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.

Ludwig Mies van der Rohe

Vorwort

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.

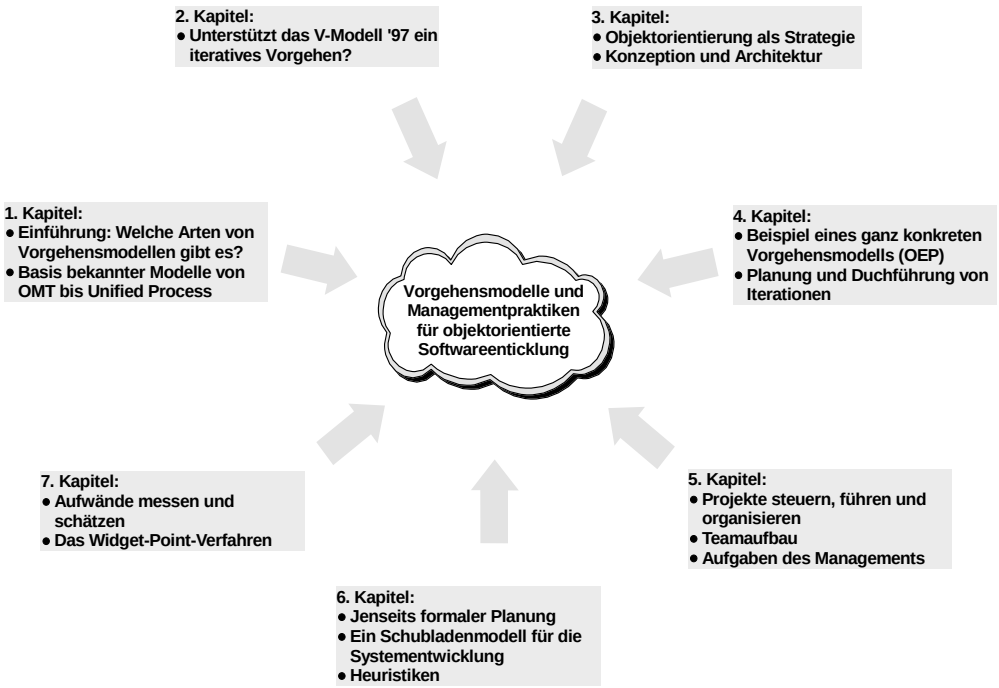
Die Objektorientierung hat mittlerweile einen festen Platz in der Softwareentwicklung und ist zu einer Basistechnologie geworden. In vielen Unternehmen hat sich die objektorientierte Softwareentwicklung jedoch noch nicht etabliert. Es laufen dort vielleicht gerade die ersten Prototypen, Pilotprojekte 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

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 Consultingabteilung 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 objektorientierter Softwareentwicklung, u. a. als Softwarearchitekt, Projektleiter usw. Seine Spezialgebiete umfassen die Themen Projektmanagement, Einführung moderner Entwicklungsprozesse und Softwarearchitektur.

Dr. Kocher hat mehrere Artikel und Buchbeiträge veröffentlicht und hält regelmäßig 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 Kompetenzcenter 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 Moeling Language* (UML) und des *Unified Process* und werden häufig einfach „die Amigos“ genannt.

Analyse
Mit (objektorientierter) Analyse werden alle Aktiviäten 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 ⇨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 ⇨Akteuren und ⇨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 ⇨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 ⇨Prototyp, mit dem die prinzipielle Brauchbarkeit und Funktionsfähigkeit einer technischen oder fachlichen ⇨Architektur nachgewiesen werden soll.

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

Artefakt
Jede Art von Ergebnis oder Produkt im Rahmen des Entwicklunsgprozesses.

Assoziation (UML: *association*) ⇨gerichtete Assoziationen, ⇨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 ⇨Projektbesetzung über eine jeweils vorgegebene Zeit. Der Gesamtaufwand des Projektes ist auch der ⇨Herstellungsaufwand des jeweiligen Produktes.

Basisklasse ⇨Oberklasse

Build
Eine gewöhnlich unvollständige und verübergehende aber ausführbare Version des Systems.

Change Management ⇨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 ⇨Standard-Implementierung

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

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

Domäne ⇨Problembereich

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

Fachliche Architektur
Ein Modell, daß 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 ⇨Reverse-Engineering
beschreibt die Überführung eines Modells in eine spezielle Programmiersprache, d. h. gewöhnlich die Erzeugung von Code.

Funktionspunkte
sind ein extrinsisches ⇨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 ⇨Widgetpunkte.

Generalisierung *(UML:* generalization*)* ⇨Spezialisierung /Konkretisierung

Geschäftsobjekt ⇨Businessobjekt

Geschäftsfall ⇨Geschäftsvorfall

Geschäftsprozeß ⇨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 ⇨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 ⇨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*) ⇨Objekt, ⇨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 ⇨Integration zusammengefügten Einzelteile korrekt zusammenarbeiten und als Gesamtheit funktionieren.

Interaktionsdiagramm (UML: *interaction diagram*)
Sammelbegriff für ⇨Sequenzdiagramm, ⇨Kollaborationsdiagramm, ⇨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. ⇨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 ⇨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 ⇨Paketes definiert. Vgl. ⇨Anwendungskomponente.

Komponentendiagramm (UML: *component diagram*)
Ein Komponentendiagramm zeigt die Organisation und Abhängigkeiten von ⇨Komponenten.

Komposition (UML: *composite*) ⇨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 ⇨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 ⇨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 ⇨Schätzung des Herstellungsaufwandes anhand globaler Parameter von Projekt und Produkt. Sie benutzt wesentlich die Produktgröße, um die Herstellungszeit und/oder die notwendige Projektbesetung zu prognostizieren. Eine Makroschätzung setzt also eine Produktschätzung voraus.

Maß
Ein Maß dient dazu, Größen und Mengen festzustellen. Bei der ⇨Messung der ⇨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 ⇨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 ⇨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*) ⇨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 ⇨Methoden.

Methodik
Bedeutsam klingendes Synonym für ⇨Methode.

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

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

Nachricht (UML: *message*) ⇨Operation, ⇨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 ⇨Polymorphismus kann eine Nachricht zum Aufruf einer von mehreren gleichlautenden ⇨Operationen führen.

Nebenläufigkeit
Zwei oder mehr Aktiviäten 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. ⇨Richtlinie.

Oberklasse, Superklasse, Basisklasse (UML: *superclass*) ⇨Generalisierung
Eine Oberklasse ist eine Verallgemeinerung ausgewählter Eigenschaften ihrer ⇨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 ⇨Attribute und ⇨Beziehungen, das Verhalten durch ⇨Operationen bzw. ⇨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 ⇨Assoziation. Ein Objekt hat eine Beziehung zu einem anderen Objekt, wenn es eine Referenz darauf besitzt. Implementiert werden diese Referenzen gewöhnlich durch ⇨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 ⇨Kollaborationsdiagramm oder eine spezielle Variante des ⇨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 zu Beschreibung von Zusicherungen, Invarianten, Vor- und Nachbedingungen und Navigation innerhalb von UML-Modellen.

OEP ⇨Object Engineering Process

OO
ist die Abkürzung für Objektorientierung.

OMG
ist die Abkürzung für *Object Management Group*.

Operation (UML: *operation*) ⇨Methode, ⇨Nachricht
Operationen sind Dienstleistungen, die von einem Objekt mit einer ⇨Nachricht angefordert werden können, um ein bestimmtes Verhalten zu bewirken. Sie werden implementiert durch ⇨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 endgültiges 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 ⇨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-, Controler- u. ä. Klassen).

Produktcontrolling

ist das Nachprüfen und die daraus abgeleitete Prognose qualitiativer Produkteigenschaften. Zu den qualitiativen Produkteigenschaften zählen neben anderen Qualitätsmaßen die ⇨Fehlerrate und die ⇨Produktgröße.

Produktgröße

wird entweder intrinsisch, dann in ⇨LOC, oder extrinsisch, dann meist in ⇨Funktionspunkten, gemessen. Siehe auch ⇨Maß. In diesem Buch werden die ⇨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 ⇨Aufwand eines Projektes, z. B. auch der ⇨Herstellungsaufwand eines Produktes, ergibt sich als Integral der Projektbesetzung über die ⇨Herstellungszeit.

Projektcontrolling

ist das Nachprüfen des erbrachten ⇨Aufwands, der Vergleich mit dem ⇨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, Verfizierung).

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

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

Schnittstellenvererbung
Innerhalb einer ⇨Spezialisierungsbeziehung wird lediglich eine ⇨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 ⇨Kollaborationsdiagramm. Sequenzdiagramme können in generischer Form existieren (Beschreibung aller möglichen Szenarien) oder in Instanzform (Beschreibung genau eines speziellen ⇨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 ⇨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 (⇨Diskriminator).

Standard
Eine ⇨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 ⇨Eigenschaftswerte und ⇨Zusicherungen. (Duden: *das* Stereotyp).

Subklasse (UML: *subclass*) ⇨Unterklasse

Subsystem ⇨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)* ⇨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 ⇨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. ⇨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 ⇨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 ⇨Maß für den Funktionsumfang und damit die die Größe eines Softwareprodukts. Widgetpunkte werden in diesem Buch definiert. Siehe auch ⇨Funktionspunkte.

Workflow ⇨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 ⇨Stereotyp oder um ⇨Eigenschaftswerte handeln, um eine freie Formulierung (⇨Notiz) oder um eine ⇨Abhängigkeitsbeziehung. Zusicherungen in Form reiner boolescher Ausdrücke werden auch ⇨*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 Functin 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 Orineted 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., Lorenson, W.: *Object-Oriented Modelling and Design*, Prentice-Hall, Englewood Cliffs, 1991.

[Rumbaugh93]
Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorenson, W.: *Objektorientiertes Modellieren und Entwerfen*, Hanser, München, 1993.

[Rumbaugh97c]
J. Rumbaugh: *Modeling througt 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