Objektorientierte Systemanalyse

– Kursübersicht –

KE 1 : 1 Grundlagen des Software Engineering
2 Objektorientierte Softwareentwicklung im Überblick

KE 2 : 3 Erstellung des Pflichtenheftes
4 Fachliche Modellierung
5 Modellierung der Benutzungsoberfläche

Inhalt dieser Kurseinheit

Inhaltsverzeichnis

Dieser Kurs wird betreut und geprüft von Univ.-Prof. Dr. Stefan Strecker,
Lehrstuhl für Betriebswirtschaftslehre, insb. Entwicklung von Informationssystemen,
Fakultät für Wirtschaftswissenschaft (http://www.fernuni-hagen.de/evis).

Erste Anlaufstelle für Fragen zu Lehrinhalten ist die mit dem Kurs
korrespondierende Moodle-Lernumgebung
(http://www.fernuni-hagen.de/evis/studium/moodle.shtml) und das dortige Diskussionsforum.

Vorwort

Der objektorientierte Ansatz repräsentiert gegenwärtig den Stand der Technik in der Softwareentwicklung. Nach weit verbreiteter Überzeugung trägt die Objektorientierung heutigen Herausforderungen der Softwareentwicklung besser Rechnung als der ältere strukturierte Entwicklungsansatz. Dies betrifft die Beherrschung von Problemstellungen hoher Komplexität ebenso wie die Wartungsfreundlichkeit und den Bedarf nach wiederverwendbarer Software. Komplexitätssprünge wurden in der jüngeren Vergangenheit insbesondere durch die Verbreitung von grafischen Benutzungsoberflächen und verteilten Anwendungen ausgelöst. Die genannten Entwicklungen erwiesen sich als Katalysatoren für den Übergang zur objektorientierten Programmierung und zur Objektorientierung überhaupt. Immer mehr Softwareentwicklerinnen und -entwickler sind mit der Aufgabe konfrontiert, sich mit der Objektorientierung vertraut zu machen.

Der objektorientierte Ansatz umfasst neben der objektorientierten Programmierung die Felder der objektorientierten Analyse und des objektorientierten Entwurfs. Der vorliegende Kurs und der Kurs „Objektorientierter Systementwurf“ (Kursnr. 00819) bieten gemeinsam eine Einführung in alle drei genannten Felder. Dabei wendet sich der vorliegende Kurs vor allem an Studierende, die künftig mit der objektorientierten Modellierung von Anwendungen zu tun haben, wie sie am Anfang einer Anwendungsentwicklung steht. Gegenstand dieses Kurses ist also die objektorientierte Analyse. Dem Anliegen der Wirtschaftsinformatik gemäß wird die objektorientierte Analyse hier als eine Modellierungsmethode für betrieblich-kommerzielle Softwareanwendungen behandelt.

Der Kurs ist wie folgt strukturiert. Das erste Kapitel gibt eine Einführung in die Grundlagen des Software Engineering. Hierbei wird der gesamte Software-Entwicklungsprozess von der Analyse bis zum Test betrachtet, ohne dass ein spezieller Entwicklungsansatz zugrunde gelegt wird. Die Darstellung fällt notwendigerweise recht allgemein und knapp aus. Wer beim ersten Lesen Verständnisschwierigkeiten hat, sollte später nochmals zum ersten Kapitel zurückkehren. Für die Einführung in das Software Engineering wurden vor allem BALZERT (1996), BALZERT (1998) sowie PAGEL und SIX (1994) zu Rate gezogen.

Im zweiten Kapitel wird der objektorientierte Ansatz der Softwareentwicklung zunächst im Überblick vorgestellt, wobei für Vergleichszwecke auch der strukturierte Ansatz charakterisiert wird. In den Kapiteln drei bis fünf werden wesentliche Aktivitäten der Analyse bei einer objektorientierten Entwicklung im Einzelnen besprochen. Dabei handelt es sich um die Erstellung des Pflichtenheftes, die fachliche Modellierung einer Anwendung sowie die Modellierung der Benutzungsoberfläche. Im Zusammenhang mit der fachlichen Modellierung werden sowohl die Konzepte der Objektorientierung systematisch eingeführt als auch die methodische Vorgehensweise bei ihrer Anwendung behandelt. Zur Notation der objektorientierten Konzepte wird durchgängig die Unified Modeling Language (UML) verwendet, die inzwischen als Standardsprache der objektorientierten Modellierung gilt. Die Darstellung der objektorientierten Analyse orientiert sich in erster Linie an dem in BALZERT (1999) vorgestellten Ansatz.

Alle Aktivitäten der objektorientierten Analyse werden anhand eines kursbegleitenden Projekts demonstriert. Es entstammt dem Bereich der Logistik, behandelt

die Erstellung eines Tourenplanungssystems für ein kleineres Unternehmen und wird im Kurs unter dem Stichwort „Tourenplanungssystem“ (kurz TPS) referenziert.

Zugleich sollen die Kursteilnehmer ein weiteres Projekt selbständig bearbeiten. Dabei handelt es sich um ein (stark vereinfachtes) Medienausleihsystem für eine Universitätsbibliothek. Dieses Projekt wird im Kurs unter dem Stichwort „Bibliotheksausleihsystem“ (kurz BAS) referenziert.

Zur Bearbeitung des Kurses erscheinen einige Hinweise angebracht:

- Voraussetzung für eine erfolgreiche Bearbeitung des Kurses ist ein Grundwissen über die Informationsverarbeitung in einem Umfang, wie es im Kurs „Grundzüge der Wirtschaftsinformatik“ (Kursnr. 00008) des Grundstudiums der Wirtschaftsinformatik vermittelt wird. Darüber hinaus wird empfohlen, vor einer Bearbeitung dieses Kurses den Kurs „Datenmodellierung und Datenbanksysteme“ (Kursnr. 00817) zu belegen.
- Der Kurs enthält zahlreiche Übungsaufgaben, deren Lösung zur Erlangung eines anwendungsbereiten Wissens erforderlich ist. Dies gilt vor allem für die Übungsaufgaben zur Anwendung der objektorientierten Konzepte, darunter die Aufgaben zur selbständigen Modellierung des Bibliotheksausleihsystems.
- Bestandteil dieses Kurses sowie des Kurses „Objektorientierter Systementwurf“ (Kursnr. 00819) sind weitere Lernmaterialien, die sich im Moodle-Kurs (https://moodle.fernuni-hagen.de/) zum Download befinden. Informationen zu den Dateien befinden sich in der Datei *readme.txt* innerhalb der *Kursprojekte-Datei* (*.zip).
- Der Download enthält alle Entwicklungsdokumente, den Quellcode sowie die ausführbaren Programme des Tourenplanungs- sowie des Bibliotheksausleihsystems. Die Entwicklungsdokumente sowie der Prototyp der Benutzungsoberfläche des BAS stellen Musterlösungen der entsprechenden Übungsaufgaben dieses Kurses dar. Die Lösungen aller übrigen Übungsaufgaben finden sich jeweils am Ende der Kurseinheiten.
- Zur Bearbeitung von Übungsaufgaben der Modellierung sowie für Einsendearbeiten sollte ein geeignetes Werkzeug verwendet werden. Angeboten wird die kostenlose Nutzung des Entwicklungstools Together der Firma TogetherSoft Corporation. Die Benutzung eines Entwicklungstools zur Modellierung wird empfohlen, allerdings nicht zur Pflicht gemacht.
- Zur Modellierung von Benutzungsoberflächen wird im Kurs ein Prototyp, d.h. eine bereits lauffähige Anwendung erstellt, die eine reduzierte Version der künftigen Benutzungsoberfläche umfasst. Für die Erstellung des Prototyps der Benutzungsoberfläche des Bibliotheksausleihsystems ist die C++-Programmierumgebung MS Visual Studio oder der GNU C-Compiler (GCC) erforderlich. MS Visual Studio ist für FernUniverstät-Studierende kostenlos über Dreamspark erhältlich, den kommandozeilen-orientierten GNU C-Compiler erhalten Sie über http://gcc.gnu.org. Grundsätzlich besteht die Möglichkeit, in den Studienzentren (STZ) der FernUniversität eine geeignete C++-Umgebung zu nutzen. Setzen Sie sich hierzu mit einem STZ in Ihrer Nähe oder auch mit der Nutzerberatung des Zentrums für Medien und IT (ZMI) in Verbindung. Eine STZ-Liste befindet sich in Ihren Studienunterlagen; Hinweise zu den STZ sowie zur ZMI-Nutzerberatung finden Sie auch auf den WWW-Seiten der FernUniversität (http://www.fernuni-hagen.de) unter den Rubriken „Informati-

onen für Studierende“ bzw. „ZMI“. Bemerkt sei noch, dass die Programmierung von Benutzungsoberflächen nicht Gegenstand von Klausuraufgaben und Einsendearbeiten ist.

- Bei technischen oder Verständnisproblemen dürfen Sie sich gern an den Lehrstuhl für Betriebswirtschaftslehre, insbesondere Entwicklung von Informationssystemen wenden. Beachten Sie die Angaben zu Betreuer und Sprechzeit im aktuellen Personal- und Kursverzeichnis der FernUniversität. Eine Kontaktaufnahme per E-Mail (Lehrstuhl.Strecker@FernUni-Hagen.de) ist ebenfalls möglich. Auch kritische Hinweise zu diesem Kurs werden gern entgegengenommen.

Für ihre Unterstützung bei der Kurserstellung bin ich allen Beteiligten zu großem Dank verpflichtet. Hermann Gehring bot mir die Möglichkeit, den Kurs zu schreiben und übernahm das Korrekturlesen. Attila Zadanji hat entscheidenden Anteil an der Erstellung des Tourenplanungssystems und schrieb einen Basistext zur Programmierung von Prototypen mit MS Visual C++. Zum Tourenplanungssystem leistete auch Markus Daiber als Diplomand einen wesentlichen Beitrag. Das Bibliotheksausleihsystem wurde von Boris Weingerl in einer Diplomarbeit erstellt. Markus Bremshey zeichnete alle Abbildungen und gestaltete das Layout des Kurses. Silvia Vecera übernahm teilweise die Erfassung des Kurstextes. Bernd Strauß sorgte immer wieder für dringend benötigte technische Unterstützung. Allen sei ganz herzlich gedankt!

Lernziele

allgemeine Lernziele

Der Kurs „Objektorientierte Systemanalyse“ behandelt die objektorientierte Analyse zur Entwicklung betrieblicher Anwendungssysteme und vermittelt darüber hinaus allgemeine Grundlagen des Software Engineering. Nach dem Durcharbeiten des Kurses sollen Sie einen Überblick des Software Engineering erworben haben sowie die Methode der objektorientierten Analyse einschließlich ihrer UML-Notation kennen und selbständig anwenden können.

Hiermit sollen Sie eine solide Basis für die eigenständige Abrundung Ihrer Kenntnisse und Fähigkeiten in der objektorientierten Analyse erworben haben, nach der Sie zur objektorientierten Modellierung von betrieblichen Anwendungssystemen in der Lage sind.

Aus dieser allgemeinen Zielsetzung leiten sich Teilziele für die einzelnen Kurseinheiten ab. Nach der Bearbeitung der ersten Kurseinheit sollen Sie im einzelnen folgende kenntnisorientierten und verhaltensorientierten Lernziele erreicht haben:

Teilziele für die erste Kurseinheit

- Sie kennen die grundlegenden Anforderungen an die industrielle Softwareentwicklung und haben ein Problembewusstsein der praktischen Erfordernisse und Schwierigkeiten der heutigen Softwareentwicklung gewonnen.
- Sie sind mit dem Begriff des Software Engineering vertraut und kennen den Gegenstand, die Zielsetzung, den allgemeinen Ansatz und wesentliche Instrumente dieser Disziplin.
- Sie können den Software-Entwicklungsprozess in grundlegende Entwicklungsaktivitäten zerlegen und haben zwei maßgebliche Prozessmodelle zur logisch-zeitlichen Strukturierung von Projekten mit ihren Vor- und Nachteilen kennengelernt. Sie können die grundsätzliche Verwendbarkeit beider Prozessmodelle anhand der Gegebenheiten eines Projekts einschätzen und wissen um die heutige Bedeutung von Werkzeugen für die Softwareentwicklung.
- Sie haben einen Überblick der Teilgebiete Projektmanagement und Qualitätssicherung des Software Engineering gewonnen und kennen die dort zu lösenden Aufgaben und einige wichtige, in der Praxis angewandte Verfahrensansätze.
- Sie kennen die wichtigsten objektorientierten Konzepte, Grundzüge des objektorientierten Entwicklungsansatzes und die Bedeutung der UML als einer einheitlichen objektorientierten Modellierungssprache.
- Sie wissen, welche Besonderheiten die grundlegenden Entwicklungsaktivitäten bei einer objektorientierten Entwicklung aufweisen und welche Vorzüge der objektorientierte Ansatz gegenüber dem strukturierten Ansatz besitzt.

1 Grundlagen des Software Engineering

1.1 Software und Softwareentwicklung

Software

Unter Software versteht man Computerprogramme sowie zugehörige Daten und Dokumentationen, die für die Anwendung eines Programms benötigt werden. Auch die bei der Erstellung eines lauffähigen Produkts erarbeiteten Entwicklungsdokumente werden unter dem Sammelbegriff Software subsumiert.

System- und Anwendungssoftware

Software lässt sich nach ihrem Einsatzzweck grob in Systemsoftware und Anwendungssoftware gliedern. Zur Systemsoftware zählen vor allem Betriebssysteme, aber z.B. auch Systeme zur Datenbankverwaltung und Programmentwicklung. Ihr Zweck besteht darin, die von der Hardware bzw. dem Computer bereitgestellte Funktionalität derart zu erweitern und zu verbessern, dass Anwendungssoftware einen geeigneten Rahmen für ihre Ausführung vorfindet und möglichst komfortabel entwickelt werden kann. Anwendungssoftware dient dagegen der Lösung von jeweils benutzerspezifischen Aufgaben der Informationsverarbeitung und ist daher für eine gewisse Benutzergruppe von unmittelbarem Interesse.

betriebliche Anwendungssysteme

Gegenstand der Wirtschaftsinformatik sind vorrangig betriebliche Anwendungssysteme, die auch als betriebliche Informationssysteme bezeichnet werden. Ihre Entwicklung nach dem objektorientierten Ansatz ist Gegenstand des Kurses. Betriebliche Anwendungssysteme kommen auf verschiedenen Ebenen und in verschiedenen Funktionsbereichen eines Unternehmens zum Einsatz. Sie lassen sich grob in Administrations- und entscheidungsorientierte Dispositionssysteme für operative Aufgaben, Kontroll- und Planungssysteme für das Controlling bzw. längerfristige Planungen sowie Führungsinformationssysteme zur Unterstützung der strategischen Unternehmenssteuerung gliedern (vgl. GEHRING 1999).

Software- und Systementwicklung

Häufig wird zwischen Softwareentwicklung und Systementwicklung differenziert. Die letztere umfasst im Falle betrieblicher Anwendungssysteme auch deren Einbettung in die Aufbau- und Ablauforganisation eines Unternehmens sowie ggf. die Beschaffung oder Entwicklung spezialisierter Hardware. Im vorliegenden Kurs wird nur die eigentliche Softwareentwicklung betrachtet.

Das Software Engineering stellt als eine Teildisziplin der Informatik wie auch der Wirtschaftsinformatik das methodische Rüstzeug für die Herstellung von Software bereit. Um eine Vorstellung von der Notwendigkeit und der Bedeutung des Software Engineering zu gewinnen, soll im Folgenden näher auf Anforderungen und Probleme eingegangen werden, die sich bei der Softwareentwicklung stellen.

1.1.1 Anforderungen und Probleme der Softwareentwicklung

industrielle Softwareproduktion und ihre ...

Zunächst ist zu betonen, dass hier stets von industrieller Softwareproduktion die Rede ist. Es geht also nicht etwa um eine Programmentwicklung für den Eigenbedarf, die von einer Einzelperson in ihrer Freizeit betrieben wird. Vielmehr handelt es sich um Softwareentwicklung als Tätigkeit eines Unternehmens, deren Produkt für einen Auftraggeber bestimmt ist. Der Auftraggeber kann dabei eine Fachabteilung desselben Unternehmens, ein Fremdunternehmen oder der – etwa durch den

Unternehmensvertrieb repräsentierte – anonyme Softwaremarkt sein. Ferner weisen industriell erstelle Softwareprodukte meist einen beachtlichen Umfang auf, werden daher arbeitsteilig produziert und ihre Herstellung erfordert neben der eigentlichen Entwicklungstätigkeit auch ein Management.

... grundlegenden Anforderungen

Aufgrund der genannten Merkmale ist eine industrielle Softwareherstellung stets mit folgenden grundlegenden Anforderungen konfrontiert:

(1) Realisierung des vorgesehenen Funktionsumfangs.
Das Softwareprodukt muss das leisten, was der Auftraggeber erwartet bzw. was ihm vertraglich zugesichert wurde. Ein Buchhaltungsprogramm etwa, das nur über Funktionen für Kreditoren, nicht aber für Debitoren verfügt, wird kaum den Beifall des Auftraggebers finden.

(2) Berücksichtigung festgelegter Qualitätsanforderungen.
Meist werden zusätzlich zum Funktionsumfang gewisse Qualitätsanforderungen für ein Softwareprodukt definiert. Dabei handelt es sich z.B. um Anforderungen an die Benutzungsoberfläche. Auch wenn die Funktionalität eines Buchhaltungsprogramms „stimmt“, kann eine wenig bedienungsfreundliche Oberfläche zur Inakzeptanz der Software führen.

(3) Einhaltung von Fertigstellungsterminen.
Bei einem konkreten Auftraggeber sind ohnehin festgelegte Termine für die Fertigstellung des Endprodukts oder auch von Teilprodukten einzuhalten. Doch auch bei einer Produktion für den anonymen Markt stellt die Entwicklungsdauer einen kritischen Erfolgsfaktor – weil Wettbewerbsfaktor – dar, weshalb in diesem Fall Fertigstellungstermine mit dem Vertrieb vereinbart werden.

(4) Einhaltung des vorgesehenen Kostenlimits.
Die Kosten einer Entwicklung sind ein entscheidender Faktor ihrer Effizienz und sollten das geplante Budget eines Softwareprojekts nicht überschreiten.

Abhängigkeiten zwischen den grundlegenden Anforderungen

Offensichtlich bestehen zwischen den beschriebenen grundlegenden Anforderungen Interdependenzen. Allerdings vermag erst die Summe aller vier Anforderungen Druck auf ein Softwareprojekt auszuüben. Spielten z.B. Entwicklungsdauer und Kosten keine Rolle, so könnte ein Softwareprodukt mit beliebig großem Funktionsumfang und beliebigen Qualitätseigenschaften erstellt werden. Sind dagegen alle vier Anforderungen simultan zu berücksichtigen, so „drückt“ je ein Faktor auf die übrigen drei. Bei einer gegebenen Produktivität des Projektteams können eine oder mehrere Anforderungen im Allgemeinen nicht erhöht werden, ohne dass die übrigen Anforderungen gelockert oder reduziert werden.

Wird z.B. der geforderte Funktionsumfang erhöht, dann gestaltet sich die Einhaltung von Terminen schwieriger, wenn gleichbleibende Qualitätsanforderungen und ein unverändertes Projektbudget unterstellt werden. Der Zusammenhang der grundlegenden Anforderungen der Softwareentwicklung wird in der Literatur unter dem Stichwort „Teufelsquadrat“ (vgl. SNEED 1987) behandelt und in Abb. 1.1 veranschaulicht.

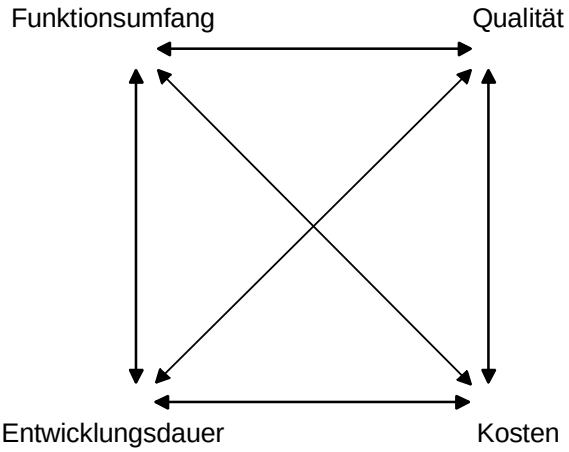


Abb. 1.1. Interdependenzen grundlegender Anforderungen bei der Softwareentwicklung.

steigende Anforderungen an Software

Mit den rasanten Fortschritten der Computertechnologie wie auch des Software Engineering sind die Anforderungen an Funktionalität und Qualität von Softwareprodukten stark gewachsen. Die folgenden Fakten, Zahlen und Tendenzen mögen hiervon eine Vorstellung vermitteln; sie wurden BALZERT (1996) entnommen, sofern keine anderen Quellen genannt werden.

(1) Umfang und Komplexität von Problemstellungen und Softwaresystemen

Umfang und Schwierigkeit von Problemstellungen und ...

- Software wird zur Lösung immer umfangreicherer und schwierigerer Aufgaben in immer neuen Anwendungsgebieten eingesetzt. Sie ist heute essentieller Bestandteil vieler technischer Produkte – z.B. Antiblockiersysteme in Automobilen – und wird zur Ausführung der meisten Dienstleistungen eingesetzt. Bis zu 80% der Entwicklungskosten für Anlagen der digitalen Vermittlungstechnik werden bei deutschen Anbietern für die Softwareentwicklung aufgewendet. Dienstleistungen im Bank- und Versicherungsgewerbe werden fast durchweg mit Hilfe von Softwaresystemen abgewickelt. Die Nachfrage nach Software steigt jährlich um 10-25%.

Softwaresystemen

- Je umfangreicher und schwieriger die Problemstellungen, desto umfangreicher und komplexer die Softwaresysteme zu ihrer Lösung. Bereits in den 1980er Jahren bestand ein großes Softwaresystem aus mindestens 50000 Zeilen Quellcode (vgl. FAIRLEY 1985, YOURDON 1988). Schätzungen zufolge wächst alle 5 Jahre der durchschnittliche Softwareumfang um eine Größenordnung.

Dimensionen der Komplexität

- Die Komplexität von Softwaresystemen besitzt verschiedene Dimensionen. Für betriebliche Anwendungssysteme sind vor allem folgende Komplexitätsdimensionen relevant:
  - Funktionale Komplexität; die funktionale Komplexität steigt mit der Anzahl der dem Benutzer angebotenen Systemfunktionen.
  - Datenkomplexität; die Datenkomplexität wächst mit der Anzahl und dem Umfang der vom System zu verwaltenden Datenstrukturen.
  - Oberflächenkomplexität; die Komplexität der Benutzungsoberfläche hängt u.a. von der Dialoggestaltung ab.
  - Algorithmische Komplexität; eine hohe algorithmische Komplexität kann z.B. bei umfangreichen und verschachtelten Fallunterscheidungen oder bei schwierigen numerischen Berechnungen vorliegen.

Beispiele zur Komplexität von Softwaresystemen

Die Tab. 1.1 charakterisiert einerseits die Komplexität üblicher betrieblich-kommerzieller Anwendungen, die etwa bei Banken und Versicherungen eingesetzt werden und andererseits die Komplexität des Standard-Anwendungssystems R/3 der SAP AG. Dabei wird für die erstgenannten Systeme von einer objektorientierten Entwicklung ausgegangen; zur Bedeutung der hier verwendeten Kennzahlen sei auch auf die Kapitel 3 bis 5 verwiesen. Das System R/3 unterstützt den gesamten Wertschöpfungsprozess eines Unternehmens – vom Einkauf bis zum Vertrieb – und stellt daher ein extrem komplexes System dar.

Kennzahlen zur Komplexität üblicher betrieblich-kommerzieller Anwendungen bei objektorientierter Erstellung (vgl. OESTEREICH 1998)	**Kennzahlen zur Komplexität des Systems R/3 der SAP AG (zit. nach BALZERT 1996)**
- 300 – 600 Einträge im Fachlexikon, d.h. relevante Begriffe des Anwendungsgebietes - 20 – 100 Anwendungsfälle, d.h. komplexere, mit dem System bearbeitbare Aufgaben mit wahrnehmbarem Ergebnis, z.B. Reisekostenabrechnung - 5 – 20 Aktivitäten, d.h. durchzuführende Teilschritte pro Anwendungsfall - 100 – 500 Fachklassen, d.h. rein anwendungsbezogene Programmbausteine - ca. 12 nicht triviale Operationen, d.h. zu programmierende Funktionen pro Klasse	- ca. 20000 unterschiedliche Systemfunktionen - ca. 21000 Reports (Listen) - ca. 17000 Menüleisten, - ca. 14000 Funktionsbausteine - ca. 150000 Funktionsaufrufe - ca. 7000000 Zeilen Quellcode

Tab. 1.1. Kennzahlen zur Komplexität betrieblich-kommerzieller Anwendungen sowie des Standard-Anwendungssystems R/3.

(2) Qualitätsanforderungen der Softwarebenutzer

Qualitätsanforderungen

- Mit der Ausdehnung der Anwendungsgebiete und der Komplexität der Problemstellungen wachsen auch die Qualitätsanforderungen an Softwareprodukte.

Zuverlässigkeit

- Dies betrifft vor allem Zuverlässigkeitsforderungen. Der Einsatz von Software in sensiblen Bereichen (z.B. Medizin, Kraftwerkssteuerung) erfordert Null-Fehler-Software, um Gefahren für Gesundheit und Leben betroffener Menschen auszuschließen. Doch auch, wenn Softwarefehler „nur“ Sachschäden oder ökonomische Verluste zur Folge haben können, verlangt der heutige Software-Einsatz häufig zwingend fehlerfreie Software. Eine Fehlerrate (Anzahl Fehler auf 1000 Zeilen Quellcode) von nur 0.1% würde pro Tag zu mehr als 10.000 verlorenen Briefen bei der Post und pro Stunde zu mehr als 20.000 nicht korrekt gebuchten Schecks führen.

komfortable Bedienung

- Eine weitere wesentliche Qualitätsanforderung stellt heute die leicht erlernbare und komfortable Bedienung der vorwiegend interaktiv, d.h. im Dialogbetrieb eingesetzten Software dar. Auch diese Forderung resultiert aus der inzwischen erreichten Intensität des Software-Einsatzes. Komfortable Benutzungsoberflächen sind einerseits produktivitätsfördernd und stellen andererseits eine Mindestbedingung menschengerechter Arbeitsplätze dar.

(3) Anforderungen an Portabilität und Änderbarkeit

Lebenszeit von Software und Anforderungen an ...

- Mit Umfang und Komplexität von Anwendungssoftware verlängern sich sowohl ihre Entwicklungsdauer wie auch ihre Einsatzdauer, die auch als Lebenszeit bezeichnet wird. Die Lebenszeit von Anwendungssoftware beträgt derzeit üblicherweise zwischen 10 und 15 Jahren. Dem stehen weitaus kürzere Lebenszeiten der Systemsoftware (v.a. Betriebssysteme) von etwa 6 Jahren und erst recht der Computerhardware von etwa 3 Jahren gegenüber.

Portabilität und ...

- Daher sind einerseits hohe Portabilitätsanforderungen einzuhalten. Unter Portabilität wird dabei die Übertragbarkeit von Softwaresystemen auf alternative Systemumgebungen (Hardware und/oder Systemsoftware) verstanden. Hohe Portabilitätsanforderungen ergeben sich ebenfalls aus dem Erfordernis, Auftraggeber bzw. Kunden mit verschiedenen Systemumgebungen bedienen zu können.

Änderungsfreundlichkeit

- Aufgrund der langen Entwicklungs- und Lebenszeiten müssen Anwendungssysteme ferner leicht an sich ständig verändernde Produktanforderungen anpassbar, d.h. leicht änderbar sein. Es ist nicht ungewöhnlich, dass ein Anwendungssystem im Ergebnis vielfacher Änderungen schließlich keine Zeile seines Originalcodes mehr besitzt (vgl. PAGEL und SIX 1994).

Der wachsenden Nachfrage und den gewachsenen Anforderungen an Softwareprodukte stehen verschiedene gravierende Probleme und Schwierigkeiten bei der Softwareentwicklung gegenüber, die anschließend umrissen werden.

(4) Mangelnde Beherrschung der Problemkomplexität und des Entwicklungsprozesses

mangelnde Komplexitätsbeherrschung

- In vielen Fällen werden Aufgabenstellungen hoher Komplexität nicht oder nur mangelhaft bewältigt. Dies betrifft vor allem Problemstellungen, die kreative Lösungskonzepte erfordern oder für die in einem Unternehmen oder Projektteam bislang nur unzureichende Erfahrungen vorliegen.
- Die mangelnde Beherrschung der Komplexität von Problemstellungen sowie des Entwicklungsprozesses drückt sich u.a. aus in erfolglos abgebrochenen Projekten oder Produkten, die am Markt bzw. an den Auftraggeberanforderungen vorbei entwickelt werden und nicht zum Einsatz kommen.
- Nach JONES (1981) werden nur 70-80% aller Projekte erfolgreich abgeschlossen. Zehn Jahre später kamen DEMARCO und LISTER (1991) zu ähnlichen Ergebnissen: 15% der von ihnen untersuchten Entwicklungsvorhaben schlugen fehl; bei größeren Vorhaben mit einem Umfang von mehr als 25 Personenjahren wurden sogar 25% nicht fertiggestellt.

(5) Fehleranfälligkeit von Software

Fehlerhäufigkeit

- Untersuchungen der 1980er Jahre besagen, dass zur damaligen Zeit ausgelieferte Software noch etwa 4 Fehler auf 1000 Quellcodezeilen aufwies (vgl. LIPOW 1982, HAMILTON 1986).
- Einer zu Beginn der 1990er Jahre angestellten Untersuchung zufolge konnte im Zeitraum 1977 bis 1990 die Software-Fehlerrate ca. um das Hundertfache reduziert werden.

- Dennoch werden die oben begründeten Zuverlässigkeitsanforderungen weiterhin nicht erreicht. Schätzungen besagen, dass jeder zweite Ausfall im industriellen Sektor durch Software-Fehler verursacht wird.

(6) Mangelnde Produktivität, überhöhte Kosten, wachsende Nachfragelücke

dominierende Softwarekosten

- Während in den Anfangszeiten der Datenverarbeitung die Computerkosten überwogen, betrug bereits 1976 der Anteil der Softwarekosten an den Gesamtkosten kompletter Datenverarbeitungssysteme („Computer + Software“) über 70%, während die Hardwarekosten unter 30% lagen. Da sich dieser Trend seither fortgesetzt hat, sind heute die Softwarekosten gegenüber den Hardwarekosten absolut dominierend (vgl. PAGEL und SIX 1994).
- Softwarekosten sind wiederum im Wesentlichen Personalkosten. Ihre Reduzierung hängt bei hohen und wachsenden Preisen für Entwicklerstunden maßgeblich von der Steigerung der Produktivität ab (vgl. SOMMERVILLE 1992).

mangelnde Produktivität

- Nach BOEHM (1981) produziert jede an einer Entwicklung beteiligte Person gemittelt über die gesamte Entwicklungsdauer weniger als 10 Zeilen ausführbaren Code pro Tag.

lange Entwicklungszeiten

- Wie schon erwähnt, wachsen die Entwicklungszeiten mit Umfang und Komplexität der Softwaresysteme. Die durchschnittliche Entwicklungsdauer für betriebliche kommerzielle Anwendungen betrug 1980 ca. 1 Jahr, dagegen 1990 3,5 Jahre.
- Hohe Produktivitätsschwankungen zwischen Firmen und Projekten weisen auf erhebliche Produktivitätsreserven hin (vgl. DEMARCO und LISTER 1987).

überhöhte Kosten, Nachfragestau

- Insgesamt reichen die erzielten Produktivitätsfortschritte und der Personalzuwachs nicht aus, um die überhöhten Softwarekosten zu reduzieren und die wachsende Nachfragelücke zu schließen. Hieran haben auch die zunehmende Außerhaus-Entwicklung (Outsourcing) sowie der wachsende Einsatz von Standardsoftware bisher wenig ändern können.

Bei der Softwareentwicklung treten teils ähnliche Schwierigkeiten auf wie bei der Entwicklung anderer komplexer technischer Produkte. Dies betrifft etwa die Finanzierung von Projekten oder die Gewinnung geeigneter Mitarbeiter für Entwicklungsvorhaben. Doch lassen sich die oben beschriebenen Probleme teilweise auch auf Besonderheiten der Produktgattung Software und der Softwareentwicklung zurückführen.

1.1.2 Besonderheiten von Software und Softwareentwicklung

essentielle Software-Merkmale

Zwei wesentliche und eng zusammenhängende Merkmale von Software sind

- ihre immaterielle Natur und
- ihr Abbildcharakter.

Als immaterielles Produkt entzieht sich Software der sinnlichen Wahrnehmung, man kann sie nicht „anfassen“. Zugleich besitzt Software Abbildcharakter, d.h. ein Softwareprodukt stellt ein spezifisches Abbild eines Problems oder eines gewissen Ausschnitts der Realität dar. Dementsprechend geht es bei der Software-

entwicklung stets „um die Abbildung eines Problems der realen Welt in ein lauffähiges Programm“ (PAGEL und SIX 1994).

Bei spezialisierten Anwendungen, z.B. einem Programm zur Erzeugung von Schnittmustern für das Zuschneiden von Blechen, leuchtet es ohne weiteres ein, dass Software Abbild einer Problemstellung bzw. eines Realitätsausschnitts ist. Dies trifft indessen auch für weniger spezialisierte Anwendungen wie z.B. Textverarbeitungsprogramme zu. Zu beachten ist in diesem Fall, dass die abgebildete Problemstellung selbst allgemeinerer Art ist. Software muss sich auf den jeweiligen besonderen Gegenstand der Informationsverarbeitung beziehen, um ihrer Funktion, nämlich der automatisierten computergestützten Informationsverarbeitung, gerecht zu werden. In diesem Sinne besitzt Software stets Abbildcharakter, der zweifellos komplizierter geartet ist als etwa der einer Fotografie.

Aus den besprochenen Software-Wesensmerkmalen – dem immateriellen und Abbildcharakter – ergeben sich weitere Merkmale von Software und teils gravierende Auswirkungen auf die Softwareentwicklung:

weitere Merkmale von Software und Softwareentwicklung

- Software ist unanschaulich bzw. schwer zu veranschaulichen. Dies erschwert die Entwicklung, da z.B. vergessene oder nicht zusammenpassende Komponenten nicht ohne weiteres auffallen.
- Software bildet reale Problemstellungen in abstrakter Weise, zugeschnitten auf das jeweils Wesentliche ab. Dies erfordert ein hohes Abstraktionsvermögen der Entwickler.
- Für die grundlegende Strukturgebung von Softwaresystemen eröffnen sich weite Spielräume, weil physikalische Gesetze nicht einschränkend wirken. Die Softwareentwicklung verlangt daher unter Umständen ein erhebliches Maß an Kreativität.
- Umfangreiche Softwaresysteme weisen eine derartige Komplexität auf, dass sie von einzelnen Entwicklern nicht mehr überblickt werden können (vgl. BOOCH 1994).
- Qualitätsmerkmale sind schlecht zu messen. Eine relativ einfache, schematische Prüfung von Kenngrößen, die wie etwa bei elektronischen Geräten als präziser Qualitätsmaßstab dienen, ist kaum möglich. Dies behindert die Entwicklung und insbesondere auch das Management von Softwareprojekten. Das Management wird ebenso dadurch erschwert, dass Entwicklungsfortschritte schwer abzuschätzen und nachzuweisen sind.
- Software ist im Verhältnis zu anderen technischen Produkten leicht änderbar. Dies erweckt die „Begierde“ des Auftraggebers nach immer neuen Modifikationen und Erweiterungen.
- Software unterliegt keinem physischen Verschleiß. Sie altert dennoch, da sich ihre Einsatzumgebung ständig weiterentwickelt. Hieraus resultierende Konsequenzen hinsichtlich Änderbarkeit und Portabilität wurden bereits erwähnt.

Bedarf nach methodischer Unterstützung

Die beschriebenen Anforderungen und Probleme sowie die Eigenarten von Software und ihrer Herstellung sollten verdeutlicht haben, dass die Softwareentwicklung allerdings einen Bedarf nach methodischer Fundierung besitzt. Dieser machte sich bereits in den 1960er Jahren bemerkbar, als viele der oben angesprochenen Probleme erstmals in großem Maßstab auftraten. Die bei der Softwareentwicklung auftretenden Schwierigkeiten wurden damals unter dem Begriff „Softwarekrise“

subsumiert und führten zur Herausbildung des Software Engineering, das nun näher zu charakterisieren ist.

Übungsaufgaben zu Kapitel 1.1

Übungsaufgabe 1.1.1

Bestimmen Sie für die folgenden Softwaresysteme jeweils, ob es sich um System- oder Anwendungssoftware handelt. Entscheiden Sie im Falle von Anwendungssoftware außerdem, ob ein betriebliches Anwendungssystem vorliegt.

a) MS Windows.
b) CAD-System zum Entwurf von Wohnhäusern.
c) Unix.
d) System zur Unterstützung von Bankmitarbeitern bei der Kreditbearbeitung.
e) MS Visual C++.
f) System zur Wetterprognose.
g) System zur Bildung optimaler Produktionslose für gleiche Teile auf der Basis eines vorab ermittelten Teile-Nettobedarfs.
h) Datenbanksystem Oracle.
i) MS Word.

Übungsaufgabe 1.1.2

Entscheiden Sie für die folgenden Aufgabenstellungen, ob eine Systementwicklung oder eine Softwareentwicklung durchzuführen ist.

a) Entwicklung eines ergänzenden Softwaresystems zur statistischen Auswertung der Auftragsabwicklung eines Handelsunternehmens. Das System soll auf dem vorhandenen PC-Netzwerk zum Einsatz kommen und durch Mitarbeiter der Auftragserfassung bedient werden.
b) Entwicklung eines Systems für den Betrieb eines Gewächshauses. Größen wie Temperatur und Luftfeuchtigkeit sollen mittels zu beschaffender Sensoren laufend gemessen und über eine angeschlossene Software automatisch reguliert werden.

Übungsaufgabe 1.1.3

Nennen Sie zusätzlich zu den im Kap. 1.1.1 genannten Dimensionen der Komplexität eine weitere Komplexitätsdimension von Software und beschreiben Sie diese kurz. Denken Sie zum Beispiel an ein Betriebssystem zur Unterstützung eines PC-Netzwerkes.

Übungsaufgabe 1.1.4

Nennen Sie für die folgenden Systemarten je eine Komplexitätsdimension, die in hohem Grade vorhanden ist.

a) Datenbanksysteme.
b) Textverarbeitungssysteme.

c) Systeme zur Durchführung statischer Berechnungen.
d) Systeme zur Wetterprognose.
e) Tabellenkalkulationssysteme.
f) Systeme für die Kraftwerkssteuerung.
g) Grafiksysteme (z.B. Designer).

Übungsaufgabe 1.1.5

Die verhältnismäßig leichte Änderbarkeit von Software wurde als ein allgemeines Softwaremerkmal ausgewiesen (s. Kap. 1.1.2). Zuvor wurde jedoch festgestellt, dass die Anforderungen an die Änderbarkeit von Software zunehmen (s. Kap. 1.1.1). Liegt hier ein Widerspruch vor?

Übungsaufgabe 1.1.6

Nennen Sie drei möglichst konkrete Beispiele für eine Änderung der Einsatzumgebung betrieblicher Anwendungssysteme und die erforderlichen Änderungen bzw. Portierungen der Systeme.

1.2 Software Engineering

Software Engineering

In der Literatur finden sich verschiedene Begriffsbestimmungen des Software Engineering (deutsch: Software-Technik, auch Software-Technologie). Die beiden folgenden Definitionen seien zitiert:

Definition und ...

- Software Engineering ist “The systematic approach to the development, operation, maintenance, and requirement of software” (ANSI, IEEE Standard 1983).
- Software Engineering ist das „Fachgebiet der Informatik, das sich mit der Bereitstellung und systematischen Verwendung von Methoden und Werkzeugen für die Herstellung und Anwendung von Software beschäftigt“ (HESSE et al. 1984).

Charakterisierung

Beide Begriffsbestimmungen erfassen wesentliche Seiten und Merkmale des Software Engineering. Ausführlicher lässt sich dieses Teilgebiet der Informatik und der Wirtschaftsinformatik wie folgt charakterisieren:

Ziele

- Das Software Engineering zielt auf die erfolgreiche Bewältigung der in Kap. 1.1.1 besprochenen vier grundlegenden Anforderungen der Softwareherstellung ab. Software soll mit vorgesehener Funktionalität und Qualität, zugleich kostengünstig und termingemäß entwickelt werden.

Ansatz

- Hierzu soll eine ingenieurmäßig-systematische an die Stelle einer spontan-„künstlerischen“ Vorgehensweise bei der Softwareentwicklung treten. Obgleich in der Frühzeit industrieller Softwareherstellung besonders dringlich, hat dieses Anliegen bis heute nicht an Aktualität eingebüßt. Um eine systematische Softwareentwicklung zu unterstützen, werden zum Teil bewährte Methoden klassischer Ingenieurwissenschaften adaptiert.

Hauptgegenstand

- Im Mittelpunkt steht die Beherrschung der Komplexität umfangreicher Problemstellungen. Dies erfordert in erster Linie die konzeptionell-methodische Unterstützung der Analyse bzw. Aufbereitung komplexer Problemstellungen sowie

des Entwurfs der zu ihrer Bearbeitung dienenden Software. Im Ergebnis von Analyse und Entwurf liegt ein Modell des zu erstellenden Softwareprodukts vor. Dieses wird als ein Softwaresystem mit hinreichend kleinen und überschaubaren Systemkomponenten spezifiziert. Die anschließende Implementierung der Systemkomponenten ist mit weitaus geringeren Schwierigkeiten konfrontiert als die genannten Aktivitäten vor der eigentlichen Programmierung. Das Software Engineering befasst sich daher vorrangig mit der Analyse komplexer Problemstellungen und dem Entwurf von Softwaresystemen.

- Ist die methodische Fundierung der Softwareentwicklung im engeren Sinne primär, so wird zugleich das Management der Softwareentwicklung unterstützt. Zu nennen sind hier insbesondere Prozessmodelle zur Planung und Leitung des Prozesses der Softwareentwicklung.

Instrumente

- Zur Unterstützung von Entwicklung und Management werden verschiedene, aufeinander abgestimmte Instrumente angeboten. Hierzu zählen neben den erwähnten Prozessmodellen vor allem (Entwicklungs-)Methoden, die ihrerseits auf Konzepten und Notationen basieren, sowie technische Werkzeuge zur Softwareentwicklung (vgl. Abb. 1.2).

Paradigmen der Softwareentwicklung

Mit dem strukturierten und dem objektorientierten Ansatz (kurz Objektorientierung) liegen heute zwei grundlegende Ansätze der Softwareentwicklung vor. Diese werden auch als Paradigmen, d.h. grundlegende Denkmuster der Softwareentwicklung bezeichnet. Beide Paradigmen werden später charakterisiert und gegeneinander abgegrenzt. Trotz tiefgreifender Unterschiede bestehen zwischen beiden Ansätzen auch grundlegende Gemeinsamkeiten. In diesem Sinne bemüht sich das vorliegende Kapitel um eine paradigmenübergreifende Einführung der Grundbegriffe des Software Engineering.

Als Teilgebiet der Informatik und der Wirtschaftsinformatik weist das Software Engineering Beziehungen zu anderen Teildisziplinen der Informatik wie auch zu anderen Wissenschaften auf. So werden natürlich bei der Entwicklung anspruchsvoller Algorithmen und Datenstrukturen Erkenntnisse des gleichnamigen Gebiets der Informatik genutzt. Auf die Beziehungen zu den Ingenieurwissenschaften wurde bereits hingewiesen. Bei der theoretischen Fundierung der Konstruktion von Benutzungsoberflächen wird z.B. auf Erkenntnisse der Psychologie zurückgegriffen.

Das Software Engineering fächert sich heute auf in die Teilgebiete:

Teilgebiete

- Entwicklung
- Management und
- Qualitätssicherung.

Die weiteren Kapitel des Kurses behandeln im Wesentlichen die objektorientierte Softwareentwicklung. Knappe Einführungen in das Projektmanagement und die Qualitätssicherung werden am Ende dieses Kapitels gegeben.

Im Folgenden werden zunächst die grundlegenden Aktivitäten der Softwareentwicklung charakterisiert, bevor die oben angesprochenen und in Abb. 1.2 dargestellten Instrumente des Software Engineering sowie ihre Wechselbeziehungen näher betrachtet werden.

Zusammenhang der Instrumente

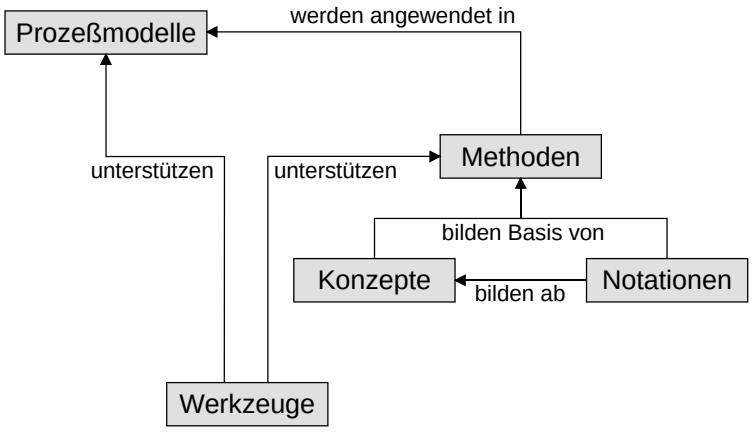


Abb. 1.2. Instrumente des Software Engineering.

Übungsaufgaben zu Kapitel 1.2

Übungsaufgabe 1.2.1

Zu Beginn des Kap. 1.2 wurden zwei Definitionen des Software Engineering zitiert. Umreißen Sie knapp die unterschiedlichen Schwerpunkte beider Definitionen.

Übungsaufgabe 1.2.2

Nennen Sie zwei Beispiele für algorithmische Teilaufgaben, die immer wieder und in vielfältigen Varianten innerhalb betrieblicher Anwendungssysteme zu lösen sind und bei denen man auf grundlegende Algorithmen der Informatik zurückgreift.

1.3 Grundlegende Aktivitäten der Softwareentwicklung

grundlegende Entwicklungsaktivitäten

In der Frühzeit der Softwareentwicklung wurden verschiedene Entwicklungsaktivitäten weder klar voneinander abgegrenzt noch in einer definierten Reihenfolge ausgeführt. Man programmierte stattdessen mehr oder minder drauflos und behob anschließend die Fehler. Später erkannte man, dass vor der Implementierung eines Softwaresystems zunächst die Problemstellung analysiert und ein Entwurf des Systems erstellt werden muss. Weitere grundlegende Entwicklungsaktivitäten stellen der Test sowie die Wartung von Softwaresystemen dar. Im Folgenden werden die genannten Aktivitäten charakterisiert.

(1) Analyse

Auftrag

Die Entwicklung eines neuen Softwareprodukts wird durch einen Auftrag initiiert, mit dem der Auftraggeber den Bedarf nach einer Softwarelösung für ein spezifisches Problem grob umreißt.

Aufgabe und Ergebnis

Aufgabe der Analyse ist die Festlegung der Gesamtheit der Anforderungen an das zu entwickelnde Softwaresystem. Die entsprechende Tätigkeit wird auch als Sys-

temanalyse, ihr Ergebnis als Anforderungsdefinition oder Produktdefinition bezeichnet.

Gegenstand

Die Anforderungsdefinition beschreibt ein Anwendungssystem (kurz auch Anwendung) aus der Sicht des Auftraggebers. Definiert wird mit anderen Worten das Außenverhalten des gewünschten Softwaresystems. Im Vordergrund steht daher, was das System leisten soll, nicht wie die geforderten Leistungen zu realisieren sind. Dies schließt ein, dass während der Analysephase hardware- und softwaretechnische Bedingungen der Realisierung wie etwa die Zielplattform nur eine untergeordnete Rolle spielen.

Bedeutung der Analyse

Weil die Analyse mit der Anforderungsdefinition das Fundament für die spätere software-technische Realisierung legt, ist sie von herausragender Bedeutung im Software-Entwicklungsprozess. Fehler zu Beginn des Entwicklungsprozesses wirken sich besonders gravierend auf die gesamte Entwicklung aus. Ihre Behebung kann sich vom Arbeitsumfang und den Kosten her extrem aufwendig gestalten. Insbesondere kann das entwickelte System nur dann die Intentionen des Auftraggebers adäquat widerspiegeln, wenn die Anforderungsdefinition dies leistet.

Hieraus ist zunächst die allgemeine Schlussfolgerung zu ziehen, dass während der Analyse mit besonderer Sorgfalt und nicht überhastet vorgegangen werden sollte. Erst wenn die zu bearbeitende Problemstellung aus fachlicher Sicht hinreichend genau verstanden wurde, sollte zum Entwurf übergegangen werden.

Um ferner den tatsächlichen Anforderungen und Wünschen des Auftraggebers, fachlicher Experten und der späteren Benutzer der Anwendung gerecht werden zu können, sind diese soweit wie irgend möglich in die verschiedenen Schritte der Analyse einzubeziehen. Hierin besteht ein Grundsatz der erfolgreichen Gestaltung der Analyse, auf dessen geeignete Umsetzung noch einzugehen ist (vgl. Kap. 1.4).

Teilaktivitäten

Zur Analyse gehören hauptsächlich zwei Teilaktivitäten, die sich methodisch deutlich voneinander abheben und zu verschiedenen Ergebnisdokumenten führen.

Pflichtenheft erstellen

In dem ersten Schritt der Analyse sind die verschiedenartigen Anforderungen an die Anwendung systematisch zu ermitteln und geeignet zu formulieren. Methoden für die Ermittlung von Anforderungen sind zum Beispiel das Dokumentenstudium oder Interviews. Die ermittelten Anforderungen werden überwiegend textuell beschrieben. Das Ergebnisdokument dieser Teilaktivität ist das sogenannte Pflichtenheft. Die im Pflichtenheft fixierten Anforderungen werden mit dem Auftraggeber meist verbindlich vereinbart. Daher wird die Anwendung bei der Abnahme anhand des Pflichtenheftes geprüft.

Fachliches Modell

Der zweite Schritt der Analyse umfasst die Modellierung der Anwendung aus fachlicher Perspektive und führt zum fachlichen Modell der Anwendung. Dieses wird auch als Analysemodell oder kurz als Fachkonzept bezeichnet.

Methode

Die zur fachlichen Modellierung verwendete Methode hängt u.a. von dem zugrunde liegenden Entwicklungsparadigma ab und benutzt meist eine formale oder semiformale Notation (vgl. Kap. 1.5). Das resultierende Analysemodell wird großenteils grafisch mit Hilfe von Diagrammen dargestellt. Betont sei nochmals, dass die fachliche Modellierung quasi von einer perfekten Technik ausgeht und z.B. Speicherplatz- oder Geschwindigkeitsbeschränkungen im Allgemeinen vernachlässigt.

Sichten der Analyse

Das Analysemodell betrachtet die Anwendung aus verschiedenen Sichten, die auch für den Entwurf (siehe unten) relevant sind:

- Die statische Sicht betrachtet die zeitunabhängigen und strukturellen Aspekte eines Systems und kann in die Daten- und Funktionssicht gegliedert werden.
- Die Datensicht beschäftigt sich mit den vom System zu verarbeitenden Daten und ihren Beziehungen.
- Die Funktionssicht stellt die einzelnen Teilfunktionen bzw. Teilaufgaben der Anwendung und die zu ihrer Umsetzung erforderlichen Algorithmen dar.
- Die dynamische Sicht berücksichtigt die zeitabhängigen Aspekte eines Systems. Differenzieren lassen sich hierbei noch die Ablauf- und die Zustandssicht.
- Die Ablaufsicht stellt zeitliche Abläufe im System dar. Betrachtet wird insbesondere das Zusammenwirken verschiedener Systemfunktionen.
- Die Zustandssicht beschäftigt sich mit dem zustandsabhängigen Verhalten der Anwendung während der Systembenutzung.
- Die Oberflächensicht betrachtet die Benutzungsoberfläche der Anwendung.

warum fachliche Modellierung?

Zur Begründung des zweiten Schrittes der Analyse sei bemerkt, dass im Pflichtenheft die Anforderungen an die Anwendung noch auf einem hohen Abstraktionsniveau dargestellt werden. Erst eine darüber hinausgehende Modellierung der Anwendung erschließt die Anforderungen in allen Details und führt zu einem hinreichenden fachlichen Verständnis.

Beispiel 1.1

Bereits im Pflichtenheft werden die von der Anwendung zu verwaltenden Daten beschrieben. Jedoch geschieht dies noch recht grob. So wird es z.B. meist genügen, Kundenadressen als ungegliedertes Datum zu betrachten. Die fachliche Modellierung muss hingegen nach den Adressbestandteilen fragen. Dabei können die Modellierer auf den Umstand stoßen, dass es ganz verschiedene Arten von Adressen gibt. So z.B. Auslands- und Inlandsanschriften, ferner Großkunden-, Postfach- und Straßenanschriften. Alle für die Anwendung relevanten Adresstypen müssen in der Modellierung berücksichtigt werden.

Die bei dem Auftraggeber, fachlichen Experten und späteren Systembenutzern ermittelten Systemanforderungen sind oft noch fragmentarisch, verschwommen, situationsbezogen und widersprüchlich. Häufig entwickeln sich die Anforderungen auch erst im Verlauf der Analyse. Gerade bei der Entwicklung größerer Systeme gewährleistet allein eine Modellierung der Anwendung, dass letztlich eine Anforderungsdefinition erstellt wird, die folgende fundamentalen Merkmale besitzt:

grundlegende Merkmale der Anforderungsdefinition

- Vollständigkeit, d.h. die Anforderungen an das System sind komplett definiert.
- Eindeutigkeit, d.h. jede Anforderung ist eindeutig definiert, besitzt also möglichst keinen Interpretationsspielraum mehr.
- Konsistenz, d.h. die verschiedenen Anforderungen sind widerspruchsfrei, also miteinander vereinbar.
- Intentionstreue, d.h. die festgelegten Anforderungen entsprechen den tatsächlichen Bedürfnissen und Wünschen des Auftraggebers.

(2) Entwurf

Entwurf

Aufgabe und Ergebnis

Der Entwurf spezifiziert auf der Grundlage der Anforderungsdefinition die software-technische Realisierung der Anwendung. Das Ergebnisdokument des Entwurfs wird als Softwarespezifikation, Produktspezifikation oder Produktentwurf bezeichnet.

Architektur erstellen

Mit dem Entwurf wird die Architektur der Anwendung festgelegt. Die Architektur einer Anwendung stellt ihre innere Struktur, d.h. ihre Gliederung in Systemkomponenten dar, wobei jede Komponente eine festgelegte Teilaufgabe übernimmt. PAGEL und SIX (1994) charakterisieren den Entwurf wie folgt: „Leitthema des Entwurfs ist die Bewältigung der Komplexität durch Zerlegung des Gesamtprodukts in kleine beherrschbare Einheiten“. Von der Qualität der Anwendungsarchitektur hängt zum einen die Änderbarkeit bzw. Wartbarkeit der Anwendung entscheidend ab. Neben der Wartbarkeit ist auch die Effizienz der Anwendung ein Maßstab für den Entwurf.

Mit den Komponenten werden zugleich ihre Schnittstellen und ihr dynamisches Zusammenwirken definiert. Die Schnittstelle einer Softwarekomponente beschreibt, welche Leistungen (Daten, Operationen usw.) die betreffende Komponente für andere Komponenten bereitstellt, welche Leistungen anderer Komponenten benutzt werden und nach welchen Vereinbarungen und Regeln sich der Leistungsaustausch vollzieht. Analoges gilt für die Schnittstellen kompletter Softwaresysteme.

Beispiel 1.2

Schnittstellen von Funktionen und Prozeduren als besonders einfacher Systemkomponenten werden durch ihren Funktions- bzw. Prozedurkopf angegeben. Dieser besitzt bei Funktionen etwa folgenden Aufbau:

```
erg_typ funktionsname(par_1: Typ_1, par_2: Typ_2,..., par_n: Typ_n);
```

Dem Funktionsnamen folgt eine Parameterliste, die im Allgemeinen sowohl Eingabe- wie auch Ausgabeparameter umfasst. Ein Parameter wird durch seinen Namen und seinen Datentyp angegeben. Über Eingabeparameter werden der Funktion bei einer Ausführung Eingabewerte zur Verfügung gestellt. Ausgabeparameter werden zur Rückgabe ermittelter Ergebniswerte benutzt. Ein Parameter kann sowohl Eingabe- als auch Ausgabeparameter sein. Darüber hinaus gibt eine Funktion über ihren Namen einen Ergebniswert zurück, dessen Typ (erg_typ) hier vor dem Funktionsnamen notiert wird.

Prozeduren geben im Unterschied zu Funktionen keinen Wert über ihren Namen zurück. Da Prozeduren einen Sonderfall von Funktionen bilden, wird künftig im Allgemeinen nur noch von Funktionen gesprochen. Programmiersprachen, die das Konzept der Funktion bzw. Prozedur unterstützen (und nicht objektorientiert sind), werden allerdings als prozedurale Sprachen bezeichnet.

softwaretechnische Aspekte

Anders als bei der Analyse sind im Entwurf sämtliche software-technischen Aspekte der Realisierung festzulegen und in der Anwendungsarchitektur zu berücksichtigen. Hierzu zählen vor allem:

- die einzusetzende Zielplattform, d.h. die Hardware und das Betriebssystem,
- die zur Implementierung benutzte Programmiersprache,

- das für die Erstellung der Benutzungsoberfläche zu verwendende Graphical-User-Interface-System (kurz GUI-System, vgl. Kap. 5),
- die gewünschte Variante der Datenhaltung bzw. -speicherung; benutzt werden können z.B. Datenbanken oder konventionelle Dateien,
- die ggf. vorgesehene Verteilung der Anwendung auf verschiedene Prozesse etwa im Rahmen einer Client-Server-Architektur,
- das Zusammenwirken mit anderen Softwaresystemen über definierte Schnittstellen,
- die Wiederverwendung von Softwarebausteinen, die z.B. aus entsprechenden Programmbibliotheken entnommen werden.

Methode

Die beim Entwurf anzuwendende Methode ist wiederum paradigmenabhängig und benutzt eine semiformale oder formale Notation (vgl. Kap. 1.5). Die resultierende Softwarespezifikation wird teils textuell, teils grafisch mittels Diagrammen erstellt.

Auch eine Softwarespezifikation stellt ein Modell der gewünschten Anwendung dar. Dieses bildet die Anwendung nun so detailliert ab, dass unmittelbar nach ihrem Entwurf zur Implementierung der einzelnen Systemkomponenten übergegangen werden kann.

(3) Implementierung

Implementierung

Mit der Implementierung werden die zuvor spezifizierten Komponenten der Anwendung in der gewählten Programmiersprache umgesetzt. Die Implementierung wird auch als Programmierung (im engeren Sinne) oder als Kodierung bezeichnet. Ergebnis der Implementierung sind die fertigen Systemkomponenten und schließlich die komplette Anwendung. Spätestens bei der Implementierung sollte auch eine Systemdokumentation sowie eine Benutzerdokumentation der Anwendung komplettiert bzw. angefertigt werden (vgl. Kap. 1.8).

Gegenstand

Den hauptsächlichen Gegenstand der Implementierung bilden die zu den verschiedenen Systemkomponenten gehörigen Operationen (Funktionen), die ggf. im Zuge der Implementierung noch detaillierter spezifiziert bzw. zerlegt werden. Hierbei kommt die Methode der strukturierten Programmierung zur Anwendung (vgl. Kap. 1.5).

(4) Test

Test, Testablauf

Bei dem Test der Anwendung wird quasi vom Teil zum Ganzen fortgeschritten. Getestet werden zunächst die einzelnen Systemkomponenten. Anschließend werden die Komponenten schrittweise zur fertigen Anwendung zusammengesetzt, wobei die jeweils bereits integrierten Komponenten in Integrationstests geprüft werden. An deren Ende steht der Systemtest, dem die gesamte Anwendung unterzogen wird. Nach der Installation der Anwendung beim Auftraggeber wird die Anwendung schließlich unter dessen „Regie“ einem Abnahmetest unterworfen.

Während sich die Komponenten- und Integrationstests vor allem auf die Funktionsfähigkeit der einzelnen Komponenten bzw. der integrierten Programmteile konzentrieren, wird bei dem System- und Abnahmetest die Einhaltung aller Anforderungen des Pflichtenheftes geprüft (vgl. Kap. 1.8.3).

(5) Wartung

Wartung

Nach dem erfolgreichen Abnahmetest beginnt der Einsatz des Softwaresystems beim Auftraggeber. Unter Wartung werden alle Modifikationen und Erweiterungen des Systems zusammengefasst, die vom Einsatzbeginn bis zum Ende der Systemlebenszeit vorgenommen werden.

Umfang

Typische Wartungsmaßnahmen sind die Behebung von Fehlern, die erst während der Einsatzzeit des Systems erkannt werden, die Portierung des Systems auf andere Plattformen bzw. Systemumgebungen, der Austausch einer veralteten Benutzungsoberfläche bei unveränderter Funktionalität sowie schließlich Änderungen oder Erweiterungen der Funktionalität des Systems.

Die Wartung ist kein Bestandteil der ursprünglichen Entwicklung des Softwareprodukts. Umfänglichere Wartungsaktivitäten werden stattdessen meist als separate Projekte abgewickelt, die wiederum die oben beschriebenen Aktivitäten einschließen.

Zusammenhang von Analyse, Entwurf und Implementierung

Nachdem die grundlegenden Entwicklungsaktivitäten für sich betrachtet wurden, sei nun noch näher auf den Zusammenhang von Analyse, Entwurf und Implementierung eingegangen.

Ausgangssituation

Die Ausgangssituation einer jeden Entwicklung lässt sich wie folgt kennzeichnen. Auf der einen Seite befinden sich ein Realweltausschnitt, eine erst grob umrissene Problemstellung und der Bedarf nach einer spezifischen Informationsverarbeitung, die gleichsam einen Problemraum aufspannen. Auf der anderen Seite befinden sich die verfügbare Systemumgebung mit ihrer Hardware und Systemsoftware, die – gemessen an der jeweiligen Problemstellung – nur elementare Fähigkeiten anbieten und den Lösungsraum konstituieren. Zwischen beiden Seiten klafft eine sogenannte „semantische Lücke“.

Analyse, Entwurf und Implementierung überbrücken gemeinsam diese semantische Lücke, wobei jede Aktivität ihre spezifische Rolle spielt.

Analyse – Abstraktion im Problemraum

Die Analyse erstellt ein fachliches Modell der künftigen Anwendung. Ihr Lebensnerv ist die Abstraktion. Abstrahieren bedeutet das Absehen oder Vernachlässigen von unwesentlichen oder zufälligen Merkmalen eines Gegenstandes, während seine wesentlichen Seiten und Merkmale herausgehoben werden. Das Wort Gegenstand ist dabei im allgemeinsten Sinne zu verstehen: er kann dinglich oder undinglich, realer oder gedanklicher Natur sein. Abstraktion und Modellbildung sind eng verwandt. Indem von unwesentlichen Zügen eines Gegenstandes abstrahiert wird, entsteht zugleich ein Modell, das gerade die wesentlichen Seiten des Gegenstandes repräsentiert. Das Wort Abstraktion beschreibt übrigens sowohl den Vorgang des Abstrahierens wie auch sein Resultat.

Ob ein Merkmal des Anwendungsbereiches bei der Abstraktion innerhalb der Systemanalyse als wesentlich oder unwesentlich zu bewerten ist, hängt von der Relevanz des Merkmals für die Anwendung ab, ist daher kontextabhängig und aus den Anforderungen an das Softwaresystem abzuleiten.

Beispiel 1.3

Sind Personen für ein Personalverwaltungssystem eines Unternehmens zu modellieren, dürfte die Augenfarbe kaum eine Rolle spielen. Handelt es sich dagegen um ein System zur Verwaltung von Personendaten für das polizeiliche Meldewesen, ist die Augenfarbe als im Personalausweis ausgewiesenes Merkmal allerdings relevant und daher als wesentliches Merkmal im Modell zu berücksichtigen.

Die bei der Erstellung eines fachlichen Modells der Anwendung vorgenommenen Abstraktionen beziehen sich stets auf den der künftigen Anwendung zugrunde liegenden Realitätsausschnitt. Die Systemanalyse findet daher vollständig im Problemraum statt.

Entwurf, Implementierung – Konkretisierung im Lösungsraum

Mit dem Entwurf wird dagegen die Grenze zum Lösungsraum überschritten. Bei den Aktivitäten Entwurf und Implementierung dominiert dementsprechend nicht die Abstraktion, sondern die Konkretisierung.

Konkretisieren bezeichnet das Gegenteil von Abstrahieren. Durch das Konkretisieren eines Gegenstandes wird dieser um zusätzliche Seiten und Merkmale ergänzt und bereichert. Der Gegenstand wird hierdurch wirklichkeitsnäher dargestellt oder auch – im Falle von Artefakten, d.h. künstlich geschaffenen Gegenständen – zunehmend ein wirklicher Gegenstand, wie etwa aus den Skizzen eines Malers ein fertiges Bild wird.

Natürlich wird der vorausgegangene Abstraktionsprozess der Analyse nicht rückgängig gemacht, was unsinnig wäre. Die nun durchzuführende Konkretisierung bezieht sich daher auch nicht auf die Problemstellung bzw. den Anwendungsbereich, sondern vielmehr auf die Systemumgebung, in die die Anwendung einzubetten ist und deren elementare Fähigkeiten es auszunutzen gilt. Entwurf und Implementierung finden mit anderen Worten im Lösungsraum statt. Sie restrukturieren und ergänzen zugleich das Analysemodell nach und nach um alle erforderlichen software-technischen Aspekte.

Dabei werden im Kern alle benutzernahen und komplexen Operationen und Datenstrukturen stufenweise auf elementare Operationen und Datenstrukturen zurückgeführt, die mittels der gewählten Programmiersprache ausgedrückt werden können, also unmittelbar durch die Systemumgebung unterstützt werden.

Damit ist die ursprüngliche semantische Lücke vollständig überbrückt, oder anders gesagt, der interessierende Realitätsausschnitt auf ein Softwaresystem abgebildet. Die Abb. 1.3 verbildlicht den dargelegten Zusammenhang von Analyse, Entwurf und Implementierung.

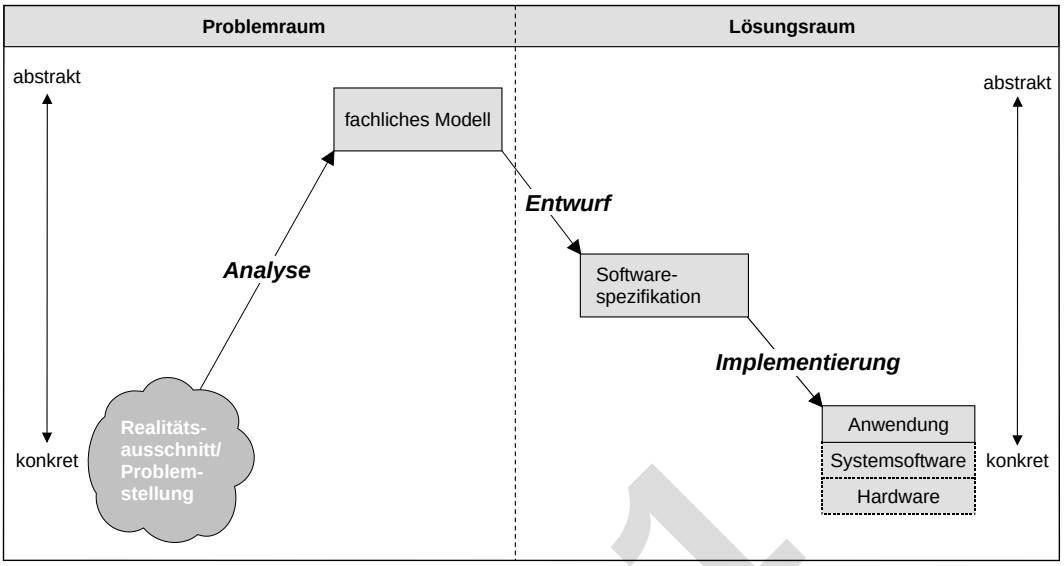


Abb. 1.3. Analyse, Entwurf und Implementierung im Zusammenhang.

Übungsaufgaben zu Kapitel 1.3

Übungsaufgabe 1.3.1

Bei der fachlichen Modellierung kann es sich herausstellen, dass Anforderungen im zuvor erarbeiteten Pflichtenheft falsch beschrieben wurden oder fehlen. Wie sollte in diesem Fall vorgegangen werden? Begründen Sie Ihre Antwort.

Übungsaufgabe 1.3.2

Ein Reiseveranstalter plant die Entwicklung eines Auftragsabwicklungssystems, mit dem Reisen gebucht und abgerechnet werden können (Reisebuchungssystem). Eine Reisebuchung umfasst stets die Reservierung eines Hotelzimmers. Geben Sie für die nachfolgenden Beispiele jeweils an, welches der grundlegenden Merkmale einer Anforderungsdefinition nicht erfüllt wird.

a) Sagt ein Kunde eine Reise vor ihrem Antritt ab, sollen zugehörige Buchungssätze storniert, d.h. passend gekennzeichnet werden. In der Anforderungsdefinition wird stattdessen ein physisches Löschen entsprechender Buchungssätze vorgesehen.

b) Das Reisebuchungssystem soll zusätzlich einen Nachweis freier Plätze für bestimmte Flüge als Informationsdienst anbieten sowie eine komplette Auswertung der Tätigkeit des Reiseunternehmens in einer gewissen Periode ermöglichen. Diese Anforderungen tauchen aber in der Anforderungsdefinition nicht auf.

c) Einige Hotels bieten als Serviceleistung das Abholen vom Flughafen mit einem hoteleigenen Bus an. Diese Leistung wird zwar im Zusammenhang mit der Reisebuchung, nicht jedoch bei der Reiseabrechnung berücksichtigt.

d) Der unter b) beschriebene Fehler wird vermieden. Es wird aber nicht darauf geachtet, dass für einen Kunden nur Flüge von Interesse sind, die zum Zeitpunkt des Reiseantritts stattfinden.

Übungsaufgabe 1.3.3

Ordnen Sie den folgenden software-technischen Anforderungen an eine Anwendung jeweils einen oder auch mehrere der im Abschnitt „Entwurf“ genannten software-technischen Realisierungsaspekte zu.

a) Die Anwendung soll einen bidirektionalen Datenaustausch mit MS Excel vornehmen.
b) Realisierung der Anwendung in C++, wobei für Such- und Sortieroperationen geeignete Algorithmen der Standard Template Library (STL) zu benutzen sind.
c) Die Anwendung soll sowohl unter Windows NT wie auch unter Unix lauffähig sein und Anwendungsdaten in einer Oracle-Datenbank ablegen.

Übungsaufgabe 1.3.4

Neben der Kodierung der eigentlichen Anwendung können in der Implementierung noch weitere Aktivitäten erforderlich sein. Nennen Sie mindestens eine derartige Aktivität. Denken Sie z.B. daran, dass in vielen Fällen alte Anwendungen abgelöst werden.

Übungsaufgabe 1.3.5

Bei der Modellierung eines Systems zur Verwaltung von Studierenden des Fachbereichs Wirtschaftswissenschaft der FernUniversität Hagen werden die nachfolgend angegebenen Merkmale für Studierende vorgesehen. Entscheiden Sie, ob die einzelnen Merkmale relevant sind oder nicht und begründen Sie stichwortartig Ihre Entscheidung.

a) Geburtsdatum.
b) Haarfarbe.
c) Geschlecht.
d) Teilzeit- oder Vollzeitstudent(in).
e) Vorgesehenes Datum der Promotion.

1.4 Prozessmodelle

Prozessmodelle, Phasen

Prozessmodelle dienen der Strukturierung des Prozesses der Softwareentwicklung. In einem Prozessmodell wird der gesamte Entwicklungsprozess in mehrere nacheinander zu durchlaufende Phasen oder Entwicklungsstufen zerlegt. Der Inhalt einer Phase wird durch die jeweils auszuführenden Aktivitäten modellspezifisch bestimmt. Ferner ist eine Phase durch die zu erstellenden Teilprodukte und Kriterien für ihre Fertigstellung definiert.

grundlegende Entwicklungsaktivitäten versus Phasen

Die in Kap. 1.3 besprochenen grundlegenden Entwicklungsaktivitäten werden auf die eine oder andere Weise in allen Prozessmodellen berücksichtigt. Diese müssen jedoch keineswegs ununterbrochen, d.h. vollständig in einem Zuge ausgeführt werden. Folglich bestehen Freiheitsgrade hinsichtlich der Zuordnung der grundlegenden Entwicklungsaktivitäten zu den Phasen eines Prozessmodells, die einen weiten Spielraum für verschiedene Prozessmodelle eröffnen. Aus diesem Grund

wird hier auch zwischen den grundlegenden Entwicklungsaktivitäten einerseits und den Phasen von Prozessmodellen andererseits begrifflich differenziert, was in der Literatur nicht allgemein üblich ist.

Prozessmodell und Projektmanagement

Mit der Auswahl eines Prozessmodells wird ein organisatorischer Rahmen für ein gegebenes Softwareentwicklungsvorhaben abgesteckt. Dieser ist nach den jeweils konkreten Bedingungen eines Projekts auszufüllen (vgl. Kap. 1.7). Den angegebenen Aspekten des Phasenbegriffs – Aktivitäten, Teilprodukte, Fertigstellungskriterien – entsprechend, unterstützt ein Prozessmodell die Planung wie auch die operative Leitung und Kontrolle eines Projekts. Hieraus resultiert die hohe Bedeutung von Prozessmodellen für das Management der Softwareentwicklung.

In anderen ingenieurwissenschaftlichen Disziplinen, die sich mit der Entwicklung technischer Produkte befassen, sind Prozessmodelle, auch als Phasenmodelle bezeichnet, seit vielen Jahrzehnten in Gebrauch. Ihre Übertragung in das Gebiet des Software Engineering stellt ein Beispiel für die oben erwähnte Adaption von Methodenwissen dar.

Im Folgenden werden zwei maßgebliche Prozessmodelle vorgestellt. Dabei handelt es sich um das Wasserfallmodell einerseits und das evolutionäre Modell andererseits. Beide Modelle werden einschließlich wichtiger Modifikationen und Varianten besprochen. Für weitere Prozessmodelle seien interessierte Leser auf BALZERT (1998) sowie GEHRING (1993a) verwiesen.

Die Darstellung beider Prozessmodelle wird auf das Wesentliche beschränkt. So werden jeweils nur die Hauptphasen der Modelle betrachtet, die in der Praxis weiter zerlegt werden. Ferner wird eine nebenläufige Abwicklung von Aktivitäten, die durch die Bildung von Subteams innerhalb des Projektteams ermöglicht wird, nur gestreift.

1.4.1 Das Wasserfallmodell und seine Modifikationen

Wasserfallmodell

Definition

Charakteristisch für das Wasserfallmodell ist, dass die grundlegenden Aktivitäten der Softwareentwicklung eins zu eins auf Modellphasen abgebildet werden. Jeder grundlegenden Entwicklungsaktivität entspricht also genau eine Phase. Dementsprechend werden im Wasserfallmodell alle grundlegenden Entwicklungsaktivitäten jeweils in einem Zuge, ohne Unterbrechung und vollständig ausgeführt.

Alle Phasen werden einmalig und strikt sequentiell, von der ersten bis zur letzten Phase durchlaufen. Damit schließt das Wasserfallmodell eine Rückkehr zu früheren Phasen aus.

Die Abb. 1.4 stellt das Wasserfallmodell dar.

Schema

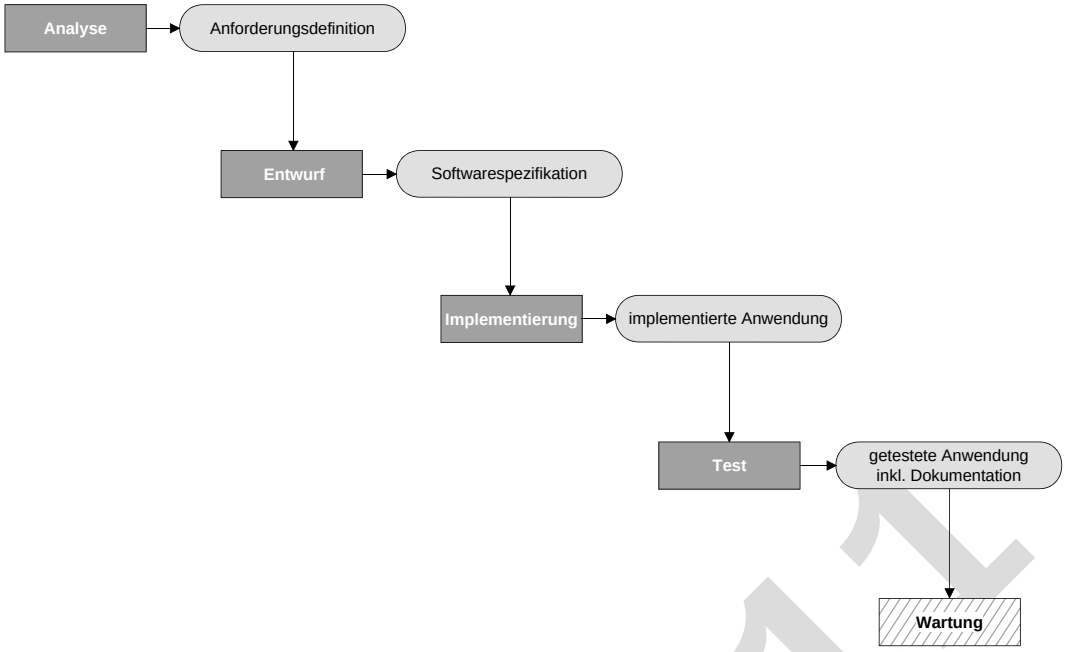


Abb. 1.4. Das Wasserfallmodell.

Jede Phase gilt als abgeschlossen, wenn die in ihr zu erarbeitenden Dokumente bzw. sonstigen Ergebnisse fertiggestellt wurden. Die Dokumente einer Phase fallen bei der in Abb. 1.4 gewählten üblichen Darstellung des Modells wie bei einem Wasserfall in die nächste Phase, was dem Modell zu seinem Namen verhalf.

Die Wartungsphase dient nicht der Entwicklung des ursprünglichen Produkts und wird in der Abbildung schraffiert dargestellt. Sie wurde dennoch berücksichtigt, weil sie die Lebensdauer eines Softwareprodukts vervollständigt.

Da die grundlegenden Entwicklungsaktivitäten und ihre Ergebnisse bereits ausführlich beschrieben wurden, erübrigen sich weitere Erläuterungen zu den Phasen des Modells. Stattdessen folgt ein Beispiel.

Beispiel 1.4

Ein Informationssystem für eine Autoverleihfirma soll folgende funktionale Hauptanforderungen erfüllen:

- Verwaltung der Stammdaten (Kunden, Lieferanten, Bankverbindungen usw.),
- Kundenberatung,
- Reservierung von Kfz,
- Vermietung von Kfz (Kfz-Verleih und Kfz-Rückgabe inkl. Abrechnung),
- Unterstützung selbst durchgeführter Kfz-Reparaturen (z.B. Teileeinkauf).

Bei der dem Wasserfallmodell folgenden Entwicklung wird zunächst ein Analysemodell erstellt, das alle fünf genannten Anforderungen berücksichtigt. Erst nach abgeschlossener Analyse wird das System wiederum unter Beachtung aller Anforderungen komplett entworfen. Nach der Implementierung des vollständigen Systems wird es schließlich getestet und bei der Autoverleihfirma eingesetzt.

weitere Merkmale, Bedeutung

Ein wesentliches Merkmal des Wasserfallmodells ist, dass eine Zusammenarbeit und Kommunikation mit dem Auftraggeber ausschließlich bei der Erarbeitung der Anforderungsdefinition während der Analysephase vorgesehen ist. Danach wird

der Auftraggeber erst wieder bei dem Abnahmetest der fertigen Anwendung aktiv (vgl. Kap. 1.3).

Das Wasserfallmodell stellt als das historisch erste Prozessmodell einen Meilenstein in der Geschichte des Software Engineering dar. Basierend auf der klaren Abgrenzung der grundlegenden Aktivitäten der Softwareentwicklung markiert das Wasserfallmodell den Übergang von einer unstrukturierten, formlosen Entwicklung zu einem strukturierten, geregelten Entwicklungsprozess. Damit leistete es einen wesentlichen Beitrag zur Bewältigung der Komplexität der Softwareentwicklung.

Das Wasserfallmodell war etwa seit dem Ende der 1960er Jahre für einige Jahrzehnte in Praxis und Theorie das dominierende Prozessmodell und wird – allerdings meist in modifizierter Form – auch gegenwärtig in Industrie und Behörden in breitem Umfang eingesetzt.

Vorteile

Kennzeichnend für das Wasserfallmodell ist seine einfache Struktur, die besonders den Bedürfnissen des Managements entgegenkommt und die Anwendung bewährter Techniken des industriellen Projektmanagements wie Netzplantechnik und Meilensteine erleichtert (vgl. Kap. 1.7).

Mängel

Nichtsdestoweniger weist das Wasserfallmodell in seiner ursprünglichen Gestalt ernsthafte Mängel auf, von denen einige genannt seien:

- In der Praxis ist es die Regel, dass in späteren Phasen Fehler entdeckt werden, die in früheren Phasen gemacht wurden. Werden etwa während des Entwurfs Fehler in der Anforderungsdefinition festgestellt, so muss diese korrigiert werden. Es ist daher unrealistisch, dass das Wasserfallmodell keine Rückkehr zu früheren Phasen vorsieht.
- Das Wasserfallmodell geht davon aus, dass jede Phase vollständig abgeschlossen sein muss, bevor die nächste beginnt. Hierdurch werden mögliche Zeitgewinne verschenkt. Diese können erzielt werden, indem z.B. bereits mit der Implementierung einiger Systemkomponenten begonnen wird, während sich andere Komponenten noch im Entwurfsstadium befinden.
- Im Wasserfallmodell kann der Auftraggeber lediglich innerhalb der Analysephase seine Anforderungen und Bedürfnisse an das Produkt artikulieren. Die Erfahrung vieler Projekte lehrt dagegen, dass sich die Anforderungen des Auftraggebers während des gesamten Zeitraums der Produktentwicklung verändern. Insbesondere werden vielfach zusätzliche Bedürfnisse und Wünsche vom Auftraggeber geäußert, wenn dieser erstmals das lauffähige Produkt sieht. Das Wasserfallmodell beachtet diese Gegebenheiten nicht.
- Nach dem Wasserfallmodell liegt eine lauffähige Anwendung erst am Schluss der Entwicklung vor. Dies kann zum einen leicht dazu führen, dass die Entwicklung an den Bedürfnissen des Auftraggebers vorbeigeht. Es birgt ferner die Gefahr, dass Fehler, die in vorherigen Phasen gemacht wurden, erst sehr spät erkannt werden, was deren Behebung immens verteuert (vgl. Kap. 1.8).

Die angeführten Mängel führten einerseits zu Modifikationen des Wasserfallmodells und andererseits zur Entwicklung grundsätzlich anderer Prozessmodelle. Hier sei zunächst auf zwei Modifikationen eingegangen, die die Nachteile des Wasserfallmodells zumindest abmildern sollen.

(1) Das Wasserfallmodell mit Rückkopplung

Wasserfallmodell mit Rückkopplung

Das Wasserfallmodell mit Rückkopplung sieht ausdrücklich die Rückkehr von späteren zu früheren Phasen vor. Die Abb. 1.5 zeigt das Wasserfallmodell mit Rückkopplung.

Schema

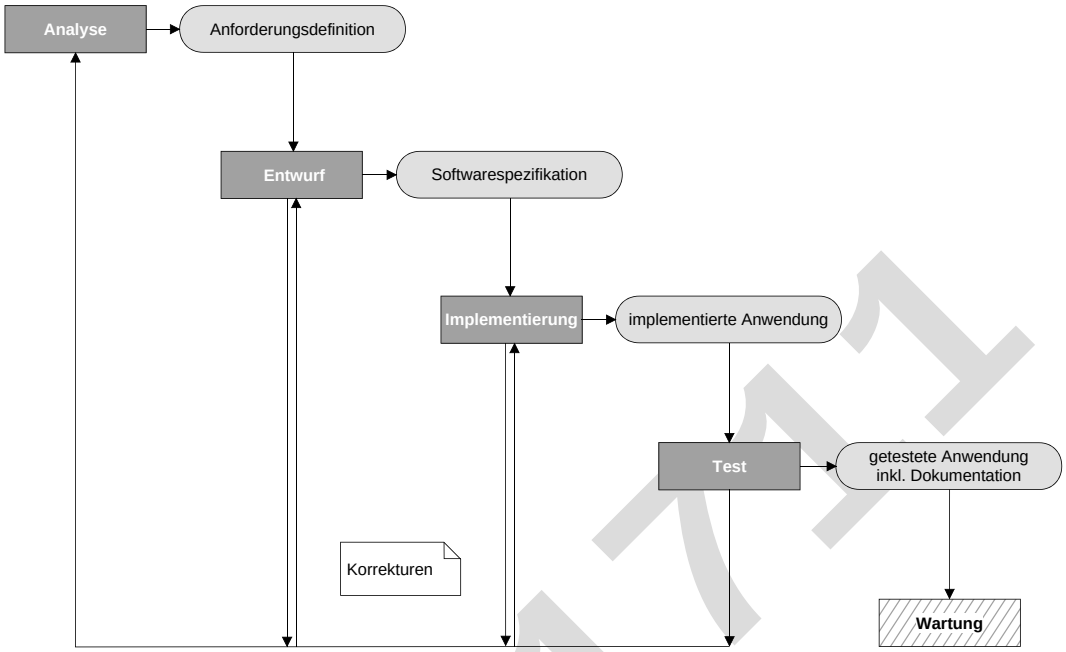


Abb. 1.5. Das Wasserfallmodell mit Rückkopplung.

Eine Rückkehr in eine frühere Phase sollte allerdings nur zu dem Zweck erfolgen, einen in dieser Phase verursachten und erst in einer folgenden Phase erkannten Fehler zu korrigieren. Gegebenenfalls kann auch mehrfach von einer oder von verschiedenen Phasen ausgehend in eine frühere Phase zurückverzweigt werden.

Die vorgestellte Modifikation lässt den Kern des Wasserfallmodells unangetastet. Wie bisher wird davon ausgegangen, dass abgesehen von Fehlerkorrekturen alle Phasen einmalig, vollständig und strikt sequentiell durchlaufen werden.

(2) Das Wasserfallmodell mit integrierter Prototyperstellung

Prototypen

Unter einem Prototyp versteht man bekanntlich ein vorläufiges Produktmuster, das während einer Produktentwicklung erstellt wird und bereits ausgewählte Eigenschaften des zu entwickelnden Zielprodukts besitzt. Auch die Anwendung von Prototypen hat das Software Engineering den klassischen Ingenieursdisziplinen entlehnt, wo diese wie etwa im Wasserstraßen- oder Automobilbau bereits viele Jahrzehnte verwendet werden.

In der Softwareentwicklung stellen Prototypen lauffähige Vorabversionen der gewünschten Anwendung dar, die verschiedenen Zwecken dienen:

Zwecke von Prototypen

- Prototypen werden erstellt, um gemeinsam mit dem Auftraggeber zu überprüfen, ob die bisherige Entwicklung seinen wahren Anforderungen entspricht und seine Akzeptanz findet. So wird häufig bereits während der Erstellung des Analysemodells ein Prototyp der Benutzungsoberfläche erstellt. Ggf. wird dann abgestimmt mit dem Auftraggeber die Anforderungsdefinition korrigiert.

- Mit Hilfe von Prototypen kann experimentell entschieden werden, ob ein bestimmter Entwurfsansatz tauglich oder welche von mehreren vorgeschlagenen Lösungsvarianten für ein Entwurfsproblem am besten geeignet ist.
- Speziell bei extremen Leistungsanforderungen – z.B. Zeitlimits für gewisse Operationen – kann mit einem Prototyp deren Einhaltung kontrolliert werden.

Prototypen werden hauptsächlich zu Beginn einer Entwicklung, nämlich während der Analyse und des Entwurfs, erstellt und benutzt.

Wegwerfprototypen ...

Unterstellt wird hier ferner eine sogenannte rasche Prototyperstellung. Bei dieser Vorgehensweise wird ein Prototyp lediglich – wie zuvor beschrieben – zur Klärung aktueller Entwicklungsprobleme verwendet und anschließend weggeworfen. Man spricht daher auch von Wegwerfprototypen.

... im Wasserfallmodell

Wegwerfprototypen werden in Verbindung mit verschiedenen Prozessmodellen angewendet. Ihr Einsatz im Rahmen des Wasserfallmodells soll vor allem dem Mangel begegnen, dass eine lauffähige Anwendung erst am Ende der Entwicklung vorliegt und vom Auftraggeber in Augenschein genommen werden kann. Die Abb. 1.6 veranschaulicht das Wasserfallmodell mit integrierter Prototyperstellung in den Phasen Analyse und Entwurf. Die Abbildung berücksichtigt zugleich die zuvor erläuterte Erweiterung um Rückkopplungen zu früheren Phasen.

Schema

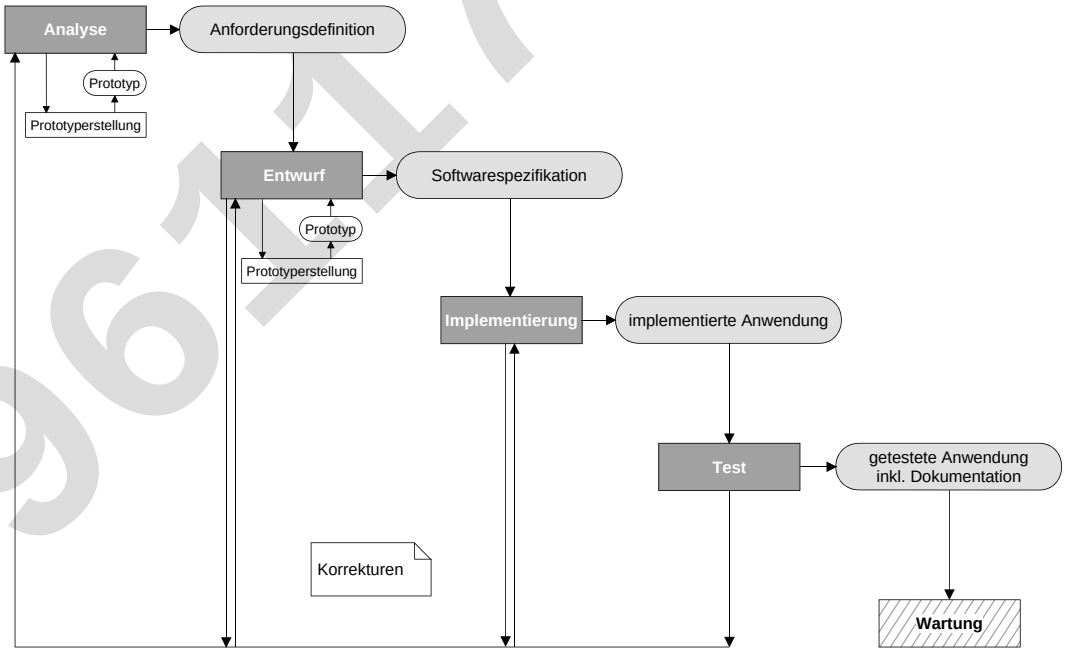


Abb. 1.6. Das Wasserfallmodell mit Rückkopplung und integrierter Prototyperstellung.

Auch die Erweiterung des Wasserfallmodells um eine integrierte Erstellung von Wegwerfprototypen lässt den Kern dieses Modells unberührt.

1.4.2 Das evolutionäre Modell und seine Varianten

Motiv für evolutionäre Entwicklung

Bei umfangreichen und komplizierten Projekten ist eine vollständige Ausführung der grundlegenden Entwicklungsaktivitäten, die wie bei dem Wasserfallmodell auf eine komplette Anwendung abzielt, oft problematisch und mit gravierenden Nachteilen verbunden.

So zeigt die Erfahrung einerseits, dass der Auftraggeber zu Beginn der Entwicklung häufig nicht in der Lage ist, seine Anforderungen vollständig zu überblicken. Wesentliche Anforderungen ergeben sich vielmehr sowohl bei einem konkreten Auftraggeber wie auch bei der Produktion für den anonymen Softwaremarkt oft erst bei fortgeschrittener Entwicklung oder gar erst beim Einsatz des Produkts. Zum anderen zieht eine vollständige Anwendungsentwicklung bei großen Projekten nach sich, dass der Auftraggeber oder der Markt jahrelang auf die Anwendung warten müssen, was empfindliche Wettbewerbsnachteile für Auftraggeber oder Softwarehersteller mit sich bringen kann.

evolutionäres Prozessmodell, Definition und Charakterisierung

Vor allem für größere Entwicklungsvorhaben bietet sich das evolutionäre Prozessmodell an, das auf die Abstellung der genannten Probleme und Nachteile abzielt. Es lässt sich wie folgt charakterisieren:

- Es werden mehrere Versionen der Anwendung bis hin zum kompletten Produkt entwickelt und an den Auftraggeber ausgeliefert, wobei jede Version die Vorgängerversion verbessert. Die komplette Anwendung wird also stufenweise entwickelt.
- Die Verbesserung einer Vorgängerversion schließt zum einen Erweiterungen der Funktionalität der Anwendung ein. Zum anderen werden entsprechend den beim praktischen Einsatz der Vorgängerversion gesammelten Erfahrungen bisherige Schwachstellen beseitigt.
- Bei der Initiierung der ersten oder einer Folgeversion durch den Auftraggeber genügt es, wenn er diejenigen Anforderungen äußert, deren Realisierung für ihn aktuell von vorrangiger Bedeutung ist und die er bereits gut überblickt. Allerdings ist es von großem Vorteil für den Gesamtprozess der Anwendungsentwicklung, wenn der Auftraggeber bereits bei der Entwicklung der ersten Version die wesentlichen Anforderungen an die komplette Anwendung artikulieren kann. Hierauf wird noch näher eingegangen. Natürlich wird auch der Auftragnehmer auf die Gestaltung der Versionen Einfluss nehmen. Aus seiner Perspektive sollten vorzugsweise die besonders kritischen und architekturbestimmenden Anforderungen in den ersten Versionen realisiert werden.

Versionen

- Pro Version wird ein vollständiger Entwicklungszyklus bestehend aus den Phasen Analyse, Entwurf, Implementierung und Test durchlaufen. Jedoch werden in jeder Phase nur die zusätzlichen oder geänderten Anforderungen der aktuellen Version berücksichtigt. Bezogen auf die komplette Anwendung werden also die grundlegenden Entwicklungsaktivitäten nicht vollständig, sondern jeweils nur partiell ausgeführt. Die Erstellung der Dokumente und des Programmcodes der neuen Version erfolgt – bei jeder Folgeversion – ausgehend von den entsprechenden Ergebnissen der Vorgängerversion.
- Eine gesonderte Wartungsphase entfällt. Auch die Durchführung reiner Fehlerkorrekturen, die erst beim Einsatz einer Version festgestellt werden, wird grundsätzlich wie die Erstellung einer neuen Version behandelt.

evolutionäre Entwicklung und Prototypen

Jede Produktversion lässt sich als Prototyp aller Nachfolgerversionen auffassen. Im Unterschied zur raschen Prototypenerstellung handelt es sich jedoch nicht mehr um Wegwerfprototypen. Diese müssen nicht mit besonderer Sorgfalt und entsprechend den beim Softwarehersteller geltenden Standards für Entwurf und Implementierung angefertigt werden. Dies gilt natürlich nicht für die gemäß dem evolutionären Modell erstellten Produktversionen, die zum einen für den Einsatz beim Auftraggeber bestimmt und zum anderen in Folgeversionen weiterentwickelt

werden sollen. Hier gelten insbesondere im Hinblick auf die Wartbarkeit dieselben Anforderungen wie für die abschließende komplette Version der Anwendung.

Im Folgenden werden zwei Varianten des evolutionären Modells vorgestellt, die sich vor allem hinsichtlich des Umfangs der Analyse bei der Erstellung der ersten Produktversion unterscheiden. Inhaltlich wird dabei BALZERT (1998) gefolgt. Allerdings werden eigene Variantenbezeichnungen benutzt, da in der Literatur vorkommende Bezeichnungen für Varianten des evolutionären Ansatzes teils inkonsistent sind. Oft wird das evolutionäre Prozessmodell auch als inkrementelles Modell bezeichnet bzw. von inkrementeller Softwareentwicklung gesprochen.

evolutionär = inkrementell

(1) Das evolutionäre Modell mit partieller Analyse

Variante mit partieller Analyse

Bei dem evolutionären Modell mit partieller Analyse wird die Analysephase für alle Produktversionen in analoger Weise behandelt. Es werden stets nur die aktuellen bzw. hinzukommenden Anforderungen analysiert und in das fachliche Modell der aktuellen Version aufgenommen, wobei allerdings auf Vorgängerversionen aufgesetzt wird. Anders gesagt wird also nicht der Versuch unternommen, gleich zu Beginn der Anwendungsentwicklung, d.h. bei der ersten Version, die Gesamtheit der fachlichen Anforderungen zu erfassen und zu modellieren.

Vor einer schematischen Darstellung der evolutionären Entwicklung mit partieller Analyse sei diese anhand eines Beispiels veranschaulicht.

Beispiel 1.5

Betrachtet sei erneut das im vorigen Beispiel skizzierte Informationssystem für eine Autoverleihfirma, dessen schließlich erreichte Gesamtfunktionalität nochmals aufgelistet sei:

- Verwaltung der Stammdaten (Kunden, Lieferanten, Bankverbindungen usw.),
- Kundenberatung,
- Reservierung von Kfz,
- Vermietung von Kfz (Kfz-Verleih und Kfz-Rückgabe inkl. Abrechnung),
- Unterstützung selbst durchgeführter Kfz-Reparaturen (z.B. Teileeinkauf).

Der Auftraggeber wünscht zunächst nur ein System zur Unterstützung der Kfz-Vermietung. Diese ist gemeinsam mit der Verwaltung zugehöriger Stammdaten Gegenstand der ersten Systemversion. In einer zweiten Version wird das System um die Reservierung und die Kundenberatung erweitert. Erst in der dritten Version wird auch die Unterstützung von Reparaturen einschließlich der Verwaltung von Lieferantendaten in das System integriert. Pro Version folgen Analyse, Entwurf, Implementierung und Test aufeinander, die jedoch jeweils nur entsprechend dem Versionsumfang vorgenommen werden. Da der Auftraggeber seine (hauptsächlichen) Anforderungen erst nach und nach überblickt bzw. Aufträge jeweils nur für einen bestimmten Teil der Gesamtfunktionalität erteilt, wird eine evolutionäre bzw. inkrementelle Entwicklung mit partieller Analyse durchgeführt.

Die Abb. 1.7 stellt das evolutionäre Modell mit partieller Analyse dar.

Schema

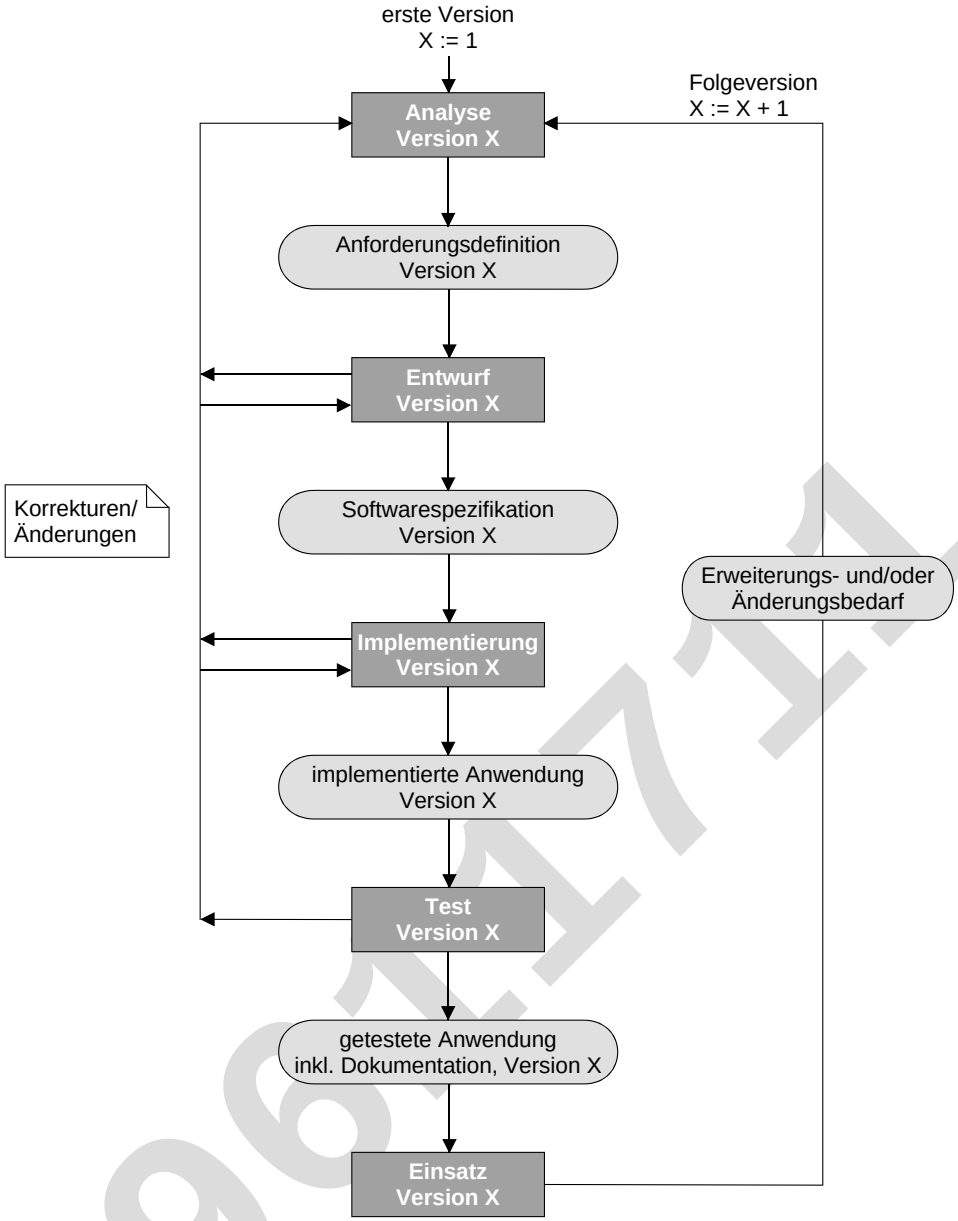


Abb. 1.7. Das evolutionäre Prozessmodell mit partieller Analyse.

Nachteil der Variante

Auf Vor- und Nachteile des evolutionären Modells wird weiter unten noch eingegangen. Hier sei nur konstatiert, dass die Variante mit partieller Analyse die Gefahr in sich birgt, dass bei der Erstellung der ersten Version Kernanforderungen an die komplette Anwendung unberücksichtigt bleiben. Dies wiederum kann dazu führen, dass grundlegende Architekturentscheidungen im Entwurf der ersten oder auch folgender Versionen sich später als ungeeignet erweisen, was dann auf einen unverhältnismäßig hohen Änderungsaufwand hinausläuft. Diesen Nachteil will die zweite hier vorgestellte Variante des evolutionären Modells vermeiden.

(2) Das evolutionäre Modell mit vollständiger Analyse

Variante mit vollständiger Analyse

Bei dem evolutionären Modell mit vollständiger Analyse werden bereits bei der Erstellung der ersten Produktversion alle wesentlichen Anforderungen an die komplette Anwendung in der Analysephase erfasst und modelliert. Auf diese Weise sollen zu Beginn der Entwicklung tragfähige Entwurfsentscheidungen un-

terstützt werden, die in der Folgezeit möglichst nicht mehr revidiert werden müssen.

Zu beachten ist, dass auch bei dieser Modellvariante Entwurf und Implementierung der ersten Produktversion sich nur auf die aktuell zu realisierenden Anforderungen beziehen. Auf eine schematische Darstellung der Modellvariante mit vollständiger Analyse sei hier verzichtet.

Vorteile evolutionärer Entwicklung

Das evolutionäre Prozessmodell besitzt gegenüber dem Wasserfallmodell wie auch anderen Modellen mit einer vollständigen Ausführung der grundlegenden Entwicklungsaktivitäten vor allem folgende Vorteile:

- Die Kommunikation mit dem Auftraggeber gestaltet sich kontinuierlicher und intensiver. Hierdurch wird der Gefahr vorgebeugt, dass an den wahren Bedürfnissen des Auftraggebers vorbei entwickelt wird.
- Übermäßig lange Wartezeiten auf einsatzfähige Produkte werden vermieden. Die beim praktischen Einsatz einer Produktversion gesammelten Erfahrungen können für die Produktverbesserung genutzt werden.
- Die verhältnismäßig rasche Erstellung eines lauffähigen Produkts hilft bei der frühzeitigen Aufdeckung grundlegender Analyse- und Entwurfsfehler.
- Der Auftraggeber wird nicht damit überfordert, dass er zu Beginn der Entwicklung die Gesamtheit der Anforderungen überblicken muss. Dies gilt allerdings bei der Modellvariante mit vollständiger Analyse nur in beschränktem Maße.

höhere Anforderungen

Den gewichtigen Vorteilen des evolutionären Modells steht die Tatsache gegenüber, dass eine evolutionäre Softwareentwicklung bedeutend höhere Anforderungen an das Management stellt als eine Entwicklung nach dem Wasserfallmodell. Sowohl die Planung der Projektressourcen als auch die Kontrolle des Projektfortschritts werden schwieriger. Außerdem sind wegen der häufigen Überarbeitung des Produkts besonders hohe Maßstäbe an die Wartbarkeit der erstellten Software anzulegen.

Übungsaufgaben zu Kapitel 1.4

Übungsaufgabe 1.4.1

Warum geht in Abb. 1.5 kein Rückkopplungspfeil von der Wartungsphase aus?

Übungsaufgabe 1.4.2

Warum werden die in den Phasen Analyse und Entwurf erarbeiteten Prototypen in Abb. 1.6 nicht als Phasenergebnisse ausgewiesen?

Übungsaufgabe 1.4.3

Warum wird die bei einer evolutionären Entwicklung auf den Test einer Version folgende Phase in Abb. 1.7 als „Einsatz“ und nicht wie bei dem Wasserfallmodell als „Wartung“ bezeichnet?

Übungsaufgabe 1.4.4

Warum sind bei einer evolutionären Entwicklung Planung und Fortschrittskontrolle für das Management grundsätzlich mit größeren Schwierigkeiten verbunden als bei dem Wasserfallmodell? Unterscheiden sich beide Varianten des evolutionären Modells auch in dieser Hinsicht?

Übungsaufgabe 1.4.5

Das in Übungsaufgabe 1.3.2 beschriebene Reisebuchungssystem soll inkrementell entwickelt werden. Schlagen Sie drei Versionen des Systems vor, wobei die aus der Sicht des Auftraggebers wichtigsten Anforderungen zuerst realisiert werden sollen.

1.5 Entwicklungsmethoden, Konzepte und Notationen

allgemeiner Methodenbegriff

Ganz allgemein versteht man unter einer Methode einen Weg oder eine Vorschrift zur Lösung eines gegebenen Problems. Sie spezifiziert gewöhnlich eine Reihenfolge mehrerer Lösungsschritte und beschreibt diese zugleich im Einzelnen. Ist keine Methode vorhanden, kann ein Problem nur versuchsweise, durch Probieren gelöst werden. Dabei wird man oft zunächst untaugliche Versuche unternehmen oder auch das Ziel gänzlich verfehlen. Eine Methode hingegen gestattet im Idealfall, dass Probleme garantiert, ohne Umwege und ohne zusätzliche Entscheidungen gelöst werden können. So können etwa die (komplexen) Wurzeln einer quadratischen Gleichung mit Hilfe einer geeigneten Berechnungsformel stets und direkt bestimmt werden.

Methoden der Softwareentwicklung

Methoden der Softwareentwicklung besitzen die genannten idealen Merkmale meist nur eingeschränkt. Insbesondere verbleiben stets zusätzliche Entscheidungen, die ein Entwickler geeignet treffen muss. Immerhin können aber durch den Einsatz von Entwicklungsmethoden untaugliche Lösungen in der Regel vermieden und verbleibende Entscheidungen erleichtert werden.

Beispiel strukturierte Programmierung

Beispiel 1.6

Die Methode der strukturierten Programmierung wird zur Programmierung von Funktionen eingesetzt. Sie schreibt vor, dass bei der Strukturierung bzw. Ablaufgestaltung ausschließlich die Steuerkonzepte (Kontrollstrukturen)

- Sequenz,
- Auswahl und
- Wiederholung

Verwendung finden. Zur Umsetzung dieser Konzepte stehen in prozeduralen und objektorientierten Programmiersprachen jeweils eine oder mehrere Anweisungen zur Verfügung; z.B. realisiert eine for-Schleife in C das Wiederholungs-Konzept. Auf die Verwendung der goto-Anweisung, die beliebige Sprünge innerhalb von Programmbausteinen ermöglicht, ist zu verzichten. Die Methode der strukturierten Programmierung führt zu weniger fehleranfälligem, besser lesbarem und daher leichter wartbarem Quellcode. Sie gewährleistet insbesondere, dass die dynamische Ausführungsreihenfolge von Anweisungen mit der statischen Reihenfolge der Anweisungen im Quellprogramm weitgehend übereinstimmt.

Konzepte der Softwareentwicklung

Softwareprodukte stellen Abbilder von Ausschnitten der Realwelt dar (vgl. Kap. 1.1.2). Um Realitätsausschnitte in Softwareprodukte zu transformieren, benötigt die Softwareentwicklung Konzepte. Diese legen fest, wie die Softwareentwicklung die reale Welt wahrnimmt oder anders gesagt, wie die Gegenstände der Welt in den zunächst erstellten Modellen sowie in dem fertigen Softwareprodukt repräsentiert werden. Konzepte stellen also Wahrnehmungsformen und Wahrnehmungsmuster dar. Der Konzeptbegriff sei wiederum anhand eines Beispiels aus der Programmierung veranschaulicht. Übrigens sind bereits die zuvor angesprochenen Kontrollstrukturen der strukturierten Programmierung Beispiele für Programmierkonzepte.

Beispiel für Konzepte

Beispiel 1.7

Zur Darstellung von Gegenständen mit mehreren relevanten Merkmalen bzw. Daten bediente man sich in prozeduralen Programmiersprachen wie Pascal oder C sogenannter Records (in C Strukturen genannt), die mehrere Datenelemente zu einem Ganzen verbinden.

So werden etwa die Artikelmerkmale Bezeichnung, Artikelnummer und Preis in einem Artikel-Record zusammengefasst. Nur diese Darstellung bringt zum Ausdruck, dass die genannten Merkmale gemeinsam ein Ganzes, nämlich einen realen Artikel, repräsentieren. Eine separate Abbildung der Merkmale auf einzelne Variablen leistet dies nicht. Mit der Record-Darstellung wird also eine im Vergleich zur separaten Abbildung einzelner Merkmale wirklichkeitsnähere Wiedergabe realer Gegenstände erreicht. Die Repräsentation von Gegenständen durch Records schließt ein, dass diese auch als Ganzes – etwa in Zuweisungen – manipuliert werden können. Dies führt zu klarerem und besser lesbarem Quellcode.

Notation

textuell oder grafisch

Eine Notation definiert eine Darstellungsform von Konzepten oder allgemeiner von Informationen. Zur Repräsentation von Konzepten werden sowohl textuelle, d.h. schriftsprachliche, wie auch grafische, also bildhafte Notationen benutzt.

Letztere umfassen in der Regel auch textuelle Elemente, z.B. Namen. Die Hauptgründe für eine grafische Modellierung bestehen in den damit gegebenen Möglichkeiten der Konzentration auf das Wesentliche, der größeren Übersichtlichkeit, der höheren Informationsdichte („Ein Bild sagt mehr als 1000 Worte!“) sowie der Werkzeugunterstützung und automatischen Modellprüfung.

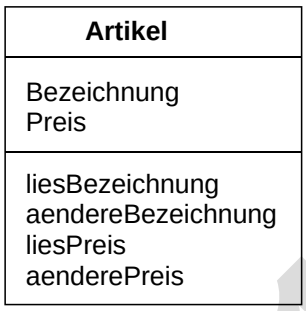
Das folgende Beispiel zeigt, dass ein- und dasselbe Konzept mittels verschiedener grafischer Notationen repräsentiert werden kann.

Beispiel:
ein Konzept – mehrere Notationen

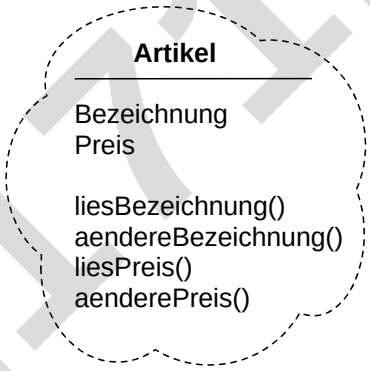
Beispiel 1.8

Das objektorientierte Konzept der Klasse fasst Attribute (Daten) und Operationen zu einer Einheit zusammen. So enthält eine Klasse Artikel etwa die Attribute Bezeichnung und Preis sowie die Operationen liesBezeichnung, aendereBezeichnung, liesPreis und aenderePreis. Nachfolgend werden zwei bekannte Notationen des Klassenkonzepts anhand der Klasse Artikel vorgestellt.

Notation nach RUMBAUGH et al. (1993)



Notation nach BOOCH (1994)



formale, informale oder semiformale Notation

Eine formale Notation liegt – grob gesagt – dann vor, wenn bei der Darstellung einzelner Konzepte und bei ihrer kombinierten Verwendung gewisse syntaktische Regeln strikt einzuhalten sind. Ist dies nicht der Fall, so spricht man von informalen Beschreibungen bzw. Notationen. Semiformale Notationen nehmen eine Mittelstellung zwischen formalen und informalen Notationen ein.

Beispiel
für formale und semiformale Notation

Beispiel 1.9

In einem C-Programm sind sämtliche Programmierkonzepte (z.B. Variablen, Kontrollstrukturen) der C-Syntax entsprechend zu kodieren; angewendet wird also – wie in jeder Programmiersprache – eine formale Notation.

Wird dagegen ein auf C basierender Pseudocode benutzt, so wird eine formale Notation etwa von Kontrollstrukturen mit einer freien, nicht durch syntaktische Regeln beschränkten Beschreibung von Anweisungen kombiniert. Zum Beispiel:

```
if (ArtikelNr bereits vorhanden)
      aendere Artikeldaten;
else erfasse Artikeldaten;
```

Pseudocode ist demnach ein Beispiel für eine semiformale Notation, während eine rein umgangssprachliche Beschreibung eines Algorithmus einer informalen Notation entspricht.

Wie das Beispiel 1.8 bereits vermuten lässt, sind grafische Notationen von Konzepten häufig zugleich semiformal, verlangen also die Einhaltung syntaktischer Regeln bei der Verwendung grafischer Symbole.

Definition Softwareentwicklungs-Methode

Unter Einbeziehung der Begriffe des Konzepts und der Notation lassen sich die Methoden der Softwareentwicklung konkreter definieren. Zu diesem Zweck sei RUMBAUGH (1996) zitiert: “A method can be divided into two major parts: the modeling language (what) and the process (how). The modeling language includes the modeling concepts and the notation to represent them.” Eine Methode basiert also auf einer Menge von Konzepten sowie einer Notation dieser Konzepte. Darüber hinaus bestimmt sie, wie diese Konzepte anzuwenden sind. Dieser, von RUMBAUGH „Process“ genannte Teil einer Methode wird hier als methodische Vorgehensweise bezeichnet. Sie umfasst einerseits strategische Richtlinien, die etwa die Reihenfolge von Schritten bei der Modellerstellung betreffen, und andererseits taktische Hinweise zur Anwendung einzelner Konzepte.

allgemeine Methoden

Auf spezifische Entwicklungsmethoden, die etwa auf einem bestimmten Satz von Konzepten basieren oder nur im Rahmen einer einzelnen Entwicklungsaktivität einsetzbar sind, ist hier noch nicht einzugehen. Doch kennt das Software Engineering auch allgemeine Methoden, die wohl besser als generelle methodische Ansätze bezeichnet werden und die im Rahmen diverser spezifischer Methoden zur Anwendung kommen. Kennzeichnend für die generellen methodischen Ansätze ist, dass sie Vorgehensrichtungen für Entwicklungsaktivitäten vorschlagen. Die Tab. 1.2 informiert über die vier wichtigsten generellen methodischen Ansätze.

Definition allgemeine Methoden

Methodischer Ansatz/ Entwicklungsrichtung	Erläuterung, Vor- und Nachteile
top down	- Entwicklung ausgehend vom Ganzen zum Detail, vom Allgemeinen zum Speziellen, - Konzentration auf das Wesentliche und Entdeckung struktureller Beziehungen wird begünstigt, - kann aber dazu führen, dass Probleme nach „unten“ abgedrängt und nicht rechtzeitig behandelt werden.
bottom up	- Entwicklung ausgehend vom Detail zum Ganzen, vom Speziellen zum Allgemeinen, - Begrenzung auf handhabbare, überschaubare Teile wird gefördert, - Wiederverwendung von Komponenten wird unterstützt, - kann aber dazu führen, dass allgemeine strukturelle Zusammenhänge nicht aufgedeckt werden, - ebenso kann die Integrationsfähigkeit von Teilprodukten leiden.
outside in	- Entwicklung vom Äußeren zum Inneren, von den Schnittstellen zur Umwelt zum Kern, - Konzentration auf Anforderungen der Umwelt wird gefördert, - kann aber dazu führen, dass interne Probleme verdeckt werden.
inside out	- Entwicklung vom Inneren zum Äußeren, vom Kern zu den Schnittstellen zur Umwelt, - Konzentration auf Systeminterna und Systemstruktur wird gefördert, - kann aber zur Vernachlässigung von Umweltanforderungen führen.

Tab. 1.2. Generelle methodische Ansätze der Softwareentwicklung.

Zur Veranschaulichung der generellen methodischen Ansätze und ihrer Umsetzung im Rahmen spezifischer Methoden dient folgendes Beispiel.

Beispiel für Top-Down- und Bottom-Up-Ansatz

Beispiel 1.10

Kleinere Programme oder Programmkomponenten werden häufig mit Hilfe der Methode der schrittweisen Verfeinerung erstellt. Dabei werden ausgehend von einem Hauptprogramm, das die Gesamtfunktionalität repräsentiert, schrittweise Teilaufgaben geringerer Funktionalität identifiziert und als Funktionen realisiert. Ist eine Teilaufgabe noch zu komplex, wird sie weiter verfeinert, d.h. in einfachere Teilaufgaben zerlegt. Andernfalls wird sie direkt als Funktion implementiert. Die Methode der schrittweisen Verfeinerung realisiert also den Top-Down-Ansatz der Entwicklung vom Allgemeinen zum Speziellen.

Umgekehrt können Programme auch bottom up entwickelt werden. In diesem Fall werden ausgehend von Funktionen für elementare Teilaufgaben schrittweise „mächtigere“ Funktionen entwickelt, bis die Gesamtfunktionalität des Programms zur Verfügung steht.

Auswahl allgemeiner Methoden

Zur Auswahl einer Entwicklungsrichtung lassen sich nur grobe Tendenzaussagen treffen. Liegen für eine spezielle Anwendungsentwicklung noch keine Erfahrungen vor, so wird man z.B. eher einem Top-Down-Ansatz als einem Bottom-Up-

Ansatz folgen. Umgekehrt kann eine Bottom-Up-Entwicklung der bessere Weg sein, wenn spezielles Know-how bereits vorhanden ist und wiederverwendbare Komponenten zur Verfügung stehen. In sehr vielen Fällen wird die Kombination zweier Ansätze, etwa des Top-Down- und des Bottom-Up-Ansatzes, der solideste und erfolgversprechendste Weg bei der Anwendungsentwicklung oder bei einzelnen Aktivitäten sein, weil hierdurch die komplementären Risiken der Ansätze gemeinsam reduziert werden können.

Zusammenhang Prozessmodelle – Methoden

Zu den Wechselbeziehungen zwischen Entwicklungsmethoden und Prozessmodellen sei hier lediglich bemerkt, dass ein Prozessmodell im Allgemeinen mit verschiedenen spezifischen Entwicklungsmethoden und eine Methode mit verschiedenen Prozessmodellen verträglich ist. Allerdings gibt es unterschiedliche Grade der Kompatibilität zwischen Prozessmodellen und Methoden, worauf in Kap. 2 noch einzugehen ist.

Übungsaufgaben zu Kapitel 1.5

Übungsaufgabe 1.5.1

Nassi-Shneiderman-Diagramme (Struktogramme) werden zur Darstellung von Algorithmen verwendet. Charakterisieren Sie stichwortartig ihre Notation anhand der eingeführten Notationsmerkmale.

Übungsaufgabe 1.5.2

Zur Entwicklung konzeptioneller Schemata für Anwendungen mit relationalen Datenbanken werden bekanntlich Entity-Relationship-Modelle (ERM) erstellt (vgl. GEHRING 1993b). Hierbei werden relevante Objekte des Anwendungsbereichs als Entitätstypen mit gewissen Datenattributen und Beziehungen zwischen den Objekten als – gegebenenfalls ebenso mit Datenattributen ausgestattete – Beziehungstypen modelliert. Beschreiben Sie kurz, wie ein ERM mittels eines Top-Down-Ansatzes sowie mittels eines Bottom-Up-Ansatzes erstellt werden kann.

1.6 Werkzeuge für die Softwareerstellung

Werkzeuge, CASE

Unter Werkzeugen versteht man Softwaresysteme, die zur Unterstützung der Softwareerstellung eingesetzt werden. Die computer- und softwarebasierte Softwareerstellung wird als Computer Aided Software Engineering (CASE), entsprechende Werkzeuge auch als CASE-Tools bezeichnet. Übergeordnete Ziele des Werkzeugeinsatzes sind die Erhöhung der Qualität und Produktivität sowohl der eigentlichen Softwareentwicklung als auch des Managements. Eine industrielle Softwareerstellung ohne Werkzeugunterstützung ist heute nicht mehr möglich. Insbesondere erfordert der sachgerechte Einsatz von modernen Entwicklungsmethoden bei größeren Vorhaben einen Aufwand, der ohne Werkzeuge nicht zu bewältigen ist. Analoges gilt für die Planung, Leitung und Kontrolle als wesentliche Aktivitäten des Projektmanagements.

Anwendungsbereiche

Die Werkzeugunterstützung erstreckt sich auf eine Vielzahl von Tätigkeiten und Bereiche der Softwareentwicklung. Einige wichtige und typische Anwendungsfelder und Aufgaben von CASE-Tools seien knapp umrissen:

- Unterstützung modellierender Aktivitäten (Analyse, Entwurf) auf der Grundlage spezifischer Entwicklungsmethoden; hierzu gehört auch die automatische Generierung von Quellcode in einer ausgewählten Programmiersprache aus Modellen, der anschließend zu erweitern und zu vervollständigen ist. Bei diesem üblichen Ablauf der Entwicklung neuer Produkte – Modellierung vor Implementierung – spricht man auch von Forward Engineering. Dagegen werden beim Reverse Engineering umgekehrt Modelle oder Modellkomponenten werkzeuggestützt aus vorhandenem Quellcode gewonnen.
- Unterstützung der Implementierung durch Editoren, Compiler, Debugger und weitere Implementierungswerkzeuge.
- Erzeugung, Änderung und Verwaltung von Entwicklungsdokumenten, Daten, Modellen, Programmcode, Bibliotheken, Teil- und Fertigprodukten. Die Dokumentenerzeugung basiert auf festgelegten Standards und Konventionen. Um Änderungen konsistent über mehrere Dokumente, z.B. Modelle, hinweg vornehmen zu können, ist eine integrierte Dokumentenverwaltung erforderlich. Dies schließt die Verwaltung von Querbezügen und Referenzen ein.
- Unterstützung der Qualitätssicherung durch automatisierte Prüfung der (fachlichen) Konsistenz von Entwicklungsdokumenten, insbesondere Modellen. Auch die Prüfung von Qualitätsmerkmalen, die durch sogenannte Metriken operationalisiert werden, kann automatisch erfolgen (vgl. Kap. 1.8.1).
- Unterstützung des Projektmanagements. Hierzu gehören die Erzeugung und Verwaltung von Dokumenten des Managements, die ablauforganisatorische (z.B. Terminpläne, standardisierte Berichte) und aufbauorganisatorische (Mitarbeiter, Teams, Rollen usw.) Aspekte betreffen. Zur Projektplanung werden z.B. Werkzeuge der Netzplantechnik eingesetzt (vgl. Kap. 1.7.2)
- Versions- und Konfigurationsmanagement für Teil- und Fertigprodukte.

Vorteile und Bedeutung

Aus dem Einsatz von Werkzeugen ergeben sich allgemeine positive Effekte, von denen einige angesprochen seien. Werkzeuge zur Unterstützung von Entwicklungsmethoden tragen zur disziplinierten und einheitlichen Umsetzung von Methoden bei. Ein erheblicher Teil möglicher Fehler wird von vornherein ausgeschlossen und die Kommunikation der Entwickler wird gefördert. Die werkzeugvermittelte Methodendisziplin verbessert vor allem die Softwarequalität. Hierzu tragen insbesondere automatische oder teilautomatische Qualitätsprüfungen bei.

Im Bereich des Managements unterstützt der Werkzeugeinsatz vor allem die operative Leitung und Kontrolle des Projektablaufs. So wird die Kontrolle und Steuerung erleichtert, indem Werkzeuge standardisierte, den festgelegten Anforderungen entsprechende Berichte und Entwicklungsdokumente mehr oder minder erzwingen. Die integrierte Dokumentenverwaltung im Bereich der eigentlichen Entwicklung wie des Managements selbst gestattet diesem den raschen Zugriff auf und den Abgleich von Projektinformationen.

Als weiterer wesentlicher Vorzug der Werkzeugunterstützung sei die Entlastung von aufwendigen Routinetätigkeiten genannt. Wird durch eine automatisierte, schnelle Ausführung von Routineoperationen einerseits die Produktivität unmit-

telbar angehoben, so können sich andererseits Entwickler, aber auch Manager andererseits verstärkt ihren eigentlichen, kreativen Aufgaben widmen.

Resultieren aus dem Werkzeugeinsatz unverzichtbare Vorteile, so muss doch vor einer übertriebenen „Werkzeugeuphorie“ gewarnt werden. Werkzeuge ermöglichen eine höhere Produktivität und verbesserte Qualität, garantieren diese aber keineswegs. Insbesondere können Werkzeuge die Entwickler nicht davon befreien, sich systematisch die benötigten Entwicklungsmethoden anzueignen. Und menschliche Kreativität kann durch Softwarewerkzeuge ohnehin nicht ersetzt werden. Schließlich muss darauf geachtet werden, dass der Werkzeugeinsatz insbesondere im Managementbereich nicht zu einer „Softwarebürokratie“ und „Papierflut“ ausartet, die die Beteiligten frustriert und der Produktivität einen Bärendienst erweist.

Softwareentwicklungsumgebungen (SEU)

Wurden ursprünglich Aufgaben wie die oben genannten durch separate Werkzeuge abgedeckt, so geht der Trend bereits seit Jahren in Richtung multifunktionaler CASE-Tools. Diese fassen mehrere Werkzeuge „unter einem Dach“ zusammen, gestatten ihre kombinierte Anwendung und werden auch als Softwareentwicklungsumgebungen (SEU) bezeichnet. Dabei ist das Ausmaß der Werkzeugintegration in SEU durchaus verschieden. So werden auch Programmierumgebungen, die im wesentlichen Implementierungswerkzeuge umfassen, zu den SEU gerechnet. Ebenso viele CASE-Tools der jüngeren Vergangenheit, die hauptsächlich Modellierungswerkzeuge für Analyse und Entwurf anbieten.

Einen weitaus höheren Grad der Integration erreichen SEU, die zugleich

- Werkzeuge für alle grundlegenden Entwicklungsaktivitäten bzw. -phasen,
- phasenunabhängige Werkzeuge z.B. zur Dokumentenverwaltung und
- managementspezifische Werkzeuge

Anforderungen an SEU

umfassen. Für ein Architekturmodell derartiger Entwicklungsumgebungen sei auf BALZERT (1998) verwiesen. Eine entscheidende Anforderung an derartige SEU ist natürlich, dass alle Einzelwerkzeuge quasi beliebig kombiniert genutzt werden können. Ausgaben eines Werkzeugs sollen etwa, wann immer dies sinnvoll ist, als Eingaben eines anderen Werkzeugs verwendbar sein. Eine weitere grundlegende Anforderung ist die nach einer offenen Architektur, die durch bereitgestellte Schnittstellen auch die Einbindung zusätzlicher, fremdproduzierter Werkzeuge gestattet. Von zunehmender Bedeutung ist schließlich die Internet- und Intranetfähigkeit von CASE-Tools. Die mit diesem Kurs bereitgestellte SEU (und ihre Dokumentation) vermittelt einen konkreten Eindruck von der Funktionalität und Komplexität aktueller Entwicklungsumgebungen.

Übungsaufgaben zu Kapitel 1.6

Übungsaufgabe 1.6.1

Informieren Sie sich ganz grob über die von der Entwicklungsumgebung Together bereitgestellte Funktionalität. Hinweis: Einen Überblick der Funktionalität von Together kann man sich unter http://www.togethersoft.com verschaffen. Geben Sie für vier grundlegende Aufgaben von CASE-Tools je ein oder zwei zugehörige Funktionen von Together an.

1.7 Projektmanagement

Softwaremanagement

Das Softwaremanagement umfasst einerseits das Projektmanagement zur Abwicklung einzelner Entwicklungsvorhaben und andererseits vorhabensübergreifende und unternehmensweite Aufgaben. Hierzu zählen z.B. die längerfristige Gestaltung der Aufbauorganisation relevanter Unternehmensbereiche, die Personalpolitik (Personaleinstellung und -fortbildung) oder die Initiierung und Durchsetzung unternehmensweiter Standards, darunter Richtlinien für die Produkt- und Entwicklungsprozessgestaltung.

Projektmanagement und ...

... seine Aufgaben

Hier wird lediglich eine Einführung in das Projektmanagement gegeben. Charakteristisch für ein Projekt ist, dass es auftragsgebunden, einmalig und zeitlich begrenzt ist. Zu den Aufgaben des Projektmanagements gehören vor allem:

- die Untersuchung der Durchführbarkeit des Projekts bei Projektbeginn,
- die Planung des Projekts sowie
- die Leitung und Kontrolle des Projekts.

Nachfolgend werden die genannten Aktivitäten im Einzelnen vorgestellt.

1.7.1 Durchführbarkeitsuntersuchung

Durchführbarkeitsuntersuchung, Durchführbarkeitsstudie

Liegt ein Bedarf oder Auftrag für ein Softwareprodukt vor, so werden zunächst in einer Durchführbarkeitsuntersuchung die Realisierbarkeit, die Wirtschaftlichkeit und die Risiken eines entsprechenden Vorhabens geprüft. Die Ergebnisse werden in einer Durchführbarkeitsstudie zusammengefasst, auf deren Basis im Anschluss die Unternehmensleitung über die Projektrealisierung entscheidet.

fachliche Vorstudie

Die Durchführbarkeitsuntersuchung wird unter Federführung und wesentlicher Beteiligung des Managements, etwa einer provisorischen Projektleitung, vorgenommen. Sie bedarf allerdings einer fachlichen Grundlage. Daher wird als erster Schritt der Durchführbarkeitsuntersuchung meist durch Softwareentwickler eine fachliche Vorstudie angefertigt. In der fachlichen Vorstudie werden bereits die wesentlichsten und kritischen Anforderungen an das Produkt identifiziert. Sie ist zugleich eine erste Version des Pflichtenheftes und gehört aus methodischer Sicht bereits zur Analyse. Die Erstellung einer fachlichen Vorstudie wird eingehend in Kapitel 3 besprochen.

Die eigentliche Durchführbarkeitsuntersuchung umfasst vor allem die folgenden Aspekte, die in der Durchführbarkeitsstudie ihren Niederschlag finden sollten.

(1) Projektkalkulation

Projektkalkulation

Die Projektkalkulation dient der ersten, noch groben Schätzung des Projektaufwands. Hierzu gehört in erster Linie der personelle Aufwand, ferner die benötigten Ressourcen wie Hardware und Entwicklungswerkzeuge. Die Projektkalkulation ist Voraussetzung für die folgende Einschätzung der Durchführbarkeit und Wirtschaftlichkeit des Projekts.

Schätzung des personellen Aufwandes

Der personelle Aufwand wird als zeitliche Größe (Realisierungszeit) etwa in Mitarbeitermonaten geschätzt. Das folgende Beispiel skizziert ein recht grobes Schätzverfahren, das dennoch praktische Bedeutung besitzt.

Beispiel für einfaches Schätzverfahren

Beispiel 1.11

Den Ausgangspunkt der Schätzung des personellen Aufwands bildet der nach der fachlichen Vorstudie und bisherigen Projekterfahrungen zu erwartende quantitative Umfang des Softwareprodukts. Der Umfang wird durch die Anzahl der Quellcodezeilen (lines of code, kurz LOC) gemessen. Dabei werden meist nur Vereinbarungen und ausführbare Anweisungen, keine Kommentare berücksichtigt.

Die Ermittlung der Realisierungszeit aus dem geschätzten Software-Umfang erfolgt ebenfalls auf der Basis früherer Erfahrungen und speziell mit Hilfe gesammelter unternehmensspezifischer Daten zur durchschnittlichen Produktivität eines Entwicklers, die in LOC pro Mitarbeitermonat ausgedrückt wird. Die Realisierungszeit in Mitarbeitermonaten bestimmt sich dann gemäß der Gleichung:

Realisierungszeit = geschätzter Code-Umfang / Entwickler-Produktivität. (1.1)

Gegen dieses simple Schätzverfahren können natürlich diverse Einwände geltend gemacht werden. So hängen der Umfang eines Softwareprodukts wie auch die Entwickler-Produktivität von der gewählten Programmiersprache ab. Ferner wirken neben den Faktoren des Produktumfangs und der Produktivität auch qualitative Anforderungen (etwa gute Wartbarkeit) sowie ggf. Termindruck durch die vorgesehene Realisierungsdauer auf die Realisierungszeit ein. Die Einflussnahme des letztgenannten Faktors erklärt sich dabei aus dem erhöhten Kommunikationsaufwand, der bei verkürzter Entwicklungsdauer aus einem notwendig vergrößerten Projektteam resultiert.

qualifizierte Schätzverfahren

In der Praxis größerer Unternehmen haben sich vor allem zwei Schätzverfahren für die Realisierungszeit kommerzieller Anwendungen etabliert, die alle vier genannten Einflussfaktoren beachten. Dabei handelt es sich um das COCOMO-Verfahren (Constructive Cost Model) sowie die Function-Point-Methode. Die Function-Point-Methode zeichnet sich besonders dadurch aus, dass der Umfang eines Produkts nicht durch die erwartete LOC-Anzahl ausgedrückt, sondern unabhängig von der verwendeten Programmiersprache aus den Produktanforderungen bestimmt wird. Für eine Darstellung der zuletzt genannten sowie weiterer Aufwandsschätzverfahren seien interessierte Leser auf KNÖLL und BUSSE (1991) verwiesen.

weitere Schätzgrößen

Aus dem personellen Aufwand sowie den erforderlichen sachlichen Ressourcen lassen sich weitere Größen abschätzen. Hierzu gehören der Personalbedarf bei vorgegebener Projektdauer, die Projektdauer bei vorgegebenem Personalaufkommen sowie die Projektkosten. Wie in Kap. 1.1 erwähnt, stellen die Personalkosten meist den absolut dominierenden Anteil der gesamten Entwicklungskosten dar.

(2) Einschätzung der Durchführbarkeit

Fragen zur Durchführbarkeit

Die Realisierbarkeit eines Projekts wird unter software-technischen wie personellen Gesichtspunkten beurteilt, wobei etwa die folgenden Fragen zu klären sind:

- Ist das Projekt software-technisch machbar, d.h. können die Anforderungen des Auftraggebers auf der vorgesehenen Zielplattform umgesetzt werden?
- Kann das Projekt mit der vorhandenen Qualifikation und Erfahrung der fachlichen Mitarbeiter und des Managements realisiert werden?
- Kann das Produkt mit der gegebenen Entwicklungsplattform und den vorhandenen bzw. beschaffbaren Werkzeugen entwickelt werden?
- Kann das Produkt mit dem verfügbaren Personal innerhalb der vorgegebenen bzw. einer vernünftig veranschlagten Projektdauer erstellt werden?

Die letztgenannte Frage ist natürlich anhand des zuvor ermittelten Personalaufwands zu prüfen.

(3) Einschätzung der Wirtschaftlichkeit

Wirtschaftlichkeit und Auftraggeber-Auftragnehmer-Situation

Die Prüfung der Wirtschaftlichkeit und der Vorteilhaftigkeit eines Projekts wird anhand einer Gegenüberstellung der zu erwartenden Kosten- und Nutzenwirkungen des Produkts vorgenommen. Sie wird maßgeblich durch die Auftraggeber-Auftragnehmer-Situation bestimmt.

Bei einer Entwicklung für eine Fachabteilung des eigenen Unternehmens sind die Projektkosten und der nicht immer monetär bewertbare Projektnutzen gemeinsam mit der Fachabteilung in einer Kosten-Nutzen-Analyse abzuwägen. Beispiele für schwierig oder nicht monetär bewertbare Nutzenarten stellen etwa die schnellere Verfügbarkeit oder die verbesserte Qualität von Informationen dar, die durch das neue Anwendungssystem bereitgestellt werden (vgl. GEHRING 1993a).

Wird für einen externen Auftraggeber oder für den anonymen Software-Markt produziert, so sind nach einer Schätzung der voraussichtlichen Projektkosten Kalkulationen von Preisen oder Deckungsbeiträgen vorzunehmen und ggf. Verhandlungen mit dem Auftraggeber zu führen. Derartige Aufgaben werden großenteils durch projekt-externe Stellen wahrgenommen.

(4) Alternative Lösungsansätze

alternative Lösungen

Die Prüfung alternativer Lösungsansätze bezieht sich häufig auf den Zukauf von Softwarekomponenten anstelle einer Eigenentwicklung („make or buy“-Entscheidung).

Ebenso kommt in Betracht, die Entwicklung einzelner Kernkomponenten des gewünschten Produkts an einen Nachauftragnehmer zu vergeben, der hierfür eine spezielle Kompetenz besitzt.

Die Prüfung der genannten alternativen Lösungsansätze setzt geeignete Kriterienkataloge voraus (vgl. GEHRING 1993a).

(5) Beurteilung der Risiken

Risikobeurteilung

Im Rahmen einer Risikobeurteilung sind die bereits erkennbaren Risiken zu identifizieren, die den Erfolg des Projekts in Frage stellen können. Auf das Risikoma-

nagement als eine ständige Teilaufgabe des Projektmanagements wird weiter unten noch eingegangen (vgl. 1.7.3).

Aus den gewonnenen Untersuchungsergebnissen sind abschließend Empfehlungen zur Projektdurchführung abzuleiten.

1.7.2 Projektplanung

Projektplanung, allgemeine Aufgabe

Die allgemeine Aufgabe der Projektplanung lässt sich angelehnt an KOONTZ und O'DONNELL (1972) wie folgt formulieren: „Planen ist Entscheiden im voraus, was zu tun ist, wie es zu tun ist, wann es zu tun ist und wer es zu tun hat." Die Planung des Projekts bildet die unverzichtbare Basis für die Projektleitung und -kontrolle (vgl. Kap. 1.7.3.).

Voraussetzungen

Die Projektplanung beginnt, nachdem die Entscheidung für ein Projekt gefallen ist. Ihren Ausgangspunkt bilden die Ergebnisse der Durchführbarkeitsuntersuchung, d.h. die fachliche Vorstudie und die Durchführbarkeitsstudie, darunter insbesondere die Projektkalkulation. Grundlage der Projektplanung ist ferner die Auswahl eines Prozessmodells. Dieses beeinflusst zum einen die einzelnen Planungsaktivitäten und zum anderen die zeitliche Verteilung der Projektplanung (vgl. unten). Zur Auswahl eines Prozessmodells gehören auch fachliche Entscheidungen etwa hinsichtlich der zu verwendenden Entwicklungsmethoden und Werkzeuge.

Aktivitäten

Die Projektplanung besitzt sowohl aufbau- wie auch ablauforganisatorische Aspekte und umfasst vor allem folgende Aktivitäten:

- die Bildung des Projektteams,
- die Terminplanung zur Festlegung von Terminen für alle Projektvorgänge,
- die Ressourcenplanung zur Festlegung des zeitabhängigen Einsatzes bzw. Verbrauchs der personellen und sachlichen Ressourcen,
- die Kostenplanung zur Ermittlung der Projektkosten und ihrer zeitlichen Verteilung.

iterative Planung

Wird die Projektplanung auch schwerpunktmäßig bei Projektbeginn geleistet, so ist sie jedoch zugleich eine projektbegleitende und iterative Aufgabe. Dies mögen folgende Beispiele verdeutlichen:

- Das Projekt wird nach dem evolutionären Prozessmodell abgewickelt. Jede neue Version muss dann gesondert geplant werden.
- Nachdem eine oder mehrere Projektphasen durchlaufen wurden, stehen zuverlässigere Grundlagen für die Schätzung von Aufwänden und Ressourcenverbrauch zur Verfügung. Daher wird die Termin- und Ressourcenplanung für die restlichen Phasen des Projekts verfeinert.
- Der Übergang zu einer neuen Projektphase kann eine Vergrößerung, Verkleinerung oder Umstrukturierung des Projektteams erforderlich machen.
- Die Anforderungen an ein Projekt verändern sich wesentlich. Dies kann einerseits auf Initiative des Auftraggebers erfolgen. Möglich ist auch, dass ein Projekt „aus dem Ruder läuft" und der Auftragnehmer in Verhandlungen eine Reduzierung der Anforderungen durchsetzt.

Bevor die oben genannten Planungsaktivitäten im Einzelnen betrachtet werden, sei ausdrücklich auf ihre starke gegenseitige Abhängigkeit hingewiesen. Daher werden hier auch die Termin- und Ressourcenplanung gemeinsam vorgestellt.

(1) Bildung des Projektteams

Projektteam bilden

Die Bildung des Projektteams umfasst zunächst die Einsetzung der Projektleitung und die Auswahl der fachlichen Projektmitarbeiter.

Kriterien

Die Auswahl der Mitarbeiter erfolgt nach dem absehbaren Personalbedarf in quantitativer wie qualitativer Hinsicht. Benötigt werden u.a. Systemanalytiker, Entwurfsarchitekten und Implementierer. Die Entwicklung verläuft in Richtung einer zunehmenden Differenzierung der erforderlichen Qualifikationen. So werden heute z.B. Spezialisten für die Erstellung von grafischen Benutzungsoberflächen benötigt. Die Gewinnung einiger weniger hochqualifizierter Mitarbeiter ist erfahrungsgemäß ein Schlüssel für den Projekterfolg.

Subteams

Bereits im Zuge der Bildung des Projektteams oder auch zu späteren Zeitpunkten werden bei großen Projekten ggf. auch Subteams gebildet und besondere Kompetenzen und Verantwortlichkeiten geregelt, die sich z.B. auf die Aufrechterhaltung einer intensiven Kommunikation mit dem Auftraggeber beziehen können. Ebenso müssen Festlegungen zum Kontroll- und Berichtswesen getroffen werden.

Die Aufgabe von Subteams hängt u.a. von dem gewählten Prozessmodell ab. Typisch für wasserfallartige Modelle sind Subteams, die mit den grundlegenden Entwicklungsaktivitäten korrespondieren; so wird es z.B. ein Analyseteam geben. Typisch für eine evolutionäre Entwicklung sind Subteams, die für einige oder alle Systemkomponenten einer Produktversion verantwortlich sind und hierbei sowohl die Analyse, als auch den Entwurf und die Implementierung übernehmen. Dabei arbeiten verschiedene Subteams oft gleichzeitig an verschiedenen Versionen bzw. Produktstufen. Subteams sollten möglichst klein sein, um den Kommunikationsaufwand in Grenzen zu halten. Eine übliche Größe beträgt etwa 3 bis 4 Personen.

(2) Termin- und Ressourcenplanung

Termin- und Ressourcenplanung, Vorgangsgliederung

Um eine Termin- und Ressourcenplanung durchzuführen, wird zunächst die gesamte Projektentwicklung in Vorgänge gegliedert. Ein Vorgang ist eine Aktivität, die in einem überschaubaren Zeitraum (einige Tage bis wenige Wochen) ein definiertes, abgeschlossenes Ergebnis hervorbringt. Beispiele sind etwa die Erstellung eines Prototyps der Benutzungsoberfläche oder eines Pflichtenheftes. Die Zerlegung der gesamten Entwicklungsaktivität in Vorgänge basiert auf dem gewählten Prozessmodell und seinen Phasen.

Ziele

Die Termin- und Ressourcenplanung zielt unmittelbar darauf ab, allen Vorgängen Start- und Endtermine sowie Ressourcen zuzuordnen. Die Terminplanung für alle Vorgänge schließt von selbst eine Ablauf- bzw. Reihenfolgeplanung der Entwicklungstätigkeit ein. Unter Ressourcen werden das Personal und die Betriebsmittel (Hardware, Werkzeuge usw.) zusammengefasst. Den Vorgängen werden konkrete Mitarbeiter zugeteilt.

Anforderungen, Gegebenheiten

Bei der Termin- und Ressourcenplanung sind etwa folgende Anforderungen und Gegebenheiten zu berücksichtigen:

- Die gesamte Projektdauer soll möglichst gering gehalten werden. Externe Terminvorgaben des Auftraggebers für das Projektende oder den Abschluss einzelner Entwicklungsphasen bzw. -aktivitäten sind ggf. einzuhalten.
- Die Kapazität der Ressourcen sollte nicht überschritten werden. Dabei ist insbesondere die zeitliche Verteilung der Verfügbarkeit des Personals zu beachten. Auch die Beanspruchung von Betriebsmitteln durch mehrere Subteams oder mehrere Projekte kann eine Rolle spielen.
- Die verfügbaren Ressourcen sollten mehr oder minder gleichmäßig ausgelastet werden. Neben Engpässen sollte auch „Leerlauf“ vermieden werden.
- Ein wesentliches Mittel zur Erreichung der genannten Anforderungen besteht in der simultanen Abwicklung von Vorgängen. Im Allgemeinen können sowohl Vorgänge einer Entwicklungsphase wie auch Vorgänge verschiedener Phasen zeitlich parallelisiert werden.
- Die Parallelisierung von Vorgängen wird jedoch zum einen durch die zum jeweiligen Zeitpunkt verfügbaren Ressourcen beschränkt. Sie findet ferner ihre Grenze in sachlichen und daraus resultierenden zeitlichen Abhängigkeiten zwischen Vorgängen. Diese werden auch Vorgangsbeziehungen genannt. Zum Beispiel kann die fachliche Modellierung nicht beginnen, bevor mit der Erstellung des Pflichtenheftes begonnen wurde. Gefordert werden kann etwa auch, dass die Modellierung erst eine Woche nach Beginn der Erstellung des Pflichtenheftes startet. Die verwendete Entwicklungsmethode kann ebenso vorschreiben, dass ein Prototyp der Benutzungsoberfläche erst begonnen wird, nachdem das Analysemodell fertig vorliegt, d.h. nachdem die zugehörigen Vorgänge beendet wurden.
- Wie die Beispiele zeigen, gibt es verschiedene Typen von Vorgangsbeziehungen, die den Start oder das Ende einer Aktivität A an den Start bzw. das Ende einer Aktivität B binden. Ist ein Vorgang A an den Vorgang B sachlich und zeitlich gebunden, so heißt A Nachfolger von B und B Vorgänger von A. Ein Vorgang kann mehrere Vorgänger und mehrere Nachfolger besitzen.

Meilensteine

- Zur Unterstützung der Fortschrittskontrolle des Projekts werden zusätzlich zu den Vorgängen Meilensteine eingeplant. Diese markieren Zeitpunkte, an denen ein oder mehrere ausgewählte Vorgänge planmäßig abgeschlossen sein sollen. Meilensteine werden z.B. am Ende von Entwicklungsphasen vorgesehen. Sie sollten aber in kürzeren Abständen und möglichst gleichmäßig verteilt über die gesamte Projektdauer platziert werden. Bei dem Erreichen eines Meilensteines wird ein Soll-Ist-Vergleich vorgenommen. Dieser prüft, ob die seit dem letzten Meilenstein zu erbringenden Arbeitsergebnisse fertiggestellt wurden.

Netzplantechnik

Für die Termin- und Ressourcenplanung industrieller Projekte werden sehr oft Methoden und Werkzeuge der Netzplantechnik verwendet. Hier sei nur deren Prinzip angedeutet. Interessierte Leser seien auf DOMSCHKE und DREXL (1995) verwiesen. Die Abb. 1.8 zeigt ein grobes Ablaufschema der Termin- und Ressourcenplanung, das die oben besprochenen Anforderungen und Gegebenheiten beachtet und einem Werkzeugeinsatz zugrunde gelegt werden kann.

Schema einer Termin- und Ressourcenplanung

Grober Ablauf einer Termin- und Ressourcenplanung

(1) Gliedere das Projekt in Vorgänge mit definiertem Ergebnis.

(2) Ermittle den zeitlichen Aufwand aller Vorgänge. Nutze hierbei u.a.:
- den kalkulierten Gesamtaufwand des Projekts,
- Erfahrungswerte zum Aufwandsanteil von Vorgängen am Gesamtprojekt,
- Aufwandsschätzverfahren.

(3) Ordne allen Vorgängen personelle und sachliche Ressourcen zu.

(4) Ermittle pro Vorgang aus Aufwand und Ressourcen die Vorgangsdauer.

(5) Ermittle bzw. definiere alle (zeitlichen) Vorgangsbeziehungen.

(6) Vorwärtsrechnung: Ermittle unter Beachtung der Vorgangsbeziehungen die frühesten Start- und Endtermine aller Vorgänge inkl. frühestem Projektende.

(7) Rückwärtsrechnung: Ermittle unter Beachtung der Vorgangsbeziehungen und des zuvor ermittelten frühesten Projektendetermins die spätesten Start- und Endtermine aller Vorgänge.

(8) Lege für alle nicht-kritischen Vorgänge den Starttermin fest.

(9) Prüfe und korrigiere den erhaltenen Netzplan unter folgenden Aspekten:
- Einhaltung gegebener Endtermine für das Projekt bzw. einzelne Vorgänge,
- Kapazitätsüberschreitungen bei Ressourcen zu gewissen Zeitpunkten,
- gleichmäßige Ressourcenauslastung im Zeitverlauf.

Nutze hierbei folgende Möglichkeiten:
- zeitliche Verschiebung von Vorgängen, insbes. nicht-kritischer Vorgänge,
- Modifikation der Ressourcenzuordnung zu ausgewählten Vorgängen,
- Einbindung zusätzlicher Projektressourcen,
- Modifikation vorgegebener Termine und/oder Anforderungen.

(10) Füge geeignete Meilensteine in den Netzplan ein.

(11) Gib den Netzplan als Diagramm und/oder tabellarisch aus.

Abb. 1.8. Grobes Ablaufschema der Termin- und Ressourcenplanung.

Erläuterung des Schemas

Einige Schritte des Ablaufschemas bedürfen noch einer zusätzlichen Erläuterung:

- Besitzt ein Vorgang mindestens einen Vorgänger, so ergibt sich sein frühester Starttermin aus den Start- und Endterminen aller Vorgänger unter Beachtung der Vorgangsbeziehungen (vgl. das folgende Beispiel). Der früheste Endtermin eines Vorgangs ermittelt sich als Summe des frühesten Starttermins und der Vorgangsdauer. Die Vorwärtsrechnung zur Bestimmung aller frühesten Termine beginnt mit einem stets vorhandenen Vorgang ohne Vorgänger (Startvorgang). Dessen frühester Starttermin fällt mit dem Projektbeginn zusammen und wird geeignet gewählt. Anschließend werden iterativ jeweils die frühesten Termine eines Vorgangs bestimmt, für dessen sämtliche Vorgänger die frühesten Termine bereits berechnet wurden.

Beispiel 1.12

Zu bestimmen sei der früheste Starttermin des Vorgangs A. Dieser besitze die Vorgänger B und C. Die Vorgangsbeziehungen lauten:

- A darf erst unmittelbar nach dem Ende von B beginnen und
- A darf erst einen Tag nach dem Anfang von C beginnen.

Der früheste Starttermin von A berechnet sich dann gemäß:

frühester Starttermin von A : =
Maximum(frühester Endtermin von B, frühester Starttermin von C + 1).

- Die Rückwärtsrechnung verläuft analog zur Vorwärtsrechnung. Der späteste Endtermin eines Vorgangs ergibt sich aus den spätesten Terminen aller Nachfolger unter Beachtung der Vorgangsbeziehungen. Die spätesten Starttermine ermitteln sich aus den spätesten Endterminen durch Subtraktion der Vorgangsdauer. Die Rückwärtsrechnung geht von einem wiederum stets vorhandenen Vorgang ohne Nachfolger (Zielvorgang) aus. Der späteste Endtermin des Zielvorgangs wird mit dem in der Vorwärtsrechnung berechneten frühesten Endtermin desselben Vorgangs identifiziert, der zugleich das früheste Projektende markiert. Nach Bearbeitung des Zielvorgangs wird stets ein Vorgang gewählt, dessen sämtliche Nachfolger bereits bearbeitet wurden.

kritische Vorgänge

- Kritische Vorgänge lassen sich dadurch definieren, dass der früheste und späteste Endtermin zusammenfallen. Sie sind insofern von besonderem Interesse, als eine tatsächliche zeitliche Verschiebung eines kritischen Vorgangs den geplanten Projektendtermin in Frage stellt. Nicht kritische Vorgänge lassen sich dagegen in Grenzen ohne terminliche Auswirkung auf andere Vorgänge zeitlich verschieben. Ihr Starttermin (aus dem der Endtermin folgt) muss daher auch nach der Rückwärtsrechnung noch gesondert festgelegt werden.
- Im Ablaufschema der Abb. 1.8 werden erst nach einer ersten Ressourcenzuteilung und Vorgangsterminierung eventuelle Verletzungen einzuhaltender Bedingungen (wie vorgegebene Endtermine) festgestellt und behoben. Dabei kann es erforderlich werden, die Netzplanberechnung ganz oder teilweise zu wiederholen, was in der Abbildung nicht dargestellt wurde.

Visualisierung von Netzplänen

- Netzpläne können auf verschiedene Weise visualisiert werden. In vielen Fällen genügen einfache Balken-Diagramme, auch Gantt-Diagramme genannt. In ihnen werden die Vorgänge als horizontale Balken abgebildet, deren Länge proportional zur Vorgangsdauer ist. Die Vorgangsbalken werden oberhalb einer Zeitachse entsprechend ihren Anfangsterminen angeordnet. Die Zuordnung von Mitarbeitern oder Betriebsmitteln zu Vorgängen wird oft in tabellarischer Form ausgegeben.

(3) Kostenplanung

Kostenplanung

Analog zur Termin- und Ressourcenplanung wird auch in der Kostenplanung die Projektkalkulation der Durchführbarkeitsuntersuchung verfeinert. Nun geht es darum, neben einer genaueren Gesamtkostenschätzung die zeitliche Verteilung der Kosten über den gesamten Projektablauf zu ermitteln.

Die Kostenplanung basiert auf den Ergebnissen der Termin- und Ressourcenplanung und erfolgt bottom up. So werden zunächst den einzelnen Vorgängen entsprechend den zugeteilten Ressourcen und der Vorgangsdauer Kosten zugeordnet. Hieraus ergeben sich die geplanten Kosten umfassenderer Entwicklungsaktivitäten, insbesondere der Projektphasen.

Da die Vorgänge sowie die umfassenderen Entwicklungsaktivitäten bereits terminiert sind, erhält man auf diese Weise auch die geplante zeitliche Entwicklung der kumulativen Projektkosten. Diese bildet eine wesentliche Grundlage für die Projektkontrolle.

1.7.3 Projektleitung und Projektkontrolle

Projektleitung und -kontrolle

Die Projektleitung und -kontrolle beinhalten zahlreiche Management-Aktivitäten während der Projektdurchführung. Einige seien kurz angesprochen. Auf das Qualitätsmanagement wird im Kap. 1.8 näher eingegangen.

(1) Führung von Mitarbeitern

Mitarbeiter führen

Der Führungsstil der Projektleitung sollte davon geprägt sein, dass es um die Leitung hochqualifizierter Mitarbeiter geht, die komplizierte fachliche Aufgaben zu bewältigen haben. Das Projektmanagement ist gut beraten, wenn es auf gegenseitiges Vertrauen und Eigenverantwortung anstelle von Bevormundung und kleinlicher Überwachung setzt.

Hierzu gehören Freiräume bei der Gestaltung des Arbeitsablaufs ebenso wie ein Mitspracherecht der Mitarbeiter bei der Teambildung und großzügige Arbeitszeitregelungen. Mehr und mehr wird dazu übergegangen, dem Projektteam oder Subteams eine Teilautonomie zu gewähren. Dabei wird die Tätigkeit vorwiegend am Ergebnis gemessen, während der einzuschlagende Weg vom Team weitgehend selbständig bestimmt wird.

Es sollte ein Anliegen der Projektleitung sein, die Kommunikation im Team, den „Teamgeist“ und die Kreativität aller Mitarbeiter zu fördern. Hierzu tragen gemeinsame Diskussionen bei, die allen Beteiligten das Gefühl vermitteln, dass ihre Vorschläge und Ansichten gefragt sind und die nicht von wenigen „Stars“ dominiert werden.

Allerdings bleibt es die Aufgabe der Projektleitung, die Ausrichtung des Projektteams auf die übergeordneten Projektziele und die Koordinierung aller Entwicklungstätigkeiten zu gewährleisten. Ein wesentliches Mittel hierfür bilden projektinterne oder unternehmensweite Standards, die die Anwendung von Prozessmodellen, Methoden und Werkzeugen regeln.

(2) Zusammenarbeit mit dem Auftraggeber

Kommunikation mit Auftraggeber, Benutzern und Experten

Das Erfordernis einer intensiven und funktionierenden Zusammenarbeit und Kommunikation mit dem Auftraggeber wurde bereits wiederholt betont. Hierbei sollten auch die künftigen Nutzer des Produkts einbezogen werden, die schließlich über dessen Akzeptanz mitentscheiden. Oft werden auch fachliche Experten der Auftraggeberseite für gewisse Zeit in das Projektteam eingebunden, um komplizierte fachliche Anforderungen zu präzisieren.

Sicherlich viel Erfahrung und Fingerspitzengefühl verlangt der Umgang mit späteren Änderungswünschen des Auftraggebers während der Projektbearbeitung. Hier geht es darum, jeweils einen für beide Seiten akzeptablen Interessenausgleich zu finden. Änderungswünsche und ihre Bearbeitung müssen wegen ihrer Auswirkungen auf Termine und Kosten jederzeit rekonstruierbar sein und daher geeignet protokolliert werden.

(3) Kontroll- und Berichtswesen leiten

Fortschritts- und Terminkontrolle

Die Projektkontrolle umfasst vor allem die Fortschritts- und Terminkontrolle, die Überprüfung der Kostenentwicklung sowie die Qualitätskontrolle.

Zu diesem Zweck erfolgen regelmäßige Soll-Ist-Vergleiche, die auf den entsprechenden Planungsunterlagen und insbesondere auf festgelegten Meilensteinen (siehe oben) basieren. Im Ergebnis der schriftlich zu protokollierenden Soll-Ist-Vergleiche sind ggf. korrigierende Aktivitäten einzuleiten.

Dabei kommt es darauf an, zwischen vorübergehenden Schwierigkeiten und Defiziten einerseits und tieferliegenden Problemen andererseits zu differenzieren. Die ersteren können durch kurzfristige Maßnahmen wie z.B. eine temporäre Personalaufstockung behoben werden. Letztere erfordern eingehende Analysen sowie evtl. weitreichende Maßnahmen wie etwa die Initiierung einer Überarbeitung der Softwarearchitektur.

Auf die Schwierigkeiten, die sich bereits der Ermittlung des tatsächlichen Ist-Zustandes eines Projekts in den Weg stellen, wurde schon früher hingewiesen. Neben vertrauensvollen Beziehungen zwischen Projektleitung und -team trägt vor allem die Wahl eines geeigneten Prozessmodells dazu bei, dieses grundlegende Problem wenigstens abzumildern. Die evolutionäre Softwareentwicklung mit frühzeitig lauffähigen Produktversionen ist grundsätzlich besser geeignet, bösen Überraschungen vorzubeugen als ein wasserfallartiges Modell.

Die Auswahl des Prozessmodells kann sich daher als eine erstrangige Maßnahme des nun abschließend zu besprechenden Risikomanagements erweisen.

(4) Risikomanagement

Risikomanagement

Softwareprojekte sind in höherem Maße mit Risiken verbunden als Entwicklungsvorhaben vieler anderer technischer Disziplinen. Gefahren für ein Softwareprojekt rechtzeitig zu erkennen und durch geeignete Maßnahmen abzuwenden, ist eine Kernaufgabe des Projektmanagements und spezieller Gegenstand des Risikomanagements.

Dieses beginnt bereits bei der Erarbeitung der Durchführbarkeitsstudie und der Projektplanung, so etwa bei der Auswahl und projektspezifischen Anpassung des Prozessmodells (vgl. Kap. 1.7.1 und 1.7.2). Das Risikomanagement stellt jedoch vor allem eine ständige, das ganze Projekt begleitende Herausforderung dar.

Top-Ten-Risiken

In einer Untersuchung von BOEHM (1991), die auf einer Umfrage bei erfahrenen Projektleitern basierte, wurden zehn immer wieder beobachtete Risiken für den Erfolg von Softwareprojekten identifiziert. Diese sogenannten Top-Ten-Risiken der Softwareentwicklung sind in der Tab. 1.3 aufgeführt.

Nr.	Risiko
①	Mangelnde Qualifikation des Personals.
②	Unrealistische Termin- und Kostenvorgaben.
③	Entwicklung von falschen Funktionen und Eigenschaften.
④	Entwicklung der falschen Benutzungsoberfläche.
⑤	„Vergolden“, d.h. über das Ziel hinausschießen.
⑥	Kontinuierliche Anforderungsänderungen.
⑦	Defizite bei extern erstellten und gelieferten Komponenten.
⑧	Defizite bei extern erledigten Aufträgen.
⑨	Defizite in der Echtzeitleistung.
⑩	Überfordern der Software-Technik.

Tab. 1.3. Die Top-Ten der Risiken in der Softwareentwicklung.

Natürlich muss das Risikomanagement dem Umstand Rechnung tragen, dass jedes Projekt individuelle Risiken besitzt bzw. die aufgeführten Top-Ten-Risiken sowie weitere Risiken jeweils mit unterschiedlichem Gewicht auftreten. Auch die Behandlung erkannter Risiken kann nicht schematisch erfolgen.

Aufgaben

Das Risikomanagement umfasst zwei Hauptaufgaben:

a) Risikobewertung
Die Risikobewertung beinhaltet die Identifikation, Analyse und Prioritätenbildung der Projektrisiken. Bei der Analyse geht es um die Abschätzung der Eintrittswahrscheinlichkeit und des möglichen Schadensumfangs aller erkannten Risiken. Die auf der Analyse fußende Prioritätenbildung der Risiken gestattet es, sich auf die Behandlung der wichtigsten Risiken des Projekts zu konzentrieren. Die Fokussierung auf die wichtigsten Risiken wird durch eine Untersuchung von BOEHM (1989) gestützt. Ihr zufolge werden 80% der Kosten für eine Überarbeitung von Software dazu aufgewandt, um 20% der Fehler zu beseitigen.

b) Risikobeherrschung
Unter Risikobeherrschung werden die Risikomanagement-Planung, die Risikoüberwindung und die Risikoüberwachung zusammengefasst. Die Risikomanagement-Planung legt Maßnahmen zur Risikobehandlung fest, die dann im Rahmen der Risikoüberwindung durchgeführt werden. Dabei kann es sich z.B. um die Erstellung von Prototypen und die nachträgliche, mit dem Auftraggeber abgestimmte Reduzierung von Anforderungen handeln. Die Risikoüberwachung kontrolliert die Fortschritte bei der Risikoreduzierung im Projektablauf.

Übungsaufgaben zu Kapitel 1.7

Übungsaufgabe 1.7.1

Bei der Konkretisierung von Prozessmodellen in Unternehmen werden u.a. vorgegebene Phaseneinteilungen verfeinert und anzuwendende Methoden, Werkzeuge und z.B. Programmierstandards festgelegt. Nennen Sie einige Vorteile einer einheitlichen, d.h. projektübergreifenden Definition von Prozessmodellen in einem Unternehmen.

Übungsaufgabe 1.7.2

Können Sie zusätzlich zu den bereits genannten Faktoren einen weiteren Faktor nennen, der den personellen Aufwand eines Projekts maßgeblich beeinflusst? Denken Sie daran, dass das „Rad“ ja nicht immer neu erfunden werden muss.

Übungsaufgabe 1.7.3

Eine wichtige Anforderung an Meilensteine lautet, dass diese überprüfbar sein müssen. Entscheiden Sie entsprechend dieser Anforderung, welche der folgenden Meilensteine sinnvoll formuliert wurden und begründen Sie kurz Ihre Entscheidungen. Die Beispiele basieren teils wieder auf der Übungsaufgabe 1.3.2.

a) Die Klassen bzw. Programmbausteine Kunde, Hotel und Reise sind vollständig implementiert und als einzelne Bausteine getestet.
b) Noch 8 Tage bis zur Fertigstellung des Pflichtenheftes.
c) Alle Testfälle für den Integrationstest der Anwendungskomponente „Reise buchen“ sind komplett abgeleitet und dokumentiert.
d) Die Anwendungskomponente „Reise abrechnen“ ist zu 80% kodiert.

Übungsaufgabe 1.7.4

Im Abschnitt „Kontroll- und Berichtswesen leiten“ von Kap. 1.7.3 wurde als eine Maßnahme zur Behebung von nicht gravierenden Schwierigkeiten eines Projekts eine vorübergehende Erweiterung des Projektteams genannt. Obwohl diese Maßnahme grundsätzlich anwendbar ist, kann sie sich auch als kontraproduktiv erweisen. Warum kann dies der Fall sein? Beachten Sie die Aussagen zu den Faktoren, die den personellen Aufwand eines Projekts bestimmen, im Kap. 1.7.2.

1.8 Qualitätssicherung

Qualität, Qualitätssicherung

Unter Software-Qualität versteht man laut Industrienorm DIN ISO 9126 die Gesamtheit der Merkmale und Merkmalswerte eines Software-Produkts, die sich auf dessen Eignung beziehen, festgelegte oder vorausgesetzte Erfordernisse zu erfüllen. Die Qualität von Software-Produkten wird mit Hilfe differenzierter Qualitätskriterien beurteilt. Diese stellen eine Grundlage der Qualitätssicherung dar. Die Software-Qualitätssicherung umfasst alle Maßnahmen, die der Gewährleistung der Software-Qualität dienen.

Die Qualitätssicherung beinhaltet einerseits Maßnahmen des Managements, die auch unter dem Begriff Qualitätsmanagement subsumiert werden. Zur Qualitätssicherung gehören andererseits fachliche Aufgaben, die von Entwicklern und anderen Fachkräften, etwa spezialisierten Software-Testern, wahrzunehmen sind.

Qualitätssicherung: konstruktiv oder analytisch

Die Qualitätssicherung in einem Projekt bezieht sich in erster Linie auf das Produkt selbst. Unterschieden wird zwischen der konstruktiven und der analytischen Qualitätssicherung. Die konstruktive Qualitätssicherung zielt auf die vorausschauende Fehlervermeidung ab. Die analytische Qualitätssicherung beschäftigt sich mit der Identifizierung von Fehlern und Schwächen der Entwicklungsergebnisse. Fehler stellen Abweichungen von Entwicklungsergebnissen gegenüber ihrer Spe-

zifikation bzw. Sollbeschreibung in Referenzdokumenten wie z.B. dem Pflichtenheft dar. Auch Abweichungen von den intendierten Vorstellungen des Auftraggebers gelten als Fehler.

Qualität der Entwicklung

Die Güte des Entwicklungsprozesses wirkt auf die Qualität des Produkts zurück. Die Qualitätssicherung bezieht sich daher auch auf den Software-Entwicklungsprozess als solchen. Die Sicherung der Qualität des Entwicklungsprozesses ist insbesondere auch Aufgabe des projektübergreifenden Softwaremanagements und findet ihren Niederschlag in unternehmensweiten Standards.

Grundsätze der Qualitätssicherung

Die produktbezogene Qualitätssicherung basiert auf verschiedenen, aus der Erfahrung gewonnenen Grundsätzen :

(1) Entwicklungsbegleitende Qualitätssicherung
Die Qualität des Produkts ist parallel zur gesamten Entwicklung zu sichern. Ein abschließender Test des fertigen Softwaresystems genügt keineswegs. Jedes Ergebnis der Analyse und des Entwurfs ist mittels geeigneter Verfahren sofort einer Prüfung zu unterziehen. Nur auf diese Weise können Fehler frühzeitig erkannt werden. Dies wiederum ist von immenser Bedeutung für die Projektkosten, weil der Kostenaufwand für die Fehlerbehebung exponentiell mit dem Zeitpunkt der Fehlererkennung wächst. Betrachtet sei z.B. ein während der Analyse gemachter Fehler. Nach BOEHM (1984) betragen die Kosten der Fehlerbeseitigung – bezogen auf den Analysezeitraum – während des Entwurfs das 2½ – 5 fache, während der Implementierung das 5 – 10 fache und während der Wartung das 100 – 200 fache.

(2) Maximale konstruktive Qualitätssicherung
Die konstruktive Qualitätssicherung folgt der Devise „Vorbeugen ist besser als Heilen“ und sollte daher in größtmöglichem Umfang angewandt werden. Felder und Aktivitäten der konstruktiven Qualitätssicherung werden später besprochen (vgl. Kap. 1.8.2).

(3) Unabhängigkeit von Entwicklung und Qualitätssicherung
Die Entwicklung und Qualitätssicherung eines bestimmten Ergebnisses sollte nicht durch dieselben Personen erfolgen. Ein Entwickler, der ein Teilprodukt erstellt hat, ist häufig blind für dessen Fehler. Außerdem werden Fehler und Schwächen nicht gern zugegeben. Das betreffende Teilprodukt sollte daher durch andere Entwickler oder spezielle Testspezialisten geprüft werden. Allerdings muss vermieden werden, dass die Entwickler ihre Zuständigkeit für die Produktqualität auf jeweils andere Personen abwälzen. Dies kann geschehen, indem bereits die Entwickler für die Einhaltung gewisser Qualitätsanforderungen verantwortlich gemacht werden, deren Einhaltung nachzuweisen ist.

Die folgenden Abschnitte gehen auf einige oben angesprochene Aspekte der Qualitätssicherung näher ein.

1.8.1 Qualitätskriterien für Software

Qualitätskriterien nach DIN ISO 9126 ...

Zur Beurteilung der Qualität von Software werden differenzierte Qualitätskriterien herangezogen, die mit verschiedenen Sichten und Interessenlagen korrespondieren. Zu unterscheiden sind in erster Linie die Sicht des Auftraggebers und der

späteren Benutzer einerseits sowie die Sicht des Softwareherstellers und der Entwickler andererseits.

... ihre Definition und ...

In der folgenden Tab. 1.4 werden die von der Norm DIN ISO 9126 vorgesehenen grundlegenden sechs Qualitätskriterien vorgestellt (vgl. DIN ISO 9126, 1991).

Qualitätskriterium	Allgemeine Definition nach DIN ISO 9126
Funktionalität	Vorhandensein von Funktionen mit festgelegten Eigenschaften. Diese Funktionen erfüllen die definierten Anforderungen.
Zuverlässigkeit	Fähigkeit der Software, ihr Leistungsniveau unter festgelegten Bedingungen über einen festgelegten Zeitraum zu bewahren.
Benutzbarkeit	Aufwand, der zur Benutzung erforderlich ist, und individuelle Beurteilung der Benutzung durch eine festgelegte oder vorausgesetzte Benutzergruppe.
Effizienz	Verhältnis zwischen dem Leistungsniveau der Software und dem Umfang der eingesetzten Betriebsmittel unter festgelegten Bedingungen.
Änderbarkeit	Aufwand, der zur Durchführung vorgegebener Änderungen notwendig ist. Änderungen können Korrekturen oder Anpassungen an Änderungen der Umgebung, der Anforderungen und der funktionalen Spezifikationen einschließen.
Übertragbarkeit	Eignung der Software, von einer Umgebung in eine andere übertragen zu werden. Umgebung kann organisatorische, Hardware- oder Software-Umgebung einschließen.

Tab. 1.4. Grundlegende Qualitätskriterien nach DIN ISO 9126.

... Erläuterung

Die grundlegenden Qualitätskriterien werden in der Norm DIN ISO 9126 mittels abgeleiteter Kriterien noch konkretisiert, worauf hier nicht im Einzelnen eingegangen sei. Stattdessen sollen folgende Erläuterungen, die teils auch andere gebräuchliche Kriterien einbeziehen, zum Verständnis der grundlegenden Qualitätskriterien beitragen. Definitorische Feinheiten stehen dabei nicht im Vordergrund.

Funktionalität, Zuverlässigkeit, Robustheit

- Die Kriterien der Funktionalität und Zuverlässigkeit stellen vor allem auf die Umsetzung der intendierten Auftraggeberanforderungen, die Korrektheit erzeugter Ergebnisse und die Robustheit von Software ab. Ein Softwaresystem gilt als robust, wenn es – insbesondere bei fehlerhaften Eingaben – nicht unkontrolliert abstürzt. Software ist umso zuverlässiger, je geringer die Häufigkeit von Fehlern und Ausfällen ist und je geringer die Auswirkungen von Fehlern sind. Erhebliche Auswirkungen können z.B. bei Datenverlusten oder falschen Ergebnissen auftreten, die nicht sofort als solche erkennbar sind.

Benutzungsfreundlichkeit

- Die Benutzbarkeit oder auch Benutzungsfreundlichkeit von Software umfasst vor allem die Aspekte der leichten Erlern- und Bedienbarkeit. Komfortable Benutzungsoberflächen entsprechen ergonomischen Kriterien und unterstützen eine intuitive Bedienung (vgl. Kap. 5). Von wesentlicher Bedeutung für die Benutzungsfreundlichkeit sind ferner eine vollständige, präzise und verständliche Benutzerdokumentation und die Verfügbarkeit von (online-)Hilfefunktionen.

Effizienz

- Die Effizienz umfasst die Laufzeit- und Speichereffizienz. Die Laufzeiteffizienz ist in Echtzeitanwendungen etwa zur Überwachung technischer Prozesse von besonderer Bedeutung, spielt jedoch auch in gewöhnlichen kommerziellen Produkten etwa wegen der erforderlichen geringen Dialogantwortzeiten eine Rolle.

- Die Kriterien der Änderbarkeit und Übertragbarkeit können auch zum Kriterium der Wartbarkeit zusammengefasst werden. Die bei der Wartung anfallenden Aufgaben (Fehlerkorrektur, Portierung, Erweiterung und Verbesserung der Funktionalität) wurden bereits in Kap. 1.3 erläutert. Empirischen Untersuchungen zufolge betragen die Wartungskosten aufgrund der langen Systemlebenszeiten durchschnittlich etwa das Doppelte der ursprünglichen Entwicklungskosten eines Softwaresystems (vgl. PAGEL und SIX 1994). Viele Softwareunternehmen sind zum größten Teil mit Wartungsaufgaben beschäftigt. Auf die Bedeutung, die die Wartbarkeit bei der Verwendung eines evolutionären Prozessmodells für die Systementwicklung selbst besitzt, wurde in Kap. 1.4.2 hingewiesen. Die Wartbarkeit eines Softwaresystems hängt entscheidend vom Entwurf, aber auch von der Implementierung der Systemkomponenten sowie von der Qualität der Systemdokumentation ab (vgl. Kap. 1.3). Die Systemdokumentation bildet die Basis aller Wartungsaktivitäten, sollte die Entwicklungsdokumente aller Phasen entsprechend dem jeweils aktuellen Stand enthalten und muss daher ebenfalls gewartet werden.

Wartbarkeit, Änderbarkeit, Übertragbarkeit

Während die zuvor erläuterten Qualitätskriterien vor allem für den Auftraggeber und die späteren Benutzer von Interesse sind, sind die Wartbarkeit bzw. Änderbarkeit und Übertragbarkeit von Softwaresystemen primär für den Softwarehersteller und die Entwickler von Relevanz. Dabei sei hier unterstellt, dass die Wartung wiederum vom ursprünglichen Auftragnehmer auszuführen ist.

verschiedene Sichten auf Qualität

Schon zu Beginn eines Projekts sollten nicht-funktionale Qualitätsanforderungen explizit vereinbart werden. Bereits grobe Schwerpunktsetzungen wie die Forderung nach einer komfortablen Benutzungsoberfläche können hilfreich sein, reichen allerdings oft nicht aus.

Häufig müssen grobe Qualitätskriterien durch abgeleitete Kriterien untersetzt und verfeinert werden – bis hin zu einer Operationalisierung der Qualität, die zu messbaren und objektiv bewertbaren Qualitätsindikatoren führt. Messbare Qualitätsgrößen werden auch als Metriken bezeichnet. Die skizzierte stufenweise Operationalisierung von Qualitätskriterien wird systematisch innerhalb von Qualitätsmodellen vorgenommen. Ein bekannter Ansatz zur Entwicklung projektspezifischer Qualitätsmodelle, bezeichnet als Goal-Question-Metric-Ansatz, wurde von ROMBACH und BASILI (1987) vorgeschlagen. Ein Beispiel für eine im Entwurf benutzbare Metrik findet sich im Kap. 2.4.1.

Qualitätsmodelle, Metriken

1.8.2 Konstruktive und analytische Qualitätssicherung

Ansatz und Anliegen der konstruktiven Qualitätssicherung wurden bereits benannt. Maßnahmen, die von vornherein sichern, dass ein Produkt bzw. Teilprodukte gewisse Qualitätsmerkmale besitzen, können technischer oder organisatorischer Art sein. Technische Maßnahmen betreffen die Auswahl von Methoden, Sprachen und Werkzeugen. Zu organisatorischen Maßnahmen zählen Standards und Richtlinien. Folgende Beispiele veranschaulichen die konstruktive Qualitätssicherung.

konstruktive Qualitätssicherung

Beispiel 1.13

- Objektorientierte Entwicklungsmethoden gestatten die Anwendung einheitlicher Konzepte in Analyse und Entwurf. Ihr Einsatz erleichtert den Übergang zwischen entsprechenden Projektphasen und trägt zu einer verbesserten Wartbarkeit bei (vgl. Kap. 2).
- Die Wahl einer Sprache ohne goto-Anweisung sichert einen linearen Kontrollfluss in Programmbausteinen und führt zu verständlicherem Programmcode.
- Vorgegebene Gliederungsschemata z.B. für das Pflichtenheft oder die Entwurfsspezifikation leisten einen Beitrag zur Vollständigkeit, Übersichtlichkeit und einheitlichen Gestaltung der Dokumente des Entwicklungsprozesses.

analytische Qualitätssicherung und ...

Maßnahmen der analytischen Qualitätssicherung werden auf die Ergebnisse aller Phasen des Entwicklungsprozesses von der Anforderungsdefinition bis hin zum fertigen Produkt angewandt. Sie sind diagnostischer Art. Ihr Ziel besteht darin, vorhandene Fehler und Schwächen aufzudecken und die Qualität der Ergebnisse unter definierten Aspekten und Kriterien zu beurteilen. Die anschließende Behebung festgestellter Fehler und Probleme ist nicht Gegenstand der analytischen Qualitätssicherung, sondern erfolgt innerhalb der Entwicklung.

... ihre Verfahren

Die analytische Qualitätssicherung umfasst analysierende und testende Verfahren. Für eine umfassende Behandlung der nachfolgend skizzierten sowie weiterer Verfahren sei auf BALZERT (1998) und LIGGESMEIER(1990) verwiesen.

(1) Analysierende Verfahren

analysierende Verfahren

Kennzeichen der analysierenden Verfahren ist, dass das jeweilige Prüfobjekt als solches (Text, Diagramm, Quellcode) untersucht und nicht dynamisch ausgeführt wird.

Reviews

Zu den wichtigsten analysierenden Verfahren gehören Reviews, bei denen Prüfobjekte in einem kleinen, geeignet zusammengesetzten Team manuell analysiert und begutachtet werden. Das Ziel besteht in der Feststellung logischer und inhaltlicher Fehler, Inkonsistenzen, Unvollständigkeiten und Abweichungen von Richtlinien und Standards. Zum Team gehören der Autor des Prüfobjekts, weitere Entwickler und evtl. auch Testspezialisten und künftige Anwender, wenn etwa die Anforderungsdefinition geprüft wird. Reviews finden im Rahmen von geplanten Teamsitzungen statt, erfordern jedoch überwiegend bereits eine vorausgehende Auseinandersetzung der Teilnehmer mit dem Prüfobjekt.

Varianten von Reviews

Reviews existieren in verschiedenen Varianten, die sich hinsichtlich der Reglementierung und Formalisierung unterscheiden. Hier sei kurz auf Inspektionen und Walkthroughs eingegangen.

Inspektionen

Inspektionen sind in starkem Maße formalisiert und werden von einem unabhängigen und ausgebildeten Moderator geleitet. Das Prüfobjekt wird während einer Sitzung von den begutachtenden Teilnehmern schrittweise sowie unter Verwendung von Referenzunterlagen durchgearbeitet, wobei vorzugsweise schwere Defekte aufgefunden werden sollen. Der Moderator ist für die Protokollierung der gefundenen Fehler verantwortlich und kontrolliert auch ihre Behebung nach der Inspektion.

Walkthrough

Wesentlich informeller verläuft ein Walkthrough, bei dem der Autor selbst moderiert und sein Ergebnis schrittweise vorstellt. Die anderen Teilnehmer stellen gezielt Fragen und weisen auf mögliche Fehler und Probleme hin, für deren Beseitigung der Autor selbst verantwortlich bleibt.

Effizienz von Reviews

Ein großer Vorzug von Reviews ist ihre Anwendbarkeit auf Dokumente früher Entwicklungsphasen. Empirische Untersuchungen bezeugen die hohe Effektivität von Reviews. Insbesondere können durch Inspektionen etwa 50 bis 75% aller Entwurfsfehler entdeckt werden (vgl. BALZERT 1998). Dabei ist allerdings zu berücksichtigen, dass Reviews selbst einen nicht zu unterschätzenden Aufwand erfordern, der bis zu 20% der Entwicklungskosten des Prüfobjekts betragen kann.

(2) Testende Verfahren

testende Verfahren

Testende Verfahren führen das Prüfobjekt dynamisch aus und werden daher nur auf implementierte Teilsysteme oder das gesamte Softwaresystem angewendet.

allgemeine Merkmale von Tests

Der Test größerer Programme kann aufgrund der astronomischen Anzahl möglicher Kombinationen von Eingabewerten und der Abhängigkeit der Verarbeitung späterer Eingaben von der Verarbeitung früherer Eingaben nicht vollständig durchgeführt werden. Ein Test wird daher stichprobenartig vorgenommen und kann nicht die Fehlerfreiheit von Programmen beweisen, sondern nur die Anwesenheit von Fehlern aufzeigen. Eben darin besteht auch das Ziel von Tests. Ein Test gilt als umso erfolgreicher, je mehr Fehler er entdeckt.

Für einen Test werden mehrere Testfälle mit geeigneten Eingabedaten gewählt. Die bei korrekter Verarbeitung zu erwartenden Ausgabedaten werden jeweils unabhängig ermittelt. Danach wird das zu testende Programm mit den gewählten Eingabedaten ausgeführt und tatsächliche mit erwarteten Ausgaben verglichen. Zu protokollierende Abweichungen weisen auf später zu beseitigende Fehler hin.

Tests lassen sich in strukturelle und funktionale Tests gliedern.

strukturelle Tests

Strukturelle Tests dienen der systematischen Überprüfung der internen Programmstruktur, setzen also ihre Kenntnis voraus und werden daher auch als white-box-Tests bezeichnet. Ermittelt wird beispielsweise, ob es unerreichbare Anweisungen im Programmcode gibt und wie sich ein Programm beim Durchlaufen selten benutzter Anweisungen verhält.

kontrollfluss- oder datenflussorientiert

Strukturelle Tests werden orientiert am Kontrollfluss oder Datenfluss durchgeführt. Bei kontrollflussorientierten Tests werden Testfälle z.B. derart ausgewählt, dass alle Zweige bei Auswahlanweisungen mindestens einmal betreten werden (Zweigüberdeckung) oder auch derart, dass sämtliche Bedingungen in Auswahlanweisungen alle möglichen Zustände annehmen (Bedingungsüberdeckung). Zur Illustration diene das folgende simple Beispiel 1.14. Datenflussorientierte Tests überprüfen im Wesentlichen Datenbenutzungen. Getestet wird beispielsweise, ob jede Definition einer Variablen in einer Berechnung oder Bedingung benutzt wird.

Beispiele für kontrollfluss-orientierte Tests

Beispiel 1.14

Betrachtet wird folgende zweiseitige Auswahl innerhalb eines C-Programms (der Operator == testet auf Gleichheit, || steht für eine Oder-Verknüpfung):

```
...
zeichen = toupper(zeichen);                    // wandelt ggf. in Großbuchstaben um
if (zeichen == 'A' || zeichen == 'E' || zeichen == 'I' ||    // wenn Vokal vorliegt
   zeichen == 'O' || zeichen =='U')
            printf("\nVokal!");                              // Ausgabe im Zweig 1
else                                                        // sonst
            printf("\nKonsonant!");                         // Ausgabe im Zweig 2
...
```

Zur Prüfung einer Zweigüberdeckung genügen die beiden Testfälle:
'A' und 'B'.

Zur Prüfung der Bedingungsüberdeckung werden folgende 6 Testfälle benötigt:
'A', 'E', 'I', 'O', 'U' und z.B. 'B'.

funktionale Tests

Funktionale Tests ignorieren die interne Programmstruktur und werden daher auch als black-box-Tests bezeichnet. Das System oder einzelne Komponenten werden in diesem Fall ausschließlich gegen ihre Entwurfsspezifikation oder gegen die Anforderungsdefinition getestet.

Für Testfälle werden Eingabewerte zum einen so ausgewählt, dass möglichst alle typischen, d.h. gleichartig zu behandelnden Kombinationen von Eingabewerten repräsentiert werden (Äquivalenzklassenbildung). Oft werden als Eingabedaten auch die Grenzwerte von Wertebereichen gewählt, weil bei deren Behandlung besonders häufig Fehler unterlaufen (Grenzwertanalyse). Zur Veranschaulichung diene wieder ein einfaches Beispiel.

Beispiele für funktionale Tests

Beispiel 1.15

Betrachtet sei die Variable *monat* mit dem zulässigen Wertebereich:
$1 \leq monat \leq 12$.

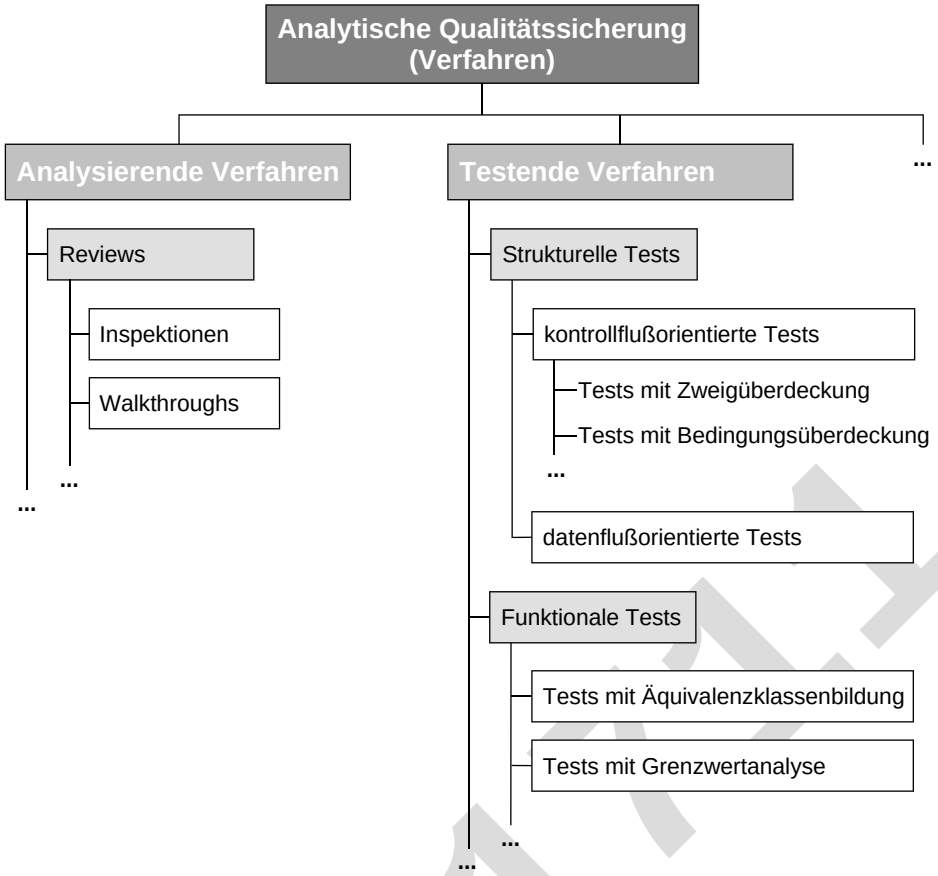
Folgende drei Äquivalenzklassen werden gebildet:
$monat < 1$, $1 \leq monat \leq 12$, $monat > 12$.

Als typische, die Äquivalenzklassen repräsentierende Werte können gewählt werden:
$monat = 0$, $monat = 5$, $monat = 14$.

Als Grenzwerte kommen in Betracht:
$monat = 0$, $monat = 1$, $monat = 12$, $monat = 13$.

Die folgende Abb. 1.9 fasst die hier kurz vorgestellten Verfahren der analytischen Qualitätssicherung zusammen.

Verfahren der analytischen Qualitätssicherung im Überblick



Legende: Auslassungspunkte (**...**) verweisen auf weitere Verfahren.

Abb. 1.9. Einige Verfahren der analytischen Qualitätssicherung.

Testphasen

Beim Test eines Anwendungssystems sind wie in Kap. 1.3 erwähnt im Einzelnen folgende Testphasen zu durchlaufen:

- Komponententest,
- Integrationstest,
- Systemtest und
- Abnahmetest.

Mit Bezug auf die vorherige Darlegung sollen die Testphasen noch etwas näher charakterisiert werden.

Bei dem Test einzelner Komponenten wie auch bei Integrationstests finden sowohl strukturelle wie auch funktionale Verfahren Anwendung.

Komponententests werden meist selbständig durch die jeweiligen Entwickler durchgeführt, während ab der Phase Integrationstest der Grundsatz der personellen Trennung von Entwicklung und Qualitätssicherung einzuhalten ist.

Bei Integrationstests sollten jeweils nur einzelne Komponenten ergänzt werden, um Fehler leichter lokalisieren zu können. Die Reihenfolge der Ergänzung der Komponenten bzw. der Integrationstests ist geeignet zu planen. Ziel der Integrationstests ist vor allem die Überprüfung der Schnittstellen der Komponenten. Um einzelne Komponenten oder auch Teilsysteme testen zu können, werden sie um Testtreiber für ihren Aufruf und ggf. durch sogenannte Teststubs zur Simulation fehlender Komponenten ergänzt.

Bei dem System- und Abnahmetest geht es darum, ob die definierten, im Pflichtenheft fixierten und mit dem Auftraggeber vereinbarten Anforderungen eingehalten werden. Daher werden lediglich funktionale Verfahren eingesetzt. Hierbei kommen insbesondere bereits im Pflichtenheft erfasste Testfälle zur Anwendung. Getestet werden die Gewährleistung der vollständigen Funktionalität sowie die Einhaltung von Leistungsanforderungen wie z.B. vorgegebene Dialogantwortzeiten. Daneben werden weitere nicht-funktionale Qualitätsanforderungen geprüft, die z.B. die Benutzungsoberfläche betreffen können. Der Abnahmetest findet nach der Systeminstallation beim Auftraggeber in der künftigen Einsatzumgebung statt.

1.8.3 Qualität des Entwicklungsprozesses und Wiederverwendung

Qualitätskriterien für den Entwicklungsprozess

Auch für den Software-Entwicklungsprozess und die zugehörigen Instrumente werden Qualitätskriterien vorgeschlagen, die zur Beurteilung und Verbesserung der Projektabwicklung in softwareproduzierenden Unternehmen genutzt werden können. Genannt seien etwa die folgenden Kriterien:

- Der Entwicklungsprozess sollte planbar, transparent und kontrollierbar sein (BALZERT 1998).
- Die Wahl und Ausgestaltung des Prozessmodells sollte dem jeweiligen Projekt angemessen sein. So eignen sich wasserfallartige Modelle z.B. aus heutiger Sicht grundsätzlich nur, wenn die Gesamtheit der Anforderungen bereits bei Projektbeginn mehr oder minder feststeht.
- In den verschiedenen Projektphasen verwendete Entwicklungsmethoden und Werkzeuge sollten aufeinander abgestimmt sein. Ihre Anwendung sollte also nicht zu strukturellen Brüchen bei Phasenübergängen führen. Im folgenden Kapitel wird gezeigt, dass der strukturierte Ansatz der Softwareentwicklung in dieser Hinsicht erhebliche Defizite aufweist.
- Methoden sollten ferner verhältnismäßig leicht erlernbar sein. Hiervon hängt u.a. ab, ob ihre Einführung in einem Unternehmen oder Projekt kostengünstig gestaltet werden kann und ob das Management überhaupt dazu bereit ist. Eine unabdingbare Voraussetzung für den Einsatz einer Methode stellt ihre Werkzeugunterstützung dar (vgl. PAGEL und SIX 1994).

Qualitätsmodelle CMM und ISO 9000

Komplexe Modelle zur Bewertung und Verbesserung der Qualität des Entwicklungsprozesses stellen das amerikanische Capability Maturity Model (CMM) sowie der ISO-9000-Ansatz dar. Das CMM unterscheidet anhand technisch-organisatorischer Kriterien fünf Stufen der Prozessreife. Verläuft die Entwicklung auf der ersten Stufe völlig spontan und unkoordiniert, so werden auf der höchsten Stufe ständige Messungen prozessbezogener Qualitätsindikatoren vorgenommen und zu einer kontinuierlichen Prozessverbesserung genutzt (vgl. z. B. COAD und YOURDON 1994, BOOCH 1994). Das ISO-9000-Normenwerk, dessen Bestandteile auch als DIN-Normen übernommen wurden, enthält u.a. Richtlinien für die Entwicklung, Lieferung und Wartung von Software (vgl. ISO 9000-3, 1992). Es erlangt vor allem wegen der möglichen Zertifizierung von Softwareherstellern zunehmende Bedeutung.

Wiederverwendung

Eine der größten Herausforderungen an die Softwareentwicklung und ihr Management stellt heute die Wiederverwendung bereits vorhandener Software dar.

was?

Für eine Wiederverwendung kommen nicht nur implementierte Systemkomponenten in Betracht. Auch Ergebnisse von Analyse und Entwurf wie fachliche Modelle, Systemarchitekturen und Komponentenspezifikationen sowie Dokumentationen früher entwickelter Systeme eignen sich grundsätzlich zur Wiederverwendung in neuen Projekten. Für die Wiederverwendung kommt sowohl im eigenen Unternehmen entwickelte wie auch zugekaufte Software in Frage.

Vorteile

Die Wiederverwendung von Softwarekomponenten verspricht entscheidende Vorteile. Diese umfassen sowohl die Steigerung der Produktivität und die Senkung der Entwicklungskosten wie auch die Verbesserung der Produktqualität und eine höhere Termintreue. Qualitätsverbesserungen können vor allem deshalb erzielt werden, weil wiederverwendete Systemkomponenten bereits zuvor, z.B. als Bestandteil anderer Produkte, getestet wurden und sich im praktischen Einsatz bewährt haben. Die Folge von Qualitätsverbesserungen sind wiederum sinkende Wartungskosten.

aktuelle Situation

Trotz ihrer zweifellosen Vorzüge wird die Wiederverwendung in der heutigen Softwareproduktion allgemein in völlig ungenügendem Ausmaß praktiziert. Das „Rad“ wird immer noch einmal neu erfunden. Die Softwareproduktion ist derzeit noch durch eine große Fertigungstiefe charakterisiert, bei der auch „Komponenten von Komponenten“ meist selbst realisiert werden, anstatt auf vorhandene Software zurückzugreifen (vgl. WEBER 1992). Die Ursachen hierfür sind vielfältiger Natur (vgl. KAUBA 1997). Eine Rolle spielen u.a. fehlende strategische Entscheidungen, fehlende Anreize, die mangelnde Einbindung in den Entwicklungsprozess, fehlendes Know-how und die Abneigung der Entwickler gegen die Übernahme fremder Software, bekannt als Not-invented-here-Syndrom. Dass Entwickler auch um ihre Jobs fürchten, sei nur am Rande vermerkt.

Anforderungen

Die angeführten Faktoren lassen bereits erkennen, dass eine umfassend und erfolgreich betriebene Wiederverwendung an diverse Voraussetzungen gebunden ist und ein koordiniertes Vorgehen des Managements und der Entwickler softwareproduzierender Unternehmen erfordert. Abschließend sei daher auf einige grundlegende Anforderungen der umfassend praktizierten Wiederverwendung eingegangen:

- Die Wiederverwendung muss vom Management als eine strategische Aufgabe begriffen und in der Geschäftsstrategie, der langfristigen Planung der Softwareproduktion und projektübergreifenden Standards für die Softwareentwicklung verankert werden. Ein hoher Grad an Wiederverwendung lässt sich nur schrittweise und im Ergebnis langfristiger Anstrengungen erreichen.
- Das ökonomische Potential und die Vorteile der Wiederverwendung können nur dann voll ausgeschöpft werden, wenn die Softwareproduktion von vornherein auf die Erstellung wiederverwendbarer Komponenten orientiert wird. Dies zieht einmalige und projektbezogene zusätzliche investive Aufwendungen nach sich. Einerseits fallen Aufwendungen für die generelle Etablierung der Wiederverwendung, etwa für Schulungsmaßnahmen und Werkzeuge, an. Andererseits lassen sich wiederverwendbare Systemkomponenten innerhalb einzelner Projekte nur mit höherem Entwicklungsaufwand erstellen. Als Erfahrungswert gilt, dass eine Komponente zumindest dreimal wiederverwendet werden muss, bevor sich der erhöhte Entwicklungsaufwand rentiert.
- Die Wiederverwendung von Komponenten durch die Entwickler eines Projekts sollte belohnt werden, bedarf also entsprechender Anreize. Diese können mate-

rieller aber auch moralischer Art sein. Mentale Widerstände gegen die Wiederverwendung wie das Not-invented-here-Syndrom sind in gemeinsamen Diskussionen von Entwicklern und Managern zu überwinden.

- Die Wiederverwendung muss organisatorisch durch eine geeignete Infrastruktur abgestützt werden. Hierzu gehört vor allem ein Wiederverwendungsarchiv, das den raschen und werkzeuggestützten Zugriff auf potentiell wiederverwendbare Systemkomponenten und Entwicklungsergebnisse gestattet.
- Erstellung und Einsatz wiederverwendbarer Systemkomponenten verlangen auf Seiten der Entwickler die Ausbildung eines speziellen Know-how. In der Analyse geht es darum, mittels eines höheren Abstraktionsgrades zu allgemeiner anwendbaren, nicht projektgebundenen fachlichen Modellen zu gelangen. In Entwurf und Implementierung stehen die Entwickler vor der Aufgabe, für die Wiederverwendung geeignete Konzepte wie z.B. Entwurfsmuster, Klassenbibliotheken und Frameworks (vgl. Kap. 2) anzuwenden. Teils wird die Einrichtung projektunabhängiger Architekturgruppen aus erfahrenen und kreativen Mitarbeitern empfohlen, die sich speziell der Entwicklung und Anwendung wiederverwendbarer Architekturen widmen (vgl. OESTEREICH 1998).

Übungsaufgaben zu Kapitel 1.8

Übungsaufgabe 1.8.1

In der Programmiersprache C/C++ können zu einem in der Sprache vordefinierten Datentyp wie *float* für reelle Zahlen weitere Typnamen vereinbart werden, wie z.B.:

```
typedef float TEMPERATUR; // TEMPERATUR ist weiterer Name für float
typedef float LAENGE; // LAENGE ist weiterer Name für float
```

Dies kann zur Verständlichkeit von Programmcode beitragen, indem etwa Variablen unter Benutzung problembezogener Alias-Typnamen definiert werden, z.B.:

```
TEMPERATUR temperatur; // eine Variable (nur) für Temperatur-Werte
LAENGE laenge; // eine Variable (nur) für Längen-Werte
```

Jedoch wird die Zuweisung von Werten beider Variablen vom Compiler nicht beanstandet:

```
temperatur = laenge; // ok – zulässige Anweisung
```

Bewerten Sie dieses Sprachmerkmal unter dem Aspekt der konstruktiven Qualitätssicherung. Können Sie sich eine unter dem genannten Gesichtspunkt bessere Alternative vorstellen?

Übungsaufgabe 1.8.2

Unter welcher Bedingung würden Sie einen Test als erfolglos und wann als erfolgreich bezeichnen? Welcher Zusammenhang besteht zwischen dem Erfolgskriterium von Tests und dem Grundsatz der personellen Trennung von Entwicklung und Test?

Übungsaufgabe 1.8.3

In einem Programm tritt eine ganzzahlige Variable *familienstand* auf, die die gültigen Werte 0 (LEDIG), 1 (VERHEIRATET), 2 (GESCHIEDEN) und 3 (VERWITWET) annehmen kann. Jeder der Fälle wird im Programm unterschiedlich behandelt. Welche Äquivalenzklassen müssen bei einem funktionalen Test betrachtet werden und durch welche Werte können diese repräsentiert werden?

Übungsaufgabe 1.8.4

Ein Programm berechnet die Fläche eines gleichseitigen Dreiecks und liest hierzu einen einzigen reellen Wert von der Tastatur ein, der die Länge aller drei Seiten angibt. Welche Äquivalenzklassen müssten bei einem funktionalen Test betrachtet werden und durch welche Werte können diese repräsentiert werden? Treffen Sie zusätzliche Annahmen zu Sonderfällen.

Übungsaufgabe 1.8.5

In einem Programm zur Verkehrssimulation werden zwei Variablen *ampel1* und *ampel2* verwendet, deren Werte zu Programmbeginn eingelesen werden. Als gültige Werte kommen 0 (ROT), 1 (GELB) und 2 (GRUEN) in Betracht, deren Kombinationen jeweils unterschiedlich behandelt werden. Wie viele Testfälle müssen entsprechend der funktionalen Testmethode der Äquivalenzklassen berücksichtigt werden, wenn nur gültige Eingabewerte betrachtet werden.

Übungsaufgabe 1.8.6

Nennen Sie zwei wesentliche und zugleich eng zusammenhängende Vorteile der Wiederverwendung. Stellen Sie auch den Zusammenhang beider Vorteile kurz dar.

Übungsaufgabe 1.8.7

Wiederverwendbare Softwarekomponenten können sowohl implementierte Softwarebausteine wie auch (Teil-)Modelle der Analyse und des Entwurfs sein. Um zur Wiederverwendung gut geeignet zu sein, müssen die Softwarekomponenten allerdings in jedem Fall gewisse grundlegende Bedingungen erfüllen. Nennen Sie einige dieser Bedingungen.

Diese Seite bleibt aus technischen Gründen frei.

2 Objektorientierte Softwareentwicklung im Überblick

2.1 Grundlegende Konzepte der Objektorientierung

grundlegende objektorientierte Konzepte und ...

Bei einer objektorientierten Softwareentwicklung erfolgen Analyse und Entwurf auf der Basis einheitlicher Konzepte. Die Implementierung wird mittels objektorientierter Programmiersprachen durchgeführt, die wiederum die objektorientierten Konzepte weitgehend direkt unterstützen.

Der objektorientierte Entwicklungsansatz bietet eine reichhaltige Auswahl von Konzepten an. Ihrer detaillierten Einführung wird eine Betrachtung des objektorientierten Ansatzes aus der Vogelperspektive vorangestellt. Die in Abb. 2.1 gezeigten grundlegenden Konzepte und ihre Beziehungen werden anschließend schrittweise erläutert.

... ihre Beziehungen

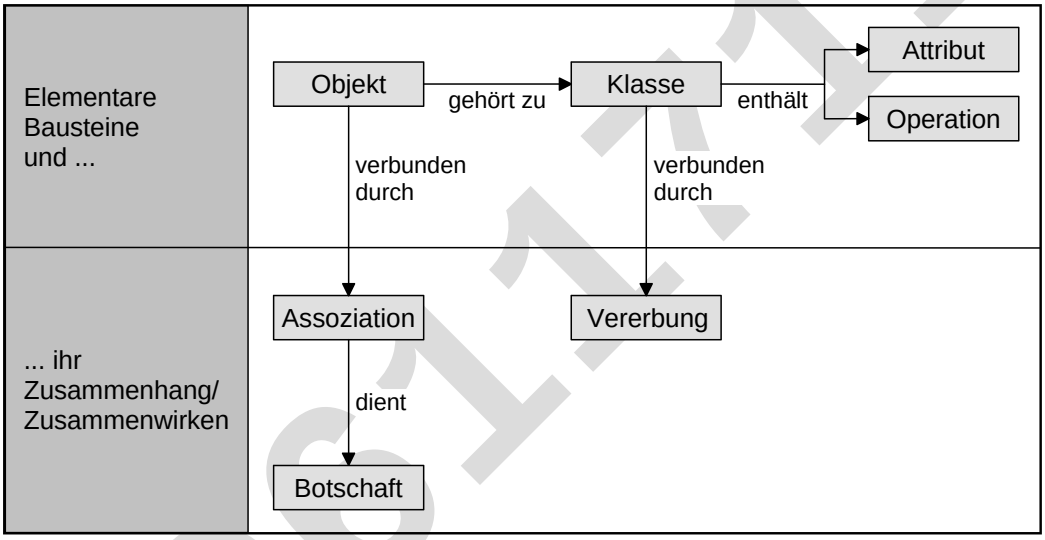


Abb. 2.1. Grundlegende objektorientierte Konzepte und ihre Beziehungen.

(1) Objekte und Klassen, Attribute und Operationen

Objekte, Klassen

Im Zentrum des objektorientierten Ansatzes stehen Objekte und ihre Klassen. Sie bilden die elementaren Bausteine objektorientierter Modelle und Programme und werden jeweils durch Abstraktion aus den realen Objekten des interessierenden Anwendungsbereiches gewonnen.

Attribute, Operationen

Ein Objekt besitzt einerseits gewisse Daten, andererseits kann es bestimmte Operationen ausführen. Jedes Objekt gehört zu einer Klasse. Eine Klasse legt einen bestimmten Typ von Objekten fest. Sie enthält Attribute, die Bedeutung und Wertebereich der Daten der zugehörigen Objekte angeben. In der Klasse werden ferner die von den Objekten ausführbaren Operationen als Algorithmen definiert.

Klasse und zugehörige Objekte

Zu einer Klasse können beliebig viele Objekte existieren. Alle Objekte einer Klasse stimmen in ihren Attributen und den ausführbaren Operationen überein. Sie unterscheiden sich jedoch im Allgemeinen hinsichtlich der Attributwerte. Dies wird in Abb. 2.2 verdeutlicht. Beide Objekte der Klasse Artikel besitzen die Attri-

bute Bezeichnung und Preis sowie einheitliche Operationen, während die Attributwerte differieren.

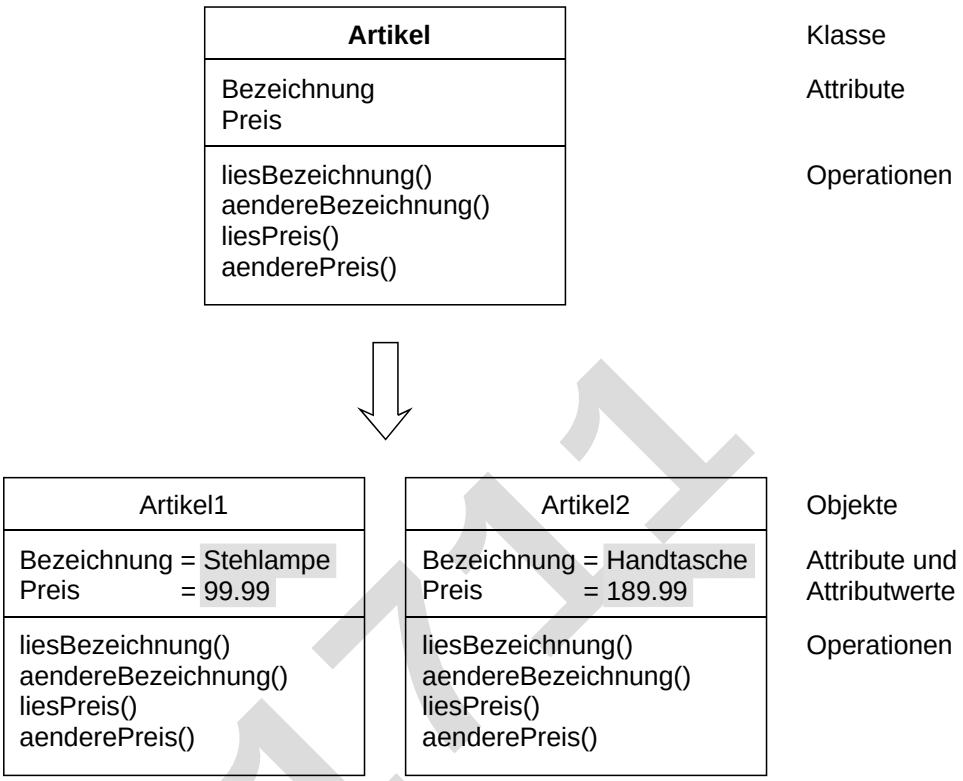


Abb. 2.2. Eine Klasse und zugehörige Objekte.

Kapselung

Charakteristisch für den objektorientierten Ansatz ist, dass Daten und Funktionen zu ihrer Verarbeitung nicht getrennt definiert, sondern vielmehr in Objekten und Klassen zu einer Einheit verschmolzen werden. Dies wird als Kapselung von Daten und Funktionen bezeichnet.

Infolge der Kapselung kann eine Funktion nur als Operation einer Klasse definiert und zugleich als Operation eines Objekts – also nicht unabhängig von einem Objekt – ausgeführt werden. Dementsprechend bezieht sich eine ausgeführte Operation auf ein bestimmtes Objekt. Der Aufruf einer Operation wird unter Angabe des Objekts notiert. So bewirkt etwa der Operationsaufruf

Artikel1.aenderePreis(49.99),

dass der Preis des Objekts Artikel1 auf 49.99 Euro geändert wird.

Geheimnisprinzip

Über die Kapselung von Daten und Funktionen hinausgehend realisieren die Konzepte Klasse und Objekt das Geheimnisprinzip (vgl. Kap. 2.4.1). So sind die in einer Klasse hinterlegten Algorithmen der Operationen für andere Klassen nicht sichtbar. Diese wissen z.B. nichts über eine evtl. Zerlegung der Algorithmen in Teilalgorithmen. Ferner – und darauf kommt es vor allem an – sind die Daten eines Objekts für andere Objekte nicht sichtbar. Der Zugriff auf die Daten eines Objekts kann daher nur über seine Operationen erfolgen. Anstelle von Kapselung und Geheimnisprinzip wird auch zusammenfassend von Datenkapselung gesprochen. Die Abb. 2.3 veranschaulicht die Datenkapselung.

Datenkapselung

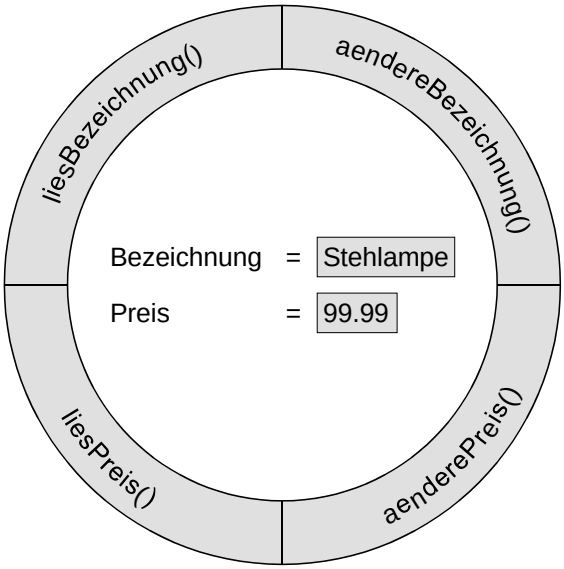


Abb. 2.3. Datenkapselung am Beispiel der Klasse Artikel.

Ergänzend sei bemerkt, dass in objektorientierten Programmiersprachen der äußere Zugriff auf Attributwerte von Objekten ggf. gesondert gesperrt werden muss. In der Systemanalyse wird jedoch üblicherweise – so auch hier – von der Einhaltung des Geheimnisprinzips bzw. der Datenkapselung ausgegangen.

Objektfabrik

Da zu einer Klasse beliebig viele Objekte angelegt werden können, werden Klassen auch als „Objektfabriken" charakterisiert. Jedes Objekt „kennt" seine Klasse und kann daher auf die in der Klasse definierten Operationen zugreifen.

(2) Botschaften und Assoziationen

In objektorientierten Modellen und Programmen wird die gesamte Funktionalität einer Anwendung einschließlich aller Anwendungsdaten auf verschiedene Klassen und ihre Objekte verteilt. Die Daten der Objekte sind gegen einen äußeren Zugriff geschützt. Die Objekte können zwar Operationen ausführen, tun dies aber nicht von selbst.

Botschaft

Die erforderliche Kooperation und Kommunikation von Objekten erfolgt über Botschaften. Mittels einer Botschaft übermittelt ein Sender-Objekt einem Empfänger-Objekt eine Aufforderung, eine bestimmte Operation auszuführen. Im Ergebnis werden Daten des Empfänger-Objekts modifiziert und/oder dem Sender-Objekt übergeben.

Beispiel zur Interaktion mittels Botschaften

Beispiel 2.1

Betrachtet sei neben der in Abb. 2.2 eingeführten Klasse Artikel eine Klasse Lieferant, die eine Operation druckeArtikelliste() besitzen soll. Die Operation gibt alle von einem Lieferanten gelieferten Artikel aus. Um die Daten der relevanten Artikel zu erhalten, werden an jedes korrespondierende Artikel-Objekt die Botschaften liesBezeichnung() und liesPreis() gesendet. Dies führt zum Aufruf der gleichnamigen Operationen. Diese senden die gewünschten Daten an das betreffenden Lieferanten-Objekt zurück. Dort können sie nun innerhalb der Operation druckeArtikelliste() gemeinsam ausgegeben werden.

Benutzung/ Ausführung einer Anwendung

Bei der hier vorwiegend interessierenden dialogorientierten Verarbeitung soll eine bestimmte Benutzung einer Anwendung zu einem vom Benutzer gewünschten, in sich abgeschlossenen Ergebnis – wie etwa einer Liste aller Artikel eines Lieferanten – führen. Wird die Anwendung objektorientiert erstellt, so stellt sich eine bestimmte Benutzung der Anwendung grob wie folgt dar:

- Der Benutzer initiiert über die Benutzungsoberfläche die gewünschte Verarbeitung, gibt benötigte Daten ein und stößt hierbei ein- oder mehrmals Operationen von Objekten an.
- Eine von der Benutzungsoberfläche angestoßene Operation führt im Allgemeinen zu einer Kette weiterer Operationen. Alle Operationen werden über Botschaften an gewisse Objekte angestoßen. Ein- und Ausgabedaten sowie Zwischenergebnisse sind grundsätzlich Attributwerte verschiedener Objekte.
- Schließlich werden die Ausgabedaten über die Benutzungsoberfläche verfügbar gemacht.
- Operationen umfassen auch die Erzeugung und Vernichtung von Objekten. Dies trifft insbesondere auf die Initiierung der Verarbeitung zu.

Objektverbindungen

Um einem Empfänger-Objekt eine Botschaft zu übermitteln, muss dieses dem Sender-Objekt bekannt sein. Es muss also eine Verbindung zwischen beiden Objekten hergestellt werden.

Assoziationen:

...bestehen zwischen Objekten

... sind dauerhafte Verbindungen

Verbindungen zwischen Objekten werden überwiegend als Assoziationen modelliert und realisiert. Assoziationen bilden nach einer treffenden Formulierung von RUMBAUGH et al. (1993) den „Klebstoff“ eines objektorientierten Modells. Objektverbindungen können temporär für eine einmalige Botschaftsübermittlung hergestellt und anschließend „vergessen“ werden. Assoziationen stellen dagegen dauerhafte Verbindungen zwischen Objekten dar, die einmal eingerichtet jederzeit wieder genutzt werden können. Die Objekte einer jeden Klasse differieren nur hinsichtlich ihrer Attributwerte. Folglich repräsentieren Objektverbindungen im Wesentlichen Beziehungen zwischen den Daten einer Anwendung.

In der Analyse werden Assoziationen oder auch temporäre Verbindungen als solche, d.h. außerhalb der Objekte modelliert. In Entwurf und Implementierung werden Objektverbindungen z.B. durch Zeiger innerhalb der Objekte realisiert. Das folgende Beispiel verdeutlicht das Konzept der Assoziation.

Beispiel zur Anwendung von Assoziationen

Beispiel 2.2

In dem zuvor betrachteten Beispiel 2.1 wird man zwischen den Objekten der Klassen Lieferant und Artikel eine Assoziation vorsehen. Dies bedeutet später für die ausführbare Anwendung, dass eine Verbindung von einem Lieferanten-Objekt zu einem Artikel-Objekt eingerichtet wird, sobald feststeht, dass der betreffende Artikel von dem entsprechenden Lieferanten geliefert wird. Dies kann etwa nach Eingaben an der Benutzungsoberfläche der Fall sein. Die ständigen Verbindungen können dann u.a. zu der im Beispiel 2.1 dargestellten Ausgabe der Artikelliste genutzt werden.

Betrachtet werden hier nur einseitige Verbindungen, wonach nur Lieferanten-Objekte ihre Artikel-Objekte kennen. Möglich sind auch der umgekehrte Fall sowie eine Kombination beider Varianten, d.h. bidirektionale Verbindungen.

Betont sei nochmals, dass Assoziationen als Objektverbindungen unmittelbar zwischen Objekten, nicht zwischen Klassen bestehen. Ebenso werden Botschaften im Allgemeinen zwischen Objekten versandt.

(3) Vererbung

Vererbung besteht zwischen Klassen und ...

Wie Botschaften und Assoziationen leisten Vererbungsbeziehungen einen Beitrag zum Zusammenhalt und Zusammenwirken der elementaren Bausteine eines objektorientierten Modells oder Programms (vgl. Abb. 2.1). Doch bestehen Vererbungsbeziehungen direkt zwischen Klassen.

Eine Vererbungsbeziehung liegt oft zwischen einer übergeordneten und mehreren untergeordneten Klassen vor. Die übergeordnete Klasse fasst gemeinsame Komponenten (Attribute, Operationen) der untergeordneten Klassen zusammen. Jede untergeordnete Klasse erbt diese gemeinsamen Komponenten und ergänzt sie durch weitere Attribute und Operationen. Die gemeinsamen Komponenten müssen nur einmal, nämlich in der übergeordneten Klasse, definiert werden.

... unterstützt Generalisierung/ Spezialisierung

Das Vererbungskonzept unterstützt die Generalisierung (Verallgemeinerung) und Spezialisierung von Klassen. Es trägt wesentlich zu übersichtlichen und redundanzarmen Modellen und Programmen bei. Die Realisierung einer Anwendung wird insbesondere durch die Wiederverwendung bereits implementierter Klassen aus Klassenbibliotheken erleichtert, die dann durch benötigte untergeordnete Klassen ergänzt werden.

Für das objektorientierte Paradigma ist das Konzept der Vererbung konstitutiv. Methoden der Analyse oder des Entwurfs wie auch Programmiersprachen, die das Konzept der Vererbung nicht unterstützen, gelten nicht als objektorientiert.

Die Abb. 2.4 zeigt ein Beispiel einer Vererbungsbeziehung.

Beispiel einer Vererbung

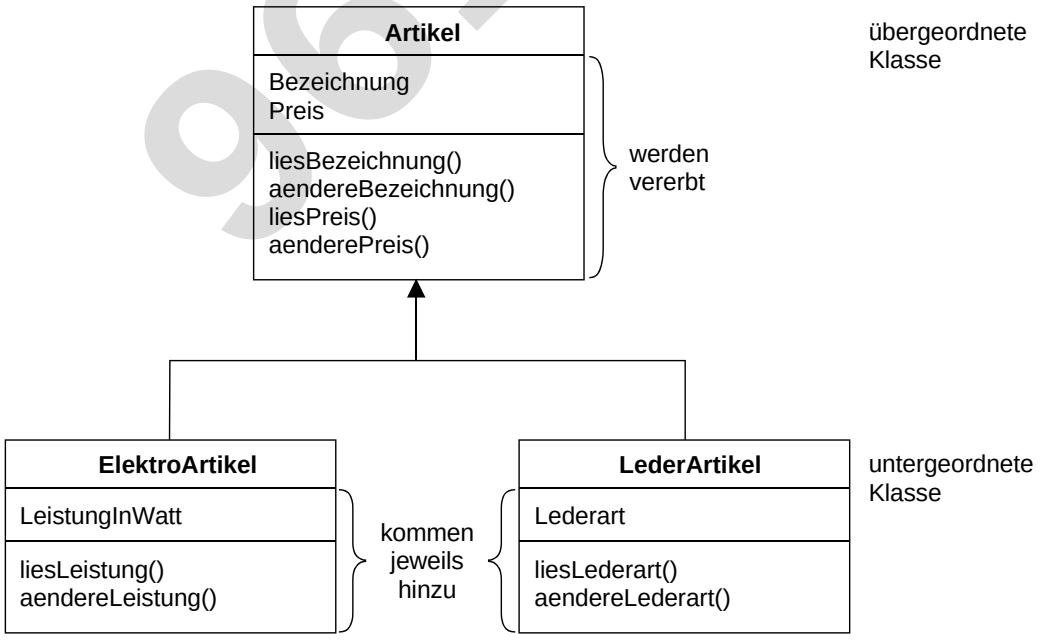


Abb. 2.4. Beispiel einer Vererbungsbeziehung.

Kern der Objektorientierung

Unter Beachtung der bisherigen Darlegung kann schließlich der Kern der Objektorientierung in folgender „Gleichung“ von COAD und YOURDON (1994) zusammengefasst werden:

„Objektorientierte Methode	=	Klassen und Objekte + Vererbung + Kommunikation mit Nachrichten.“

Anstelle des Begriffs Nachricht wird hier ausschließlich das Synonym Botschaft benutzt.

Übungsaufgaben zu Kapitel 2.1

Übungsaufgabe 2.1.1

Betrachtet sei ein Handelsunternehmen, das mit zahlreichen Lieferanten kooperiert und von diesen über Lieferantenaufträge Artikel bezieht, die wiederum auf der Grundlage von Kundenaufträgen an verschiedene Kunden geliefert werden. Jeder Lieferantenauftrag und jeder Kundenauftrag besteht aus meist mehreren Auftragspositionen. Jede Auftragsposition umfasst eine gewisse Menge eines Artikels. Das Handelsunternehmen benötigt ein Anwendungssystem zur integrierten Abwicklung seiner Geschäftsbeziehungen mit Lieferanten und Kunden. Schlagen Sie – ganz intuitiv – für ein objektorientiertes Anwendungssystem ca. sechs Klassen vor, die sich an den realen Objekten des Anwendungsbereiches orientieren. Die Klassen sollen nur benannt werden, während Attribute und Operationen der Klassen nicht spezifiziert werden müssen.

Übungsaufgabe 2.1.2

Betrachtet sei wiederum das in Aufgabe 2.1.1 beschriebene Handelsunternehmen. Kundenaufträge und Lieferantenaufträge werden jeweils zunächst erfasst, später evtl. noch modifiziert, einzeln oder als Listen ausgegeben sowie abgerechnet. Hieraus ergibt sich das Erfordernis bestimmter Assoziationen für die Versendung von Botschaften. Nennen Sie zwei Assoziationen, die bei der Bearbeitung von Kunden- und Lieferantenaufträgen genutzt werden können und begründen Sie kurz Ihre Wahl.

Übungsaufgabe 2.1.3

Betrachtet sei nochmals das in Aufgabe 2.1.1 beschriebene Handelsunternehmen. Nennen Sie zwei sinnvolle Vererbungsbeziehungen, die sich bei der objektorientierten Modellierung des Anwendungsbereiches anbieten. Führen Sie hierzu eine Generalisierung der in der Aufgabe 2.1.1 ermittelten Klassen durch und schlagen Sie geeignete übergeordnete Klassen vor. Veranschaulichen Sie die Vererbungsbeziehungen, indem Sie jeweils ein oder zwei Attribute der übergeordneten Klasse benennen, die an die untergeordneten Klassen vererbt werden.

Übungsaufgabe 2.1.4

Grenzen Sie die Begriffe der Kapselung und der Datenkapselung bezogen auf das objektorientierte Klassenkonzept voneinander ab. Betrachten Sie die Abb. 2.3, die die Datenkapselung veranschaulicht. Finden Sie eine simple grafische Darstellung, die nur die einfache Kapselung der Bestandteile der Klasse Artikel verbildlicht. Beziehen Sie auch das Record-Konzept prozeduraler Sprachen (vgl. Kap.

1.5) in Ihre Überlegungen ein: Bilden Records ein Beispiel für die Kapselung oder für die Datenkapselung?

2.2 Die Entwicklung der Objektorientierung und die UML

Meilensteine der Objektorientierung

Die Geschichte des objektorientierten Ansatzes reicht bis in die 1960er Jahre zurück. In Tab. 2.1 sind einige Meilensteine der Objektorientierung zusammengestellt.

Entwicklung der Objektorientierung

Die Entwicklung der Objektorientierung sei wie folgt kommentiert:

- Die Tab. 2.1 lässt mehrere Entwicklungsstränge der Objektorientierung erkennen. Unterscheiden lassen sich die Entwicklung von Programmiersprachen, von Methoden für Analyse und Entwurf, ferner die Entwicklung von Systemsoftware und Werkzeugen wie z.B. objektorientierten Datenbanksystemen sowie die Entwicklung objektorientierter Standards. Natürlich bestehen gewisse Überschneidungen und enge Zusammenhänge zwischen den Strängen.
- Analog zur Entwicklung des älteren strukturierten Ansatzes wurden zunächst objektorientierte Programmiersprachen erfunden und bereits praktisch eingesetzt, bevor geeignete Methoden zur Unterstützung von Analyse und Entwurf im objektorientierten Paradigma vorgeschlagen wurden.
- Während die objektorientierten Programmiersprachen einerseits einen Bedarf nach entsprechenden Analyse- und Entwurfs-Methoden hervorriefen, wirkten sie zugleich als „Konzept-Spender“. Doch speiste sich die Entwicklung auch aus anderen Quellen. Hingewiesen sei hier nur auf die Entity-Relationship-Modellierung und ihre in der semantischen Datenmodellierung vorgenommene Erweiterung (vgl. COAD und YOURDON 1994). Zusammen boten sie bereits Konzepte wie Assoziation oder Attributvererbung an – und zwar einschließlich einer geeigneten grafischen Notation. Die Übernahme dieser und anderer Konzepte und die Bereitstellung einer grafischen Notation für die objektorientierten Konzepte erwies sich als ein Schlüsselfaktor für den Durchbruch bei den objektorientierten Entwicklungsmethoden zu Beginn der 1990er Jahre.
- Insgesamt wurden in den ersten 1990er Jahren ca. 50 objektorientierte Methoden vorgeschlagen (vgl. STEIN 1994). Die in Tab. 2.1 genannten Werke stellen also nur eine kleine Auswahl von Methoden vor. Es sind allerdings gerade die erfolgreichsten und populärsten Methoden, die in der Folgezeit in breitem Umfang eingesetzt wurden und werden. Die Methoden von COAD und YOURDON bzw. von BOOCH werden nach ihren Begründern benannt. Die von RUMBAUGH et al. begründete Methode trägt den Namen *Object Modeling Technique* (OMT), die von JACOBSON et al. kreierte Methode wird als *Object Oriented Software Engineering* (OOSE) bezeichnet.

UML

- Ein besonders wichtiges Ereignis bildete in den letzten Jahren die Entwicklung der Unified Modeling Language (UML) als einheitlicher Modellierungssprache für die Objektorientierung. Den Anstoß gab die Erkenntnis, dass vor allem die erfolgreichsten Methoden wesentliche Gemeinsamkeiten aufweisen und weitgehend auf den gleichen grundlegenden Konzepten basieren. Die überflüssige Vielfalt der Notation der gleichen objektorientierten Konzepte in verschiedenen Methoden wird mit der UML überwunden (vgl. Kap. 1.5, Beispiel 1.8).

Zeit	Ereignis / Erläuterung
1967	Die erste objektorientierte Programmiersprache (OOPL) Simula-67 wird vorgestellt. Simula-67 kennt bereits Klassen und abgeleitete Klassen (Vererbung).
1970er Jahre	Die OOPL Smalltalk wird bei der Firma XEROX entwickelt. Besondere Verbreitung erfährt die 1980 vorgestellte Version Smalltalk-80. Zusammen mit der Sprache werden fundamentale Konzepte und Mechanismen grafischer Benutzungsoberflächen entdeckt und erstmals umgesetzt.
1980	Die früheste Version der Sprache C++ erscheint unter dem Namen „C with classes“. Der Name „C++“ wird erst 1983 geprägt. C++ wird später mehrfach um zusätzliche Konzepte erweitert und ist ab Anfang der 1990er Jahre die dominierende OOPL für industrielle Anwendungen.
1988	MEYER (1988) erscheint als eines der ersten Bücher zum objektorientierten Entwurf. B. MEYER ist zugleich Erfinder der OOPL Eiffel.
1989	Die Object Management Group (OMG) wird gegründet. Sie dient der Entwicklung internationaler Standards für die Objektorientierung. Ein besonders wichtiger OMG-Standard ist der sog. CORBA-Standard zur Spezifikation von Schnittstellen verteilter objektorientierter Anwendungen. 1999 umfasst die OMG mehr als 800 Mitglieder (u.a. Hard- und Softwarehersteller).
erste 1990er Jahre	Die Bücher BOOCH (1991), COAD und YOURDON (1991a), COAD und YOURDON (1991b), RUMBAUGH et al. (1991) und JACOBSON et al. (1992) erscheinen und tragen wesentlich zur Etablierung der objektorientierten Analyse und des objektorientierten Entwurfs bei.
1991	Die Object Database Management Group (ODMG) wird von Herstellern und Nutzern objektorientierter Datenbanken zur Entwicklung entsprechender Standards gegründet. 1997 wird der Standard ODMG 2.0 verabschiedet, der Schnittstellen für Smalltalk, C++ und Java enthält.
1995	Das grundlegende Buch GAMMA et al. (1995) über Entwurfsmuster erscheint.
1996	Die Sprache Java wird vorgestellt. Sie erlangt rasch wachsende Bedeutung u.a. als Programmiersprache für Internet-basierte Anwendungen.
1997	Die OMG standardisiert die Modellierungssprache Unified Modeling Language (UML) in der Version UML 1.1. Die UML wurde ab 1994 von G. BOOCH, J. RUMBAUGH und I. JACOBSON entwickelt. Nach 1997 erscheinen weitere UML-Versionen; vgl. etwa UML (1999).
1998	Der ANSI-C++-Standard wird als internationaler (ISO-)Standard verabschiedet.

Tab. 2.1. Einige historische Meilensteine der Objektorientierung.

Die UML unterstützt die Entwicklung eines breiten Spektrums von Anwendungen, darunter interaktive betriebliche Anwendungen, verteilte und Echtzeitanwendungen. Darüber hinaus eignet sich die UML z.B. auch für die Unternehmensmodellierung (vgl. BALZERT 1998). Ab der Version 1.3 kann die UML als Industriestandard für die objektorientierte Modellierung betrachtet werden (vgl. OESTEREICH 1998). Alle bedeutenden Werkzeughersteller bieten UML-Unterstützung.

Die UML ist eine Sprache und keine Entwicklungsmethode (vgl. Kap. 1.5). Sie stellt objektorientierte Konzepte und ihre grafische Notation bereit, die in verschiedenen Modellen bzw. Diagrammen kombiniert verwendet werden können. Die UML enthält jedoch keine methodische Vorgehensweise. Diese muss also bei der Benutzung der UML geeignet ergänzt werden.

UML-Diagramme

Die angebotenen Konzepte lassen sich um verschiedene Diagrammtypen herum gruppieren, wobei jedem Diagrammtyp ein bestimmter Satz von Konzepten entspricht. Ein Überblick der Diagramme stellt daher zugleich einen Überblick der UML her. Die Diagramme können wiederum einerseits den grundlegenden Entwicklungsaktivitäten Analyse, Entwurf und Implementierung und andererseits den verschiedenen, bereits eingeführten Sichten auf ein Softwaresystem zugeordnet werden (vgl. Kap. 1.3). Dies geschieht in Tab. 2.2.

Sicht / Aktivität	statische Sicht		dynamische Sicht	
	Funktionssicht	**Datensicht**	**Ablaufsicht**	**Zustandssicht**
Analyse	Anwendungsfall-diagramm, Klassen-diagramm	Klassen-diagramm, Objekt-diagramm	Sequenz-diagramm, Kollaborations-diagramm	Zustands-diagramm, Aktivitäts-diagramm
Entwurf	Klassen-diagramm	Klassen-diagramm Objekt-diagramm	Sequenz-diagramm, Kollaborations-diagramm	Zustands-diagramm, Aktivitäts-diagramm
Implementierung	Komponentendiagramm Verteilungsdiagramm			

Tab. 2.2. UML-Diagrammtypen gruppiert nach grundlegenden Entwicklungsaktivitäten und Sichten.

Sequenz- und Kollaborationsdiagramme werden gemeinsam als Interaktionsdiagramme bezeichnet. Die Diagramme und Konzepte der Analyse und des Entwurfs werden später im benötigten Umfang und entsprechend der UML-Version 1.3 eingeführt. Die Implementierungsdiagramme werden nicht weiter betrachtet. Komponentendiagramme stellen die Komponenten eines ausführbaren Programms (u.a. Binärcode-Module, DLLs) und ihre gegenseitigen Beziehungen, darunter Abhängigkeitsbeziehungen, dar. Verteilungsdiagramme visualisieren u.a., welche Komponenten auf welchen Rechnern eines Rechnernetzes ablaufen sowie die Kommunikationsbeziehungen zwischen Komponenten und Rechnern. Eine kompakte Beschreibung der UML (Version 1.3) findet sich in OESTEREICH (1998).

Übungsaufgaben zu Kapitel 2.2

Übungsaufgabe 2.2.1

Was bedeutet es konkret, dass die UML keine Methode, sondern nur eine Modellierungssprache ist? Stützen Sie sich bei Ihrer Antwort auf den im Kap. 1.5 eingeführten Methodenbegriff.

Übungsaufgabe 2.2.2

Erläutern Sie, welche Vorteile der einheitliche Einsatz der UML im Unterschied zur gleichzeitigen Verwendung verschiedener grafischer Notationen der Vorgängermethoden (von BOOCH, RUMBAUGH und JACOBSON) in der objektorientierten Softwareentwicklung bietet.

2.3 Objektorientierung und grundlegende Entwicklungsaktivitäten

OO Analyse, OO Entwurf, OO Implementierung

Wurden in Kap. 1.3 die grundlegenden Aktivitäten einer Softwareentwicklung unabhängig vom Entwicklungsparadigma beschrieben, so soll nun knapp umrissen werden, wie sich Analyse, Entwurf und Implementierung im objektorientierten Ansatz gestalten. Wie im Vorwort erwähnt, orientiert sich die Darstellung wesentlich an der von BALZERT (1999) vorgeschlagenen Vorgehensweise.

(1) Analyse

OOA

Analysemethoden, die auf objektorientierten Konzepten basieren, werden unter dem Oberbegriff objektorientierte Analyse (kurz OOA) zusammengefasst.

Pflichtenheft

Der erste Analyseschritt ist die Erstellung eines Pflichtenheftes, wobei häufig noch eine fachliche Vorstudie als erste Pflichtenheftversion angefertigt wird (vgl. Kap. 1.7.1). Auch im objektorientierten Ansatz ist das Pflichtenheft ein überwiegend textuelles Dokument. Bereits in die Erarbeitung des Pflichtenheftes integriert wird hier die stets am Anfang der Analyse stehende Untersuchung der Anwendungsfälle, d.h. der wesentlichen Varianten der Systembenutzung. Die Gesamtheit der Anwendungsfälle wird u.a. durch ein Anwendungsfalldiagramm visualisiert. Bemerkt sei, dass Anwendungsfälle kein eigentlich objektorientiertes Konzept darstellen und ebenso gut im strukturierten Entwicklungsansatz genutzt werden können.

OOA-Modell, Fachkonzept

Im zweiten Schritt der Analyse, der ihren Kern ausmacht, wird ein fachliches Modell erstellt, das ab jetzt hauptsächlich als Fachkonzept oder OOA-Modell bezeichnet sei. Die Modellierung erfolgt aus statischer und dynamischer Sicht. Das Fachkonzept gliedert sich daher in ein statisches und ein dynamisches Modell.

statisches Modell

Das statische Modell wird als Klassendiagramm erstellt und umfasst die aus fachlicher Perspektive erforderlichen Klassen und ihre strukturellen Beziehungen (Assoziationen, Vererbungen). Zusätzlich dient das Konzept der Pakete der übersichtlichen Gliederung größerer Modelle.

Der Kapselung als einem Grundzug der Objektorientierung entspricht es, dass das Klassendiagramm sowohl die Funktionssicht wie auch die Datensicht der Anwendung umfasst. Zum statischen Modell gehört hier auch ein Klassenlexikon, in dem z.B. die einzelnen Attribute der Klassen genauer beschrieben werden.

dynamisches Modell

Das dynamische Modell beschreibt einerseits mit Sequenz- und Kollaborationsdiagrammen die zeitlichen Abläufe bei der Ausführung von Anwendungsfällen, die noch in Szenarien differenziert werden (Ablaufsicht). Mittels Zustandsdiagrammen wird berücksichtigt, dass Ausführbarkeit und Ergebnis von Operationen von Objektzuständen, vor allem von Attributwerten der Objekte, abhängen können (Zustandssicht).

Prototyp der Benutzungsoberfläche

Im dritten Schritt der Analysephase wird bereits ein Prototyp der Benutzungsoberfläche entwickelt, der allerdings nur die Anwendungsdaten zeigt und noch keine Funktionalität bereitstellt. Hierfür sprechen mehrere Gründe:

- Das Fachkonzept berücksichtigt nur einige Aspekte der Benutzungsoberfläche, etwa welche Daten zwischen ihr und der Fachkomponente der Anwendung ausgetauscht werden. Nicht festgelegt wird, wie die Daten an der Oberfläche dargestellt werden und wie der Benutzer mit dem System interagiert.

- Ein ablauffähiger Prototyp kann unter Verwendung geeigneter Werkzeuge komfortabel und rasch erstellt und nach einer Weiterentwicklung im Entwurf als endgültige Oberflächen-Komponente in die Anwendung integriert werden.
- Die Gestaltung der Benutzungsoberfläche ist heute von besonderer Bedeutung für die Akzeptanz der Anwendung. Ein früh erstellter Prototyp, mit dem experimentiert werden kann, erleichtert dem Auftraggeber und späteren Systembenutzern, Einfluss auf eine bedarfsgerechte Benutzungsoberfläche zu nehmen.
- Ein ablauffähiger, „anfassbarer" Prototyp der Benutzungsoberfläche ist geeignet, die gesamte Anforderungsdefinition unter Einbeziehung des Auftraggebers auf den „Prüfstand" zu stellen. Dies gilt insbesondere für die Merkmale Intentionstreue und Vollständigkeit der Anforderungsdefinition. Das Pflichtenheft und das Fachmodell sind wegen der abstrakten und teils formalen Darstellung zur Überprüfung der Anforderungsdefinition weniger geeignet.

Reihenfolge der OOA-Schritte

Die drei Schritte der Analyse sollten in der angegebenen Reihenfolge durchlaufen werden. Insbesondere sollte der Prototyp aus dem Fachkonzept abgeleitet werden; das gegenteilige Vorgehen kann zu einem unübersichtlichen fachlichen Modell führen, das sich auch negativ auf den Entwurf auswirkt.

Meist müssen allerdings die genannten Schritte mehrfach durchlaufen werden. Die Erstellung der Anforderungsdefinition ist also ein iterativer Prozess, der erst abbricht, wenn die Anforderungsdefinition als hinreichend konsolidiert gilt.

(2) Entwurf und Implementierung

OOD, OOI

Auf objektorientierten Konzepten fußende Entwurfsmethoden werden unter dem Oberbegriff objektorientierter Entwurf (kurz OOD) subsumiert. Bei Verwendung einer objektorientierten Programmiersprache wird von objektorientierter Implementierung oder Programmierung (kurz OOI bzw. OOP) gesprochen.

Schichtenarchitektur

Der Entwurf spezifiziert die Anwendungsarchitektur, die auch die Benutzungsoberfläche, die Datenhaltung sowie weitere software-technische Aspekte berücksichtigt. Im objektorientierten Ansatz werden vorzugsweise Schichtenarchitekturen eingesetzt. Drei-Schichten-Architekturen bestehen aus der problemorientierten Fachkonzeptschicht, der Schicht der Benutzungsoberfläche (kurz GUI-Schicht) sowie der Datenhaltungsschicht für eine permanente Datenspeicherung und gelten als Stand der Technik. Vorrangiges Anliegen einer Drei-Schichten-Architektur ist eine von der GUI-Schicht unabhängige Realisierung der Fachkonzept- und der Datenhaltungsschicht, die also keine Voraussetzungen über die GUI-Schicht treffen und nicht auf ihre Klassen zugreifen sollen. Dies ermöglicht eine einfache Modifikation der Benutzungsoberfläche oder gar ihren Austausch ohne Eingriffe in die anderen Schichten.

OOD-Modell

Die Einheitlichkeit der Konzepte in Analyse und Entwurf bildet auch die Grundlage einer weitgehend analogen Modellstruktur. Wie das Fachkonzept der Analyse gliedert sich auch die Softwarespezifikation, die nun auch als Entwurfs-Modell oder OOD-Modell bezeichnet wird, in ein statisches und ein dynamisches Modell. Kern des statischen Modells ist erneut das (OOD-)Klassendiagramm der Anwendung, während das dynamische Modell vor allem Sequenzdiagramme umfasst. Klassendiagramm, Klassenlexikon und Sequenzdiagramme berücksichtigen nun alle Schichten der Anwendung und ihr Zusammenwirken. Die GUI-Schicht wird ausgehend von dem in der Analyse erstellten Prototypen weiterentwickelt.

Weiterentwicklung des OOA-Modells

Besonders bedeutsam ist, dass das Fachkonzept der Analyse bereits als erste Version des OOD-Modells der Fachkonzeptschicht gelten kann. Die Klassen des fachlichen Modells und ihre Beziehungen können also grundsätzlich in das Entwurfs-Modell übernommen werden. Allerdings sind wesentliche Modifikationen und Erweiterungen des Fachkonzepts erforderlich, von denen einige beispielhaft genannt seien:

- Es werden zusätzliche Klassen, nämlich sogenannte Container-Klassen zur Verwaltung der Objekte der Klassen des Fachkonzepts eingeführt.
- Es werden Vorkehrungen zur Realisierung der Assoziationen des Fachkonzepts getroffen. Dies betrifft etwa die Einführung geeigneter Attribute (vgl. Kap. 2.1).
- Im Zusammenhang mit der Vererbung kommt ggf. auch der Polymorphismus als weiteres wesentliches objektorientiertes Konzept zur Anwendung. Von Polymorphismus (Vielgestaltigkeit) wird gesprochen, wenn gleichlautende Botschaften, die an Objekte verschiedener Klassen versendet werden, zur Ausführung gleichnamiger, aber in ihrer Wirkung unterschiedlicher Operationen führen. Mit Hilfe des Polymorphismus können vor allem Fallunterscheidungen effizient und erweiterungsfreundlich umgesetzt werden.

Die Erweiterungen des Fachkonzepts sowie die Einführung zusätzlicher Schichten bedingen, dass das Entwurfs-Modell meist wesentlich umfangreicher ist und weitaus mehr Klassen umfasst als das Fachkonzept der Analyse.

Entwurfsmuster

Weitere in Entwurf und Implementierung zu beachtende Aspekte betreffen die Verwendung von Entwurfsmustern, Klassenbibliotheken und Frameworks. Entwurfsmuster stellen bewährte schablonenartige Lösungen von immer wieder vorkommenden Entwurfsproblemen dar, die u.a. als Klassendiagramme notiert werden. So gibt es beispielsweise ein Zustandsmuster, das bei einer zustandsabhängigen Ausführung von Operationen anwendbar ist und im Entwurf die Zustandsdiagramme der Analyse im Sinne einer direkt implementierbaren Lösung ersetzen kann.

Klassenbibliotheken

Klassenbibliotheken enthalten Sammlungen von Klassen für verschiedene, mehr oder minder spezialisierte Anwendungsgebiete wie z.B. Benutzungsoberflächen. Die Klassen sind oft in Vererbungshierarchien angeordnet. Bei Bedarf werden Objekte relevanter Bibliotheksklassen angelegt und ihre Operationen angewendet oder eigene untergeordnete Klassen durch Vererbung gewonnen, die dann unmittelbar genutzt werden.

Frameworks

Frameworks stellen spezielle Klassenbibliotheken dar, die bereits eine bestimmte Anwendungsarchitektur festlegen. Ein bekanntes Beispiel für eine mächtige Klassenbibliothek, die zugleich ein Framework ist, stellen die *Microsoft Foundation Classes* dar.

Entwurf und Implementierung

Entwurf und Implementierung werden nicht in getrennten Phasen, sondern eng verzahnt durchgeführt. Attribute und Deklarationen (Funktionsköpfe) der Operationen (Funktionen) der Klassen werden bereits im Entwurf festgelegt. Wesentlicher Gegenstand der Implementierung mittels einer objektorientierten Sprache ist das Programmieren der Operationen aller Klassen der Anwendung.

Die Abb. 2.5 fasst abschließend die Ergebnisse von Analyse, Entwurf und Implementierung bei einer objektorientierten Entwicklung zusammen. Bemerkt sei, dass hier von der Entwicklung von Anwendungen mit grafischer Benutzungsoberfläche ausgegangen wird. Weitere Erläuterungen zum GUI-Modell finden sich in Kap. 5.

Ferner werden Implementierung und Test hier unter Implementierung zusammengefasst, an deren Ende daher auch die Dokumentation der Anwendung komplett vorliegen muss. Die Systemdokumentation umfasst neben dem Quellcode der Anwendung alle Analyse- und Entwurfsdokumente.

Ergebnisse von OOA, OOD, OOI

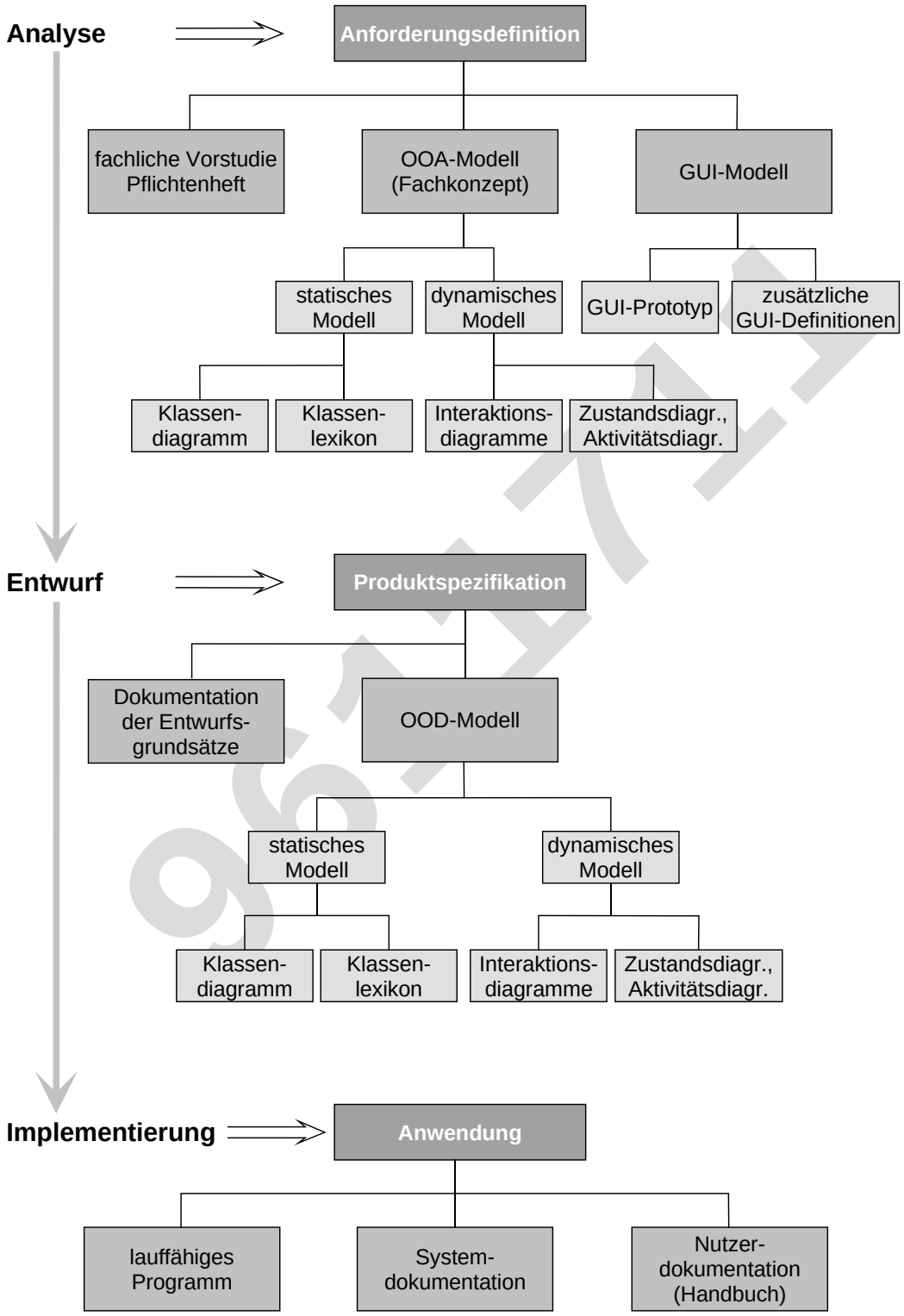


Abb. 2.5. Die Ergebnisse von Analyse, Entwurf und Implementierung bei objektorientierter Entwicklung.

Die in Abb. 2.5 präsentierte Übersicht der Ergebnisdokumente, aus denen sich die Teilaktivitäten der objektorientierten Entwicklung ergeben, wird in den weiteren

Kurskapiteln wie auch im anschließenden Kurs „Objektorientierter Systementwurf“ als Entwicklungsschema zugrunde gelegt. Analyse und Entwurf werden aus didaktischen Gründen separat, also gemäß einem Wasserfallmodell dargestellt. Wie bereits erwähnt, kommt jedoch bei der Benutzungsoberfläche eine evolutionäre Prototypentwicklung zum Zuge; ferner werden Entwurf und Implementierung gemeinsam behandelt.

Übungsaufgaben zu Kapitel 2.3

Übungsaufgabe 2.3.1

Begründen Sie, warum im Klassendiagramm die Datensicht und die Funktionssicht auf eine Anwendung kombiniert dargestellt werden. Benutzen Sie für Ihre Antwort den in Kap. 2.3 grob umrissenen Inhalt des Klassendiagramms ebenso wie die Erläuterung der grundlegenden objektorientierten Konzepte in Kap. 2.1.

Übungsaufgabe 2.3.2

In dem Klassenlexikon des Fachkonzepts einer Anwendung werden die Attribute aller Klassen detailliert beschrieben. Hierzu gehören z.B. Angaben zur fachlichen Bedeutung, zum Wertebereich und einzuhaltenden Restriktionen. Welche Informationen sollten ebenfalls in ein Klassenlexikon aufgenommen werden, um eine möglichst vollständige fachliche Beschreibung der Klassen zu erhalten?

Übungsaufgabe 2.3.3

Warum sollte zunächst ein fachliches Modell der Anwendung und erst danach ein Prototyp der Benutzungsoberfläche erstellt werden? Gehen Sie davon aus, dass der Prototyp später evolutionär zu einer vollständigen Benutzungsoberfläche weiterentwickelt wird. Berücksichtigen Sie ferner, dass die gesamte Anwendung, die u.a. eine fachliche Komponente (Fachkonzeptschicht) und die Benutzungsoberfläche (GUI-Schicht) umfasst, eine möglichst klare und einfache Struktur besitzen sollte, die sich an den grundlegenden realen Objekten des Anwendungsbereiches orientiert.

2.4 Objektorientierte und strukturierte Softwareentwicklung

Um die beiden Hauptansätze der Softwareentwicklung vergleichen zu können, wird zunächst auch das strukturierte Paradigma grob skizziert. Die Charakterisierung des strukturierten Ansatzes und der folgende Vergleich beider Ansätze konzentrieren sich auf Analyse und Entwurf.

2.4.1 Charakterisierung der strukturierten Softwareentwicklung

strukturierte Softwareentwicklung,

Strukturierte Methoden der Softwareentwicklung zur Unterstützung von Analyse und Entwurf wurden etwa im Zeitraum von 1970 bis 1990 in zahlreichen Varianten vorgeschlagen. Die folgende Charakterisierung des strukturierten Ansatzes

orientiert sich an besonders leistungsfähigen strukturierten Methoden: der von YOURDON (1989) eingeführten Modernen Strukturierten Analyse (kurz MSA) sowie dem modularen Entwurf, einer Entwurfsmethode, die in verschiedenen Varianten existiert. Für umfassende Darstellungen strukturierter Analyse- und Entwurfsmethoden sei auf BALZERT (1996), GEHRING (1993a) sowie PAGEL und SIX (1994) verwiesen.

MSA, modularer Entwurf

(1) Analyse

Ein zu erstellendes Analysemodell umfasst meist mehrere Teilmodelle, die verschiedene Sichten auf eine Anwendung repräsentieren. Hierzu gehören ein Datenflussmodell, ein Datenmodell sowie eventuell ein Ereignismodell.

strukturierte Analyse, Teilmodelle

Das Datenflussmodell beschreibt, welche Informationen bzw. Daten von wo nach wo durch die Anwendung fließen, die man sich als bereits vorhanden vorstellt. Es besteht in der Regel aus mehreren Datenflussdiagrammen (DFD). Die Abb. 2.6 zeigt ein einfaches Datenflussdiagramm zur Verwaltung von Kundenaufträgen und zugehörigen Stammdaten, das mittels der Notation von GANE und SARSON (1979) erstellt und aus GEHRING (1993a) entnommen wurde.

Datenflussmodell, DFD

DFD-Beispiel

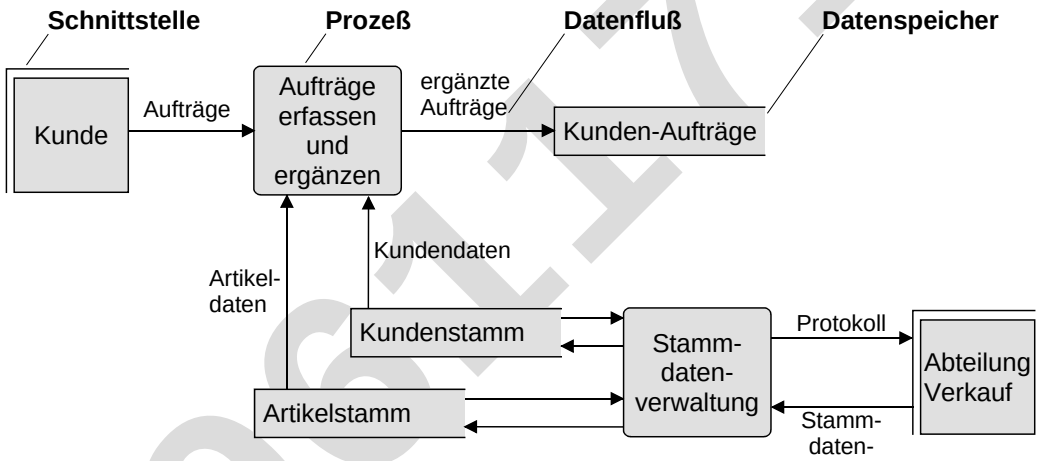


Abb. 2.6. DFD einer Auftragsverwaltung in der Notation von GANE und SARSON (1979).

In einem DFD kommen stets die in Tab. 2.3 erläuterten vier Bausteintypen bzw. Konzepte zur Anwendung.

DFD-Konzepte

DFDs werden – dem Anliegen der Analyse gemäß – ausschließlich zur Modellierung auf der logischen Ebene benutzt, während von physischen Aspekten wie etwa dem Speichermedium von Datenspeichern abgesehen wird.

Das Datenflussmodell umfasst in der Regel mehrere, hierarchisch angeordnete DFDs. Die Spitze der Hierarchie bildet ein grobes DFD, welches das gesamte System abbildet. Durch eine schrittweise Verfeinerung, die jeweils an den Prozessen eines DFD ansetzt, werden weitere, nachgeordnete DFDs erstellt. Hierbei können auch zusätzliche Datenspeicher eingeführt werden. Im obigen Beispiel wird man etwa den Prozess der Stammdatenverwaltung weiter verfeinern.

DFD-Hierarchie

Das Datenflussmodell wird durch ein Data Dictionary ergänzt. In diesem werden die meist strukturierten Daten der Datenflüsse und Datenspeicher unter Verwendung einer formalen Notation schrittweise in ihre atomaren Bestandteile zerlegt.

Data Dictionary

So kann etwa eine Kundenadresse in die Bestandteile Name, Vorname, Straße, Postleitzahl und Ort gegliedert werden.

Baustein	Erläuterung
Schnittstelle	Schnittstellen bilden die Systemumwelt ab und sind typischerweise Klassen von Personen oder Organisationseinheiten. Eine Schnittstelle stellt eine Quelle dar, aus der Daten in das System gelangen oder/und eine Senke, in der Daten des Systems verschwinden.
Datenfluss	Datenflüsse repräsentieren Daten, die zwischen den übrigen Bausteinen ausgetauscht werden. Ein Datenfluss definiert einen Typ von Datensätzen (Records). Alle nacheinander fließenden Datensätze eines Datenflusses besitzen den gleichen Aufbau.
Prozess	Prozesse transformieren einlaufende in auslaufende Datenflüsse und bilden die Verarbeitung der Anwendungsdaten, d.h. die verschiedenen Funktionen des Systems ab.
Datenspeicher	Datenspeicher dienen der Ablage von Daten. Schreibende Zugriffe auf Datenspeicher werden durch einlaufende Datenflüsse, lesende Zugriffe durch auslaufende Datenflüsse visualisiert. Auch ein Datenspeicher definiert einen Datensatztyp (Record-Typ), dem alle in einem Datenspeicher abgelegten Datensätze entsprechen.

Tab. 2.3. Bausteine eines Datenflussdiagramms.

Prozessspezifikation

Komplettiert wird das Datenflussmodell schließlich durch eine mehr oder minder formalisierte logische Spezifikation aller Prozesse der DFDs der untersten Hierarchieebene, welche die jeweils auszuführenden Datentransformationen beschreibt.

Datenflussmodell und Funktionssicht

Obwohl das Datenflussmodell die Anwendungsdaten berücksichtigt, liegt der Schwerpunkt auf der Transformation der Anwendungsdaten durch die Prozesse bzw. Funktionen der Anwendung. Daher wird das Datenflussmodell der Funktionssicht zugeordnet.

Datenmodell, ERM und Datensicht

Das Datenmodell betrachtet dagegen eine Anwendung aus der Datensicht. Es wird gewöhnlich als Entity-Relationship-Modell (ERM) erstellt. ERM werden bei der Modellierung relationaler Datenbanken verwendet und hier als bekannt vorausgesetzt (vgl. GEHRING 1993b). Die Abb. 2.7 zeigt ein einfaches ERM für die zuvor als DFD modellierte Auftragsverwaltung.

ERM-Beispiel

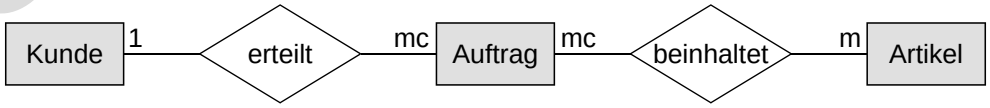


Abb. 2.7. ERM einer Auftragsverwaltung.

Bedeutung des Datenmodells

Erst das Datenmodell widerspiegelt die Anwendungsdaten in vollem Umfang und erschließt insbesondere die gegenseitigen Beziehungen zwischen den Anwendungsdaten. So wird z.B. in dem betrachteten (simplen) Beispiel erst durch das ERM abgebildet, dass jeder Auftrag von genau einem Kunden erteilt wird und einen oder mehrere Artikel umfasst.

Ereignismodell und dynamische Sicht

Während das Datenflussmodell sowie das Datenmodell die statische Sicht auf die Anwendung repräsentieren, ist das Ereignismodell als drittes Teilmodell der Analyse der dynamischen Sicht zuzuordnen. Eine Ereignismodellierung wird durchgeführt, wenn die Ausführung von Funktionen der Anwendung abhängig von Systemzuständen bzw. den aktuellen Werten von Anwendungsdaten ist. Beispiels-

weise können stornierte Aufträge eines Kunden nicht mehr abgerechnet werden. Ereignismodelle werden in Form von Zustandsübergangsdiagrammen dargestellt, die etwa den Zustandsdiagrammen des objektorientierten Ansatzes entsprechen und daher hier nicht näher betrachtet seien.

(2) Entwurf

strukturierter Entwurf, modularer Ansatz, Modul

Strukturierte Entwurfsmethoden spezifizieren eine Anwendung als ein System von Modulen, die untereinander Leistungen austauschen. In der folgenden Tab. 2.4 werden Module als elementare Bausteine des strukturierten Entwurfs aus verschiedenen Blickwinkeln charakterisiert.

Aspekte von Modulen

Modulaspekt	Erläuterung
Modulinhalt	Ein Modul enthält im Allgemeinen Funktionen, (modulweit sichtbare) Variablen, Konstanten sowie benutzerdefinierte Datentypen.
Modulaufbau	Ein Modul gliedert sich in einen Modulrumpf und eine Modulschnittstelle. Der Rumpf beinhaltet Definitionen der vorher genannten Modulelemente. In der Schnittstelle werden Modulelemente deklariert, die für andere Module bereitgestellt werden (Exportschnittstelle) sowie Elemente anderer Module angegeben, die im Modul benutzt werden (Importschnittstelle).
Technische Anforderungen an Module	Ein implementierter Modul sollte nur wenige hundert Zeilen Quellcode umfassen und von einer Person in kurzer Zeit (ca. 1 Woche) realisiert werden können.
Moduldarstellung	Schnittstelle und Rumpf einzelner Module werden im Entwurf meist semiformal, etwa mittels eines Pseudocodes dargestellt.

Tab. 2.4. Einige Aspekte von Modulen.

Darüber hinaus sollten ein Modul und eine modulare Zerlegung gewisse qualitative Anforderungen erfüllen:

qualitative Anforderungen an Module

- Die für andere Module bereitgestellten Elemente eines Moduls sollten ohne Kenntnis ihrer modulinternen Realisierung anwendbar sein. Für die Benutzung einer Funktion sollte z.B. die Kenntnis des in der Exportschnittstelle hinterlegten Funktionskopfes und der Semantik der Funktion hinreichend sein.
- Ein Modul sollte eine einzige, klar abgegrenzte Teilaufgabe übernehmen, zu der alle Modulelemente beitragen. Anders gesagt: die Elemente eines Moduls sollten einen möglichst starken inneren Zusammenhang aufweisen. Das Ausmaß, in dem alle Teile eines Moduls logisch zusammengehören, wird als Kohäsion bezeichnet. Ein Modul, der Aufträge verwaltet und zugleich eine Auftragsstatistik erstellt, besitzt sicherlich keine ausreichende Kohäsion und sollte zerlegt werden.
- Verschiedene Module sollten nur eine geringe Kopplung, d.h. eine möglichst geringe gegenseitige Abhängigkeit aufweisen. Die Kopplung von Modulen ist umso geringer, je schmaler die Schnittstellen der Module ausfallen. So sollten etwa Funktionen nur wenige und einfache Parameter besitzen.

Beispiel für eine Metrik

Auf Metriken als messbare Indikatoren für die Erfüllung von Qualitätsanforderungen wurde bereits hingewiesen (vgl. Kap. 1.8.1). Eine geringe Modulkopplung kann etwa durch eine Metrik nachgewiesen werden, die die durchschnittliche Parameteranzahl der Funktionen aller Module bestimmt.

modularer Entwurf und Geheimnisprinzip

Der modulare Entwurf als fortgeschrittene Entwurfsmethode zeichnet sich durch eine konsequente Umsetzung des bereits in Kap. 2.1 erwähnten Geheimnisprinzips aus. Das Geheimnisprinzip geht auf PARNAS (1972) zurück und stellt ein fundamentales Prinzip des Software Engineering mit vielfältigen Anwendungsmöglichkeiten dar. In allgemeiner Fassung besagt es etwa, dass eine Softwarekomponente stets so spezifiziert und implementiert werden sollte, dass der Umwelt nur Informationen zur Verfügung gestellt werden, die für die Benutzung der Komponente zwingend erforderlich sind. Dagegen sollen alle weiteren Realisierungsaspekte einer Komponente vor der Umwelt verborgen werden. Diese Vorgehensweise wird im Englischen als Information Hiding bezeichnet.

Abstraktion im Entwurf

Im modularen Entwurf kommen zwei Arten von Abstraktionen zur Anwendung, die jeweils das Geheimnisprinzip auf spezifische Weise umsetzen und denen verschiedene Modultypen entsprechen. Aus der Erläuterung des Geheimnisprinzips ergibt sich, dass es bei der Abstraktion im Entwurf anders als in der Analyse nicht um die Vernachlässigung unwesentlicher Merkmale des Anwendungsbereiches, sondern direkt um die Gestaltung von Softwarekomponenten geht. Diese sollen im Sinne der Komplexitätsbeherrschung ohne Kenntnis ihrer Details angewendet werden können.

funktionale Abstraktion

Eine funktionale Abstraktion stellt eine Verarbeitung von Daten zur Verfügung, während der Algorithmus der Verarbeitung verborgen bleibt. Funktionale Abstraktionen werden mittels Funktionen realisiert. Bei ihrer Benutzung werden lediglich Eingabeparameter übergeben und Ausgabeparameter bzw. Rückgabewerte übernommen. Ein Zugriff auf den zugrunde liegenden Algorithmus ist weder möglich noch aus Anwendersicht von Interesse. Bei einer funktionalen Abstraktion wird also von einem Algorithmus abstrahiert.

funktionale Module

Ein funktionaler Modul exportiert ausschließlich Funktionen und realisiert daher eine oder mehrere funktionale Abstraktionen. Die Ergebnisse eines Funktionsaufrufs hängen nur von den übergebenen Eingabeparametern, nicht von den Werten modulinterner Daten zum Zeitpunkt des Funktionsaufrufs ab. Gleiche Eingabewerte führen also immer zu identischen Ausgabewerten. Ein funktionaler Modul besitzt daher kein „Gedächtnis“. Funktionale Module bündeln häufig logisch verwandte Operationen, wie das folgende Beispiel zeigt.

Beispiel eines funktionalen Moduls

Beispiel 2.3

In einem Modul werden elementare mathematische Funktionen wie sin(x), cos(x), exp(x) usw. implementiert. Die (Export-)Schnittstelle beinhaltet dementsprechend folgende Funktionsköpfe, die hier in C notiert werden:

```
double sin(double x);
double cos(double x);
double exp(double x);
usw.
```

Ein übergebener Parameterwert (*x*) vom Standarddatentyp double bestimmt jeweils eindeutig den zurückgegebenen Ergebniswert desselben Typs. Es liegt also ein funktionaler Modul vor. Ein entsprechender Modul befindet sich in der ANSI-C-Standardbibliothek, seine Schnittstelle wird in der Datei *math.h* bereitgestellt.

Datenabstraktion, Definition und ...

Die Datenabstraktion ist auf die Manipulation komplexer Datenobjekte mit zahlreichen Komponenten zugeschnitten. Bei einer Datenabstraktion können Datenobjekte ausschließlich über Zugriffsfunktionen gelesen oder modifiziert werden, während ein direkter Zugriff etwa auf einzelne Komponenten nicht möglich ist. Eine Datenabstraktion beschreibt daher Datenobjekte durch die Definition der auf ihnen ausführbaren Operationen, während die Struktur des Datenobjekts bei seiner Benutzung verborgen bleibt. Bei einer Datenabstraktion wird also von einer Datenstruktur abstrahiert. Da auch die Algorithmen der Zugriffsfunktionen verborgen werden, umfasst eine Datenabstraktion zugleich funktionale Abstraktionen.

... Stufen

Es lassen sich zwei Stufen der Datenabstraktion, nämlich abstrakte Datenobjekte (ADO) und abstrakte Datentypen (ADT), unterscheiden.

ADO und Datenobjekt-Module

Ein ADO repräsentiert ein einzelnes Datenobjekt und wird durch einen Datenobjekt-Modul realisiert. Im Rumpf des Datenobjekt-Moduls werden eine Datenstruktur – d.h. das eigentliche Datenobjekt – sowie zugehörige Zugriffsfunktionen definiert. Die (Export-)Schnittstelle stellt nur die Zugriffsfunktionen bereit. Die modulinterne Datenstruktur „überlebt" die einzelnen Ausführungen von Zugriffsfunktionen und stellt das „Gedächtnis" des Datenobjekt-Moduls dar. Das Ergebnis einer Funktionsausführung hängt im Allgemeinen sowohl von den Eingabeparametern wie auch von den aktuell in der internen Datenstruktur aufbewahrten Werten ab. Mit der Implementierung eines Datenobjekt-Moduls ist das zugehörige ADO vorhanden; es muss also nicht mehr gesondert deklariert werden. Das folgende Beispiel skizziert einen Datenobjekt-Modul.

Beispiel für ein Datenobjekt-Modul

Beispiel 2.4

Ein Stapel bzw. Stack dient der Aufbewahrung von Daten bzw. einzelner Objekte gleichen Typs, z.B. ganzer Zahlen. Er stellt eine lineare Datenstruktur dar, in der die einzelnen Einträge nach dem Prinzip „last in, first out" verwaltet werden (vgl. GEHRING 1992). Das zuletzt eingefügte Objekt wird stets als erstes wieder entnommen. Auf einem Stack sind standardmäßig folgende Operationen ausführbar:

- *push*, d.h. Einfügen eines Objekts,
- *pop*, d.h. Entnehmen eines Objekts,
- *top*, d.h. Lesen eines Objekts, das jedoch nicht entnommen wird.

Ein Stapel-ADO kann durch einen Datenobjekt-Modul realisiert werden, in dessen Exportschnittstelle folgende Zugriffsfunktionen bereitgestellt werden:

```
push(in object : TObject, out full : Bool);
pop(out object : TObject, out empty : Bool);
top(out object : TObject, out empty : Bool);
```

Eingabeparameter sind durch *in*, Ausgabeparameter durch *out* gekennzeichnet. Der Typbezeichner *TObject* gibt den (importierten) Datentyp der Stack-Objekte an. Pro Zugriffsfunktion wird entweder ein Objekt übergeben (*push*) oder übernommen (*pop*, *top*). Außerdem wird jeweils ein Boolscher Statuswert zurückgegeben, wobei der Wert *false* den Erfolg, der Wert *true* den Misserfolg einer Funktionsausführung signalisiert. Eine *push*-Ausführung schlägt fehl, wenn der Stapel voll, d.h. nicht mehr erweiterbar ist. Eine *pop*- bzw. *top*-Ausführung schlägt fehl, wenn der Stapel leer ist.

Im Rumpf des Moduls sind einerseits die Zugriffsfunktionen, andererseits der Stapel selbst zu definieren. Dieser kann z.B. durch einen Vektor fester Länge mit dem Basistyp *TObject* einschließlich eines Zählers für die aktuelle Anzahl der Stapeleinträge implementiert werden. Da die interne Stapelstruktur offenbar als Modulgedächtnis fungiert und genau ein abstraktes Stapelobjekt realisiert wird, liegt ein Datenobjekt-Modul vor. Sein Aufbau wird in Abb. 2.8 veranschaulicht.

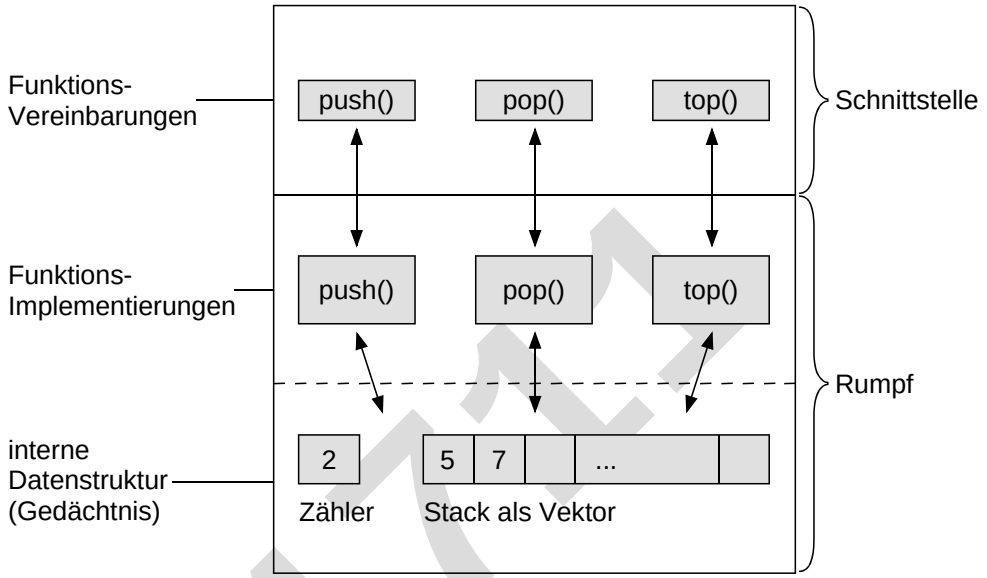


Abb. 2.8. Aufbau eines Datenobjekt-Moduls für eine Stackverwaltung.

ADT und Datentyp-Module

Abstrakte Datentypen stellen eine Verallgemeinerung abstrakter Datenobjekte dar und werden durch Datentyp-Module realisiert. Ein ADT bzw. der zugehörige Datentyp-Modul definiert lediglich einen Typ von abstrakten Datenobjekten, während zugehörige ADO zunächst noch nicht vorhanden sind. Diese können dann – analog zu einem gewöhnlichen Datentyp – durch separate Vereinbarungen als Variablen des ADT angelegt werden. Jedes aus einem ADT erzeugte abstrakte Datenobjekt besitzt mit seiner internen Datenstruktur ein eigenes Gedächtnis.

Vorteile der Datenabstraktion

Der Einsatz der Datenabstraktion ist charakteristisch für den modularen Entwurf und besitzt eminente Vorteile:

- Der Anwender einer Datenabstraktion muss die Details der zugehörigen internen Datenstruktur nicht kennen, die zuweilen hochkomplex ist.
- Unkontrollierte äußere Zugriffe auf die Datenstruktur einer Datenabstraktion, die diese in einen inkonsistenten Zustand versetzen können, werden ausgeschlossen.
- Häufig ergibt sich das Erfordernis, die interne Datenstruktur einer Datenabstraktion etwa aus Effizienzgründen zu modifizieren oder komplett zu ersetzen. So könnte im letzten Beispiel 2.4 der Stapel intern durch eine lineare („verzeigerte“) Liste anstelle eines Feldes realisiert werden. Dies kann ohne jegliche Modifikation benutzender Module geschehen, wenn nur die Vereinbarungen (Funktionsköpfe) der Zugriffsfunktionen in der Exportschnittstelle selbst unverändert bleiben.

Zusammengefasst bildet die Datenabstraktion ein mächtiges Werkzeug zur Beherrschung von Komplexität und zum Entwurf wartungsfreundlicher Anwendungen. Vorläufermethoden des modularen Entwurfs waren rein funktionsorientiert,

d.h. sie nutzten nur das Mittel der funktionalen Abstraktion bzw. funktionale Module. Dies führte zu Architekturen, bei denen die Veränderung von Datenstrukturen gravierende Modifikationen der Anwendung nach sich zog.

Interaktion von Modulen und Architektur

Standen bisher die Module als elementare Einheiten des strukturierten Entwurfs im Vordergrund, so sei nun kurz auf deren Zusammenwirken und ihre Anordnung in der Architektur einer Anwendung eingegangen.

Benutzbarkeitsbeziehungen

Der Austausch von Leistungen zwischen Modulen wird im Entwurf durch sogenannte Benutzbarkeitsbeziehungen abgebildet. Stellt ein Modul A ein Modulelement in seiner Exportschnittstelle bereit und nimmt ein Modul B dieses Modulelement in seine Importschnittstelle auf, so besteht zwischen beiden Modulen eine Benutzbarkeitsbeziehung. Exportiert und importiert werden vorrangig Funktionen und Datentypen, insbesondere ADT. Die eigentliche Benutzung importierter Modulelemente findet vor allem in den Funktionsdefinitionen der Modulrümpfe statt, wo z.B. importierte Funktionen aufgerufen werden. Importierte Datentypen (Typnamen) werden auch in den Schnittstellen der importierenden Module benutzt.

hierarchische Architektur

Benutzbarkeitsbeziehungen sind asymmetrisch zu gestalten. Importiert Modul B Elemente von Modul A, so sollte nicht zugleich Modul A Elemente von Modul B importieren. Aus der Gesamtheit der Benutzbarkeitsbeziehungen resultiert in diesem Fall eine hierarchische Anwendungsarchitektur. Die Spitze der Hierarchie wird durch ein Modul gebildet, das die gesamte Anwendung repräsentiert und eine Funktion enthält, die bei der Ausführung zuerst aufgerufen wird. Ein exportierender Modul steht in der Hierarchie unterhalb eines zugehörigen importierenden Moduls. Neben der hierarchischen Struktur besitzt eine Anwendungsarchitektur häufig folgende Merkmale:

weitere Architekturmerkmale

- Module mit gleichartiger Aufgabenstellung und ähnlichem Abstraktionsniveau werden im Entwurf zu (horizontalen) Schichten zusammengefasst. Unterschieden werden z.B. eine Steuer-Schicht für die Ablauf- und Dialogsteuerung als oberste Schicht, eine problemorientierte Schicht zur Umsetzung problemorientierter Operationen, eine Datenverwaltungs-Schicht zur Manipulation mehrerer Datenobjekte eines Typs und eine Datenzugriffs-Schicht für den Zugriff auf einzelne Datenobjekte eines Typs als unterste Schicht (vgl. GEHRING 1993a sowie GEHRING und RÖSCHER 1989).
- Funktionale Module werden vorrangig zur Ablauf- und Dialogsteuerung benutzt und finden sich daher vor allem in den oberen Schichten einer Architektur. Datenobjekt- und Datentyp-Module sind dagegen vorzugsweise in den unteren Schichten anzutreffen, in denen Aufgaben der Datenverwaltung zu lösen sind.

Moduldiagramm

Anwendungsarchitekturen werden in Moduldiagrammen visualisiert. Die Abb. 2.9 stellt ein Moduldiagramm mit vier Schichten schematisch dar.

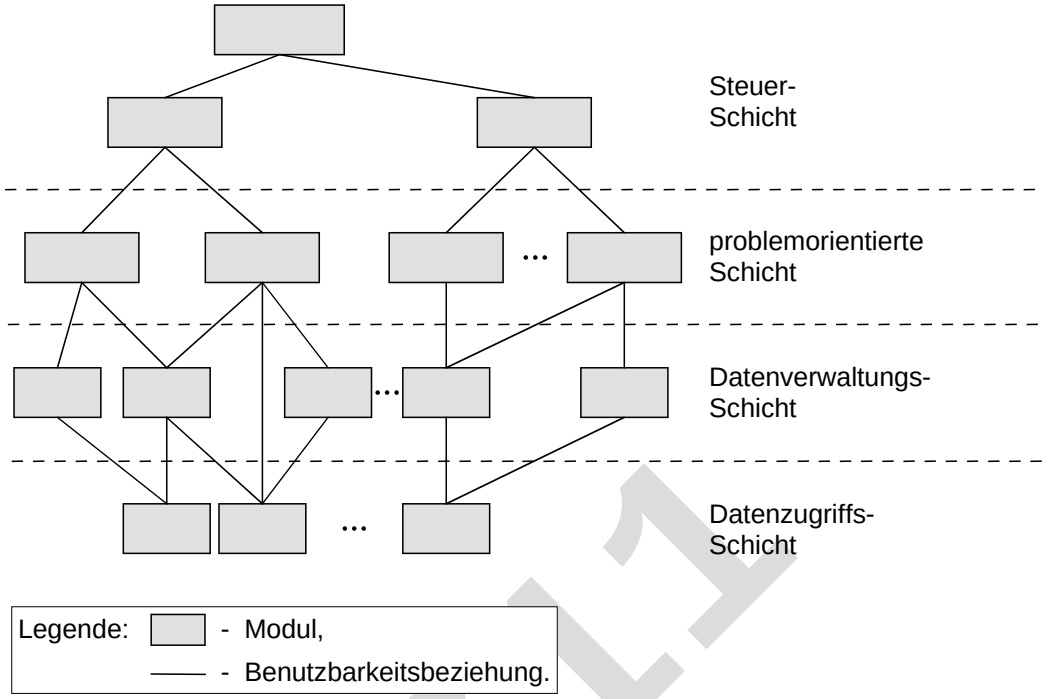


Abb. 2.9. Moduldiagramm mit vier Schichten.

2.4.2 Objektorientierter und strukturierter Ansatz im Vergleich

Die Merkmale und Vorzüge des objektorientierten Ansatzes der Softwareentwicklung treten deutlicher hervor, wenn man ihn einem Vergleich mit dem älteren strukturierten Ansatz unterzieht. Der folgende Vergleich zielt allerdings darauf ab, neben wesentlichen Differenzen auch Gemeinsamkeiten und Zusammenhänge beider Ansätze festzustellen. Dem Vergleich zugrunde liegende Details der vorherigen Kapitelabschnitte werden meist nicht mehr erwähnt.

(1) Einheitlichkeit der Modellierung in der Analyse

Analyse

Die strukturierte Analyse erstellt für die beiden wichtigsten Sichten auf eine Anwendung, nämlich die Funktions- und die Datensicht, separate Teilmodelle. Bei größeren Projekten erweist es sich als ausgesprochen schwierig, die Konsistenz des Datenflussmodells und des Datenmodells zu gewährleisten. Dies liegt nicht nur an der unterschiedlichen Modellierungsperspektive selbst. Hinzu kommt, dass für beide Teilmodelle sehr verschiedene Konzepte und Notationen verwendet werden. Ferner werden in der Praxis oft unterschiedliche Teams mit der Erstellung beider Teilmodelle beauftragt (vgl. COAD und YOURDON 1994).

Bei der objektorientierten Analyse wird für die Funktions- und die Datensicht nur ein Modell, nämlich das Klassendiagramm erstellt. Allerdings muss auch hier die Konsistenz zum dynamischen Modell (Sequenzdiagramme usw.) wie übrigens auch zum Klassenlexikon gewahrt werden. Dies fällt jedoch wesentlich leichter, da jeweils die gleichen Konzepte genutzt werden.

(2) Übergang von der Analyse zum Entwurf

Analyse und Entwurf

Beim Übergang von der Analyse zum Entwurf tritt im strukturierten Ansatz ein tiefgreifender Strukturbruch auf. Im Entwurf werden grundsätzlich andere Konzepte und Notationen verwendet als in der Analyse. Die Folge ist, dass die in der Analyse gewonnenen Informationen innerhalb des Entwurfs weitgehend neu strukturiert werden müssen.

Die Umsetzung der Analyseergebnisse in eine Anwendungsspezifikation kann bei funktionsorientierten Entwurfsmethoden auf der Grundlage von mehr oder minder detaillierten Regeln erfolgen. Doch ist dies gerade bei dem modularen Entwurf als der fortgeschrittensten strukturierten Entwurfsmethode nicht der Fall. Hier werden die Analyseergebnisse oft nur als eine „informelle Vorlage“ (BALZERT 1996, vgl. auch COAD und YOURDON 1994) verwendet. Die Architektur der Anwendung ist von Grund auf neu zu entwerfen.

Der objektorientierte Ansatz vermeidet einen Strukturbruch beim Übergang von der Analyse zum Entwurf. Beide Aktivitäten verwenden die gleichen grundlegenden Konzepte, die gleiche Notation und analoge Modelle. Doch die Homogenität und Passfähigkeit von Analyse und Entwurf im objektorientierten Ansatz ist nicht nur formaler Natur. Das in der Analyse erarbeitete fachliche Modell ist selbst bereits die erste Version der Entwurfsspezifikation der Fachkonzeptschicht, wenn ein übliches Schichtenmodell der Anwendung unterstellt wird. Dies besagt, dass die Strukturierung der problemorientierten Komponente einer Anwendung bereits weitgehend während der Analyse erfolgt. Um es einmal überpointiert zu sagen: die Hälfte des Entwurfs wird während der Analyse erledigt.

(3) Allgemeine Durchgängigkeit der Entwicklung

allgemeine Durchgängigkeit

Die Durchgängigkeit der objektorientierten Entwicklung erstreckt sich von der Analyse über den Entwurf bis hin zur Implementierung. Alle drei Aktivitäten basieren auf einheitlichen Konzepten. Betont sei nochmals, dass die gängigen objektorientierten Sprachen (C++, Java und Smalltalk) die Umsetzung der objektorientierten Konzepte in der Regel direkt unterstützen, d.h. über korrespondierende Programmierkonzepte verfügen.

Im strukturierten Ansatz bereitet auch der Übergang vom Entwurf zur Implementierung eher Probleme. Während beispielsweise die (ursprünglich prozedurale) Sprache Ada abstrakte Datentypen unterstützt, kann dieses fundamentale Entwurfskonzept in den am meisten verbreiteten prozeduralen Programmiersprachen wie z.B. C nur nachgebildet werden.

(4) Kommunikationsfreundlichkeit

Kommunikation mit der Auftraggeberseite

Oftmals wird dem objektorientierten Ansatz attestiert, dass er wegen der unmittelbaren Orientierung an den Objekten des Anwendungsbereichs zu Modellen führt, die besonders verständlich für Auftraggeber, fachliche Experten und spätere Benutzer sind. Die höhere Kommunikationsfreundlichkeit des objektorientierten gegenüber dem strukturierten Ansatz kann jedoch bezweifelt werden.

Bereits die Datenflussmodellierung und die Entity-Relationship-Modellierung mit ihrem jeweils sparsamen Konzept-Arsenal stellen relativ kommunikationsfreund-

liche Ansätze dar. Die Komplexität größerer Projekte setzt auch bei objektorientierter Modellierung dem Verständnis und Nachvollzug von Analysemodellen durch den genannten Personenkreis Grenzen. Von größerer Bedeutung für eine funktionierende Kommunikation zwischen Entwicklern einerseits und Auftraggeber, Benutzern und fachlichen Experten andererseits erscheinen der Einsatz von Prototypen und die Möglichkeit einer inkrementellen Entwicklung.

(5) Kompatibilität mit dem evolutionären Prozessmodell

Tauglichkeit für inkrementelle Entwicklung

Größere Projekte werden immer häufiger auf der Grundlage evolutionärer bzw. inkrementeller Prozessmodelle abgewickelt, während wasserfallartige Prozessmodelle auf dem Rückzug sind. Die Gründe hierfür wurden früher erläutert (vgl. Kap. 1.4).

Die Durchgängigkeit des objektorientierten Ansatzes erleichtert eine inkrementelle Entwicklung, bei der Analyse-, Entwurfs- und Implementierungsphasen einander abwechseln, in hohem Maße. Umgekehrt ist der strukturierte Ansatz aufgrund der dargelegten Inhomogenität der grundlegenden Entwicklungsaktivitäten für eine inkrementelle Entwicklung grundsätzlich weniger geeignet.

(6) Kontinuität und Wandel benutzter Konzepte und Prinzipien

Kontinuität von Konzepten und Prinzipien

Zwischen den objektorientierten Konzepten Objekt und Klasse einerseits und den Konzepten des modularen Entwurfs andererseits besteht eine enge Verwandtschaft. Hierzu können zusammenfassend folgende Aussagen getroffen werden:

- Im modularen Entwurf und im objektorientierten Ansatz werden weitgehend die gleichen Prinzipien angewendet und analoge Konzepte benutzt. Fundamental sind in beiden Fällen Kapselung von Daten und Funktionen in einheitlichen Bausteinen, das Geheimnisprinzip und die Verwendung der funktionalen und der Datenabstraktion zur Komplexitätsbeherrschung.
- Die Datenabstraktion basiert auf der Datenkapselung, d.h. dem ausschließlichen Zugriff auf Daten über entsprechende Zugriffsfunktionen. Der besondere Akzent bei der Datenabstraktion liegt allerdings darauf, die Realisierung komplexer Datenstrukturen zu verbergen.
- Die Objekte des objektorientierten Ansatzes stellen eine Variante abstrakter Datenobjekte im Sinne des modularen Entwurfs dar. Ebenso sind Klassen eine Form abstrakter Datentypen.

was ist neu im objektorientierten Ansatz?

Das Neue des objektorientierten Ansatzes besteht demnach nicht darin, die Konzepte der fortgeschrittensten strukturierten Entwurfsmethode zu nutzen. Es besteht vielmehr darin, sie *nicht* mehr *nur* im Entwurf zu nutzen! Erst der objektorientierte Ansatz hat dazu geführt, dass Datenkapselung, Datenabstraktion, abstrakte Datenobjekte und -typen bereits in der Analyse ihren Platz finden. Eben dies bildet die Basis für die Durchgängigkeit des objektorientierten Ansatzes und deren oben besprochene positiven Auswirkungen.

Neu und konstitutiv für den objektorientierten Ansatz sind ferner die Konzepte der Vererbung und des Polymorphismus, deren Vorteile bereits umrissen wurden. Zwar kennt schon die Entity-Relationship-Modellierung die Vererbung, jedoch nur von Attributen, nicht von Operationen. Zudem blieb es wieder dem objektori-

entierten Ansatz vorbehalten, die Vererbung zu einem durchgängig während der gesamten Entwicklung benutzten Konzept zu erweitern.

(7) Stabilität der Anwendungsarchitektur

Stabilität des Entwurfs bei objektorientiertem Ansatz ...

Die Stabilität einer Anwendungsarchitektur wird umso höher sein, je stärker sie sich an den grundlegenden und langlebigen Gegebenheiten und Anforderungen des Anwendungsbereiches orientiert. Funktionale Anforderungen sind unbeständig. Sie werden während der Entwicklung und des Einsatzes einer Anwendung ständig modifiziert oder erweitert. Auch bei den Anwendungsdaten sind häufige Änderungen die Regel. Am stabilsten sind die grundlegenden realen Objekte und Objekttypen des Anwendungsbereiches bzw. die diese widerspiegelnden Begriffe und fachbezogenen Konzepte.

Der objektorientierte Ansatz zielt bereits in der Analyse darauf ab, die grundlegenden und relevanten Objekttypen des Anwendungsbereiches zu identifizieren. Aus diesen abstrahierte Klassen bilden dann den Kern des fachlichen Modells und der späteren Anwendungsarchitektur. Diese Vorgehensweise führt bei adäquater Umsetzung zu einer stabilen und wartungsfreundlichen Architektur, die später in ihren Grundzügen nicht mehr verändert werden muss. Zur Veranschaulichung der Vorgehensweise diene folgendes aus COAD und YOURDON (1994) entnommene Beispiel; zum Aspekt stabiler Entwürfe vgl. auch MEYER (1990).

Beispiel 2.5

„So werden wir bei der Definition eines Systems zur Flugsicherung ... immer dieselben grundlegenden Klassen-&-Objekte vorfinden, mit denen die Analyse und letztlich auch die Spezifikation strukturiert werden: Flugzeug, Fluglotse, Flugraum usw. Ein teures System wird bei einigen Klassen-&-Objekte mehr Attribute aufweisen. ... Das teure System wird auch bessere Services für die einzelnen Klassen-&-Objekte aufweisen (so kann „Flugzeug“ zum Beispiel einen Service zur automatischen Flugzeuglokalisierung ... beinhalten). ... Und dennoch wird der stabile Aspekt des Systems (die Klassen-&-Objekte der Anwendungsgebiete) gleich bleiben – auch bei größeren Veränderungen der Anforderungsdefinition.“ (COAD und YOURDON 1994, S. 46).

Anmerkung: Klasse-&-Objekte bedeuten bei COAD und YOURDON eine Klasse und ihre Objekte. Services bezeichnen bei ihnen Operationen.

... und strukturiertem Ansatz

Der strukturierte Ansatz begünstigt mit seiner in der Analyse separaten Betrachtung von Funktionen einerseits und Daten andererseits den Entwurf stabiler Architekturen zunächst nicht. Mit dem modularen Entwurf können dennoch – mit relativ hohem Aufwand – stabile und wartungsfreundliche Architekturen erreicht werden. Ältere funktionsorientierte Entwurfsmethoden führen erfahrungsgemäß oft zu unübersichtlichen und wartungsunfreundlichen Architekturen.

(8) Unterstützung der Wiederverwendung

Wiederverwendung in der Implementierung

Feststellen lässt sich, dass der objektorientierte Ansatz zu einer starken Zunahme der Softwarewiederverwendung im Bereich der Implementierung geführt hat. Daran haben vor allem auf dem Markt angebotene Klassenbibliotheken und Frame-

works großen Anteil. Zwar bieten beispielsweise auch prozedurale Ada-Versionen vergleichbare Modulbibliotheken. Jedoch wird die Wiederverwendung im Bereich der objektorientierten Programmierung insgesamt in weitaus höherem Umfang praktiziert.

Was die projektübergreifende Wiederverwendung innerhalb von Unternehmen sowie die Wiederverwendung von Analyse- und Entwurfsmodellen anbelangt, fällt die Bilanz entschieden ungünstiger aus. Zwar liegen positive Einzelbeispiele für beide Entwicklungsansätze vor (vgl. BALZERT 1998). Sie können jedoch nicht verdecken, dass auch die Objektorientierung bisher keinen Durchbruch bei den genannten und besonders wünschenswerten Arten der Wiederverwendung erreicht hat.

Zusammenfassung

Zusammengefasst belegt der Vergleich der beiden grundlegenden Paradigmen der Softwareentwicklung einerseits, dass der objektorientierte Ansatz wesentliche Vorzüge aufweist. Er gestattet vor allem eine Softwareentwicklung „aus einem Guss“ und unterstützt durch die Orientierung an den Objekten des Anwendungsbereiches und die Datenkapselung die Entwicklung stabiler Modelle und wartungsfreundlicher Anwendungen. Andererseits sollten auch Gemeinsamkeiten und der enge Zusammenhang beider Ansätze deutlich geworden sein. Insbesondere kann der modulare Entwurf als Bindeglied zwischen strukturiertem und objektorientiertem Ansatz aufgefasst werden.

geeignete Anwendungsgebiete der Objektorientierung

Ihre Stärken stellt die Objektorientierung insbesondere in den Bereichen interaktive Anwendungen mit grafischen Benutzungsoberflächen sowie verteilte Anwendungen unter Beweis. Hier bietet sich in konzeptioneller Hinsicht, aber auch im Hinblick auf verfügbare Hilfsmittel (Klassenbibliotheken, Frameworks, Standards) eine objektorientierte Entwicklung besonders an.

Allerdings sollte nicht vergessen werden, dass auch der objektorientierte Ansatz nur bei einer sachgemäßen Anwendung zu hochwertiger Software führt.

Übungsaufgaben zu Kapitel 2.4

Übungsaufgabe 2.4.1

Betrachtet sei das DFD der Abb. 2.6. Zu beantworten sind folgende Fragen:

a) Welche Schnittstellen des DFD sind Quellen, welche sind Senken von Anwendungsdaten?

b) Das DFD bildet offenbar nur einen Ausschnitt der gesamten Auftragsverwaltung ab. Nennen Sie eine Funktion, die durch einen zusätzlichen Prozess dargestellt werden könnte.

c) Ein Datenfluss, der in einen Datenspeicher mündet oder von ihm herkommt, kann (typmäßig) auch eine Teilmenge der Daten des Datenspeichers enthalten. In diesem Fall muss der entsprechende Datenfluss im DFD explizit benannt werden. Geben Sie Beispiele für diesen Fall im betrachteten DFD an.

Übungsaufgabe 2.4.2

Betrachtet sei nun eine DFD-Hierarchie, an deren Spitze ein grobes DFD der gesamten Anwendung mit drei Prozessen steht. Jeder Prozess wird seinerseits zu drei weiteren Prozessen in einem hierarchisch nachgeordneten DFD verfeinert.

Dies gelte für alle DFDs nachgeordneter Hierarchieebenen entsprechend. Wie viele DFDs sind anzufertigen, wenn es insgesamt vier Hierarchieebenen geben soll? Wächst die Anzahl der DFDs linear oder quadratisch oder exponentiell mit der Anzahl der Hierarchieebenen?

Übungsaufgabe 2.4.3

Im Anschluss an die Abb. 2.7 wurden bereits Informationen angegeben, die nur das Entity-Relationship-Modell der Kundenauftragsverwaltung, nicht aber das zugehörige DFD der Abb. 2.6 enthält. Können Sie noch weitere analoge Informationen angeben, die nur im ERM der Abb. 2.7 dargestellt werden?

Übungsaufgabe 2.4.4

Zufallszahlen werden z.B. für Simulationsexperimente benötigt und in Programmen durch eine iterative Vorschrift berechnet, die jeweils aus der zuletzt ermittelten Zufallszahl eine weitere Zufallszahl erzeugt. Die zuletzt erzeugte Zufallszahl muss also bis zur Berechnung der nächsten aufbewahrt werden. (Die erforderliche Initialisierung der Zufallszahlenfolge sei hier nicht betrachtet.)

Werden die Zufallszahlen durch separate Aufrufe einer Funktion berechnet, so sind innerhalb der Funktion vereinbarte Variablen nach ihrer Ausführung nicht mehr vorhanden. Die letzte Zufallszahl muss daher außerhalb der Funktion, etwa in einer globalen Variablen abgelegt werden. Diese ist während der gesamten Programmlaufzeit verfügbar, aber auch an jeder Stelle des Programms sichtbar, d.h. lesbar und modifizierbar.

Eine Alternative bietet ein Datenobjekt-Modul, in dessen interner Datenstruktur (Gedächtnis) die jeweils letzte Zufallszahl aufbewahrt wird. Skizzieren Sie ein entsprechendes Datenobjekt-Modul. Geben Sie zum einen den Funktionskopf der einzigen Zugriffsfunktion erzeugeZZahl() an, wobei der Typ der Zufallszahl durch INTEGER zu bezeichnen ist. Zeichnen Sie außerdem den Aufbau des Moduls und begründen Sie kurz den Vorteil dieser Variante gegenüber dem Einsatz einer globalen Variablen.

Übungsaufgabe 2.4.5

Die im modularen Entwurf, also bereits innerhalb des strukturierten Ansatzes genutzte Datenabstraktion wird in Gestalt von Objekten und Klassen in der Objektorientierung konsequent und durchgängig eingesetzt. Beschreiben Sie nochmals knapp drei Vorzüge der Datenabstraktion.

Diese Seite bleibt aus technischen Gründen frei.

Lösungen zu den Übungsaufgaben

Lösung zu Übungsaufgabe 1.1.1 (Aufgabentext S. 14)

a) Systemsoftware.
b) Anwendungssoftware.
c) Systemsoftware.
d) Anwendungssoftware, betriebliches Anwendungssystem.
e) Systemsoftware.
f) Anwendungssoftware.
g) Anwendungssoftware, betriebliches Anwendungssystem.
h) Systemsoftware.
i) Anwendungssoftware.

Lösung zu Übungsaufgabe 1.1.2 (Aufgabentext S. 14)

a) Softwareentwicklung.
b) Systementwicklung.

Lösung zu Übungsaufgabe 1.1.3 (Aufgabentext S. 14)

Eine zusätzliche Komplexitätsdimension ist die Zustands- und Zeitkomplexität. Die Zustands- und Zeitkomplexität ist umso höher, je stärker die Ausführbarkeit von Systemfunktionen bzw. ihre Ergebnisse von verschiedenen möglichen Zuständen des Systems zur Laufzeit und zeitlichen Bedingungen abhängen. Das Betriebssystem eines PC-Netzwerkes muss z.B. in der Lage sein, konkurrierende Zugriffe auf Server zeitlich zu koordinieren.

Lösung zu Übungsaufgabe 1.1.4 (Aufgabentext S. 14)

a) Datenkomplexität.
b) Funktionale Komplexität.
c) Algorithmische Komplexität.
d) Algorithmische Komplexität.
e) Funktionale Komplexität.
f) Zustands- und Zeitkomplexität.
g) Oberflächenkomplexität.

Lösung zu Übungsaufgabe 1.1.5 (Aufgabentext S. 15)

Wird die leichte Änderbarkeit von Software als Merkmal dieser Produktgattung ausgewiesen, so ist gemeint, dass die bei anderen technischen Produkten physisch bedingten Widerstände gegen Änderungen entfallen. Software ist „nur Text“ – und insofern leicht änderbar. Wenn von zunehmenden Anforderungen an die Änderbarkeit von Softwareprodukten die Rede ist, so ist gemeint, dass Softwaresysteme mit möglichst geringem Aufwand an veränderte fachliche Anforderungen anpassbar sein sollen. Dies hängt davon ab, ob Softwaresysteme bei ihrer Ent-

wicklung geeignet strukturiert wurden. Ein Widerspruch zwischen beiden Aussagen liegt nicht vor.

Lösung zu Übungsaufgabe 1.1.6 (Aufgabentext S. 15)

a) Geänderte gesetzliche Bestimmungen für den Jahresabschluss eines Unternehmens, die z.B. in Systemen der Finanzbuchhaltung nachvollzogen werden müssen.

b) Erstmaliger Einsatz eines betrieblichen Anwendungssystems in ausländischen Filialen eines Unternehmens. Das System muss den Benutzern nun eine Benutzerführung und Hilfefunktionen in mehreren Sprachen anbieten.

c) Umstellung der betrieblichen Informationsverarbeitung von einem Dateiverwaltungssystem auf ein Datenbanksystem. Die Datenhaltung der betroffenen Anwendungssysteme muss entsprechend portiert werden.

Lösung zu Übungsaufgabe 1.2.1 (Aufgabentext S. 17)

Die Definition des IEEE Standards hebt neben dem Erfordernis einer systematischen Vorgehensweise bei der Softwareentwicklung vor allem die verschiedenen grundlegenden Aktivitäten bei der Softwareentwicklung und -anwendung hervor. Die Definition von HESSE et al. betont hingegen die systematische Bereitstellung und Verwendung der Instrumente des Software Engineering.

Lösung zu Übungsaufgabe 1.2.2 (Aufgabentext S. 17)

In betrieblichen Anwendungssystemen werden sehr häufig Such- und Sortieralgorithmen benötigt. Dies gilt insbesondere für Administrationssysteme zur Unterstützung einfacher operativer Aufgaben im Leistungsprozess von Unternehmen. Diese verarbeiten aktuelle betriebliche Daten im Massenumfang. Um auf einzelne benötigte Daten oder auf eine bestimmte Datenmenge in gewünschter Reihenfolge effizient zugreifen zu können, werden geeignete Such- und Sortierverfahren eingesetzt. Zu ihrer Entwicklung greift man auf entsprechende grundlegende Algorithmen der Informatik zurück.

Lösung zu Übungsaufgabe 1.3.1 (Aufgabentext S. 24)

Die Entwicklungsdokumente sollten untereinander in einem konsistenten Zustand gehalten werden. Das gilt insbesondere für das Pflichtenheft und das Fachkonzept, weil einerseits das Fachkonzept Ausgangspunkt der späteren Realisierung der Anwendung ist, während andererseits das Pflichtenheft meist das einzige mit dem Auftraggeber verbindlich vereinbarte Dokument ist. Daher müssen fehlende oder falsch beschriebene Anforderungen im Pflichtenheft ggf. später nachgetragen bzw. korrigiert werden. Hierbei ist eine Abstimmung mit dem Auftraggeber erforderlich.

Lösung zu Übungsaufgabe 1.3.2 (Aufgabentext S. 24)

a) Intentionstreue.

b) Vollständigkeit.

c) Konsistenz.

d) Eindeutigkeit/Vollständigkeit.

Lösung zu Übungsaufgabe 1.3.3 (Aufgabentext S. 25)

a) Zusammenwirken mit anderen Softwaresystemen.

b) Programmiersprache und Wiederverwendung von Softwarebausteinen.

c) Zielplattform (Betriebssystem) und Variante der Datenhaltung.

Lösung zu Übungsaufgabe 1.3.4 (Aufgabentext S. 25)

Zur Implementierung gehören z.B. auch die Erstellung von Hilfsprogrammen für eine Datenkonvertierung bei der Übernahme von Daten aus Altsystemen oder die Einrichtung von Tabellen bei der Benutzung relationaler Datenbanksysteme.

Lösung zu Übungsaufgabe 1.3.5 (Aufgabentext S. 25)

a) Relevant, z.B. für Statistiken.

b) Nicht relevant.

c) Relevant, z.B. für Korrespondenzzwecke.

d) Relevant, z.B. für Bestimmung der Bearbeitungsdauer der Diplomarbeit.

e) Nicht relevant, da Promotion im Rahmen der angebotenen Studiengänge nicht möglich.

Lösung zu Übungsaufgabe 1.4.1 (Aufgabentext S. 34)

Die Wartungsphase ist, wie in Kap. 1.4.1 erläutert, keine Phase der ursprünglichen Projektentwicklung. Eine spätere Wartung wird vielmehr meist als eigenständiges neues Projekt abgewickelt. Daher wird auch von der Wartungsphase nicht in Phasen der Projektentwicklung zurückverzweigt.

Lösung zu Übungsaufgabe 1.4.2 (Aufgabentext S. 34)

Bei den im Rahmen des Wasserfallmodells betrachteten Prototypen handelt es sich um Wegwerfprototypen, die lediglich zur Klärung und Lösung aktueller, daher meist phaseninterner Problemstellungen angefertigt und anschließend nicht mehr weiter benutzt werden. Daher werden sie typischerweise auch nicht in die jeweils nächste Phase weitergereicht.

Lösung zu Übungsaufgabe 1.4.3 (Aufgabentext S. 34)

Wie bei der Charakterisierung der evolutionären Entwicklung ausgeführt, werden Wartungsmaßnahmen einschließlich Fehlerkorrekturen oder Portierungen generell als Erstellung einer neuen Systemversion durchgeführt. In Abb. 1.7 ist also bei einer Wartung die „Entwicklungsschleife“ ein weiteres Mal zu durchlaufen. Die auf eine Testphase folgende Phase wird daher in Abb. 1.7 als „Einsatz“ bezeichnet.

Lösung zu Übungsaufgabe 1.4.4 (Aufgabentext S. 35)

Bei dem evolutionären Prozessmodell werden in jeder Version diejenigen Anforderungen des Auftraggebers umgesetzt, die er bereits vollständig überblickt oder die für ihn momentan am wichtigsten sind. Im Vergleich zum Wasserfallmodell bedeutet dies eine wesentlich höhere Einflussnahme des Auftraggebers auf den

gesamten Entwicklungsprozess, die zu einer schwer kontrollierbaren Dynamik des Projekts führen kann. Man beachte insbesondere, dass Anforderungen des Auftraggebers für weitere Versionen von seinen schlecht vorhersehbaren Erfahrungen mit vorherigen Versionen abhängen werden. Planung und Fortschrittskontrolle werden also schwieriger, weil noch zu erwartende Anforderungen und bereitzustellende Ressourcen schwerer abzuschätzen sind. Dies gilt grundsätzlich für beide vorgestellten Varianten des evolutionären Modells, jedoch in höherem Grade für die Variante mit partieller Analyse, weil hier am Anfang nicht der Versuch unternommen wird, bereits die Gesamtheit der Anforderungen zu modellieren.

Lösung zu Übungsaufgabe 1.4.5 (Aufgabentext S. 35)

Version 1: Realisierung der Funktionen „Reise buchen“ (einschließlich Reservierung von Hotelzimmern) und „Reise abrechnen“ sowie der Verwaltung benötigter Stammdaten (Hotels, Kunden, Tarife usw.).

Version 2: Zusätzliche Realisierung der Funktion „Fluginformationsdienst“.

Version 3: Zusätzliche Realisierung der Funktion „Auswertung“.

Lösung zu Übungsaufgabe 1.5.1 (Aufgabentext S. 40)

a) Grafische Notation, da grafische Elemente für Steueranweisungen benutzt werden.

b) Semiformale Notation, da nach bestimmten Regeln verwendete grafische Elemente mit frei wählbarem Text kombiniert werden.

Lösung zu Übungsaufgabe 1.5.2 (Aufgabentext S. 40)

Eine Modellierung, bei der zuerst nach den Entitätstypen und ihren Beziehungstypen gesucht wird, bevor deren Attribute bestimmt werden, folgt einem Top-Down-Ansatz. Werden dagegen zuerst relevante elementare Daten gesammelt, bevor die Entitätstypen und ihre Beziehungstypen selbst identifiziert werden, liegt ein Bottom-Up-Ansatz vor.

Lösung zu Übungsaufgabe 1.6.1 (Aufgabentext S. 42)

Nachfolgend werden pro Anwendungsfeld beispielhaft Auszüge aus den Core Features von Together angegeben.

(1) Unterstützung modellierender Aktivitäten einschließlich Forward Engineering und Reverse Engineering:
Simultaneous Round-Trip Engineering: Java, ..., and C++,
UML 1.3 Diagrams (Class, Use Case, Sequence Diagram etc.),
Data Modeling (Entity Relationship Diagram).

(2) Integrierte Werkzeuge für die Implementierung:
Robust Programming Editor (Java, C++),
Distributed, Multithreaded Java Debugger.

(3) Erzeugung, Änderung und Verwaltung von Entwicklungsdokumenten:
Multi-level, Flexible Documentation Generation,
(umfasst u.a.) Net-Ready HTML Doc Generation.

(4) Unterstützung der Qualitätssicherung:
Quality Assurance umfasst u.a. Metriken für Java und C++.

Anmerkung: Das Round Trip Engineering beinhaltet sowohl das Forward Engineering wie auch das Reverse Engineering. Auf Metriken wird kurz in Kap. 1.8.1 und 2.4.1 eingegangen.

Lösung zu Übungsaufgabe 1.7.1 (Aufgabentext S. 53)

Vielfach weisen die in einem Unternehmen abzuwickelnden Softwareprojekte starke Analogien auf, die u.a. ihren Inhalt und den Schwierigkeitsgrad betreffen. Eine unternehmensweite Standardisierung von Prozessmodellen führt zu Rationalisierungseffekten, weil die Beteiligten mit dem Projektablauf und einzuhaltenden Regeln und Konventionen aus früheren analog abgewickelten Projekten bereits vertraut sind. Die Einheitlichkeit der angewendeten Methoden und Werkzeuge führt auch dazu, dass diese mit zunehmender Projekterfahrung besser beherrscht werden, was sich vorteilhaft auf die Qualität später erstellter Produkte auswirkt. Schließlich gestatten einheitliche Prozessmodelle einfachere und aussagekräftigere Vergleiche zwischen verschiedenen Projekten. Allerdings ist zu beachten, dass neuartige Projekte mit bisher nicht gekanntem Schwierigkeitsgrad auch eine Modifizierung des zuvor benutzten Prozessmodells erforderlich machen können.

Lösung zu Übungsaufgabe 1.7.2 (Aufgabentext S. 54)

Der personelle Aufwand und die Entwicklungsdauer eines Projekts werden maßgeblich reduziert, wenn zum größten Teil vorhandene Software wiederverwendet wird. Als Faustregel gilt, dass in diesem Fall nur etwa 25% der Ressourcen und der Entwicklungsdauer einer Neuentwicklung benötigt werden (vgl. BALZERT 1996). Kann einerseits ein erheblicher Teil einer zu entwickelnden Anwendung durch Wiederverwendung und leichte Modifizierung erstellt werden, während für den anderen Teil eine Neuentwicklung erforderlich ist, so wird man den personellen Aufwand für beide Teile zunächst getrennt schätzen und bei dem ersten Teil die „25%-Regel“ bezogen auf dessen Umfang anwenden.

Lösung zu Übungsaufgabe 1.7.3 (Aufgabentext S. 54)

a) Tauglicher Meilenstein. Aussage kann ggf. anhand von Testprotokollen für die einzelnen Bausteine geprüft werden.
b) Untauglicher Meilenstein. Aussage zu vage, keine Überprüfung möglich.
c) Tauglicher Meilenstein. Die Vollständigkeit der Testfälle kann ggf. anhand des Pflichtenheftes geprüft werden.
d) Untauglicher Meilenstein. Aussage zu vage, keine Überprüfung möglich.

Lösung zu Übungsaufgabe 1.7.4 (Aufgabentext S. 54)

Zusätzliche Mitarbeiter erfordern einen erhöhten Aufwand für die Kommunikation im Projekt. Dies gilt vor allem unmittelbar nach der Aufnahme neuer Mitarbeiter (Einarbeitung, Schulung) und kann die Produktivität der bereits eingearbeiteten Projektmitglieder reduzieren. Ein bereits stark in Terminverzug geratenes Projekt kann durch eine Erweiterung des Projektteams Rückstände meist nicht mehr aufholen. Diese Erfahrung wurde als sogenanntes BROOKS’sches Gesetz formuliert: “Adding manpower to a late project makes it later.” (BROOKS 1975). Das bedeutet freilich nicht, dass sich eine Projektteamerweiterung unter allen Umstän-

den negativ auswirkt. Wesentlich sind die Wahl des Zeitpunktes und die Gestaltung effektiver Kommunikationsbeziehungen.

Lösung zu Übungsaufgabe 1.8.1 (Aufgabentext S. 64)

Werden wie im Beispiel der Aufgabe Alias-Typnamen eingeführt, so verfolgt dies den Zweck, in zugehörigen Variablen ausschließlich Werte der durch den Typnamen angegebenen Größenart zu speichern. Zuweisungen der im Aufgabentext angegebenen Art sind daher nicht sinnvoll und verweisen auf semantische (inhaltliche) Fehler. Vom Standpunkt der konstruktiven Qualitätssicherung bedeutet die Akzeptanz solcher Anweisungen durch den Compiler, dass eine Möglichkeit vorausschauender Fehlervermeidung verschenkt wird. Anders gesagt, wäre es im Sinne der konstruktiven Qualitätssicherung besser, die Einführung problembezogener Alias-Typnamen mit einem entsprechenden Zuweisungsverbot zu kombinieren. Diese Idee ist z.B. in den Sprachen Ada und Eiffel auch realisiert worden.

Lösung zu Übungsaufgabe 1.8.2 (Aufgabentext S. 64)

Ein Programmtest gilt als erfolglos, wenn er keinen Programmfehler aufdeckt und als erfolgreich im gegenteiligen Fall. Ein Test ist also stets destruktiver Natur. Die Autoren eines Programms, die es entwickelt und viel Mühe darauf verwendet haben, werden wenig Neigung verspüren, dessen Fehler in Tests herauszufinden. Sie können gegenüber ihrem eigenen Produkt kaum eine destruktive Haltung einnehmen. Hieraus ergibt sich das Erfordernis einer personellen Trennung von Entwicklung und Test.

Lösung zu Übungsaufgabe 1.8.3 (Aufgabentext S. 65)

Es müssen die Äquivalenzklassen LEDIG, VERHEIRATET, GESCHIEDEN, VERWITWET sowie außerdem die Klasse der ungültigen Werte betrachtet werden. Die ersten vier Klassen sind durch ihren zugehörigen Wert, d.h. 0, 1, 2 bzw. 3 zu repräsentieren. Da die Variable *familienstand* eine ganzzahlige Variable ist, kann die Klasse der ungültigen Werte durch eine beliebige andere ganze Zahl, z.B. 5, repräsentiert werden.

Lösung zu Übungsaufgabe 1.8.4 (Aufgabentext S. 65)

Unterstellt wird, dass auch die Eingabe des Wertes 0.0 für die Seitenlänge zulässig sei. Damit ist auch ein Flächeninhalt von 0.0 Flächeneinheiten ein gültiges Ergebnis, das übrigens auch bei der Eingabe sehr kleiner positiver Zahlen entstehen kann. Folgende Äquivalenzklassen sind zu berücksichtigen, die durch die angegebenen Werte repräsentiert werden können:

(1) negative Werte, -1.2,

(2) Nullwert, 0.0,

(3) positive Werte, 1.5,

(4) Werte, die betragsmäßig größer als die größte darstellbare reelle Zahl sind, ein Repräsentant ist maschinen- bzw. compilerabhängig zu wählen.

Lösung zu Übungsaufgabe 1.8.5 (Aufgabentext S. 65)

Bei einer Beschränkung auf gültige Eingabewerte ergeben sich alle zu betrachtenden Testfälle aus der Gesamtheit der geordneten Paare bzw. Kombinationen (*i*, *j*), wobei *i* den Wert von *ampel1*, *j* den Wert von *ampel2* angibt und *i* und *j* die Werte 0, 1 und 2 unabhängig durchlaufen. Es sind also neun Testfälle zu berücksichtigen.

Lösung zu Übungsaufgabe 1.8.6 (Aufgabentext S. 65)

Zwei wesentliche Vorteile der Wiederverwendung bestehen in der Reduzierung des Entwicklungsaufwands und der Projektkosten einerseits und der Erhöhung der Softwarequalität andererseits. Die Erhöhung der Softwarequalität wirkt sich in einer verbesserten Wartbarkeit der Produkte aus, führt also abermals zu verringerten Kosten bei der Produktwartung.

Lösung zu Übungsaufgabe 1.8.7 (Aufgabentext S. 65)

Sollen Softwarekomponenten für eine Wiederverwendung geeignet sein, so müssen sie zum einen eine hohe Qualität aufweisen, wobei je nach Art der Softwarekomponente (Analysemodell, Entwurfsmodell, Programmkomponente) verschiedene konkrete Qualitätsmaßstäbe anzulegen sind. Ferner müssen zur Wiederverwendung geeignete Softwarekomponenten einen hohen Allgemeinheitsgrad besitzen, d.h. allgemeine Lösungen für bestimmte Probleme darstellen. Dies trifft z.B. für die im objektorientierten Entwurf verwendeten Entwurfsmuster zu (vgl. Kap. 2.3). Schließlich müssen wiederverwendbare Komponenten besonders gut dokumentiert sein.

Lösung zu Übungsaufgabe 2.1.1 (Aufgabentext S. 72)

Ausgehend von der Beschreibung des Anwendungsbereiches können zunächst folgende Klassen vorgesehen werden: Lieferant, Kunde, Lieferantenauftrag, Kundenauftrag, Auftragsposition und Artikel. Vgl. aber Aufgabe 2.1.3.

Lösung zu Übungsaufgabe 2.1.2 (Aufgabentext S. 72)

Von einem Kunden- oder einem Lieferantenauftrag wird bei den im Aufgabentext angegebenen Arbeitsschritten auf die zugehörigen Auftragspositionen und von diesen auf die zugehörigen Artikel zugegriffen, d.h. es werden entsprechende Botschaften versendet. Auf diese Weise kann z.B. ein bereits erfasster Kundenauftrag mit allen zugehörigen Daten ausgegeben werden. Um den jederzeitigen direkten Zugriff auf alle Bestandteile bzw. Daten eines Kunden- oder Lieferantenauftrages zu ermöglichen, wird man daher folgende Assoziationen vorsehen:

- Assoziation von Objekten der Klasse Kundenauftrag bzw. Lieferantenauftrag zu den zugehörigen Objekten der Klasse Auftragsposition,
- Assoziation von den Objekten der Klasse Auftragsposition zu dem jeweils zugehörigen Objekt der Klasse Artikel.

Lösung zu Übungsaufgabe 2.1.3 (Aufgabentext S. 72)

Folgende Vererbungsbeziehungen bieten sich an:

- Die Klassen Kundenauftrag bzw. Lieferantenauftrag werden zu der Klasse Auftrag generalisiert, die also zur übergeordneten Klasse wird. Diese enthält allgemeine Attribute, die unabhängig von der speziellen Auftragsart sind wie z.B. das Erfassungsdatum des Auftrags oder ein Attribut, das den Bearbeitungsstand des Auftrags angibt.
- Die Klassen Kunde und Lieferant werden zu der Klasse Geschäftspartner generalisiert. Die übergeordnete Klasse Geschäftspartner enthält wiederum Attribute, die von der speziellen Art des Geschäftspartners unabhängig sind wie z.B. die Adresse des Geschäftspartners oder den Namen eines Ansprechpartners.

Lösung zu Übungsaufgabe 2.1.4 (Aufgabentext S. 72)

Von Kapselung wird im Zusammenhang mit Klassen gesprochen, weil Daten und Funktionen in einem Baustein, eben einer Klasse, zusammengefasst werden. Dies besagt zunächst keineswegs, dass auf die Daten nur über Funktionen der gleichen Klasse zugegriffen werden kann. Ebenso möglich sind direkte Zugriffe auf die Daten bzw. Attribute einer Klasse von außen, die ohne Inanspruchnahme der Klassenoperationen erfolgen. Dies ist etwa bei Klassen in C++ der Fall, wenn Attribute einer Klasse mit dem Sichtbarkeitsmodus *public* vereinbart werden.

Die Datenkapselung geht über die einfache Kapselung hinaus. Sie erst fordert, dass Daten bzw. Attribute einer Klasse ausschließlich über entsprechende Zugriffsfunktionen eben dieser Klasse gelesen oder modifiziert werden können.

Eine grafische Veranschaulichung der einfachen Kapselung wird die Attribute und Operationen einer Klasse gleichrangig, etwa horizontal nebeneinander plazieren. Am Beispiel der Klasse Artikel (vgl. Abb. 2.3) sieht dies etwa so aus:

Bezeichnung Preis	liesBezeichnung() aendereBezeichnung() liesPreis() aenderePreis()

Das Record-Konzept prozeduraler Sprachen bildet ein Beispiel für die einfache Kapselung, weil verschiedene Daten in einem Baustein vereinigt werden. Eine Datenkapselung liegt nicht vor – wie sollte auch sonst auf die Daten eines Records zugegriffen werden, der ja nur Daten und keine Funktionen kapselt.

Lösung zu Übungsaufgabe 2.2.1 (Aufgabentext S. 75)

Die UML bietet nur ein System von Konzepten sowie eine zugehörige grafische Notation an. Von einem System anstelle einer Menge von Konzepten sollte gesprochen werden, weil die UML-Diagramme selbst auch als Konzepte aufgefasst werden können, die wiederum jeweils einen ganzen Satz elementarer Konzepte kombiniert anwenden. Letztere sind allerdings über die verschiedenen Diagrammtypen hinweg weitgehend einheitlich. Klassen und ihre Objekte erscheinen z.B. sowohl im Klassendiagramm wie auch in Interaktionsdiagrammen.

Die UML bietet jedoch keine methodische Vorgehensweise an. Sie legt weder eine strategische Reihenfolge von Modellierungsschritten fest, noch gibt sie taktische Hinweise zur Anwendung einzelner Konzepte. Darüber hinaus wird den Entwicklern auch überlassen, ob sie einzelne Diagrammtypen überhaupt anwenden.

Lösung zu Übungsaufgabe 2.2.2 (Aufgabentext S. 75)

Die im Aufgabentext genannten Vorgängermethoden der UML sowie weitere Methoden wie etwa die von COAD und YOURDON basieren weitgehend auf den gleichen oder analogen objektorientierten Konzepten, die nur unterschiedlich notiert werden. Die Verwendung verschiedener grafischer Notationen für die gleichen Konzepte in der objektorientierten Softwareentwicklung kann als eine Redundanz des objektorientierten Ansatzes aufgefasst werden, die mit der UML überwunden wurde.

Die Durchsetzung der UML trägt zur besseren Verständlichkeit objektorientierter Modelle bei, erleichtert die Kommunikation der Entwickler innerhalb und zwischen Unternehmen sowie die Kommunikation mit Auftraggebern und Fachexperten. Diese Vorteile spielen vor allem bei sehr großen Projekten eine Rolle, die meist von vielen verschiedenen Unternehmen gemeinsam abgewickelt werden. Vorher verbreitete Verständigungsschwierigkeiten, die allein auf der Benutzung verschiedener Notationen beruhten, treten nicht mehr auf. Die UML beendet also die babylonische Sprachverwirrung in der Objektorientierung.

Hinzu kommt, dass die Entwicklung der UML die Möglichkeit bot, die Stärken der einzelnen Vorgängermethoden bezüglich der Konzepte selbst wie auch ihrer Notation zu kombinieren. So wurde z.B. von der OOSE-Methode von JACOBSON das Konzept der Anwendungsfälle übernommen, das die anderen Methoden nicht kannten.

Lösung zu Übungsaufgabe 2.3.1 (Aufgabentext S. 80)

Das Klassendiagramm enthält zum einen Klassen mit ihren Attributen und ihren Operationen. Mit der Angabe der Operationen aller Klassen wird die funktionale Sicht auf die Anwendung modelliert. Die Attribute aller Klassen entsprechen der Datensicht auf die Anwendung. Diese wird ergänzt durch die Assoziationen, die Verbindungen zwischen den Objekten verschiedener Klassen beschreiben. Die Assoziationen sind der Datensicht zuzurechnen, da sie wesentlich Beziehungen zwischen den Anwendungsdaten abbilden. Die Vererbungsbeziehungen lassen sich teils der funktionalen und teils der Datensicht zuordnen, da sowohl Operationen wie auch Attribute vererbt werden.

Lösung zu Übungsaufgabe 2.3.2 (Aufgabentext S. 80)

Das Klassenlexikon umfasst zusätzliche Angaben zu den Klassen des fachlichen Modells in textueller Form, die in das Klassendiagramm aus Gründen der Übersichtlichkeit nicht aufgenommen werden. Hierzu zählen neben Attributbeschreibungen auch Beschreibungen der Operationen aller Klassen. Diese spezifizieren die Wirkung der Operationen und umfassen bei komplizierteren Operationen auch ihre problembezogene algorithmische Beschreibung.

Lösung zu Übungsaufgabe 2.3.3 (Aufgabentext S. 80)

Auch der Prototyp der Benutzungsoberfläche wird als ein System verschiedener Klassen aufgebaut. Soll sich die Struktur der gesamten Anwendung einheitlich an den grundlegenden realen Objekten des Anwendungsbereiches orientieren, so muss sich dies auch in der Struktur der Klassen der Benutzungsoberfläche widerspiegeln. Diese sollten daher so ausgewählt werden, dass sie jeweils einen engen inhaltlichen Zusammenhang zu gewissen Klassen des Fachkonzepts besitzen. Gibt es etwa im Fachkonzept eine Klasse Auftrag, so wird es in der Benutzungsoberfläche eine oder mehrere Klassen zur Abwicklung von Dialogen der Auftragsbearbeitung geben. Die gewünschte strukturelle Orientierung des später zu komplettierenden Prototypen der Benutzungsoberfläche an der Klassenstruktur des Fachkonzepts bedingt ohne weiteres, dass zunächst das Fachkonzept selbst erstellt wird.

Lösung zu Übungsaufgabe 2.4.1 (Aufgabentext S. 92)

a) Kunde ist eine Datenquelle. Abteilung Verkauf ist sowohl Datenquelle (Datenfluss Stammdaten) als auch Senke (Datenfluss Protokoll) von Anwendungsdaten.

b) Ergänzt werden sollte z.B. eine Funktion zur Ausgabe (Anzeige/Druck) von Kundenaufträgen, was durch einen weiteren Prozess geschehen kann. Bisher gibt es ja keinen Lesezugriff auf den Speicher Kunden-Aufträge, während in einem vollständigen DFD auf jeden Datenspeicher sowohl schreibend als auch lesend zugegriffen wird („what comes in, must go out“).

c) Zur Ergänzung von Kundenaufträgen werden den Speichern Artikelstamm und Kundenstamm Daten entnommen. Dabei wird es sich jedoch nicht um alle Daten eines Artikels bzw. eines Kunden, sondern etwa nur um die Artikelnummer und -bezeichnung sowie die Kundennummer und den Kundennamen handeln. Die Datenflüsse Artikeldaten und Kundendaten werden also weniger Daten als die zugehörigen Datensätze in den genannten Speichern enthalten.

Lösung zu Übungsaufgabe 2.4.2 (Aufgabentext S. 92)

Insgesamt sind 1 + 3 + 9 + 27 = 40 DFDs zu erstellen. Die Anzahl der DFDs wächst unter den gegebenen Voraussetzungen exponentiell – nämlich entsprechend der Formel $(3^n - 1)/2$ – mit der Anzahl der Hierarchieebenen n. Die Betrachtung zeigt, dass der Verfeinerung von DFDs Grenzen gesetzt sind. Sie kann und sollte abbrechen, wenn sich jeder Prozess eines DFDs hinreichend einfach beschreiben lässt. Ein einzelnes DFD sollte nicht mehr als etwa sieben Prozesse umfassen, um nicht unübersichtlich zu werden.

Lösung zu Übungsaufgabe 2.4.3 (Aufgabentext S. 93)

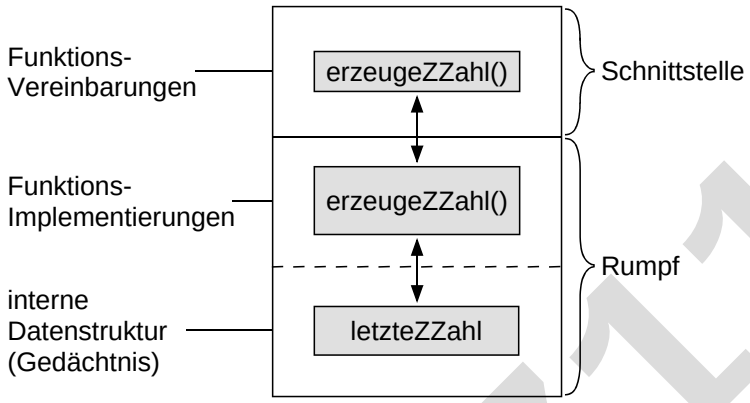
Dem ERM lässt sich entnehmen, dass zu einem Artikel sowie zu einem Kunden jeweils beliebig viele (keiner, einer oder mehrere) Aufträge existieren können. Auch diese Informationen werden in dem DFD der Kundenauftragsverwaltung nicht abgebildet.

Lösung zu Übungsaufgabe 2.4.4 (Aufgabentext S. 93)

Der Funktionskopf der einzigen Zugriffsfunktion des Datenobjekt-Moduls lautet:

```
INTEGER erzeugeZZahl();
oder auch:
erzeugeZZahl(out zzahl: INTEGER);
```

Den Aufbau des Moduls zeigt die folgende Abbildung.



Im Vergleich zu der Lösungsvariante mit einer einfachen Funktion und einer globalen Variablen wird mit dem Datenobjekt-Modul ausgeschlossen, dass die letzte Zufallszahl von anderen Programmteilen aus manipuliert wird. Die Datenabstraktion erweist sich gegenüber der reinen funktionalen Abstraktion als überlegen.

Lösung zu Übungsaufgabe 2.4.5 (Aufgabentext S. 93)

Entscheidende Vorzüge der Datenabstraktion sind:

- Entlastung des Anwenders einer Datenabstraktion, der diese benutzen kann ohne die Details der internen Datenstruktur der Datenabstraktion zu kennen.
- Verhinderung unkontrollierter äußerer Zugriffe auf die interne Datenstruktur einer Datenabstraktion.
- Wartungsfreundlichkeit, da bei unveränderten Schnittstellen der Zugriffsfunktionen die interne Datenstruktur ohne Auswirkungen auf anwendende Module beliebig modifiziert werden kann.

Diese Seite bleibt aus technischen Gründen frei.

Literaturverzeichnis

ANSI IEEE STANDARD (1983);
ANSI IEEE Standard Glossary of Software Engineering Terminology 1983.

BALZERT (1996)
Balzert, Helmut: Lehrbuch der Software-Technik. Software-Entwicklung, Spektrum Akademischer Verlag, Heidelberg 1996.

BALZERT (1998)
Balzert, Helmut: Lehrbuch der Software-Technik. Software-Management Software-Qualitätssicherung Unternehmensmodellierung, Spektrum Akademischer Verlag, Heidelberg 1998.

BALZERT (1999)
Balzert, Heide: Lehrbuch der Objektmodellierung. Analyse und Entwurf, Spektrum Akademischer Verlag, Heidelberg 1999.

BOEHM (1981)
Boehm, B. W.: Software Engineering Economics, Prentice Hall, New York 1981.

BOEHM (1984)
Boehm, B. W.: Verifying and Validating Software Requirements and Design Specifications, in: IEEE Software, Jan. 1994, pp. 75 – 88.

BOEHM (1989)
Boehm, B. W.: Software Risk Management, IEEE Computer Society Press, Washington 1989.

BOEHM (1991)
Boehm, B. W.: Software Risk Management: Principles and Practices, in: IEEE Software, Jan. 1991, pp. 32 – 41.

BOOCH (1991)
Booch, G.: Object-oriented Analysis and Design with Applications, The Benjamin / Cummings Publishing Company, Redwood City 1991.

BOOCH (1994)
Booch, G.: Objektorientierte Analyse und Design: Mit praktischen Anwendungsbeispielen. Addison-Wesley, Bonn 1994.

BROOKS (1975)
Brooks, F. P.: The Mythical Man-Month. Addison-Wesley, Reading 1975.

COAD und YOURDON (1991a)
Coad, P., Yourdon, E.: Object-oriented Analysis, 2. Auflage, Yourdon Press, Prentice Hall, Englewood Cliffs 1991.

COAD und YOURDON (1991b)
Coad, P., Yourdon, E.: Object-oriented Design, Yourdon Press, Prentice Hall, Englewood Cliffs 1991.

COAD und YOURDON (1994)
Coad, P., Yourdon, E.: Objektorientierte Analyse, Prentice Hall, München 1994.

DEMARCO und LISTER (1987)
DeMarco, T., Lister, T.: Peopleware, Dorset House Publishing Co., New York 1987.

DEMARCO und LISTER (1991)
DeMarco, T., Lister, T.: Wien wartet auf Dich. Der Faktor Mensch im DV-Management, Hanser, München 1991.

DIN ISO 9126 (1991)
Informationstechnik – Beurteilen von Softwareprodukten, Qualitätsmerkmale und Leitfaden zu deren Verwendung, 30.9.1991.

DOMSCHKE und DREXL (1995)
Domschke, W., Drexl, A.: Einführung in Operations Research, Springer, Berlin 1995.

FAIRLEY (1985)
Fairley, R.E.: Software Engineering Concepts, McGraw-Hill Book Company, New York 1985.

GAMMA et al. (1995)
Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns. Elements of Reusable Object-Oriented Software, Addison- Wesley, Reading 1995.

GAMMA et al. (1996)
Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software. Addison-Wesley, Bonn 1996.

GANE und SARSON (1979)
Gane, C., Sarson, T.: Structured Systems Analysis, Prentice Hall, Englewood Cliffs 1979.

GEHRING und RÖSCHER(1989)
Gehring, H., Röscher, P.: Einführung in Modula-2 – Programmierung und Systementwicklung, de Gruyter, Berlin 1989.

GEHRING (1992)
Gehring, H.: Algorithmen und Datenstrukturen, FernUniversität Hagen, Fachbereich Wirtschaftswissenschaft, Hagen 1992.

GEHRING (1993a), Gehring, H.: Software Engineering, FernUniversität Hagen, Fachbereich Wirtschaftswissenschaft, Hagen 1993.

GEHRING (1993b)
Gehring, H.: Datenbanksysteme, FernUniversität Hagen, Fachbereich Wirtschaftswissenschaft, Hagen 1993.

GEHRING (1999)
Gehring, H.: Betriebliche Anwendungssysteme, FernUniversität Hagen, Fachbereich Wirtschaftswissenschaft, Hagen 1999.

HAMILTON (1986)
Hamilton, M.: Zero-defect software: The elusive goal, in: IEEE Spectrum, March 1986, pp. 48 – 53.

HESSE et al. (1984)
Hesse, W., Keutgen, A., Luft, A.L., Rombach, H.D.: Ein Begriffssystem für die Softwaretechnik – Vorschlag zur Terminologie, in: Informatik Spektrum 7 (1984), pp. 200 – 213.

HEUER (1992)
Heuer, A.: Objektorientierte Datenbanken. Addison-Wesley, Bonn 1992.

ISO 9000-3 (1992)
DIN ISO 9000 Teil 3 Qualitätsmanagement und Qualitätssicherungsnormen – Leitfaden für die Anwendung von ISO 9001 auf die Entwicklung, Lieferung und Wartung von Software, Beuth Verlag, Berlin 1992.

JACOBSON et al. (1992)
Jacobson, I., Christerson, M., Jonsson, P., Övergaard, G.: Object-Oriented Software Engineering – A Use Case Driven Approach, Addison Wesley, Wokingham 1992.

JONES (1981)
Jones, T. C.: Programming Productivity: Issues for the Eighties, in: Jones, T. C. (Ed.): Program quality and programmer productivity, IEEE Computer Society Press, Washington DC 1981.

KAUBA (1997)
Kauba, E.: Software Re-Use ist eine Frage guter Organisation, in: Computerwoche 2(1997), pp. 13 – 14.

KNÖLL und BUSSE (1991)
Knöll, H.D., Busse, J.: Aufwandschätzung von Software-Projekten in der Praxis, BI Wissenschaftsverlag, Mannheim 1990.

KOONTZ und O'DONNELL (1972)
Koontz, H., O'Donnell, C.: Principles of Management: An Analysis of Managerial Functions, 5. Auflage, McGraw-Hill Book Company, New York 1972.

LIGGESMEIER (1990)
Liggesmeyer, P.: Modultest und Modulverifikation: State of the Art, BI Wissenschaftsverlag, Mannheim 1990.

LIPOW (1982)
Lipow, M.: Number of faults per line of code, in: IEEE Transactions on Software Engineering, July 1982, pp. 437 – 439.

MEYER (1988)
Meyer, B.: Object-oriented Software Construction, Prentice Hall International, London 1988.

MEYER (1990)
Meyer, B.: Objektorientierte Softwareentwicklung, Hanser, Wien und Prentice Hall International, London, 1990.

OESTEREICH (1998)
Oestereich, B.: Objektorientierte Softwareentwicklung. 4. Auflage, Oldenbourg, München 1998.

PAGEL und SIX (1994)
Pagel, B.-U., Six, H.-W.: Software Engineering, Addison-Wesley, Bonn 1994.

PARNAS (1972)
Parnas, D.: A Technique for Software Module Specification with Examples, in: Communications of the ACM 15 (1972) 5, pp. 330 – 336.

ROMBACH und BASILI (1987)
Rombach, H.D., Basili, V.R.: Quantitative Software-Qualitätssicherung, in: Informatik-Spektrum 10(1987), pp. 145 – 158.

RUMBAUGH et al. (1991)
Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.: Object-Oriented Modeling and Design, Prentice Hall, Englewood Cliffs 1991.

RUMBAUGH et al. (1993)
Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.: Objektorientiertes Modellieren und Entwerfen, Hanser, München, Prentice Hall International, London, 1993.

RUMBAUGH (1996)
Rumbaugh, J.: OMT Insights: Collected Writing from the Journal of the Object-Oriented Programming, Prentice Hall, Englewood Cliffs 1996.

SNEED (1987)
Sneed, H.M.: Software Management, Verlagsgesellschaft Rudolf Müller GmbH, Köln 1987.

SOMMERVILLE (1992)
Sommerville, I.: Software Engineering, 4. Auflage, Addison-Wesley, Reading 1992.

STEIN (1994)
Stein, W.: Objektorientierte Analysemethoden. Vergleich Bewertung Auswahl, BI Wissenschaftsverlag, Mannheim 1994.

STROUSTRUP (1998)
Stroustrup, B.: Die C++-Programmiersprache. 3. Auflage, Addison-Wesley, Bonn 1998.

UML (1999)
OMG Unified Modeling Language Specification (draft), Version 1.3, March 1999. Siehe www.rational.com/uml

WEBER (1992)
Weber, H.: Die Software-Krise und ihre Macher, Springer, Berlin 1992.

YOURDON (1988)
Yourdon, E.: Techniques of Program Structure and Design, 2. Auflage, Prentice Hall, Englewood Cliffs 1988.

YOURDON (1989)
Yourdon, E.: Modern Structured Analysis, Prentice Hall, Englewood Cliffs 1989.

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Übungsaufgaben

Index

Diese Seite bleibt aus technischen Gründen frei.

Index zur KE 1

F

G

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

Diese Seite bleibt aus technischen Gründen frei.