreten Notation dargestellt. Dafür werden zum Beispiel die natürliche Sprache, UML oder eine Programmiersprache verwendet. Die Dokumente dieser Sprachen werden *Artefakte* genannt. Dies sind beispielsweise Anforderungsanalysen, Modelle, Code, Reviewergebnisse oder ein Glossar. Ein Artefakt kann hierarchisch gegliedert sein.

Transformation. Als Transformation kann die Entwicklung eines neuen Artefakts oder einer verbesserten Version eines gegebenen Artefakts verstanden werden, die automatisiert oder manuell erfolgen kann. Letztendlich können fast alle Aktivitäten als Transformationen der im Projekt gegebenen Menge von Artefakten verstanden werden.

Abbildung 2.1. Begriffsdefinitionen im Softwareentwicklungsprozess

es daher sinnvoll, möglichst viele Vorgehensweisen aus dem derzeit vorhandenen Portfolio zu kennen.

War in den 90’er Jahren ein starker Trend hin zu vollständigen und dadurch eher bürokratischen Softwareentwicklungsprozessen zu beobachten, so haben sich die agilen Methoden in den 2000’er Jahren von diesem Trend abgekoppelt. Ermöglicht haben diese Umkehr das deutlich gewachsene Verständnis der Aufgaben bei der Entwicklung komplexer Softwaresysteme sowie die Verfügbarkeit verbesserter Programmiersprachen, Compiler und einer Reihe von weiteren Entwicklungswerkzeugen. So ist es heute nahezu genauso effizient eine GUI direkt zu implementieren, wie sie zu spezifizieren. Entsprechend kann die Spezifikation durch einen für den Anwender ausprobierbaren und in der Realisierung weiterverwendbaren Prototyp ersetzt werden. Der Trend zur Reduzierbarkeit der notwendigen

Entwicklerkapazitäten wird dadurch verstärkt, dass bei weniger Projektbeteiligten auch weniger organisatorischer Overhead notwendig ist und damit die Personalaufwände weiter reduziert werden können.

Auch die mit agilen Methoden verstärkte Betonung der individuellen Fähigkeiten und Bedürfnisse der am Projekt beteiligten Entwickler und Kunden erlaubt die weitere Reduktion von Projektbürokratie zugunsten verstärkter Eigenverantwortung. Diese Teamorientierung kann auch in anderen Bereichen des Wirtschaftslebens, wie zum Beispiel bei flachen Management-Hierarchien, beobachtet werden und basiert auf der Annahme, dass mündige und motivierte Projektbeteiligte von sich aus verantwortungsvoll und couragiert handeln werden, wenn durch das Projektumfeld die Möglichkeiten dafür geschaffen werden.

2.2 Extreme Programming (XP)

Extreme Programming (in Kurzform: XP) ist eine „agile“ Softwareentwicklungsmethode, deren wesentlicher Kern in [Bec04] beschrieben ist. Obwohl XP als Vorgehensweise unter anderem in Softwareentwicklungsprojekten einer Schweizer Bank definiert und verfeinert wurde, zeigt schon die Namensgebung eine starke Beeinflussung durch die nordamerikanische, von Pragmatik geprägte Softwareentwicklungs-Kultur. Trotz des gewählten Namens ist XP allerdings keine Hacker-Technik, sondern besitzt einige sehr detailliert ausgearbeitete und rigoros zu verwendende methodische Aspekte. Diese erlauben es ihren Befürwortern zu postulieren, dass durch XP mit verhältnismäßig wenig Aufwand qualitativ hochwertige Software unter Einhaltung der Budgets zur Zufriedenstellung der Kunden erstellt werden kann. Statistisch aussagekräftige, über Anekdoten hinausgehende Untersuchungen von XP-Projekten gibt es mittlerweile einige [DD08, RS02, RS01].

XP erfreut sich in der Praxis einer hohen Popularität. Mittlerweile sind viele von Bücher über XP erschienen, von denen auch die ersten [Bec04, JAH00, BF00, LRW02] unterschiedliche Aspekte der Thematik sowie jeweils aktuelle Wissensstände der sich noch in Weiterentwicklung befindlichen Methodik detailliert betrachten oder Fallbeispiele durchgeführter Projekte illustrieren [NM01]. [Wak02, AM01] beinhalten vor allem praktische Hilfestellungen zur Umsetzung von XP, [Woy08] betrachtet kulturelle Aspekte und [BF00] diskutiert die Planung in XP-Projekten.

Eine kritische Beschreibung mit expliziter Diskussion von Schwachstellen von XP bieten [EH00a] und [EH01]. Darin wird unter anderem der fehlende Einsatz von Modellierungstechniken wie etwa der UML bemängelt und ein kritischer Vergleich zu Catalysis [DW98] gezogen. Eine dialektische Diskussion der Vor- und Nachteile von Extreme Programming beinhaltet [KW02]. Darin werden unter anderem die Notwendigkeit zum disziplinierten Vorgehen, die starken und gegenüber klassischen Ansätzen deutlich

geänderten Anforderungen insbesondere an den Teamleiter und den Coach hervorgehoben.

Nachfolgend werden wesentliche Elemente von XP gemäß den Einführungen aus [Bec04, Rum01] dargestellt und diskutiert. Weiterführende Themen in der Literatur behandeln mittlerweile XP parallel mit anderen agilen Methoden [Han10, Leh07, Ste10, HRS09] und erhalten so ein Portfolio agiler Techniken, oder sie adaptieren agile Methoden für verteilte Teams [Eck09] oder behandeln die Migration von Unternehmen hin zu agilen Methoden [Eck11].

Übersicht über XP

XP ist eine leichtgewichtige Softwareentwicklungsmethode. Darin wird auf eine Reihe von Elementen der klassischen Softwareentwicklung verzichtet, um so eine schnellere und effizientere Codierung zu erlauben. Die dabei für die Qualitätssicherung möglicherweise entstehenden Defizite werden durch stärkere Gewichtung anderer Konzepte (insbesondere der Testverfahren) kompensiert. XP besteht aus einer Reihe von Konzepten, von denen im Rahmen dieses Überblicks nur die wesentlichsten behandelt werden.

XP versucht explizit nicht auf neue oder nur wenig erprobte methodische Konzepte zu setzen, sondern integriert bewährte Techniken zu einem Vorgehensmodell, das auf Wesentliches fokussiert und auf organisatorischen Ballast soweit wie möglich verzichtet. Weil der Programmcode das ultimative Ziel einer Softwareentwicklung ist, fokussiert XP von Anfang an genau auf diesen Code. Alles an zusätzlicher Dokumentation wird als zu vermeidender Ballast betrachtet. Eine Dokumentation ist aufwändig zu erstellen und oft sehr viel fehlerhafter als der Code, weil sie normalerweise nicht ausreichend automatisiert analysierbar und testbar ist. Zusätzlich reduziert sie die Flexibilität bei der Weiterentwicklung und Anpassung des Systems als schnelle Reaktion auf neue oder veränderte Anforderungen der Kunden, wie es in der Praxis häufig der Fall ist. Folgerichtig wird in XP-Projekten (außer dem Code und den Tests) nahezu keine Dokumentation erstellt. Zum Ausgleich wird dafür Wert auf eine gute Kommentierung des Quellcodes durch Codierungsstandards und eine umfangreiche Testsammlung gelegt.

Die primären Ziele von XP sind die effiziente Entwicklung qualitativ hochwertiger Software unter Einhaltung von Zeit- und Kostenbudgets. Welche Mittel dazu eingesetzt werden, wird anhand der *Werte*, der *Prinzipien*, der grundlegenden *Aktivitäten* und der darin umgesetzten *Entwicklungspraktiken* in Pyramide in Abbildung 2.2 veranschaulicht.

Erfolgsfaktoren von XP

Nach mittlerweile einigen Jahren des Einsatzes von XP kristallisieren sich einige der für den Erfolg von XP-Projekten wesentlichen Faktoren heraus:

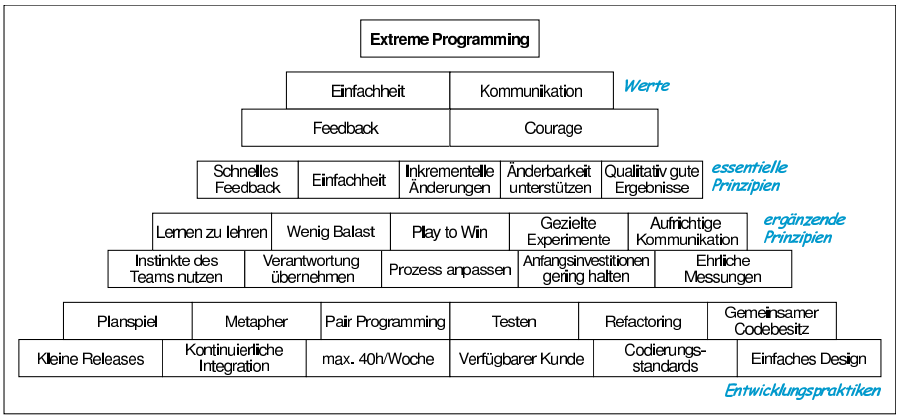

Abbildung 2.2. Aufbau des Extreme Programming

- Das Team ist motiviert und das Arbeitsumfeld für XP geeignet. Das bedeutet zum Beispiel, dass Arbeitsplätze für Pair Programming eingerichtet sind und die Entwickler in räumlicher Nähe untereinander und zum Kunden arbeiten.
- Der Kunde arbeitet in seiner Rolle aktiv am Projekt mit und steht für Fragen zur Verfügung. Die Studie [RS02] hat gezeigt, dass dies als einer der kritischen Erfolgsfaktoren zu werten ist.
- Die Wichtigkeit der Tests auf allen Ebenen erschließt sich, sobald Änderungen stattfinden, neue Entwickler hinzukommen oder das System eine gewisse Größe erhält und damit manuelle Tests nicht mehr durchführbar sind.
- Der in allen Bereichen diskutierte Zwang nach Einfachheit führt zum Weglassen von Dokumentation genauso wie zu einem möglichst einfachen Design und erlaubt daher eine signifikante Reduktion der Arbeitslast.
- Das Fehlen eines Pflichtenhefts und die Anwesenheit eines Kunden, mit dem auch während des Projekts über Funktionalität verhandelt werden kann, führt zu einer intensiveren Einbindung des Kunden in den Projektverlauf. Dies hat zwei Auswirkungen: Zum einen erlaubt es eine schnelle Reaktion auf sich verändernde Kundenwünsche. Zum anderen können so auch die Kundenwünsche durch das Projekt beeinflusst werden. Der Projekterfolg wird dadurch auch eine soziale Übereinkunft zwischen Kunden und Entwicklerteam und nicht nur eine objektiv überprüfte, auf Dokumenten basierende Zielerreichung.

Gerade der letzte Aspekt entspricht der XP-Philosophie, weniger zu kontrollieren und stattdessen mehr Eigenverantwortung und Engagement zu fordern. So könnte ein für alle Projektbeteiligten zufriedenstellenderes Ergebnis

entstehen, als es mit festgeschriebenen Pflichtenheften in einer Welt der sich wandelnden Anforderungen möglich ist.

Grenzen der Anwendbarkeit von XP

XP ist in Bezug auf die Projektdokumentation und auf die Einbindung von Kunden eine durchaus revolutionäre Vorgehensweise. Entsprechend gibt es eine Reihe von Einschränkungen und Anforderungen an Projektgröße und Projektumfeld, die für XP gelten. Diese werden unter anderem in [Bec04, TFR02, Boe02] erörtert. XP ist vor allem für Projekte bis zu zehn Personen gut geeignet [Bec04], aber es ist eine offensichtliche Problematik, XP für große Projekte zu skalieren, wie dies zum Beispiel in [JR01] diskutiert wird. XP ist eben nur eine weitere Vorgehensweise im Portfolio der Softwaretechnik, die wie viele andere Techniken auch nur unter den gegebenen Prämissen eingesetzt werden kann.

Die Grundannahmen, Techniken und Konzepte von XP führen zu einer relativ starken Polarisierung der Meinungen. Auf der einen Seite glauben Programmierer manchmal, dass XP das Hacking zur Vorgehensweise erhebt. Auf der anderen Seite wird XP nicht ganz ernst genommen, weil es vieles irgnoriert, was in den letzten Jahrzehnten an Entwicklungsprozessen erarbeitet worden ist. Tatsächlich ist beides nur sehr bedingt richtig. Zum einen können Hacker tatsächlich leichter zu einer XP-artigen Vorgehensweise als zu einem Vorgehen nach dem RUP motiviert werden. Zum anderen erkennen viele Softwareentwickler bei genauerer Betrachtung bereits bisher gelebte Entwicklungspraktiken wieder. Außerdem ist XP in seiner Vorgehensweise sehr rigoros und erfordert Disziplin in der Umsetzung.

Sicherlich richtig ist, dass XP eine leichtgewichtige Softwareentwicklungsmethode ist, die explizit als Gegengewicht zu schwergewichtigen Methoden wie dem RUP [Kru03] oder dem V-Modell XT [HH08] positioniert ist. Wesentliche Unterschiede stellen die Konzentration alleine auf den Code als Ergebnis und die Einbeziehung der Bedürfnisse der Projektbeteiligten dar. Der interessanteste Unterschied ist aber die gesteigerte Fähigkeit von XP auf Änderungen des Projektumfelds oder der Anforderungen der Anwender flexibel zu reagieren. Aus diesem Grund gehört XP zur Gruppe der „agilen Methoden“.

Die Fehlerbehebungskosten in XP-Projekten

Eine der grundlegenden Annahmen in XP stellt wesentliche Erkenntnisse der Softwaretechnik infrage. Während man bisher davon ausgegangen ist, dass die Kosten zur Behebung von Fehlern oder zur Durchführung von Änderungen wie in [Boe81] beschrieben exponentiell mit der Zeit steigen, geht XP davon aus, dass diese im Projektverlauf abflachen. Abbildung 2.3 zeigt diese beiden Kostenkurven schematisch.