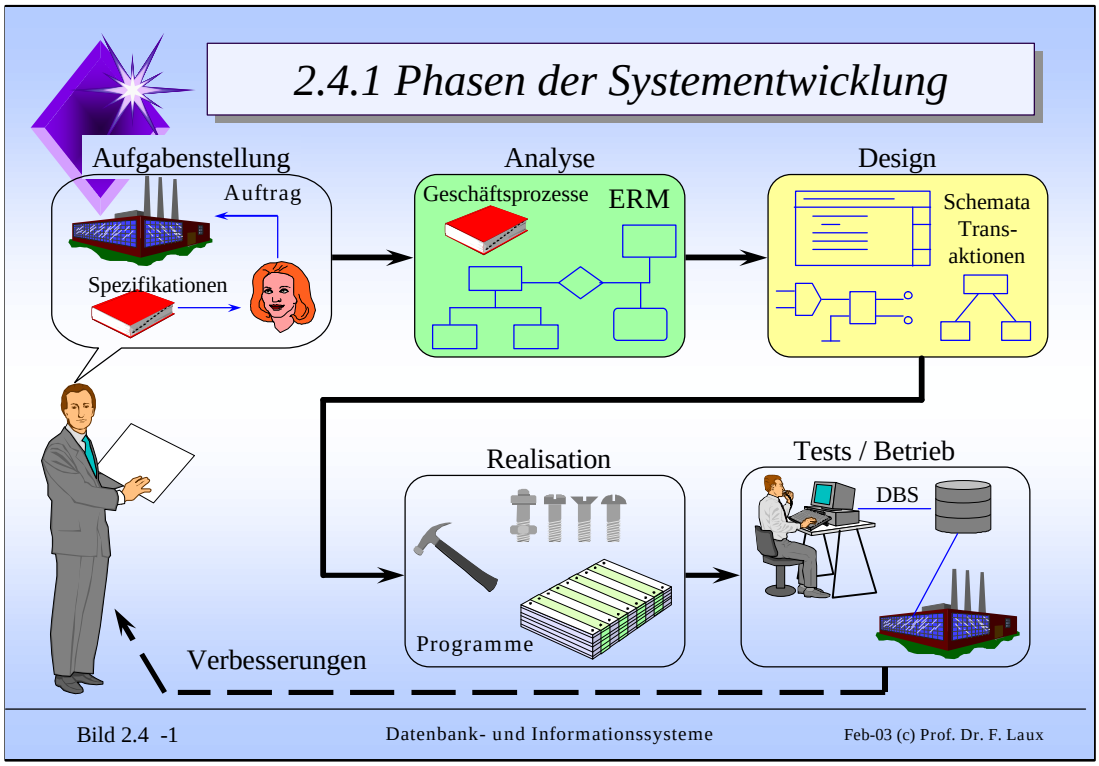



Wie bei allen großen Projekten durchläuft auch die Entwicklung eines Informationssystems mehrere Phasen. Wir betrachten vor allem die Entstehungsphasen. Obwohl wir die Phasen nicht als streng sequentiellen Prozeß auffassen wollen, lassen sich die Entwicklungschritte in einer natürlichen Reihenfolge beschreiben:

Die Planungsphase beginnt mit der *Aufgabenstellung* und der Anforderungsspezifikation, gefolgt von der *Analyse*. Während dieser Phase konzentriert sich die Entwicklung auf eine fachliche, implementierungsunabhängige, formale Spezifikation und Modellierung der Aufgabe. Das Design ist durch die Festlegung einer plattformspezifischen Architektur und durch Implementierungsdetails charakterisiert. Die eigentliche Entwicklung wird durch die Realisierung (Programmierung) abgeschlossen. Nach den Abnahmetests folgt schließlich die Einführung und der Betrieb. Wenn während der Entwicklungs- oder Betriebsphase neue Anforderungen oder Fehler auftauchen, können die Entwickler erneut in die Planungsphase eintreten. Weil der Entwicklungsprozeß häufig sehr komplex ist, werden die Phasen meistens mehrfach durchlaufen werden, deshalb spricht man auch von einem *Entwicklungszyklus*.

Ein sequentielles Vorgehen hat folgende Nachteile:

• geringe Flexibilität bezüglich neuer Anforderungen

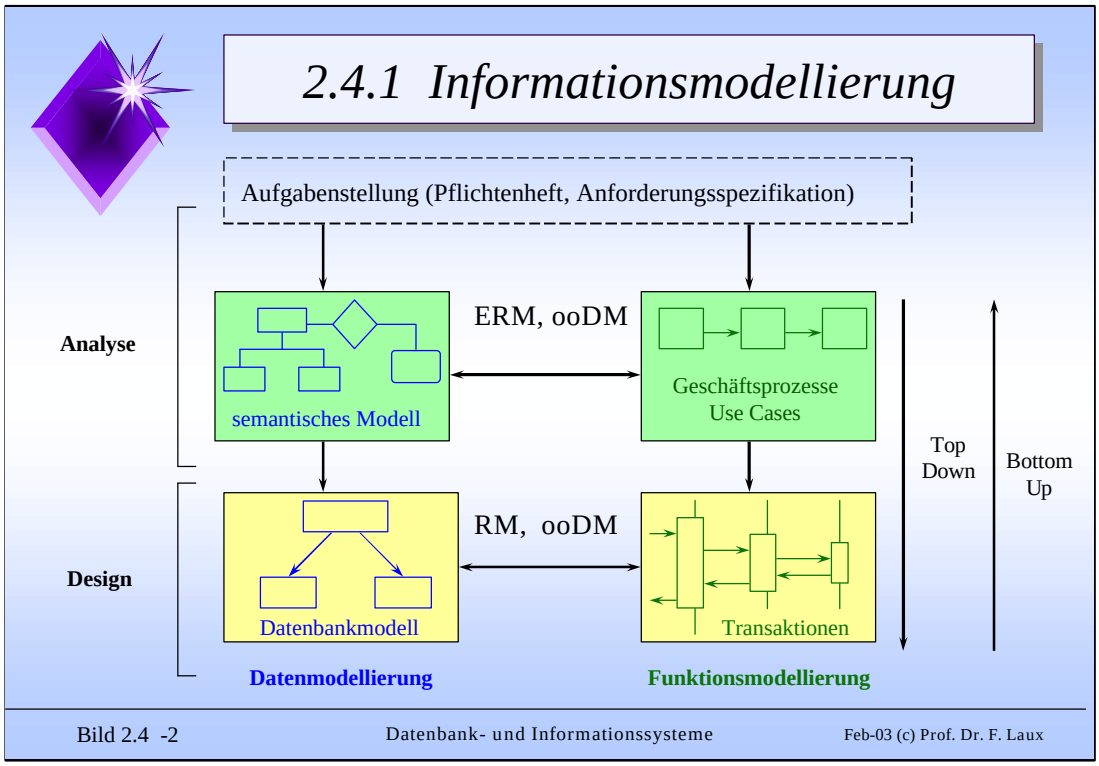
• späte Realisierung, dadurch späte Beurteilung der Lösung

Vorteil:

• gut strukturierte Software (architekturbetont)

Anstelle einer sequentiellen, werden zunehmend iterative Vorgehensweisen eingesetzt. Ein typischer Vertreter ist die evolutionäre Entwicklung. Der Entwickungszyklus schrumpft zu Stunden oder Tagen in denen ein Prototyp entsteht. Mit dessen Hilfe werden kann die Anforderungen weiterentwickelt.

ÜA: Nennen Sie Vor- und Nachteile einer Entwicklung mit Prototypen.



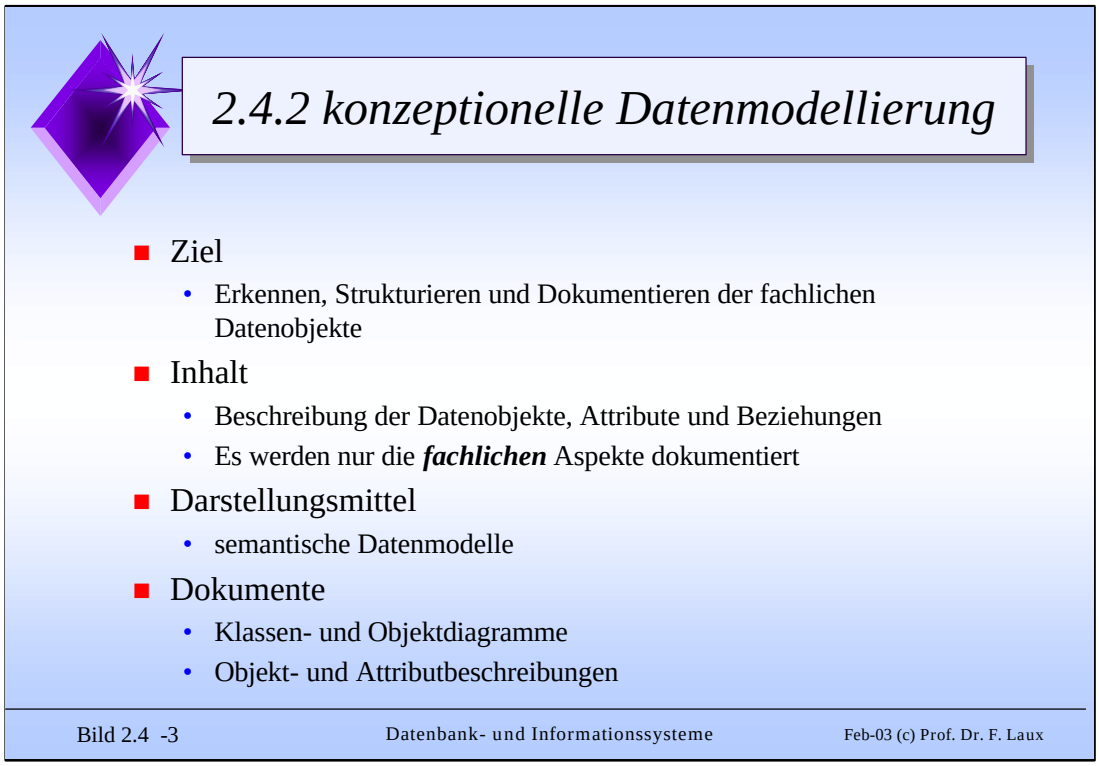

Die erste Phase, die *Aufgabenstellung*, liefert als Ergebnis ein *Pflichtenheft* (*Anforderungsspezifikation*), das die Basis für die nachfolgenden Phasen darstellt. Eine vollständige Spezifikation der Anforderungen ist ein schwieriger und langwieriger Prozess. Es existieren viele Vorgehensmodelle, die versuchen diese wichtige Phase zu unterstützen. (siehe Vorlesung **Software Engineering**)

Betrachten wir nun die Analyse- und Designphase etwas genauer, da hier die fachlichen Anforderungen durch Modelle formalisiert und an technische Voraussetzungen angepasst werden. In beiden müssen Struktur und Funktion (d.h. Verhalten) des Systems analysiert und modelliert werden. Bei einer objektorientierten Modellbildung werden beide Aspekte zusammen entwickelt. Trotzdem werden sie aus Gründen der Übersichtlichkeit meistens getrennt dargestellt.

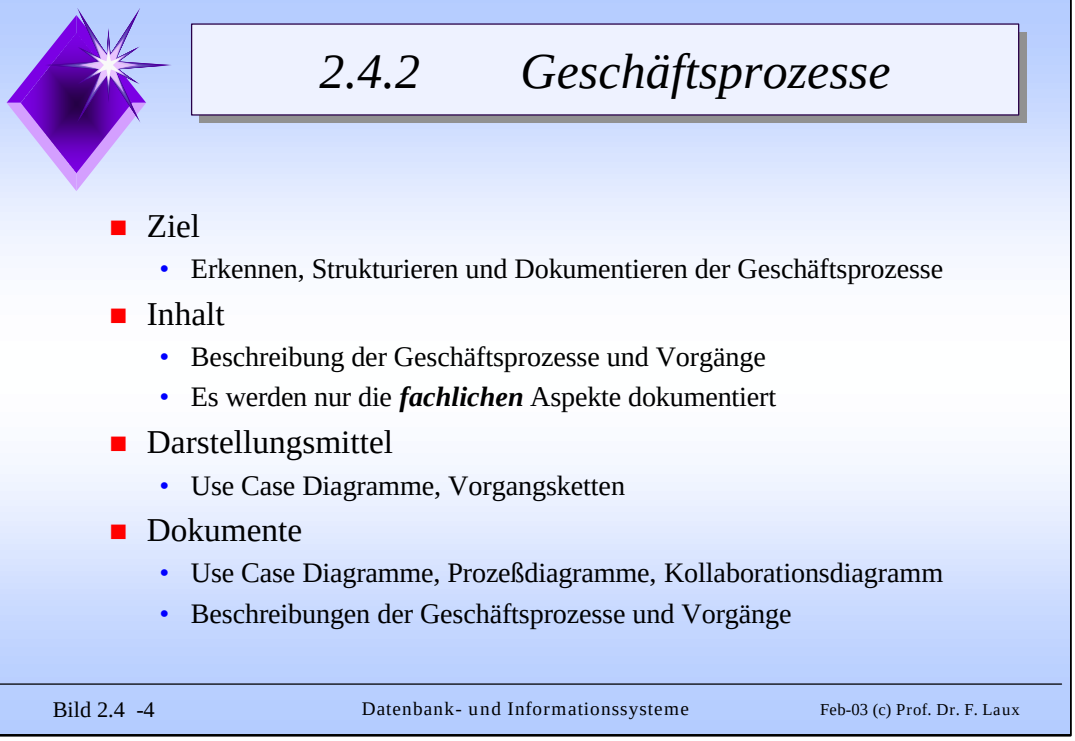
In der *Analyse* werden die Anforderungen durch Datenmodelle und Use Cases (Vorgänge, Geschäftsprozesse) formalisiert. Es werden nur fachliche Aspekte modelliert.

Ausgehend von den fachlichen Modellen werden im *Design* diese verfeinert (z.B. normalisiert, durch Algorithmen präzisiert), an die Implementierungsplattform (z.B. Betriebs- und Datenbanksystem) angepasst und durch geeignete Benutzer- und Systemschnittstellen erweitert.

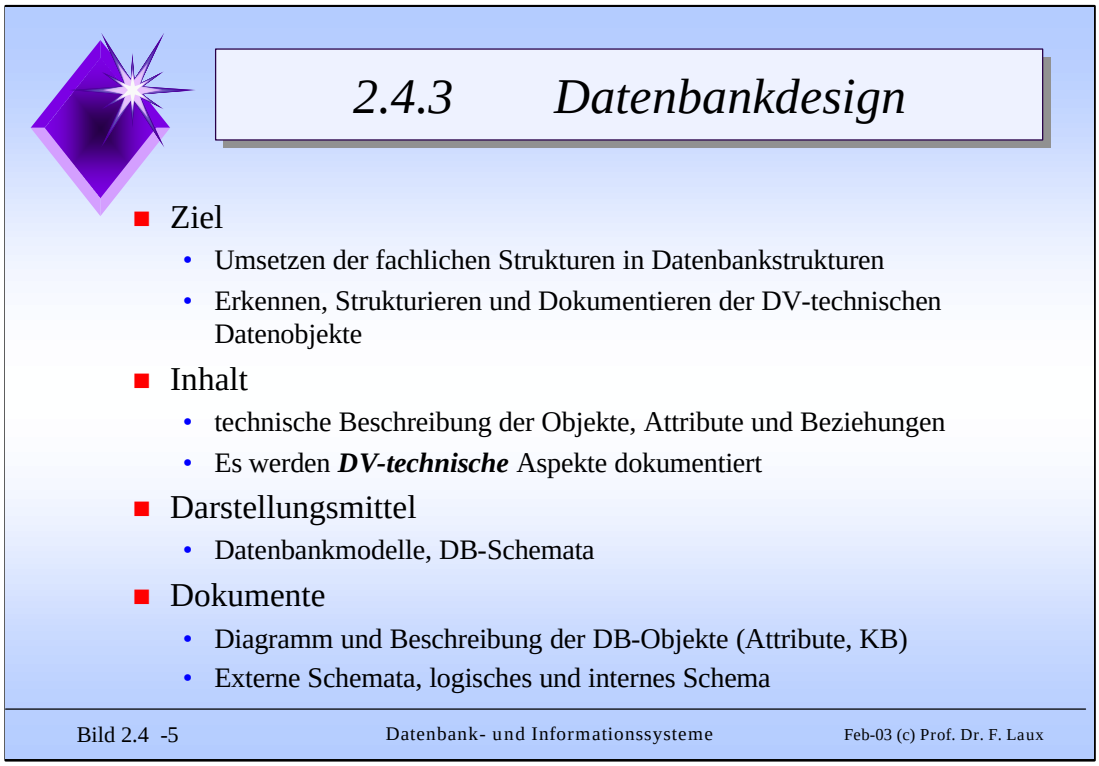
Wenn das System zuerst nur grob gegliedert (z.B. in Subsysteme) und nach und nach immer feiner untersucht wird, spricht man von einer '*Top-Down*' Vorgehensweise. Werden jedoch zuerst die Detailfakten (z.B. Variablen, Elementarfunktionen) erfaßt, anschließend gruppiert und klassifiziert (z.B. Module, Klassen etc gebildet), dann liegt ein '*Bottom-Up*' Vorgehen vor.




Bei den Zielsetzungen der *semantischen Datenmodellierung* geht es in erster Linie um die Strukturierung der Daten aus **fachlicher Sicht**. Die Darstellungsmittel sind demzufolge semantische Datenmodelle wie *Entity-Relationship-Modell*, *Klassenhierarchie* und *Objektdiagramme*, mit denen die Daten und ihre Beziehungsstruktur graphisch dargestellt werden kann. Textliche Beschreibungen der Objekte und Attribute ergänzen die Diagramme.




Bei der *Modellierung der Geschäftsprozesse* werden Vorgänge, Ereignisse (Actor) und Prozesse aus **fachlicher Sicht** untersucht. Als Darstellungsmittel dienen Prozeß- und Vorgangsmodelle wie *Use-Case Diagramme* und *Vorgangsketten*, mit denen die Prozesse, ihre auslösenden Ereignisse und ihre Abfolge graphisch dargestellt werden können. Textliche Beschreibungen der Prozesse und Bedingungen für Vorgänge ergänzen die Diagramme.




Bei der Zielsetzung für das *Datenbankdesign* geht es in erster Linie um die Anpassung der Daten an das Datenbankmodell und Optimierungen aus **technischer Sicht**. Die Darstellungsmittel sind graphische Datenbankmodelle wie *Bachman-Diagramme (relationale Datenbank), Klassenhierarchie* und *Objektdiagramme* (oo Datenbank), mit denen die Daten und ihre Beziehungsstruktur graphisch und textuell dargestellt werden kann. Bei oo Diagrammen werden häufig auch die Namen und Schnittstellen der Methoden (Transaktionen) mit aufgenommen (siehe auch Bild 2.2-5). Die formale textliche Definition der Datenbank (DB-Schemata) kann direkt zur Erstellung der Datenbank verwendet werden.

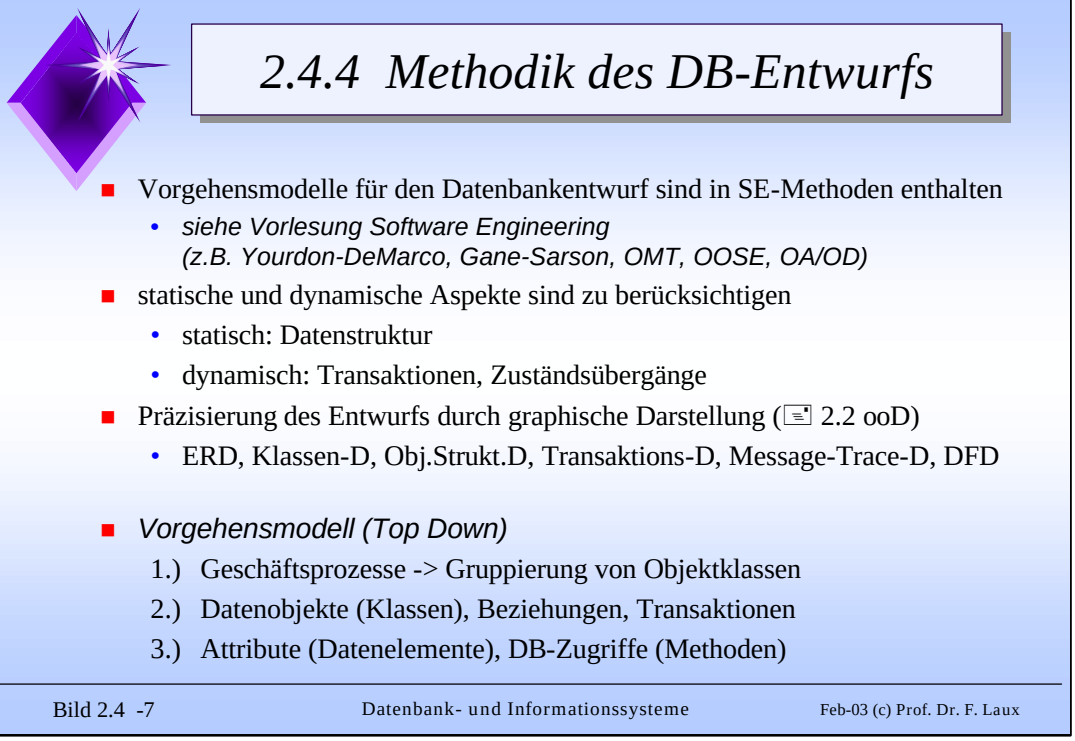
## 2.4.3 Transaktionen

- Ziel
  - Erkennen und Dokumentieren der DB-Transaktionen
- Inhalt
  - Beschreibung der DB-Transaktionen und Funktionen (Methoden)
  - Es werden ***DV-technische*** Aspekte dokumentiert
- Darstellungsmittel
  - Transaktions- bzw. Message-Trace (MT)-Diagramme
- Dokumente
  - Transaktionsbeschreibungen, Transaktions-Diagramme
  - Beschreibung der DB-Operationen (Methoden) inkl. ihrer Vor- und Nachbedingungen

Bild 2.4 -6 Datenbank- und Informationssysteme Feb-03 (c) Prof. Dr. F. Laux

Bei der *Modellierung der Transaktionen* werden Datenbankoperationen (Methoden) und ihre Konsistenzbedingungen (Pre- und Postconditions) aus **technischer Sicht** untersucht. Als Darstellungsmittel dienen Transaktions- und Zustandsübergangsdiagramme wie *Message-Trace*, *Kollaborationsdiagramme* und *Harel Diagramme* (um Operationen erweiterte Zustandsdiagramme), mit denen die Prozesse, ihre auslösenden Ereignisse und ihre Abfolge graphisch dargestellt werden kann. Textliche Beschreibungen der Prozesse und Bedingungen für Zustandsübergänge und Vorgänge können die Diagramme ergänzen.

Die Transaktionsmodelle können in Stored Procedures (rel. Datenbanken) oder Methoden (oo Datenbanken) umgesetzt werden.




Als Vorgehensmodelle zum Datenbankentwurf können grundsätzlich alle Software Engineering Methoden

•*Yourdon-DeMarco's SA (structured Analysis)-> Entity-Relationship Modell,*

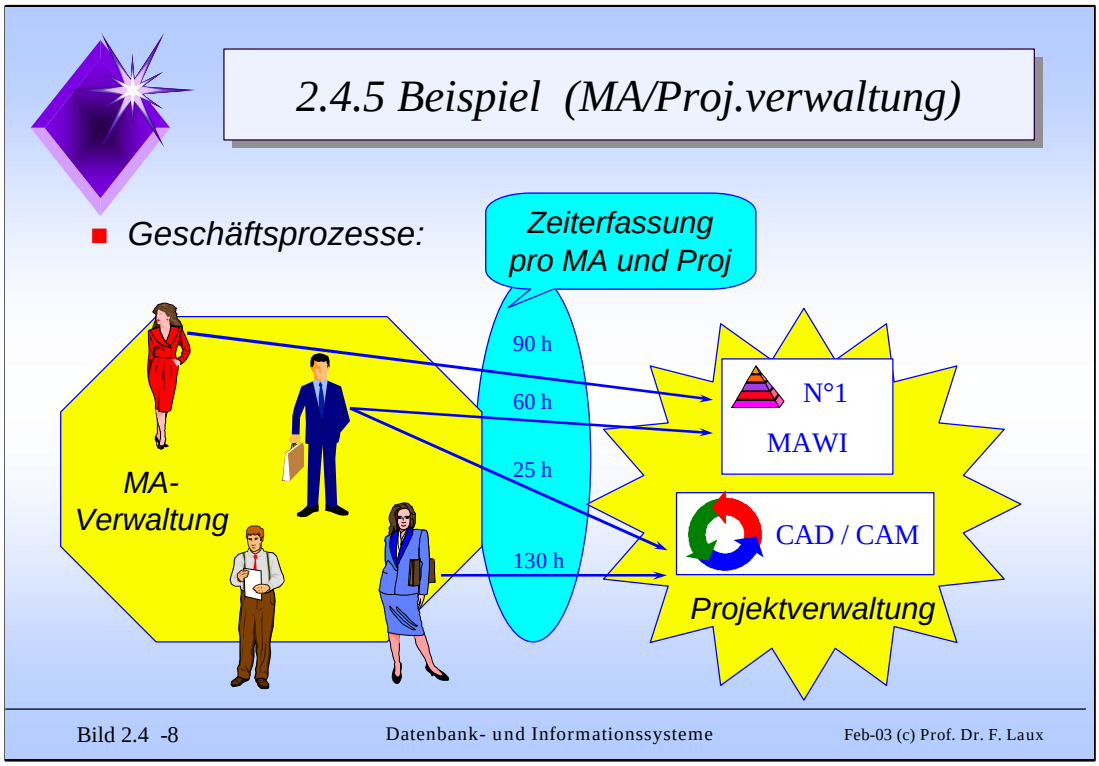
•*Rumbaugh's OMT (Object Modeling Technique) bzw. der Rational Process -> UML,*

•*Jacobson's OOSE (Object oriented Software Engineering) -> Use Cases,*

•*Coad-Yourdon's OA/OD (objectoriented Analysis and Design) -> ooM*

verwendet werden, die das gewünschte Datenbankmodell unterstützen. Die Schwierigkeit liegt trotz der Vielzahl der sich anbietenden Modelle nicht so sehr in der Auswahl, sondern in ihrer richtigen Anwendung. Dies ist jedoch von jedem einzelnen Projektmitarbeiter abhängig. Auch wenn ein Mitarbeiter keine Methode benennen kann oder will, nach der er vorgeht, so hat der doch implizit eine Vorgehensstrategie (die gelegentlich auch wechselt). Die einmal angewöhnte Arbeitsweise von Personen ist nur schwer zu beeinflussen; dies macht den erfolgreichen Einsatz von Vorgehensmodellen so schwierig. Außerdem lassen viele Modelle Interpretationsspielraum. Auf jeden Fall ist der Datenbankentwurf ein kreativer Vorgang, so daß auch bei einer regelgerechten Vorgehensweise ein "gutes" Design nicht garantiert werden kann.

Das in der Folie empfohlenen Top-Down Vorgehen ist demzufolge auch nur grob skizziert um jedem Entwickler genügend Spielraum für seine Kreativität zu lassen. Wichtig ist jedoch, daß die Semantik der verwendeten Diagramme für alle klar und eindeutig ist. Dieses Modell ist auch für ein evolutionäres, iteratives Vorgehen geeignet, wenn die einzelnen Schritte zunächst nur unvollständig durchgeführt werden, so daß bereits nach kurzer Zeit ein Prototyp erstellt werden kann.




Im folgenden soll exemplarisch eine Datenbank für die oben gezeigte Aufgabe (vgl. auch 2.1-5) modelliert werden. Obwohl das Beispiel sehr einfach gehalten ist, können aus Platzgründen nicht alle Aspekte vollständig untersucht werden. Jedoch wird für jeden Aufgabentyp wenigstens ein Beispiel entwickelt.

Betrachten wir ein Unternehmen, das mit seinen Mitarbeitern Projekte durchführt. Die Aufgabenstellung umfaßt die folgenden Geschäftsprozesse:

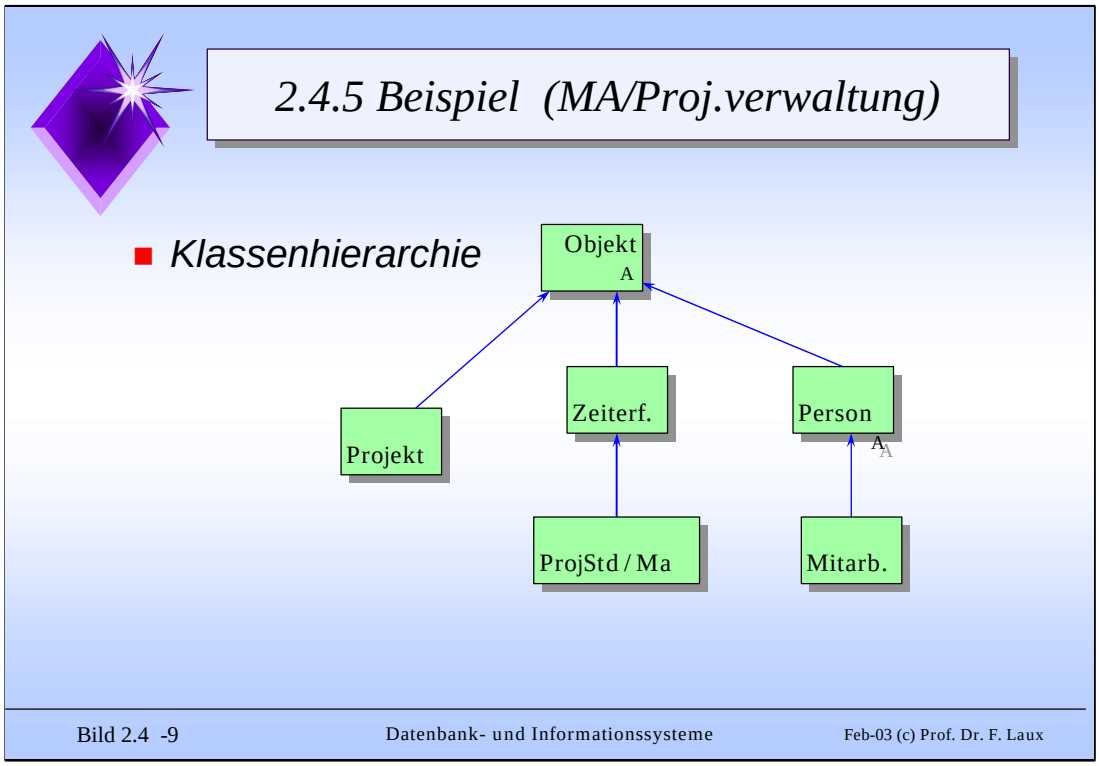
- Projektverwaltung
- Mitarbeiterverwaltung
- Stundenerfassung pro Mitarbeiter und Projekt

Anforderungsspezifikation:

Geschäftsprozeß: Stundenerfassung

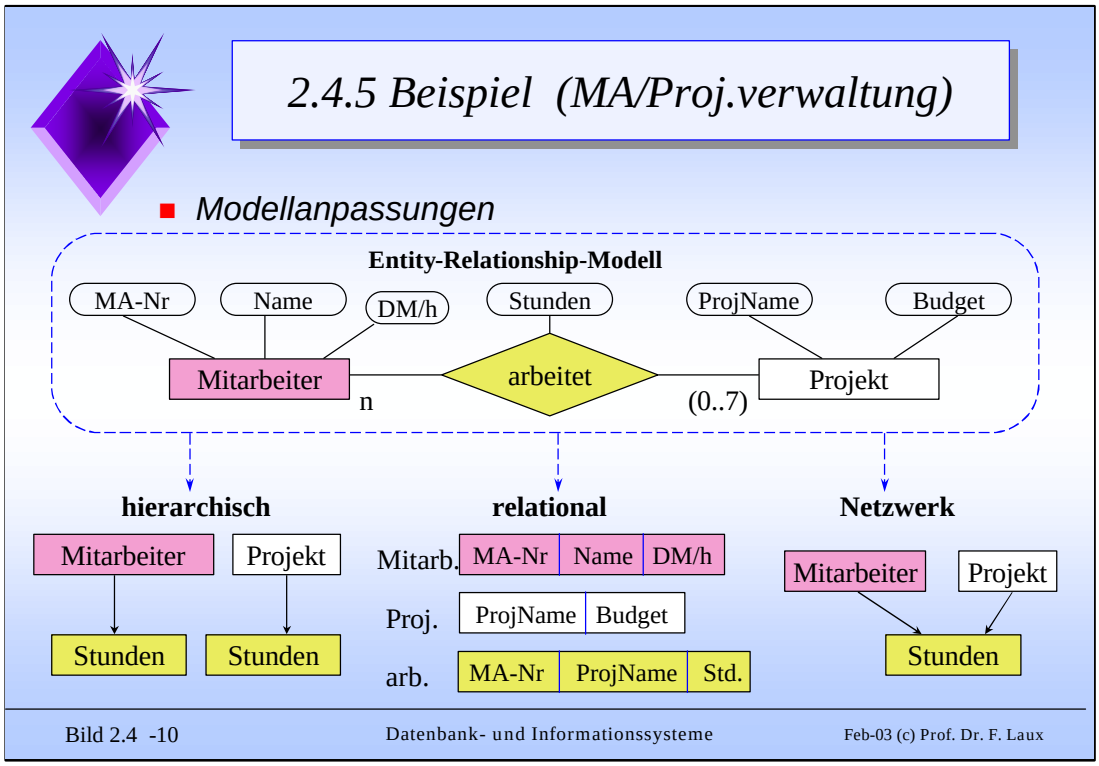
Jeder Mitarbeiter kann an bis zu 7 Projekten arbeiten. Es wird jeweils nur die aktuelle Gesamtstundenzahl pro Mitarbeiter und Projekt registriert (Projektabrechnung). Ein Projekt erhält ein Budget, das durch die geleistete Arbeitskosten (:= Arbeitskosten * Stundensatz/MA) belastet wird. Die Arbeitszeiterfassung der MA für das Personalbüro ist nicht Gegenstand der Spezifikation. Wenn ein neues Projekt angelegt wird, muß wenigstens der Projektleiter zugewiesen werden. Die Stunden werden mit einer Auflösung von 6 Minuten (1/10 h) erfaßt.

ÜA: Spezifizieren Sie analog die restlichen Geschäftsprozesse




Aus der Aufgabenstellung und den Spezifikationen für die Geschäftsprozesse lassen sich die Klassen Mitarbeiter und Projekt sofort erkennen. Der Geschäftsprozess der Zeiterfassung erhält sinnvoller weise ebenfalls eine Klasse, denn die zu erfassenden Zeiten sind sowohl projekt- als auch mitarbeiterspezifisch. Deshalb können die Zeitdaten weder dem Projekt noch dem Mitarbeiter allein zugeordnet werden. Außerdem scheint die Zeiterfassung auch als allgemeines Konzept brauchbar zu sein, deshalb wurde eine eigene Hierarchie für die Klassen Zeiterfassung und Projektstd./Mitarbeiter gewählt. Anwendungsobjekte zu finden ist ein Prozess, der Erfahrung erfordert, denn es wären sicherlich auch andere Klassen denkbar, die unserer Aufgabenstellung angemessen wären. Neben der Grundklasse (Objekt) sind auch Person und Zeiterfassung abstrakte Objekte. Es ist denkbar, daß neben Mitarbeitern später auch noch Kunden und andere Personen hinzukommen. Ebenso sind verschiedene Klassen von Zeiterfassungen (z.B. Maschinenzeiten, Reisezeiten, etc) für ein Unternehmen relevant, so daß hier auch die abstrakte Klasse Zeiterfassung gerechtfertigt ist.

Die softwaretechnischen Klassen werden in der Analyse nicht betrachtet. Wir werden auch später, wenn wir unser Modell verfeinern, nicht darauf eingehen, da diese Klassen von der verwendeten Plattform abhängen. Beispielsweise benötigen wir für eine graphische Benutzeroberfläche solche Klassen wie z.B. Fenster, Zeichenfläche, Ereignis, Menü, etc.

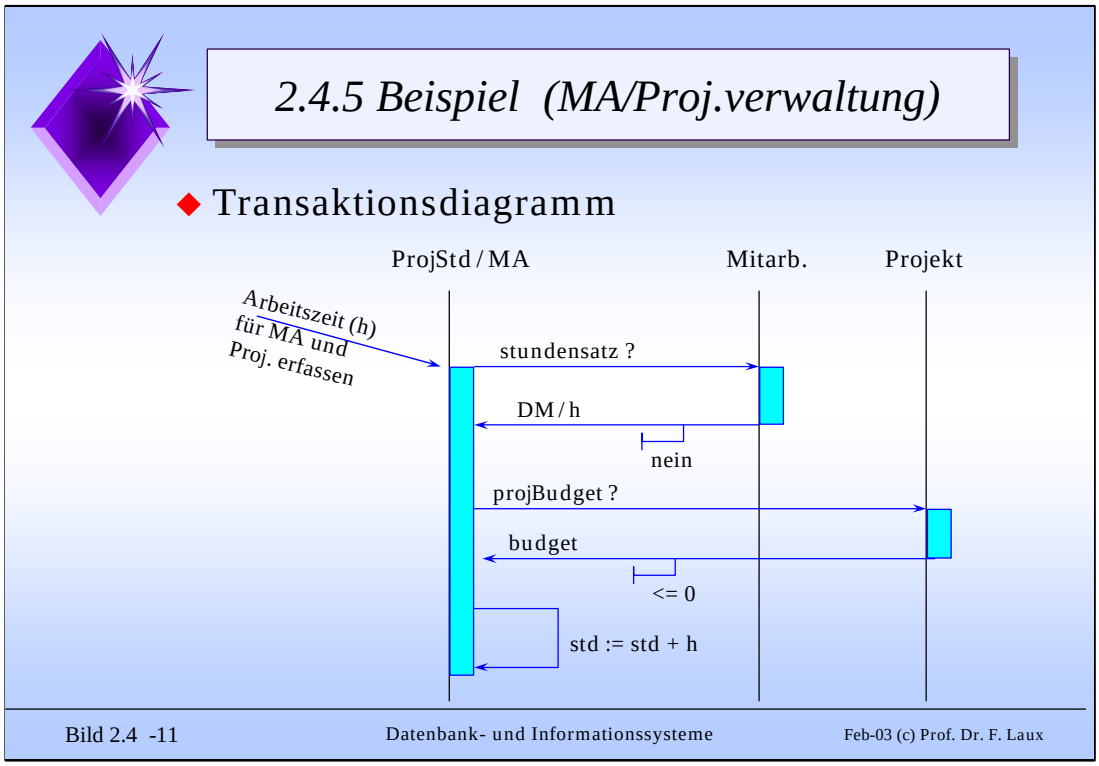



**Exkurs:**

Für das Design ist unser konzeptionelles Objektmodell (bisher haben wir nur die Klassenhierarchie entwickelt) dem eingesetzten Datenbankmodell anzupassen, falls wir keine objektorientierte Datenbank einsetzen.

Wir wollen dies am Beispiel eines Entity-Relationship Modells demonstrieren, das in die drei klassischen Datenbankmodelle transformiert wird. Das ER-Modell spiegelt die Aufgabenstellung von 2.4-8 wieder. Das Modell unterstützt keine Vererbung, jedoch können Spezialisierungen/Generalisierungen (vgl. 2.1-9+10) ausgedrückt werden. Von dieser Möglichkeit wird hier jedoch kein Gebrauch gemacht. Falls dies gewünscht würde, müssten die Daten für *Mitarbeiter* und *Projektstunden* (*arbeitet*) entsprechend der Klassenhierarchie in zwei Datenbankobjekte aufgeteilt werden. Dies soll für das relationale Modell am Beispiel *Mitarbeiter* gezeigt werden: Anstatt einer Relation *Mitarbeiter* sind zwei Relationen (*Person, Mitarbeiter*) notwendig, wobei die Relation *Person* die allgemeinen Personendaten wie Name, Adresse, etc enthält und die Relation *Mitarbeiter* nur die Daten, die für einen Mitarbeiter spezifisch sind (MA-Nr, Abteilung, Position, Gehalt, etc).

Da unsere Entitäten (Objekte) relativ einfach strukturiert sind, lassen sie sich eins-zu-eins auf Relationen abbilden. Ähnliches gilt für den Satzaufbau der hierarchischen und Netzwerkdatenbank. Die Unterschiede liegen in der Struktur der Sätze zueinander: beim hierarchischen Modell sind nur 1:n-Beziehungen erlaubt, beim Netzwerkmodell kann ein Satz mehrere übergeordnete Sätze (Owner) haben. Die Struktur wird über Satzverweise (Pointer) realisiert. Im Gegensatz hierzu sind die Beziehungen zwischen Relationen nur über Dateninhalte und nur während einer Verbundabfrage (Join) existent.




Die dargestellte Transaktion *‚Arbeitszeit erfassen‘* zeigt, welche Objekte und welche Nachrichten dafür notwendig sind. Wir setzen voraus, daß bereits ein Objekt existiert, das Stunden für einen Mitarbeiter und ein Projekt aufnehmen kann. Dies bedeutet, daß der Mitarbeiter einem Projekt zugewiesen wurde.

Von der Benutzerschnittstelle wird eine Nachricht *Arbeitszeit (h) für MA und Projekt erfassen* an das Objekt *ProjStd/MA* geschickt, dieses prüft, welcher Stundensatz für den Mitarbeiter eingetragen ist und ob das Budget für das Projekt ausreichend ist. Diese Prüfungen erfolgen durch Nachrichten an die zuständigen Objekte. Falls eine dieser Prüfungen negativ ausgeht (z.B. MA in Urlaub oder krank gemeldet, Budget unterschritten), wird die Transaktion abgebrochen. Ansonsten werden die gemeldeten Stunden zu den bereits vorhanden hinzuaddiert.

## 2.4.5 Beispiel (MA/Proj.verwaltung)

- Objektdiagramm:

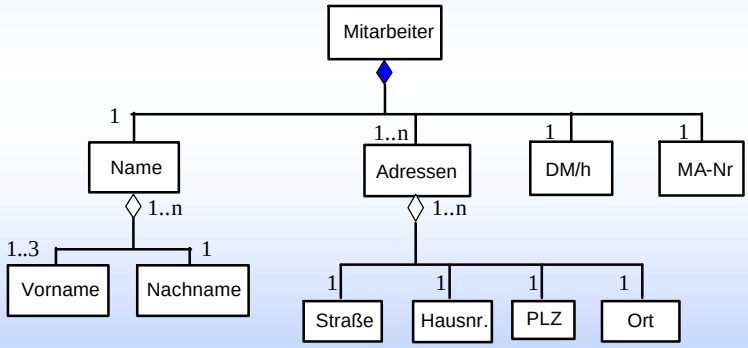
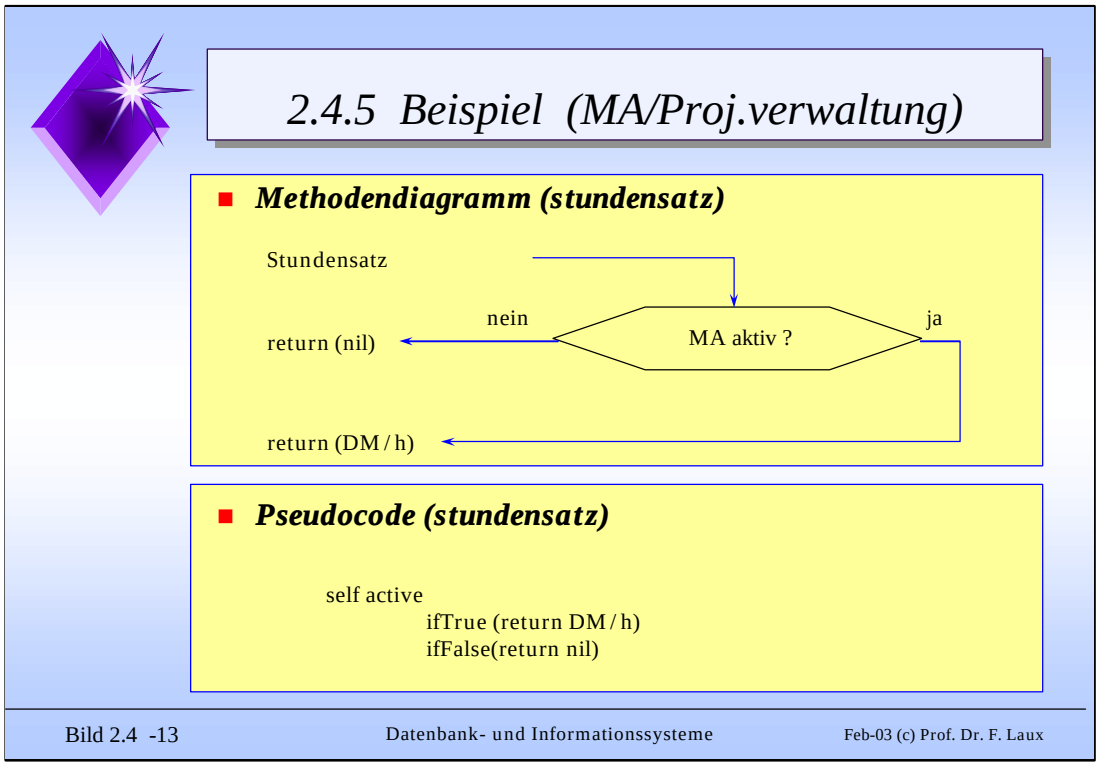


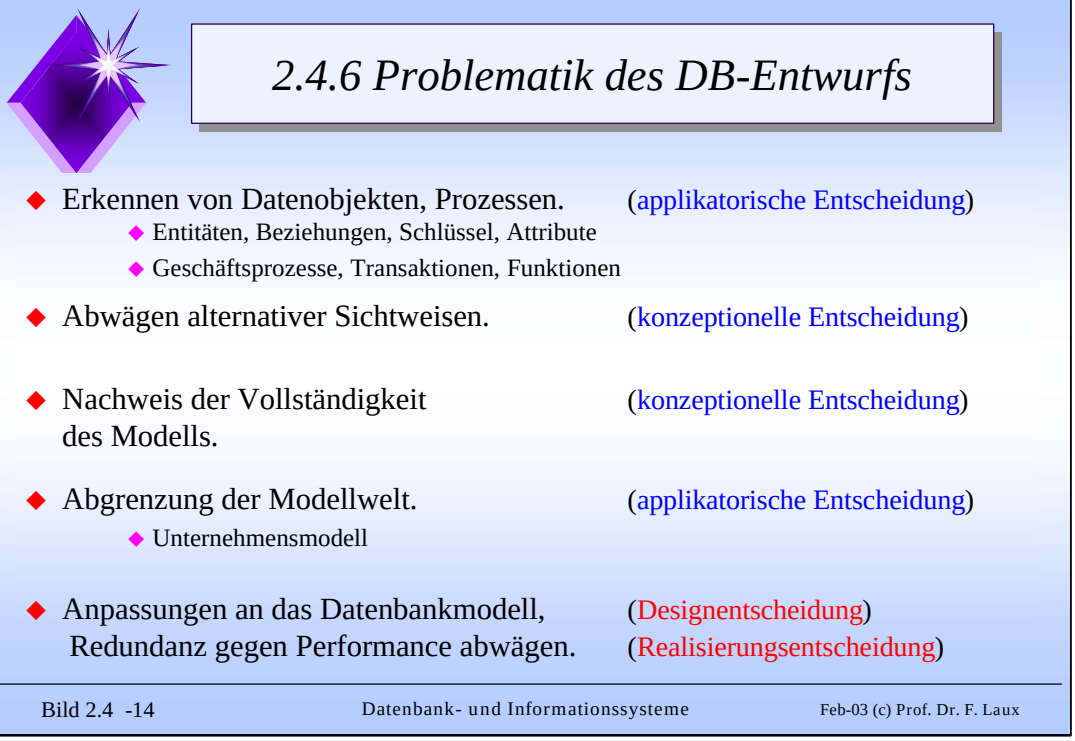

Bild 2.4 -12 Datenbank- und Informationssysteme Feb-03 (c) Prof. Dr. F. Laux

Wir verfeinern die Objektstruktur indem die Elemente (Objekte/Daten) unserer Klassen graphisch verfeinert werden. Ein Mitarbeiterobjekt ist eine Komposition (abhängige Komponenten) aus 4 Objekten, dem *Namen*, *Adressen* (mindestens eine), *Stundensatz* und der *MA-Nr*. Während *MA-Nr* und *Stundensatz* einfach numerische Objekte (Datentypen) sind, bestehen *Name* und *Adresse* aus weiteren Komponenten. Da diese hier nicht weiter spezifiziert sind, nehmen wir einfache (vordefinierte) Datentypen (z.B. String) an. Denkbar wären natürlich auch spezifische Klassen wie z.B. *Vorname* (nur Strings, die einen sinnvollen Voramen darstellen) oder *PLZ* (nur PLZs der BRD). Damit wären implizit schon Konsistenzbedingungen gegeben.




In einem weiteren Detailierungsschritt wollen wir die Methode *stundensatz* der Klasse *Mitarbeiter* genauer betrachten. Das Empfängerobjekt (ein Mitarbeiter; hier *self* genannt, da er der Empfänger der Nachricht und ausführendes Objekt der Methode *stundensatz* ist) prüft, ob der Mitarbeiter aktiv ist. Wie dies exakt gemacht wird, ist nicht dargestellt. Es könnte eine boolsche Variable *aktiv* geben, die dies anzeigt oder es könnten Krankmeldung, Urlaub etc. im Mitarbeiterobjekt gespeichert sein, anhand dessen die Methode den Arbeitsstatus ermitteln könnte. Die Implementierung ist jedoch Sache des Empfänger und hat den Sender nicht zu interessieren.

Wenn der Mitarbeiterstatus 'aktiv' ist, wird der *Stundensatz* (DM/h) zurückgegeben, sonst wird durch *nil* (Nichts) angezeigt, daß der Mitarbeiter z.Zt. nicht arbeitet, also auch keine Stunden erfassen kann.




Im Verlauf eines Datenbankmodellierungsprozesses sind eine Reihe von Entscheidungen zu treffen. Diesen Entscheidungen stehen gelegentlich nahezu gleichwertige Alternativen gegenüber. Damit wird die Entscheidung zu einem gewissen Grad willkürlich und erhält damit einen kreativen Charakter. Aus diesem Grund gilt diese Tätigkeiten als so reizvoll.

Erfahrung ist bei diesen Entwurfsentscheidungen natürlich hilfreich, denn gute Designer haben ein großes Repertoire an Entwurfsmuster (Musterlösungen) im Kopf, die sie geeignet abwandeln und anwenden können.

## *Literatur zu Kapitel 2*


- P.P. Chen
  "The Entity-Relationship Model - Toward a Unified View of Data"
  ACM Transactions on Database Systems Vol. 1, No 1, March 1976, S. 9 - 36
- A. Moos, G. Daues
  "Datenbank-Engineering", Vieweg Verl. 1997
- G. Booch, J Rumbaugh, I. Jacobson: Unified Modeling Language (UML) V. 1.3
  Rational Software Corp., http://www.rational.com/uml/index.jtmpl
- I. Jacobson, G. Booch, J. Rumbaugh
  "The Unified Software Development Process "
  Addison Wesley, 1999
- R.G.G. Cattel, Ed.
  "Object Databases: The ODMG-93 Standard", Morgan Kaufmann, 1993
- Heidi Grosch: Objektorientierte Datenmodelle, Diplomarbeit, FH Reutlingen 1993




Die Fertigstellung der Version 1.0 der UML im Januar 1997 bewirkte, daß sich viele Autoren mit dieser Sprache auseinandersetzen und Sekundärliteratur dazu erstellen:

(vgl. URL: http://www.awl.com/cseng/otseries)


Rainer Burkhard: UML - Unified Modeling Language, Addison Wesley, 1997

Martin Fowler; Kendall Scott: UML distilled, Addison-Wesley. 1997

Bernd Oestereich: Objektorientierte Softwareentwicklung mit der UML, 3. Auflage, Oldenbourg, 1997

Grady Booch, J. Rumbaugh, I. Jacobson: Unified Modeling Language User Guide, Addison-Wesley, 1998

Philippe Kruchten: The Rational Unified Process : An Introduction, Addison-Wesley, 1999


Eine schöne Einführung in verschiedene Methoden der Objektmodellierung stellt die Arbeit von H. Grosch dar. Für Kenner der Programmiersprache *Smalltalk* ist die Diplomarbeit von P. Grieshaber, Objektorientierter Entwurf eines graphischen Klassenbrowsers nach der OMT-Methode und Realisierung unter VisualAge für Smalltalk, 1997, zu empfehlen.