Software Engineering

Informatik II.

5. Software-Entwicklung

– Konzeption –

Dipl.-Inform. Hartmut Petters

Vorwort – was ich noch zu sagen hätte ...

Basis dieser Vorlesung sind vor allem die folgenden Ausarbeitungen

- Vorlesungsskript „Software Engineering“
  von Prof. Dr. Martin Glinz Universität Zürich
  http://www.ifi.unizh.ch/groups/req/courses/ses/
- Vorlesungsskript „Informatik II – Software Engineering“
  von Frau Prof. Dr. Kühn FH Karlsruhe FB W
  http://www.home.fh-karlsruhe.de/~kuin0001/inhalt.htm
- Das Buch „Software Engineering“ 6. Auflage/2001
  von Prof. Ian Sommerville University of Lancaster (UK)
  Addison Wesley ISBN 3-8273-7001-9

Konkret entnommene Beiträge sind i.d.R. mit einem Quellen-Verweis gekennzeichnet – sollte dieser fehlen bitte ich um Nachsicht.

Den „roten Faden“ durch die Vorlesung habe ich dem Skript der Vorlesung von Prof. Dr. Martin Glinz entnommen und um eigene Beiträge erweitert bzw. aus den beiden anderen Quellen ergänzt.

Für die Möglichkeit der Verwendung der wesentlichen Inhalte möchte ich mich an dieser Stelle bei den Autoren herzlich bedanken.

Warum - Ausgangsbasis + Motivation

- Minimal-Software / überschaubare Entwicklung
  - Kein systematischer Entwurf erforderlich
- „Professionelle + richtige“ Software
  - Braucht systematischen + strukturierten Aufbau
  - Lösungskonzept zwingend erforderlich
    - Lösung *verstehen*
    - Entwicklungs*aufwand verteilen* auf mehrere Personen
    - Lösung *einbetten* in „Umgebung“
    - Lösung *geografisch verteilen*
  - Lösungskonzept legt Grundstein für *leicht pflegbare Systeme*
  - Konzeptions- / Architekturfehler sind kostspielig

Definition - Begriffsbestimmung

- Konzeption einer Lösung (architectural design)
  Erstellung und *Dokumentation* der *Architektur* oder des Grobentwurfs eines Systems
- Dabei werden die wesentlichen Komponenten der Lösung und die Interaktionen zwischen diesen Komponenten festgelegt.
- Architektur (architecture)
  Die *Organisationsstruktur* eines Systems oder einer Komponente
- Entwurf (design)
  1. Der *Prozess des Definierens* von *Architektur, Komponenten, Schnittstellen* und anderen Charakteristiken eines Systems oder einer Komponente
  2. Das *Ergebnis des Entwurfs-Prozesses* (s. 1.)
- Lösungskonzeption (software architecture, system architecture)
  Das *Dokument*, welches das Konzept der Lösung, d.h. die Architektur der zu erstellenden Software dokumentiert

Entwurfsprinzipien – Struktur + Abstraktion

- Struktur
  Gliedern der Lösung in *Komponenten* und *Interaktionen*
- Abstraktion
  Verstehen durch systematisches *Vergröbern / Verfeinern*
  - Gewinnung von *Übersicht* – Weglassung der Details
  - Die Darstellung eines *Details* – Weglassung / Vergröberung des Rests
  - Herstellung eines systematischen *Zusammenhangs* zwischen Übersichten + Detailsichten
  - Hauptsächlich 4 Arten
    - *Komposition*
    - *Benutzung*
    - *Spezialisierung*
    - *Aspektbildung*

Entwurfsprinzipien – Modularität

- Modularisierung
  ist eine Hauptaufgabe bei der Konzeption
- Modul (module)
  Eine *benannte*, klar *abgrenzbare Komponente* eines Systems
- Gute Module haben folgende Eigenschaften
  - In sich *geschlossene* Einheit
  - *Ohne Kenntnisse* über den inneren Aufbau *verwendbar*
  - *Kommunikation* mit der Umgebung erfolgt ausschließlich über klar *definierte Schnittstellen*
  - Im „Inneren" *rückwirkungsfrei änderbar*
  - *Korrektheit* ohne Kenntnis der Einbettung in das Gesamtsystem *überprüfbar*

Messung der Güte der Modularisierung

- Zwei charakteristische Maße
  - *Kohäsion*
  - *Kopplung*
- Kohäsion (cohesion)
  Ein Maß für die *Stärke des inneren Zusammenhangs* eines Moduls
  - Je höher die Kohäsion, desto besser die Modularisierung
    - *Schlecht* - zufällig, zeitlich
    - *Gut* - funktional, objektbezogen
- Kopplung (coupling)
  Ein Maß für die *Abhängigkeit zwischen zwei Modulen*
  - Je geringer die wechselseitige Kopplung zwischen den Modulen ist, desto besser ist die Modularisierung
    - *Schlecht* - Inhaltskopplung, globale Kopplung
    - *Akzeptabel* - Datenbereichskopplung
    - *Gut* - Datenkopplung

Entwurfsprinzipien – Geheimnisprinzip

- Geheimnisprinzip (information hidding)
  Kriterium zur *Gliederung* eines Systems in *Komponenten* (Daten, Abläufe + Algorithmen), so dass
  - Jede Komponente eine *Leistung* (oder eine Gruppe logischer, eng zusammenhängender Leistungen) *vollständig bereitstellt / erbringt*
  - außerhalb der Komponente nur bekannt ist, *was* die Komponente leistet
  - Nach außen *verborgen* bleibt, *wie* sie diese Leistung erbringt

  [Parnas 1972]

  - Fundamentales Prinzip zur *Beherrschung* komplexer Systeme
  - Auch im *täglichen Leben* fortwährend benutzt
  - Liefert *gute Modularisierungen*
  - *Stellt* die *Überprüfbarkeit* der Komponenten *sicher*

Das Geheimnisprinzip beim Software-Entwurf

- Komponente = *Modul*
- Leistung = *Funktionalität* des Moduls
- WAS? = *Modulschnittstelle*
- WIE? = *Entwurfsentscheidungen*
  + deren Realisierung

- Vier typische Arten von Entwurfsentscheidungen bei Software
  - Wie eine *Funktion realisiert* ist
  - Wie ein *Objekt aus dem Anwendungsbereich repräsentiert / realisiert* ist
  - Wie eine *Datenstruktur aufgebaut* ist / *bearbeitet* werden kann
  - Wie *Leistungen Dritter* realisiert sind

Entwurfsprinzipien – Schnittstellen + Verträge

- Schnittstelle (interface)
  Beschreibung des Leistungsangebots einer Komponente + deren Verfügbarkeit
  - Möglichst *realisierungsneutral*
  - In der Regel durch *Operationen + Datentypen*
  - Typische *Beschreibungsmittel*
    - *Voraussetzungen*
      (Vorbedingungen, Preconditions)
    - *Ergebniszusicherung*
      (Nachbedingungen, Postconditions, assertions)
    - *Invarianten* (invariants)
    - *Verpflichtungen* (obligations)
  - Bilden zusammen den Vertrag *zwischen Modul* als Leistungserbringer *und Klient* als Leistungsverwender

Entwurfsprinzipien – Nebenläufigkeit

- Problem
  Gleichzeitige, koordinierte Bearbeitung *mehrerer Aufgaben*
  - Realisiert durch Prozesse
- Prozess (process)
  Eine, durch ein Programm gegebene *Folge von Aktionen*, die sich in Bearbeitung befinden
- Nebenläufigkeit (concurrency)
  Die *parallele* oder *zeitlich verzahnte Bearbeitung* mehrerer Aufgaben
- Erfordert Kommunikation
  - *Informationsaustausch*
  - *Synchronisation* des Arbeitsfortschritts

Entwurfsprinzipien - Ressourcen

- *Zuordnung* der *Komponenten* zu Ressourcen ist notwendig
  - *Abschätzung* der technischen *Machbarkeit*
  - *Erfüllbarkeit* der gestellten *Anforderungen* (vor allem der Leistungen)
- Zuzuordnen sind
  - *Module* zu Prozessen
  - *Prozesse* zu Prozessoren
  - *Daten* zu Speichermodulen bzw. Speichermedien
  - *Prozesskommunikation* zu Kommunikations-Technologien + -Medien

Entwurfsprinzipien - Aspektorientierung

- Beschreibung von Querschnittsaufgaben
- Muss *systemweit*, dafür aspektspezifisch geschehen
- Typische Aspekte sind
  - *Datenhaltungskonzept*, insbesondere das konzeptionelle Datenbankschema bei Verwendung einer Datenbank
  - *Mensch-Maschine-Kommunikationskonzept* für die Gestaltung der Benutzerschnittstelle
  - *Fehlerbehandlungs-, Fehlertoleranz-, Sicherheitskonzepte*

Entwurfsprinzipien - Wiederverwendung

- Wo irgend möglich – *nicht neu entwickeln*
  - Wiederverwendung + Beschaffung
- Zu untersuchen
  - *Vollständige Beschaffung*
    (Standard-Software, konfigurierbare Bausteine)
  - Beschaffung *abgeschlossener Teilsysteme*
    (z.B. Datenbank-Systeme)
  - Realisierung durch *Einbettung in* einen existierenden Software-*Rahmen* (Framework)
  - *Nutzung* einzelner *Komponenten*
    (Programm- / Klassenbibliothek)
  - Wiederverwendung von *Architektur-* und *Entwurfsideen*
    (Architekturmetaphern, Entwurfsmuster)
  - *Modifikation des Lösungskonzepts* zwecks Verwendung von *Standardkomponenten*

Entwurfsprinzipien – Ästhetik

- Wahl und *konsequente Verwendung* eines Architekturstils
- Klar erkennbare, gestaltete Strukturen
- Wenig „irgendwie gewordene" Funktionen
- Kein „hingebasteltes" Stückwerk
- Der Struktur des *Problems* angemessene Struktur der *Architektur*
- Wahl der *einfachsten + klarsten Lösung* aus der Menge der möglichen Lösungen

  ↳ *„Geniale" Lösungen haben einfache + klare Formen (Gestaltstheorie)*

Entwurfsprinzipien – Qualität

- Merkmale guter Entwürfe
  - Effektivität
    Erfüllt die Vorgaben und *löst das Problem* das Auftraggebers
  - Wirtschaftlichkeit
    Gebrauchstauglich, kostengünstig + mehrfachverwendbar bzw. mehrfach verwendet
  - Software-technische Güte
    Leicht *verständlich, robust, zuverlässig + änderungsfreundlich*

  ↳ Erfordert *kontinuierliche Prüfmaßnahmen* im Entwurfsprozess

Das Lösungskonzept – 1te

- Dokumentation des Ergebnisses des Architektur-Entwurfs
- Mögliche Gliederung

1. *Einleitung*
   (1) Überblick
   (2) Ziele + Vorgaben
   (3) Einbettung + Abgrenzung
   (4) Lösungsalternativen
2. *Struktur der Lösung*
   (1) Übersicht
   (2) Prozess-Struktur
   (3) Modulare Struktur
   (4) Entwurf der Module
   (5) Physische Struktur

Das Lösungskonzept – 2te

3. *Aspektbezogene Teilkonzepte*

   Ein Unterkapitel je interessierendem Aspekt, z.B.

   - Datenhaltungskonzept
   - Mensch-Maschine-Kommunikation
   - Fehlerbehandlungskonzept
   - Fehlertoleranz-Konzept
   - Sicherheitskonzept
   - ...

4. *Voraussetzungen + benötigte Hilfsmittel / Abgrenzungen*

   (1) Benötigte Software

   (2) Benötigte Hardware

   (3) Benötigtes Umfeld

   (4) Was gehört nicht dazu / Was wird nicht gemacht

5. *Quellennachweis*

Der Entwurfprozess

- Erster Schritt der Lösung des Problems
- *Anforderungsspezifikation* als *Vorgabe* notwendig
- Zeitliche + hierarchische *Verzahnung* von Anforderungsspezifikation + Architekturentwurf möglich
- Ergebnisse immer in *separaten Dokumenten*
- Architekturentwurf
  Komponenten, Schnittstellen, Interaktionen, ...
- Detail-Entwurf
  Algorithmen + interne Datenstrukturen

Aufgaben des Architektur-Entwurfs – 1te

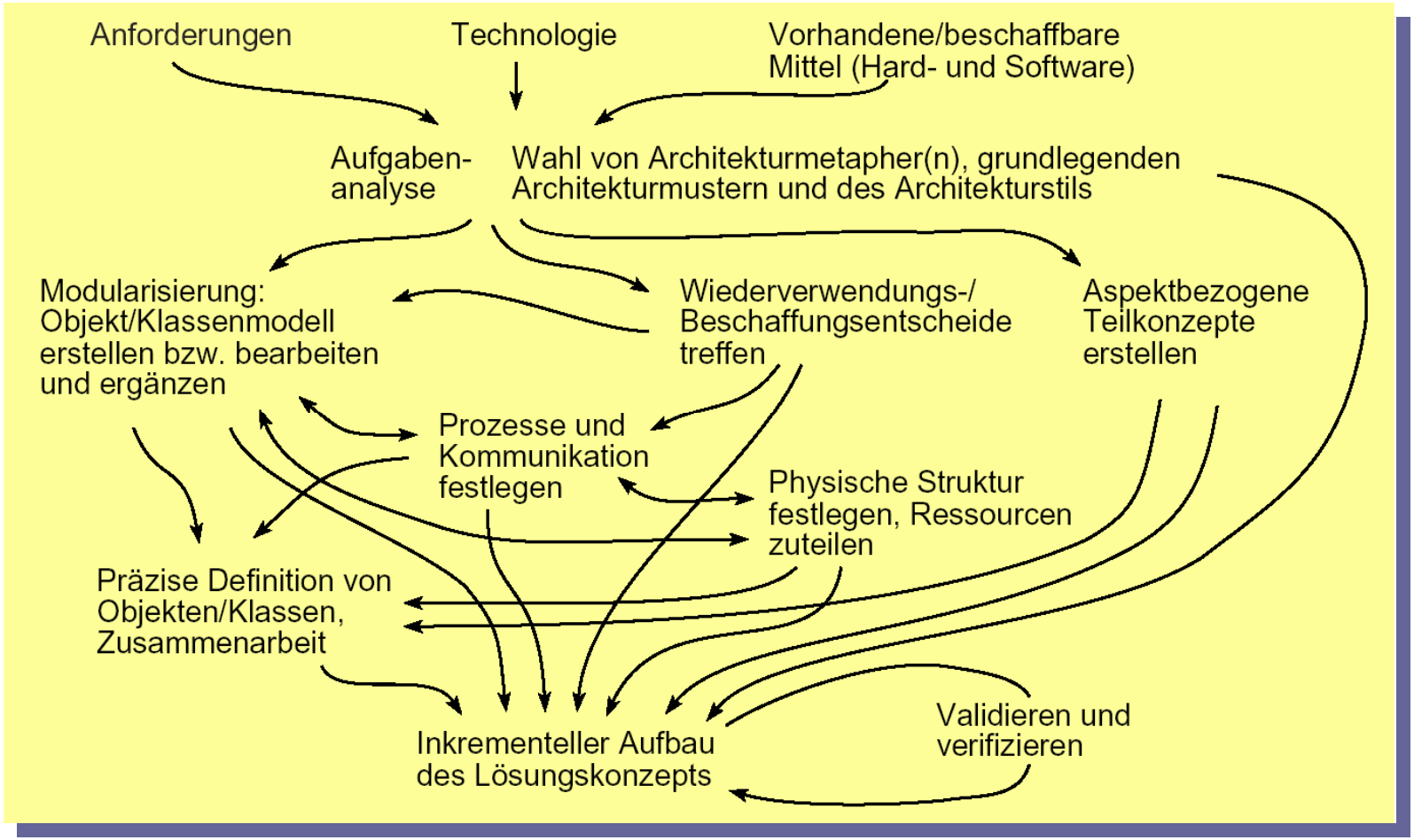
- Aufgabe analysieren
  - Anforderungen verstehen
  - Vorhandene, bzw. beschaffbare Technologien + Mittel analysieren

- Architektur modellieren + dokumentieren
  - Grundlegende Systemstruktur (Architektur) festlegen
    - Muster, Metaphern
      - ↳ Stil
  - Modularisieren
  - Nebenläufige Lösungen in Prozesse gliedern
  - Wiederverwendungs- und Beschaffungsentscheidungen treffen
  - Ressourcen klären + zuordnen
  - Aspektbezogene Teilkonzepte für Querschnittsaufgaben erstellen
  - Lösungskonzept (als Dokument) erstellen

Aufgaben des Architektur-Entwurfs – 2te

- Lösungskonzept prüfen
  - Anforderungen erfüllt?
  - Software-Technik OK?
  - Gutes Konzept?
  - Wirtschaftlich OK?

Vorgehen + Zusammenhänge



Lösungsvarianten

- Ganzen Lösungsraum betrachten
- Lösungs*varianten*
  - *Erkennen*
  - *Verfolgen*
  - *Abwägen*
    - Was kostet es, das Optimum zu verfehlen?
    - Was kostet die Untersuchung (Geld, Zeit, Ressourcen)?
  - *Entscheiden*
  - *Dokumentieren*
- *Kosten*
  - Nicht nur Entwicklungskosten der Varianten betrachten!
  - Auch Betriebskosten, Pflegekosten, Folgekosten (auch anderswo)
- Beschaffungsvarianten generell immer betrachten!

Beschaffung + Wiederverwendung

1. *Anforderungen* analysieren
2. *Hauptkriterien* definieren („KO"-Kriterien)
3. *Marktübersicht* verschaffen, *Grobauswahl* treffen
4. *System-Kandidaten evaluieren*
5. *Entscheidungsgrundlage* erarbeiten
6. *Entscheidungen forcieren* + dokumentieren

- Zu treibender Aufwand hängt ab von
  - Wichtigkeit / Priorität
  - Preis
  - Nutzungsdauer

Architekturstil

- Architekturstil (architectural style)
  Leitlinie für die *Gestaltung* einer Architektur
  - Verwendete *Modularten*
  - Verwendete *Arten von Kooperation* zwischen den Modulen
  - Benutzung passender *Architekturmuster*
  - Orientierung an einer *Architekturmetapher*

  } *Architekturstil*

- Im Folgenden
  - ↳ *Vorstellung ausgewählter Stilarten*

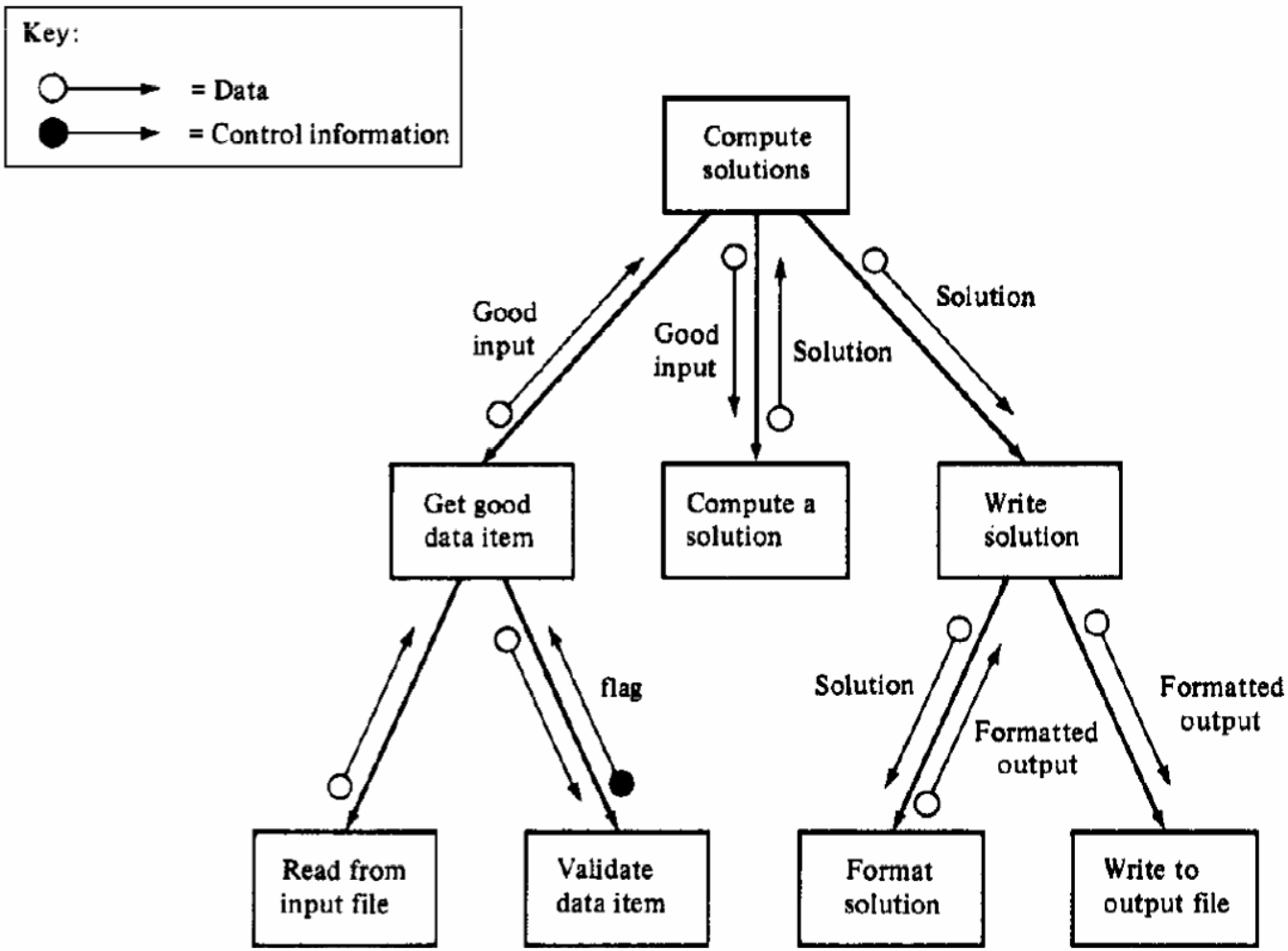
Funktionsorientierte Architektur

- Funktionsorientierte Architektur (structured design)
  - Jedes Modul führt eine *Funktion* aus
  - Zur Realisierung einer Funktion können andere, einfachere Funktionen aufgerufen werden

    ↳ Entwurf

    = *Hierarchie aufeinander aufbauender Funktionen*
  - Häufig gegliedert nach
    - Eingabe
    - Verarbeitung
    - Ausgabe
  - *Ungenügend Abstraktionsmöglichkeiten*
  - Liefert in vielen Fällen *keine gute Modularisierung*

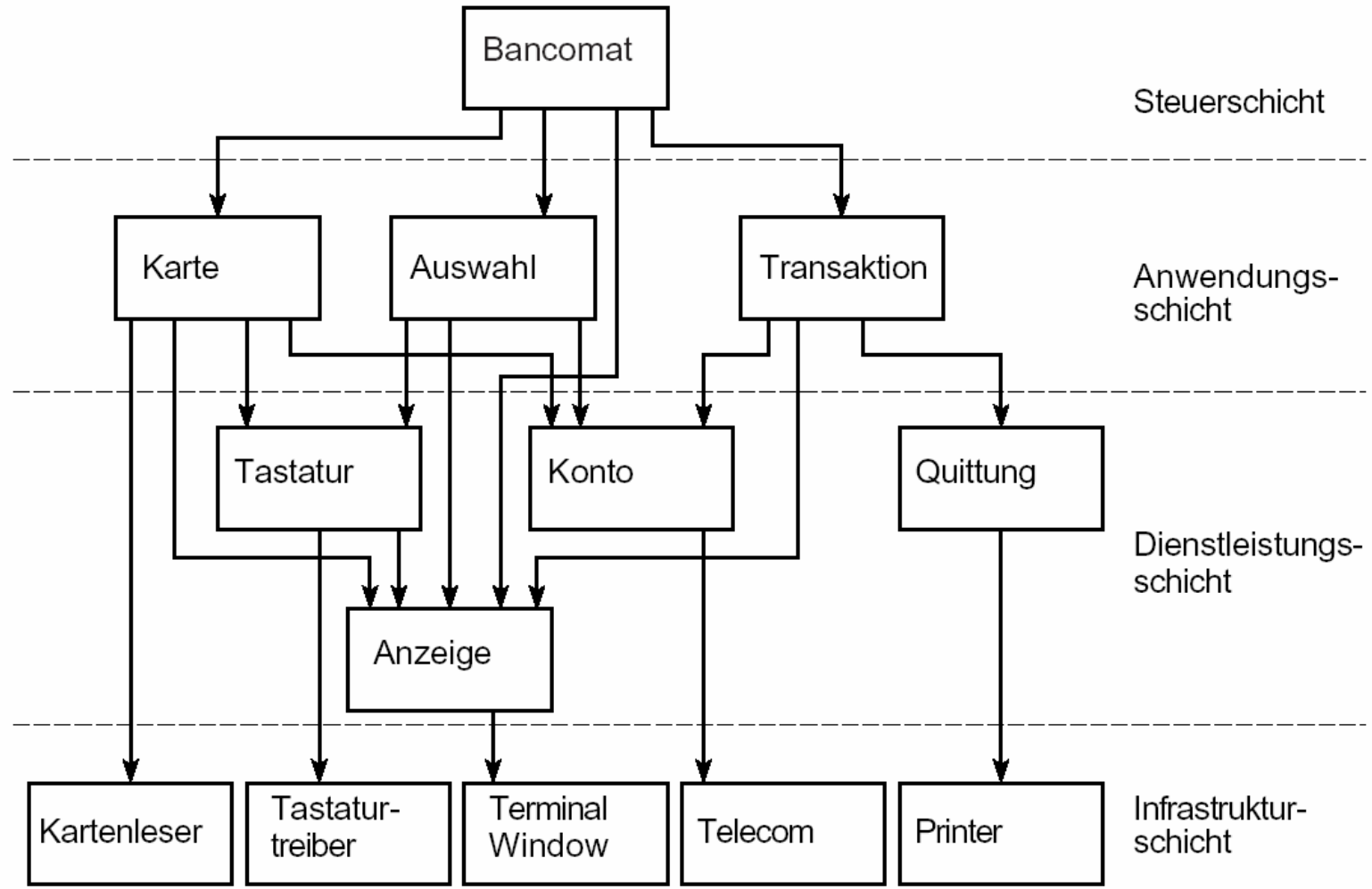
Beispiel: Funktionsorientierte Architektur



Datenorientierte Architektur

- Datenorientierte Architektur (Entwurf mit Datenabstraktion)
  - Modularisierung nach dem Geheimnisprinzip
  - Datenstrukturen und alle darauf möglichen Operationen sind zusammengefasst: *Datenabstraktion*
    - Notwendig zum Verbergen von Entwurfsentscheidungen
    - Realisierung durch Abstrakte Datentypen
  - System besteht aus einer *Menge aufeinander aufbauender Datenabstraktionen*
    - Jedes Modul *bietet Leistungen an*
    - Module *benutzen die Leistungen* anderer Module zur Realisierung der eigenen Leistungen
      - ↳ *Benutzungshierarchie*
    - Zusammenarbeit durch Verträge (Design by Contracts)
  - Zusätzlich häufig Gliederung in *Schichten*

Beispiel: Datenorientierte Architektur

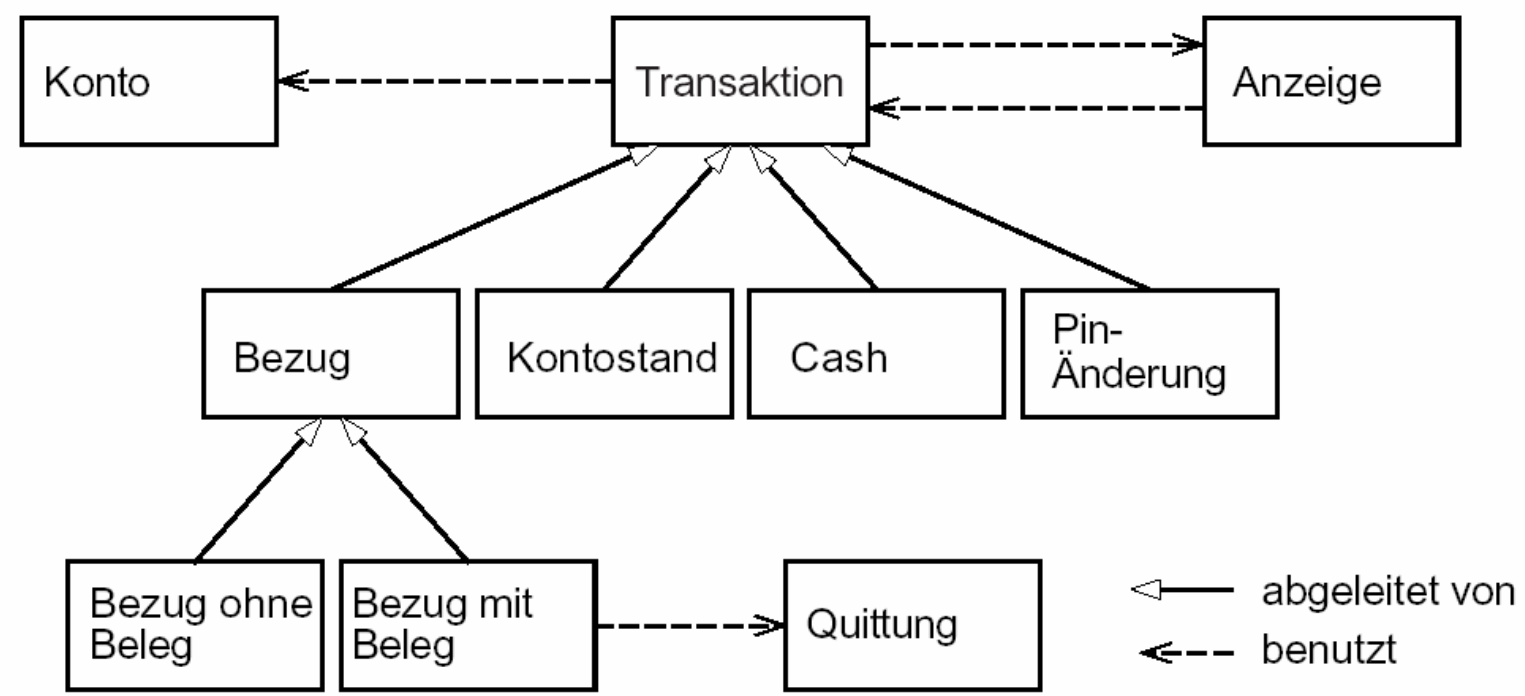


Objektorientierte Architektur 1te

- System besteht aus einer Menge *kooperierender Objekte*
- Module repräsentieren *Objekte des Problembereichs* oder benötigtes *Informations-Element*
- Abstrakte Beschreibung gleichartiger Objekte → Klasse
- Geeignet gebildete Klassen beachten das Geheimnisprinzip
  - ↳ *Gute Modularisierung*
- *Systematischer Zusammenhang* zwischen allgemeinen + speziellen Objekten
  - Objekte von Spezialklassen erben alle Strukturen + Operationen der übergeordneten, allgemeinen Klassen
    - ↳ Spezialisierungs- (Generalisierungs-) Hierarchie
  - Ermöglicht *problemnahe* Modellierung
  - Analog der *Begriffshierarchie* im menschlichen Denken
  - Schränkt jedoch die Anwendbarkeit des Geheimnisprinzips ein

Beispiel: Objektorientierte Architektur

Ausschnitt aus einem Bancomat-System



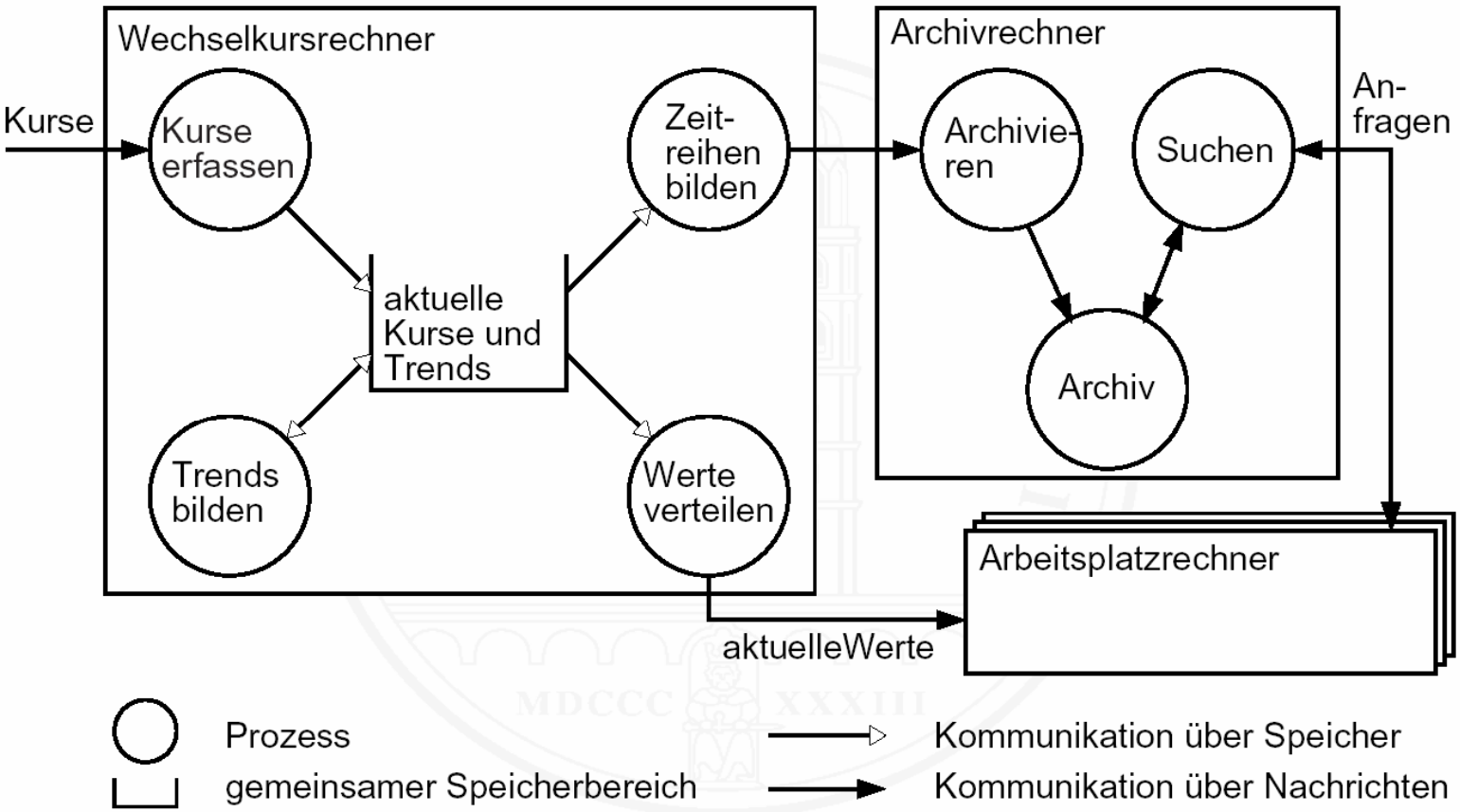
Objektorientierte Architektur – 2te

- Zwei Kooperationsmechanismen
  - *Benutzung*
  - *Vererbung*
  - ↳ Objektorientierter Entwurf ist anspruchsvoll
- Bei richtiger Anwendung
  - ↳ *Qualitativ hochwertige Modelle*
- Bei falscher Anwendung
  - ↳ Der Alptraum jedes Wartungs-Programmierers

Prozessorientierte Architektur

- System besteht aus einer Menge *unabhängig arbeitender*, untereinander *kooperierender* (systeminterner) *Akteure*
- Akteure sind typisch als Prozesse realisiert
- Prozesse *kooperieren* durch *Austausch von Nachrichten* oder durch Zugriff auf *gemeinsame Speicherbereiche*
- Prozesse sind die Module der obersten Stufe
- Jeder Prozess ist typisch ein sequentiell ablaufender Systemteil
- Prozesse sind selbst wieder modularisiert, z.B. in objektorientiertem Stil
- Entwurfsaufgaben
  - Prozess-Struktur
  - Kommunikationsarchitektur (oft zugekauft)
  - Interne Architekturen der Prozesse

Beispiel: Prozessorientierte Architektur



Komponentenorientierte Architektur

- Komponentenorientierte Architektur (verteilte Objekte)

  Komponente (im engeren Sinn)
  Stark gekapselte Menge zusammenhöriger Objekte / Klassen, die eine *gemeinsame Aufgabe lösen*
- System besteht aus einer *Menge von* (möglicherweise geografisch verteilter) *Komponenten*, die über *Makler / Agenten kommunizieren*
- Komponenten

  kennen

  ➢ *Schnittstellen* ihrer Partnerkomponenten

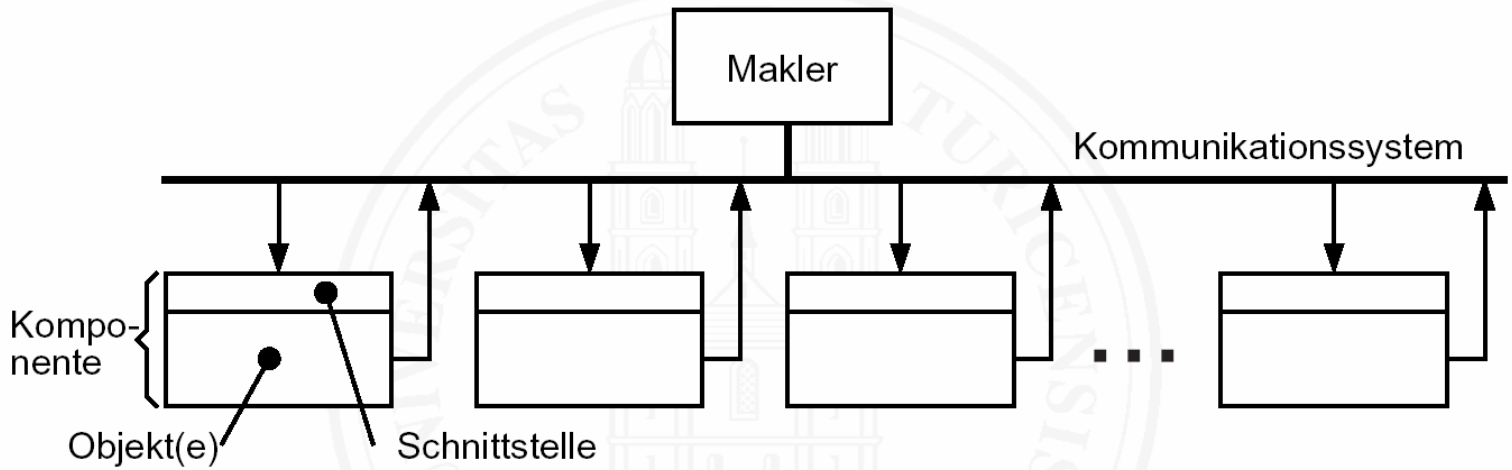
  ... kennen nicht

  ➢ *Art der Realisierung* der Kommunikation

  ➢ Geografische *Lokalisierung*

  ➢ *Implementierung* der Partnerkomponenten
- Typische Vertreter
  - Client-Server-Architektur
  - Middleware-Architektur

Beispiel: Komponentenorientierte Architektur



Wiederverwendung von Entwurfswissen

- Entwurfsmuster (design pattern)
  Eine spezielle Komponente, die eine *allgemeine, parametrierbare Lösung für ein typisches Entwurfsproblem* bereitstellt.

In der Architektur von Software-Systemen

- Architekturmuster
  Eine vorgefertigte, *parametrierbare Schablone* für die Gestaltung der Architektur eines Systems oder einer Komponente
- Rahmen (framework)
  Eine Menge kooperierender Programm-Module, die das *Grundgerüst für die Lösung* einer bestimmten Klasse von Problemen bilden

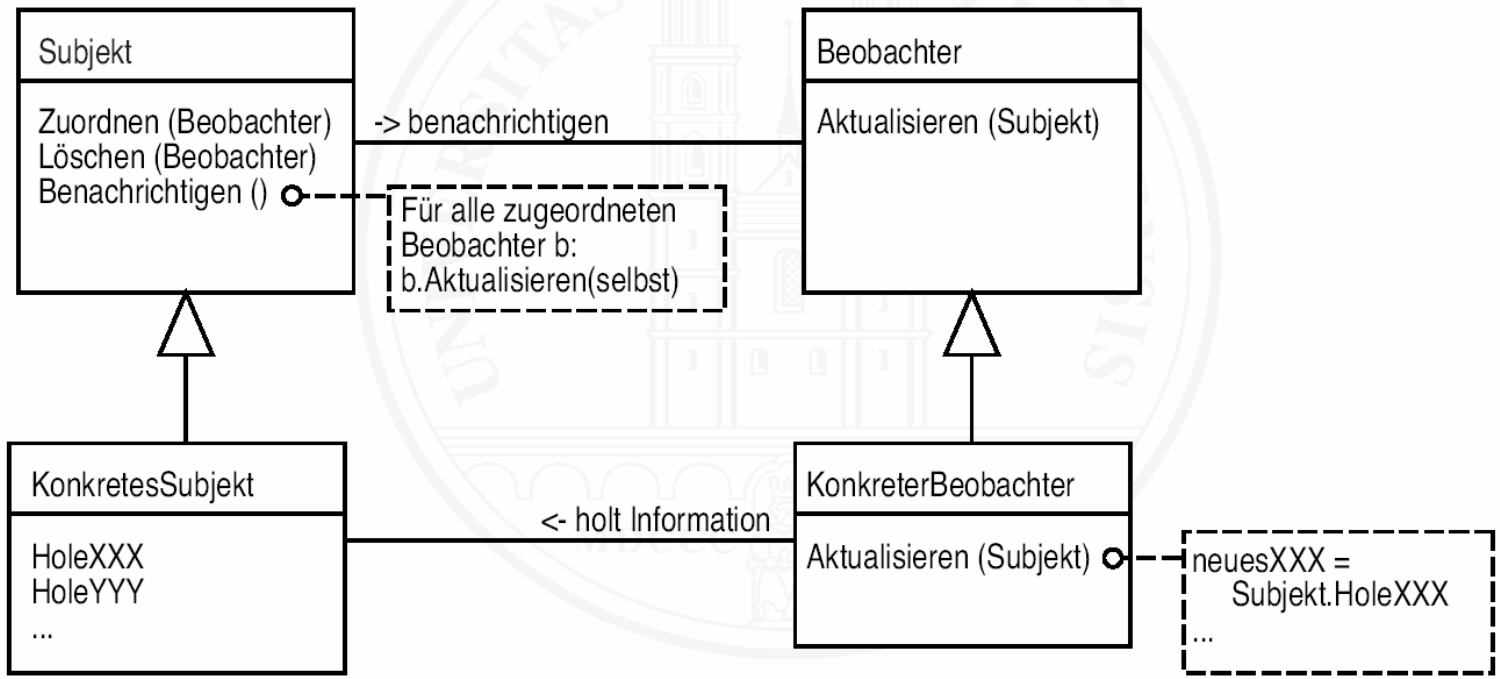
Entwurfsmuster

- Manche Entwurfsprobleme treten in sehr ähnlicher Form immer wieder auf
  - Idee
    - Problem *nicht jedes Mal* aufs Neue *lösen*
    - ... *sondern einmal* eine vorgefertigte, parametrierbare Lösungsschablone bereitstellen
    - ... von der *konkreten Lösungen* schnell *ableitbar* sind

  - *Wiederverwendung* von *Entwurfswissen*
  - *Begriffliche Basis* für die Kommunikation unter den Beteiligten

Das Beobachtungsmuster (observer pattern)

- Problem
  Entkopplung der Bearbeitung und Speicherung von Daten von ihrer Auswertung und externen Darstellung



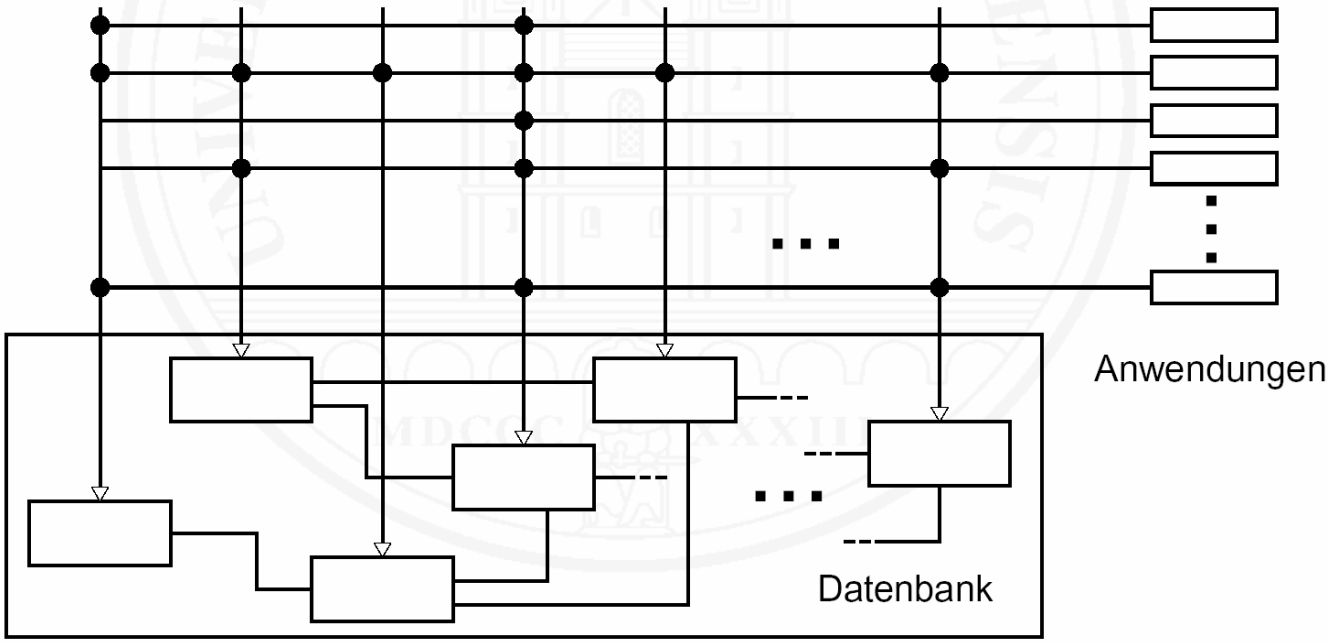
Architekturmuster

- *Vorgefertigte Strukturen* für typische Architekturprobleme
- *Wiederverwendung von Wissen* über Architekturen
- *Begriffliche Basis* für die Kommunikation unter den Beteiligten

- Beispiele
  - Strukturmuster
    - Matrixmuster
  - Steuermuster
    - EVA (Eingabe – Verarbeitung – Ausgabe)
    - Hauptschleife
    - Hollywood
  - Modularisierungs-/ Entkopplungsmuster
    - MVC (Model – View – Controller)

Beispiel: Architekturmuster - Matrixstruktur

- System besteht aus Menge von Daten- und Funktionsmodulen
- Jede Funktion kann auf jedes Datum zugreifen
- Funktionsmodule enthalten keine permanenten Daten
- Muster für die klassische Architektur datenbankbasierter Systeme



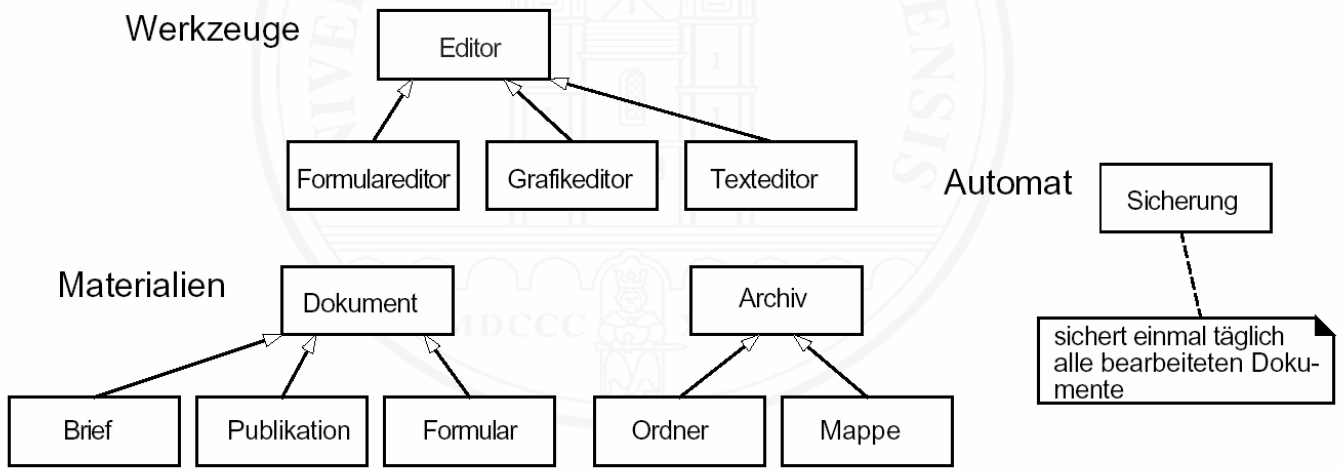
Architekturmetaphern

- Metapher
  sprachlicher Ausdruck, bei dem ein Wort aus seinem Bedeutungszusammenhang in einen anderen übertragen – als *Bild* – verwendet wird.
- Architekturmetapher
  Leitbild für die Gestaltung einer Architektur
  erschließt das Verständnis über *analoge*, vertraute *Bilder*

- *Beispiele*
  - WAM (Werkzeug-Automat-Material)
  - Organisationshierarchie
  - Virtuelle Maschinen
  - Steckersystem

Beispiel: Architekturmetapher - WAM

- Werkzeug
  - Gegenüber Materialien aktiv – *bearbeitet Materialien*
  - Gegenüber Menschen *assistierend* – Mensch bedient
- Automat
  - *Aktiv* – arbeitet *vollautomatisch*
- Material
  - *Passiv* – ist *Arbeitsgegenstand* oder *Arbeitsergebnis*



Zusammenfassung - Konzeption

- Anforderungsspezifikation als Vorgabe erforderlich
- Systematischer Entwurf + strukturierter Aufbau wichtig
- Lösungskonzept zwingend erforderlich
- Lösungsvarianten eruieren + betrachten
- Einheitliche Architektur vorgeben
- Architekturfehler sind kostspielig
- Gliederung in Komponenten + Interaktionen
- Hauptaufgaben
  - Modularisierung
  - Geheimnisprinzip einsetzen
  - Schnittstellen festlegen
  - Wiederverwendung praktizieren

Literatur – Software Engineering

- Skript Informatik II — Prof. Dr. Kühn / Fb W FH Karlsruhe
  http://www.home.fh-karlsruhe.de/~kuin0001/inhalt.htm
- Skript Software Engineering — Prof. Dr. Martin Glinz Universität Zürich
  http://www.ifi.unizh.ch/groups/req/courses/se_I/
- Skript Software Engineering II — Bernd Kahlbrandt FH Hamburg
  http://www.informatik.fh-hamburg.de/~khb/st4se2/
- Software Engineering
  Ian Sommerville — (ISBN3-82737-001-9)
- Software Engineering
  - Grundkurs für Praktiker –
  Roger S. Pressman — (ISBN 3-89028-163-X)
- Software Entwurf
  - Methoden und Werkzeuge –
  A. Schulz — (ISBN 3-486-21608-2)
- Software Engineering und Prototyping
  Thorsten Spitta — (ISBN 3-540-17542-3)
- CASE
  Helmut Balzert — (ISBN 3-411-03224-3)
- Software-Qualitätssicherung
  Ernest Wallmüller — (ISBN 3-446-15846-4)

Software Engineering

Informatik II.

6. Software-Entwicklung

– Aufwandsabschätzung –