# Software Engineering

## Software-Entwurf

Prof. Dr. Peter Jüttner

Hochschule Deggendorf

Besonderer Dank gebührt
Prof. Dr. Jürgen Mottok
für die Überlassung
wesentlicher Teile des
Materials für Kapitel 5.2

# Inhalt

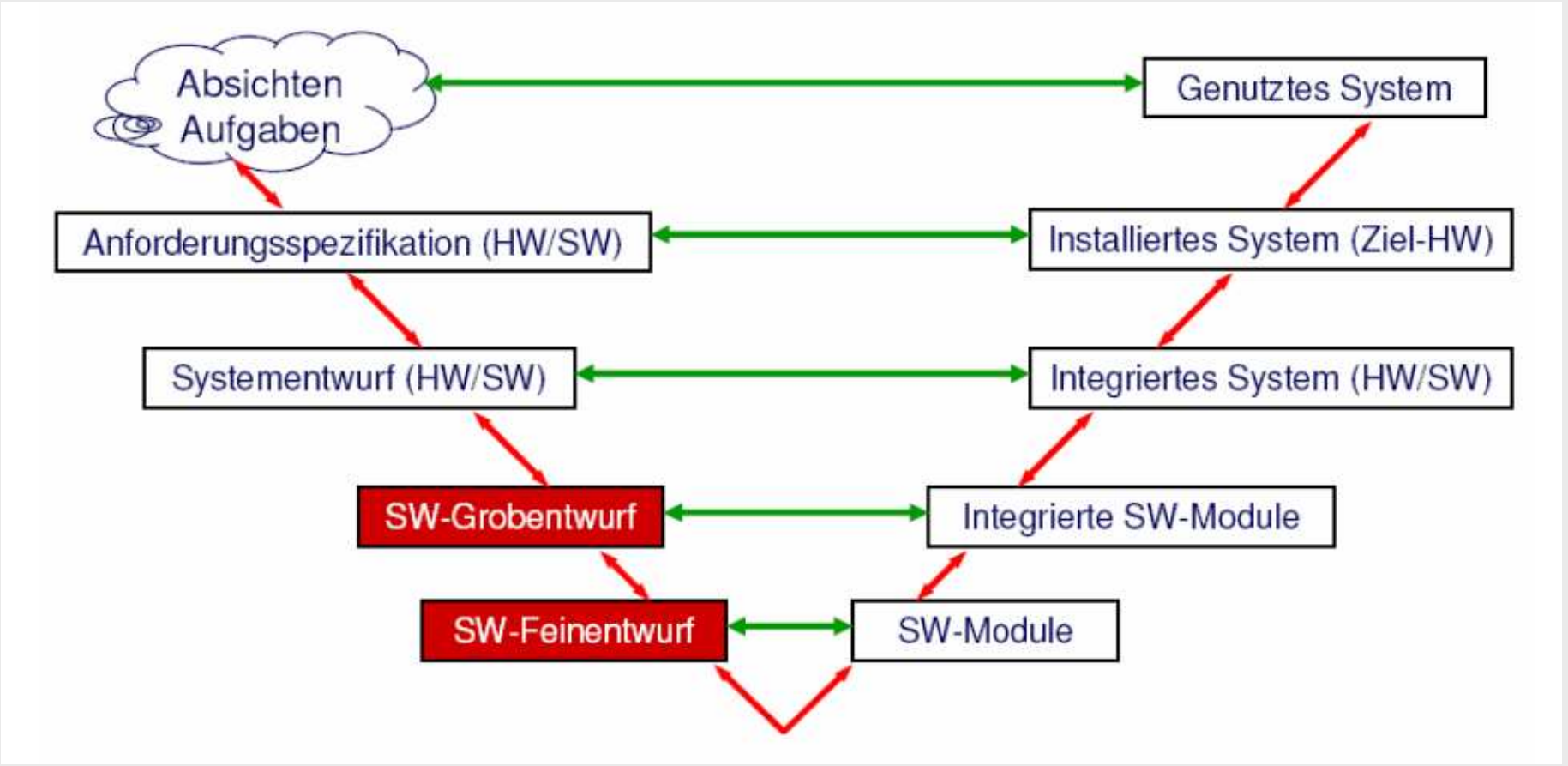
# Literatur zur Vertiefung

- Helmut Balzert
  **Lehrbuch der Software-Technik, Teil 1**
  Spektrum Akademischer Verlag, 2001

- Ian Sommerville
  **Software Engineering**
  Addison Wesley, 2001

- Bernd Kahlbrandt
  **Software-Engineering mit der UML**
  Springer-Verlag, 2001

# Bereich des V-Modells

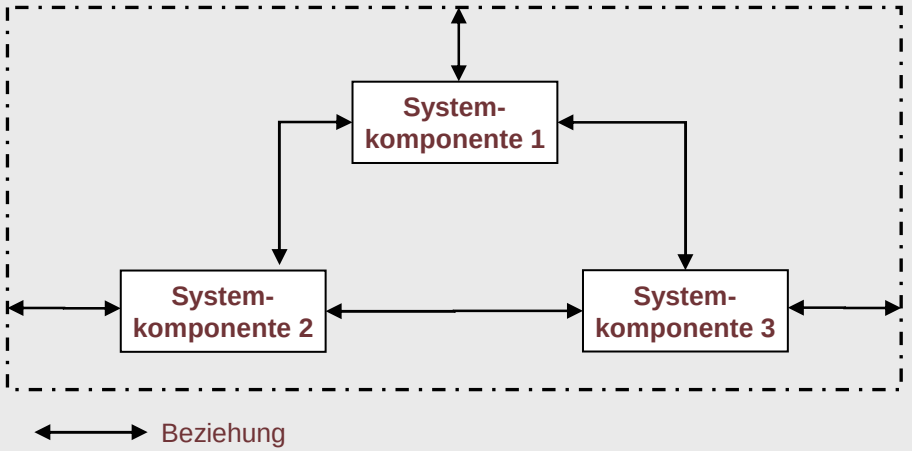


# Aufgaben der Entwurfsphase

- Aus dem Ziel des Software-Entwurfs ergeben sich die Aufgaben der Entwurfsphase:
  - Entwerfen einer Software-Architektur
    - Zerlegung des definierten Systems in Systemkomponenten
    - Strukturierung des Systems durch geeignete Anordnung der Systemkomponenten
    - Beschreibung der Beziehungen zwischen den Systemkomponenten
    - Festlegung der Schnittstellen, über die die Systemkomponenten miteinander kommunizieren
  - Spezifikation des Funktions- und Leistungsumfanges der Systemkomponenten
  - Detaillierung der Systemkomponenten durch Beschreibung ihrer Operationen

# Software-Architektur

- Eine Software-Architektur beschreibt die Struktur des Software-Systems durch Systemkomponenten und ihre Beziehungen untereinander.




- Eine Systemkomponente ist ein abgegrenzter Bereich eines Software-Systems. Sie dient als Baustein für die physikalische und logische Struktur einer Anwendung.
- Beispiele für Systemkomponenten sind Funktionen/Prozeduren, abstrakte Datentypen oder Klassen

# Inhalt

# Ziel des Software-Grobentwurfs

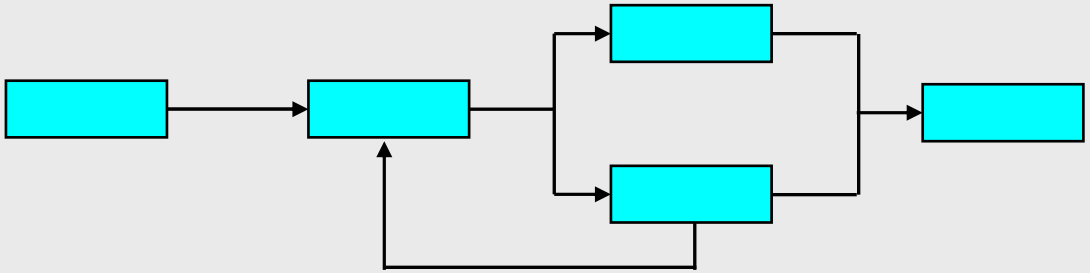
- Ziel des Software-Grobentwurfs ist es, für das zu entwerfende Produkt eine Software-Architektur zu erstellen,
  - die die funktionalen und nicht-funktionalen Produktanforderungen sowie
  - allgemeine und produktspezifische Qualitätsanforderungen erfüllt und
  - die Schnittstellen zur Umgebung versorgt

# Einzusetzende Prinzipien

- Prinzip der Zerlegung
  - Aufteilung des Systems in miteinander interagierende Bausteine
- Prinzip der Abstraktion
  - Identifikation eines Teilsystems mit der funktionalen Rolle, die es im Gesamtsystem zu übernehmen hat, zunächst ohne Berücksichtigung seiner Implementierungsdetails

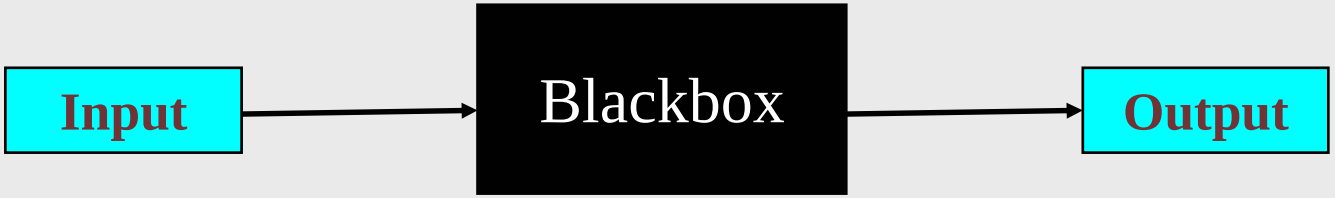
# Prinzip der Zerlegung

- Um ein System übersichtlicher und verständlicher zu machen, kann es in Teilsysteme unterteilt werden,
  - die über Relationen verbunden,
  - miteinander kommunizieren können,
  - und somit das Gesamtsystem widerspiegeln



# Prinzip der Abstraktion

- Unter Abstraktion versteht man eine Verallgemeinerung, das Absehen vom Besonderen und Einzelnen.
- Abstraktion ist das Gegenteil von Konkretisierung.




- Da viele zu modellierende Systeme sehr komplex und somit sehr unverständlich sind, verwendet man das Prinzip der Abstraktion dazu, um eine möglichst vereinfachte Sicht des Systems zu gewinnen.

# Top-Down- und Bottom-Up-Entwurf

- Top-Down Vorgehensweise
  - Spezialisierung
  - schrittweise Verfeinerung
  - das Gesamtsystem wird schrittweise in Teilsysteme zerlegt
- Bottom-Up Vorgehensweise
  - Generalisierung
  - schrittweise Verallgemeinerung
  - das Gesamtsystem wird schrittweise aus Teilsystemen zusammengesetzt
- Beide Vorgehensweisen sind kombinierbar.

# Ergebnis des Grobentwurfsprozesses

Als Ergebnis des Grobentwurfsprozesses erhält man einen Produkt-Entwurf, der aus folgenden Teilen besteht

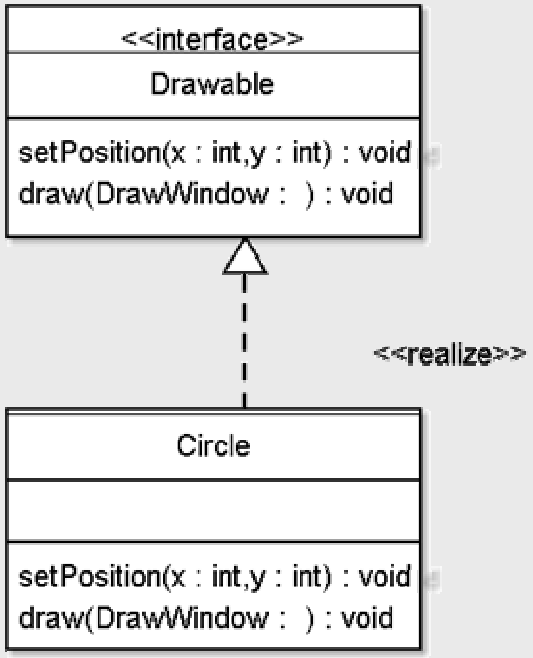
- Software-Architektur, d.h.
  - die strukturierte oder hierarchische Anordnung der Systemkomponenten und
  - ihre Beziehungen untereinander.
- Spezifikation jeder Systemkomponente, d.h. Festlegung von
  - Schnittstellen
  - Funktions- und
  - Leistungsumfang

# Schnittstellen von Komponenten

- Eine Schnittstelle einer Komponente ist eine Spezifikation für ihr externes Verhalten.
- Eine Schnittstelle spezifiziert insbesondere eine Menge von Operationen, gibt jedoch für diese Operationen keine Implementierung an.
- Die Spezifikation einer Schnittstelle kann enthalten
  - eine Liste von Operationen, die die Schnittstelle zur Verfügung stellt,
  - Zusicherungen an die Komponente, z.B.
    - Invarianten der Komponente
      z.B. Einschränkung an die Größe einer Datenstruktur
      („Liste hat höchstens n Elemente”)
    - Vor- und Nachbedingungen für jede einzelne Operation
      z.B. Stack: Falls [Anzahl der Elemente) = n ≥ 1 (Vorbedingung), dann [Anzahl der Elemente nach Aufruf von POP) = n - 1 (Nachbedingung)

# Schnittstellen: Darstellung in UML

- In der UML-Notation werden Schnittstellen für Klassen
- zur Unterscheidung von einer gewöhnlichen Klasse mit dem Stereotyp <<Interface>> gekennzeichnet
- als Rechtecke dargestellt mit
  - den Schnittstellen-Namen
  - der Deklaration der Schnittstellenoperationen
  - der annotierten Bedeutung der Operationen
  - evtl. mittels OCL annotierten Zusicherungen
- Eine Komponente, die die Schnittstelle realisiert, wird durch eine (gestrichelte) Realisierungskante mit der Schnittstelle verbunden



# Klassische Architekturmodelle

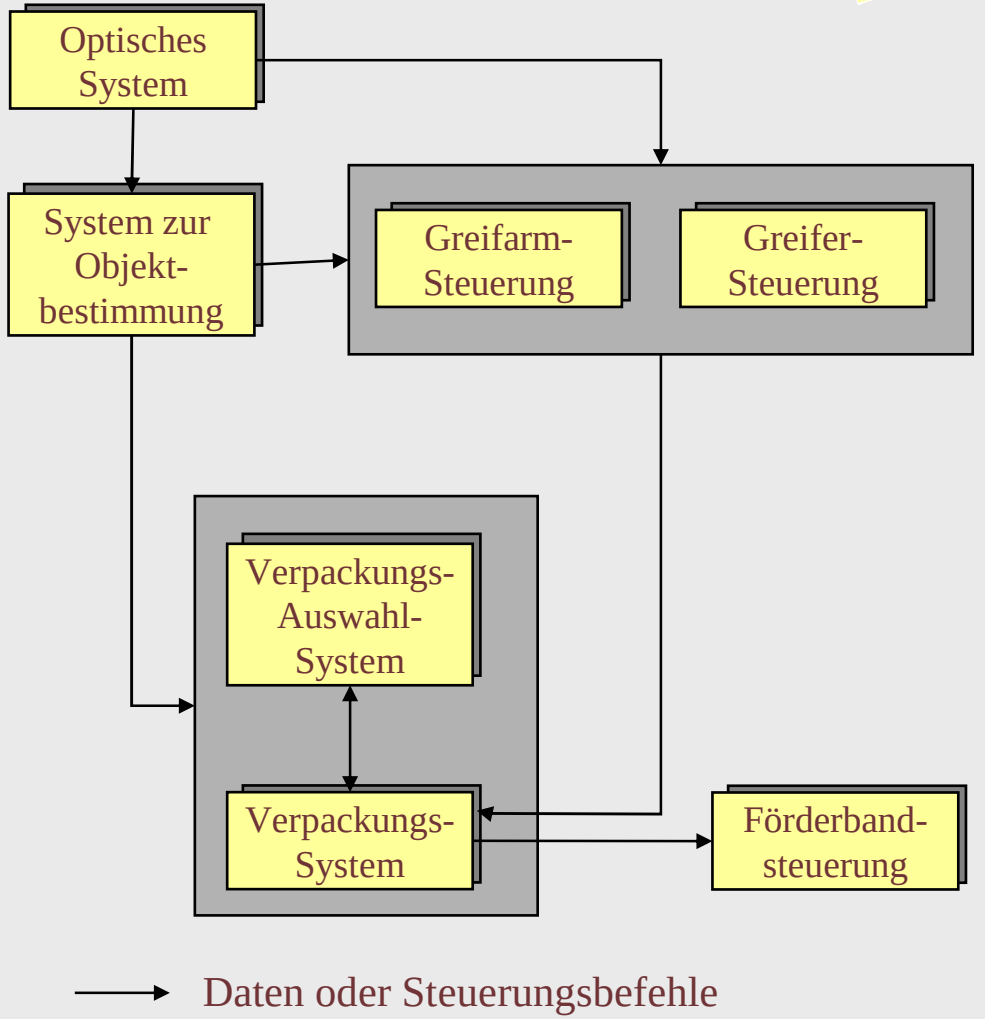
- Blockdiagramm
  - sehr allgemeine Darstellung, geeignet als allererster Ansatz
- Datenspeichermodell (= Datenbank-Schema)
  - geeignet speziell für den Einsatz globaler Datenbanken
- Schichtenmodell
  - hierarchische Anordnung zunehmend abstrahierter Sichten
- Client/Server-Modell
  - gemeinsamer Zugriff auf verteilte Dienste
- Verteiltes System
  - Verteilung des Softwaresystems über mehrere Rechner

# Blockdiagramm

- Stärkste Vereinfachung des Architekturentwurfs
  - jeder Block stellt ein Subsystem dar
  - Blöcke innerhalb von Blöcken bedeuten, dass ein Subsystem selbst wiederum aus Subsystemen aufgebaut ist.
  - Pfeile deuten an, dass Daten oder Steuerungsbefehle zwischen den Subsystemen in Richtung der Pfeile übertragen werden.
- In der ersten Phase des Architekturentwurfs verwendet, um das System in eine Anzahl miteinander zusammenarbeitender Subsysteme aufzuteilen.
  - gibt einen Überblick über die Systemstruktur
  - ist für die verschiedenen Entwickler, die in den Systementwicklungsprozess einbezogen sein können, einfach zu verstehen.

# Beispiel für ein Blockdiagramm: Automatische Verpackung

- Das Robotersystem kann verschiedenartige Objekte verpacken.
- Es benutzt dazu ein optisches Subsystem, um Objekte auf einem Förderband auszuwählen, den Typ des Objektes zu bestimmen und die entsprechende Verpackungsart auszuwählen.
- Danach werden die Objekte vom Förderband zur Verpackungseinheit transportiert.
- Verpackte Objekte werden auf ein anderes Förderband gelegt.



# Übung

## Ampelanlage

**Beschreibung:**
Eine moderne Ampelanlage besteht aus einer zentralen Ampelsteuerungslogik, den Zeitsignalen, Fußgängertaster und Fahrzeug-Kontaktschleife. Neben den 3 Farben gibt es auch ein akustisches Signal sowie ein Display zur Anzeige der Restsekunden bis zur nächsten Grünphase. Außerdem gibt es eine Verbindung zum Leitsystem, das die Phasenlängen entsprechend der Tageszeit bzw. Verkehrssituation anpasst.

**Aufgabe:**
Führen Sie eine Zerlegung der Gesamtanlage in logische Systemkomponenten (HW-/SW-unabhängig) durch.
Stellen Sie die Verbindungen zwischen den Systemkomponenten dar und erläutern Sie die Art/Richtung der Verbindung.
Stellen Sie zusätzlich die Systemgrenzen der Ampelanlage dar mit den relevanten Komponenten, die sich außerhalb der Systemgrenzen befinden.

**Tipp:**
Überlegen Sie sich, was Ihr Produkt sein soll: eine komplette Ampelanlage oder nur das Ampelsteuergerät. Dementsprechend müssen die Systemgrenzen gewählt werden.

**Zeit:**
15 Minuten, arbeiten Sie ggf. zusammen mit einem Partner

# Datenmodelle

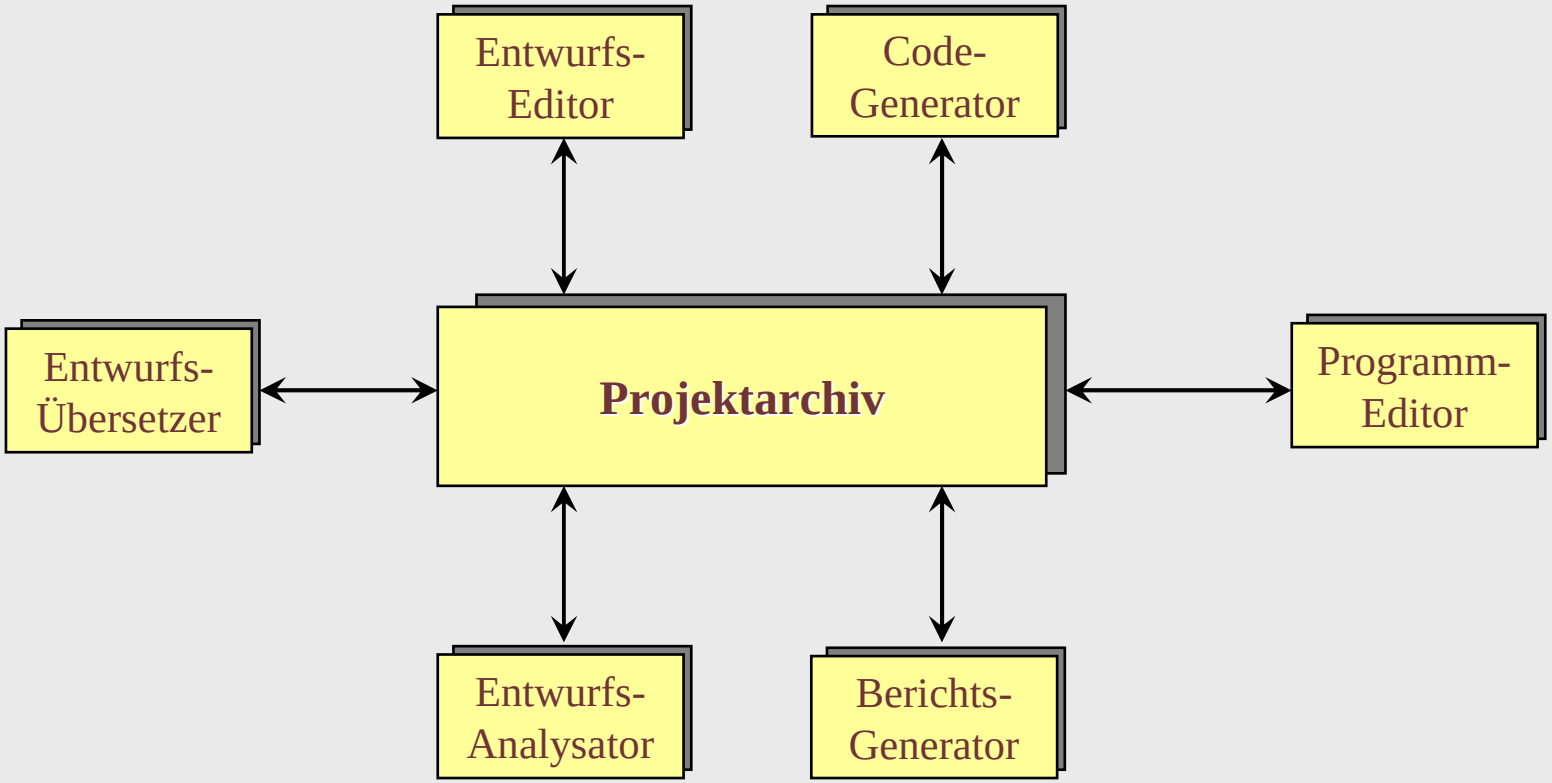
- Die Subsysteme, aus denen ein Gesamtsystem besteht, müssen Informationen austauschen, um auf effektive Weise miteinander zu arbeiten
- Es gibt zwei grundlegende Methoden, wie dies geschehen kann:

1. Datenspeichermodell (auch Repository-Modell):
   Alle gemeinsam benutzten Daten werden in einer zentralen Datenbank gespeichert, die für alle Subsysteme zugänglich ist.
2. Dezentrales Datenmodell:
   Jedes Subsystem unterhält seine eigene Datenbank.
   Der Datenaustausch mit anderen Subsystemen erfolgt durch das Versenden von Nachrichten.

# Datenspeichermodell

- Das Datenspeichermodell eignet sich besonders für Systeme, die auf einer großen globalen Datenbank bzw. Repository aufgebaut sind.
- Dies gilt insbesondere für Anwendungen, bei denen große Datenmengen von einem Subsystem generiert und von einem anderen verarbeitet werden
- Beispiele:
  - Betriebsleitsysteme
  - Produktionsplanungssysteme
  - CAD-Systeme
  - CASE-Werkzeugsammlungen

# Beispiel für Datenspeichermodell: Werkzeugsammlung



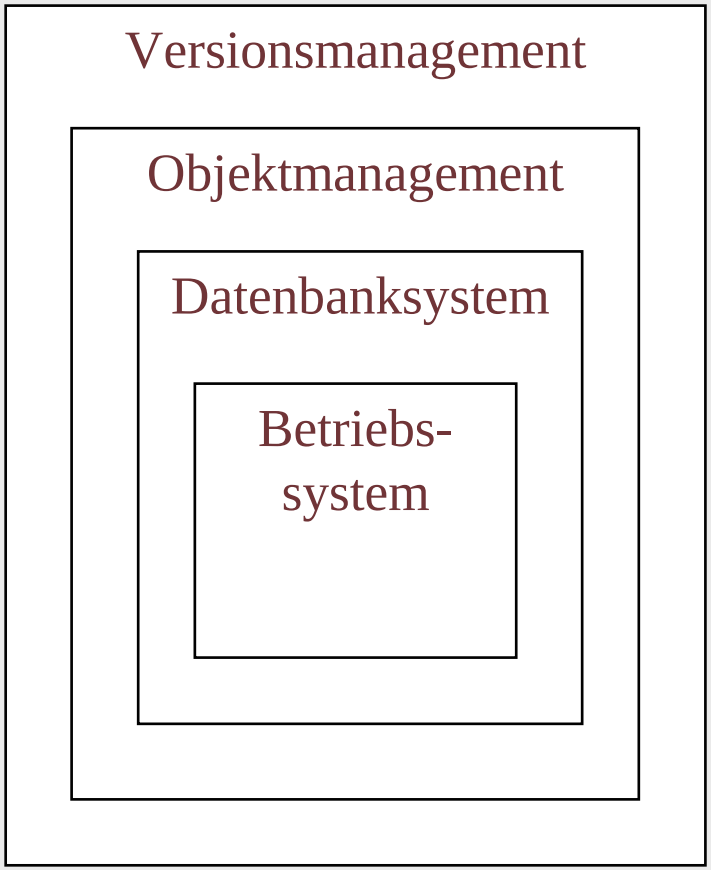

↔ Daten oder Steuerungsbefehle

# Schichtenmodell

- Das Schichtenmodell ist ein Architekturmodell, das
  - auf einer Hierarchie aufsteigender Abstraktionsebenen aufbaut und
  - entsprechend dieser Hierarchie schichtweise aufgebaut ist
- Jede Schicht des Modells
  - stellt ein Paket von Diensten bereit
  - zu deren Implementierung nur die von den darunter liegenden Schichten bereitgestellten Dienste verwendet werden (dürfen)

# Bsp. für Schichtenmodell: Konfigurationsmanagement

- Das Versionsmanagementsystem unterstützt das Konfigurationsmanagement, indem es Versionen von Objekten verwaltet.
- Dazu greift es auf ein Objektmanagement-system zurück, welches Speicher- und Verwaltungsdienste für die Objekte bereitstellt.
- Das Objektmanagementsystem verwendet seinerseits ein Datenbanksystem, das die grundlegende Speicherung der Daten und Dienste wie Transaktionsmanagement, Rückgängigmachung und Wiederherstellung sowie Zugriffskontrolle anbietet.
- Das Datenbanksystem benutzt schließlich die vom Betriebssystem angebotenen Dienste, insbesondere zur Dateispeicherung, zu seiner Implementierung.



# Bsp. für Schichtenmodell: OSI-Referenzmodell

- 7 Schichten des Open System Interconnection (OSI-) Referenzmodells für Netzwerkprotokolle

| | | |
|---|---|---|
| 7 | Anwendung | Bereitstellung einer Reihe von Funktionalitäten auf Anwendungsebene (z.B. Datenübertragung, E-Mail, Remote login) |
| 6 | Darstellung | Standardisierte Formatierung der Datenstrukturen zwecks einheitlicher Semantik ausgetauschter Daten (u.a. Kodierung, Kompression, Kryptographie) |
| 5 | Sitzung | Zuordnung einer Reihe diskreter Kommunikationsvorgange zu einem kontinuierlichen benutzerspezifischen Kommunikationsprozess<br>(u.a. organisatorische Maßnahmen zum Aufbau und Abbau der Sitzung) |
| 4 | Transport | vollständige Punkt-zu-Punkt (Sender-Empfänger, verbindungsorientierte) Kommunikation (u.a. Segmentierung von Datenpaketen und Routing-Optimierung zur Stauvermeidung) |
| 3 | Vermittlung | lokale (verbindungslose) Weitervermittlung von Datenpaketen<br>(u.a. Verwaltung interner Routing-Tabellen) |
| 2 | Sicherheit | Gewährleistung fehlerfreier Datenübertragung, geregelter Zugriff auf das Übertragungsmedium<br>(z.B. Fehlerkorrektur, Vermeidung und Erkennung von Zugriffskollisionen) |
| 1 | Bitübertragung | physikalische Schicht, Übertragungsmedium (z.B. Glasfaserkabel) |

# Vorteile des Schichtenmodell

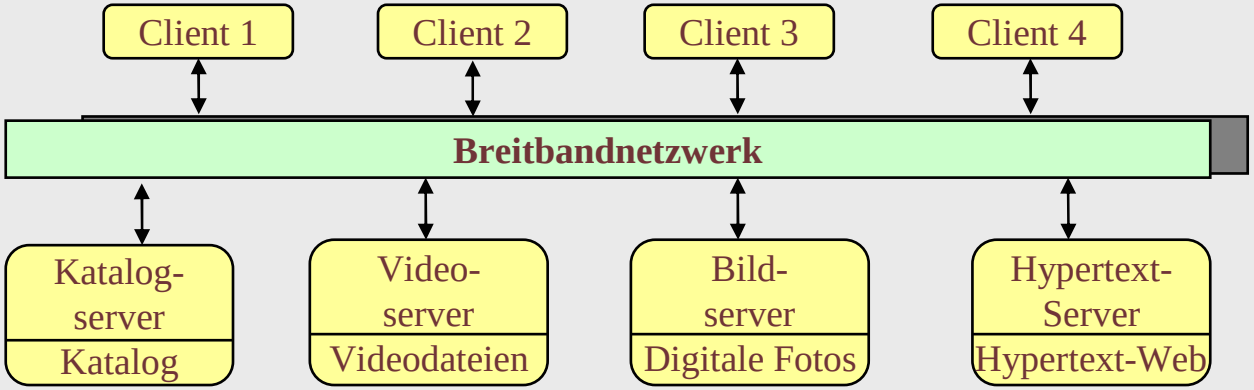
- Der Schichtenansatz begünstigt die schrittweise Entwicklung von Systemen. Sobald eine Schicht entwickelt worden ist, können einige der in dieser Schicht enthaltenen Dienste für die Benutzer verfügbar gemacht werden.
- Diese Architektur ist zudem veränderbar und portierbar
  - Solange die Schnittstellen nicht verändert werden, kann man eine Schicht komplett durch eine andere ersetzen.
  - Wenn sich die Schnittstellen einer Schicht hingegen ändern, wird davon nur die angrenzende Schicht betroffen.
  - Da bei Schichtensystemen die maschinenabhängigen Eigenschaften in den unteren Schichten angeordnet sind, können sie relativ leicht auf anderen Computern implementiert werden, indem man die inneren, maschinenspezifischen Schichten neu erstellt.

# Client/Server-Modell

- Das Client/Server-Architekturmodell ist ein verteiltes Systemmodell, das darstellt, wie Daten und Prozesse über eine Menge von Prozessoren verteilt werden können.
- Die Hauptkomponenten dieses Systems sind:

  1. Eine Menge unabhängiger Server, welche Dienste für andere Subsysteme anbieten.
     z.B. Druckserver (Druckerdienste), Dateiserver (Dateiverwaltungsdienste), Compiler-Server (Übersetzungsdienste)
  2. Eine Menge i. A. unabhängiger Clients, welche die von den Servern angebotenen Dienste abrufen. Es kann durchaus vorkommen, dass ein Clientprogramm mehrmals gleichzeitig ausgeführt wird.
  3. Ein Netzwerk, über das die Clients auf die Dienste zugreifen können. (prinzipiell nicht erforderlich, falls Clients und Server auf demselben Computer laufen)

# Beispiel für Client/Server-Modell: Hypertextsystem




- Hypertextsystem für eine Film- und Fotobibliothek
- Diverse Server zeigen die verschiedenen Medienarten an und verwalten sie.
  - Einzelbilder von Videos erfordern eine schnelle und synchrone Übertragung, aber nicht unbedingt eine hohe Auflosung und können in einem Speicher komprimiert werden.
  - Bei Standbildern ist dagegen eine Übertragung in hoher Auflosung erforderlich.
  - Der Katalog muss in der Lage sein, eine Vielfalt von Anfragen zu bewältigen und Verbindungen zu dem Hypertextinformationssystem anzubieten.
- Bei dem Clientprogramm handelt es sich lediglich um eine integrierte Bedienoberfläche zu diesen Diensten.

# Verteilte Systeme

- Bei verteilten Systemen läuft die Software auf einer durch das Netzwerk nur lose miteinander verbundenen Gruppe von Rechnern.
- Verteilte Systeme bestehen aus lose integrierten, unabhängigen Teilen.
- Die Middleware ist eine Software, die die Teile des Systems verwaltet und die plattformübergreifende Kommunikation zwischen den Systemteilen ermöglicht.

# Vorteile verteilter Systeme

- **Ressourcenteilung**
  - ermöglicht gemeinsame Nutzung von Hard- und Software seitens der verteilten Module
- **Offenheit**
  - Hard- und Software können von unterschiedlichen Herstellern stammen.
- **Nebenläufigkeit**
  - Unabhängige Module können parallel laufen.
- **Skalierbarkeit**
  - Kapazitäten können durch Hinzufugen von Ressourcen erweitert werden.
- **Fehlertoleranz**
  - Hard- und Softwarefehler können durch mehrfaches Vorhandensein von Ressourcen oder Daten toleriert werden.
- **Transparenz**
  - Die verteilte Struktur ist für den Benutzer unsichtbar.

# Nachteile verteilter Systeme

- **Erhöhte Komplexität**
  - Verhaltenweise des Gesamtsystems ist schwerer zu verstehen.
- **Erschwertes Sicherstellen der Informationssicherheit**
  - Der Datentransport im Netzwerk könnte belauscht und manipuliert werden.
- **Erschwerte Verwaltbarkeit**
  - Unterschiedliche Plattformen, Betriebssysteme und SW-Versionen
  - Fehler in einem Computer können sich auf andere Computer übertragen
- **Unvorhersagbarkeit**
  - Die Reaktionen des Systems sind von der Systemauslastung und Netzwerkbelastung abhängig.

# Middleware: CORBA

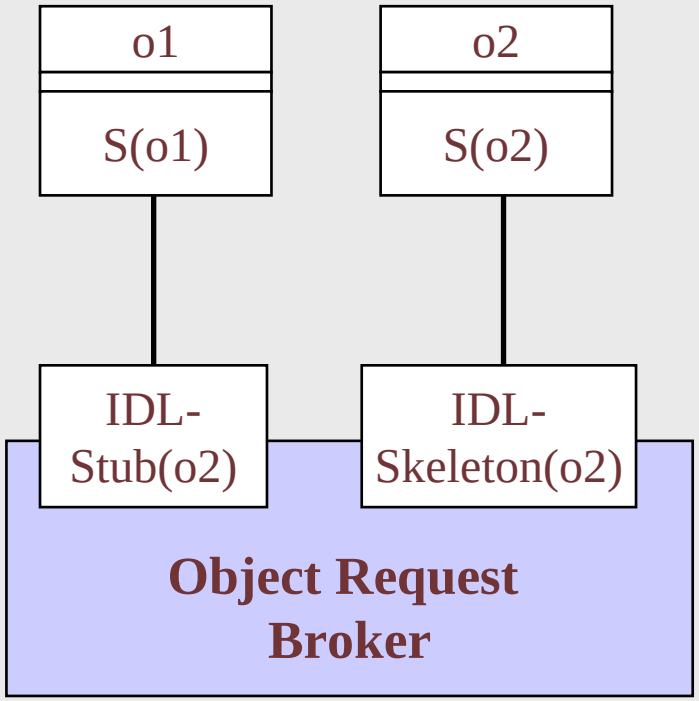
- CORBA (Common Objects Request Broker Architecture) ist eine konkrete Architektur für verteilte Systeme
- CORBA basiert auf einer Reihe von Standards, die von der Object Management Group (OMG) definiert wurden
- Es sind eine Reihe von CORBA-Implementierungen, sowohl für Windows- als auch für UNIX-Plattformen von verschiedenen Herstellern entwickelt worden
- Ein CORBA-System besteht aus einer Menge verteilter Objekte, die in unterschiedlichen Programmiersprachen implementiert sein können
- Die Middleware zur Kommunikation zwischen den verteilten Objekten wird Object Request Broker genannt
- DCOM ist die von Microsoft entwickelte "Konkurrenz" zum CORBA. (Es gibt sogar CORBA-DCOM-Bridges)

# Middlware:CORBA

**Bestandteile von CORBA**:

- Modell für Anwendungsobjekte, umfasst u.a.
  - Aufrufsemantik der Objekte
  - sprachneutrale Beschreibung der Schnittstellen mittels IDL (Interface Definition Language)
- Object Request Broker
  - vermittelt Anfragen an Objektdienste
- Allgemeine Objektdienste, z. B.:
  - Persistenz-Dienste (ermöglichen das Sichern von Objekten)
  - Verzeichnis-Dienste (ermöglichen das einfache Auffinden von Objekten)
  - Transaktions-Dienste (ermöglichen die atomare Ausführung einer Folge von Aufrufen)
- Gemeinsame Komponenten, die auf diesen Objektdiensten aufgebaut sind und von Anwendungen benötigt werden.

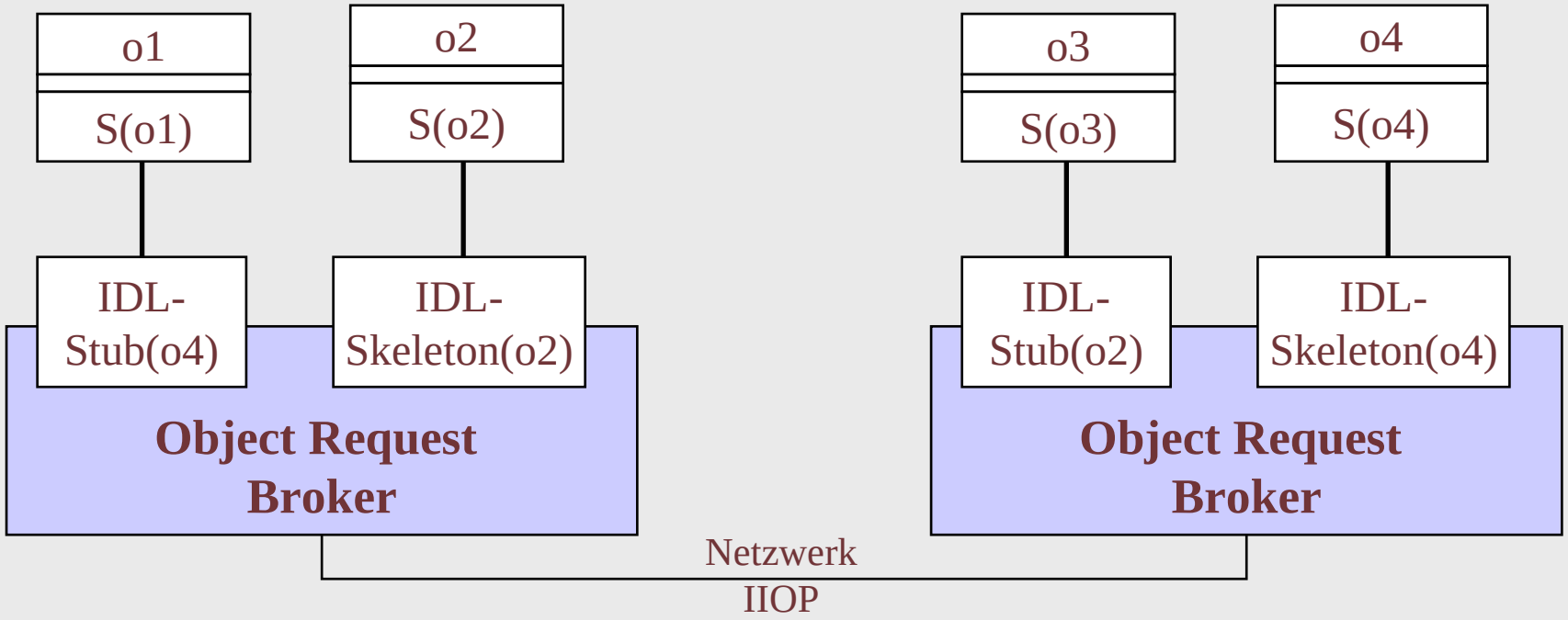
# Objektkommunikation über ein ORB



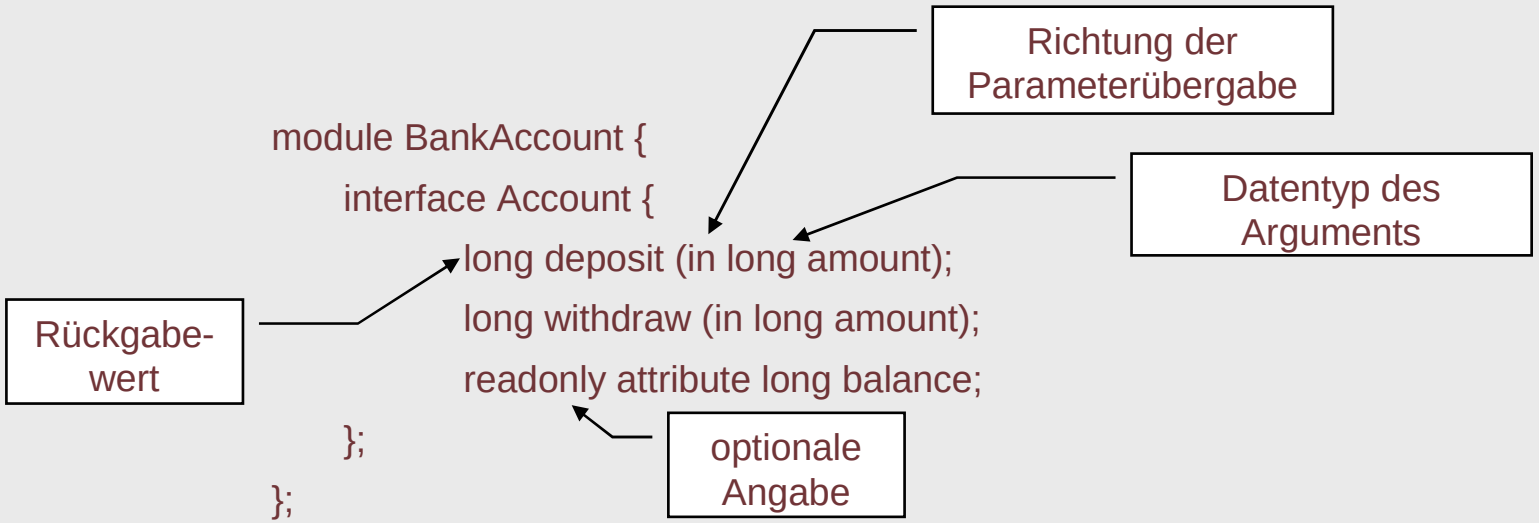

- Jedes diensterbringende Objekt besitzt einen IDL-Skeleton (der die Objektdienste aufruft)
- Zu jedem Aufrufer eines Objektes wird ein IDL-Stub erzeugt (der die Schnittstelle des diensterbringenden Objektes anbietet)
- Beispiel:
  - o1 sei das aufrufende Objekt
  - o2 biete den aufgerufenen Dienst S(o2) an.
  - Das IDL-Stub(o2) bietet die Schnittstelle des aufgerufenen Objekts o2 an und gibt die Aufrufe von o1 über den Object Request Broker an den IDL-Skeleton(o2) weiter.
  - Der IDL-Skeleton(o2) ruft den Dienst S(o2) auf.

# Kommunikation zwischen mehreren ORBs

- Die Kommunikation zwischen den ORBs erfolgt über das TCP/IP basierte Internet Inter-ORB-Protocol (IIOP)
- Dies ermöglicht die Anforderung von Diensten auf anderen Rechnern im Netzwerk.
- Beispiel: o1 ruft o4 auf und o3 ruft o2 auf



# CORBA: Beispiel für IDL-Beschreibung




Es stehen 3 Arten der Parameterübergabe zur Verfügung

- in Übergabe vom aufrufenden Objekt (Client) zum aufgerufenen Objekt (Server)
- out Übergabe vom Server zum Client
- inout Übergabe vom Client zum Server und zurück

# CORBA: Beispiel für IDL-Beschreibung

## Das "echo" Beispiel:

```
//  Eine einfache CORBA Test-Klasse, die einen Eingabe-String
//  wieder zurück "echoed"

module Test
   { interface Echo
        {                              //  Echo the given string back to the client.
          string                       //  The returned string - which is the same as Message
            Echo_String (
              in string Message        //  The string which should be echoed.
                );
         };
};
```

Quelle: Wikipedia:
File: test-echo.idl ([http://cvs.sourceforge.net/viewcvs.py/adacl/WikiBook_Ada/Source/test-echo.idl?only_with_tag=HEAD&view=markup view])

# Zum Schluss dieses Abschnitts ...

# Inhalt

# Ziel des Software-Feinentwurfs

- Ziel des Software-Feinentwurfs ist es, die im Software-Grobentwurf erstellte Software-Architektur zu verfeinern.
- Der Feinentwurf beschreibt die Detailstruktur des Systems, evtl. angepasst an die Besonderheiten der Implementierungssprache und Plattform.
  - z.B. Sprachenparadigma (funktional, objektorientiert)
  - z.B. Speicherbeschränkungen (eingebettete Systeme)
  - z.B. besonderer Befehlssatz des Zielprozessors
- Der Feinentwurf sollte so detailliert sein, dass der Programmierer den Programmcode ohne eigene Interpretationen der Aufgabenstellung in ausführbaren Code überführen kann.

# Detaillierung des Software-Grobentwurfs

- Beim Software-Feinentwurf steht die Detaillierung der Systemkomponenten im Vordergrund.
- Beim Software-Grobentwurf steht die Aufteilung des Gesamtsystems in Systemkomponenten und die Beschreibung der Beziehungen der Systemkomponente im Vordergrund.
- Im Software-Feinentwurf wird der Grobentwurf so verfeinert, dass alle Funktionen/Operationen der Systemkomponenten auf Realisierungsebene detailliert beschrieben werden

# Betrachtung einzelner Systemkomponenten

- Die Betrachtung einzelner Systemkomponenten ermöglicht eine Aufteilung der Entwurfs- und Implementierungstätigkeiten auf mehrere Mitarbeiter

- Diese Aufteilung ist speziell bei größeren Projekten unerläßlich, damit das zu entwickelnde Softwareprodukt in einem angemessenen Zeitrahmen entwickelt werden kann.

# Programmiersprachenneutrale Notationen

- Ziel:
  Beschreibung von Operationen (typischerweise Algorithmen) durch Abstrahieren syntaktischer Programmiersprachendetails
- Vorteile:
  - Wahl der Programmiersprache kann später erfolgen
  - Durch Abstraktion kann man die Aufmerksamkeit auf die Korrektheit des Algorithmus führen (leichtere Analysierbarkeit)
- Beispiele:
  - Struktogramme
  - Pseudocode

# Struktogramme

- Ein Struktogramm (Nassi-Shneiderman-Diagramm) ist eine Entwurfsmethode für die strukturierte Programmierung. Es ist genormt nach DIN 66261.
  Benannt wurde es nach seinen Vätern Dr. Ike Nassi und Dr. Ben Shneiderman.
- Mittels Struktogrammen lassen sich Algorithmen programmier-sprachenneutral (bezüglich prozeduraler Sprachen) darstellen.
- Die Methode zerlegt das Gesamtproblem, das man mit dem gewünschten Algorithmus lösen will, in immer kleinere Teilprobleme, bis schließlich nur noch elementare Grundstrukturen wie Sequenzen und Kontrollstrukturen zur Lösung des Problems übrig bleiben. Diese können dann durch ein Struktogramm visualisiert werden.

# Elemente des Struktogramms

- Ein Struktogramm ist eine graphische Darstellung des Kontroll- und Datenflusses eines in prozeduraler Sprache darzustellenden Algorithmus.
- Dabei werden die vier Elemente eines wohlstrukturierten Programms
  - Befehl
  - Sequenz
  - Verzweigung
  - Schleife

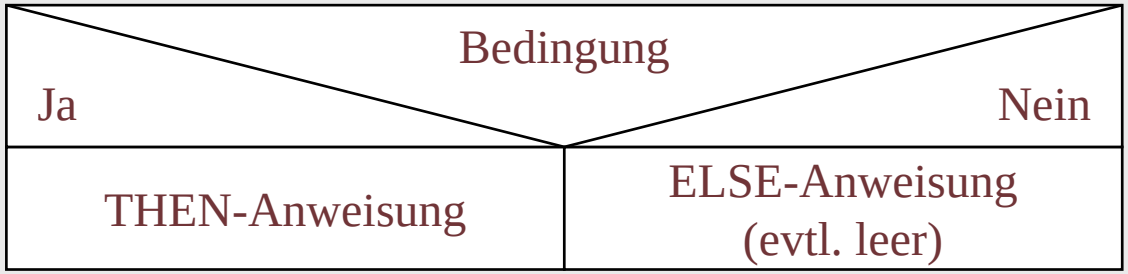
  durch die im folgenden angegebenen Diagrammarten versinnbildlicht.
- Aus diesen lässt sich ein Struktogramm rekursiv herleiten.

# Sequenz von Anweisungen

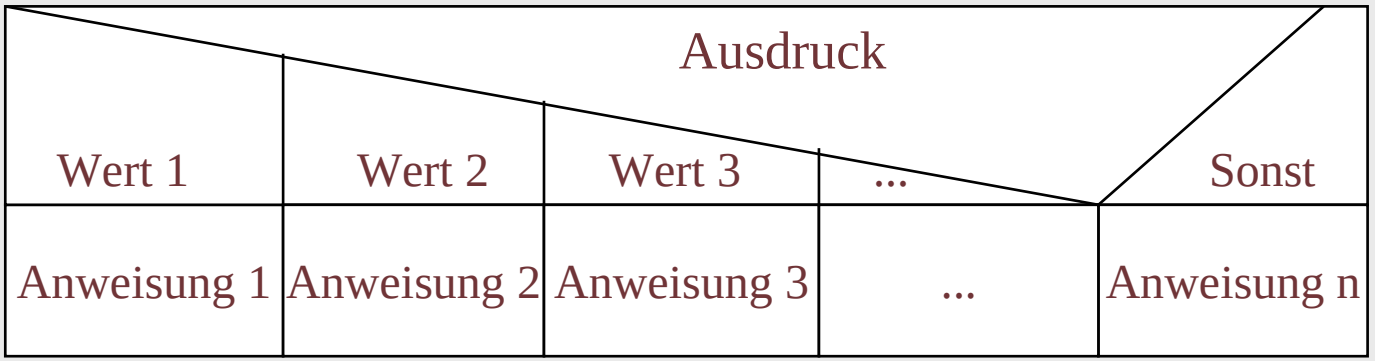
| |
|---|
| 1. Anweisung |
| 2. Anweisung |
| 3. Anweisung |
| ... |

# Verzweigung

- IF-Anweisung

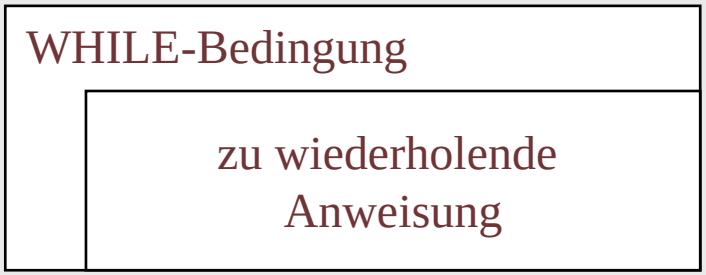



- CASE-Anweisung

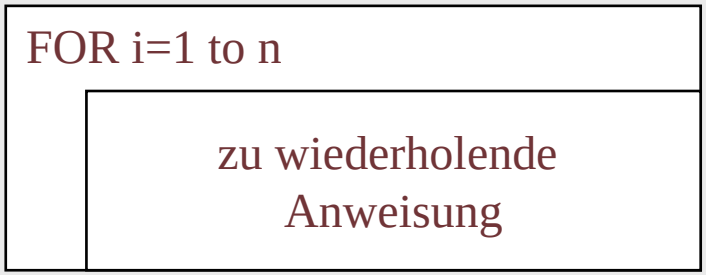


# Schleifen

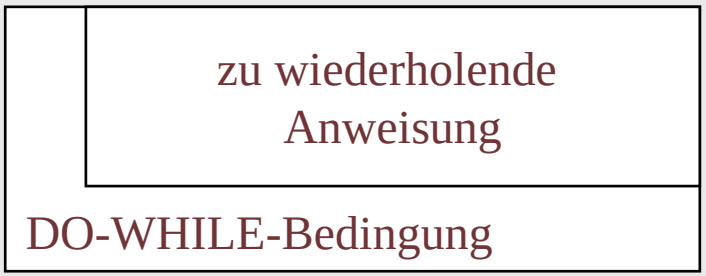
- WHILE-Anweisung




- FOR-Anweisung



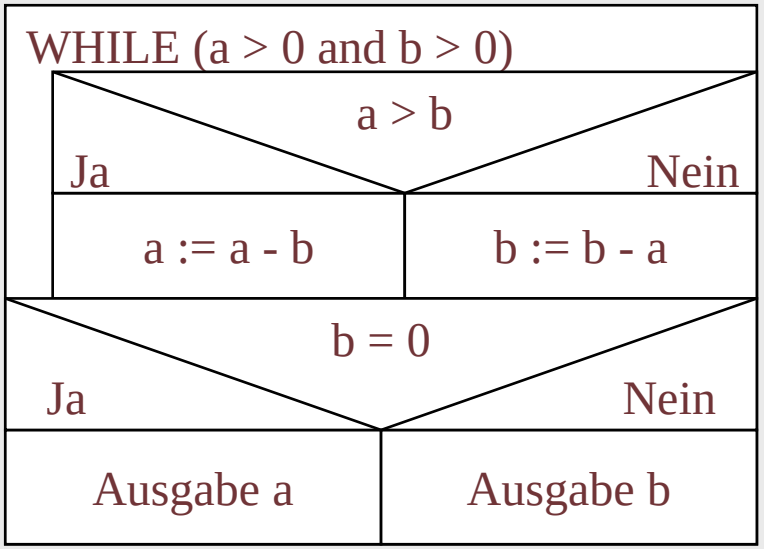

- DO-WHILE-Anweisung




Die innere Anweisung wird wiederholt, solange die Bedingung wahr ist

# Beispiel: Berechnung des ggT

- Das folgende Programm berechnet den größten gemeinsamen Teiler zweier natürlicher Zahlen a und b (euklidische Algorithmus)




Folge einer WHILE-Anweisung (mit enthaltener IF-Anweisung)
und

einer IF-Anweisung

# Pseudocode

- Pseudocode ist eine Darstellung eines Algorithmus in einer intuitiv verständlichen Sprache, die an eine Programmiersprache angelehnt ist, aber noch leichter lesbar ist als ausformulierter Programmcode.
  - Man verwendet vornehmlich natürlichsprachliche bzw. mathematische Darstellungselemente.
  - Syntaktische Details der Zielsprache stehen nicht im Vordergrund.
- Beispiel:
  - Der Algorithmus liest Zahlen aus einer Datei ein und berechnet deren Summe

```
sum := 0
read i from file
while i > 0 do
  sum := sum + i
  read i from file
end-while
```

# Zum Schluss dieses Abschnitts ...