Inhalt

Artikel

Referenzen

Artikellizenzen

Strukturierte Analyse

Die **Strukturierte Analyse** (**SA**) ist eine hauptsächlich von Tom DeMarco entwickelte Methode zur Erstellung einer formalen Systembeschreibung im Rahmen der Softwareentwicklung. Sie wird während der Analysephase eines Software-Projekts eingesetzt. Strukturiertes Design verfeinert die Ergebnisse der SA soweit, dass sie dann umgesetzt werden können. Sie ist eine Methode der Systemanalyse.

Das Ergebnis der Strukturierten Analyse ist ein hierarchisch gegliedertes Anforderungsdokument für Umfang und Inhalt der betrieblichen Anwendung, die in dem geplanten Softwaresystem realisiert werden soll. Die Strukturierte Analyse ist eine graphische Analysemethode, die mit Hilfe eines Top-Down-Vorgehens ein komplexes System in immer einfachere Funktionen bzw. Prozesse aufteilt und gleichzeitig eine Datenflussmodellierung durchführt. In ihrer Grundform ist die SA eine statische Analyse, die jedoch später um Methoden für dynamische Analysen erweitert wurde.

Historische Entwicklung

- 1960er-Jahre Bemühungen, den Prozess der Systementwicklung zu systematisieren
- 1974 Vorschlag von JACKSON für eine grafische Darstellungsform für strukturierte Analyse
- 1970er-Jahre Entwicklung der SA von Tom DeMarco und anderen
- 1977 Veröffentlichung der SA von Tom DeMarco und anderen

Strukturierte Analyse

In der Strukturierten Analyse werden folgenden Elemente verwendet:

- Kontextdiagramm (*engl. Context-Diagram*): Dieses Diagramm ist die Wurzel des Analyse-Baums. Es grenzt das System von seiner Umwelt ab und definiert damit, welche Aspekte von der Analyse betrachtet werden und welche nicht.
- Eine hierarchische Darstellung der Struktur der Anwendung in Form einer (einem Organigramm ähnlichen) "Baumstruktur". Deren oberstes Diagramm ist o. g. Kontextdiagramm. Zu jedem tieferen Zweig der Baumstruktur gehört ein "Datenflussdiagramm" (s. u.). Zu jedem "Blatt" der Baumstruktur (Endpunkt, der nicht weiter verfeinert wird) gehört als Beschreibung der darin enthaltenen Prozesse eine "Minispezifikation" (s. u.).
- Datenflussdiagramm (*engl. Data Flow Diagram, kurz DFD*): Ein DFD visualisiert in welche Teilprozesse sich der auf dem DFD dargestellte Prozess aufteilt und wie die Verwendung der Daten in diesem Prozess abläuft.
- Minispezifikation (*engl. Mini-Specification*): Die Mini-Spec ist eine formale Beschreibung eines im Rahmen der Analyse nicht mehr weiter geteilten Elementarprozesses. Die Beschreibung erfolgt mit Hilfe eines Pseudocodes, der nicht genormt ist und im Regelfall von der später verwendeten Programmiersprache **un**abhängig ist, also die logischen Konstrukte der Strukturierten Programmierung verwendet. Weitere Möglichkeiten der Beschreibung sind Entscheidungstabellen und Entscheidungsbäume.
- Datenwörterbuch (*engl. Data Dictionary, kurz DD*): Eine Sammlung aller Datendefinitionen, die in der Analyse verwendet werden.

Die ersten beiden Diagramme verwenden folgende grafischen Elemente:

- Datenfluss, dargestellt als ein Pfeil
- Daten, Beschriftung am Pfeil
- Speicher, zwei parallele waagerechte Linien, dazwischen der Name des Speichers
- Teil- und Elementarprozesse, Kreis mit dem Namen und der Nummer des Teilprozesses in dem Kreis
- Externe Datenempfänger/sender (nur auf dem Kontextdiagramm), Viereck mit eingeschlossenem Namen

Strukturierte Real-Time-Analyse (RT)

Die Strukturierte Real-Time-Analyse erweitert die normale strukturierte Analyse um eine Echtzeitkomponente. Erreicht wird dies durch die Festlegung des Verhaltens der Prozessschicht unter allen möglichen externen und internen Bedingungen und Betriebsarten. Entworfen wurde das System von Imtiaz A. Pirbhai und Derek J. Hatley.

Dynamische Analyse

Neben den Definitionen der Statischen Analyse werden zusätzlich folgende Elemente definiert:

- Entscheidungstabelle (*engl. Decision Table, kurz DT*): Aus mehreren Eingangswerten wird in tabellarischer Form definiert wie der Ausgangswert gesetzt wird.
- Zustandsübergangsdiagramm (*engl. State Transition Diagram, kurz STD*): Zustände werden auf diesem Diagramm als Vierecke und Übergänge als Pfeile dargestellt. Das STD hat Eingangs- und Ausgangswerte, die in Abhängigkeit von den Übergängen und Zuständen gesetzt werden.
- Prozessaktivierungstabelle (*engl. Process Activation Table, kurz PAT*): Die Tabelle beschreibt die Reihenfolge der Aktivierung der in der Tabelle aufgezählten Prozesse.

Ein DFD beinhaltet stets nur eine PAT und beliebig viele DT und STD. Alle drei neuen Elemente werden grafisch durch einen senkrechten Strich dargestellt. Pfeile von links sind die Eingangs-, Pfeile nach rechts die Ausgangsparameter.

- Kontrollflüsse (*engl. Control Flow*): Dargestellt als gestrichelter Pfeil werden über Kontrollflüsse nur Daten mit Boolescher Definition gesendet. Diese dienen der Ansteuerung der DT und STD und tragen selbst keine wahren Daten, sondern dienen nur der Modellierung des dynamischen Ablaufs.

Verwendung in der Praxis

Eins der größten Softwareprojekte, die mit Hilfe der Strukturierten Analyse in Deutschland realisiert wurden, ist die Software für den Zentralrechner des Kampfflugzeugs Tornado.

Ansonsten ist die Strukturierte Analyse vielerorts durch die Unified Modeling Language abgelöst, wird aber noch in vielen Projekten eingesetzt.

Literatur

- Edward Yourdon: *Modern Structured Analysis*, Yourdon Press Computing Series, 1999, ISBN 0135986249
- Keith Edwards: *Real-Time Structured Methods, System Analysis*, Wiley, 1993, ISBN 0-471-93415-1
- Derek J. Hatley, Imtiaz A. Pirbhai: *Strategies for Real Time System Specification*, John Wiley and Sons Ltd, 1988, ISBN 0932633048
- Stephen J. Mellor und Paul T. Ward: *Structured Development for Real-Time Systems: Implementation Modeling Techniques: 003*, Prentice Hall, 1986, ISBN 013854803X
- Tom DeMarco: *Structured Analysis and System Specification.* Prentice Hall, 1979, ISBN 0138543801

Siehe auch

- Softwaretechnik
- Strukturiertes Design (SD)
- Objektorientierte Analyse

Weblinks

- Seminararbeit zur Strukturierten Analyse (statisches Modell, PDF) [1] (288 kB)
- Präsentation zur Seminararbeit, PDF [2] (1,05 MB)

Referenzen

[1] http://ebus.informatik.uni-leipzig.de/www/media/lehre/seminar-pioniere04/kiess-ausarbeitung-demarco.pdf
[2] http://ebus.informatik.uni-leipzig.de/www/media/lehre/seminar-pioniere04/kiess-folien-demarco.pdf

Tom DeMarco

Tom DeMarco (* 20. August 1940 in Pennsylvania, USA) ist der Erfinder der Strukturierten Analyse und hat durch seine Beiträge zum Software-Management das Gebiet der Softwaretechnik entscheidend mitgeprägt. Besonders bekannt wurde er durch ein belletristisches Werk: *Der Termin - Ein Roman über Projektmanagement*. DeMarco steht für die These, dass die entscheidenden Aspekte der Softwareentwicklung menschliche sind, und nicht technische, und was hier die Aufgaben des Managements sind.

Leben

Tom DeMarco war lange Jahre in der Softwareentwicklung, z. B. bei den Bell Laboratories, tätig. Mitte der 1980er hat er u. a. mit Tim Lister die Firma 'The Atlantic Systems Guild' gegründet, bei der er bis heute tätig ist.

Hinsichtlich der Softwaretechnik ist insbesondere sein Beitrag zur Anforderungsspezifikation von Softwareprojekten sowie verschiedene Beiträge zum Software-Management, wie beispielsweise Untersuchungen zu den Einflüssen auf die Produktivität, zu betonen. Seine Bücher zeichnen sich durch viel Humor und die Grundaussage aus, dass der Mensch und nicht die Technik der entscheidende Faktor in der Softwareentwicklung ist.

Er besitzt den *Bachelor of Science in Electrical Engineering* der Universität Cornell, den *Master of Science* der Universität Columbia, ein Diplom der Sorbonne sowie einen Ehrendoktor in *Science* der City University London.

Bücher

- Tom DeMarco: *Structured Analysis and System Specification.* Prentice Hall, 1979, ISBN 0138543801
- Tom DeMarco: *Controlling Software Projects: Management, Measurement, and Estimates.* Prentice Hall, 1986, ISBN 0131717111
- Tom DeMarco: *Warum ist Software so teuer?* Carl Hanser Verlag München Wien 1997, ISBN 3446189025
- Tom DeMarco: *Der Termin.* (Original: *The Deadline: A Novel About Project Management.*) Hanser Fachbuchverlag, Leipzig 1998, ISBN 3446194320
- Tom DeMarco, Timothy Lister: *Wien wartet auf Dich!* (Original: *Peopleware: Productive Projects and Teams*) Hanser Fachbuchverlag, Leipzig 1999, ISBN 3446212779
- Tom DeMarco: *Spielräume* (Original: *Slack – Getting Past Burnout, Busywork, and the Myth of Total Efficiency*) Carl Hanser Verlag, München Wien 2001, ISBN 3446216650
- Tom DeMarco, Timothy Lister: *Bärentango.* Hanser Fachbuchverlag, Leipzig 2003, ISBN 3446223339

- Tom DeMarco et al.: *Adrenalin Junkies & Formular Zombies - Typisches Verhalten in Projekten* (Original: *Project Behaviors: from Adrenalin Junkies to Template Zombies*) Carl Hanser Verlag, München, Wien 2007, ISBN 9783446412545

Auszeichnungen

- 1986 mit dem Warnier Prix D'Informatique für *Excellence in Information Science*
- 1999 mit dem Stevens Award für seine *contributions to the methods of software development*

Weblinks

- Tom DeMarcos Webseite [1]
- Literatur von und über Tom DeMarco [2] im Katalog der Deutschen Nationalbibliothek

Referenzen

[1] http://www.systemsguild.com/GuildSite/TDM/Tom_DeMarco.html
[2] https://portal.d-nb.de/opac.htm?query=Woe%3D112180965&method=simpleSearch

Strukturiertes Design

Strukturiertes Design (**SD**) ist ein Entwurfsmuster in der Softwaretechnik nach Edward Yourdon und Larry Constantine, welches modulares Design unterstützt, um neben der reinen Funktionshierarchie auch die Wechselwirkungen von übergeordneten Modulen zu beschreiben. SD wird mit der Strukturierten Analyse (SA) in der Softwaretechnik verwendet.

Das Strukturierte Design schlägt eine Brücke zwischen der technologieneutralen Analyse und der eigentlichen Implementierung. Im Strukturierten Design werden technische Randbedingungen eingebracht und die Grobstruktur des Systems aus technischer Sicht festgelegt. Es stellt damit die inhaltliche Planung der Implementierung dar.

Die Methodik stellt mittels Strukturdiagrammen funktionale Module hierarchisch dar und zeigt dadurch die einzelnen Aufrufhierarchien der Module untereinander. Ein funktionales Modul besteht aus einer oder mehreren funktionalen Abstraktionen. Diese wiederum stellt eine der ersten Abstraktionsmechanismen dar und gruppiert mehrere zusammengehörende Programmbefehle zu Einheiten (Funktionen). Ein Beispiel wäre die Berechnung der Quadratwurzel sqrt(x). Der Benutzer muss keine Details über die Implementierung wissen, sondern wendet die Funktion nur an. Dafür benötigt er eine entsprechende Schnittstellenbeschreibung, die ebenso zum Strukturierten Entwurf gehört wie das Erstellen der Modulhierarchie. Ein Funktionales Modul besitzt kein internes *Gedächtnis*, das heißt es beinhaltet keine Daten (private Daten), die nur im Modul sichtbar sind. Es kann nur in globalen Daten Informationen hinterlegen (beispielsweise bei der Berechnung einer Zufallszahl). Spätere darauf aufbauende Methoden, wie das Modulare Design (MD), führen abstrakte Datentypen und Datenobjekte ein.

Bei Banken, Versicherungen und im Embedded-Bereich finden noch viele Systementwicklungen mit strukturierten Methoden statt. Insbesondere im Bereich des m-Business werden oft Rechnersysteme verwendet, die über limitierte Ressourcen verfügen, für die eine objektorientierte Realisierung mit ihrem Overhead zu teuer ist. Weiterhin sind im Rahmen der Integration von bestehenden Anwendungen im Rahmen von EAI oft Teilsysteme zu realisieren, die nicht mit objektorientierten Sprachen umgesetzt werden können. Daher würden objektorientierte Analyse und Design falsche Implementierungsvorbereitungen darstellen.

Funktionsorientierte Methode

Aufgaben werden top-down in Teilaufgaben zerlegt und dann diese auf die Module abgebildet (Prinzip der Modularisierung).

Beschreibungsmittel sind Strukturdiagramme in denen die Module und die Verbindungen zwischen Modulen dargestellt werden.

Beispiel

Menü Kundenverwaltung wird unterteilt in *Formular Kunde* und *Bericht Kunde*.

Formular Kunde wird erneut unterteilt in *Aktualisieren* und *Umsatzrabatt*, *Bericht Kunde* in *Seitenansicht* und *Drucken*.

Komponente (Software)

Eine **Komponente** ist in der Softwareentwicklung in Bezug auf Softwarearchitektur ein Teil einer Software.

Definition

Vom lateinischen *componere* abstammend, was „zusammensetzen“ bedeutet, bzw. im Gerundiv *componendum* „Das Zusammenzusetzende“, wird der Komponentenbegriff jedoch inhaltlich unterschiedlich verwendet. Oft wird damit fälschlicherweise ein Software-Modul bezeichnet, was die Ähnlichkeit beider Begriffe verdeutlicht (siehe Kapitel *Komponenten-Interface*).

1996 wurde die Softwarekomponente bei der European Conference on Object-Oriented Programming (ECOOP) folgendermaßen definiert:

> "A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties." [1]

Allgemeiner und im Zusammenhang mit dem neuen Konzept der komponentenbasierten Entwicklung wird eine Komponente beispielsweise folgendermaßen definiert:

> "Eine Software-Komponente ist ein Software-Element, das konform zu einem Komponentenmodell ist und gemäß einem Composition-Standard ohne Änderungen mit anderen Komponenten verknüpft und ausgeführt werden kann." [2]

Eine Komponente zeichnet sich also dadurch aus, dass sie ein Element einer komponentenbasierten Anwendung darstellt und definierte Schnittstellen zur Verbindung mit anderen Komponenten besitzt. Die genaue Form einer Komponente ist abhängig vom jeweiligen Komponentenmodell.

Komponenten-Interface

Das Interface der Komponente ist eine verbindliche *Schnittstelle* (*Interface*) zum Rest der Software. Die Schnittstelle kann daher mit einem Vertrag zwischen der Komponente und dem Rest der Software verglichen werden. Durch die explizite Definition der Kontextabhängigkeiten wird deutlich, wie unabhängig eine Komponente tatsächlich von ihrer Umgebung ist.

Die Schnittstellen und die wohldefinierten Kontextbedingungen ermöglichen die Wiederverwendung der Komponente. Je geringer die Kontextabhängigkeiten einer Komponente sind, desto weniger Anforderungen müssen für den Einsatz einer Komponente erfüllt werden. Daraus folgt: je weniger Abhängigkeiten, desto einfacher ist die Wiederverwendung. Zugleich ermöglichen die geringen Abhängigkeiten eine entsprechend unabhängige Pflege und Entwicklung der Komponente. Andererseits führt die Unabhängigkeit der Komponenten dazu, dass diese Redundanzen beinhalten. Der Entwickler einer Komponente muss daher einen Kompromiss finden.

Die Schnittstelle kann mit einem Vertrag zwischen der Komponente und dem Rest der Software verglichen werden. Ein Interface definiert daher, wie eine Komponente wieder verwendet werden kann. Zugleich definiert sie, wie andere Komponenten mit dieser Komponente interagieren können.

Komponenten, die eine Software erweitern, werden in manchen Fällen auch als *Add-on*, *Modul* oder *Plug-in* bezeichnet. Dabei ist zu beachten, dass dies umgekehrt nicht notwendigerweise der Fall sein muss. So ist es beispielsweise möglich, eine Ansammlung von verschiedenen mathematischen Funktionen als Modul zu bezeichnen. Das Modul ist möglicherweise in seinen Funktionen unabhängig. Wenn es keine allgemein verbindliche Schnittstelle besitzt, genügt das Modul allerdings nicht den Anforderungen einer Komponente.

Interfaces können in verschiedene Typen unterschieden werden. Beispiele von Interfacetypen sind:

- *graphical user interface* (GUI), auch human machine interface (HMI) genannt: Gestattet eine Interaktion der Komponente mit dem Benutzer durch eine grafische Benutzeroberfläche. Sie wird beispielsweise über die Computermaus bedient.
- *command line interface* (CLI): Insbesondere dann von Interesse, wenn Komponenten ohne Zutun des Benutzers durch das System aufgerufen werden sollen, beispielsweise, um in periodischen Abständen immer wiederkehrende Aufgaben abzuarbeiten. Ein solches Interface wird durch Eingabe von Befehlen in eine Kommandozeile angesprochen.
- *data interface*: Erlauben das Ein- und Auslesen von Daten der Komponente. Auf dieses Interface wird programmintern zugegriffen.
- *application programming interface* (API): Durch diese Schnittstelle ist es dem Programmierer und anderen Komponenten möglich, die von der Komponente angebotenen Funktionalitäten und Dienste durch Programmierbefehle anzusprechen. Soweit nicht anders angegeben wird mit Interface im Folgenden immer eine API gemeint sein.

Eine Komponente kann verschiedene Interfaces desselben Typs besitzen. Dies kann beispielsweise nötig sein, um ein und dieselbe Komponente in verschiedene Systeme einzubinden. Dadurch werden die Möglichkeiten einer Wiederverwendung vergrößert.

Auswirkungen

Fehlverhalten

Komponenten können fehlerhaft sein. Dies führt zu einer weiteren Forderung: Unabhängigkeit einer Komponente beinhaltet auch, dass die Komponente ihre möglichen Fehler selbst behandelt. Dadurch wird sie zu einer abgeschlossenen Einheit [1] . Im Fehlerfall ist der Fehler so leichter zu lokalisieren. Eine Ausnahme dieser Regel kann nur gemacht werden, wenn dieses Verhalten Teil des Schnittstellenvertrages ist. Dies führt dazu, dass ein Fehler in der Komponente nicht zu einem fehlerhaften Verhalten der ganzen Komponente führt, da diese sich wie

vertraglich festgelegt verhält.

Wiederverwendung

Die Hauptintention der Komponentenentwicklung ist deren Wiederverwendung [3] . Die Entwicklungskosten amortisieren sich durch die wiederholte Verwendung bei der Softwareentwicklung. Diese wiederum wird durch den Einsatz von Komponenten beschleunigt, da sie im Idealfall nur im Zusammenfügen und Parametrieren von Komponenten besteht.

Anhand der Wiederverwendungsform der Komponente kann diese wie folgt grob aufgeteilt werden:

- *blackbox*: Die Komponente wird als eine abgeschlossene Einheit in das zu entwickelnde System aufgenommen. Diese Komponente kann nicht verändert werden. Über ihren internen Aufbau und ihre Funktionsweise kann ebenfalls keine Aussage gemacht werden. Die Verwendung der Komponente geschieht ausschließlich auf Basis der definierten Schnittstellen und Spezifikationen der Komponente.
- *whitebox*: Die Komponente wird als eine offene Einheit wiederverwendet. Das Wort offen beschreibt, dass die Einheit veränderbar ist. Sie kann an die neuen Anforderungen angepasst werden. Dazu ist ihr interner Aufbau einsehbar und somit analysierbar. Die Komponente wird daher als Softwarefragment betrachtet. Die Verwendung der Komponenten geschieht nicht ausschließlich auf Basis der definierten Schnittstellen, sondern auch durch das Analysieren der aktuellen Umsetzung dieser Komponente.
- *greybox*: Die Zwischenformen von black- und whitebox.

Komponenten zur Entwicklungszeit

Komponenten können beispielsweise in die Entwicklungsumgebung integriert werden. Dann zeigen sie ihre Eigenschaften und ihr Verhalten bereits zur Entwicklungszeit. Für den Programmierer ist dies ein großer Vorteil: Er sieht schon während des Programmierens, wie die Komponente aussehen oder arbeiten wird.

Beispiel einer Komponentenpalette

Erst durch das Verwenden von vorgefertigten Komponenten ist ein Rapid Application Development möglich.

Implementierungen

Standards

In der Software ist die Komponenten-Technologie in der Meinung Vieler ein Eckstein der Softwareentwicklung der nächsten Jahre[1] . Es koexistieren verschiedene Standards. Abgesehen von CORBA sind diese Standards im Allgemeinen programmiersprachen-, anwendungs- oder plattformspezifisch. Sie bilden so genannte Komponentenwelten oder -märkte. Beispiele solcher Welten sind:

- Microsoft (.NET, COM, DCOM, OLE, ActiveX, COM+)
- Object Management Group (CORBA)
- Sun Microsystems (JavaBeans, Servlets, Applets, Enterprise JavaBeans)
- OSGi Alliance (OSGi)

Entwicklungswerkzeuge

Für komponentenbasierte Entwicklungen gibt es spezielle Entwicklungsumgebungen und Programmiersprachen, wie zum Beispiel:

- BlackBox Component Builder
- Component Pascal

Literatur

- Olaf Zwintzscher: Software-Komponenten im Überblick, W3L, 2004, ISBN 3937137602
- Clemens Szyperski: Component Software - Beyond Object-Oriented Programming, Second Edition, 2002, ISBN 0-201-74572-0
- M. D. McIlroy: *Mass produced software components.* In: *Software Engineering, Report on a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968.* 1969, S. 138-155 (txt [4]).

Weblinks

- Components, Remoting Middleware and Webservices – how it all fits together [5] (englische Präsentation)
- UML Components [6] Gute Einführung in Komponentendesign für Applikationskomponenten
- Webservices als Komponenten [7] Diplomarbeit zum Thema Software-Komponenten bei Webanwendungen (PDF-Datei; 2,16 MB)

Quellen

[1] Snoopy; Müller, Martin (Deutsche Übersetzung): *Open Source – kurz & gut.* http://www.oreilly.de/german/freebooks/os_tb/toc.html

[2] William T. Councill, George T. Heineman: *Component-Based Software Engineering*. Addison-Wesley, 2001, ISBN 0-201-70485-4

[3] Dumke, Reiner: *Software Engineering*. Friedr. Vieweg & Sohn Verlagsgesellschaft/GWV Fachverlage GmbH, 4.Auflage, Wiesbaden 2003.

[4] http://www.cs.dartmouth.edu/~doug/components.txt

[5] http://www.voelter.de/conferences/index/detail-1572300803.html

[6] http://www.syntropy.co.uk/umlcomponents/umlcsupport.htm

[7] http://www.schwinkendorf.info/Diplomarbeit-Webservices-als-Komponenten.pdf

Modul (Software)

Ein **Modul** (neutrum, das Modul[1]) ist eine abgeschlossene funktionale Einheit einer Software, bestehend aus einer Folge von Verarbeitungsschritten und Datenstrukturen. Inhalt eines Moduls ist häufig eine wiederkehrende Berechnung oder Bearbeitung von Daten, die mehrfach durchgeführt werden muss.

Module bieten eine Kapselung (*encapsulation*) durch die Trennung von Schnittstelle und Implementierung:

- Die Schnittstelle eines Moduls definiert die Datenelemente, die als Eingabe und Ergebnis der Verarbeitung durch das Modul benötigt werden.
- Die Implementierung enthält den tatsächlichen Programmcode.

Nicht zu verwechseln ist ein Modul mit einer Komponente, die in der Funktionalität eine Hierarchieebene höher angesiedelt sind und hier funktionale Module zu Diensten zusammenfassen.

Ein Modul wird z. B. als Funktion oder Unterprogramm aufgerufen, führt eine Reihe von Verarbeitungsschritten durch und liefert als Ergebnis Daten zurück an das aufrufende Programm. Ein Modul kann selbst weitere Module aufrufen - so ist eine Hierarchie von Programmaufrufen möglich.

Module sind aus mehreren Gründen von Bedeutung:

- Programmlogik wird wiederverwendbar, ohne dass Code redundant erstellt und gepflegt werden muss. Codewiederholungen werden somit vermieden.
- Module können in vielen Programmiersprachen separat kompiliert und in Form von Programmbibliotheken bereitgestellt werden.
- Große, komplexe Programme können durch den Einsatz von Modulen gegliedert und strukturiert werden. Funktionalitäten können nach dem Baukastenprinzip eingebunden und für kommerzielle Anwendungen separat lizenziert werden.
- Mehrere Entwicklergruppen können unabhängig voneinander einzelne Module bearbeiten und testen.

Entwurf und Definition von Modulen und Schnittstellen ist Teil der Designphase in der Softwareentwicklung.

Zu unterscheiden sind Module von den Klassen der objektorientierten Softwareentwicklung:

- Von Klassen können Exemplare in Form von Objekten erzeugt (instanziiert) werden,
- Klassen können Eigenschaften an andere Klassen vererben,
- Polymorphismus erlaubt es Klassen, Eigenschaften zur Laufzeit zu verändern – Beziehungen zwischen Modulen sind statisch.

Das Modulkonzept wurde zuerst von David Parnas publiziert.

Zahlreiche Programmiersprachen unterstützen das Modulkonzept durch integrierte Sprachmittel, beispielsweise Ada, COBOL, D, F, Fortran, Haskell, ML, Modula-2, Oberon und Component Pascal. Daneben sind Skriptsprachen wie Perl, Python, PHP und Ruby zu nennen.

Siehe auch

- Programmbibliothek
- Kernel-Modul

Einzelnachweise

[1] Duden, Band 5, Fremdwörterbuch, 7. neu bearbeitete und erweiterte Auflage, Mannheim 2001

Systemanalyse

Die **Systemanalyse** ist eine praktisch anwendbare Methode der Systemtheorie. Dabei konstruiert der Betrachter des Systems ein Modell eines bereits existierenden oder geplanten Systems zunächst als Black Box und verfeinert dieses im weiteren Verlauf. Dabei hat der Bearbeiter eine Auswahl bezüglich der relevanten Elemente und Beziehungen des Systems zu treffen. Das erstellte Modell ist – insbesondere bei komplexen Systemen – meist ein begrenztes, reduziertes, abstrahiertes Abbild der Wirklichkeit, mit dessen Hilfe Aussagen über vergangene und zukünftige Entwicklungen und Verhaltensweisen des Systems in bestimmten Szenarien gemacht werden sollen. Der Vorgang ist auf nahezu jedes System anwendbar, einschließlich Physik, Biologie, Demografie, Wirtschaft, Geografie, Technik und Informatik.

Definition

Der ganzheitliche Ansatz Systemanalyse ist ein iterativer, heuristischer und rückgekoppelter Prozess, der durch die Dimensionen „Organisation“, „Technologie“ und „Motivation“ gekennzeichnet werden kann.

Arbeitsschritte

1. Erhebung und Analyse einer gegebenen Problemstellung
2. Konkretisierung einer allgemeinen Zielsetzung
3. Festlegen der Systemgrenzen zur Unterscheidung von System und Umwelt.
4. Feststellen derjenigen Systemelemente, die für die Fragestellung als relevant betrachtet werden.
5. Feststellen derjenigen Beziehungen zwischen den Systemelementen, die für die Fragestellung als relevant betrachtet werden.
6. Feststellen der Systemeigenschaften auf der Makroebene.
7. Feststellen der Beziehungen des Systems zur Umwelt bzw. zu anderen Systemen, wenn von der Betrachtung des Systems als isoliertes oder geschlossenes System zum offenen System übergegangen wird.

Darstellung

Darstellung der Analyseergebnisse:

- qualitativ: Concept-Map, Flussdiagramm, Wirkungsdiagramme
- halbquantitativ: Pfeildiagramm (je-desto-Beziehungen)
- quantitativ: x-y-, x-t-Diagramme unter anderem, mathematische Gleichungssysteme

Für die Systemanalyse werden formale und grafische Methoden eingesetzt.

Keith Edwards behilft sich in seinem Werk mit den folgenden Elementen, um damit diverse Muster-Systeme darzustellen:

- DFD (Data Flow Diagramm): Datenflussdiagramm, stellt die Verarbeitung und Speicherung der Datenströme dar.
- STD (State Transition Diagram): Zustandsübergangsdiagramm, zeigt zeitliches Verhalten.
- ERD (Entity Relationship Diagram): Gegenstands-Beziehungs-Diagramm, stellt Datenverknüpfungen zueinander dar.
- ESTD (Entity State Diagram): Gegenstands-Zustands-Diagramm, als Mischform aus STD und ERD. Zeigt Statusänderungen in Abhängigkeit von zeitlichen Ereignissen.

Weiterhin benennt er noch die folgenden theoretisch möglichen Kombinationen, die aber praktisch nur sehr begrenzt zweckdienlich sind:

- Zuordnung zwischen Datenstromdarstellung und Datenspeichern (zur Verifikation).
- Zeitliche Veränderung der Datenverarbeitung durch Steuersignale (zur Funktionskontrolle).

Die Herleitung von Zuständen („States“) durch Ereignisse („Events“) und umgekehrt ist möglich. Eine ständige Begrenzung auf eine für die jeweilige Detaillierungsebene sinnvolle Elementmenge ist nötig, um zu einem tauglichen, sprich durchschaubaren und damit brauchbaren Ergebnis zu kommen. Die Darstellung unterscheidet zwischen Steuerströmen, Datenströmen, Augenblicksereignissen und physikalischen Strömen von Materie oder Energie.

Beispiele

Informatik

Unter Systemanalyse wird in der Informatik die erste Phase im Entwurfsprozess verstanden. Der Systemanalytiker beschreibt die für seine Fragestellung relevanten Systemelemente und deren Beziehungen zueinander (in der Regel mit einem Informationsmodell). Ziel der Systemanalyse ist es zum Beispiel die Umwelt ohne Maschine (Ist-Zustand) zu beschreiben, um ausgehend von diesem Ist-Modell eine Maschine zu planen. Das Soll-Modell zeigt, wie die Maschine aussehen soll. Durch die Unterschiede zwischen Ist- und Soll-Modell wird deutlich, was die zu konstruierende Maschine leisten soll. Im Rahmen der Systemanalyse wird nicht untersucht, wie die Maschine implementiert wird. Als *Maschine* sind in diesem Zusammenhang Hardware und Software als eine Einheit zu verstehen. Die Systemanalyse kann auch vor der Optimierung, Migration und Konvertierung von Systemen eingesetzt werden.

Anwendungssystementwicklung

Bei der Erstellung von Anwendungssystemen im betriebswirtschaftlichen Kontext oder bei der Anpassung von Standardsoftware („customizing“) kann es sinnvoll sein, die relevanten Geschäftsprozesse (zum Beispiel mit ereignisgesteuerten Prozessketten [EPK]) zu modellieren. Diese Modelle dienen nicht nur als Grundlage zur Planung organisatorischer Maßnahmen (Prozessmanagement), sondern eignen sich ebenso zur Anforderungsermittlung für Anwendungssysteme, um Geschäftsprozesse medienbruchfrei und effizient durch IT zu unterstützen.

Literatur

- Norbert Bischof: *Struktur und Bedeutung. Eine Einführung in die Systemtheorie für Psychologen*, (2. Aufl.), 1998, ISBN 3456830807 (mit einer Einführung in die Methoden der mathematischen Systemanalyse – einschließlich Z-Transformation – nur mit Abiturmathematik als Voraussetzung)
- Keith Edwards: *Real-Time Structured Methods, System Analysis*, Wiley, 1993, ISBN 0-471-93415-1
- Andreas Häuslein: *Systemanalyse*, 2003, ISBN 3800727153
- Diederich Hinrichsen, Anthony J. Pritchard: *Mathematical Systems Theory*, Springer, Heidelberg, 2005, ISBN 978-3-540-44125-0
- Dieter M. Imboden, Sabine Koch: *Systemanalyse – Einführung in die mathematische Modellierung natürlicher Systeme*, Berlin, 2003, ISBN 3540439358 (Grundlagen-Lehrbuch. Schwerpunkt: Ökologie.)
- Andrei Korotajew, Artemy Malkov, Daria Khaltourina: *Introduction to Social Macrodynamics: Compact Macromodels of the World System Growth.* [1] Moscow: URSS, 2006. ISBN 5-484-00414-4
- Krallmann, Schoenherr, Trier (2007): *Systemanalyse im Unternehmen – prozessorientierte Methoden der Wirtschaftsinformatik* , Oldenbourg Verlag Muenchen Wien, 5.Auflage, ISBN 3486584464
- Michael Rudolf Luft: *Systematik Die universale Systemtheorie Ein interdisziplinäres Basismodell und allgemeingültiges Ordnungsschema allen Seins*, Berlin 2006, ISBN 3939000949

Siehe auch

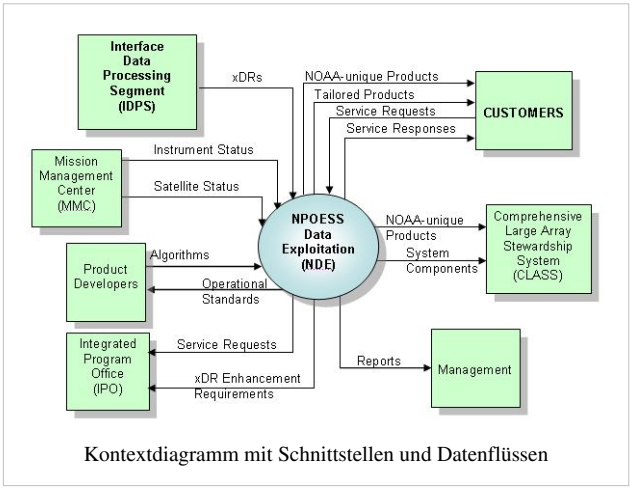
- Anforderungserhebung
- Regelkreis
- Regler
- Strukturierte Analyse
- Strukturiertes Design
- Systemidentifikation

Referenzen

[1] http://cliodynamics.ru/index.php?option=com_content&task=view&id=124&Itemid=70

Kontextdiagramm

Ein **Kontextdiagramm** dient der Modellierung einer Systemumgebung in einer frühen Entwurfs- oder Analysephase (siehe Strukturierte Analyse). Das Kontextdiagramm stellt die oberste Hierarchieebene von Datenflussdiagrammen dar. Es handelt sich um ein abstraktes Datenflussdiagramm, mit dem die Schnittstellen des Systems zu dessen Umwelt abgebildet werden. Es besteht aus genau einem Prozess, der als Kreis dargestellt und mit der Nummer 0. versehen wird. Dieser Prozess wird schrittweise feiner gegliedert und in seine Bestandteile zerlegt. Die Komponenten, die mit dem System interagieren werden als Rechtecke dargestellt, Datenflüsse werden durch Pfeile vom oder zum Prozess repräsentiert. Das Kontextdiagramm stammt ursprünglich aus der Wirtschaftsinformatik und wird zum Beispiel bei Enterprise Application Integration verwendet, um die Anforderungen an ein System zu visualisieren. Nur wenn die *strukturierte Analyse* zur *dynamischen Analyse* erweitert wird, sind auch die sonst nicht vorgesehenen Kontrollflüsse darstellbar.



Kontextdiagramm mit Schnittstellen und Datenflüssen

Bei Verwendung der UML übernimmt gewöhnlich das Anwendungsfalldiagramm die Rolle des Kontextdiagramms.

Hierarchie

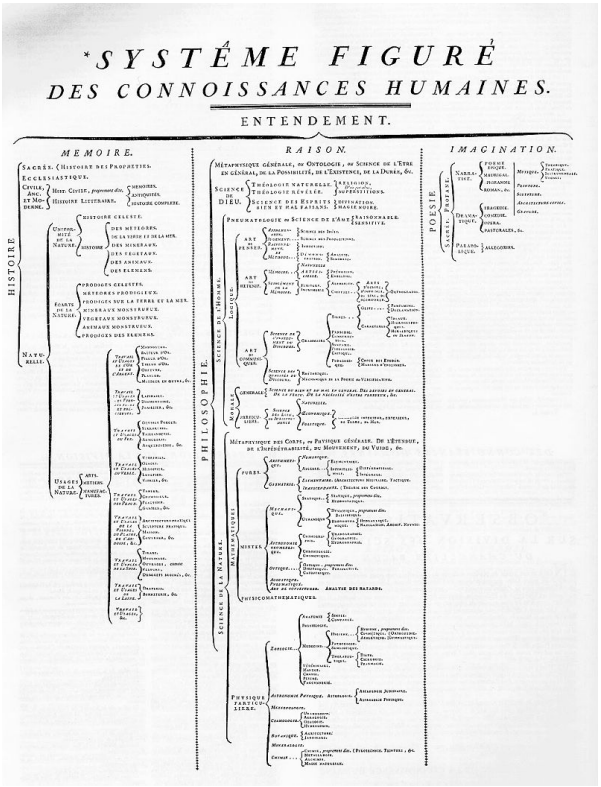
Als **Hierarchie** (gesprochen [hirarˈçiː] oder [hi̯erarˈçiː], altgr. ιεραρχία, ein Kompositum aus ιερός, *hierós* ‚heilig' und αρχή, *arché*, Anfang, Führung bzw. ‚Herrschaft', daraus ab dem 17. Jahrhundert kirchenlateinisch *hierarchia*: Rangordnung der Weihen) bezeichnet man ein System von Elementen, die einander über- bzw. untergeordnet sind. Im Sinne der Monohierarchie ist dabei jedem Element höchstens ein anderes Element unmittelbar übergeordnet, während bei einer Polyhierarchie auch mehrere über- und untergeordnete Elemente möglich sind.

Mathematisch betrachtet bedarf eine Hierarchie einer Ordnungsrelation, die einen Baum (Monohierarchie) oder gerichteten azyklischen Graphen (Polyhierarchie) definiert. Das Komplement ist die Heterarchie.

Die Einteilung (Klassifizierung) oder Einordnung (Klassierung) von Objekten in eine Hierarchie impliziert häufig eine Wertigkeit, die bereits in der Rangordnung, nach der die Objekte geordnet werden, enthalten ist. Grundsätzlich sind sie allerdings einfacher als komplexe Netzwerkstrukturen zu erfassen.

Beispiele

Spezielle Arten von hierarchischen Strukturen, die auch häufig synonym verwendet werden, sind Klassifikationen oder Taxonomien. In der Biologie werden Lebewesen nach verschiedenen Kriterien in einer hierarchischen Systematik geordnet, während der Stammbaum einer Person theoretisch eindeutig bestimmbar ist.



Der Stammbaum des menschlichen Wissens zu Beginn von Band 1 der *Encyclopédie ou Dictionnaire raisonné des sciences, des arts et des métiers* (28 Bände, vollendet im Jahr 1772)

Hierarchien in sozialen Systemen

Aus dem Blickwinkel der Soziologie, auch im Kontext von Sozialen Systemen, sind Hierarchien oft mit Verhältnissen von Herrschaft und Autorität verbunden - beispielsweise in der Linienorganisation eines Unternehmens, bei Behörden, im Militärwesen oder in der Kirche. Vor allem die katholische Kirche ist sehr stark hierarchisch strukturiert. Hierarchien werden auch allgemein zur Ordnung von Objekten zum Beispiel in einer Klassifikation verwendet. Bildlich werden Hierarchien häufig mit einer Pyramide verglichen. Die Elemente lassen sich meist in Ebenen anordnen, wobei jedes Element (bis auf das oberste) nur mit einem (Monohierarchie) oder mehreren (Polyhierarchie) Elementen der jeweils nächsthöheren Ebene verbunden ist.

Ein klassisches Beispiel für eine Hierarchie sind militärische Dienstgrade. Allerdings spiegeln in der Praxis die Dienstgrade nicht unbedingt das tatsächliche Vorgesetztenverhältnis wider. Auch in Unternehmen gibt es eine Hierarchie von Vorgesetzten und Abteilungsleitern, in der festgelegt ist, wer wem Weisungen erteilen kann. Auch dabei handelt es sich jedoch, meist aufgrund zahlreicher Gremien und Mitspracherechte, selten um eine rein hierarchische Entscheidungsstruktur.

Strenger geordnet ist die Gerichtsbarkeit mit ihren Instanzen, die jeweils Urteile der untergeordneten Instanz aufheben können. Zur Ordnung von Informationen sind Klassifikationen ein gängiges Mittel. Auch die hierarchische Ordnung von Dateien in einer Verzeichnisstruktur ist üblich.

Geplante soziale Netzwerke in der Betriebswirtschaftslehre mit wenig Herrschaftselementen nehmen nicht selten „Herrschaftsfreiheit“ in Anspruch, sind aber informell regelmäßig nicht hierarchiefrei. Weiterhin kann es, bei Matrixorganisation, auch zu Überschneidungen von Hierarchien kommen, die weiterer spezifischer Regelungen befürfen. Im besonderen beschäftigt sich die Organisationsforschung mit organisatorischen Phänomenen.

Hierarchische Strukturen kommen auch in der Tierwelt vor, die Rangordnung beschreibt die Stellung des Individuums vom Alphatier bis zu den Paria genannten Außenseitern.

Literatur

- Gerhard Schwarz, *Die 'Heilige Ordnung der Männer. Hierarchie, Gruppendynamik und die neue Rolle der Frauen*, 5., überarbeitete Aufl. (2007), VS Verlag, ISBN 3-531-15498-2
- Parya Memar, *Hierarchie in der Baukunst, architekturtheoretische Betrachtungen in Ost und West*, 1. Aufl. (2009), Philipp von Zabern Verlag, ISBN 978-3-8053-4061-8

Siehe auch

- Dyarchie, Herrschaft
- Baumstruktur, Gilles Deleuze, Baum des Wissens, Rhizom (Philosophie)
- Organisation (Wirtschaft), Linienorganisation, Peter-Prinzip
- Hierarchische Datenbank, Hierarchische Recherche
- Levellers
- Klassifikation

Organigramm

Das **Organigramm** (*Organisationsplan, Organisationsschaubild, Stellenplan*) ist eine grafische Darstellung der Aufbauorganisation. Organisatorische Einheiten sowie deren Aufgabenverteilung und Kommunikationsbeziehungen werden ersichtlich.

Allgemeines

Übliche Darstellungsformen in der Praxis sind das horizontale und das vertikale Organigramm sowie Mischformen aus beiden. Zur Visualisierung werden Symbole verwendet.

Auskünfte über folgende organisatorische Sachverhalte sind in einem Organigramm enthalten:

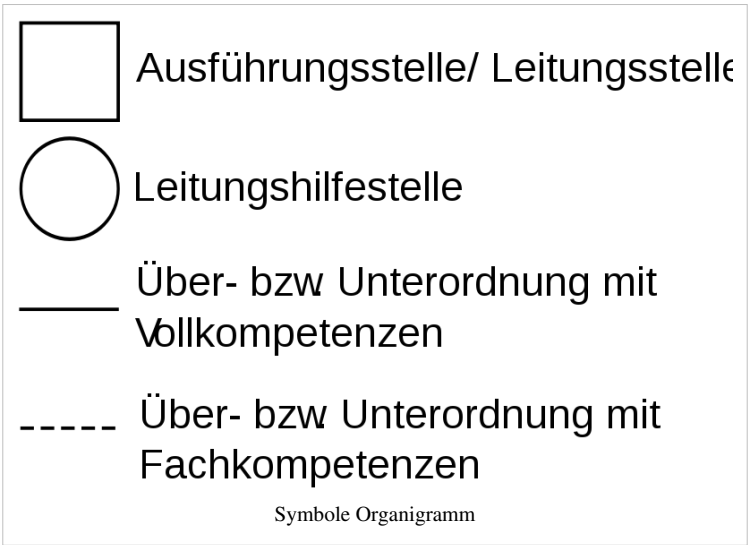
- Verteilung betrieblicher Aufgaben auf Stellen und Abteilungen
- hierarchische Struktur der Aufbau- bzw. Leitungsorganisation und der Weisungsbeziehungen
- Einordnung von Leitungshilfsstellen
- Personelle Besetzung (Stäbe, Stellen, Abteilungen)

Wichtige Spielregeln einer Organisation werden für alle sichtbar, seine Innenwirkung darf deshalb nicht unterschätzt werden, es ist die Landkarte jedes Unternehmens. Die formale Erstellung von Organigrammen ist meist firmenintern festgelegt.

Bei der Erstellung eines Organigramms ist der Detaillierungsgrad von Bedeutung. Es sollte geprüft werden, ob jeder Mitarbeiter des Unternehmens abgebildet werden soll oder ob einzelne Mitarbeitergruppen ausreichend sind. Handelt es sich etwa um eine Reorganisation, muss jede Stelle berücksichtigt werden. In einem modernen Organigramm sollte der Kunde explizit integriert werden. Mit Pfeilen kann man darüber hinausgehende Kommunikations- und Informationsströme abbilden.

Grafische Darstellung

Bezüglich der grafischen Darstellung gibt es keine allgemein gültige Regelung, allerdings hat sich in der Praxis Folgendes etabliert:



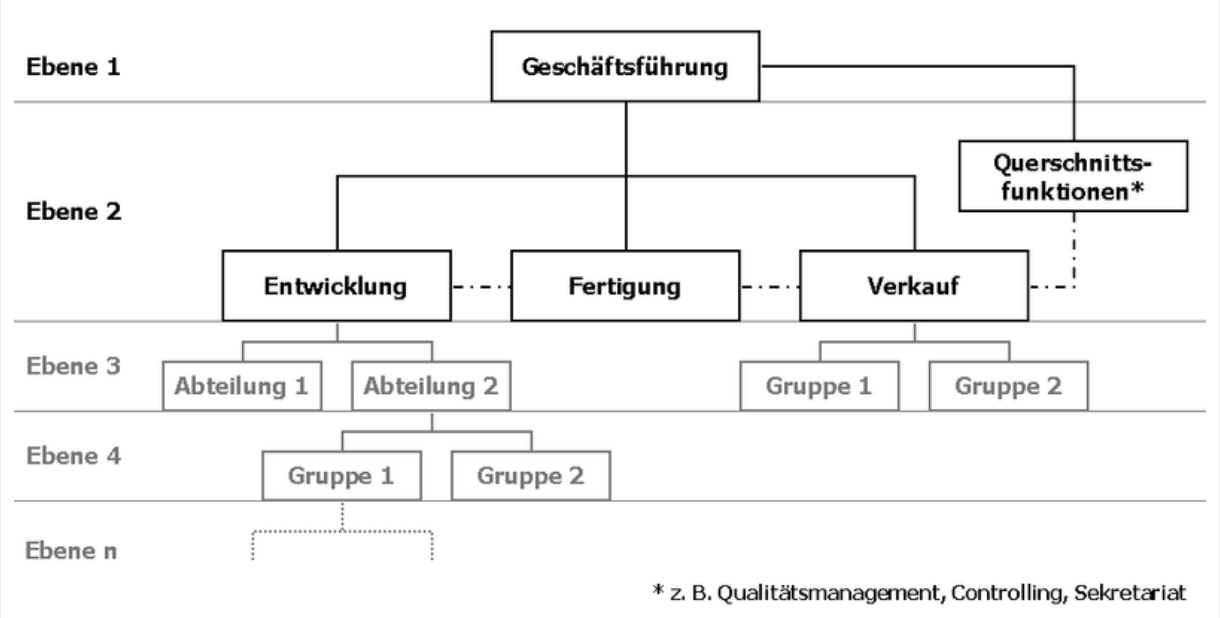
Symbole Organigramm

Des Weiteren gibt es folgende Möglichkeiten der Darstellung:

- In einem Viereck steht nur eine Person, die die jeweilige Stelle innehat und deren Sekretär/in.
- Vierecke, die eine Verbindung nach unten haben, beinhalten die Rolle des Vorgesetzten; z.B. ist Ebene 1 der Ebene 2 vorgesetzt, etc.

- Querschnittfunktionen sind als unterstützende oder fachlich bestimmende Stellen meist als Stabsstelle neben der Geschäftsführung in Form eines Kreises dargestellt. In der Darstellung ist die Unterstützung zusätzlich durch eine gestrichelte Linie angegeben.
- Die Organisation kann auch in Form von Kreisen und Ellipsen abgebildet werden. Die Kreise symbolisieren mehr Offenheit, Flexibilität, Kommunikation. Pfeile können zeigen, wie wichtige Prozesse ablaufen, in welche Richtungen Informationen weitergegeben werden etc.

Beispiel für ein Organigramm:



Beispiel: Organigramm

Vor- und Nachteile

Vorteile	Nachteile
• hierarchische Strukturen werden ersichtlich (Organisationsform) • Regelung der Zuständigkeiten • unterstützend bei kritischen Geschäftsentscheidungen • effizienter Informationsaustausch	• stark vereinfachend • Probleme der Darstellungstechnik • ohne geeignete Software sehr zeitintensiv und aufwändig • keine DIN-Normen

Software

Es gibt eine Reihe von Software-Produkten, mit denen man unter anderem auch Organigramme zeichnen kann. Dazu gehören Dia, KDissert und Kivio für Linux, Microsoft Visio, SmartDraw und OpenOffice.org Draw für Windows und OmniGraffle für Mac OS X.

Darüber hinaus gibt es dezidierte Organigramm-Software wie z. B. OrgPlus, welche Organigramme basierend auf vorhandenen Daten (wie z. B. aus SAP, PeopleSoft oder Oracle ERP) erstellen können.

Siehe auch

- Plan
- Visualisierung
- Navigationsplan
- Dotted-Line-Prinzip

Literatur

- Manfred Schulte-Zurhausen: *Organisation.* Vahlen: München 2002, ISBN 3-8006-2825-2.
- Jean-Paul Thommen: *Betriebswirtschaftslehre*, Band 3, ISBN 3-908143-03-9.
- Jean-Paul Thommen: *Allgemeine Betriebswirtschaftslehre*, 4. Auflage, Gabler-Verlag, Wiesbaden 2003.

Weblinks

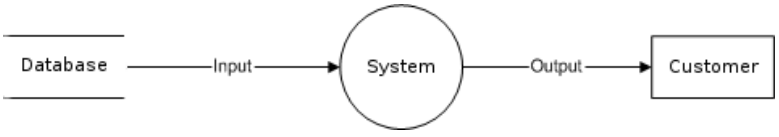
- Beispiel eines Organigramms, UNO [1]
- Cogmap [2] ist eine Webseite für öffentliche und private, online erstellbare Organigramme.

Referenzen

[1] http://www.unric.org/html/german/organe.htm
[2] http://www.cogmap.com/

Datenflussdiagramm

Ein **Datenflussdiagramm** oder **Datenflussplan** (engl. *data flow diagram*) stellt die Art der Verwendung, die Bereitstellung und Veränderung von Daten innerhalb eines Programms dar. Es kann auch dazu verwendet werden, den Datenfluss eines Prozesses oder einer Tätigkeit wiederzugeben (z. B. die Datenverwendung und Veränderung bei der Angebotserstellung in einem Handelsunternehmen). Ein Datenflussdiagramm hat keinen Kontrollfluss, es gibt keine Entscheidungsregeln und keine Schleifen. Die konkreten Operationen auf den Daten können durch einen Programmablaufplan dargestellt werden.



Datenflussdiagramm mit Datenspeicher, Datenflüssen, Funktion und Schnittstelle

Beim Datenflussdiagramm werden vier Elementtypen mit folgender Semantik unterschieden:

- *Datenspeicher*: dargestellt durch zwei parallele Linien, zwischen denen der Speichername steht (in der UML als Pufferknoten modellierbar).
- *Datenfluss*: dargestellt durch einen Pfeil mit Namen. Greift eine Funktion lesend und schreibend auf einen Datenspeicher zu kann man dies entweder mit zwei getrennten Pfeilen oder mit einem Doppelpfeil darstellen.
- *Funktion (oder Prozess)*: dargestellt durch einen Kreis mit Namen (vergleichbar der Aktivität in der UML).
- *Schnittstelle zur Umwelt*: dargestellt durch ein Rechteck, das den Schnittstellennamen enthält (externer Partner). Schnittstellen, an denen Daten in das System einfließen, werden Datenquellen genannt. Schnittstellen, an denen Daten aus dem System verschwinden heißen Datensenken.

Es gibt verschiedene Notationen zur Darstellung von Datenflussdiagrammen. Die oben vorgestellte Notation wurde 1979 von Tom DeMarco im Rahmen der Strukturierten Analyse beschrieben. Früher wurden die Symbole aus DIN 66001 verwendet. Diese Notation ist heute aber eher unüblich.

Bei jedem Datenfluss muss mindestens einer der Endpunkte (Quelle und/oder Ziel) ein Prozess sein. Die verfeinerte Darstellung eines Prozesses kann in einem weiteren Datenflussdiagramm erfolgen, das diesen Prozess in Subprozesse unterteilt.

Das Datenflussdiagramm ist das wesentliche Modellierungsinstrument der Strukturierten Analyse.

Bei Verwendung der UML übernimmt gewöhnlich das Aktivitätsdiagramm die Rolle des Datenflussdiagramms.

Eine Sonderform des Datenflussplans ist der stellenorientierte Datenflussplan, auch wer/was-Diagramm genannt. Dabei werden die Tätigkeiten den einzelnen Teilnehmern in vertikalen Swimlanes pro Teilnehmer zugeordnet.

Weblinks

- Datenflussdiagramm [1]

Referenzen

[1] http://www.teialehrbuch.de/Kostenlose-Kurse/SQL/14650-Datenflussdiagramm.html

Entscheidungstabelle

Entscheidungstabellen sind eine vieler Möglichkeiten, komplexe Regelwerke in übersichtlicher Weise darzustellen. Unter einer Regel ist dabei eine Vorschrift zu verstehen, welche **Aktionen** bei Vorliegen einer gegebenen Kombination von **Bedingungen** durchzuführen sind. Ein Regelwerk ist eine Zusammenstellung unterschiedlicher Regeln.

Einsatz

Entscheidungstabellen werden unter anderem beim Entwurf von Projekten, Computerprogrammen und speicherprogrammierbaren Steuerungen (SPS) eingesetzt, um komplexe Abhängigkeiten zwischen mehreren Bedingungen und dem jeweils auszuführenden Code oder den Aktionen übersichtlich und vollständig darzustellen. Ergänzend wird das Regelwerk oft durch einen übersichtlicheren Entscheidungsbaum graphisch oder in Matrizen dargestellt. Die Tabelle(n) sind jedoch systematischer und können deshalb leichter als der Baum auf Widerspruchsfreiheit und Vollständigkeit überprüft werden. Dabei darf der Arbeitsaufwand bei der Formulierung, bei der Abstimmung mit den Beteiligten und der Kontrolle nicht unterschätzt werden. Dafür ergibt sich eine enorme Erleichterung bei der Projektabwicklung, dem Änderungsdienst und der Kostenkalkulation.

Entscheidungstabellen werden auch in einem Business-Rule-Management-System (BRMS) verwendet, um sowohl die Definition und die automatische Ausführung von Regelwerken (Business-Rules) zu ermöglichen. Die Entscheidungstabellen werden dort in einem Business-Rule-Repository verwaltet.

Struktur

Eine Entscheidungstabelle besteht aus vier Teilbereichen:

- Einer Auflistung der zu berücksichtigenden Bedingungen
- Einer Auflistung der möglichen Aktionen
- Einem Bereich, in dem die möglichen Bedingungskombinationen zusammengestellt sind
- Einem Bereich, in dem jeder Bedingungskombination die jeweils durchzuführenden Aktivitäten zugeordnet sind.

Tabellenbezeichnung	R1	R2	R3	R4	R5	R6	R7	R8
Bedingungen								
Lieferfähig	j	j	j	j	n	n	n	n
Angaben vollständig	j	j	n	n	j	j	n	n
Bonität in Ordnung	j	n	j	n	j	n	j	n
Aktionen								
Lieferung mit Rechnung	x		x					
Lieferung als Nachnahme		x		x				
Angaben vervollständigen			x	x				
Mitteilen: nicht lieferbar					x	x	x	x

Die Spalten R1 bis R8 bezeichnen die jeweiligen Regeln. Am Beispiel der Regel 7 sei erläutert, wie die Regeln zu lesen sind: Wenn die Bedingung 3 erfüllt ist, die Bedingungen 1 und 2 hingegen nicht, dann ist die Aktion 4 auszuführen.

Eigenschaften

Im Folgenden werden vier Kriterien zur Beurteilung von Entscheidungstabellen dargestellt.

Vollständigkeit

Eine Entscheidungstabelle ist vollständig, wenn sämtliche möglichen Bedingungskombinationen erfasst sind. Bei *n* Bedingungen sind dies 2^n Kombinationen. Man erkennt, dass die Zahl der möglichen Kombinationen mit der Anzahl der Bedingungen exponentiell wächst – bei zehn Bedingungen gibt es bereits $2^{10} = 1024$ Kombinationen. In den meisten Fällen führt dies jedoch nicht zu einer gleich hohen Anzahl an Regeln, da Regeln oft redundant sind. Wenn trotz aller Sorgfalt oder durch geänderte Umstände eine Entscheidungstabelle nicht (mehr) alle Fälle abdeckt, kommt man am Ende ohne Ergebnis und Aktion aus der letzten Bedingungsspalte. Daher ist es sinnvoll, vor allem für Computerprogramme, nach der letzten Bedingungsspalte eine zusätzliche Spalte (z. B. "RF") mit allen Bedingungen "n" einzuführen. Läuft das Programm in diese Spalte, kann als Aktion ein Hinweis ausgegeben werden, z. B. "Fehler - Entscheidungstabelle unvollständig".

Konsolidierung

Eine Entscheidungstabelle ist konsolidierbar, wenn mehrere Regeln zu einer zusammengefasst werden können. In dem oben dargestellten Grundmuster können beispielsweise die Regeln R5, R6, R7 und R8 zusammengefasst werden: Da die durchzuführenden Aktionen nur von der Bedingung 1 abhängt und die Bedingungen 2 und 3 irrelevant sind, lassen sie sich durch eine einzige Regel darstellen:

Tabellenbezeichnung	**R5 (neu)**
Bedingungen	
Bedingung 1	n
Bedingung 2	-
Bedingung 3	-
Aktionen	
Aktion 1	
Aktion 2	
Aktion 3	
Aktion 4	x

Die waagerechten Striche werden als „Irrelevanzanzeiger“ oder „Don't care-Zeichen“ bezeichnet. Sie zeigen an, dass die Bedingungen 2 und 3 für die Festlegung der durchzuführenden Aktion irrelevant sind, sofern die Bedingung 1 erfüllt ist.

Allgemein gibt es für die Konsolidierung wiederum zwei Regeln: Führen 1. zwei Regeln zur selben Aktion oder Aktionsfolge UND unterscheiden sich diese 2. nur in einer Bedingungsanzeige, so können diese beiden Regeln konsolidiert werden. Anstelle der unterschiedlichen Bedingungsanzeiger tritt in der konsolidierten Regel der Irrelevanzzeiger.

Redundanz

Eine Entscheidungstabelle ist redundant, wenn mehrere Regeln identische Fälle enthalten. Ein Fall ist eine Bedingungskombination (z. B. hier: "j,n,n"), die kein "don't care" (Irrelevanzanzeiger) aufweist. In der folgenden Tabelle enthält die Regel R4 ("j,j,n" und "j,n,n") auch den Fall der Regel R3. Dabei führen beide Regeln zu denselben Aktionen, deswegen liegt Redundanz vor, andernfalls Widerspruch. Also kann die Regel R3 entfallen, ohne dass ein Informationsverlust eintritt.

Tabellenbezeichnung	**R3**	**R4**
Bedingungen		
Bedingung 1	j	j
Bedingung 2	n	-
Bedingung 3	n	n
Aktionen		
Aktion 1		
Aktion 2		
Aktion 3	x	x
Aktion 4	x	x
Aktion 5		

Widerspruchsfreiheit (Konsistenz)

Sobald eine Entscheidungstabelle Irrelevanzzeiger enthält, besteht die Möglichkeit, dass die Tabelle inkonsistent wird. Auch hierzu ein Beispiel:

Tabellenbezeichnung	**R3**	**R4**
Bedingungen		
Bedingung 1	j	j
Bedingung 2	n	n
Bedingung 3	-	n
Aktionen		
Aktion 1		
Aktion 2		
Aktion 3	x	x
Aktion 4	x	
Aktion 5		

Die Regel R3 besagt, dass die Aktionen 3 und 4 ausgeführt werden sollen, sobald die Bedingung 1 erfüllt und Bedingung 2 nicht erfüllt ist. Die Regel R4 besagt hingegen, dass nur die Aktion 3 ausgeführt werden soll, wenn zusätzlich Bedingung 3 nicht erfüllt ist. Beide Regeln widersprechen sich also, die Entscheidungstabelle ist daher inkonsistent.

Entscheidungstabellen zur Testdatenermittlung

Nachdem Entscheidungstabellen erstellt und validiert wurden, kann man sie zur Testdatengenerierung nutzen. Im Folgenden sind Strategien genannt, wie Testdaten generiert werden können und wie eine Testsuite sinnvoll eingeschränkt werden kann.

- All-Explicit Variants
- All-Variants, All-True, All-False, All-Primes
- Each-Condition/All-Conditions
- Binary Decision Diagram (BDD) Determinants
- Variable Negation (*wird nicht darauf eingegangen)
- Nonbinary Variable Domain Analysis

Bei allen Verfahren stellt sich immer die Problematik zwischen Testumfang und den Kosten für den jeweiligen Test. Die Schwierigkeit bei Entscheidungstabellen liegt darin, dass ihre Erstellung sehr aufwändig ist. Software enthält viele Bedingungen und in der Regel sind die Spezifikationen unvollständig, so dass eine Herleitung einer Entscheidungstabelle zu aufwändig ist. Hat man erstmal eine Entscheidungstabelle erstellt, lassen sich die genannten Teststrategien automatisiert anwenden.

Umfangreiche bzw. komplizierte Abläufe kann man leichter in Entscheidungstabellen erfassen, indem man mehrere zusammenhängende Entscheidungstabellen erstellt. Dazu werden Aktionen, die selbst in weitere Unteraktionen aufgegliedert werden können, als eigene Entscheidungstabellen dargestellt. Die Abzweigung aus der ersten Entscheidungstabelle erfolgt dann dort z. B. durch die Aktion "weiter in ET 2". Es ist dabei hilfreich, die erste Entscheidungstabelle so zu gestalten, dass sie in erster Linie den direkten, schnellsten oder am häufigsten beschrittenen Ablauf darstellt, die Abzweigungen die jeweils weniger häufig beschrittenen. Dadurch ergibt sich eine Art Hierarchie der Entscheidungstabellen. Das gesamte System aus Abläufen wird dadurch übersichtlich.

Literatur

- Irrgang, R., 1995, Entscheidungstabellentechnik, Entscheidungstabellen erstellen und analysieren, 185 S., Diskette (Ed.es, 19) Kt. Expertverlag ISBN 3-8169-1106-4
- Binder, R., 1999, Testing Object Oriented Systems, Models, Patterns and Tools, Addison Wesley
- Jüttner, G., 1989, Entscheidungstabellen und wissensbasierte Systeme, Oldenbourg ISBN 3-486-21454-3
- Strunz, H., 1977, Entscheidungstabellentechnik, Hanser ISBN 3-446-12382-2
- Moritz, B., Klausdorf/Schwentine 1977, Diagramm-Techniken für den EDV-Praktiker, Selbstverlag
- DIN 66241

Siehe auch

- Entscheidungsbaum

Weblinks

- http://www.helmut-dressler.de/HD_F-ETAB.html

Entscheidungsbaum

Entscheidungsbäume sind geordnete, gerichtete Bäume, die der Darstellung von Entscheidungsregeln dienen. Sie veranschaulichen hierarchisch, aufeinanderfolgende Entscheidungen. Sie haben eine Bedeutung in zahlreichen Bereichen, in denen automatisch klassifiziert wird oder aus Erfahrungswissen formale Regeln hergeleitet oder dargestellt werden sollen.

Eigenschaften und Einsatz

Entscheidungsbäume sind eine Methode, die zur automatischen Klassifikation von Datenobjekten und damit zur Lösung von Entscheidungsproblemen dient. Sie werden außerdem zur übersichtlichen Darstellung von formalen Regeln genutzt. Ein Entscheidungsbaum besteht immer aus einem Wurzelknoten und beliebig vielen inneren Knoten sowie mindestens zwei Blättern. Dabei repräsentiert jeder Knoten eine logische Regel und jedes Blatt eine Antwort auf das Entscheidungsproblem.

Die Komplexität und Semantik der Regeln sind von vornherein nicht beschränkt. Bei binären Entscheidungsbäumen kann jeder Regelausdruck nur einen von zwei Werten annehmen. Alle Entscheidungsbäume lassen sich in binäre Entscheidungsbäume überführen.

Entscheidungsbäume kommen in zahlreichen Bereichen zum Einsatz, z.B.

- in der Stochastik zur Veranschaulichung bedingter Wahrscheinlichkeiten (Beispiel bei Absolute Häufigkeit)
- im Maschinellen Lernen als Methode zur automatischen Klassifikation von Objekten
- im Data-Mining
- in der Entscheidungstheorie
- in der ärztlichen Entscheidungsfindung (Medizin) und in der Notfallmedizin
- in Business-Rule-Management-Systemen (BRMS) für die Definition von Regeln

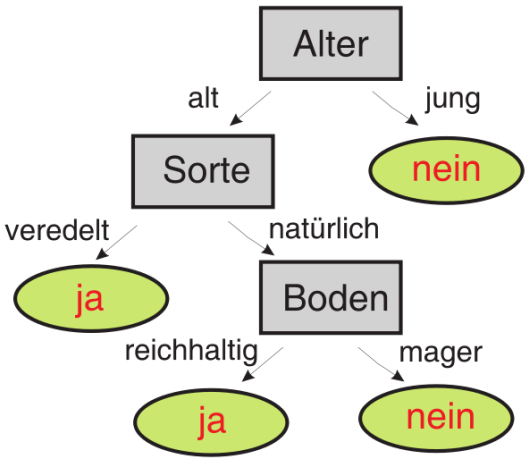
Funktionsweise

Klassifizieren mit Entscheidungsbäumen

Um eine Klassifikation eines einzelnen Datenobjektes abzulesen, geht man vom Wurzelknoten entlang des Baumes abwärts. Bei jedem Knoten wird ein Attribut abgefragt und eine Entscheidung über die Auswahl des folgenden Knoten getroffen. Diese Prozedur wird so lange fortgesetzt, bis man ein Blatt erreicht. Das Blatt entspricht der Klassifikation. Ein Baum enthält grundsätzlich Regeln zur Beantwortung von nur genau einer Fragestellung.

Beispiel: Einfacher Entscheidungsbaum

Der nebenstehende binäre Entscheidungsbaum gibt eine Antwort auf die Frage, ob ein Apfelbaum Früchte tragen wird. Als Eingabe benötigt der Baum einen Vektor mit Angaben zu den Attributen eines Apfelbaumes. Ein Apfelbaum kann beispielsweise die Attribute *alt*, *natürliche Sorte* und *reichhaltiger Boden* besitzen. Beginnend mit dem Wurzelknoten werden nun die Entscheidungsregeln des Baumes auf den Eingabevektor angewendet. Dabei wird im Beispielbaum an jedem Knoten ein Attribut des Eingabevektors abgefragt, am Wurzelknoten etwa das Alter des Apfelbaumes. Die Antwort entscheidet über den Folgeknoten und damit über die nächste anzuwendende Entscheidungsregel, in diesem Falle die Frage zur Sorte, und danach die Frage nach der Bodenbeschaffenheit. Gelangt man nach einer Folge von ausgewerteten Regeln an ein Blatt, hat man die Antwort auf die ursprüngliche Frage. Nicht immer müssen alle Ebenen des Entscheidungsbaums durchlaufen werden. Für den oben beschriebenen Apfelbaum ist die Antwort **ja**, also dass der Baum Früchte tragen wird. Diesen Entscheidungsvorgang nennt man formal Klassifikation.



Einfacher Entscheidungsbaum zur Vorhersage, ob ein Apfelbaum Früchte tragen wird

Induktion von Entscheidungsbäumen

Entscheidungsbäume können entweder von Experten manuell aufgeschrieben werden oder mit Techniken des maschinellen Lernens automatisch aus gesammelten Erfahrungswerten induziert werden. Hierzu gibt es mehrere konkurrierende Algorithmen.

Die Induktion der Entscheidungsbäume erfolgt üblicherweise rekursiv im Top-down-Prinzip. Dazu ist es von entscheidender Bedeutung, dass ein geeigneter Datensatz mit verlässlichen Erfahrungswerten zum Entscheidungsproblem (der sogenannte *Trainingsdatensatz*) vorliegt. Das bedeutet, dass zu jedem Objekt des Trainingsdatensatzes die Klassifikation des Zielattributs bekannt sein muss. Bei jedem Induktionsschritt wird nun das Attribut gesucht, mit welchem sich die Trainingsdaten in diesem Schritt bezüglich des Zielattributs am *besten* klassifizieren lassen. Als Maß für die Bestimmung der *besten* Klassifizierung kommen Entropie-Maße, Gini-Index oder andere zur Anwendung. Das ermittelte Attribut wird nun zur Aufteilung der Daten verwendet. Auf die so entstandenen Teilmengen wird die Prozedur rekursiv angewendet, bis in jeder Teilmenge nur noch Objekte mit einer Klassifikation enthalten sind. Am Ende ist ein Entscheidungsbaum entstanden, der das Erfahrungswissen des Trainingsdatensatzes in formalen Regeln beschreibt.

Die Bäume können nun zum automatischen Klassifizieren anderer Datensätze oder zum Interpretieren und Auswerten des entstandenen Regelwerks genutzt werden.

Algorithmen im Vergleich

Die Algorithmen zur automatischen Induktion von Entscheidungsbäumen setzen alle auf dem gleichen rekursiven Top-down-Prinzip auf. Sie unterscheiden sich nur in ihren Kriterien zur Auswahl der Attribute und Werte der Regeln an den Knoten des Baumes, in ihren Kriterien zum Abbruch des Induktionsprozesses und in möglichen *Nachbearbeitungsschritten*, die einen bereits berechneten Ast eines Baumes (oder ganze Bäume) nachträglich anhand diverser Kriterien optimieren.

Die Praxis unterscheidet verschiedene Familien von Algorithmen:

- Die älteste Familie sind die CHAIDs (*Chi-square Automatic Interaction Detectors*) von 1964[1] . Sie können Bäume auf Basis von diskreten Attributen erzeugen.
- Die Familie der CARTs (*Classification And Regression Trees*) erweitert die *CHAIDS* vor allem um die Möglichkeit, reellwertige Attribute verarbeiten zu können, indem ein *Teilungs-Kriterium* (engl. *split-criterion*) zur Aufteilung reellwertiger Attribute in diskrete Kategorien genutzt wird. Zudem werden einige Optimierungsstrategien wie z.B. das *Pruning* eingeführt.
- Die jüngsten Algorithmen sind der ID3-Algorithmus und seine Nachfolger C4.5 und *C5.0*[2] [3] . Sie zählen formal zur Familie der *CART*s, werden aber häufig als eigenständige Gruppe gesehen, da sie im Vergleich zu *CART* zahlreiche neue Optimierungsstrategien einführen.

Fehlerrate und Wirksamkeit

Die Fehlerrate eines Entscheidungsbaumes ist die Anzahl der inkorrekt klassifizierten Datenobjekte im Verhältnis zu allen Datenonbjekten eines Datensatzes. Diese Zahl wird regelmäßig entweder auf den benutzten Trainingsdaten oder besser auf einer zu den Trainingsdaten disjunkten Menge an möglichst korrekt klassifizierten Datenobjekten ermittelt.

Je nach Anwendungsbereich kann es von besonderer Bedeutung sein, entweder die Anzahl der falsch positiv oder falsch negativ klassifizierten Objekte im speziellen niedrig zu halten. So ist es etwa in der Notfallmedizin wesentlich unschädlicher einen gesunden Patienten zu behandeln, als einen kranken Patienten nicht zu behandeln. Die Wirksamkeit von Entscheidungsbäumen ist daher immer auch eine kontextabhängige Größe.

Vor- und Nachteile

Interpretierbarkeit und Verständlichkeit

Ein großer Vorteil von Entscheidungsbäumen ist, dass sie gut erklärbar und nachvollziehbar sind. Dies erlaubt dem Benutzer, das Ergebnis auszuwerten und Schlüsselattribute zu erkennen. Das ist vor allem nützlich, wenn grundlegende Eigenschaften der Daten von vornherein nicht bekannt sind. Damit ist die Induktion von Entscheidungsbäumen auch eine wichtige Technik des Data Minings.

Klassifikationsgüte in reellwertigen Datenräumen

Ein oft benannter Nachteil der Entscheidungsbäume ist die relativ geringe Klassifikationsgüte in reellwertigen Datenräumen, wenn man die Bäume zur automatischen Klassifikation einsetzt. So schneiden die Bäume aufgrund ihres diskreten Regelwerks bei den meisten Klassifikationsproblemen aus der realen Welt im Vergleich zu anderen Klassifikationstechniken wie beispielsweise Neuronalen Netzen oder Support-Vektor-Maschinen etwas schlechter ab[4] [5] . Das bedeutet, dass durch die Bäume zwar für Menschen leicht nachvollziehbare Regeln erzeugt werden können, diese verständlichen Regeln aber für Klassifikationsprobleme der realen Welt statistisch betrachtet oft nicht

die bestmögliche Qualität besitzen[6] [7] .

Größe der Entscheidungsbäume und Überanpassung

Ein weiterer Nachteil ist die mögliche Größe der Entscheidungsbäume, wenn sich aus den Trainingsdaten keine einfachen Regeln induzieren lassen. Dies kann sich mehrfach negativ auswirken: Zum einen verliert ein menschlicher Betrachter schnell den Gesamtüberblick über den Zusammenhang der vielen Regeln, zum anderen führen solche großen Bäume aber auch meistens zur Überanpassung an den Trainingsdatensatz, so dass neue Datensätze nur fehlerhaft automatisch klassifiziert werden. Es wurden deshalb so genannte Pruning-Methoden entwickelt, welche die Entscheidungsbäume auf eine vernünftige Größe kürzen. Beispielsweise kann man die maximale Tiefe der Bäume beschränken oder eine Mindestanzahl der Objekte pro Knoten festlegen.

Weiterverarbeitung der Ergebnisse

Oft bedient man sich der Entscheidungsbäume nur als Zwischenschritt zu einer effizienteren Darstellung des Regelwerkes. Um zu den Regeln zu gelangen, werden durch verschiedene Verfahren unterschiedliche Entscheidungsbäume generiert. Dabei werden häufig auftretende Regeln extrahiert. Die Optimierungen werden überlagert, um einen robusten, allgemeinen und korrekten Regelsatz zu erhalten. Dass die Regeln in keinerlei Beziehungen zueinander stehen und dass widersprüchliche Regeln erzeugt werden können, wirkt sich nachteilig auf diese Methode aus.

Weiterentwicklungen und Erweiterungen

Entscheidungswälder

Eine Methode, um die Klassifikationsgüte beim Einsatz von Entscheidungsbäumen zu steigern, ist der Einsatz von Mengen von Entscheidungsbäumen anstelle von einzelnen Bäumen[8] . Solche Mengen von Entscheidungsbäumen werden als *Entscheidungswälder* (englisch: *decision forests*) bezeichnet[9] .

Der Gebrauch von Entscheidungswäldern zählt im maschinellen Lernen zu den sogenannten *Ensemble-Techniken*. Die Idee der Entscheidungswälder ist, dass ein einzelner Entscheidungsbaum zwar keine optimale Klassifikation liefern muss, die Mehrheitsentscheidung einer Menge von geeigneten Bäumen dies aber sehr wohl leisten kann[10] . Verbreitete Methoden zur Erzeugung geeigneter Wälder sind Boosting[11] , *Bagging*[12] und *Arcing*.

Nachteil der Entscheidungswälder ist, dass es für Menschen nicht mehr so einfach ist, die in allen Bäumen enthaltenen Regeln in ihrer Gesamtheit in einfacher Weise zu interpretieren, so wie es bei einzelnen Bäumen möglich wäre[13] .

Kombination mit Neuronalen Netzen

Entscheidungsbäume können mit neuronalen Netzen kombimiert werden.

So ist es möglich, ineffiziente Äste des Baumes durch neuronale Netze zu ersetzen, um eine höhere, allein mit den Bäumen nicht erreichbare Klassifikationsgüte zu erreichen[14] . Auch können die Vorteile beider Klassifikationsmethoden durch die Abbildung von Teilstrukturen in die jeweils andere Methodik genutzt werden: Die Bäume brauchen nicht so viele Trainingsdaten zur Induktion wie die neuronalen Netze. Dafür können sie ziemlich ungenau sein, besonders wenn sie klein sind. Die Neuronalen Netze hingegen klassifizieren genauer, benötigen aber mehr Traingsdaten. Deshalb versucht man, die Eigenschaften der Entscheidungsbäume zur Generierung von Teilen neuronaler Netze zu nutzen, indem so genannte TBNN (Tree-Based Neural Network) die Regeln der Entscheidungsbäume in die neuronalen Netze übersetzen.

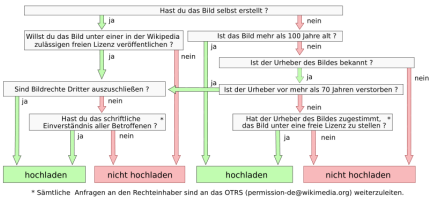
Praktische Anwendungen

Anwendungsprogramme

Es gibt etliche Anwendungsprogramme, die Entscheidungsbäume implementiert haben, so werden sie zum Beispiel in den Statistiksoftwarepaketen GNU R, SPSS und SAS angeboten. Die letzteren beiden Pakete verwenden übrigens – wie die meisten anderen Data Mining-Software-Pakete auch – den CHAID-Algorithmus.

Kundenklassifikation

Eine Bank möchte mit einer Direct-Mailing-Aktion einen neuen Service verkaufen. Um den Gewinn zu maximieren, sollen mit der Aktion diejenigen Haushalte angesprochen werden, welche der Kombination von demografischen Variablen entsprechen, die der entsprechende Entscheidungsbaum als optimal erklärt hat. Dieser Prozess wird Data Segmentation oder auch *Segmentation Modeling* genannt.

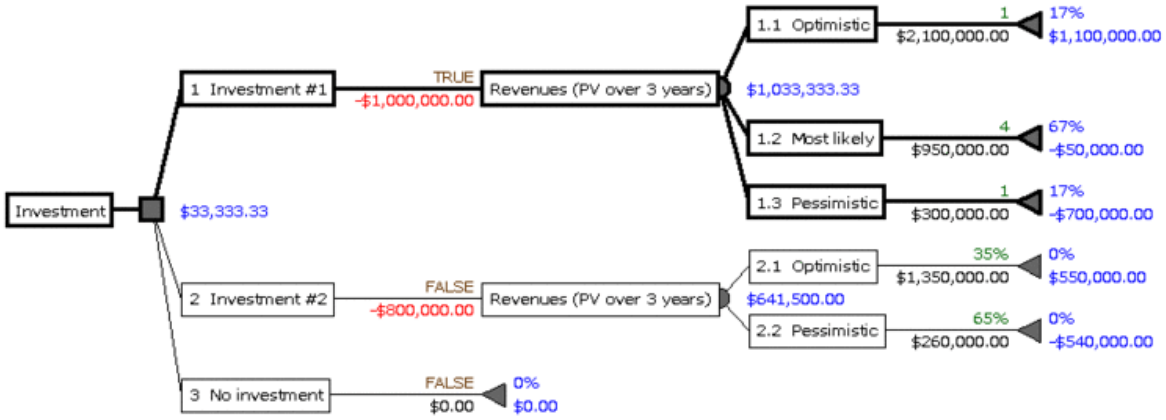


Beispiel:Entscheidungsbaum für das Hochladen von Bildern in Wikipedia

Der Entscheidungsbaum liefert also gute Tipps, wer positiv auf den Versand reagieren könnte. Dies erlaubt der Bank, nur diejenigen Haushalte anzuschreiben, welche der Zielgruppe entsprechen.

Entscheidungsbäume in der BWL

Im Gebiet der Entscheidungslehre innerhalb der BWL werden Entscheidungsbäume von links nach rechts gezeichnet (und umgekehrt berechnet), um Payoffs aus Ausgaben und Einnahmen zu errechnen und die Entscheidung für maximalen Payoff zu optimieren.



Bei diesen Entscheidungsbäumen stehen Quadrate für Entscheidungen, Kreise für Möglichkeiten, und Dreiecke terminieren die Zweige. Entscheidungsbäume dieses Typs können mit anderen Verfahren kombiniert werden, im Beispiel oben werden NPV (Net Present Value, Netto-Barwert), lineare Verteilung von Annahmen (Investment #2) und PERT 3-Punkt-Schätzung (Investment #1) verwendet.

Literatur

- Leo Breiman, Jerome H. Friedman, Richard A. Olshen, Charles J. Stone: *Classification and regression trees*, Wadsworth International Group, Belmont CA, 1984, ISBN 0-412-04841-8
- Tom M. Mitchell: *Machine Learning*, McGraw-Hill, Boston USA, 1997, ISBN 0-07-115467-1
- Jiawei Han, Micheline Kamber: *Data Mining*, Morgan Kaufmann publishers, San Francisco CA, 2006 (2nd edition), ISBN 978-1558609013
- Peter Dörsam: *Entscheidungstheorie anschaulich dargestellt*, Pd-Verlag, Heidenau, 2007 (5. Auflage), ISBN 978-3867073059
- Günter Bamber, Adolf G. Coenenberg, Michael Krapp: *Betriebswirtschaftliche Entscheidungslehre*, Verlag Franz Vahlen, München, 2008 (14. Auflage), ISBN 978-3-8006-3506-1

Einzelnachweise

[1] J.A. Sonquist, J.N. Morgan: *The Detection of Interaction Effects*, Survey Research Center, Institute for Social Research, University of Michigan, Ann Arbor.

[2] J.R. Quinlan: *Induction of decision trees*, Machine learning, 1(1):81-106, 1986

[3] http://www.rulequest.com/

[4] R.King, C.Feng, A.Sutherland: *Comparison of classification algorithms on large real-world problems*, Applied Artificial Intelligence, 9(3):259-287, Mai/Juni 1995

[5] O.Gascuel, B.Bouchon-Meunier, G.Caraux, P.Gallinari, A. Guénoche, Y.Guermeur: *Twelve numerical, symbolic and hybrid supervised classification methods*, International Journal of Pattern Recognition and Artificial Intelligence, 12(5):517-571, 1998

[6] A.Flöter: *Analyzing biological expression data based on decision tree induction*, Dissertation, Universität Potsdam, 2006

[7] M.Murata, Q.Ma, H.Isahara: *Comparison of three machine-learning methods for thai part-of-speech tagging*, ACM Transaction on Asian Language Information Processing, 1(2):145-158, Juni 2002

[8] Y.Freund: *Boosting a weak learning algorithm by majority*, Information and Computation, 121(2):256-285, 1995

[9] W.Tong, Q.Xie, H.Hong, H.Fang, L.Shi, R.Perkins, E.F.Petricoin: *Using decision forests to classify prostate cancer samples on the basis of seldi-tof ms data: Assessing chance correlation and prediction confidence*, Environmental Health Perspectives, 112(16):1622-1627, 2004

[10] E.Bauer, R.Kohavi: *An empirical comparison of voting classification algorithms: Bagging, Boosting, and variants*, Machine Learning, 36(1-2):105-139, 1998

[11] R.E.Schapire: *A short introduction to boosting*, Japanese Society for Artificial Intelligence, 14(5):771-780, 1999

[12] L.Breiman: *Bagging predictors*, Machine Learning, 24(2):123-140, 1996

[13] R.Nock: *Inducing interpretable voting classifiers without trading accuracy for simplicity: Theoretical results, approximation algorithms, and experiments*, Journal of Artificial Intelligence Research, 17:137-170, August 2002

[14] A.K. Seewald, J. Petrak, G. Widmer: *Hybrid decision tree learners with alternative leaf classifiers*, Proceedings of the 14th International FLAIRS conference, AAAI Press, Menlo Park CA, 2000

Siehe auch

- Heuristik
- Künstliche Intelligenz
- Maschinelles Lernen
- Künstliches neuronales Netz
- Realoptionen
- Top-Down Induction of Decision Trees
- Klassifikationsbaum-Methode
- Entscheidungstabelle

Data Dictionary

Ein **Data Dictionary** – in deutscher Übersetzung auch **Datenwörterbuch**, **Datenkatalog** oder etwas unscharf *Datenverzeichnis* genannt – ist ein Katalog von Metadaten, der die Definitionen und Darstellungsregeln von Datenelementen enthält. Es beschreibt alle Anwendungsdaten eines Unternehmens und die Beziehungen zwischen den verschiedenen Datenobjekten mit dem Ziel einer redundanzfreien und zentralen Definition der Datenstrukturen und entspricht somit einer Darstellung des anwendungsspezifischen Datenmodells.

Im Kontext einer relationalen Datenbank ist ein Data Dictionary eine Menge von Tabellen und Sichten, die von der Abfragesprache (etwa SQL) nur gelesen werden (read-only). Das Data Dictionary ist dabei wiederum als eine Datenbank aufgebaut, enthält aber nicht direkt die Anwendungsdaten, sondern Metadaten, das heißt Daten, welche die Struktur (nicht den Inhalt) der Anwendungsdaten beschreiben. Aufbau und Pflege des Data Dictionary erfolgen typischerweise über einen interaktiven Pflegedialog oder mit einer Datendefinitionssprache (DDL).

Aktive, passive und integrierte Data Dictionaries

Ein **aktives Data Dictionary** reflektiert jederzeit den aktuellen detaillierten Stand des Datenmodells. Änderungen an der Struktur einer Datenbank können direkt in der Pflegeoberfläche des Data Dictionary erfolgen, oder mit anderen Mitteln, zum Beispiel einem Kommandointerpreter einer DDL. Unabhängig davon, wie diese Änderungen erfolgen, ist die Aktualität eines aktiven Data Dictionary immer automatisch gewährleistet.

In einem **passiven Data Dictionary** ist diese Synchronität nicht gegeben. Änderungen an der Struktur des DBMS müssen im Data Dictionary (DD) manuell nachgepflegt werden, falls das gewünscht und wirtschaftlich möglich ist. Insbesondere DD-Produkte zur Modellierung und Dokumentation des konzeptionellen Datenmodells leiden unter dieser Problematik.

Schnittstellen vom Data Dictionary zu Mensch und Software

Ein Data Dictionary kann über folgende Schnittstellen konsultiert werden:[1]

- Benutzerschnittstelle
  - Datendesigner, -modellierer
  - Anwendungsprogrammierer
  - Endbenutzer
  - Datenbankadministrator
- Software- und DBMS-Schnittstelle
  - Compiler/Precompiler
  - Integrierte Entwicklungsumgebung (IDE) bei Integration des Data Dictionary im Sinne von CASE
  - Anwendungsprogramme
  - Berichts-/Formulargeneratoren
  - Query Optimizer
  - Integritity Constraint Enforcer

Unterscheidung von Data Dictionaries nach Modellierungsebene

In der Entwicklung und Pflege von Datenmodellen werden unterschiedliche Modellierungsebenen unterschieden:

- Konzeptionelle Ebene (in der Regel bezogen auf ein Anwendungsgebiet, in der Wirtschaftsinformatik oft auch unternehmensweit oder sogar unternehmensübergreifend)
- Logische Ebene
- Physische Ebene, in der das konzeptionelle/logische Datenmodell bezogen auf ein bestimmtes DBMS abgebildet und umgesetzt wird.

Entsprechend den unterschiedlichen Ebenen der Datenmodellierung können die Data Dictionaries nach Unterstützung dieser Modellebenen unterschieden werden. Je nach Ebene unterscheiden sich dabei die Data Dictionaries nach Art, Inhalt und auch Datentypen der notwendigen Metadaten, aber auch bezüglich ihrer Funktionen und Auswertungsmöglichkeiten.

Data Dictionary zur konzeptionellen/logischen Datenmodellierung

Zu einem **Data Dictionary zur konzeptionellen/logischen Datenmodellierung** gehören:

- Definition der Entitäten, Datenelemente und der Beziehungen zwischen den Entitäten
- Betriebswirtschaftliche Definitionen und Erläuterungen derselben

Neben der Festlegung der wesentlichen Datenobjekte bzw. -elemente und ihren Beziehungen werden typischerweise auch ausführliche beschreibende Texte auf Ebene der jeweiligen Entitäten hinterlegt, welche untereinander mittels Hyperlink-Technik verknüpft sind. Wenn eine Organisation ein unternehmensweites Datenmodell (UwDM) aufbaut, werden zu jeder Entität und jedem Datenelement Informationen zur anwendungsbezogenen Semantik, zum Datentyp und zur -darstellung zusammengetragen. Die semantischen Komponenten fokussieren auf die Abbildung der genauen Bedeutung eines Datenelements und sind als Fließtext formuliert. Die Darstellungsregeln definieren, wie Datenelemente gespeichert werden (z. B. Datentyp wie Integer, Text, maximale Textlänge, Eingabeformat, Ausgabeformate, zulässige Wertebereiche als Prüfregel, statische oder dynamische Menge, usw.). Diese erste Form ist häufig im Funktionsumfang eines DBMS nicht als Standardfunktion enthalten. Deshalb müssen hier oft abgesetzte Insellösungen verwendet werden. Diese stellen aber in Bezug auf das DBMS ein **passives Data Dictionary** dar. Änderungen am konzeptionellen Datenmodell können nicht automatisch in das physische Datenmodell des DBMS übernommen werden.

Unter *ISO/IEC 10027* (siehe unten) sind Spezifikationen erarbeitet worden, die einen hersteller- und plattformübergreifenden Austausch von Informationsressourcen zwischen verschiedenen Data Dictionaries erlauben sollen.

Ein Data Dictionary kann auch als Glossar genutzt werden, indem Informationsobjekte/Entitäten, Datenelemente/Attribute und auch Beziehungen/Relationships als Begriffe betrachtet werden, deren Definitionen im jeweiligen Beschreibungsteil abgelegt werden. Das Data Dictionary kann zu vollständigen Ontologien bzw. Klassen bzw. Geschäftsprozessmodellen weiterentwickelt werden. Wenn neben der Datenstruktur auch die Methoden zur Datentransformation beschrieben werden, spricht man von einem Repository.

Data Dictionary zur physischen Datenmodellierung

Zu einem **Data Dictionary zur physischen Datenmodellierung** gehören genaue Angaben zu:

- Tabellen und Datenfeldern
- Primär- und Fremdschlüsselbeziehungen
- Integritätsbedingungen, z. B. Prüfinformationen
- Stored Procedures und Triggers
- Zugriffslegitimationen (Benutzernamen, Rollen)
- physischen Datenbankstruktur, z. B. Speicherallokationen und Indizes
- Verweis- und Verwendungsnachweis

Diese Form ist in jedem DBMS als aktives Data Dictionary vorhanden, ist jedoch nicht in jedem Fall für den Anwendungsprogrammierer sichtbar. Wo ein solches Data Dictionary nicht sichtbar ist, bildet es dennoch die Datenbankstruktur als Datenbankschema, ist jedoch in verborgenen Systemtabellen abgelegt. Bei jedem Zugriff auf die Datenbank liest die DBMS-Systemsoftware das Datenbankschema, um die Struktur und den Speicherort der abgefragten Daten erkennen zu können.

Funktion des Data Dictionary in der Anwendungsentwicklung

Sinnvoll ist in jedem Fall die Integration der Metadaten aus dem Data Dictionary in die Integrierte Entwicklungsumgebung (IDE). Für die dynamische bzw. generische Programmierung von Formularen und Berichten ist jedoch darüber hinaus ein für die Bedürfnisse der Anwendungsprogrammierung sinnvoll strukturiertes und sichtbares Data Dictionary eine notwendige Voraussetzung.

Die Funktionen für die konzeptionelle und die physische Datenmodellmodellierung sind häufig nicht in einem Data Dictionary integriert. Gravierender noch, Änderungen an der detaillierten Datenbankarchitektur werden nicht ins konzeptionelle Datenmodell zurückgespiegelt. Entweder ist eine aufwendige manuelle Nachpflege notwendig, oder Aktualität des dokumentierten Datenmodells geht verloren.

Ein Data Dictionary, das sowohl in die Datenbank, die Programmentwicklungsumgebung und die Datenmodellierungsinstrumente integriert ist, erfüllt vielfältige Funktionen:

- Es beschreibt alle persistenten Daten eines Anwendungsgebiets, z. B. in Form eines unternehmensweiten Datenmodells
- Aufgrund der Data Dictionary -Daten können Bildschirmmasken automatisch generiert werden (siehe generative Programmierung).
- Die Struktur einer Datenbanktabelle kann von einem Programm ausgelesen werden.
- Programme können die Datentypen und –strukturen von Tabellen lesen, die zum Zeitpunkt des Programmentwurfs noch gar nicht existiert haben. Bei geeigneter Sprachunterstützung wird eine statisch fixierte Datendefinition im Programmtext obsolet. Das Data Dictionary ist somit ein zentrales Instrument in der Anwendungsentwickung, wenn es darum geht, Datendefinition und -modellierung von der Programmentwicklung zu entkoppeln.

Anwendungsübergreifende Datenkonsistenz

Einer der Vorteile eines wohldefinierten Data Dictionary ist die Konsistenz der definierten Datenelemente über verschiedene Tabellen einer Datenbank. Beispielsweise können verschiedene Tabellen das Datenelement *TelefonNr* enthalten; mit einem Data Dictionary kann gewährleistet werden, dass alle Tabellen auf das gleiche Datenelement verweisen. Somit kann eine datenbankweite Konsistenz und ein Verwendungsnachweis für alle Tabellenfelder und Datenelemente erreicht werden.

BNF-Syntax zum Schreiben von Data Dictionaries (DD)

– Kontext: Datenflussdiagramm (DFD), ERM, Systemmodellierung

Ein DD kann in seiner groben Strukturierung in der BNF-Notation geschrieben werden und besteht dann aus mehreren Definitionen, die jeweils in einer Zeile stehen. In einer Definition wird in Form einer Zuweisung geschrieben:

<Nicht-Atomarer Begriff> = <Ausdruck>

Links von der Zuweisung steht ein nicht-atomarer Begriff. Rechts von der Zuweisung steht eine Regel. Eine Regel besteht aus einer Kombination von atomaren und nicht-atomaren Begriffen. Wiederholungen sind möglich. Zirkuläre Definitionen sind verboten, Rekursionen hingegen erlaubt.

= Zuweisung

() Optional

[|] Auswahl

{ } Wiederholung

Beispiel:

Kundenkarte = Anrede + (Titel) + Vorname + Name + {Adresse} + Telefon

Adresse = Strasse + Hausnummer + PLZ + Ort

Telefon = Vorwahl + { [0|1|2|3|4|5|6|7|8|9] }

Vorwahl = 0 + [1|2|3|4|5|6|7|8|9] + { [0|1|2|3|4|5|6|7|8|9] }

Strasse = string

Hausnummer = string

Vorname = string

Name = string

Siehe auch: Backus-Naur-Form (BNF, EBNF), Syntaxdiagramme

Eine Darstellung in dieser Form erlaubt allerdings nur das Aufzeigen der Metabegriffe und ihrer Beziehungen untereinander. In der Anwendungsentwicklung ist der einzelne Metabegriff in der Regel Träger von vielfältigen Zusatzinformationen wie

- fachliche Beschreibung in Umgangssprache
- Domänenwerte
- Feldbeschreibungen (Kurz-, Mittel- und Langtext) für den Formularentwurf
- Information, ob das Datenelement leer (z. B. NULL) sein darf
- Verweise auf Prüftabellen
- etc.

Deshalb kann die BNF-Notation zwar dazu dienen, die Entitäten aufzulisten und deren Beziehungen untereinander darzustellen. Für eine feingliederige Attributierung und tiefergehende Dokumentation wird die BNF-Notation sinnvollerweise durch ein eigentliches DD-Instrument unterstützt.

Siehe auch

- Datenelement
- Datenmodell
- Metadatenverzeichnis
- Semantisches Spektrum

Weblinks

- *ISO/IEC 11179 (2003–2005)* in der englischsprachigen Wikipedia
  - Information technology – Metadata registries [2] – ISO-Eintrag
- *ISO/IEC 10027 (1990)* in der englischsprachigen Wikipedia
  - Information technology – Information Resource Dictionary System (IRDS) framework [3] – ISO-Eintrag

Literatur

- Ramez Elmasri, Shamkant B. Navathe: *Grundlagen von Datenbanksystemen*; Pearson Studium, München 2004. ISBN 3-8273-7021-3
- Thomas Connolly, Carolyn Begg, Anne Strachan: *Datenbanksysteme. Eine praktische Anleitung zu Design, Implementierung und Management*; Addison-Wesley, München 2002. ISBN 3-8273-2013-5

Einzelnachweise

[1] Elmasri, S. 625

[2] http://www.iso.org/iso/en/StandardsQueryFormHandler.StandardsQueryFormHandler?scope=CATALOGUE&keyword=11179

[3] http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=17985&scopelist=

Entity-Relationship-Modell

Das **Entity-Relationship-Modell**, kurz ER-Modell oder **ERM**, deutsch **Gegenstands-Beziehungs-Modell**, dient dazu, im Rahmen der semantischen Datenmodellierung einen Ausschnitt der realen Welt zu beschreiben. Das ER-Modell besteht aus einer Grafik (ER-Diagramm) und einer Beschreibung der darin verwendeten Elemente, wobei Dateninhalte (d. h. die Bedeutung oder Semantik der Daten) und Datenstrukturen dargestellt werden.

Ein ER-Modell dient sowohl in der konzeptionellen Phase der Anwendungsentwicklung der Verständigung zwischen Anwendern und Entwicklern (dabei wird nur das *Was*, also die Sachlogik, und nicht das *Wie*, also die Technik, behandelt), als auch in der Implementierungsphase als Grundlage für das Design der – meist relationalen – Datenbank.

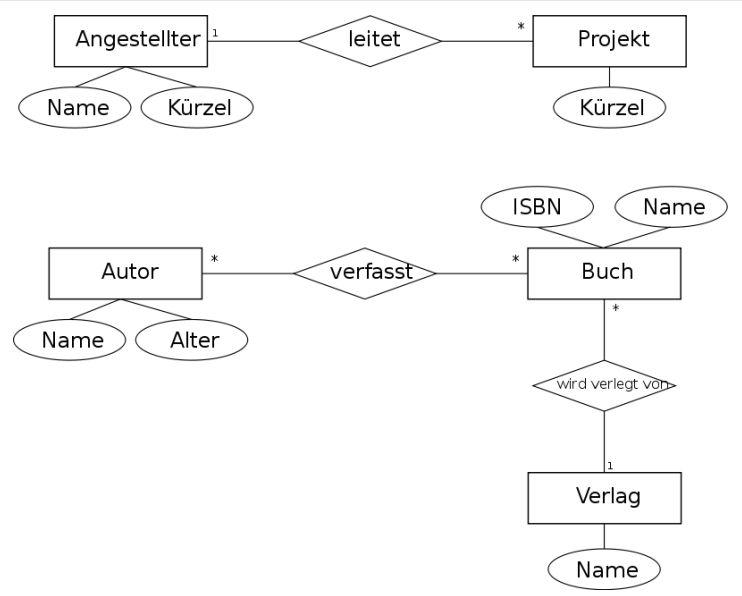
Der Einsatz von ER-Modellen ist der De-facto-Standard für die Datenmodellierung, auch wenn es unterschiedliche grafische Darstellungsformen gibt.

Das ER-Modell wurde 1976 von Peter Chen in seiner Veröffentlichung *The Entity-Relationship Model* vorgestellt. Die Beschreibungsmittel für Generalisierung und Aggregation wurden 1977 von Smith and Smith eingeführt. Danach gab es mehrere Weiterentwicklungen, so Ende der 1980er Jahre durch Wong und Katz.

Begriffe

Grundlage der Entity-Relationship-Modelle ist die Typisierung von Objekten und deren Beziehungen untereinander:

- Entität (Entity): individuell identifizierbares Objekt der Wirklichkeit (zum Beispiel Angestellter „Müller“, Projekt „3232“)
- Entitätstyp: Typisierung gleichartiger Entitäten (zum Beispiel Angestellter, Projekt, Buch, Autor, Verlag)
- Beziehung (Relationship): Verknüpfung zwischen zwei oder mehreren Entitäten (zum Beispiel „Angestellter Müller **leitet** Projekt 3232“)
- Beziehungstyp: Typisierung gleichartiger Beziehungen (zum Beispiel „Angestellter **leitet** Projekt“)



Beispiele für ERMs in Chen-Notation

- reflexive (selbstbezügliche) Beziehung: Beziehung zwischen einzelnen Entitäten ein und desselben Entitätstyps, somit ein Beziehungstyp zwischen dem selben Entitätstyp (zum Beispiel die Baumstruktur einer Aufbauorganisation durch „Organisationseinheit **gliedert sich in** Organisationseinheit“ und die Netzstruktur einer Stückliste durch „Teil **wird verwendet in** Teil“)
- Grad oder Komplexität eines Beziehungstyps: Anzahl der Entitätstypen, die an einem Beziehungstyp beteiligt sind. Die Regel ist der Grad 2 (binärer Beziehungstyp); selten tritt der Grad 3 (ternärer Beziehungstyp) oder ein höherer Grad auf. Ternäre und höhergradige Beziehungstypen lassen sich näherungsweise auf binäre Beziehungstypen durch Einführung eines neuen Entitätstyps, der den ursprünglichen Beziehungstyp beschreibt, zurückführen. Diese Abbildung kann aber verlustbehaftet sein, d.h. es gibt Sachverhalte, die nur durch mehrstellige Beziehungstypen exakt darstellbar sind.

- Kardinalität: Die Kardinalität eines Beziehungstyps legt fest, an wie vielen Beziehungen eine Entität teilnehmen kann (zum Beispiel kann ein Angestellter mehrere Projekte leiten, während ein Projekt von genau einem Angestellten geleitet wird).
- Attribut: Eigenschaft eines Entitätstyps (zum Beispiel Nachname und Geburtsdatum von Entitätstyp Angestellter). Das Attribut oder die Attributkombination eines Entitätstyps, deren Wert(e) die Entität eindeutig beschreiben, d.h. diese identifizieren, heißen identifizierende(s) Attribut(e) (zum Beispiel ist das Attribut Projektnummer identifizierend für den Entitätstyp Projekt). Üblicherweise haben 1:n-Beziehungstypen keine Attribute, da diese immer einem der beteiligten Entitätstypen zugeordnet werden können. Im Falle eines n:m-Beziehungstyps kann aus dem Beziehungstyp ein eigenständiger Entitätstyp mit Beziehungstypen zu den ursprünglich beteiligten Entitätstypen geschaffen werden. Dem neuen Entitätstyp kann dann das Attribut zugeordnet werden (zum Beispiel Attribut Projektbeteiligungsgrad beim n:m-Beziehungstyp „Angestellter arbeitet am Projekt“ zwischen den Entitätstypen Angestellter und Projekt).
- Starker Entitätstyp: Die Identifikation einer Entität ist durch ein oder mehrere Werte von Attributen des gleichen Entitätstyps möglich; so ist z. B. die Auftragsnummer für den Entitätstyp Auftrag identifizierend.
- Schwacher Entitätstyp: Zur Identifikation einer solchen Entität ist ein Attributwert einer anderen mit der schwachen Entität in Beziehung stehenden Entität starken Typs erforderlich; so ist z. B. für die Identifikation des schwachen Entitätstyps Auftragsposition neben der Positionsnummer die Auftragsnummer des anderen starken Entitätstyps Auftrag erforderlich. In Erweiterungen des ER-Modells wie bspw. dem SERM werden Schwacher Entitätstyp und dazugehöriger Beziehungstyp zu einem sogenannten ER-Typen zusammengezogen, wodurch Diagramme kompakter werden.

Bei der Datenmodellierung wird in den Diskussionen und Beispielen oft mit den konkreten Objekten gearbeitet (Entitäten und Beziehungen). Das Modell selbst besteht aber immer ausschließlich aus Entitätstypen und Beziehungstypen. Vielleicht wird aus diesem Grund oftmals nicht sauber zwischen den Begriffen unterschieden.

Beziehungen mit spezieller Semantik

Die inhaltliche Bedeutung der Beziehungstypen zwischen Entitätstypen kommt im ER-Diagramm lediglich durch einen kurzen Text in der Raute (meistens ein Verb) oder als Beschriftung der Kante zum Ausdruck, wobei es dem Modellierer freigestellt ist, welche Bezeichnung er vergibt. Nun gibt es Beziehungen mit spezieller Semantik, die relativ häufig bei der Modellierung vorkommen. Daher hat man für diese Beziehungstypen spezielle Bezeichner und grafische Symbole definiert. Spezialisierung und Generalisierung sowie Aggregation und Zerlegung sind ergänzende Beschreibungsmittel mit einer speziellen Semantik. Mit diesen beiden speziellen Beziehungen kann die Realwelt verfeinert oder vergröbert modelliert werden. Mit fest definierten Namen und speziellen grafischen Symbolen wird gezeigt, dass es sich um semantisch vorbesetzte Beziehungen handelt.

Spezialisierung und Generalisierung mittels *is-a*-Beziehung

Bei der Spezialisierung wird ein Entitätstyp als Teilmenge eines anderen Entitätstyps deklariert, wobei sich die Teilmenge (spezialisierte Menge) durch besondere Eigenschaften (spezielle Attribute und/oder Beziehungen) gegenüber der übergeordneten (generalisierten Menge) auszeichnet. Da es sich bei einem Einzelobjekt einer spezialisierten Menge um dasselbe Einzelobjekt der generalisierten Menge handelt, gelten alle Eigenschaften – insbesondere die Identifikation – und alle Beziehungen des generalisierten Einzelobjektes auch für das spezialisierte Einzelobjekt.
Die Beziehung Spezialisierung/Generalisierung wird durch *is-a*/*can-be* (‚ist ein'/‚kann ein … sein') beschrieben. Für *is-a* wird gelegentlich auch *a-kind-of* (‚eine Art …') benutzt. Es handelt sich hierbei um eine 1:c-Beziehung.

Beispiel zur *is-a*-Beziehung: Dackel is-a Hund
und in anderer Leserichtung: Hund can-be Dackel

Die hier beschriebene *is-a*-Beziehung zwischen Mengen darf nicht mit der *is-element-of*-Beziehung, der Zugehörigkeit eines Elements zu einer Menge, verwechselt werden, für die gelegentlich auch die Schreibweise *is-a* verwendet wird, wie z. B. Waldi is-a Dackel.

Die Spezialisierung erhält man durch Aufteilung, während die Generalisierung durch Zusammenführen von gleichen Einzelobjekten mit gemeinsamen Eigenschaften und Beziehungen, die in verschiedenen Entitätstypen vorkommen, in einem neuen Entitätstyp begründet ist. So können z. B. Kunden und Lieferanten zusätzlich zu Geschäftspartnern zusammengeführt werden, da Name, Anschrift, Bankverbindung etc. sowohl bei den Kunden als auch bei den Lieferanten vorkommen.

Die Visualisierung von Spezialisierung und Generalisierung ist im ursprünglichen ERM Diagramm nicht vorgesehen, aber in Erweiterungen wie z. B. dem SERM.

Aggregation und Zerlegung mittels *is-part-of*-Beziehung

Werden mehrere Einzelobjekte (z. B. Person und Hotel) zu einem eigenständigen Einzelobjekt (z. B. Reservierung) zusammengefasst, dann spricht man von Aggregation. Dabei wird das übergeordnet eigenständige Ganze Aggregat genannt; die Teile, aus denen es sich zusammensetzt, heißen Komponenten. Aggregat und Komponenten werden als Entitätstyp deklariert.

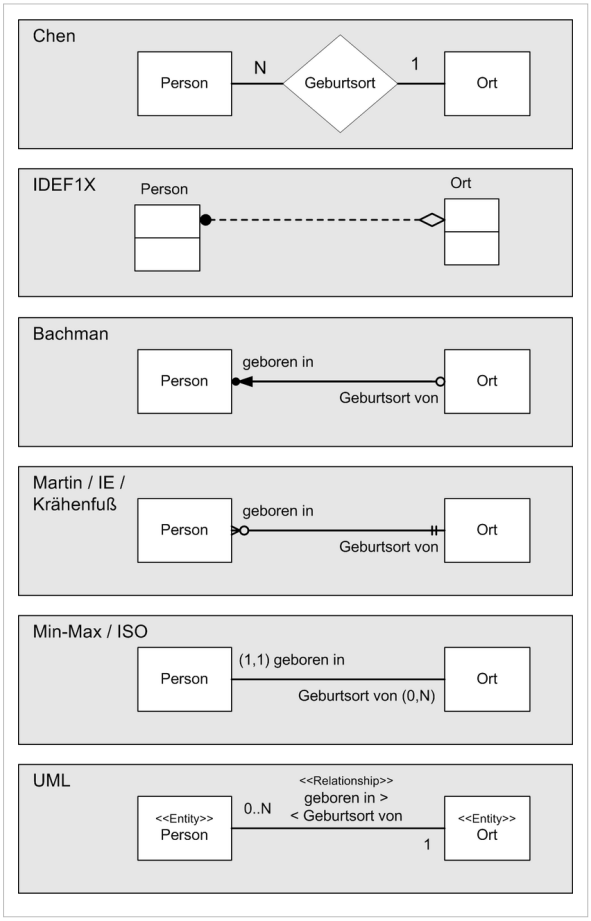
Bei Aggregation/Zerlegung wird zwischen Rollen- und Mengenaggregation unterschieden:
Eine Rollenaggregation liegt vor, wenn es mehrere rollenspezifische Komponenten gibt, diese zu einem Aggregat zusammengefasst werden und es sich um eine 1:c-Beziehung handelt.

Beispiel zur *is-part-of*-Beziehung: Fußballmannschaft is-part-of Fußballspiel und Spielort is-part-of Fußballspiel und in anderer Leserichtung: Fußballspiel besteht-aus Fußballmannschaft und Spielort.

Eine Mengenaggregation liegt vor, wenn das Aggregat durch Zusammenfassung von Einzelobjekten aus genau einer Komponente entsteht. Hier liegt eine 1:cN-Beziehung vor.

Beispiel zur Mengenaggregation: Fußballspieler is-part-of Fußballmannschaft
und in anderer Leserichtung: Fußballmannschaft besteht aus (mehreren, N) Fußballspielern.

ER-Diagramme



Die grafische Darstellung von Entitätstypen und Beziehungstypen wird *Entity-Relationship-Diagramm* (ERD) oder *ER-Diagramm* genannt. Es sind unterschiedliche Darstellungsformen in Gebrauch. Für den Entitätstyp wird meistens ein Rechteck verwendet, der Beziehungstyp meistens in Form einer Verbindungslinie mit besonderen Linienenden oder Beschriftungen, die die Kardinalitäten des Beziehungstyps darstellen.

Es gibt heute eine Vielzahl unterschiedlicher *Notationen*, die sich unter anderem in Klarheit, Umfang der grafischen Sprache, Unterstützung durch Standards und Werkzeuge unterscheiden. Im Folgenden finden sich einige wichtige Beispiele, die vor allem deutlich machen, dass bei allen grafischen Unterschieden die Kernaussage der ER-Diagramme nahezu identisch ist.

Von besonderer – zum Teil historischer – Bedeutung sind unter anderem:

- die Chen-Notation von Peter Chen, dem Entwickler der ER-Diagramme, 1976;
- die IDEF1X als langjähriger De-Facto-Standard bei US-amerikanischen Behörden;
- die Bachman-Notation von Charles Bachman als weit verbreitete Werkzeug-Diagramm-Sprache;
- die Martin-Notation (Krähenfuß-Notation) als weit verbreitete Werkzeug-Diagramm-Sprache (Information Engineering);
- die (min, max)-Notation von Jean-Raymond Abrial, 1974.
- UML als Standard, den selbst ISO in eigenen Normen als Ersatz für ER-Diagramme verwendet. Attribute (im Schaubild nicht zu sehen) können als Klassenattribute dargestellt werden; Relationship-Attribute hingegen werden mit Hilfe von Assoziationsklassen modelliert.

Alle nebenstehenden Notationen drücken auf ihre Art den folgenden Sachverhalt aus:

- *Eine Person ist in maximal einem Ort geboren. Ein Ort ist Geburtsort von beliebig vielen Personen.*
- *Ein Ort kann ein Geburtsort sein, muss es aber nicht sein.*

Bis auf das Chen-Diagramm wird diese Aussage ergänzt um:

- *Eine Person muss in einem Ort geboren sein, oder ist in genau einem Ort geboren.*

Die (min, max)-Notation unterscheidet sich grundlegend von den anderen Notationsformen im Hinblick auf die Bestimmung der Kardinalität und den Ort, an dem die Häufigkeitsangabe im ER-Diagramm vorgenommen wird. Bei allen anderen Notationen wird die Kardinalität eines Beziehungstyps dadurch bestimmt, dass für eine Entität des einen Entitätstyps nach der Anzahl der möglichen beteiligten *Entitäten* des anderen Entitätstyps gefragt wird. Bei der Min-Max-Notation hingegen ist die Kardinalität anders definiert. Hierbei wird für jeden der an einem Beziehungstyp beteiligten Entitätstyp nach der kleinst- und größtmöglichen Anzahl der *Beziehungen* gefragt, an denen eine Entität des jeweiligen Entitätstyps beteiligt ist. Das jeweilige Min-Max-Ergebnis wird bei dem Entitätstyp notiert, für den die Frage gestellt worden ist.

Der zahlenmäßige Unterschied zwischen Min-Max-Notation und allen anderen Notationen tritt erst bei ternären und höhergradigen Beziehungstypen hervor. Bei binären Beziehungstypen ist der Unterschied lediglich in einer Vertauschung der Kardinalitätsangaben ersichtlich.

Einsatz in der Praxis

Das ER-Modell kann bei der Erstellung von Datenbanken genutzt werden. Hierbei wird mit Hilfe von ER-Modellen zunächst die Konzeption der Datenbank vorgenommen, auf deren Grundlage dann die Implementierung der Datenbank erfolgt. Die Umsetzung der in der Realwelt erkannten Objekte und Beziehungen in ein Datenbankschema erfolgt dabei in mehreren Schritten:

- Erkennen und Zusammenfassen von Objekten zu Entitätstypen durch Abstraktion (z. B.: Die Kollegen Fritz Maier und Paul Lehmann und viele weitere zum Entitätstyp *Angestellter*);
- Erkennen und Zusammenfassen von Beziehungen zwischen je zwei Objekten zu einem Beziehungstyp (Beispiel: Der Angestellte Paul Lehmann leitet das Projekt *Verbesserung des Betriebsklimas*, und der Angestellte Fritz Maier leitet das Projekt *Effizienzsteigerung in der Verwaltung*. Diese Feststellungen führen zum Beziehungstyp „Angestellter leitet Projekt“.);
- Bestimmung der Kardinalitäten, d. h. der Häufigkeit des Auftretens (Wie z. B.: Ein Projekt wird immer von genau einem Angestellten geleitet, und ein Angestellter darf mehrere Projekte leiten).

All dieses lässt sich in einem ER-Modell darstellen.

Weiter sind folgende Schritte notwendig, deren Ergebnis meistens jedoch nicht grafisch dargestellt wird (so z. B. in der obigen Grafik):

- Bestimmung der relevanten Attribute der einzelnen Entitätstypen.
- Markierung bestimmter Attribute eines Entitätstyps als identifizierende Attribute, so genannte Schlüsselattribute.
- Generierung des Schemas einer relationalen Datenbank mit all seinen Tabellen- und zugehörigen Felddefinitionen mit ihren jeweiligen Datentypen.

Überführung in ein relationales Modell

Die Überführung eines Entity-Relationship-Modells in das Relationen-Modell basiert i.w. auf den folgenden Abbildungen:

- Entitätstyp → Relation
- Beziehungstyp → Fremdschlüssel; im Falle eines n:m-Beziehungstyps → Relation
- Attribut → Attribut.

Die genaue Überführung, die automatisiert werden kann, erfolgt in 7 Schritten:

1. Starke Entitätstypen

Für jeden starken Entitätstyp wird eine Relation *R* mit den Attributen $R = \{a_1, a_2, ..., a_n\} \cup k$ mit *k* als Primärschlüssel und $a_1, a_2, \ldots, a_n$ als Attribute der Entität erstellt.

2. Schwache Entitätstypen

Für jeden schwachen Entitätstyp wird eine Relation *R* erstellt mit den Attributen $R = \{a_1, a_2, ..., a_n\} \cup \{k\}$ mit dem Fremdschlüssel *k* sowie dem Primärschlüssel $\{k\} \cup \{a_x\}$, wobei $\{a_x\}$ den schwachen Entitätstyp und *k* den starken Entitätstyp identifizieren.

3. 1:1-Beziehungstypen

Für einen 1:1-Beziehungstyp der Entitätstypen *T*, *S* wird eine der beiden Relationen um den Fremdschlüssel für die jeweils andere Relation erweitert.

4. 1:N-Beziehungstypen

Für den 1:N-Beziehungstyp der Entitätstypen *T*, *S* wird die mit der Kardinalität N (bzw. 1 in min-max-Notation) eingehende Relation *T* um den Fremdschlüssel der Relation *S* erweitert.

5. N:M-Beziehungstypen

Für jeden N:M-Beziehungstyp wird eine neue Relation *R* mit den Attributen $R = \{a_1, a_2, ..., a_n\} \cup \{k_T\} \cup \{k_S\}$ mit $\{a_1, a_2, ..., a_n\}$ für die Attribute der Beziehung sowie $k_T$bzw. k_S für die Primärschlüssel der beteiligten Relationen erstellt.

6. Mehrwertige Attribute

Für jedes mehrwertige Attribut in *T* wird eine Relation *R* mit den Attributen $R = \{k\} \cup \{a_x\}$ mit $\{a_x\}$ als mehrwertiges Attribut und *k* als Fremdschlüssel auf *T* erstellt.

7. n-äre Beziehungstypen

Für jeden Beziehungstyp mit einem Grad $n > 2$wird eine Relation *R* erstellt mit den Attributen $R = \{k_1, k_2, ..., k_n\} \cup \{a_1, a_2, ...a_m\}$ mit $\{k_1, k_2, ..., k_n\}$ als Fremdschlüssel auf die eingehenden Entitätstypen sowie $\{a_1, a_2, ...a_m\}$ als Attribute des Beziehungstyps. Wenn alle beteiligten Entitytypen mit Kardinalität > 1eingehen, so ist der Primärschlüssel die Menge aller Fremdschlüssel. In allen anderen Fällen umfasst der Primärschlüssel $n - 1$ Fremdschlüssel, wobei die Fremdschlüssel zu Entitytypen mit Kardinalität > 1 in jedem Fall im Primärschlüssel enthalten sein müssen.

Siehe auch

- Liste von Datenmodellierungswerkzeugen
- Structured Entity Relationship Model, auf ER aufbauend, Methodik führt zu Modell in 3NF, incl. Generalisierung/Spezialisierung, kompaktere Darstellung nach Existenzabhängigkeiten sortiert, daher direktes Ablesen von Einstiegspunkten, Zyklen und Schlüsselvererbung möglich.

Literatur

- Peter Pin-Shan Chen: *The Entity-Relationship Model--Toward a Unified View of Data* [1]. In: ACM Transactions on Database Systems 1/1/1976 ACM-Press ISSN , S. 9–36
- Peter Pin-Shan Chen: *Entity-Relationship Modeling--Historical Events, Future Trends, and Lessons Learned* [2]. In: Software Pioneers: Contributions to Software Engineering, Broy M. and Denert, E. (eds.), Springer-Verlag, Berlin, Lecturing Notes in Computer Sciences, June 2002, pp. 296-310, ISBN 3-540-43081-4
- J.M. Smith, D.C.P. Smith: *Database Abstractions: Aggregation and Generalization*, ACM Transactions on Database Systems, Vol. 2, No. 2 (1977), S. 105–133
- J. M. Smith and D. C. P. Smith: *Database Abstraction: Aggregation*, Communications of the ACM, Vol. 20, Nr. 6, pp. 405–413, June 1977
- Ramez Elmasri, Shamkant B. Navathe: *Fundamentals of database systems.* Addison Wesley, ISBN

Referenzen

[1] http://csc.lsu.edu/news/erd.pdf

[2] http://bit.csc.lsu.edu/~chen/pdf/Chen_Pioneers.pdf

Zustandsübergangsdiagramm

Ein **Zustandsübergangsdiagramm** ist eine grafische Darstellung von endlichen Automaten, d. h. Zuständen und deren Übergangsbedingungen, um die enthaltenen Verknüpfungen möglichst durchschaubar und eindeutig zu visualisieren.

Anwendung findet das Zustandsübergangsdiagramm im Rahmen der Systemtheorie in den verschiedensten Bereichen der Informatik. Eine wesentliche Vereinheitlichung wurde durch David Harels **Statechart**-Notation erreicht, welche weithin als die allgemeingültige Form von Zustandsübergangsdiagrammen gesehen wird. Für die objektorientierte Softwareentwicklung ist es mittlerweile im Rahmen der Unified Modeling Language (UML) normiert als Zustandsdiagramm (UML). Es existieren insbesondere im historischen Kontext noch weitere Varianten der Darstellung.

Statecharts

Statechart ist eine Darstellungsform eines Endlichen Automaten in der Informatik, die von David Harel eingeführt wurde. Die Notation erlaubt die präzise Spezifikation von zustandsbasierten Systemen. David Harel führte mehrere Notationselemente ein, um die Komplexität großer Systeme mittels Endlicher Automaten handhabbar zu machen:

- **Hierarchie** mit Unterzustandsautomaten, in denen in einem Zustand einer höheren Ebene ein weiterer vollständiger Zustandsautomat steckt. Die Unterzustandsautomaten können entweder einen eigenen Startzustand haben, oder aber können Unterzustände direkt angesprungen werden.
- **Komposition** für die Darstellung von parallelen Zustandsautomaten. Hierbei sind AND und OR-Komposition möglich, die ein gleichzeitiges oder abwechselndes Schalten der Automaten vorsehen
- **Inter-Level-Transitionen**, welche auch einen Unterzustand in einen Zustand einer anderen Ebene überführen können und vice versa.
- **History**-Konnektor, der für einen Unterzustandsautomat bei dessen Verlassen den zuletzt eingenommenen Zustand speichert, um beim Wiedereintritt in den Unterzustandsautomat diesen Zustand wieder einzunehmen. Der History-Konnektor wird mit einem eingekreisten H notiert.
- **Condition**-Konnektor, der einen Zustandsübergang (eine Transition) abhängig von einer Bedingung in disjunkt in verschiedene Zielzustände überführt. Der Condition-Konnektor wird mit einem eingekreisten C notiert.
- **Temporale Logik** kann in den Transitionen verwendet werden, um beispielsweise Timeouts anzugeben.
- **Entry**-, **Exit**-, **Throughout**-Actions von Zuständen, welche Aktionen angeben, die beim Eintreten, Verlassen bzw. während des Aufenthalts in einem Zustand ausgeführt werden. Die Anzahl der Ausführungen der **Throughout**-Aktion hängt somit auch von der Taktung des Zustandsautomaten ab.

Werkzeuge

Neben den UML-Werkzeugen sind Zustandsübergangsdiagramme auch in einer Reihe weiterer sowohl offener als auch kommerzieller Werkzeuge implementiert, z.B. (Liste unvollständig)

- The MathWorks Stateflow für MATLAB/Simulink
- case/4/0
- ETAS ASCET
- Telelogic Statemate
- National Instruments MatrixX
- EasyCODE

Siehe auch

- Zustandsdiagramm

Literatur

- David Harel: Statecharts: *A Visual Approach to Complex Systems*, CS84-05, Department of Applied Mathematics, The Weizmann Institute of Science, 1984
- David Harel: Statecharts: A Visual Formalism for Complex Systems. In: *Science of Computer Programming*. 8/1987, North Holland, S. 231-274, (PDF [1])

Referenzen

[1] http://www.informatik.uni-stuttgart.de/fmi/szs/teaching/ss2006/SoftTechII/literatur/Statecharts-Harel87.pdf

Schnittstelle (UML)

Eine **Schnittstelle** (engl. *interface*) ist ein Modellelement in der Unified Modeling Language (UML), einer Modellierungssprache für Software und andere Systeme.

Eine Schnittstelle deklariert eine Liste von Attributen, Operationen und Signalempfängern, die alle öffentliche Sichtbarkeit haben.

Die UML2 unterscheidet zwischen angebotenen und benötigten Schnittstellen. Eine *angebotene Schnittstelle* ist eine Schnittstelle, die ein Classifier realisiert und damit anbietet. Eine *benötigte Schnittstelle* ist eine Schnittstelle, die ein Classifier benötigt, um seine Funktion wahrzunehmen.

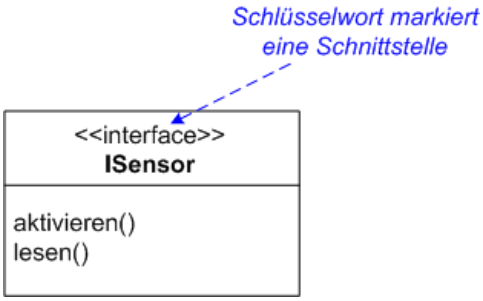
Wenn ein Classifier eine Schnittstelle *anbietet*, dann sichert er damit zwei Dinge zu. Erstens deklariert er, dass er alle Operationen der Schnittstelle realisiert und zweitens verspricht er, dass er alle Attribute auf eine geeignete Art und Weise umsetzt. Dass er dabei über genau die gleichen Attribute wie die Schnittstelle verfügen muss, ist nicht zwingend. Es reicht aus, dass der Classifier ein Attribut zum Beispiel mit einem Paar von Operationen nachbildet, wobei die eine Operation den lesenden und die andere den schreibenden Zugriff auf das Attribute simuliert. Man spricht in diesem Zusammenhang auch von einem Paar von *Setter*- und *Getter*-Operationen.

Im Fall einer *benötigten* Schnittstelle gelten diese Aussagen sinngemäß nicht als Zusicherung sondern als Anforderung. Ein Classifier, der eine Schnittstelle benötigt, erwartet, dass die Operationen und Attribute auf geeignete Art und Weise durch einen zweiten Classifier, der die Schnittstelle realisiert, zur Verfügung gestellt werden.

Dass eine Schnittstelle eine angebotene oder eine benötigte Schnittstelle eines Classifiers ist, stellt ein UML-Modell immer als Abhängigkeitsbeziehung zwischen dem Classifier und der Schnittstelle dar. Im Fall der angebotenen Schnittstelle handelt es sich um eine Schnittstellenrealisierungsbeziehung, bei einer benötigten Schnittstelle um eine Verwendungsbeziehung. Die beiden Arten von Schnittstellen werden in einen Klassendiagramm unterschiedlich und in verschiedenen Varianten dargestellt, wobei die zugrunde liegenden Abhängigkeitsbeziehungen nicht jeder Notationsvariante explizit ausgewiesen wird (siehe nächster Abschnitt).

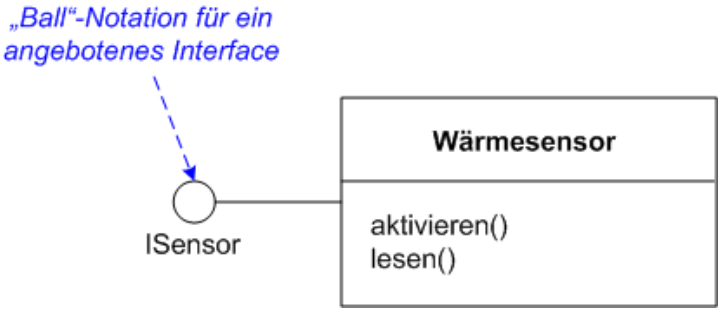
Notation

Eine Schnittstelle wird ähnlich wie eine Klasse mit einem Rechteck dargestellt. Blaue Texte sind erläuternde Kommentare und gehören nicht zur Notation der UML2.



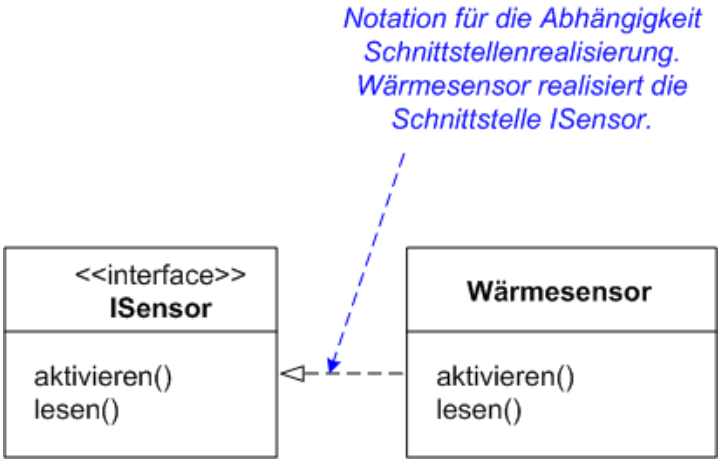
Eine Schnittstelle wird mit dem Schlüsselwort <<interface>> markiert

Die beiden folgenden Abbildungen zeigen zwei Möglichkeiten für die Darstellung von angebotenen Schnittstellen.

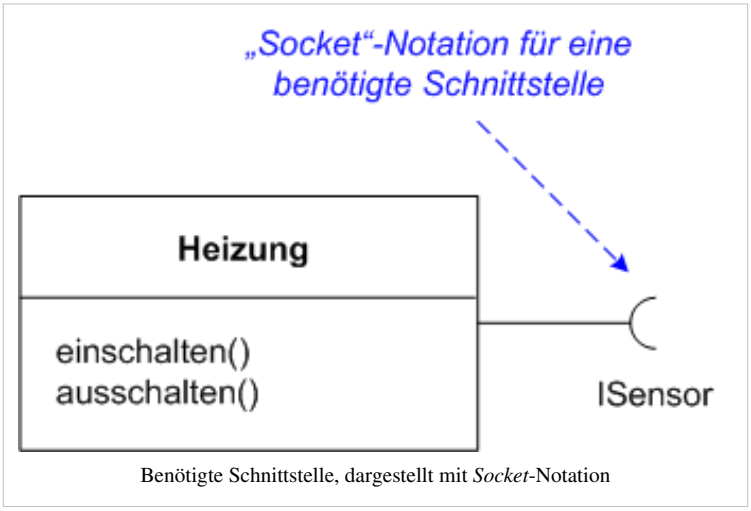


Angebotene Schnittstelle, dargestellt mit *Ball*-Notation (auch *Lollipop*-Notation)

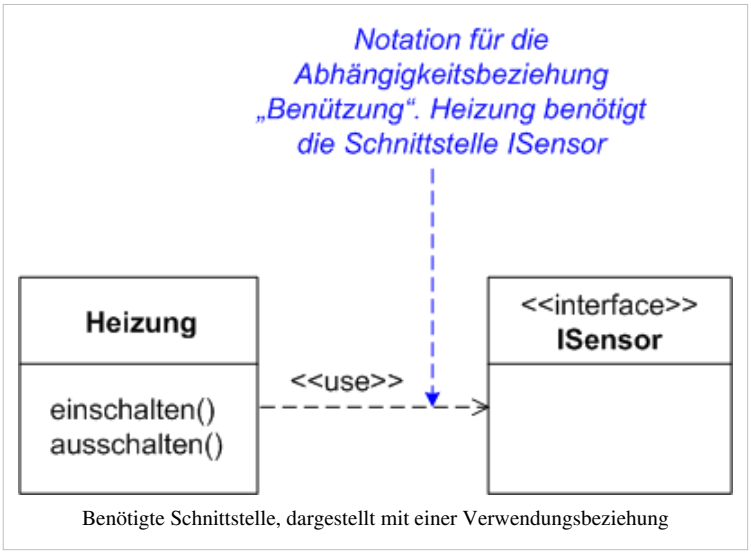
Die beiden folgenden Abbildungen zeigen zwei Möglichkeiten für die Darstellung von benötigten Schnittstellen.



Angebotene Schnittstelle, dargestellt mit einer Schnittstellenrealisierungsbeziehung



Benötigte Schnittstelle, dargestellt mit *Socket*-Notation



Benötigte Schnittstelle, dargestellt mit einer Verwendungsbeziehung

Unterschiede zur UML 1.4

In der UML 1.4, der Vorgängerversion von UML2, gab es ebenfalls ein Modellelement *Schnittstelle*. Die Möglichkeiten zur Modellierung wurden in der UML2 jedoch wesentlich erweitert. Nun können Schnittstellen auch Attribute und Signalempfänger enthalten sowie über Assoziationen mit anderen Schnittstellen verbunden sein.

Neu ist auch die Unterscheidung in angebotene und benötigte Schnittstelle sowie die *Socket*-Notation für benötigte Schnittstelle.

Siehe auch

- Klasse
- Classifier
- Interfacedesign

Pseudocode

Pseudocode ist eine sprachliche Mischung aus natürlicher Sprache, mathematischer Notation und einer höheren Programmiersprache. Wie Flussdiagramme und Nassi-Shneiderman-Diagramme ist auch Pseudocode eine Möglichkeit, Algorithmen darzustellen. Ein Algorithmus wird in Pseudocode einerseits genauer beschrieben als in natürlicher Sprache, anderseits aber noch nicht so detailliert wie durch ein Computerprogramm. Ein Programm in Pseudocode dient ausschließlich dazu, um von Menschen gelesen, nicht aber von einem Computer ausgeführt zu werden. Pseudocode ist also *keine* Programmiersprache.

Für Pseudocode gibt es keine verbindlichen Vorschriften. Jeder kann seine eigene Variante zurechtschneidern. Das Ziel ist, Algorithmen verständlich und klar auszudrücken, ohne auf die Eigenheiten einer Programmiersprache Rücksicht nehmen zu müssen.

Verwendung

Um einen Algorithmus zu verstehen, kann man ihn als Programm untersuchen. Das wird aber erschwert durch die Eigenheiten der Programmiersprache, vor allem ihre Syntax. Zudem haben verschiedene Programmiersprachen unterschiedliche Syntaxen. Jede Formulierung als Programm in einer bestimmten Programmiersprache schließt alle Leser aus, die dieser Sprache nicht mächtig sind. Deshalb formuliert man den Algorithmus zwar ähnlich einem Programm, aber ohne auf eine bestimmte Programmiersprache einzugehen: in Pseudocode.

Pseudocode wird dann eingesetzt, wenn die Funktionsweise eines Algorithmus erklärt werden soll und Einzelheiten der Umsetzung in eine Programmiersprache stören würden. Ein typisches Beispiel sind die Felder, die in Pascal von Eins an indiziert werden, in C dagegen von Null an. In Lehrbüchern werden deshalb Algorithmen gelegentlich in Pseudocode wiedergegeben.

Man kann ein Programm durch Pseudocode spezifizieren. Das sollte allerdings eher vermieden werden, denn die Formulierung als Pseudocode ist bereits eine Programmiertätigkeit, die von der Konzentration auf die Anforderungen ablenkt.[1]

Auch bei der Entwicklung von Algorithmen und der Umformung von Programmen (Programmtransformation, Refactoring) wird Pseudocode eingesetzt.

Aussehen und Stilrichtungen

Pseudocode hat den Anspruch, intuitiv klar zu sein. Geeignete Metaphern aus der Umgangssprache geben einen Verfahrensschritt prägnant wieder, ohne dass dazu eine Erklärung nötig ist, zum Beispiel "durchlaufe das Feld a mit Index i" oder "vertausche die Inhalte der Variablen x und y". Solche Stilmittel verbessern die Übersicht.

Pseudocode kann sich in seinem Stil an eine bestimmte höhere Programmiersprache anlehnen, zum Beispiel an Pascal oder an C.

Im Pascal-Stil werden Schlüsselwörter wie `begin`, `end`, `then`, `do`, `repeat`, `until` benutzt. Im C-Stil werden stattdessen geschweifte Klammern `{,}` gesetzt und das Schlüsselwort `then` wird ausgelassen. Dieser Stil wird oft von Programmierern benutzt, die solche Sprachen verwenden. Beide Stile findet man in Lehrbüchern.

Die Blockstruktur wird gelegentlich auch nur durch Einrücken wiedergegeben.

Eine Liste häufig verwendeter Schlüsselwörter:

Module

- `program` *`Programmname`* `...` `end` *`Programmname`*
- `klasse` *`Klassenname`* `{ ... }`

Fallunterscheidungen

- `if ... then ... else ... end if/exit`
- `wenn ... dann ... sonst ... wenn_ende`
- `falls ... dann ... falls_nicht ... falls_ende`

Schleifen

- `wiederhole ... solange/bis ... wiederhole_ende`
- `while ... do ...`
- `repeat ... until ...`
- `for ... to ... step` *`Schrittweite`* `... next`

Kommentare

- `// kommentar`
- `# kommentar`
- `/* kommentar */`

Definition von Funktionen

- `function() ... begin ... end`
- `funktion() ... start ... ende`

Zusicherungen

- `assert`
- `jetzt gilt`

Beispiele

```
program Name und Kurzbeschreibung
 LiesDatenStruktur()
 LiesDatenInhalt()
 ...
 if DatenUnvollständig then FehlerMelden und exit
 HauptstatistikBerechnen
 ZusammenstellungBerechnen
 Resultate in HTML-Datei schreiben
end program Name
```

Prozedur: euklid
Zweck: Euklidischer Algorithmus zur Berechnung des größten gemeinsamen Teilers
Parameter: natürliche Zahlen m, n
1. **Falls** $m > n$ **, dann** m und n miteinander vertauschen.
2. **Jetzt gilt** $m \leq n$.
3. **Solange** $m > 0$ **wiederhole**
4. **Setze** $n = n - m$.
5. **Falls** $m > n$ **, dann** m und n miteinander vertauschen.
6. **Jetzt gilt** $m \leq n$.

Ergebnis: n.

Siehe auch

- Programmablaufplan
- Jana (Beschreibungssprache)
- Nassi-Shneiderman-Diagramm
- Kontrollstruktur

Einzelnachweise

[1] Johannes Siedersleben (Hrsg.): *Softwaretechnik.* Hanser, München 2003, ISBN 3-446-21843-2, S. 44ff..

Programmablaufplan

DIN	DIN 66001
Bereich	Informationsverarbeitung
Titel	Sinnbilder für Datenfluss- und Programmablaufpläne
Letzte Ausgabe	1983
ISO	5807

Ein **Programmablaufplan** (**PAP**) ist ein Ablaufdiagramm für ein Computerprogramm, das auch als *Flussdiagramm* (engl. *flowchart*) oder *Programmstrukturplan* bezeichnet wird. Es ist eine graphische Darstellung zur Umsetzung eines Algorithmus in einem Programm und beschreibt die Folge von Operationen zur Lösung einer Aufgabe.

Die Symbole für Programmablaufpläne sind in der **DIN 66001** genormt. Dort werden auch Symbole für Datenflusspläne definiert. Programmablaufpläne werden oft unabhängig von Computerprogrammen auch zur Darstellung von Prozessen und Tätigkeiten eingesetzt (z. B. als Beschreibung des Arbeitsablaufs bei der Angebotserstellung in einem Handelsunternehmen). Im Bereich der Softwareerstellung werden sie nur noch selten verwendet: Programmcode moderner Programmiersprachen bietet ähnlichen Abstraktionsgrad, ist jedoch einfacher zu erstellen und in der Regel sehr viel einfacher zu verändern als ein Ablaufdiagramm.

Das Konzept der Programmablaufpläne stammt, ebenso wie das etwas jüngere Nassi-Shneiderman-Diagramm (Struktogramm), aus der Zeit des imperativen Programmierparadigmas. Bei der Abbildung objektorientierter Programmkonzepte durch UML finden erweiterte Programmablaufpläne (Aktivitätsdiagramme) Anwendung.

Elemente

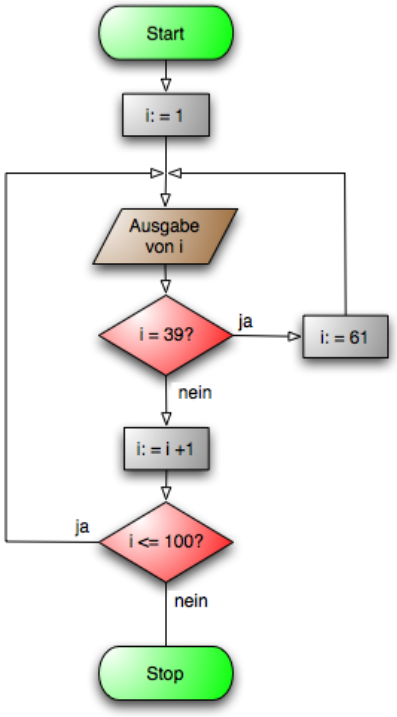
Hauptsächlich werden die folgenden Elemente verwendet (vollständige Liste s. Weblink zu DIN 66001):

- 6.4.1: Oval: Start, Stopp, weitere Grenzpunkte

- 6.3.1: Pfeil, Linie: Verbindung zum nächstfolgenden Element
- 6.1.1: Rechteck: Operation
- 7.2.4: Rechteck mit doppelten, vertikalen Linien: Unterprogramm aufrufen
- 6.1.3: Raute: Verzweigung
- 6.2.1: Parallelogramm: Ein- und Ausgabe (nicht nach DIN 66001 1983)

Beispiel

Die nebenstehende Abbildung zeigt eine Zählschleife. Die Zählvariable i wird vor Beginn der Schleife auf ihren Startwert i=1 gesetzt. Danach wird die erste Anweisung der Schleife, das Ausgeben der Variable i, ausgeführt. Die nachfolgende zweite Anweisung ist eine einseitige Auswahl, die prüft, ob i den Wert 39 besitzt. Falls dies der Fall ist, wird i auf den Wert 61 gesetzt und die Schleife beginnt mit dem nächsten Durchlauf. Falls i nicht 39 ist, wird i in der nachfolgenden Anweisung um eins erhöht und anschließend geprüft, ob die Schleifenabbruchbedingung i>100 erreicht ist. Falls nicht, erfolgt ein nochmaliger Schleifendurchlauf. Ausgegeben würden alle natürlichen Zahlen von 1 bis 39 sowie 61 bis 100 (jeweils einschließlich).



Beispiel eines Flussdiagramms (Programmablaufplans)

Literatur

- Hans Westermayer: *Programmierlogik, Programmablaufpläne.* Oldenbourg, München 1971, ISBN 3-486-38881-9.
- Norbert von Bertoldi, Jutta Bayer: *Programmablaufpläne (PAPs) und Struktogramme professionell erstellen: kaufmännische Prüfungsaufgaben erfolgreich lösen.* IWT-Verlag, Vaterstetten 1993, ISBN 3-88322-448-0.

Weblinks

- DIN 66001 – Sinnbilder für Datenfluss- und Programmablaufpläne (Version von 1966!) [1] (PDF; 1,14 MB)
- DIN 66001 [2]
- Fortran-Beispiel [3]
- PapDesigner-Software für Windows [4]

Referenzen

[1] http://www.fh-jena.de/~kleine/history/software/DIN66001-1966.pdf
[2] http://www.cabeweb.de/html/din66001.htm
[3] http://www.rz.uni-bayreuth.de/lehre/fortran90/vorlesung/V03/V03.html
[4] http://www.gso-koeln.de/papdesigner/

Quelle(n) und Bearbeiter des/der Artikel(s)

Strukturierte Analyse *Quelle*: http://de.wikipedia.org/w/index.php?oldid=73106723 *Bearbeiter*: 1001, Abdull, Alexander.stohr, André Huf, Ath, Beyer, Blubbalutsch, Brigitte Evertz, Cami de Son Duc, Comc, Dkoelle, Erik Warmelink, Fleshgrinder, Frank Jacobsen, Go4wiki, Gubaer, Hardenacke, Jens611, Jpp, KAMiKAZOW, Krawi, Ksweber, Ma-Lik, Millbart, Mion, Ocrho, Pylon, Ramtam, Rax, Rdb, Reinhard Kraasch, S1, SebastianBreier, Sparti, Staro1, Stylor, Wasseralm, Zahnradzacken, 47 anonyme Bearbeitungen

Tom DeMarco *Quelle*: http://de.wikipedia.org/w/index.php?oldid=79729902 *Bearbeiter*: AN, Andim, Asdert, Bademantel, DerHexer, Effactory, Fleshgrinder, Gödeke, HaSee, Jerry Fischer, JohnnyB, Lu Wunsch-Rolshoven, Ma-Lik, Michael Hüttermann, Pelz, Peter200, Pylon, Sinn, Teepott, USt, Video2005, Wasseralm, Zahnradzacken, 17 anonyme Bearbeitungen

Strukturiertes Design *Quelle*: http://de.wikipedia.org/w/index.php?oldid=57375679 *Bearbeiter*: Alexander.stohr, Andrease, GNosis, Habemus pampam, Hubertl, Lostintranslation, Ma-Lik, Monument of the unknown editor, Ocrho, S1, Staro1, YMS, 40 anonyme Bearbeitungen

Komponente (Software) *Quelle*: http://de.wikipedia.org/w/index.php?oldid=75243743 *Bearbeiter*: A Ruprecht, Abdull, Abubiju, Aka, ChristianErtl, CommonsDelinker, DerFred, ErnstRohlicek, Gms, Grammling, Hans Genten, Kako, Kaneiderdaniel, Membeth, Morini4me, Pelz, Schmiddtchen, Shmia, Sparti, W!B:, WikipediaMaster, Wirthi, Wumpus3000, 28 anonyme Bearbeitungen

Modul (Software) *Quelle*: http://de.wikipedia.org/w/index.php?oldid=73754250 *Bearbeiter*: Bautsch, Chpfeiffer, Dertoni, Eggermanuel, Exilfranke, Geher, Hans Genten, Inkowik, Joachimschiele, Jpp, Kadeck, Kopoltra, Levin, Lichtkind, Manecke, Mannerheim, Nikolaus, Raphael Frey, Ri st, S.K., SonniWP2, Sparti, Speck-Made, Stefan h, TheBadWulf, Timk70, W!B:, Westiandi, Zinnmann, 21 anonyme Bearbeitungen

Systemanalyse *Quelle*: http://de.wikipedia.org/w/index.php?oldid=79329541 *Bearbeiter*: 1001, Acky69, Aka, Alexander.stohr, Axel K, ChriFi, Cyrus Grisham, Ephraim33, Erkan Yilmaz, Fleshgrinder, Fmrauch, HannesH, Hati, Heinte, Hubertl, Jan Giesen, Johannes Mockenhaupt, Karl-Henner, Krawi, Ma-Lik, Markus Mueller, Martin k, Matthias4445, Matze12, Mdd, Mion, Norbbi, Rusnak2000, Spitschan, Staro1, Sulfolobus, ThePeritus, Tomdo08, Ulrim, Unscheinbar, WIKImaniac, YMS, Алиса, 35 anonyme Bearbeitungen

Kontextdiagramm *Quelle*: http://de.wikipedia.org/w/index.php?oldid=78033648 *Bearbeiter*: CBlitz, Christian Storm, Jpp, Mike Krüger, Staro1, StillesGrinsen, Wittas, Xerbsd, 16 anonyme Bearbeitungen

Hierarchie *Quelle*: http://de.wikipedia.org/w/index.php?oldid=79878763 *Bearbeiter*: Aberglaube, Aka, Andre Engels, Andrsvoss, Andys, Anima, Ben-Zin, Bender235, Bertonymus, Blaubahn, Bugfix, C.Löser, ChristianGlaeser, ChristophDemmer, ChristophLanger, Complex, DasBee, Der Messer, DerHexer, Detektiv, Diba, Dr. Otterbeck, Duesentrieb, Edoe, El Cazangero, Engie, ErnstGruber, Erwin E aus U, Fischkopp, Fiveop, Flominator, Florian.b, Fristu, Fritzbox, Gerbil, Hans J. Castorp, Hoss, Hutschi, Hæggis, J. 'mach' wust, JakobVoss, Jekub, Joni2, Josemaria, Juditten, Kai-Hendrik, Kalli R, Koethnig, Lexoldie, M-sch, Mardil, Marilyn.hanson, Markus108, Matt1971, Muck31, Musik-chris, Nerd, Nicor, Obersachse, Ocrho, P. Birken, Peter200, Pik-Asso, Rabanus Flavus, Regi51, Reinhard Kraasch, Rho, Robert Huber, Sadduk, Schewek, Seewolf, Sei Shonagon, Sgoo, Sinn, Staro1, Stay cool, Stefan64, StefanWesthoff, Stephan Deichstetter, T.M.L.-KuTV, Tim Pritlove, Tönjes, Umweltschützen, VanGore, Vigala Veia, Volmar, W!B:, WHell, Wegner8, Wikitoni, Wst, Xario, Zumbo, €pa, 105 anonyme Bearbeitungen

Organigramm *Quelle*: http://de.wikipedia.org/w/index.php?oldid=79603177 *Bearbeiter*: A. B., Amens, AndreR, Archwizard, Avron, Berny68, Bloo, Ciciban, Crux, Daferdi, DerAbt, Diba, Dundak, El., Engie, Ernestoz, Erwin E aus U, Euphoriceyes, EvaK, Felix Stember, Flothi, Frauhottelmann, Galameli, Gnu1742, HaSee, Helmut Zenz, HumanConcepts DE, JCBrunner, Jivee Blau, Jpp, JuTa, KWa, Kibert, Krawi, LKD, Logograph, Louis Bafrance, Ma-Lik, Markobr, McLar, Meph666, MichaelDiederich, Mlowin, Morki, Nachtagent, Nicolas G., Noclador, Olegator, Ottomanisch, P. Birken, Peter200, Ploebsi, Polu' Nord, Ppmp3, Purodha, RSX, Regi51, Rosion, SLSCHNÖLL, Sebastian.Dietrich, Sinn, SirJective, Sonjavanderlinden, SonniWP2, Sprenger, Stern, Th., Thorbjoern, Tolliver, U7o, Ulrich.fuchs, Vintagesound, WAH, WissensDürster, Wolfgang1018, Wst, Xqt, Yotwen, YourEyesOnly, Zanki, Zombi, Æshættr, 129 anonyme Bearbeitungen

Datenflussdiagramm *Quelle*: http://de.wikipedia.org/w/index.php?oldid=78033895 *Bearbeiter*: Abschalom, Alexander.stohr, Ancoron, ChristophDemmer, Cjesch, Flominator, HsT, Ilario, Jpp, Kdwnv, LKD, LinveggieLÄ, MatthiasDD, Mm pedia, Sparti, Staro1, Svelix, Tobias Bergemann, Xqt, Zefram, Zeno Gantner, 24 anonyme Bearbeitungen

Entscheidungstabelle *Quelle*: http://de.wikipedia.org/w/index.php?oldid=78547938 *Bearbeiter*: Abubiju, Ath, BSDev, Christian1985, Devil m25, Dodo von den Bergen, Flominator, Florian3, Franz Schlacher, Gadelor, H005, HAL Neuntausend, Happoen, Hardenacke, Hein.Mück, Howwi, KleinPhi, Kubrick, Media lib, Norro, Pik-Asso, Ragalla, Saehrimnir, Snert, Sparti, Tapir2008, Tec, Thomas S., Thomas.Taeger, Tucka, Wasseralm, Westiandi, Wkrautter, 49 anonyme Bearbeitungen

Entscheidungsbaum *Quelle*: http://de.wikipedia.org/w/index.php?oldid=77434621 *Bearbeiter*: Admiral kay, AndreasE, Berni, Bücherwürmlein, Chrislb, ChristophDemmer, Cnagl, Ephraim33, Extremophile, Fkoch, Fluppens, Forrester, Fuenfundachtzig, Gadelor, H005, Head, Hedwig in Washington, Jonathan Hornung, Karawane 71, Karl-Henner, Kinimod, Kku, Korelstar, LosHawlos, Michael Kümmling, MichaelFrey, Mikue, Muck31, Napa, Nerd, Odder, Pemu, Peter Steinberg, Rainbowfish, Sievers, Sigbert, Sircorrectdude, Srixen, Stefan Birkner, Stefan Kühn, Stern, Tucka, WebScientist, 29 anonyme Bearbeitungen

Data Dictionary *Quelle*: http://de.wikipedia.org/w/index.php?oldid=77444485 *Bearbeiter*: Aka, Avron, Bildungsbürger, Blunt., Chrisfrenzel, Dapete, Frank Roeing, HarPaX1209, JCS, Jpp, Ollio, PerfektesChaos, Sparti, Staro1, Thorny, TomVision, Tzeh, 20 anonyme Bearbeitungen

Entity-Relationship-Modell *Quelle*: http://de.wikipedia.org/w/index.php?oldid=79450959 *Bearbeiter*: Achim Raschka, Aka, Alexander.stohr, Alfred Grudszus, Andorna, Androl, Avron, Ben Ben, Bense, Bernd vdB, Bernhard Wallisch, Blubbalutsch, Cami de Son Duc, Chaosdeckel, Christian Storm, Colognese, Complex, Conny, D, D235, Der Hakawati, Der fette mo, DerHexer, Diba, Duesentrieb, Edoe, Eisenberg, Empro2, Erzbischof, Euphoriceyes, Finrod, Fleshgrinder, Frank Roeing, Freedomsaver, Friedemann Lindenthal, Geof, Gerbil, Gerold Broser, Gnu1742, Guandalug, Gurumaker, Hardenacke, He3nry, Head, Herr Th., Howwi, JakobVoss, Jengelh, Jochen, JohnTB, Kallistratos, Koerpertraining, KraetziChriZ, Krawi, Kubrick, Kurt Jansson, LKD, Laluenne, Liberal Freemason, Logograph, MAK, MFM, MaZder, Magnus, Maqqusz, MarkusHagenlocher, Matze12, MauriceKA, MichaelDiederich, Michail, MiersA, Mk85, Mm pedia, Mrieken, MsChaos, Nephiliskos, Netzmeister, Obstriegel, OecherAlemanne, Oemmler, Orcus, P. Birken, Peter200, Pfalzfrank, Rat, Revvar, Robb der Physiker, RobertRoggenbuck, S.K., STBR, Schoschi, Schusch, Sebastian.Dietrich, Semper, Sgeureka, Sicherlich, Sinn, Small Axe, Sparti, Stahlkocher, Staro1, Stefan, Stern, Thetawave, TigerDE2, Tsor, Tzeh, Tönjes, Ulis, Umweltschützen, Volker Fritzsche, WIKImaniac, Wolfgang1018, Woody, YMS, YourEyesOnly, ZenoCosini, °, 280 anonyme Bearbeitungen

Zustandsübergangsdiagramm *Quelle*: http://de.wikipedia.org/w/index.php?oldid=76444734 *Bearbeiter*: Abubiju, Alexander.stohr, Avron, Curtis Newton, DasBee, Easycode, Jpp, Jschlosser, Juranet, Karsten Strehl, Kuhlo, Ma-Lik, Olliwood, Peter O. Mueller, Prettyprinter, Quickfix, RacoonyRE, Reinhard Kraasch, ReqEngineer, S.K., S1, Sommerkom, Staro1, Wuryel, 20 anonyme Bearbeitungen

Schnittstelle (UML) *Quelle*: http://de.wikipedia.org/w/index.php?oldid=60078483 *Bearbeiter*: Comc, Exil, Guandalug, Gubaer, Hildegund, Jpp, Ma-Lik, Macador, Mareike.graf, Mps, Noddy93, Patchworker, 2 anonyme Bearbeitungen

Pseudocode *Quelle*: http://de.wikipedia.org/w/index.php?oldid=79442092 *Bearbeiter*: $traight-$hoota, Aka, AlfonsGeser, Andre Engels, Cocyhok, Complex, Deki, DerHexer, Gstueb, HHK, INM, Jkrieger, Joey-das-WBF, Jonathan Hornung, Kh555, Kinley, Kku, MFM, MH, OecherAlemanne, Peter200, Piecestory, Pz6j89, Rbb, Simon04, Sparti, Sprezzatura, Thornard, Tkarcher, Trustable, Tsor, Umweltschützen, 50 anonyme Bearbeitungen

Programmablaufplan *Quelle*: http://de.wikipedia.org/w/index.php?oldid=79894362 *Bearbeiter*: A.Abdel-Rahim, Aka, Avron, Bacchus81, BeGood, Blaufisch, Bücherwürmlein, Cepheiden, Chip62 m, Cinymini, Conny, Daniel, DerHexer, Diba, Diesterne, Druffeler, Entlinkt, Euphoriceyes, Flaallan, Fleasoft, Flominator, Gaius L., Gnu1742, Grauß, He3nry, Herrick, Homer Landskirty, Hubi, Ian Dury, JanRieke, Jpp, Kaisersoft, Krawi, LKD, Martin-vogel, Mathias Schindler, MatthiasDD, Mmmkay, Mnh, Muvon53, Nevetsjc, Nicolas G., Numbo3, Ocrho, P. Birken, PZ0151, PerfektesChaos, Perni, Peter200, Polarlys, Poupée de chaussette, Regi51, ReiseAxelito, Revvar, Roterraecher, Rp.baumann, Rudolf.l.s, S.lukas, Schmiddtchen, Seewolf, Sinn, Slartidan, Slimcase, Sparti, SpecialAgent, Staro1, Stefan Kühn, Thornard, To old, Tsor, Tönjes, VanGore, WAH, Westiandi, Xqt, Zumbo, Zuse, 172 anonyme Bearbeitungen

Quelle(n), Lizenz(en) und Autor(en) des Bildes

Datei:NDE_Context_Diagram.jpg *Quelle*: http://de.wikipedia.org/w/index.php?title=Datei:NDE_Context_Diagram.jpg *Lizenz*: Public Domain *Bearbeiter*: NDE

Datei:ENC SYSTEME FIGURE.jpeg *Quelle*: http://de.wikipedia.org/w/index.php?title=Datei:ENC_SYSTEME_FIGURE.jpeg *Lizenz*: Public Domain *Bearbeiter*: See

Datei:Beispiel Organigrammsymbole.svg *Quelle*: http://de.wikipedia.org/w/index.php?title=Datei:Beispiel_Organigrammsymbole.svg *Lizenz*: unbekannt *Bearbeiter*: Benutzer:SLSCHNÖLL

Datei:Organigramm 2.png *Quelle*: http://de.wikipedia.org/w/index.php?title=Datei:Organigramm_2.png *Lizenz*: GNU Free Documentation License *Bearbeiter*: Original uploader was Sprenger at de.wikipedia

Datei:DataFlowDiagram Example.png *Quelle*: http://de.wikipedia.org/w/index.php?title=Datei:DataFlowDiagram_Example.png *Lizenz*: Creative Commons Attribution-Sharealike 3.0 *Bearbeiter*: User:AutumnSnow, User:Ilario

Bild:entscheidungsbaum.svg *Quelle*: http://de.wikipedia.org/w/index.php?title=Datei:Entscheidungsbaum.svg *Lizenz*: unbekannt *Bearbeiter*: Chaddy, Karawane 71

Bild:DEU Tutorial - Hochladen von Bildern.svg *Quelle*: http://de.wikipedia.org/w/index.php?title=Datei:DEU_Tutorial_-_Hochladen_von_Bildern.svg *Lizenz*: Public Domain *Bearbeiter*: user:odder

Image:Investment Decision Occam s Tree.gif *Quelle*: http://de.wikipedia.org/w/index.php?title=Datei:Investment_Decision_Occam_s_Tree.gif *Lizenz*: unbekannt *Bearbeiter*: Extremophile

Datei:Er-diagramm.svg *Quelle*: http://de.wikipedia.org/w/index.php?title=Datei:Er-diagramm.svg *Lizenz*: unbekannt *Bearbeiter*: Der Hakawati, Grmon, MaZder, 1 anonyme Bearbeitungen

Datei:ERD Darstellungen.png *Quelle*: http://de.wikipedia.org/w/index.php?title=Datei:ERD_Darstellungen.png *Lizenz*: Public Domain *Bearbeiter*: Frank Roeing

Bild:Interface-1.png *Quelle*: http://de.wikipedia.org/w/index.php?title=Datei:Interface-1.png *Lizenz*: GNU Free Documentation License *Bearbeiter*: Gubaer

Bild:Interface-2.png *Quelle*: http://de.wikipedia.org/w/index.php?title=Datei:Interface-2.png *Lizenz*: GNU Free Documentation License *Bearbeiter*: Gubaer

Bild:Interface-3.png *Quelle*: http://de.wikipedia.org/w/index.php?title=Datei:Interface-3.png *Lizenz*: GNU Free Documentation License *Bearbeiter*: Gubaer

Bild:Interface-4.png *Quelle*: http://de.wikipedia.org/w/index.php?title=Datei:Interface-4.png *Lizenz*: GNU Free Documentation License *Bearbeiter*: Gubaer

Bild:Interface-5.png *Quelle*: http://de.wikipedia.org/w/index.php?title=Datei:Interface-5.png *Lizenz*: GNU Free Documentation License *Bearbeiter*: Gubaer

Bild:DIN-Logo.svg *Quelle*: http://de.wikipedia.org/w/index.php?title=Datei:DIN-Logo.svg *Lizenz*: unbekannt *Bearbeiter*: unbekannt. Original uploader was Afrank99 at de.wikipedia. Later version(s) were uploaded by Schnelliboy at de.wikipedia.

Datei:Oval (Programmablaufplan).png *Quelle*: http://de.wikipedia.org/w/index.php?title=Datei:Oval_(Programmablaufplan).png *Lizenz*: Creative Commons Attribution-Sharealike 2.0 *Bearbeiter*: User:Daniel

Datei:Flussdiagramm (Programmablaufplan).png *Quelle*: http://de.wikipedia.org/w/index.php?title=Datei:Flussdiagramm_(Programmablaufplan).png *Lizenz*: Creative Commons Attribution-Sharealike 2.0 *Bearbeiter*: User:Daniel

Lizenz

Wichtiger Hinweis zu den Lizenzen

Die nachfolgenden Lizenzen bezieht sich auf den Artikeltext. Im Artikel gezeigte Bilder und Grafiken können unter einer anderen Lizenz stehen sowie von Autoren erstellt worden sein, die nicht in der Autorenliste erscheinen. Durch eine noch vorhandene technische Einschränkung werden die Lizenzinformationen für Bilder und Grafiken daher nicht angezeigt. An der Behebung dieser Einschränkung wird gearbeitet.
Das PDF ist daher nur für den privaten Gebrauch bestimmt. Eine Weiterverbreitung kann eine Urheberrechtsverletzung bedeuten.

Creative Commons Attribution-ShareAlike 3.0 Unported - Deed

Diese "Commons Deed" ist lediglich eine vereinfachte Zusammenfassung des rechtsverbindlichen Lizenzvertrages (http://de.wikipedia.org/wiki/Wikipedia:Lizenzbestimmungen_Commons_Attribution-ShareAlike_3.0_Unported) in allgemeinverständlicher Sprache.
Sie dürfen:

- das Werk bzw. den Inhalt **vervielfältigen, verbreiten und öffentlich zugänglich machen**
- **Abwandlungen und Bearbeitungen** des Werkes bzw. Inhaltes anfertigen

Zu den folgenden Bedingungen:

- **Namensnennung —** Sie müssen den Namen des Autors/Rechteinhabers in der von ihm festgelegten Weise nennen.
- **Weitergabe unter gleichen Bedingungen —** Wenn Sie das lizenzierte Werk bzw. den lizenzierten Inhalt bearbeiten, abwandeln oder in anderer Weise erkennbar als Grundlage für eigenes Schaffen verwenden, dürfen Sie die daraufhin neu entstandenen Werke bzw. Inhalte nur unter Verwendung von Lizenzbedingungen weitergeben, die mit denen dieses Lizenzvertrages identisch, vergleichbar oder kompatibel sind.

Wobei gilt:

- **Verzichtserklärung —** Jede der vorgenannten Bedingungen kann aufgehoben werden, sofern Sie die ausdrückliche Einwilligung des Rechteinhabers dazu erhalten.
- **Sonstige Rechte —** Die Lizenz hat keinerlei Einfluss auf die folgenden Rechte:
  - Die gesetzlichen Schranken des Urheberrechts und sonstigen Befugnisse zur privaten Nutzung;
  - Das Urheberpersönlichkeitsrecht des Rechteinhabers;
  - Rechte anderer Personen, entweder am Lizenzgegenstand selber oder bezüglich seiner Verwendung, zum Beispiel Persönlichkeitsrechte abgebildeter Personen.
- **Hinweis —** Im Falle einer Verbreitung müssen Sie anderen alle Lizenzbedingungen mitteilen, die für dieses Werk gelten. Am einfachsten ist es, an entsprechender Stelle einen Link auf http://creativecommons.org/licenses/by-sa/3.0/deed.de einzubinden.

Haftungsbeschränkung
Die „Commons Deed" ist kein Lizenzvertrag. Sie ist lediglich ein Referenztext, der den zugrundeliegenden Lizenzvertrag übersichtlich und in allgemeinverständlicher Sprache, aber auch stark vereinfacht wiedergibt. Die Deed selbst entfaltet keine juristische Wirkung und erscheint im eigentlichen Lizenzvertrag nicht.

GNU Free Documentation License

Version 1.2, November 2002

> Copyright (C) 2000,2001,2002 Free Software Foundation, Inc.
>
> 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
>
> Everyone is permitted to copy and distribute verbatim copies
>
> of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.
This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.
We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.
A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.
A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.
The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.
The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.
A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".
Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.
The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.
A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.
The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.
You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.
If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.
If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.
It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- **A.** Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- **B.** List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- **C.** State on the Title page the name of the publisher of the Modified Version, as the publisher.
- **D.** Preserve all the copyright notices of the Document.
- **E.** Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- **F.** Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- **G.** Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- **H.** Include an unaltered copy of this License.
- **I.** Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- **J.** Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- **K.** For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- **L.** Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- **M.** Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- **N.** Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- **O.** Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.
You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.
You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.
The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.
The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.
You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.
If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.
If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.
Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME.
>
> Permission is granted to copy, distribute and/or modify this document
>
> under the terms of the GNU Free Documentation License, Version 1.2
>
> or any later version published by the Free Software Foundation;
>
> with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
>
> A copy of the license is included in the section entitled
>
> "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the
>
> Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.
If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.