

B. Oestereich (Hrsg.),
P. Hruschka, N. Josuttis, H. Kocher,
H. Krasemann, M. Reinhold

Erfolgreich mit Objekt- orientierung

Vorgehensmodelle und
Managementpraktiken
für die objektorientierte
Softwareentwicklung

1. Auflage



Oldenbourg

www.system-bauhaus.com/vm-buch



B. Oestereich (Hrsg.)
P. Hruschka/ N. Josuttis/H. Kocher/
H. Krasemann/M. Reinhold

Erfolgreich mit Objektorientierung

Vorgehensmodelle und Managementpraktiken
für die objektorientierte Softwareentwicklung

Fragen, die sich Projektleiter täglich stellen:

- Wie geht man bei iterativ-inkrementellen Projekten in der Praxis vor?
- Wie plant und steuert man professionell Projekte, Phasen, Iterationen, Meilensteine?
- Wie führt und organisiert man Projekte und Teams?
- Wie schätzt und mißt man Aufwände, z. B. mit dem neuen Widget-Point-Verfahren?
- Wo finde ich ein praktisches Beispiel für ein Vorgehensmodell mit ganz konkreten Anleitungen?

Die Autoren dieses Buches zeigen es. Sie haben als Berater und Projektleiter die unterschiedlichsten objektorientierten Projekte begleitet.

Sie vergleichen die Grundkonzepte des deutschen V-Modells '97 und die „Amigo“-Vorgehensmodelle von der OMT bis zum Unified Process.

Sie sprechen typische Probleme und Widrigkeiten bei der Durchführung von Projekten an und zeigen Ihnen Wege, diese erfolgreich zu meistern.

Da viele objektorientierte Projekte weniger an technischen Problemen als an der systematischen Vorgehensweise oder den richtigen Managementtechniken kranken, trägt dieses Buch dazu bei, daß noch mehr Projekte **erfolgreich mit Objektorientierung** durchgeführt werden können.

ISBN 3-486-25277-1

**Nur eine Idee hat die Kraft,
sich so weit zu verbreiten.**

Vorwort

Ludwig Mies van der Rohe

Einige der Autoren dieses Buches hatten einmal die Idee, alleine ein Buch zu Vorgehensmodellen und Managementpraktiken zu schreiben. Alle sind sie als Berater und Projektleiter in objektorientierten Projekten tätig, haben langjährige Erfahrung mit der Materie, aber - vielleicht gerade deswegen - leider kaum die Zeit, ein Buch zu schreiben. Zumindest nicht ein ganzes - und so kam es zu der Idee einer gemeinsamen Arbeit.

Wir bieten Ihnen mit diesem Buch einen substantiellen Einblick in unsere Erfahrungen und Erkenntnisse, die wir in den vergangenen Jahren mit objektorientierten Projekten gesammelt haben.

Jedes Kapitel beleuchtet das Thema aus einer anderen Perspektive und hat andere Hauptautoren. Trotz grundsätzlich übereinstimmender Ansichten und Erfahrungen hebt jeder Autor ganz bewußt jeweils seine eigene Sichtweise und seine speziellen Erfahrungen in diesem Buch hervor.

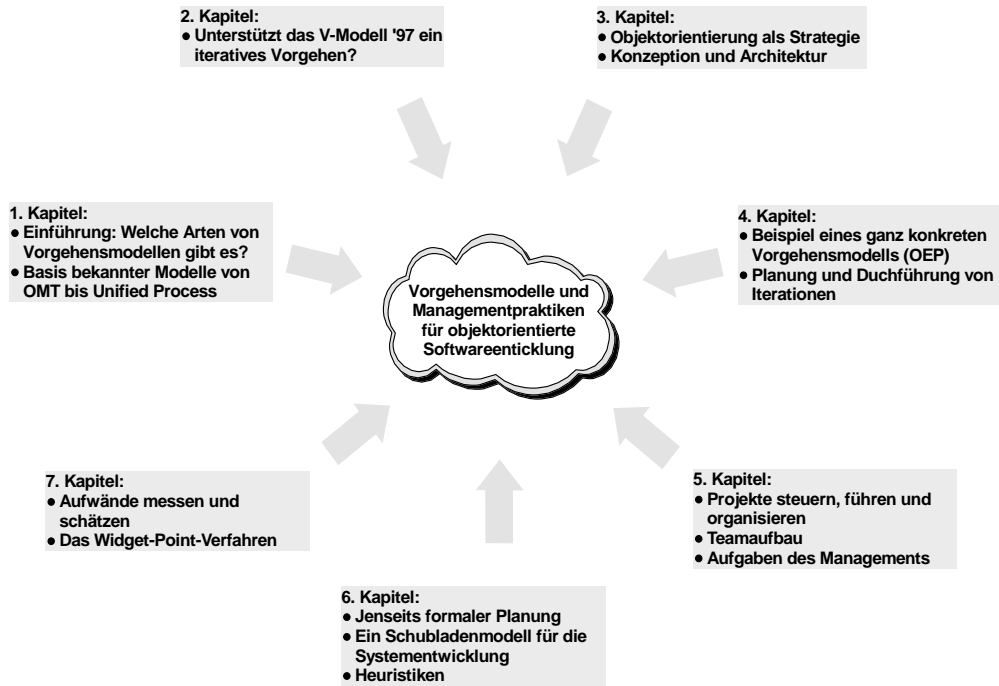
Die Objektorientierung hat mittlerweile einen festen Platz in der Softwareentwicklung und ist zu einer Basistechnologie geworden. In vielen Unternehmen hat sich die objektorientierte Softwareentwicklung jedoch noch nicht etabliert. Es laufen dort vielleicht gerade die ersten Prototypen, Pilotprojekte oder Migrationen, und der Erfolg steht noch aus. Die Autoren haben im Laufe ihrer Arbeit auch einige sehr problembeladene und schwierige Projekte gesehen. Von einigen dieser Projekte hängt das Ansehen der Objektorientierung innerhalb eines Unternehmens oder gar des Unternehmens in der Branche ab.

Wir kennen auch viele erfolgreiche Projekte; sie bestärken uns und die Idee der Objektorientierung. Da viele, gerade auch die großen Projekte aber weniger an technischen Problemen, denn an einer systematischen Vorgehensweise oder den richtigen Managementtechniken kranken, glauben wir, mit diesem Buch dazu beitragen zu können, daß noch mehr Projekte **erfolgreich mit Objektorientierung** durchgeführt werden können.

Auf jeden Fall hoffen wir, daß wir speziell Ihnen wichtige Anregungen und Hinweise geben können. Über Rückmeldungen, Kritik und Anregungen zu diesem Buch freuen wir uns auch, schreiben Sie an *vm-buch@system-bauhaus.de*.

Bernd Oestereich

Gliederung dieses Buches



Inhaltsverzeichnis

1. Kapitel: Einführung	13
1.1. Einleitung.....	15
1.2. Vorgehensmodelle	16
1.2.1. Vorgehensmodell-Varianten.....	17
1.2.2. Welcher Prozeß für welches Projekt?	18
1.3. Vorgehensmodelle und UML	27
1.3.1. OMT (Rumbaugh)	28
1.3.2. Booch-Methode.....	28
1.3.3. OOSE (Jacobson)	29
1.4. Der Unified Software Development Process	30
1.4.1. Phasen	31
1.4.2. Iterationen	32
1.4.3. Prozesse und Aktivitäten.....	33
1.4.4. Produkte.....	35
1.4.5. Rollen	35
2. Kapitel: V-Modell '97	37
2.1. Grundstruktur	39
2.1.1. Der objektorientierte Lebenszyklus im V-Modell '97	41
2.1.2. Evolutionär versus iterativ-inkrementell	42
2.1.3. Die Kernaktivitäten und -Produkte der Systemerstellung	46
2.1.4. Anpaßbarkeit an Randbedingungen	48
2.1.5. Vorgehen versus Dokumentenstruktur	49
2.2. Unterstützte Bereiche des V-Modells '97.....	51
2.3. Zusammenhang V-Modell und Phasenmodell	53
2.4. Zusammenfassung	54
3. Kapitel: Architektur und Konzeption	55
3.1. Strategien für erfolgreiche Projekte.....	57
3.2. Motivation	58
3.3. Highlevel-Studie	62
3.3.1. Highlevel-Analyse	62
3.3.2. Die Kunst der Abstraktion	63

3.3.3.	Highlevel-Design	64
3.3.4.	Risiken und Prioritäten	66
3.3.5.	Ergebnis der Highlevel-Studie	67
3.3.6.	Dauer der Highlevel-Studie	68
3.4.	Realisierung.....	68
3.4.1.	Realisierung des ersten Subsystems.....	68
3.4.2.	Ergebnisse der ersten Realisierung.....	69
3.4.3.	Sofortige Integration.....	70
3.5.	Wiederverwendung	71
3.5.1.	Mythos Wiederverwendung	71
3.5.2.	Wiederverwendung von Fremdsoftware	74
3.5.3.	Design von Wiederverwendung	74
3.5.4.	Refaktorisierung.....	76
3.6.	Prototypen	77
3.6.1.	Der Durchstich	77
3.6.2.	Die Bedienoberfläche.....	78
3.6.3.	Einbindung von Prototypen in das konzeptionelle Vorgehen	80
3.7.	Inkrementelle Entwicklung.....	81
3.8.	Zeitvorgaben.....	84
3.9.	Das Team	85
3.10.	Gesamtbetrachtung des konzeptionellen Vorgehens	87
3.10.1.	Zusammenfassende Richtlinien.....	88

4. Kapitel: Planung und Prozeßmanagement objektorientierter Projekte.....	91	
4.1. Der Object Engineering Process (OEP)	93	
4.1.1.	Wege zu einem Prozeßleitfaden	93
4.1.2.	Struktur des Vorgehensmodells	95
4.1.3.	Struktur des Prozeßleitfaden	96
4.1.4.	Angenommene Rahmenbedingungen	99
4.1.5.	Anwendung und Einführung des Prozesses	103
4.1.6.	Organisation der Projektleitung.....	104
4.2. Prozeß-Gliederung im Überblick	106	
4.2.1.	Anforderungsanalyse	107
4.2.2.	Systemerstellung.....	108
4.2.3.	Systemerstellung.Fachliche Architektur.....	108
4.2.4.	Systemerstellung.Umfeld und Batch.....	108
4.2.5.	Systemerstellung.Projektextern	109
4.2.6.	Systemerstellung.Subsysteme.....	109
4.2.7.	Qualitätssicherung, Test, Abnahmen.....	110
4.2.8.	Einsatz und Verteilung	111
4.2.9.	Konfigurations- und Umgebungsmanagement	112

4.2.10. Projektmanagement	112
4.2.11. Iterations-, Prozeß- und Änderungsmanagement.....	113
4.3. Entwicklungsphasen und Meilensteine im Überblick	113
4.3.1. Vorbereitungsphase	115
4.3.2. Entwurfsphase	117
4.3.3. Konstruktionsphase.....	118
4.3.4. Einführungsphase	119
4.3.5. Betriebsphase	119
4.4. Iterationen planen, steuern und durchführen	120
4.4.1. Meilensteine	120
4.4.2. Planung	121
4.4.3. Iterations-Planungs-Matrix	124
4.4.4. Durchführung	128
4.4.5. Wann ist eine Iteration zu Ende?	131
4.4.6. Timeboxing.....	132
4.4.7. Besondere Projekt Ereignisse und -ausnahmen	133
4.5. Konkrete Aktivitäten im Überblick	134
4.5.1. Aktivitäten der Vorbereitungsphase	136
4.5.2. Aktivitäten der Entwurfssphase.....	138
4.5.3. Aktivitäten der Konstruktionsphase.....	143
4.5.4. Aktivitäten der Einführungsphase	148
4.6. Exemplarische Aktivitätsbeschreibungen.....	150
4.7. Exemplarische Ergebnistypen.....	167
5. Kapitel: Steuerung objektorientierter Projekte.....	171
5.1. Objektorientierung umfaßt viele Faktoren.....	173
5.2. Auswahl des Vorgehensmodells.....	175
5.2.1. Rolle der Entwurfsmethode bei der Software-Entwicklung	175
5.2.2. Einführung eines Entwicklungsprozesses.....	178
5.3. Projektdurchführung	179
5.3.1. Risikomanagement	179
5.3.2. Planung und Überwachung.....	187
5.3.3. Anpassung des Entwicklungsprozesses	192
5.3.4. Projekte in der Krise	195
5.4. Teamaufbau	200
5.4.1. Teamstruktur bei objektorientierten Projekten	201
5.4.2. Kreative Teams	205
5.4.3. Feature Teams.....	206
5.4.4. Tiger Teams	206
5.4.5. Motivation von Entwicklern.....	207

5.5. Dokumentation	211
5.5.1. Anwenderdokumentation	212
5.5.2. Managementdokumentation	213
5.5.3. Entwicklungsdokumentation	215
5.6. Werkzeuge	218
5.7. Aufgaben des Managements	221
5.7.1. Planung und Überwachung.....	222
5.7.2. Führung und Steuerung	223
5.7.3. Teammanagement.....	224
5.7.4. Durchführung des Projekts	225
5.8. Zusammenfassung	226
6. Kapitel: Heuristiken zur Systementwicklung	229
6.1. Heuristiken - micro-codierte Handlungsanweisungen.....	231
6.2. DAS Vorgehensmodell gibt es nicht!	231
6.3. Das Schubladenmodell der Systementwicklung	232
6.3.1. Schubladen definieren, was hinein soll, ohne eine Reihenfolge vorzugeben	233
6.3.2. Schubladen sind mit Ergebnismustern als Denkhilfe versehen.....	234
6.3.3. Der Schrank für die Problemstellung und der Schrank für die Lösung..	234
6.3.4. Mini-Aktivitäten.....	235
6.3.5. Vom Geschäftsprozeßmodell zum fertigen System.....	236
6.4. Geschäftsprozeßanalyse ist Analyse!	236
6.5. UML für Geschäftsprozeß- und Gesamtsystemanalyse	237
6.5.1. Das gewünschte Ergebnis	237
6.5.2. Mini-Aktivitäten zur Entwicklung eines Geschäftsprozeßmodells	238
6.5.3. Wie könnte man anfangen?	239
6.5.4. Pragmatische Hilfsmittel zur Geschäftsprozeßmodellierung	242
6.5.5. Auch das gute, alte Kontextdiagramm hilft!	244
6.5.6. Wieviel ist genug?	245
6.5.7. Zusammenfassung zur Geschäftsprozeßanalyse.....	246
6.6. Aktivitäten zur Präzisierung der Problemstellung.....	246
6.6.1. Die Schubladen der detaillierten Analyse	246
6.6.2. Mini-Aktivitäten der Detailanalyse	246
6.6.3. Heuristiken als Hilfsmittel	248
6.6.4. Beispiele für Heuristiken zur Erstellung des Klassenmodells	249
6.6.5. Pragmatische Hilfsmittel	258
6.6.6. Die Schubladen des dynamischen Modells	258
6.6.7. Füllen der Schubladen mit dynamischen Modellen	260
6.6.8. Verbindlichkeit der Elemente der Detailspezifikation	268
6.7. Die Systemebene und die Detailebene	269

6.8. Aktivitäten zur Lösung der Problemstellung	270
6.8.1. Designen heißt „Entscheidungen treffen“	270
6.8.2. Ziel ist eine System- und Software-Architektur	272
6.8.3. Probleme lösen = erweitern + bündeln	273
6.8.4. Zusammenfassung für das Design	276
6.9. Klassifizierung von Klassen	276
6.10. Vorgehensweise bei sehr großen Systemen.....	280
6.11. Zusammenfassung	282
6.12. Formulare.....	282
7. Kapitel: Messen und Schätzen.....	287
7.1. Was messen - wie schätzen?.....	289
7.2. Messen und Schätzen im Projekt	291
7.2.1. Produktgröße, Herstellungsaufwand und -zeit.....	293
7.2.2. Was ist Schätzen?	294
7.2.3. Was kann man messen?.....	296
7.2.4. Messen in iterativen Projekten	298
7.2.5. Frühwarnsysteme im Projekt.....	299
7.2.6. Automatisierung von Messungen.....	302
7.2.7. Projektnachlese.....	302
7.3. Messungen	303
7.3.1. Aufwandsmessungen.....	303
7.3.2. Größenmessungen	304
7.3.3. Messung der Produktivität.....	311
7.3.4. Produktivitätsmonitor.....	312
7.3.5. Qualitätsmonitor	312
7.3.6. Iterationsmonitor	314
7.3.7. Automatisierung der Messungen	315
7.4. Schätzungen.....	317
7.4.1. Aufwandsschätzung	318
7.4.2. Die Softwaregleichung und Faustformeln für die Makroschätzung	322
7.4.3. Anwendung der Makroschätzung	326
7.4.4. Produktschätzung mit Beispielen.....	329
7.5. Empfehlungen	335
7.5.1. Empfehlungen für das Schätzen.....	335
7.5.2. Ein Meßprogramm	336
7.5.3. Schätzen und Messen in iterativen Projekten.....	338
7.6. Hinweise für weiteres Lesen.....	339
7.7. Formulare.....	340
7.7.1. Funktionspunkte-Formular	341
7.7.2. Widgetpunkte-Formular.....	342

8. Kapitel: Die Autoren.....	343
9. Kapitel: Glossar.....	351
10. Kapitel: Literatur.....	371
11. Kapitel: Index.....	379

8. Kapitel: Die Autoren

**Bernd Oestereich**

oose.de GmbH

E-Mail: boe@oose.de

Bernd Oestereich ist Geschäftsführer der oose.de Dienstleistungen für innovative Informatik GmbH und Autor international verlegter, teilweise prämierter Buch- und Zeitschriftenpublikationen.

Er ist Mitglied in verschiedenen regionalen und überregionalen Arbeitskreisen zu objektorientierten Themen und Partner des System Bauhaus.

Mit der objektorientierten Softwareentwicklung beschäftigt er sich seit Mitte der 80er Jahre, u. a. als Coach, Projektleiter, Analytiker, Designer, Entwickler, Trainer und Berater. Seine vorrangig bearbeiteten Sachgebiete sind Vorgehensmodelle, Methodik, objektorientierte Analyse/Design, UML, Projektplanung, -organisation und -staffing.

Er hat verantwortlich in großen Projekten mitgewirkt, strategische Projekte mit angeschoben, beratend viele Projekte begleitet und systematisch objektorientiertes Know-how in verschiedenen Unternehmen aufgebaut.

Seine Publikationen sowie seine Beratungs- und Schulungstätigkeit geben immer wieder wichtige Impulse für die objektorientierte Softwareentwicklung im deutschsprachigen Raum.



Dr. Peter Hruschka

E-Mail:
phruschka@compuserve.com

Dr. Peter Hruschka ist Spezialist für Technologietransfer in der Software- und Systementwicklung. Seit 1980 gibt er Seminare und Workshops in allen Bereichen der Software-Technologie, insbesondere für objektorientierte und strukturierte Methoden. Er berät große Entwicklungsprojekte als Coach, Berater und Controller und hilft Organisationen bei der Einführung moderner Methoden und Verfahren.

Dr. Hruschka ist Mitglied der Atlantic Systems Guild, einem internationalen „Think Tank“ von Methodengurus, deren Arbeiten den State-of-the-Art wesentlich mitgestaltet haben, und Partner des System Bauhaus.

Er ist auch Autor einiger Fachbücher sowie zahlreicher Artikel. Seine Veröffentlichungen haben u. a. CASE und Real-Time-Systeme zum Thema. Er ist Redaktionsmitglied des Cutter IT-Journal von Ed Yourdon und des Objekt-Spektrum, wo er regelmäßig über objektorientierte Analyse, Design und Management schreibt.

**Nicolai M. Josuttis**

Solutions in Time

E-Mail: solutions@josuttis.de

Nicolai M. Josuttis arbeitet seit vielen Jahren im Umfeld der objektorientierten Systementwicklung. 1990 hat er für seine erste objektorientierte Entwicklung den deutschen Hochschul--Software-Preis erhalten.

Seitdem arbeitet er als Architekt, Systemanalytiker, Systementwickler, Projektleiter, Berater und Ausbilder für namhafte deutsche Firmen in Bereichen wie Kommunikation, Finanzwesen, Verkehrstechnik und Maschinenbau.

Nicolai M. Josuttis ist Autor verschiedener deutsch- und englischsprachiger Bücher und verfaßte Buchbeiträge zu den Themen „Objektorientierte Softwareentwicklung“, „C++“ und „Graphische Bedienoberflächen“.

Er veröffentlicht regelmäßig Artikel zur objektorientierten Softwareentwicklung in Zeitschriften wie Objekt-Spektrum und iX und hält regelmäßig Vorträge auf verschiedenen Konferenzen sowie für die Gesellschaft für Informatik.

Nicolai M. Josuttis ist Mitglied des C++--DIN--Arbeitskreises, ist als offizieller deutscher Vertreter an der internationalen Standardisierung von C++ aktiv beteiligt und Partner des System Bauhaus.

**Dr. Hartmut Kocher**

Cortex Brainware GmbH

E-Mail: hwk@cortex-brainware.de

Dr. Hartmut Kocher war seit 1993 Berater und später Leiter der Consulting-
abteilung bei der Firma Rational. Seit 1998 ist er einer der Geschäftsführer
der Firma Cortex Brainware Consulting & Training GmbH.

Er verfügt über mehr als 10 Jahre praktischer Erfahrungen im Bereich ob-
jektorientierter Softwareentwicklung, u. a. als Softwarearchitekt, Projektleiter
usw. Seine Spezialgebiete umfassen die Themen Projektmanagement, Ein-
führung moderner Entwicklungsprozesse und Softwarearchitektur.

Dr. Kocher hat mehrere Artikel und Buchbeiträge veröffentlicht und hält re-
gelmäßig Vorträge und Schulungen zu allen Themen im OO-Bereich.

**Dr. Hartmut Krasemann**

debis Systemhaus GmbH
Telekommunikation / Öffentlicher Bereich /
Verkehr

E-Mail: Krasemann@debis.com

Hartmut Krasemann ist Berater und Coach für objektorientierte Technologien, inkrementelle und iterative Vorgehensweisen und Projektmanagement bei debis Systemhaus.

Er gehört zu den Pionieren der Objektorientierung und der inkrementellen und iterativen Vorgehensweisen in seinem Unternehmen. In dieser Eigenschaft hat er Methoden weiterentwickelt und erprobt, die unternehmensinterne Ausbildung mitgestaltet und für die Akzeptanz von Smalltalk als Entwicklungsumgebung im Unternehmen und bei Kunden gesorgt.

Seine Erfahrungen in Entwicklung, Coaching und Anwendung der Technologie wie auch der Vorgehensweisen reicht von eingebetteten Realzeitsystemen bis hin zu größten Informationssystemen, von Einsatz- und Leitsystemen bis hin zu der operativen Software in Versicherungen und Banken.

Nach der Promotion in theoretischer Hochenergiephysik mit anschließender Forschungstätigkeit wechselte er schon früh in die Software-Industrie. Heute blickt er auf 18 Jahre Erfahrung in der Entwicklung von Systemen und Software zurück, von Assembler bis Smalltalk, vom Wasserfallmodell bis zu inkrementellen und iterativen Vorgehensmodellen.

**Markus Reinhold**

IABG mbH

E-Mail: markus.reinhold@t-online.de

Markus Reinhold studierte Informatik an der Fachhochschule Nürnberg, wie auch an der Universität Erlangen. Er befaßt sich seit mehr als 10 Jahren mit dem Thema Software-Engineering und Toolunterstützung.

Während dieser Zeit unterstützte er mehrere Firmen im In- und Ausland bei der Einführung moderner Technologien (z. B. Objektorientierung) und Werkzeuge (CASE-Tools). In diesem Zusammenhang veröffentlichte er verschiedene Artikel in diversen Fachzeitschriften und ist auch Mitautor der Studie „OO-CASE Tools und Methoden“.

Markus Reinhold ist zur Zeit Mitarbeiter der Firma IABG mbH in Ottobrunn bei München, wo er das Kompetenzzentrum Objektorientierung leitet.

Die IABG mbH ist Hauptautor des international anerkannten Entwicklungsstandards für IT-Systeme (V-Modell). Markus Reinhold war an der Integration der UML (Unified Modeling Language) in diesen Entwicklungsstandard beteiligt und hat etliche Firmen bei der Erstellung eines firmenspezifischen Vorgehensmodells auf Basis des V-Modells unterstützt.

Neben dieser beratenden Funktion ist er auch Projektleiter großer Projekte im Bereich Objektorientierung/Internet.

9. Kapitel: Glossar

Ablauforganisation

Ablauforganisation ist die zeitliche und räumliche Ordnung betrieblicher Prozesse innerhalb des durch die Aufbauorganisation geschaffenen Rahmens.

Abstraktion

Abstraktion ist eine Methode, bei der unter einem bestimmten Gesichtspunkt die wesentlichen Merkmale eines Gegenstandes oder Begriffes herausgesondert werden.

Aggregation (UML: *aggregation*) ⇨ Assoziation, ⇨ Komposition

Eine Aggregation ist eine Sonderform der Assoziation, bei der die beteiligten Klassen keine gleichwertige Beziehung führen, sondern eine Ganzes-Teile-Hierarchie darstellen. Eine Aggregation beschreibt, wie sich etwas Ganzes aus seinen Teilen zusammensetzt.

Akteur (UML: *actor*)

Ein Akteur ist eine eine außerhalb des Systems liegende ⇨ Klasse, die an der in einem ⇨ Anwendungsfall beschriebenen Interaktion mit dem System beteiligt ist. Akteure nehmen in der Interaktion gewöhnlich eine definierte Rolle ein. Ein Akteur ist eine ⇨ stereotypisierte Klasse.

Aktivität

Eine Aktivität ist die konkrete Durchführung von definierten Aktionen innerhalb eines Entwicklungsprozesses und Ausprägung eines ⇨ Aktivitätstyps.

Aktivität (UML: *action state*)

Ein Zustand mit einer internen Aktion und einer oder mehreren ausgehenden Transitionen, die automatisch dem Abschluß der internen Aktion folgen. Eine Aktivität ist ein einzelner Schritt in einem Ablauf. Eine Aktivität kann mehrere ausgehende Transitionen haben, wenn diese durch Bedingungen unterschieden werden können.

Aktivitätsdiagramm (UML: *activity diagram*)

Ein Aktivitätsdiagramm ist eine spezielle Form des Zustandsdiagramms, das überwiegend oder ausschließlich ⇨ Aktivitäten enthält.

Aktivitätstyp

Ein Aktivitätstyp ist eine abstrakte Beschreibung von Tätigkeiten, um ein oder mehrere definierte Ergebnisse zu erzeugen. Ein Aktivitätstyp ist somit eine Art Arbeitsanleitung.

Amigos

Grady Booch, James Rumbaugh und Ivar Jacobson sind die Initiatoren der *Unified Modeling Language* (UML) und des *Unified Process* und werden häufig einfach „die Amigos“ genannt.

Analyse

Mit (objektorientierter) Analyse werden alle Aktivitäten im Rahmen des Softwareentwicklungsprozesses bezeichnet, die der Ermittlung, Klärung und Beschreibung der Anforderungen an das System dienen (d. h. die Klärung, *was* das System leisten soll).

Änderungsmanagement (Tätigkeit)

Management zur Aufnahme, Verfolgung und Durchführung von Änderungen, Erweiterungen und Fehlernbeseitigungen.

Änderungsmanagement (Organisationseinheit)

Entscheidungs- und Steuerungsgremium für das Änderungsmanagement.

Anwendungsarchitektur

Die fachliche und technische \Rightarrow Architektur einer Software-Anwendung.

Anwendungsfall (UML: *use case*)

Ein Anwendungsfall beschreibt eine Menge von Aktivitäten eines Systems aus der Sicht seiner Akteure, die für die Akteure zu einem wahrnehmbaren Ergebnis führen. Ein Anwendungsfall wird stets durch einen Akteur initiiert. Ein Anwendungsfall ist ansonsten eine komplette, unteilbare Beschreibung.

Anwendungsfalldiagramm (UML: *use case diagram*)

Ein Diagramm, das die Beziehungen zwischen \Rightarrow Akteuren und \Rightarrow Anwendungsfällen zeigt.

Anwendungsfallmodell (UML: *use case model*)

Ein Modell, das die funktionalen Anforderungen an ein System in Form von Anwendungsfällen beschreibt.

Arbeitsauftrag

Beschreibt für eine Person oder ein Team den Auftrag, ein definiertes Ergebnis herzustellen. Rahmenbedingungen wie geschätzter bzw. geplanter Aufwand, Start- und Endtermin, Form und Qualität des Ergebnisses, ggf. einzusetzende Werkzeuge u. ä. werden beschrieben.

Arbeitspaket

Eine Menge von konkreten \Rightarrow Arbeitsaufträgen oder die Beschreibung einer Aufgabe, die in mehrere konkrete Arbeitsaufträge zerlegbar ist.

Architektur

ist die Spezifikation der grundlegenden Struktur eines Systems.

Architektur-Prototyp

Ein \Rightarrow Prototyp, mit dem die prinzipielle Brauchbarkeit und Funktionsfähigkeit einer technischen oder fachlichen \Rightarrow Architektur nachgewiesen werden soll.

Architekturzentriert

Im Kontext des Entwicklungsprozesses bedeutet dies, daß bereits der Entwicklungsprozeß die besonderen Gegebenheiten der \Rightarrow Architektur (Konzepte, Abstraktionen, Artefakte u. ä.) berücksichtigt.

Artefakt

Jede Art von Ergebnis oder Produkt im Rahmen des Entwicklungsprozesses.

Assoziation (UML: *association*) \Rightarrow gerichtete Assoziationen, \Rightarrow bidirektionale Assoziationen

Eine Assoziation beschreibt eine Relation zwischen Klassen, d. h. die gemeinsame Semantik und Struktur einer Menge von Objektbeziehungen.

Attribut (UML: *attribute*)

Eine benannte Eigenschaft eines Typs. Ein Attribut ist ein Datenelement, das in jedem Objekt einer Klasse gleichermaßen enthalten ist und von jedem Objekt mit einem individuellen Wert repräsentiert wird. Im Gegensatz zu Objekten haben Attribute außerhalb des Objektes, von dem sie Teil sind, keine eigene Identität. Attribute sind vollständig unter der Kontrolle der Objekte, von denen sie Teil sind.

Audit

Ein Audit ist eine Untersuchung einer Organisationseinheit oder eines Projektes, um die Einhaltung und Wirksamkeit von Regelungen, Verfahren und Standards zu ermitteln und zu dokumentieren.

Aufwand

Aufwand ist eine Größe der Dimension [Zahl von Personen] * [Zeit]. Der Aufwand in einem Projekt ergibt sich als Integral der \Rightarrow Projektbesetzung über eine jeweils vorgegebene Zeit. Der Gesamtaufwand des Projektes ist auch der \Rightarrow Herstellungsaufwand des jeweiligen Produktes.

Basisklasse \Rightarrow Oberklasse

Build

Eine gewöhnlich unvollständige und verübergene aber ausführbare Version des Systems.

Change Management \Rightarrow Änderungsmanagement

CRC-Karten (Klassenkarte)

Karteikarten, auf denen der Name der Klasse (Class), ihre Aufgaben (Responsibilities) und ihre Beziehungen (Collaborations) beschrieben werden.

Datenabstraktion

Hierunter versteht man das Prinzip, nur die auf ein Objekt anwendbaren Operationen nach außen sichtbar zu machen. Die tatsächliche innere Realisierung der Operationen und die innere Struktur des Objektes werden verborgen, d. h. man betrachtet abstrakt die eigentliche Semantik und läßt die tatsächliche Implementierung außer acht.

Default-Implementierung \Rightarrow Standard-Implementierung

Delegation

ist ein Mechanismus, bei dem ein Objekt eine Nachricht nicht (vollständig) selbst interpretiert, sondern an ein anderes Objekt weiterleitet (pro-pagiert).

Design

Mit (objektorientiertem) Design werden alle Aktivitäten im Rahmen des Softwareentwicklungsprozesses bezeichnet, mit denen ein Modell logisch und physisch strukturiert wird, und die beschreiben, *wie* das System die in der \Rightarrow Analyse beschriebenen Anforderungen erfüllt.

Domäne \Rightarrow Problembereich

Domänenmodell \Rightarrow Klassenmodell

Ein Modell, beispielsweise ein Klassenmodell, welches die fachlich relevanten Sachverhalte repräsentiert, nicht aber technisch oder fachfremd motivierte Sachverhalte.

Entwurfsmuster

Entwurfsmuster sind generalisierte Lösungsideen zu immer wiederkehrenden Entwurfsproblemen. Sie sind keine fertig codierten Lösungen, sie beschreiben lediglich den Lösungsweg.

Ergebnistyp

Ein Ergebnistyp ist ein abstrahiertes Ergebnis, d. h. es wird damit definiert, welche Inhalte, Struktur und Form ein bestimmter Typ von Ergebnissen hat (syntaktische und semantische Beschreibung). Zur Beschreibung des Ergebnistyps gehören auch Aussagen zur Qualität und Granularität.

Evolutionäres Vorgehen

Vorgehen, bei dem ausgehend von zunächst unvollständigen Anforderungen ein Ergebnis in mehreren aufeinander aufbauenden Zwischenschritten erstellt wird.

Exemplar \Rightarrow Objekt, \Rightarrow Instanz

Fachabteilung

Eine Organisationseinheit, die dem eigentlichen Geschäftszweck einer Organisation dient. Sie ist Partner der Softwareentwicklung, artikuliert Bedarf, Anforderungen und Änderungen für Anwendungen und nutzt Anwendungen.

Fachklassenmodell \Rightarrow Domänenmodell

Ein Klassenmodell, das ausschließlich oder vorwiegend fachlich motivierte Klassen enthält.

Fachliche Architektur

Ein Modell, das die grundsätzlichen fachlichen Zusammenhänge eines Anwendungsbereiches repräsentiert.

Fehler

Ein Fehler ist die Nicht-Erfüllung einer festgelegten Anforderung.

Fehlerrate

Die Zahl der während einer spezifischen Aktivität auftretenden Fehler. Die spezifische Aktivität ist z. B. eine Stunde Test, eine Stunde Benutzung, die Durchführung eines Testfalls o. ä.

Fertigstellungsgrad

ist der bereits erbrachte Anteil vom Herstellungsaufwand. Ein Fertigstellungsgrad von 70% besagt, daß noch 30% des Herstellungsaufwandes zu leisten sind, bevor das Produkt vollständig hergestellt ist.

Forward-Engineering \Rightarrow Reverse-Engineering

beschreibt die Überführung eines Modells in eine spezielle Programmiersprache, d. h. gewöhnlich die Erzeugung von Code.

Funktionspunkte

sind ein extrinsisches \Rightarrow Maß für den von außen sichtbaren Funktionsumfang und damit die Größe eines Softwareprodukts, das keine wesentliche verborgene Funktionalität enthält. Funktionspunkte sind von der International Function Points Users Group [IFPUG94] standardisiert. Vgl. auch \Rightarrow Widgetpunkte.

Generalisierung (*UML*: generalization) \Rightarrow Spezialisierung / Konkretisierung**Geschäftsobjekt** \Rightarrow Businessobjekt**Geschäftsfall** \Rightarrow Geschäftsvorfall**Geschäftsprozeß** \Rightarrow Workflow

Ein Geschäftsprozeß ist eine Zusammenfassung von organisatorisch evtl. verteilten, fachlich jedoch zusammenhängenden Aktivitäten, die notwendig sind, um einen Geschäftsvorfall (z. B. einen konkreten Antrag) ergebnisorientiert zu bearbeiten. Die Aktivitäten eines Geschäftsprozesses stehen gewöhnlich in zeitlichen und logischen Abhängigkeiten zueinander. Ein Geschäftsvorfall entsteht gewöhnlich durch ein Ereignis (z. B. Antragseingang).

Geschäftsvorfall (Vorgang)

Ein Geschäftsvorfall ist ein geschäftliches Objekt (z. B. ein konkreter Vertrag), das durch ein Ereignis ausgelöst (z. B. Antragseingang) durch die innerhalb eines Geschäftsprozesses beschriebenen Aktivitäten bearbeitet wird.

GUI

Graphical User Interface, Grafische Benutzeroberfläche

Herstellungsaufwand

Der \Rightarrow Aufwand, der erforderlich ist, um ein bestimmtes Produkt herzustellen.

Herstellungszeit

Die Zeitspanne zwischen dem Beginn der Produkterstellung und ihrem Ende, auch Projektdauer.

Heuristik

Eine Heuristik ist eine mit einer Wahrscheinlichkeit behaftete Regel („Daumenregel“).

Inkrement

Ein Inkrement ist die Erweiterung eines Produktes. Ein Inkrement ist gewöhnlich gekennzeichnet durch die Differenz zwischen zwei \Rightarrow Builds.

Inkrementelles Vorgehen

Eine Vorgehensweise, bei der ein Produkt schrittweise in wachsenden Zwischenprodukten entsteht.

Instantiierung

ist das Erzeugen eines Exemplars aus einer Klasse.

Instanz (UML: *instance*) \Rightarrow Objekt, \Rightarrow Exemplar

Für den Hausgebrauch können Instanz, Objekt und Exemplar synonym betrachtet werden. In der UML 1.0 finden sich jedoch teilweise inkonsistente Akzentuierungen.

Integration

Zusammenfügen von Teilsystemen oder Komponenten zu einem Gesamtsystem.

Integrationstest

Überprüfung, ob die durch eine \Rightarrow Integration zusammengeführten Einzelteile korrekt zusammenarbeiten und als Gesamtheit funktionieren.

Interaktionsdiagramm (UML: *interaction diagram*)

Sammelbegriff für \Rightarrow Sequenzdiagramm, \Rightarrow Kollaborationsdiagramm, \Rightarrow Aktivitätsdiagramm.

Invariante

Eine Eigenschaft oder ein Ausdruck, der über den gesamten Lebenszeitraum eines Elementes, z. B. eines Objektes gegeben sein muß.

Inspektion

Eine Sitzung zur Überprüfung einzelner Dokumente und Ergebnisse. Vgl. \Rightarrow Review.

Iteration

Eine Iteration ist ein in ähnlicher Weise mehrfach vorkommender zeitlicher Abschnitt in einem Prozeß.

Iteratives Vorgehen

Ist eine Vorgehensweise, bei der der Entwicklungsprozeß in mehrere gleichartige Zeitabschnitte zerlegt wird.

Klasse (UML: *class*)

Eine Klasse ist die Definition der Attribute, Operationen und der Semantik für eine Menge von Objekten. Alle Objekte einer Klasse entsprechen dieser Definition.

Klassenbibliothek

Eine Klassenbibliothek ist eine Sammlung von Klassen.

Klassendiagramm (UML: *class diagram*)

Ein Klassendiagramm zeigt eine Menge statischer Modellelemente, vor allem Klassen und ihre Beziehungen.

Kollaborationsdiagramm (UML: *collaboration diagram*)

Eine Kollaborationsdiagramm zeigt eine Menge von Interaktionen zwischen einer Menge ausgewählter Objekte in einer bestimmten begrenzten Situation (Kontext) unter Betonung der Beziehungen zwischen den Objekten und ihrer Topographie. Ähnlich dem \Rightarrow Sequenzdiagramm.

Komponente (UML: *component*)

Eine Komponente ist ein ausführbares Softwaremodul mit eigener Identität und wohldefinierten Schnittstellen (Sourcecode, Binärcode, DLL oder ausführbares Programm). Außerhalb der UML wird Komponente häufig anders, mehr im Sinne eines \Rightarrow Paketes definiert. Vgl. \Rightarrow Anwendungskomponente.

Komponentendiagramm (UML: *component diagram*)

Ein Komponentendiagramm zeigt die Organisation und Abhängigkeiten von \Rightarrow Komponenten.

Komposition (UML: *composite*) \Rightarrow Aggregation

Eine Komposition ist eine strenge Form der Aggregation, bei der die Teile vom Ganzen existenzabhängig sind.

Konfiguration

Eine Konfiguration ist die Gesamtheit zusammenpassender Software-Elemente.

Konfigurationseinheit

nennt man jeden einzeln versionierten und verwalteten Teil einer Software. In der Regel ist ein Entwickler für eine Konfigurationseinheit verantwortlich.

Konfigurationsdiagramm \Rightarrow Einsatzdiagramm

Konfigurationskontrollsystem

nennt man die Menge der Verfahren, Mechanismen und Werkzeuge, mit denen Konfigurationseinheiten verwaltet werden.

Konfigurationsmanagement

Vorgehensweise zur Überwachung und Kontrolle von Programmänderungen und -erweiterungen. Dazu gehört unter anderem die Festlegung der Systembestandteile sowie die Versionierung und Freigabe.

Legacy System

Synonym für ein schwer wartbares, schwer anpaßbares oder inkompatibles Alt-System.

LOC Lines of Code

ist ein intrinsisches \Rightarrow Maß für die Größe eines Software-Programms. LOC mißt die Zeilen der textuellen Quelle (nicht dagegen die Zahl der Anweisungen). Wichtig ist noch die Unterscheidung in Brutto-LOC (inklusive aller Kommentare und Leerzeilen) und Netto-LOC (exklusive aller Kommentare und Leerzeilen)

Makroschätzung

Die \Rightarrow Schätzung des Herstellungsaufwandes anhand globaler Parameter von Projekt und Produkt. Sie benutzt wesentlich die Produktgröße, um die Herstellungszeit und/oder die notwendige Projektbesetzung zu prognostizieren. Eine Makroschätzung setzt also eine Produktschätzung voraus.

Maß

Ein Maß dient dazu, Größen und Mengen festzustellen. Bei der \Rightarrow Messung der \Rightarrow Produktgröße von Software unterscheidet man zwischen intrinsischen und extrinsischen Maßen. Intrinsische Maße beziehen sich auf innere, von der Software-Produktionsumgebung abhängige Eigenschaften und sind deshalb nicht zwischen verschiedenen Umgebungen übertragbar. Extrinsische oder äußere Maße abstrahieren davon.

Mehrfachvererbung \Rightarrow Multiple Vererbung**Meilenstein**

Ein Meilenstein definiert einen Termin, zu dem eine Menge von Ergebnissen in vorgegebener Detaillierung und Vollständigkeit nachprüfbar und formal dokumentiert vorliegen soll. Ein Meilenstein ist ein Hilfsmittel zur Planung und Überwachung eines Projektes.

Messung

Die Bestimmung einer Größe oder Menge durch Vergleich mit einem \Rightarrow Maß. Das Ergebnis einer Messung wird als Produkt von Maßzahl und Maß angegeben.

Metamodell (UML: *meta model*)

Ein Modell, das die Sprache definiert, mit der ein Modell definiert werden kann.

Methode

Eine Methode ist eine Handlungsvorschrift die beschreibt, wie ein Ziel bzw. Ergebnis unter gegebenen Bedingungen erreicht werden kann.

Methode (UML: *method*) \Rightarrow Operation

In Smalltalk werden Operationen Methoden genannt. In der UML wird eine Methode als Implementierung einer Operation definiert. Für die Praxis ist es unkritisch, Methode und Operation synonym zu verwenden.

Methodologie

Methodologie ist die Lehre von den \Rightarrow Methoden.

Methodik

Bedeutsam klingendes Synonym für \Rightarrow Methode.

Metrik

Eine Meßmethode. Zu einer Metrik oder Meßmethode gehört ein \Rightarrow Maß und eine Anleitung, wie bei der Messung zu verfahren ist.

Mikroschätzung

Die Schätzung eines \Rightarrow Herstellungsaufwandes anhand detaillierter Eigenschaften von Projekt und Produkt. Zu den Eigenschaften zählen sowohl die Summe aller Aktivitäten im Projekt, die \Rightarrow Projektbesetzung und Struktur als auch eine Zerlegung des Produkts in solche Teile, die innerhalb einzelner Aktivitäten erstellt werden können.

Nachricht (UML: *message*) \Rightarrow Operation, \Rightarrow Methode

Nachricht ist ein Mechanismus, mit dem Objekte untereinander kommunizieren können. Eine Nachricht überbringt einem Objekt die Information darüber, welche Aktivität von ihm erwartet wird, d. h. eine Nachricht fordert ein Objekt zur Ausführung einer Operation auf. Eine Nachricht besteht aus einem Selektor (einem Namen), einer Liste von Argumenten und geht an genau einen Empfänger. Der Sender einer Nachricht erhält ggf. ein Antwort-Objekt zurück. Durch \Rightarrow Polymorphismus kann eine Nachricht zum Aufruf einer von mehreren gleichlautenden \Rightarrow Operationen führen.

Nebenläufigkeit

Zwei oder mehr Aktivitäten werden zeitgleich (parallel) ausgeführt.

Norm

Eine Richtlinie ist die Vorgabe eines Handlungsmusters, daß befolgt, oder einer Qualität, die eingehalten werden *muß*. Sie dient dazu, unabhängig voneinander entstehende Artefakte mit einheitlichen Eigenschaften oder Qualitäten herzustellen. Vgl. \Rightarrow Richtlinie.

Oberklasse, Superklasse, Basisklasse (UML: *superclass*) \Rightarrow Generalisierung

Eine Oberklasse ist eine Verallgemeinerung ausgewählter Eigenschaften ihrer \Rightarrow Unterklasse(n).

Objekt (UML: *object*)

Ein Objekt ist eine konkret vorhandene und agierende Einheit mit eigener Identität und definierten Grenzen, das Zustand und Verhalten kapselt. Der Zustand wird repräsentiert durch die \Rightarrow Attribute und \Rightarrow Beziehungen, das Verhalten durch \Rightarrow Operationen bzw. \Rightarrow Methoden. Jedes Objekt ist Exemplar (Synonym: Instanz) einer Klasse. Das definierte Verhalten gilt für alle Objekte einer Klasse gleichermaßen, ebenso die Struktur ihrer Attribute. Die Werte der Attribute sind jedoch individuell für jedes Objekt. Jedes Objekt hat eine eigene, von seinen Attributen u. a. unabhängige, nicht veränderbare Identität.

Objektbeziehung (UML: *link*)

Eine konkrete Beziehung zwischen zwei Objekten, d. h. die Instanz einer \Rightarrow Assoziation. Ein Objekt hat eine Beziehung zu einem anderen Objekt, wenn es eine Referenz darauf besitzt. Implementiert werden diese Referenzen gewöhnlich durch \Rightarrow Attribute, was für die Modellierung jedoch unerheblich ist.

Objektdiagramm (UML: *object diagram*)

Ein Diagramm, das Objekte und ihre Beziehungen untereinander zu einem bestimmten Zeitpunkt zeigt. Gewöhnlich ein \Rightarrow Kollaborationsdiagramm oder eine spezielle Variante des \Rightarrow Klassendiagramms.

Objektidentität

ist eine Eigenschaft, die ein Objekt von allen anderen unterscheidet, auch wenn es möglicherweise die gleichen Attributwerte besitzt.

Object Engineering Process (OEP)

Ein objektorientiertes, UML-basiertes konkretes Vorgehensmodell (Prozessleitfaden), siehe [OEP99]. Kommerzielles Produkt der Fa. oose.de GmbH.

OCL, Object Constraint Language

Die OCL definiert eine Sprache zur Beschreibung von Zusicherungen, Invarianten, Vor- und Nachbedingungen und Navigation innerhalb von UML-Modellen.

OEP \Rightarrow Object Engineering Process**OO**

ist die Abkürzung für Objektorientierung.

OMG

ist die Abkürzung für *Object Management Group*.

Operation (UML: *operation*) \Rightarrow Methode, \Rightarrow Nachricht

Operationen sind Dienstleistungen, die von einem Objekt mit einer \Rightarrow Nachricht angefordert werden können, um ein bestimmtes Verhalten zu bewirken. Sie werden implementiert durch \Rightarrow Methoden. In der Praxis werden Operation und Methode häufig synonym verwendet.

Organisationseinheit

Eine Gruppe von Personen innerhalb eines Unternehmens, die eine definierte Rolle wahrnehmen. Die Organisationseinheit ist ein Element der Aufbauorganisation des Unternehmens. Die unterste Ebene ist gewöhnlich die Abteilung.

Paket (UML: *package*)

Pakete sind Ansammlungen von Modellelementen beliebigen Typs, mit denen das Gesamtmodell in kleinere, überschaubare Einheiten gegliedert wird. Ein Paket definiert einen Namensraum, d. h. innerhalb eines Paketes müssen die Namen der enthaltenen Elemente eindeutig sein. Jedes Modellelement kann in anderen Paketen referenziert werden, gehört aber zu genau einem (Heimat-)Paket. Pakete können wiederum Pakete beinhalten. Das oberste Paket beinhaltet das gesamte System.

pattern, design pattern ⇨ Entwurfsmuster

Persistentes Objekt

Persistente Objekte (persistent: lat. „anhaltend“) sind solche, deren Lebensdauer über die Laufzeit einer Programmsitzung hinausreicht. Die Objekte werden hierzu auf nichtflüchtigen Speichermedien (z. B. Datenbanken) gehalten.

Phase

Eine Phase ist ein zeitlicher bzw. sachlogischer Gliederungsabschnitt eines Projektes. Eine Phase faßt eine Menge von Aktivitäten und Ergebnissen zu einer Planungs- und Kontrolleinheit zusammen. Am Ende jeder Phase steht ein Meilenstein, der die in der Phase zu erzielenden Inhalte definiert.

Pilot, Pilotprojekt

Ein Pilot ist ein mit einer bestimmten Verfahrensweise oder Architektur erstmals erstelltes vollständiges Ergebnis, mit dem gewöhnlich die Brauchbarkeit nachgewiesen werden soll. Während ⇨ Prototypen nur ausgewählte Aspekte des endgültigen Produktes repräsentieren, ist ein Pilot ein vollständiges und rahmenbedingungs-konformes Ergebnis. Ein Pilotprojekt ist ein Projekt, mit dem ein Pilot erstellt werden soll.

Polymorphismus

Polymorphismus (Vielgestaltigkeit) heißt, daß gleichlautende Nachrichten an kompatible Objekte unterschiedlicher Klassen ein unterschiedliches Verhalten bewirken können. Beim dynamischen Polymorphismus wird eine Nachricht nicht zur Compilierzeit, sondern erst beim Empfang zur Programmlaufzeit einer konkreten Operation zugeordnet. Voraussetzung hierfür ist das dynamische Binden.

Problembereich \Rightarrow Domäne

Anwendungsgebiet bzw. Problembereich, innerhalb dessen die fachliche Modellierung stattfindet. Als Problembereichsmodell (Domänenmodell) wird in der Regel der Teil des Gesamtmodells verstanden, der sich auf den eigentlichen fachlichen Problembereich bezieht (auch fachliches Modell genannt). Technische, querschnittliche u. ä. Aspekte gehören nicht dazu. Im Kontext von Anwendungsarchitektur ist zumeist das fachliche Klassenmodell gemeint (d. h. ohne Framework-, GUI-, Controller- u. ä. Klassen).

Produktcontrolling

ist das Nachprüfen und die daraus abgeleitete Prognose qualitativer Produkteigenschaften. Zu den qualitativen Produkteigenschaften zählen neben anderen Qualitätsmaßen die \Rightarrow Fehlerrate und die \Rightarrow Produktgröße.

Produktgröße

wird entweder intrinsisch, dann in \Rightarrow LOC, oder extrinsisch, dann meist in \Rightarrow Funktionspunkten, gemessen. Siehe auch \Rightarrow Maß. In diesem Buch werden die \Rightarrow Widgetpunkte als weiteres extrinsisches Maß für die Produktgröße vorgeschlagen.

Produktivität

ist der Quotient aus Produktgröße und Herstellungsaufwand. Da die Produktgröße extrinsisch oder intrinsisch angegeben werden kann, gibt es auch zwei „Arten“ von Produktivität: Die intrinsische oder LOC-Produktivität und die „richtige“ extrinsische Produktivität, die eine Messung der Produktgröße in Funktionspunkten oder Widgetpunkten voraussetzt.

Projekt

Ein Projekt ist ein einmaliges Vorhaben, daß zeitlich, finanziell und personell begrenzt ist und zur Erreichung eines definierten Zieles geschaffen wird. Ein Projekt verfügt über eine projektspezifische Organisation.

Projektbesetzung

ist eine Kurve über der Zeit, die angibt, wieviele Personen zur Zeit in einem Projekt arbeiten. Der \Rightarrow Aufwand eines Projektes, z. B. auch der \Rightarrow Herstellungsaufwand eines Produktes, ergibt sich als Integral der Projektbesetzung über die \Rightarrow Herstellungszeit.

Projektcontrolling

ist das Nachprüfen des erbrachten \Rightarrow Aufwands, der Vergleich mit dem \Rightarrow Fertigstellungsgrad und die daraus abgeleitete Prognose für den weiteren Verlauf des Projekts. Es liefert damit sozusagen die Positionsbestimmung des Projektleiters.

Projektmanagement

Gesamtheit der Aufgaben und Techniken zur Führung und Organisation eines Projektes.

Projektorganisation

Gesamtheit der Organisationseinheiten eines Projektes. Gewöhnlich setzt sich die Projektorganisation aus Bestandteilen der vorhandenen Betriebsorganisation zusammen.

Projektplan

Ein Plan der den grundsätzlichen „Fahrplan“ des Projektes darstellt, er enthält ⇒Meilensteine und beschreibt die vorgesehenen ⇒Phasen und ⇒Iterationen des Projektes.

Protokoll

Eine Menge von ⇒Signaturen.

Prototyp

Ein Prototyp ist ein vorläufiges oder temporäres Produkt, mit dem ausgewählte Eigenschaften oder Aspekte des zu entwickelnden endgültigen Produktes erfahrbar und beurteilbar gemacht werden sollen. Prototypen können explorativ sein, d. h. zur Erforschung eines Sachverhaltes dienen, sie können experimentell sein, d. h. zur Überprüfung der Machbarkeit oder Funktionsfähigkeit dienen, oder evolutionär sein, d. h. vorab ein Teilprodukt bereitstellen.

Prototyping

Herstellung eines ⇒Prototyps.

Prozeßmodell ⇒Vorgehensmodell**Qualität**

Grad der Erfüllung von geforderten Leistungen und Eigenschaften.

Qualitätssicherung

Maßnahmen, die zur Erlangung, Steigerung oder Kontrolle von Qualität beitragen. Dazu gehören planerische Maßnahmen, konstruktive Maßnahmen (Werkzeuge, Richtlinien, Konventionen etc.), analytische Maßnahmen (Test, Validierung, Verifizierung).

Rahmenwerk (UML: *framework*)

Ein Rahmenwerk ist eine Menge kooperierender Klassen, die unter Vorgabe eines Ablaufes („*Don't call the framework, the framework calls you*“) eine generische Lösung für eine Reihe ähnlicher Aufgabenstellungen bereitstellen.

Rational Unified Process (RUP)

Ein objektorientiertes, UML-basiertes konkretes Vorgehensmodell, siehe [Kruchten98]. Kommerzielles Produkt der Fa. Rational Software.

Refaktorisierung

Umstrukturierung eines Systems zugunsten besserer Wart- oder Erweiterbarkeit ohne Änderung der Funktionalität.

Referentielle Integrität

Regel, die die Integrität von Objektbeziehungen beschreibt, vor allem für den Fall, daß eines der beteiligten Objekte oder die Objektverbindung selbst gelöscht werden sollen.

Regressionstest

Hiermit wird die Wiederholung früherer Tests bezeichnet. Er dient dazu, nachzuweisen, daß bereits zu einem früheren Zeitpunkt korrekt vorhandene Systemfunktionalität immer noch vorhanden ist, auch wenn zwischenzeitlich neue Funktionalität hinzugekommen ist.

Release

Ein ⇒Build, das an externe Anwender ausgeliefert werden kann.

Restaufwandsschätzung

Die Schätzung des noch verbleibenden ⇒Herstellungsaufwandes zu einem fortgeschrittenen Zeitpunkt im Projekt, das ein bestimmtes Produkt herstellt.

Reverse-Engineering ⇒Forward-Engineering, ⇒Roundtrip-Engineering

Hiermit bezeichnet man einen Vorgang, bei dem aus vorhandenem Programmiersprachen-Code ein (visuelles) Modell erzeugt werden soll.

Review

Ein Review ist eine Sitzung zur kritischen Begutachtung des aktuellen Projektstands auf Basis repräsentativer (Teil-)Ergebnisse mit dem Ziel, den Status zu ermitteln und mögliche Beiträge zur weiteren Problemlösung aufzunehmen.

Richtlinie

Eine Richtlinie ist die Vorgabe eines Handlungsmusters, daß befolgt, oder einer Qualität, die eingehalten werden *sollte*. Sie dient dazu, unabhängig voneinander entstehende ⇒Artefakte mit einheitlichen, ähnlichen oder vergleichbaren Eigenschaften oder Qualitäten herzustellen. Vgl. ⇒Norm.

Roundtrip-Engineering ⇒Forward-Engineering, ⇒Reverse-Engineering

Hiermit bezeichnet man den Versuch, durch abwechselndes Forward- und Reverse-Engineering Code und Modell permanent oder regelmäßig konsistent zu halten.

RUP ⇒Rational Unified Process**Schätzung**

ist eine Prognose. Man kann die Größe eines Produktes schätzen oder mit einer Projektschätzung den erforderlichen Aufwand zur Herstellung des Produktes.

Schnittstelle (UML: *interface*) \Leftrightarrow Schnittstellenklassen

Schnittstellen beschreiben einen ausgewählten Teil des extern sichtbaren Verhaltens von Modellelementen (hauptsächlich von Klassen und Komponenten), d. h. eine Menge von \Leftrightarrow Signaturen.

Schnittstellenklasse

Schnittstellenklassen sind \Leftrightarrow abstrakte Klassen (genauer: Typen), die ausschließlich \Leftrightarrow abstrakte Operationen definieren. Schnittstellenklassen sind Klassen, die mit dem \Leftrightarrow Stereotyp «*interface*» gekennzeichnet sind. Sie sind Spezifikationen des extern sichtbaren Verhaltens von Klassen und beinhalten eine Menge von \Leftrightarrow Signaturen für Operationen, die Klassen, die diese Schnittstelle bereitstellen wollen, implementieren müssen.

Schnittstellenvererbung

Innerhalb einer \Leftrightarrow Spezialisierungsbeziehung wird lediglich eine \Leftrightarrow Schnittstelle vererbt.

Sequenzdiagramm (UML: *sequence diagram*)

Eine Sequenzdiagramm zeigt eine Menge von Interaktionen zwischen einer Menge ausgewählter Objekte in einer bestimmten begrenzten Situation (Kontext) unter Betonung der zeitlichen Abfolge. Ähnlich dem \Leftrightarrow Kollaborationsdiagramm. Sequenzdiagramme können in generischer Form existieren (Beschreibung aller möglichen Szenarien) oder in Instanzform (Beschreibung genau eines speziellen \Leftrightarrow Szenarios).

SEU

Software-Entwicklungsumgebung

Signatur

Die Signatur einer Operation setzt sich zusammen aus dem Namen der Operation, ihrer Parameterliste und der Angabe eines evtl. Rückgabetyps.

Software-Element

Ein Software-Element ist die kleinste unteilbar zu behandelnde und eindeutig zu identifizierende Einheit, die einem Änderungs- und Freigabeverfahren unterworfen werden kann.

Softwareentwicklungsplan

Zusammenfassung aller zur Projekt- und Iterationsplanung notwendigen Unterlagen und Ergebnisse.

Spezialisierung, Generalisierung \Leftrightarrow Vererbung

Eine Generalisierung (bzw. Spezialisierung) ist eine taxonomische Beziehung zwischen einem allgemeinen und einem speziellen Element (bzw. umgekehrt), wobei das speziellere weitere Eigenschaften hinzufügt, die Semantik erweitert und sich kompatibel zum allgemeinen verhält. Generalisierung und Spezialisierung sind Abstraktionsprinzipien zur hierarchischen Strukturierung der Modellsemantik unter einem diskriminierenden Aspekt (\Leftrightarrow Diskriminator).

Standard

Eine \Rightarrow Richtlinie oder Konvention, um bestimmte Mindesteigenschaften sicherzustellen.

Standard-Implementierung

Konkrete Implementierung einer eigentlich abstrakten Operation, um für Subklassen ein Standardverhalten bereitzustellen.

Stereotyp (UML: *stereotype*)

Stereotypen sind projekt-, unternehmens- oder methodenspezifische Erweiterungen vorhandener Modellelemente des UML-Metamodells. Entsprechend der mit der Erweiterung definierten Semantik wird das Modellierungselement, auf das es angewendet wird, direkt semantisch beeinflusst. In der Praxis geben Stereotypen vor allem die möglichen Verwendungszusammenhänge einer Klasse, einer Beziehung oder eines Paketes an. Andere erweiternde Mechanismen in der UML sind \Rightarrow Eigenschaftswerte und \Rightarrow Zusicherungen. (Duden: *das Stereotyp*).

Subklasse (UML: *subclass*) \Rightarrow Unterklasse**Subsystem \Rightarrow Komponente**

Ein Subsystem ist eine sehr große Komponente oder eine, die sich aus einer Menge von Einzelkomponenten zusammensetzt. Diese Unterscheidung ist für die Gliederung sehr großer Systeme hilfreich.

Superklasse (UML: *superclass*) \Rightarrow Oberklasse**System Bauhaus**

Das System Bauhaus ist ein Netzwerk international bekannter und im deutschsprachigen Raum tätiger unabhängiger Fachleute für objektorientierte Systementwicklung (www.system-bauhaus.de).

Systemtest \Rightarrow Integrationstest**Szenario** (UML: *scenario*)

Ein Szenario ist eine spezifische Folge von Aktionen. Beispielsweise ein konkreter Ablaufpfad in einem Anwendungsfall (sozusagen eine Instanz des Anwendungsfalls). Vgl. \Rightarrow Sequenzdiagramm.

Test

Testen ist eine Tätigkeit mit der Absicht, Fehler zu finden. Ein Test kann die Existenz eines Fehlers beweisen. Die Abwesenheit von Fehlern ist nicht beweisbar.

Typ (UML: *type*)

Definition einer Menge von Operationen und Attributen. Andere Elemente sind typkonform, wenn sie über die durch den Typen definierten Eigenschaften verfügen. Wird in der Praxis häufig gleichgesetzt mit der Beschreibung von \Rightarrow Schnittstellen.

UML ⇨ Unified Modeling Language (UML)

Unified Modeling Language (UML)

Durch die Object Management Group (OMG) international standardisierte Notation und Semantik zur Beschreibung von Software-Modellen. Unified Process

Unified Process ⇨ Unified Software Development Process (USDP)

Unterklasse, Subklasse (UML: *subclass*)

Eine Unterklasse ist die Spezialisierung einer Oberklasse und erbt alle Eigenschaften der Oberklasse.

USDP ⇨ Unified Software Development Process (USDP)

Use case ⇨ Anwendungsfall

Validierung

Überprüfung, ob das richtige Ergebnis bzw. Produkt hergestellt wurde.

Verifizierung

Überprüfung, ob ein Ergebnis bzw. Produkt richtig ist, d. h. gegebene Anforderungen erfüllt.

Vererbung (UML: *inheritance*) ⇨ Einfachvererbung, ⇨ Multiple Vererbung, ⇨ Multiple Klassifikation, ⇨ Dynamische Klassifikation

Vererbung ist ein Programmiersprachenkonzept für die Umsetzung einer Relation zwischen einer Ober- und einer Unterklasse, wodurch Unterklassen die Eigenschaften ihrer Oberklassen mitbenutzen können. Vererbung implementiert normalerweise ⇨ Generalisierungs- und Spezialisierungsbeziehungen. Alternativen: ⇨ Delegation, ⇨ Aggregation, ⇨ generische Programmierung, ⇨ generisches Design.

Verteilungsdiagramm (UML: *deployment diagram*)

Ein Diagramm, welches die Konfiguration der zur Laufzeit vorhandenen (eingesetzten) ⇨ Knoten und ihrer ⇨ Komponenten, Prozesse und Objekte zeigt.

V-Modell

Vorgehensmodell. Meistens wird damit das erstmalig 1992 in Form des „Softwareentwicklungsstandard der Bundeswehr“ veröffentlichte Vorgehensmodell bezeichnet. Vgl. ⇨ 37ff. und [VM97].

Vorgehensmodell

Modellhafte und häufig formale Beschreibung, wie in Projekten vorgegangen werden soll.

Wartung

ist derjenige Teil des Software-Lebenszyklus, der nach dem Ende der Garantiezeit liegt. Hierunter werden auch Tätigkeiten verstanden, die eine bestehende Anwendung verbessern, optimieren, reparieren oder überprüfen mit dem Ziel, die Anwendung weiterhin bzw. besser nutzen zu können.

Widget

Ein Widget ist ein vorgefertigtes Oberflächenelement in den Programmierumgebungen für moderne grafische Benutzungsoberflächen. Der Programmierer benutzt es wie ein Bauteil, entweder deklarativ in einem Kompositionseditor oder durch prozeduralen Aufruf.

Widgetpunkte

sind ein extrinsisches \Rightarrow Maß für den Funktionsumfang und damit die die Größe eines Softwareprodukts. Widgetpunkte werden in diesem Buch definiert. Siehe auch \Rightarrow Funktionspunkte.

Workflow \Rightarrow Geschäftsprozeß

Ein Workflow ist die computergestützte Automatisierung und Unterstützung eines Geschäftsprozesses oder eines Teils davon.

Zusicherung (UML: *constraint*)

Eine Zusicherung ist ein Ausdruck, der die möglichen Inhalte, Zustände oder die Semantik eines Modellelementes einschränkt, und der stets erfüllt sein muß. Bei dem Ausdruck kann es sich um ein \Rightarrow Stereotyp oder um \Rightarrow Eigenschaftswerte handeln, um eine freie Formulierung (\Rightarrow Notiz) oder um eine \Rightarrow Abhängigkeitsbeziehung. Zusicherungen in Form reiner boolescher Ausdrücke werden auch \Rightarrow *assertions* genannt.

Zustandsdiagramm (UML: *state diagram, state machine*)

Ein Zustandsdiagramm zeigt eine Folge von Zuständen, die ein Objekt im Laufe seines Lebens einnehmen kann und aufgrund welcher Stimuli Zustandsänderungen stattfinden. Ein Zustandsdiagramm beschreibt eine hypothetische Maschine (Endlicher Automat), die sich zu jedem Zeitpunkt in einer Menge endlicher Zustände befindet.

10. Kapitel:Literatur

[Albrecht79]

A. J. Albrecht: *Measuring Application Development Productivity*, Proc. Joint SHARE/GUIDE/IBM Application Development Symposium, Okt. 1979, S. 83-92; nachgedruckt in T. C. Jones: *Programming Productivity - Issues for the Eighties*, IEEE Press, Nr. EH0239-4, 1986, S. 35-44.

[Alexander77]

C. Alexander et al.: *A Pattern Language*, Oxford University Press, New York, 1977.

[Alexander79]

C. Alexander et al.: *The Timeless Way of Building*, Oxford University Press, New York, 1979.

[Beck99]

K. Beck: *Extreme Programming*, Vortrag auf der OOP-Konferenz, München 1999.

[Bittner95]

U. Bittner, W. Hesse, J. Schnath: *Praxis der Software-Entwicklung, Methoden, Werkzeuge, Projektmanagement - eine Bestandsaufnahme*. Oldenbourg, München, 1995.

[Boehm81]

B. W. Boehm: *Software Engineering Economics*, Prentice Hall, Englewood Cliffs, 1981.

[Boehm86]

B. W. Boehm: *A spiral model of software development and enhancement*, Software Engineering Notes 11(4), 1986.

[Boehm88]

B. Boehm: *A spiral Model of Software Development and Enhancement*, Computer Magazine, S. 61-72, Mai 1988.

[Boehm89]

B. Boehm: *Software Risk Management*, IEEE Computer Press, 1989.

[Booch94]

G. Booch: *Object-oriented analysis and design with applications*, 2nd ed., Benjamin/Cummings, Redwood City, 1994. Deutsche Ausgabe: *Objektorientierte Analyse und Design; Mit praktischen Anwendungsbeispielen*. Addison-Wesley, Bonn, 1994.

[Booch96]

G. Booch: *Object Solutions: Managing the Object-Oriented Project*, Addison-Wesley, Reading, MA, 1996.

[Booch98]

Booch, G.; Rumbaugh, J., Jacobson, I.: *The Unified Modeling Language User Guide*, Addison Wesley, 1999

[Brooks75]

F. P. Brooks: *The Mythical Man-Month*, Addison-Wesley, Massachusetts, 1975. Deutsche Ausgabe: *Vom Mythos des Mann-Monats*, Addison-Wesley, Bonn 1987.

[Buschmann96]

F. Buschmann, et al.: *Pattern-Oriented Software Architecture - A system of patterns*, John Wiley & Sons, Chicester, GB, 1996.

[Cantor98]

M. R. Cantor: *Object-Oriented Project Management with UML*, Wiley, New York, 1998.

[CMU95]

Carnegie Mellon University: *The Capability Maturity Model: Guidelines for improving the software process*, Addison-Wesley, Reading, MA, 1995.

[Cockburn97]

A. Cockburn: *Surviving Object-Oriented Projects*, Addison Wesley, Reading, 1997

[Constantine99]

Larry L. Constantine, Lucy A. D. Lockwood: *Software for Use, A Practical Guide to the Models and Methods of Usage-Centered Design*, ACM Press/Addison Wesley, New York, 1999.

[deChampeaux97]

D. deChampeaux: *Object Oriented Development Process and Metrics*, Prentice Hall, Upper Saddle River 1997.

[DeMarco87]

Tom DeMarco, Timothy Lister: *Peopleware: Productive Projects and Teams*, Dorset House, 1987 (deutscher Titel: *Wien wartet auf Dich*).

[DeMarco98]

T. DeMarco: *Der Termin: Ein Roman über Projektmanagement*, Hanser 1998.

[Denert92]

Ernst Denert: *Software-Engineering - Methodische Projektabwicklung*, Springer, 1992.

[Dröschel98]

W. Dröschel et al., *Inkrementelle und objektorientierte Vorgehensweise mit dem V-Modell '97*, Oldenbourg Verlag, München, 1998

[Fowler97]

M. Fowler: *Analysis Patterns: Reusable Object Models*, Addison-Wesley, 1997.

[Gamma95]

E. Gamma, et al.: *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[Goldberg84]

Goldberg, Adele, *Smalltalk-80: The Interactive Programming Environment*. Addison-Wesley, Reading, MA, 1984

[Goldberg95]

A Goldberg, K. S. Rubin: *Succeeding with Objects, Decision Frameworks for Project Management*, Addison-Wesley, Reading, 1995

[Graham95]

I. Graham: *Migrating to Object Technology*, Addison Wesley, Wokingham 1995.

[Graham97]

Graham, Ian, Henderson-Sellers, Brian, Younessi, Houman: *The OPEN Process Specification*, Addison-Wesley, New York, 1997

[Henderson96]

B. Henderson-Sellers: *Object-Oriented Metrics*, Prentice Hall, Upper Saddle River 1996.

[Henderson97]

B. Henderson-Sellers, I. G. Graham, D. G. Firesmith: *Methods unification: The OPEN methodology*, JOOP, May 1997, S. 41ff.

[Hatley99]

Derek Hatley, Peter Hruschka, Imtiaz Pirbhai: *Process for System Architecture and Requirements Engineering*, Dorset House, New York, 1999.

[Hruschka96]

Hruschka, P: *Wohin mit den Funktionen im Objektmodell?*, in: Objekt-Spektrum Mai/Juni 1996, Nr. 3

[Humphrey89]

W. Humphrey: *Managing the Software Process*, Addison Wesley, Reading 1989.

[IFPUG94]

International Function Point Users Group (IFPUG): *Counting Practices Manual*, Westerville, Ohio, January 1994

[Jacobson92]

I. Jacobson, M. Christerson, P. Jonsson, G. Övergaard: *Object-Oriented Software Engineering, A Use Case Driven Approach*, Addison-Wesley, Workingham, 1992.

[Jacobson99]

I. Jacobson, G. Booch, J. Rumbaugh: *The Unified Software Development Process*, Addison-Wesley, Reading, MA, 1999.

[Jones96]

T. C. Jones: *Applied Software Measurement*, 2nd. ed., McGraw-Hill, New York 1996.

[Jones98]

T. C. Jones: *Estimating Software Costs*, 2nd. ed., McGraw-Hill, New York 1998.

[Joneswww]

www.spr.com/library/0langtbl.htm.

[JSR97]

James Robertson: *On setting the context - Some Notes*, <http://www.atlsysguild.com/Site/James/contextart.html>

[Kemerer93]

C. F. Kemerer: *Reliability of Function Point Measurement: A Field Experiment*, Comm. of the ACM, 36 (1993) 85

[Knuth 92]

Knuth, Donald E.: *Literate Programming*, CSLI Lecture Notes no. 27

[Kruchten95]

P. Kruchten: *The 4+1 View Model of Architecture*, IEEE Software, November 1995.

[Kruchten98]

P. Kruchten: *The Rational Unified Process – An Introduction*, Addison-Wesley, Reading, MA, 1998.

[Larman98]

C. Larman: *Applying UML and Patterns*, Prentice Hall, Upper Saddle River, 1998.

[Londeix87]

B. Londeix: *Cost Estimation for Software Development*, Addison Wesley, Wokingham 1987.

[Lorenz94]

M. Lorenz, J. Kidd: *Object Oriented Software Metrics*, Prentice Hall, Englewood Cliffs 1994.

[McMenamin88]

McMenamin, S.; Palmer, J: *Essential Systems Analysis*, Yourdon Press/Prentice Hall 1984; deutsch: *Strukturierte Systemanalyse*, Carl-Hanser-Verlag, 1988

[Martin95]

Robert C. Martin: *Designing Object-Oriented C++ Applications Using the Booch Method*, Prentice Hall, Englewood Cliffs, NJ, 1995.

[Martin96a]

Robert C. Martin: *Applying the Booch Method*, OOP, 1996.

[Martin96b]

Robert C. Martin: *Implementing Object Technology, A Case Study of OOD and Reuse in C++, OOP*, 1996.

[McCon96]

S. McConnell: *Rapid Development - Taming wild software schedules*, Microsoft Press, Redmond, WA, 1996.

[Meyer 97]

Meyer B.: *Object-Oriented Software Construction*, 2nd. ed., Prentice Hall, Upper Saddle River, 1997

[Müller-Ettrich98]

G. Müller-Ettrich: *Objektorientierte Prozeßmodelle: UML einsetzen mit OOTC, V-Modell, Objectory*, Addison Wesley Longman, Bonn, 1998.

[Norden60]

P. V. Norden: *On the Anatomy of Development Projects*, IRE Transactions on Engineering Management, PGEM, EM-7 (1960) 34.

[OEP99]

Object Engineering Process (OEP): www.oose.de/oep

[Oestereich96]

B. Oestereich: *Objektorientierung braucht ein evolutionäres Vorgehensmodell*, in: Computerwoche 12, 22.3.1996, S. 22.

[Oestereich98]

B. Oestereich: *Objektorientierte Softwareentwicklung: Analyse und Design mit der UML*, 4. Auflage, Oldenbourg, München, 1998.

[Oestereich98]

B. Oestereich: *Developing Software with UML: Object-Oriented Analysis and Design in Practice*, Addison Wesley, Harlow, 1999.

[OMG 93]

Object Management Group: *Object-oriented Analysis and Design Reference Model*, Framington, MA (1993)

[Putnam78]

L. H. Putnam: *A General Empirical Solution to the Macro Software Sizing and Estimating Problem*, IEEE Trans. Software Engr., July 1978, 345

[Reinhold98]

M. Reinhold, G + C. Versteegen, *CASE und OO-Methoden*, IT-Research, 1998

[Robertson94]

Robertson, J. & S.: *Complete Systems Analysis: The Textbook, the Workbook, The Answers*, Dorset House, New York, 1998; deutsch: *Vollständige Systemanalyse*, Carl-Hanser-Verlag, München, 1996

[Royce98]

W. Royce: *Software Project Management – A Unified Framework*, Addison-Wesley, Reading, MA, 1998.

[Rumbaugh91]

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.: *Object-Oriented Modelling and Design*, Prentice-Hall, Englewood Cliffs, 1991.

[Rumbaugh93]

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.: *Objektorientiertes Modellieren und Entwerfen*, Hanser, München, 1993.

[Rumbaugh97c]

J. Rumbaugh: *Modeling through the development process*, in: JOOP May 1997, S. 5ff.

[Shaw 96]

Shaw, Mary, Garlan, David: *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall, Upper Saddle River, NJ, 1996

[Smalltalk 80]

Goldberg, Adele und Robson, David, *Smalltalk-80: The Language and its Implementation*, Addison-Wesley, Reading, MA, 1986

[Soukup94]

Jiri Soukup: *Taming C++ - Pattern Classes and Persistence for Large Projects*, Addison-Wesley, 1994.

[Stroustoup98]

Bjarne Stroustoup: *Die C++ Programmiersprache*, 3. Auflage, Addison Wesley, Bonn, 1998.

[VM97]

www.v-modell.iabg.de

[Waldén95]

K. Waldén, J.-M. Nerson: *Seamless Object-Oriented Software Architecture, Analysis and Design of Reliable Systems*, Prentice Hall, London, 1995.

[Weinand97]

André Weinand: *Tücken der Objektorientierung - Ein Erfahrungsbericht*, OOP, 1997.

[Wirfs-Brock90]

R. Wirfs-Brock, B. Wilkerson, L. Wiener: *Designing Object-Oriented Software*, Prentice Hall, Englewood Cliffs, 1990. Deutsche Ausgabe: *Objektorientiertes Software-Design*, Hanser, München, 1993.

11. Kapitel:Index

- A**
- Ablauforganisation 353
 - Abnahme 110
 - Abnahmetest 314, 319
 - Abnahmeumgebung 118
 - Abstraktion 63, 353
 - Aggregation 353
 - Akteur 35, 97, 239, 353
 - Aktivität 39, 43, 48, 97, 134, 235, 353
 - debitiv 135
 - fakultativ 135
 - im USDP 33
 - im V-Modell 46
 - obligatorisch 135
 - periodisch 52
 - zur Präzisierung der Aufgabe 246
 - Aktivitätsdiagramm 353
 - Aktivitätstyp 353
 - Altsysteme 125
 - Amigos 93, 353
 - Analyse 236, 354
 - Analyse-Modell 35
 - Änderungshäufigkeit 291, 298, 313, 337
 - je Konfigurationseinheit 314
 - Änderungsmanagement 113, 354
 - Änderungsstatistik 317
 - Anforderungsanalyse 100, 107
 - Anforderungsspezifikation 117
 - Angebot 292
 - Anti-Pattern 193
 - Anwenderforderungen 41, 49
 - Anwendertest 111
 - Anwendungsarchitektur 354
 - Anwendungsfall 100, 354
 - finden 239
 - Anwendungsfalldiagramm 354
 - anwendungsfallgetriebenes Vorgehen 99
 - Anwendungsfallmodell 35, 354
 - Arbeitsauftrag 124, 354
 - Arbeitspaket 354
 - Architekt 85
 - Architektur 41, 55, 65, 117, 272, 354
 - Prototyp 77, 354
 - architekturspezifisch 99
 - Architekturteam 202
 - architekturzentriert 101, 355
 - Artefakt 97, 355
 - Aspekt 83
 - Assoziation 355
 - Attribut 355
 - Audit 355
 - Aufwand 291, 302, 355
 - Abhängigkeit von der Produktgröße 323, 324
 - Abhängigkeit von der Projektdauer 323
 - je Iteration 314
 - je Konfigurationseinheit 314
- Autoren** 343
- B**
- Backfiring 326, 328
 - Basisklasse 361
 - Baustein 102
 - begleitende Aktivität 134
 - Beteiligte 35
 - Betriebsorganisation 116
 - Betriebsphase 119
 - Booch, Grady 28
 - Booch-Methode 28
 - Build 112, 355
 - Bündelklasse 279
 - bündeln 274
 - Business-View-Klasse 278
- C**
- Capability Maturity Model 176, 178, 194
 - Change Management 355
 - Checkliste 54
 - CMM *siehe* Capability Maturity Model
 - COCOMO 191
 - constraint 370
 - Construction 96
 - Controller 79
 - CRC-Karte 158, 355
 - crisis 84, 195
- D**
- Datenabstraktion 356
 - Datenbank-Interface-Klasse 278
 - Datenflußdiagramm 264
 - Dauer
 - der Highlevel-Studie 68
 - von Iterationen 32, 121
 - von Projekten 323
 - debitive Aktivität 135
 - Default-Implementierung 356
 - Delegation 356
 - deployment diagram 369
 - Design 270, 356
 - design pattern 363
 - Design-Modell 35
 - design-to-schedule 83
 - Dokumentation 86, 211
 - Anwenderdokumentation 212
 - Entwicklungsdokumentation 215
 - Entwicklungsplan 213
 - Iterationsplan 214
 - Managementdokumentation 213
 - Produktvision 213
 - Statusberichte 214
 - Domäne 356
 - Domänenexperte 85
 - Domänenmodell 356
 - Durchstich 77

- E**
- Einführungsphase 96, 119
 - Einsatzmodell 35
 - Elaboration 96
 - encapsulation 86
 - Entity-Klasse 277
 - Entwicklungsleitung 105
 - Entwicklungsprozeß 177
 - Auswahl des 18
 - bei strategischen Projekten 58
 - Development Case 193
 - Einführung 178
 - iterativ und inkrementell 22
 - Optimierung 26
 - projektspezifische Anpassung 193
 - Wasserfallprozeß 19
 - Entwurfsmuster 356
 - Entwurfsphase 96, 117
 - Ereignisliste 240
 - Ergebnis 97
 - Ergebnistyp 356
 - Essentielle Systemanalyse 239
 - evolutionär 42, 82, 87, 356
 - evolutionäre Auslieferung 83
 - evolutionäres Prototyping 80
 - evolutionary delivery 83
 - evolutionary prototyping 80
 - Exemplar 356
 - externer Meilenstein 120, 121
 - Extreme Programming 120
 - exzeptionelle Aktivität 134
- F**
- Fachabteilung 356
 - Fachklassenmodell 357
 - fachliche Architektur 357
 - fachliche Projektleitung 104
 - fachliches Modell 364
 - fakultative Aktivität 135
 - Feature Team 206
 - Fehler 357
 - Mikroschätzung 295
 - Statistik 296
 - Fehlerdichte 298
 - Fehlerhäufigkeit 291, 314
 - C++ 314
 - in Modulen 314
 - Smalltalk 314
 - Fehlerrate 300, 302, 313, 337, 357
 - Fehlerstatistik 317
 - Fertigstellungsgrad 299, 357
 - Flexibilität 64
 - Forward-Engineering 357
 - framework 365
 - Fremdsoftware 74
 - Funktionspunkte 306, 340, 357
 - Rückwärtsrechnen 326
- G**
- Generalisierung 357, 367
 - generischer Entwicklungsprozeß 39
 - Gesamtprojektleitung 105
 - Geschäftsfall 357
 - Geschäftsobjekt 357
 - Geschäftsprozeß 16, 357, 370
 - Geschäftsprozeßanalyse 116, 236
 - Geschäftsvorfall 357
 - Glossar 351
 - große Systeme 280
 - GUI 358
- H**
- Herstellungsaufwand 293, 358
 - Herstellungskosten 293
 - Herstellungszeit 293, 294, 358
 - Heuristik 229
 - Highlevel-Analyse 62
 - Highlevel-Design 64
 - Highlevel-Studie 62
 - Hruschka, Peter 346
- I**
- Implementierungs-Modell 35
 - Inception 96
 - Information-Hiding 86
 - Inkrement 358
 - inkrementell 22, 42, 81, 99, 103, 358
 - Inspektion 358
 - Instantiierung 358
 - Instanz 358
 - Integration 70, 112, 358
 - Integrationstest 118, 218, 319, 358
 - Integrationstest 313
 - Integrität, referentielle 366
 - Interaktionsdiagramm 358
 - interner Meilenstein 120, 121, 132
 - Invariante 358
 - IP-Matrix 124
 - ISO 9000 176
 - Ist-Zustand 62
 - Iteration 96, 359
 - im USDP 32
 - Konsolidierung 25
 - Iterationsende 117, 118
 - Iterationsmanagement 113
 - Iterations-Meilenstein 121
 - Iterationsmonitor 299, 314
 - Iterationsplan 124
 - Iterations-Planungs-Matrix 124
 - Iterations-Zielplanung 124
 - iterativ 22, 42, 76, 99, 103, 359
 - iterative Aktivität 134
- J**
- Jacobson, Ivar 29
 - Josuttis, Nicolai 347
- K**
- Kapselung 86
 - Kernteam 85

- Klasse 359
 Kategorisierung 65, 72, 276
 Klassenbibliothek 359
 Klassendiagramm 359
 Klassenmodell 249
 Klassifizierung 276
 Kocher, Hartmut 348
 Kollaborationsdiagramm 359
 Kommunikation 63
 Komponente 102, 359
 Komponentendiagramm 359
 Komposition 359
 Konfiguration 359
 Konfigurationsdiagramm 359
 Konfigurationseinheit 312, 359
 Konfigurationskontrollsystem 300, 360
 Konfigurationsmanagement . 39, 51, 112, 360
 Konstruktionsphase 96, 118
 Kontextdiagramm 244
 Konzeption 55, 62, 87
 Konzeptualisierung 96
 Krasemann, Hartmut 349
 kreatives Team 205
 Krise 84, 195
 Krisenmanagement 195
- L**
- Legacy System 360
 Lenkungsausschuß 121
 Lines of Code 360
 Literaturverzeichnis 371
 LOC 360
- M**
- Makroschätzung 294, 318, 321, 360
 Beispiele 326
 Management 221
 Maß 360
 Brutto-LOC 305
 extrinsisch 297, 305
 intrinsisch 297
 Netto-LOC 305
 Qualität 303
 Zahl der Klassen 305, 312
 Zahl der Worte 306
- Matrix
 Iterationsplanung 124
 Subsysteme und Aspekte 83
 zum Vorgehen 82
- Mehrfachvererbung 360
 Meilenstein 96, 121, 188, 360
 der Einführungsphase 119
 der Entwurfsphase 117
 der Konstruktionsphase 118
 der Vorbereitungsphase 115
 extern 120, 121
 intern 120, 121, 132
- Messen 303
 Aufwand 303
 Automatisierung 302, 315
- Funktionspunkte 307
 Produktgröße 304
 Produktivität 311
 Prozesse 291
 Meßprogramm 336
 Messung 303, 360
 Metamodell 360
 Methode 361
 Methodik 361
 Methodologie 361
 Metrik 361
 Klassengröße 301
 LOC / Klasse 302, 313, 316, 337
 LOC / Widgetpunkt 337
 LOC / Widget-Punkt 316
 LOC/Widgetpunkt 315
- Migrationskonzept 111
 Mikroprozeß 102
 Mikroschätzung 295, 318, 361
 Mitarbeiter, Anzahl 324
 Modell, fachlich 364
 Modellbildung 176
 Model-View-Controller 79
 Modularisierung 101
 Motivation 207
 MVC 79
- N**
- Nachbar-Interface-Klasse 279
 Nachkalkulation 302
 Nachricht 361
 Nebenläufigkeit 361
 Nichtlinearität 294
 Norm 361
 Notation 177
- O**
- Oberklasse 361
 Object Constraint Language 362
 Object Engineering Process 93, 362
 Object Modeling Technique 28
 Objekt 362
 persistent 363
 Objektbeziehung 362
 Objektdiagramm 362
 Objektidentität 362
 obligatorische Aktivität 135
 OCL 362
 OEP 93, 362
 Oestereich, Bernd 345
 OMG 362
 OMT 28
 OO 362
 OOSE 29
 Operation 362
 Organisationseinheit 363
- P**
- Paket 65, 102, 363
 Paketest 218

- Partitionierung 65
 pattern 363
 periodische Aktivität 134
 persistentes Objekt 363
 Phase 96
 Phasen 363
 im USDP 31
 Meilenstein 121
 Phasenmodell 31, 53
 Pilot 363
 Pilotprojekt 363
 Planung 188
 und iteratives Vorgehen 25
 Planungsdokumente 123
 Polymorphismus 363
 Priorität 66
 Problembereich 364
 Produkt 39, 43, 47, 48
 Fehler in der Schätzung 329
 im USDP 35
 im V-Modell 46
 Qualität 292
 Schätzung 329
 Zerlegung 329
 Produktcontrolling 291, 292, 364
 Produktgröße 291, 297, 302, 304, 364
 Lines of Code 305
 Produktionsumgebung 119
 Produktivität 292, 311, 337, 364
 extrinsisch 297, 311
 intrinsisch 297, 311
 Technologieabhängigkeit 322
 Produktivitätsmonitor 312
 Projekt 58, 364
 Aufwand 293
 strategisch 58
 Team 85
 Projektaufgaben
 global 319
 Management 319
 nach Phase 319
 Projektbesetzung 294, 364
 Projektcontrolling 291, 292, 364
 Projektdauer 324
 Abhängigkeit von der Produktgröße 323
 Projektdurchführung 225
 Projektleiter 85, 104
 Projektleitfaden 54
 Projektleitung 85, 104
 Entwicklungsleitung 105
 fachlich 104
 technisch 105
 Projektmanagement 30, 39, 51, 112, 364
 fatal 60
 Projektnachlese 292, 302, 319
 Projektorganisation 365
 Projektphasen 318
 Projektplan 124, 365
 Projektsteuerung 171
 Protokoll 365
 Prototyp 77, 365
 Prototyping 365
 Prozeß 97 *siehe* Entwicklungsprozeß
 im USDP 33
 Prozeßframework 48
 Prozeßhandbuch 178
 Prozeßleitfaden 93
 Prozeßmanagement 113
 Prozeßmodell 365
- Q**
- Qualität 365
 Qualitätsmanagement 105
 Qualitätsmonitor 300, 312
 Qualitätssicherung 39, 51, 105, 110, 365
- R**
- Rahmenwerk 365
Rational Unified Process 35, 365
 Rayleigh-Verteilung 294, 322
 Realisierung 68
 Refaktorisierung 76, 365
 referentielle Integrität 366
 Regressionstest 221, 366
 Reinhold, Markus 350
 Release 366
 Ressourcen 35
 Restaufwandsschätzung 366
 Reverse-Engineering 366
 Review 111, 366
 revolutionär 58
 Rhythmus 88
 Richtlinie 366
 für konzeptionelles Vorgehen 88
 Risiko 18, 66, 181
 Analyse 182
 Bewertung 180
 Steuerung 183
 Risikomanagement 66, 179
 „Top Ten“-Liste 185
 Rolle 35, 46, 48, 97
 beim USDP 35
 Root Cause Analysis 194
 Roundtrip-Engineering 366
 Rumbaugh, James 28
 RUP 35, 366
- S**
- SAP 125
 Schätzen 294
 Angebot 291
 Aufwand 291, 317, 318
 Fehlerquellen 295
 Funktionspunkte 333
 Iterationen 320
 Klassen 331
 Kosten 318
 Produkt 317
 Projekt 317
 prozedurale Zerlegung 330

- Prozesse 291
 Restaufwand 291, 299
 T-Stich 335
 Widgetpunkte 333
 Worte einer Spezifikation 332
 Zerlegung 295
 Schätzfehler 330
 Schätzung 190, 317, 366
 Schichtenmodell 100
 Schnittstelle 367
 Schnittstellenklasse 367
 Schnittstellenvererbung 367
 Schubladenmodell 232
 Seeheim-Modell 79
 Sequenzdiagramm 260, 367
 SEU 367
 Signatur 367
 singuläre Aktivität 134
 Software Engineering Institute 176
 Softwarearchitektur 55, 204, 272
 Software-Element 367
 Softwareentwicklungsplan 117, 118, 124, 367
 Softwaregleichung 322
 Software-IC 65
 Software-Pflege und -Änderung 47
 Softwarequalität 300
 Soll-Zustand 63
 Spezialisierung 367
 Spiralenmodell 27
 staged delivery 83
 Standard 368
 Standard-Implementierung 368
 Stereotyp 368
 Steuerung 171
 Steuerungsklasse 277
 Strategie 57
 strategisches Projekt 58
 stufenweise Auslieferung 83
 Subklasse 368, 369
 Submodelle des V-Modell 39
 Subsystem 65, 83, 102, 368
 Superklasse 361, 368
 SW-Architektur 47
 SW-Entwurf 47
 System Bauhaus 368
 Systemanalyse 239
 Systemarchitektur 47, 55, 272
 Systemerstellung 28, 30, 39, 108
 Systemerstellung.Fachliche 108
 Systemerstellung.Projektextern 109
 Systemerstellung.Subsysteme 109
 Systemerstellung.Umfeld 108
 System-IC 65
 Systempartitionierung 101
 Systemtest 300, 302, 313, 319, 368
 Szenario 47, 241, 368
- T**
- Team 85, 200
 Abstraktionisten 87, 204
- Architekt 85
 Architekturteam 202
 Domänenexperte 85
 Entwickler 204
 Feature Team 206
 Kernteam 85
 kreative Teams 205
 Projektleiter 85
 SWAT-Team 206
 Tiger Team 206
 Teamgeist 210
 Teammanagement 224
 Technische Anforderungen 47
 Technologiefaktor 321, 322, 325
 Test 110, 368
 Anwender 111
 formal 218
 Integration 118
 Paket 218
 Regressionstest 221
 Testaufwand 189, 303
 Testdaten 111
 Testfälle 35, 69, 111, 218
 Test-Modell 35
 Testplan 111
 Testprogramm 221
 Teststrategie 24, 69
 Testumgebung 112, 118, 119
 Testvorbereitung 102
 Tiger Team 206
 Timeboxing 132
 Transition 96
 Typ 368
- U**
- Umgebungsfaktor 321
 Umgebungsmanagement 112
 UML 15, 27, 93, 177, 369
 zur Gesamtsystemanalyse 237
 zur Geschäftsprozessanalyse 237
 Umstrukturierung 76
 Unified Method 15
 Unified Modeling Language *Siehe* UML
 Unified Process *Siehe* USDP
 Unified Software Development Process96
Siehe USDP
 Unterklasse 369
 USDP 15, 30, 93, 96, 369
 Aktivitäten 33
 Iterationen 32
 Phasen 31
 Produkte 35
 Prozesse 33
 Rollen 35
 use case 354 *siehe* Anwendungsfall
 use case diagram 354
 use case model 354
 Use-Case-Spezifikation 242
 User-Interface-Klasse 278

V

Validierung.....	369
Verbheuristik.....	252
Vererbung.....	369
und Wiederverwendung	72
Verifizierung	369
Verteilungsdiagramm.....	369
Verteilungsmodell	35
V-Modell.....	37
Aktivitäten.....	46
Produkte	46
Vorbereitungsphase.....	115
Vorgang.....	357
Vorgehensmatrix	82
Vorgehensmodell	16, 17, 93
Auswahl	175
Vorgehensweise ... <i>siehe</i> Entwicklungsprozeß bei strategischen Projekten.....	58
risikoorientiert	69, 186
Vorstudie	115

W

Wallrabe, Arne	79
Wartung.....	119, 369
Wasserfallprozeß.....	19, 58
Weiterverwendung	74
Werkzeug	218
Widget.....	309, 370
Widgetpunkte	308, 340, 370
automatisch zählen	311
zählen	309
Zählung validieren	311
Wiederverwendung	39, 71
<i>Worker</i>	35, 97
Workflow	97, 370

Z

zeitpunktgerichteter Entwurf.....	83
Zeitvorgaben	84
Zielplanung (Iteration)	124
Zielvereinbarung	129
Zusicherung.....	370
Zustandsdiagramm.....	260, 370