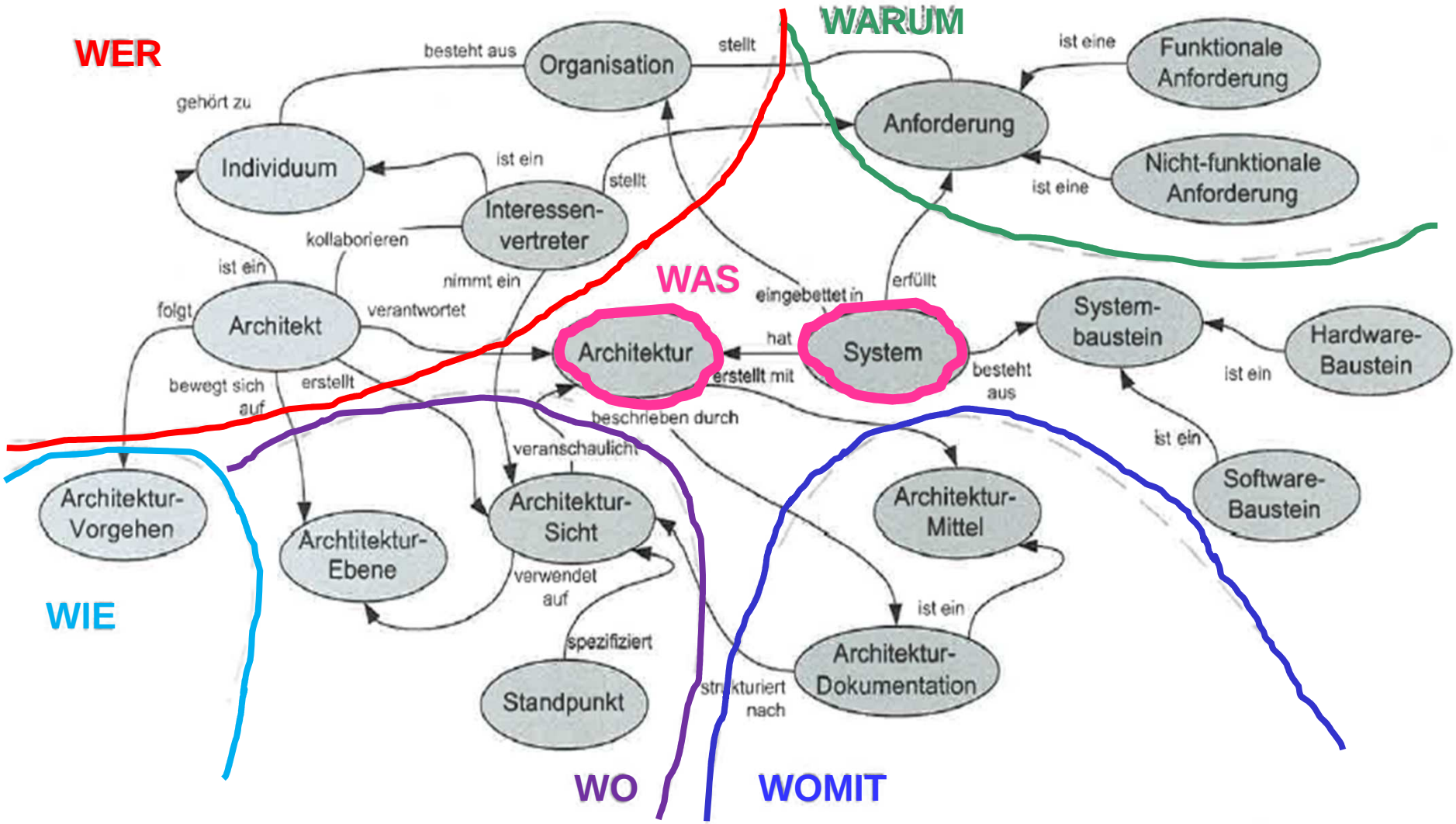
Softwarearchitektur und Design
Block 3b – Architekturen; Organisationen

SS2014

DI Dr. Gottfried Bauer

LV-Typ: VO, UE
Semester: 2
LV-Nummer: **S 2012 ILV**
LV-Bezeichnung: Softwarearchitektur und Design

Mindmap zur SW-Architektur



O.Vogel et.al., Software-Architektur, Grundlagen-Konzepte-Praxis, Spektrum, 2009

Inhalt - Block 3b

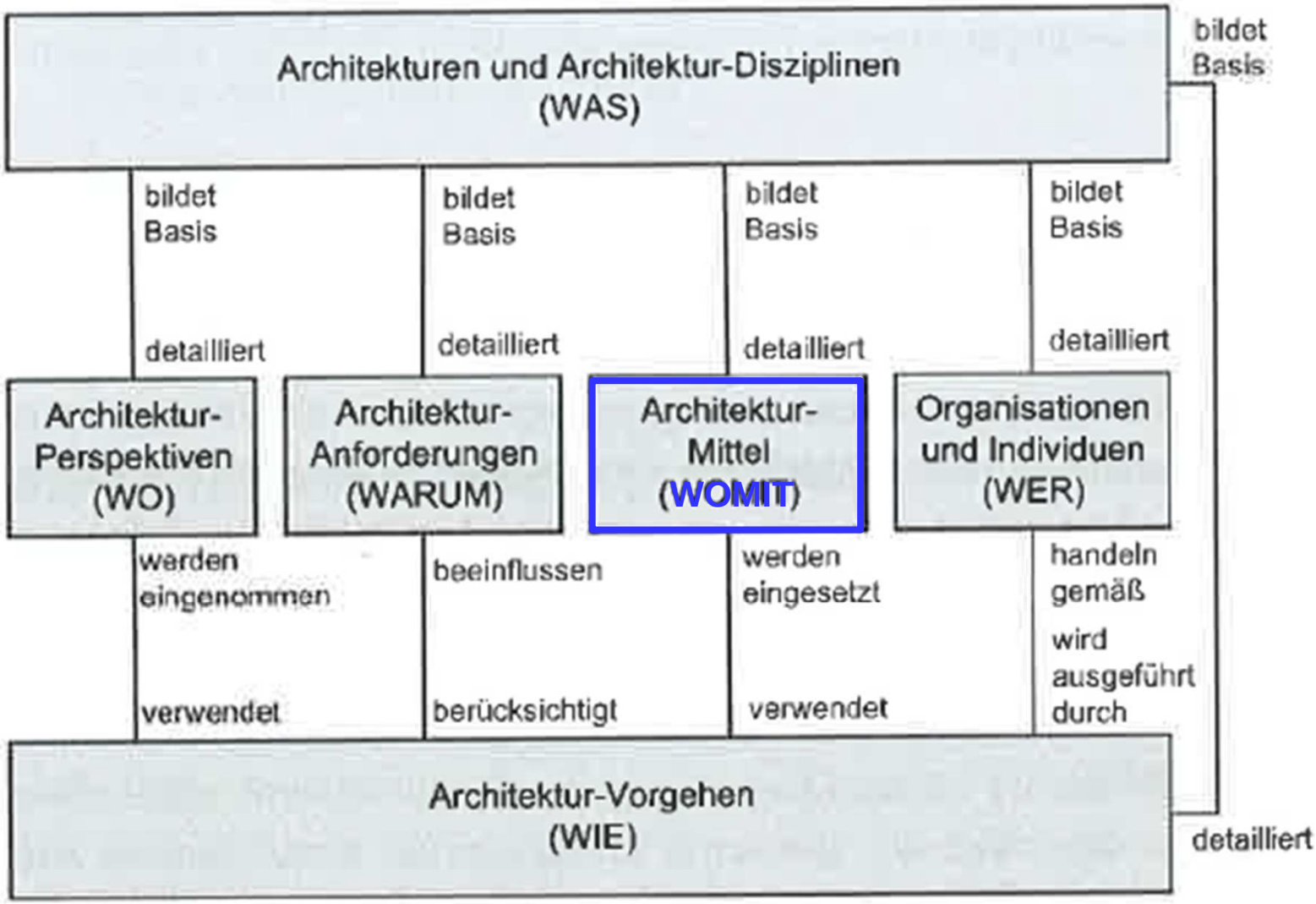
- **4 – WOMIT**
  Architektur-Mittel
  Basisarchitekturen, Referenzarchitekturen, …
  Modellierung(smittel); Technologien, …

- **5 – WER**
  Architektur-Organisation und Rollen

Architektur-Mittel - WOMIT

Architektur - Mittel
WOMIT

Architektur-Mittel - WOMIT-42



O.Vogel et.al., Software-Architektur, Grundlagen-Konzepte-Praxis, Spektrum, 2009

Architektur-Mittel - WOMIT-43

Was meint Mittel und WOMIT:

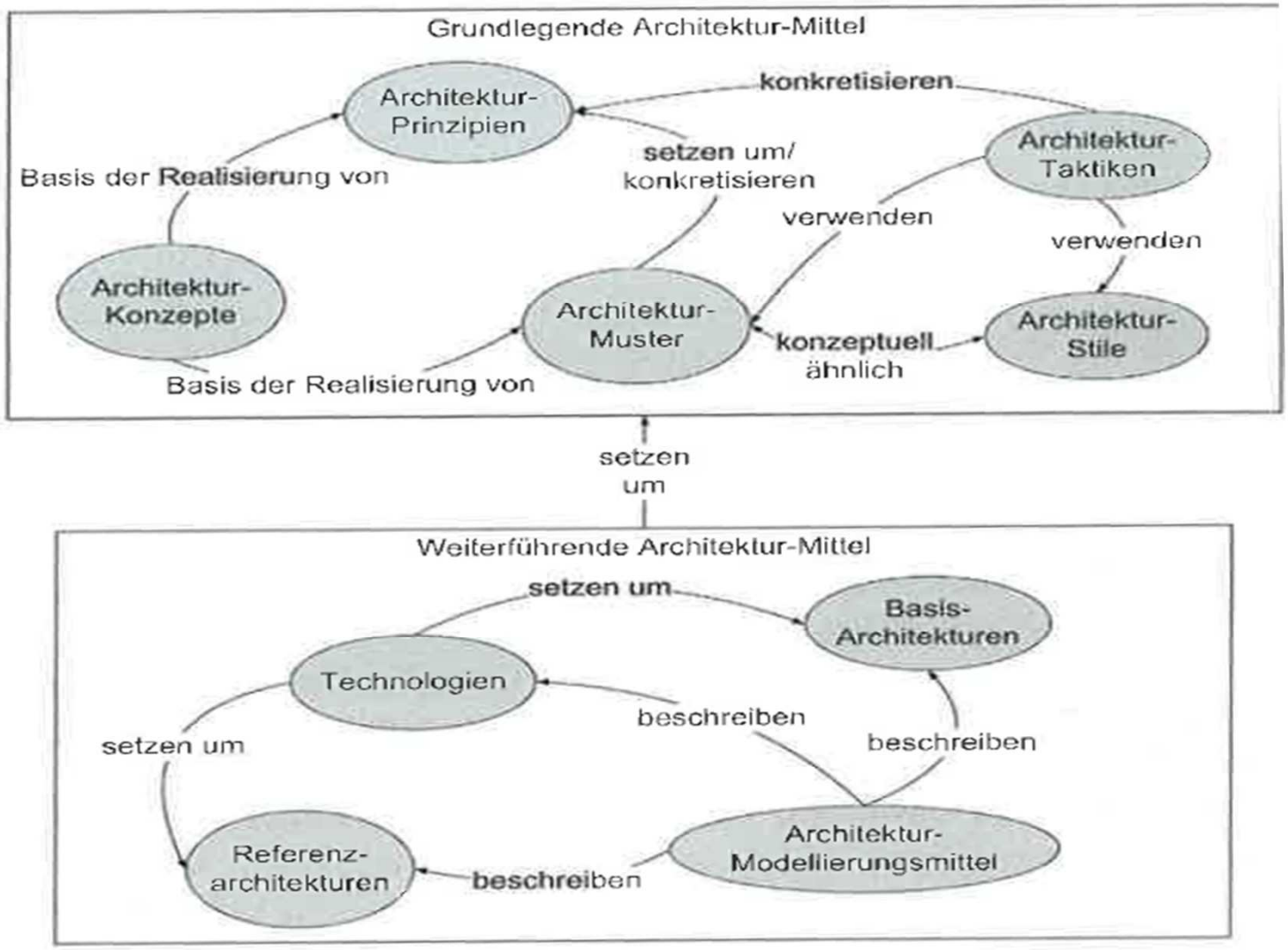
- Werkzeugkasten des SW-Architekten
- Werkzeuge: Konzepte, Techniken, …

Zwei Gruppen von Architektur-Mitteln:

- Grundlegende Architektur-Mittel
  in zunehmender Konkretisierung:
  Prinzipien, Konzepte, Taktiken, Arch.-Stile und -Muster
- Weiterführende Architektur-Mittel
  in zunehmender Konkretisierung:
  Basisarchitekturen (z.B. Schichtenarchitektur),
  Technologien, Modellierungsmittel (z.B. UML, DSL, …),
  Referenzarchitekturen

Architektur-Mittel - WOMIT-44

SAD Mittel

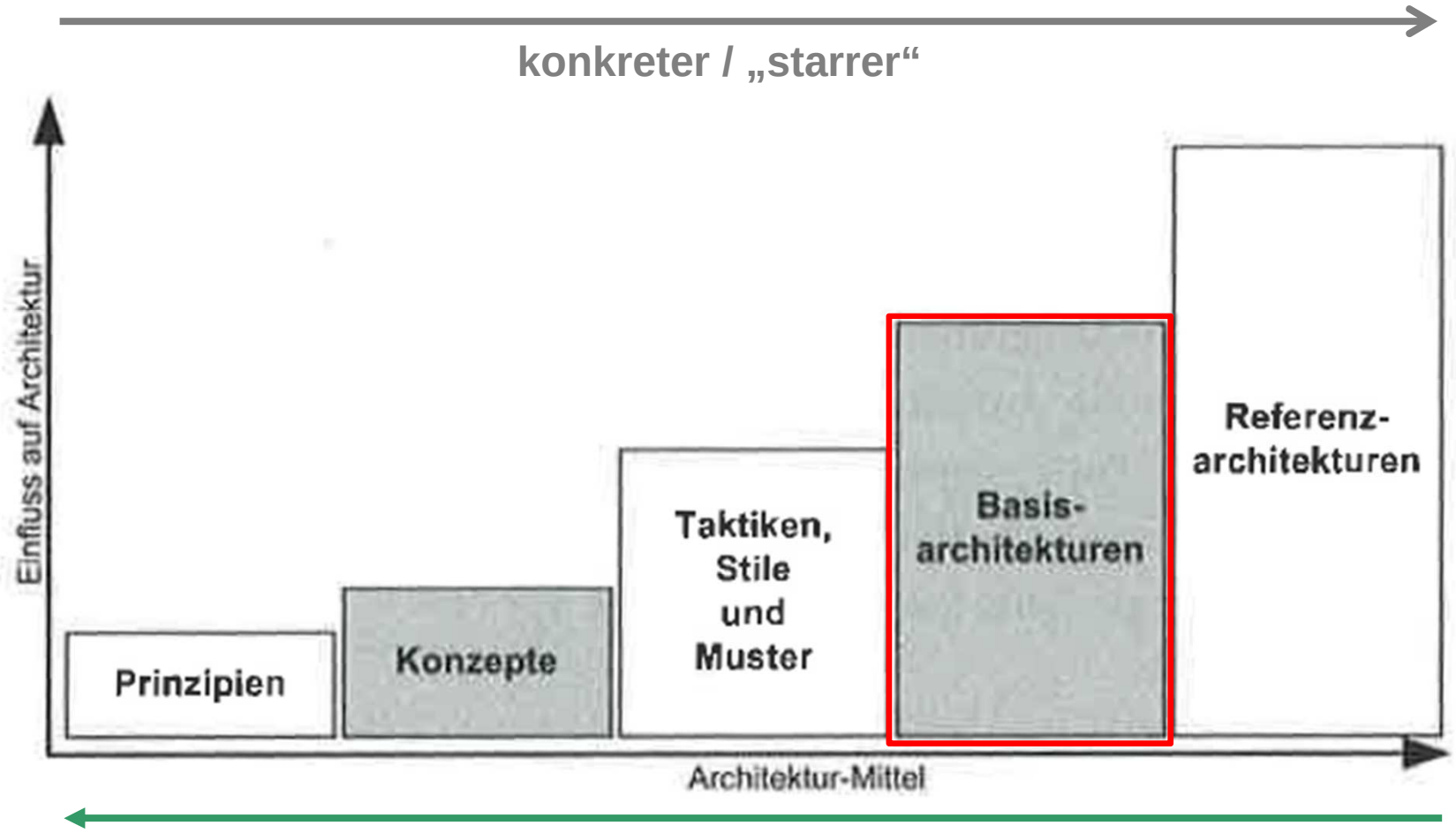


O.Vogel et.al., Software-Architektur, Grundlagen-Konzepte-Praxis, Spektrum, 2009

Architektur-Mittel - WOMIT-45

SAD Mittel

Architekturmittel und Einfluss auf die Architektur:



O.Vogel et.al., Software-Architektur, Grundlagen-Konzepte-Praxis, Spektrum, 2009

Arc.Mittel/Basisarchit.-WOMIT-46

SAD Mittel

Basisarchitekturen:

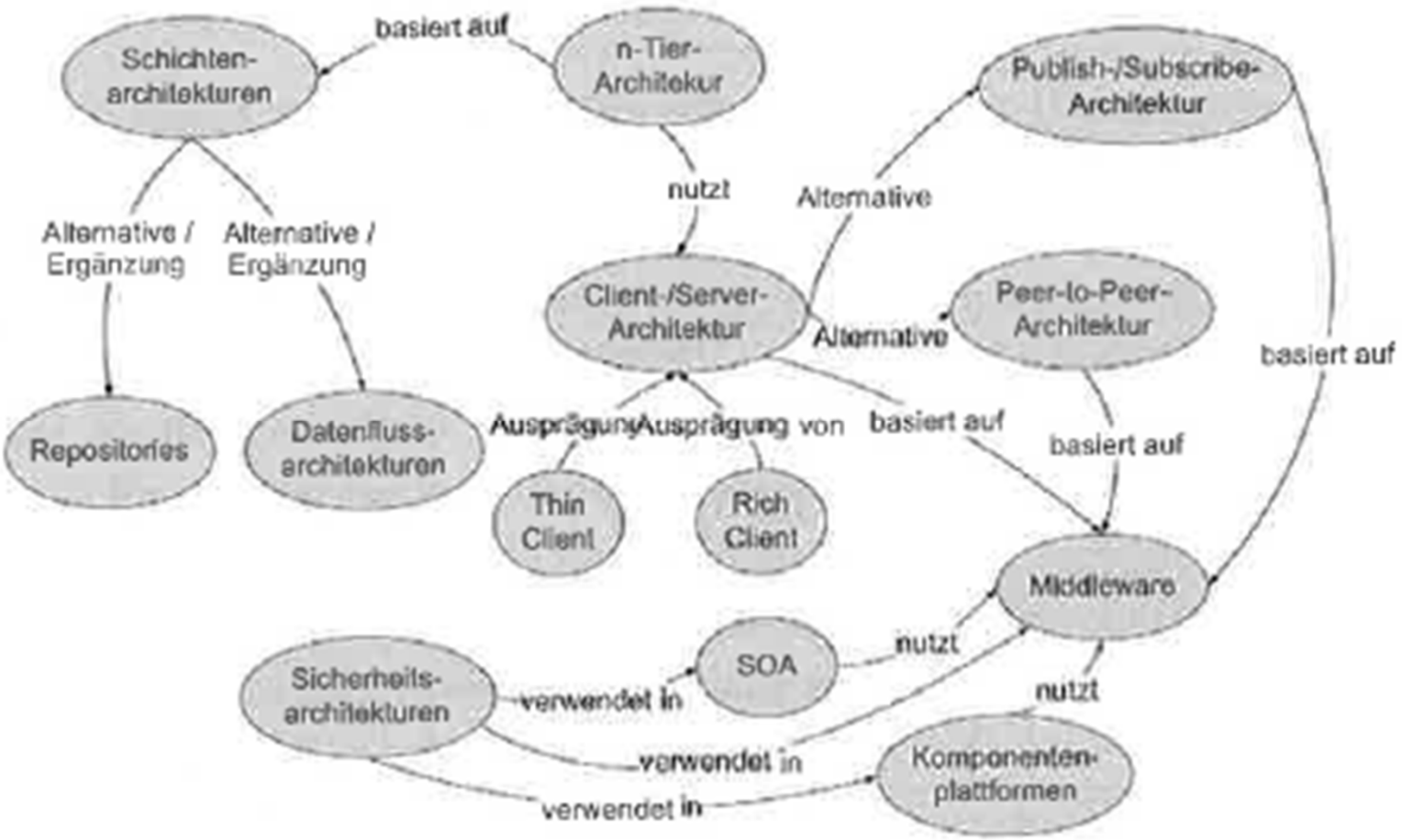
- Schichtenarchitekturen
- Datenflussarchitekturen
- Repositories
- Zentralisierung/Dezentralisierung
- n-Tier-Architekturen
- Rich Client/Thin Client
- Peer-to-Peer-Architekturen
- Publish/Subscribe Architekturen
- Middleware
- Komponentenplattformen
- Service Orientierte Architekturen (SOA) – (*)
- Sicherheitsarchitekturen - (*)
- …

(*) sind (etwa im Vergleich zu Schichtenarchitekturen) „grössere" bzw. „umfangreichere" Basisarchitekturen

Arc.Mittel/Basisarchit.-WOMIT-47

SAD Mittel

Basisarchitekturen - Übersicht:



O.Vogel et.al., Software-Architektur, Grundlagen-Konzepte-Praxis, Spektrum, 2009

Arc.Mittel/Basisarchit.-WOMIT-48

SAD Mittel

Basisarchitekturen:

- Einsatz der Architektur-Mittel (siehe im vorangegangenen Stoff zu **WOMIT**)
- Sind konkretere Architektur-Mittel, mit denen man Systeme ganzheitlich strukturieren kann bzw. sind Vorlagen

Der **MONOLITH** als die schlimmste Form der Architektur – alles (unstrukturiert) in einem Systembaustein mit erheblichen Nachteilen und Problemen verbunden:

- Prinzipien, Konzepte, … lassen sich damit kaum umsetzen
- Alles bzw. Wichtiges "hard coded"
- Geschäftslogik ist über den (gesamten) Code "verteilt"
- Findet sich in/bei Altsystemen
- Typus der SW: "Spaghetti"-programmiert; SW der "1000+1 Balkone"
- Anpassungen sind schwierig
- Kleine Änderungen -> Grosser Aufwand
- Nur jene, die das System implementiert haben kennen sich aus …

Arc.Mittel/Basisarchit.-WOMIT-49

Basisarchitekturen:

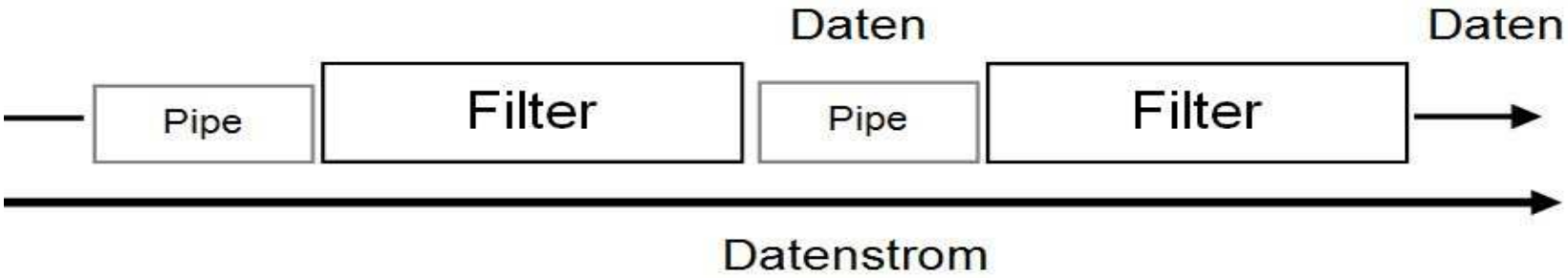
- **Schichtenarchitekturen - Schichtenbildung:**
  - Unterteilung von Systemen in Gruppen von Bausteinen mit ähnlichen Funktionen bzw. Verantwortlichkeiten
  - Gruppen sind in Layers organisiert
  - Innerhalb eines Layers können die zugehörigen Bausteine frei agieren – "horizontal"
  - Über Layers hinweg können nur unmittelbar benachbarte Schichten aufeinander zugreifen (oft auch nur unidirektional): nur über ganz definierte Schnittstellen zwischen den Layers
  - Jede Schicht bietet Dienste für die höhere Schicht an;

    Schicht n hängt nur von den darunter liegenden Schichten ab („Strict layering" oder „Layer Bridging")
- **Hauptziele der Schichtenbildung:**
  - Veränderbarkeit des Systems erhöhen
  - System portierbar gestalten
  - Wiederverwendbarkeit des Systems erhöhen

Basisarchitekturen:

Datenflussarchitekturen:

- Batch-Sequential-Stil:
  Gesamtaufgabe wird in Teilschritte zerlegt – jeder Teilschritt wird durch einen unabhängigen Baustein abgearbeitet – Daten werden von Baustein zu Baustein übergeben
- Pipes-and-Filters Muster:
  Verarbeitung von Datenströmen …



Filter: Liest und verarbeitet Daten;
Umwandlung Dateneingabe in Datenausgabe
Pipe: Leitet Daten von Filter zu Filter weiter

Beispiel: Java-Servlet-Filter (als Filter: Logging, Verschlüsselung, Entschlüsselung, Komprimierung, Dekomprimierung, …)

Arc.Mittel/Basisarchit.-WOMIT-52

Basisarchitekturen:

- **Repositories:**
  - Hauptaufgabe von Repositories: verschiedenen Bausteinen den gleichzeitigen Zugriff auf Daten zu ermöglichen
  - Repository zentral – stellt APIs für den Datenzugriff zur Verfügung – Locking-Mechanismen, Transaktionsmethoden
  - Einfaches Beispiel: Metadaten-Repository für Enterprise-Architekturen
- **Zentralisierung versus Dezentralisierung:**
  - Zentralisierung: bündeln von Aufgaben auf einen Baustein
  - Dezentralisierung: verteilen von Aufgaben auf mehrere Bausteine
  - Sind zwei grundlegende Aspekte (Ausgleich/Kompromiss wichtig): (z.B. Repositories: Datenhaltung zentral, Clients dezentral
  - dezentral: weist auf verteilte Systeme hin – Systembausteine sind auf Rechner/Prozessoren/Prozesse verteilt … dezentrale Entwicklungen: z.B. File Sharing
  - Vorteile – Dezentralisierung: niedrigere HW-Kosten; bezüglich HW flexibler (Ausfall)
  - Vorteile – Zentralisierung: besser in den Griff bekommbar sind z.B. Sicherheit, Logging, Kontrolle, Monitoring, Deployment neuer SW(-Teile), Datenverwaltung, …
  - Client/Server Modell bzw. Systeme: auf Clients werden Anwenderprogramme betrieben, welche auf Ressourcen des Servers zugreifen bzw. diese nutzen

Basisarchitekturen:

Rich Client versus Thin Client:

- Es geht um die Aufteilung der Funktionalitäten zwischen Client und Server
- Thin Client: "nur" Konsole; Rich Client: viel Funktionalität (und Berechnungen) auf dem Client
- Kriterien und Überlegungen zur Abwägungen ob „Thin" oder „Rich" Client:
  - Thin Clients belasten Server (Ressourcen) mehr
  - Netzwerkabhängigkeit und –auslastung bei Thin Clients höher
  - Auslieferung / Update / Austausch von SW(-Teilen) am Server leichter (zentraler) möglich
  - Rich Clients: oft müssen (clientlokale)Daten am Server oft aktualisiert (synchronisiert) werden
  - Thin Clients: Webbrowser
  - Mittelding/Kompromiss: z.B. Web 2.0, Ajax

Basisarchitekturen:

- **n-Tier-Architekturen:**
  - es gibt 2,3,5, … n-tier Architekturen
  - klassische Client/Server-Architektur: 2-tier
  - die Einführung weiterer Schichten resultiert prinzipiell aus Anforderungen betreff:
    - hohe Lastzahlen
    - hohe Anzahl von "concurrent usern"
    - hohe Sicherheitsanforderungen
    - hohe Ausfallsicherheitsanforderungen
    - hohe Zuverlässigkeitsanforderungen

    z.B. Einziehen einer Schicht zur Ansteuerung redundanter Server

… aufgrund der zentralen Bedeutung von Schichtenarchitekturen folgen an dieser Stelle einige, weitere Folien als Vertiefung …

=>

Arc.Mittel/Basisarchit.-WOMIT-55

SAD Mittel

Basisarchitekturen:

■ Schichten = Layers (= Tiers)

Jede Schicht hat „definierte" Rolle. Anwendung nach klaren **Regeln**:

- Obere Schicht verbirgt darunterliegende Schichten und interne Komplexität – saubere Trennung der Schichten (unterstützt verteiltes Arbeiten in Teams)
- Top-down Kommunikation – Komponenten der höheren Schicht verwenden Dienste der unteren Schicht und nicht umgekehrt (*z.B. Einsatz des Facade-Patterns für Zugriff auf Dienste*)
- Komponenten innerhalb einer Schicht sind von ähnlichem Abstraktionsgrad (z.B. Persistenz betreff Daten-Schicht)
- Design einer Schicht soll lose Kopplung zu anderen Schichten ermöglichen
- Kommunikation zwischen den Schichten erfolgt über klar definierte Schnittstellen und Protokolle (kein „unkontrolliertes Reingreifen")

„Best Practice Software-Engineering", Alexander Schatten et.al, 2010

Arc.Mittel/Basisarchit.-WOMIT-56

SAD Mittel

Basisarchitekturen:

- **„Beliebtheit" bzw. „Bewährtheit" der Schichtenarchitektur:**
  - relativ einfach, verständlich UND **flexibel**
  - Schichtenarchitektur stellt ein einfaches und effizientes **Architekturmuster** dar
  - Schicht **kann als eigenständige Programmkomponente** gesehen werden (ohne die Gesamtarchitektur zu kennen)
  - **Schicht kann als Subsystem** entworfen werden, das wieder aus Teilsystemen besteht (z.B. Services, Geschäftsprozesse,..)
  - bei Einhaltung der Regeln: **minimale Abhängigkeiten von Schichten, (leichte) Austauschbarkeit von Schichten** (auch in Bezug auf Infrastrukturen wie HW, OS, …), **gute Wartbarkeit und Erweiterbarkeit**
- **Schichtenbildung horizontal und vertikal**:
  - horizontal: abgegrenzte Aufgaben (z.B. Datenzugriffsschicht)
    (Top-down Zugriff im Stapel der horizontalen Schichten)
  - vertikal: Querschnitts-Funktionalitäten (mit Vollzugriff auf andere Schichten – z.B. Data Transfer Schicht)

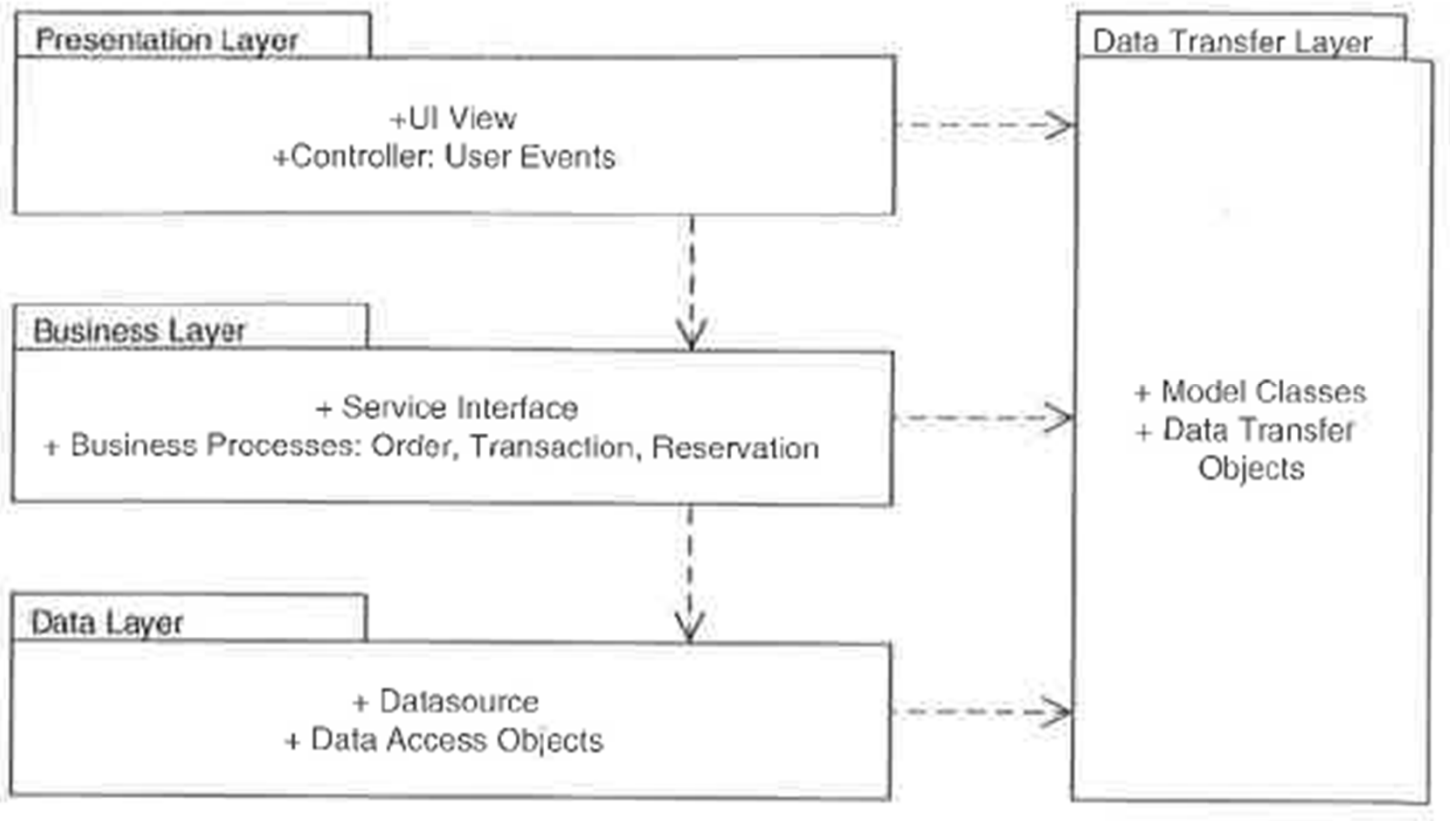
„Best Practice Software-Engineering", Alexander Schatten et.al, 2010

Arc.Mittel/Basisarchit.-WOMIT-57

SAD Mittel

Basisarchitekturen:

- **Schichtenarchitektur – allgemeineres Bspl.**



„Best Practice Software-Engineering", Alexander Schatten et.al, 2010

Arc.Mittel/Basisarchit.-WOMIT-58

Basisarchitekturen:

- **Nachteile der Schichtenarchitektur**
  - Durchreichen durch Schichten: eventuell komplexe Funktionen für triviale Operationen (gezwungenermassen) erforderlich
  - Mehr Schichten bedeuten mehr Komplexität und Implementierungsaufwand
  - Weniger Schichten bedeuten stärkere Kopplung und weniger Flexibilität
  - Änderungen in Schnittstellen der unteren Schichten können Änderungen in oberen Schichten bedeuten
- **Anzahl der Schichten:**
  - Abhängig von konkreten Anforderungen und Komplexität
  - Abwägen der Vor- und Nachteile (siehe oben)
    -> ausbalancieren und angemessen …

„Best Practice Software-Engineering", Alexander Schatten et.al, 2010

Arc.Mittel/Basisarchit.-WOMIT-59

SAD Mittel

Basisarchitekturen:

- **2-Schichtenarchitektur:**
  - Client/Server Architektur bei verteilten Systemen
    z.B. Client beinhaltet GUIs und Business Logic („fat client")
- **3-Schichtenarchitektur:**
  - Presentation Layer (GUIs)- *Präsentations-Schicht*
    Benutzerinteraktionen, Logik der Bedienelemente von GUIs
  - Business Layer (Kernfunktionalität) - *Logik-Schicht*
    eventuell nochmals unterteilt in ***Data Access Layer (DAOs)*** und ***Service Layer***
  - Data Layer (Daten) - *Daten-Schicht*
    Persistierung von Daten
- **5-Schichtenarchitektur:**
  - Schichten wie bei 3-Schichtenarchitektur
  - dazu noch Splitting der Business Layer in/nach:
    - Data Access Layer, Data Mapping, Service Layer
  - Zwischen Logik und Präsentationsschicht:
    - Prozess-Schicht (Services atomar -> wiederverwendbar)

„Best Practice Software-Engineering", Alexander Schatten et.al, 2010

Arc.Mittel/Basisarchit.-WOMIT-60

Basisarchitekturen:

- **Peer-to-Peer-Architektur:**
  - Client/Server: jeder Client kommuniziert (explizit) mit einem Server
  - Peer-to-Peer (P2P): es gibt keinen zentralen Server – jeder Peer kann im Netz Services anbieten und konsumieren
  - Der Gesamtzustand des Systems ist über die Peers verteilt
  - Services können hinzugefügt und weggenommen werden (daher: Peers müssen sich Informationen holen, welche Services es aktuell gibt
  - es gibt reine P2, aber auch viele hybride Systeme
- **Publisher/Subscriber (P/S)-Architektur:**
  - Publish/Subscribe (ist u.a. ein Muster) ermöglicht es allgemein einem Ereigniskonsumenten, sich für bestimmte (interessierende) Ereignisse zu registrieren. Wenn ein solches ereignis eintritt, informiert der Produzent des Ereignisses alle registrierten Konsumenten mithilfe eines Publish/Subscribe-Systems. Damit wird eine Entkopplung von Produzenten und Konsumenten von Ereignissen erreicht.
  - P/S entspricht dem Implicit-Invocation-Stil im Gegensatz zu Client/Server und P2P, die dem Explizit-Invocation-Stil entsprechen

Basisarchitekturen:

Middleware:

- (Kommunikations-)Middleware ist eine Plattform, die Anwendungen Dienste für alle Aspekte der Verteilung anbietet:
  verteilte Aufrufe, effiziente Zugriffe auf Netzwerke, Transaktionen, ..
  Middleware deckt "Basis-Sachen" gut ab und Entwickler können sich auf die Implementierung des fachlich Geforderten konzentrieren
- verteilte Systeme umgesetzt z.B. auf Basis des Broker Musters
- Zusammenarbeit von räumlich verteilten Partnern über Netzwerke
- verteilte Systeme: stemmen hohe Systemlasten; höhere Performanz, Skalierbarkeit; höhere Fehlertoleranz
- vielfältige Einsätzgebiete: Internet-Systeme, Telekommunikationsnetzwerke, embedded systems, …
- Herausforderungen: Aufrufzeiten länger (als bei lokalen Aufrufen), Vorhersagbarkeit der Aufrufzeiten, Nebenläufigkeit (-> nichtdeterministisches Verhalten; Deadlocks), Skalierbarkeit des Systems (Hochlast…), teilweiser Systemausfall
- "Schichtenverhalten": Middleware darf nicht umgangen werden
- Verteilungsstile in Middlewaresystemen: RPC/Messaging/Streaming Systeme

Basisarchitekturen:

- **Komponentenplattformen:**
  - im Enterprise-Umfeld – z.B. JEE, .NET
  - Basieren auf Trennung von technischen und fachlichen Belangen
  - Technische Belange im Enterprise-Umfeld: Verteilung, Sicherheit, Persistenz, Transaktionen, Ressourcenmanagement, …
  - Die technischen Belange werden vom "Container" als zentralem Baustein automatisch übernommen
  - Fachliche Anforderungen werden durch die Komponenten realisiert (z.B.: Entitäten-persistent, Sessions, Services)
  - Der Container kontrolliert die lebenszyklen von Komponenteninstanzen (Optimierung der Ressourcen) – Lebenszyklusoperationen: *aktivieren, zerstören, passivieren, …*
  - *…*
  - viele Komponentenplattformen sind in Application Server integriert
  - Wiederverwendung(en): nur blackbox-artig bzw. -basiert

Arc.Mittel/Basisarchit.-WOMIT-63

SAD Mittel

Basisarchitekturen:

- **Service-Orientierte-Architektur (SOA):**
  - Basisarchitektur, welche die fachlich funktionalen Schnittstellen von Software-Bausteinen als wiederverwendbare, verteilte, lose gekoppelte und standardisiert zugreifbare Dienste repräsentiert …
  - Services (Dienste) zeichnen sich in einer SOA durch folgende Eigenschaften aus:
    - sind stärker strukturiert als Komponenten(schnittstellen)
    - kommunizieren technologieneutral und standardisiert mit synchronen und asynchronen Nachrichten
    - erlauben anonyme Nutzung (Client und Service lose gekoppelt)
    - sind in gewissem Ausmaß selbstbeschreibend
    - sind idempotent (liefern reproduzierbare Ergebnisse), zustandsfrei, transaktional abgeschlossen
    - bestehen aus der Service-Schnittstelle (“Vertrags”-Template) und Schnittstellenimplementierung (ist austauschbar)
    - Einsatzszenarien: heterogene Situationen auf technischer + fachlicher Ebene (Bspl.: Fusion von Unternehmen+Infrastrukt.)

Arc.Mittel/Basisarchit.-WOMIT-64

Basisarchitekturen:

- **Service-Orientierte-Architektur (SOA):**
  - besitzen Infrastrukur-Bausteine: Bus, Repository (Umsetzung oft durch Middleware -> SOI: Service Oriented Infrastructure
  - Bus: unterstützt die Übermittlung von Nachrichten Repository: dient der Registrierung des Service-Anbieters beide Bausteine sind konfigurierbar …
  - SOA-Ansatz erlaubt insbesonder das Herauslösen von nicht-funktionalen Aspekten sowohl aus dem Service-Konsumenten, wie auch dem Service-Anbieter
  - elementare SOA: kann auf Basis von Komponentenplattformen realisiert werden (wie z.B. CORBA, RMI, .NET) – SOIs haben oft noch weitergehende Fähigkeiten: komplexe Dienste und Orchestrierung von Diensten, Unterstützung von Internetstandards, Unterstützung von Enterprise Application Integration, Injektionspunkte für Aspekte (z.B. Sichrheit), Bausteine für SAO Governance, Bausteine zur Unterstützung von Prozess(analysen), …

Basisarchitekturen:

Service-Orientierte-Architektur (SOA):

- SOAs können in sich wieder geschichtet aufgebaut sein (z.B. vorteilhaft zur Auftrennung nach: geschäftsprozess-spezifischen Services und geschäftsprozess-neutralen Basis-Services)
- SOA unterstützt Wiederverwendbarkeit (von Services)
- SOA "deckt" Enterprise ab (kann sehr umfangreich sein) -> "Gefahr" hoher Komplexität

- Wichtige Aufgabe des Architekten bei SOA-Ansatz: Dekomposition in sinnvolle und wiederverwendbare Services
- Güte von SOA-Entwürfen stark abhängig vom fachlichen Zugang und fachlichem Verständnis + Erfahrung der beteilgten Experten
- Modellierungsansätze: top-down, bottom-up, meet-in-the-middle)

Arc.Mittel/Basisarchit.-WOMIT-66

SAD Mittel

Basisarchitekturen:

- **Service-Orientierte-Architektur (SOA):**
  - „neues" Paradigma der Architektur
  - Technik-Sichtweise: Technik, Architektur, Problemlösungen
    Management-Sichtweise: Schnelle und kosteneffiziente Lösungen
  - Verbindung bzw. Zusammenführung der beiden Sichtweisen
  - SOA ist keine Technologie und kein Produkt
  - SOA Architekturen sind eine Zusammenfassung von Design-Patterns und Architekturansätzen über die letzten 20 Jahre hinweg
  - SOA Realisierung/Umsetzung: z.B. mit Web Services
  - „Wurzeln" von SOA: Programmiersprachen, Plattformen, Business Computing (Abwicklung der Geschäftsprozesse – ERP, CRM)
  - Ursachen: für Geschäftsprozesse viele Insellösungen; heterogene IT-Systeme; Nutzung von Daten und Services über Systemgrenzen hinweg
  - Bestandteile von SOA:
    - Zentraler Bestandteil ist: Service
    - Zentrale „Akteure": Service Consumer, Service Provider
    - Zentrale Service-Zentrale: Service Repository

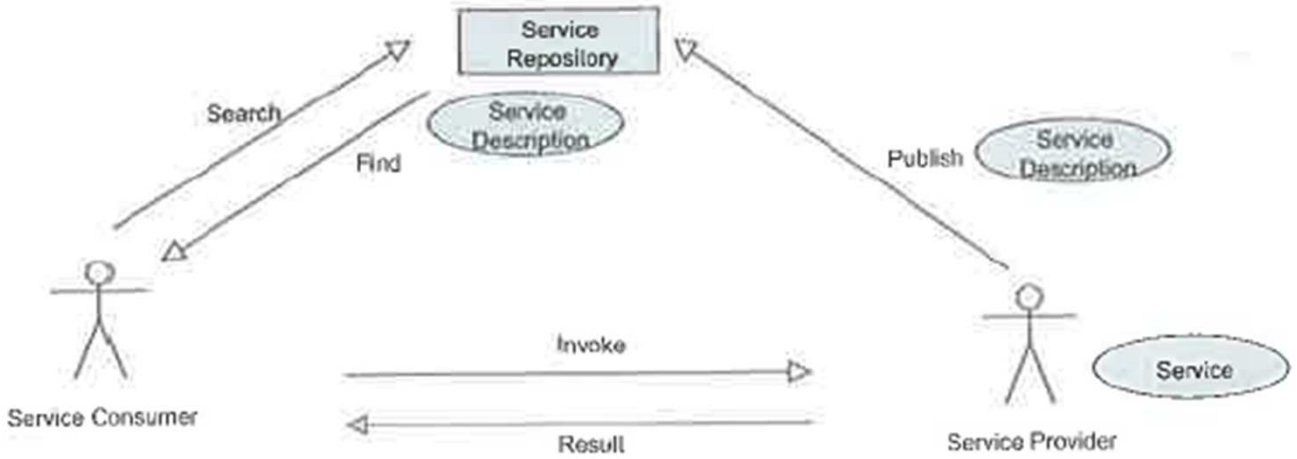
„Best Practice Software-Engineering", Alexander Schatten et.al, 2010

Arc.Mittel/Basisarchit.-WOMIT-67

SAD Mittel

Basisarchitekturen:

- **Service-Orientierte-Architektur (SOA):**



Plattformunabhängiger Primär-Ansatz von SOA:

- Services, Beschreibung der Services, Transportprotokolle: werden in einer plattformunabhängigen („neutralen") Weise gehalten, sodass keine bestimmte IT-Infrastruktur oder SW-Plattform vorausgesetzt wird (z.B. Verwendung von SOAP, WSDL, …)
- Service Provider und Consumer können in beliebiger Technologie implementiert werden

„Best Practice Software-Engineering", Alexander Schatten et.al, 2010

Arc.Mittel/Basisarchit.-WOMIT-68

SAD Mittel

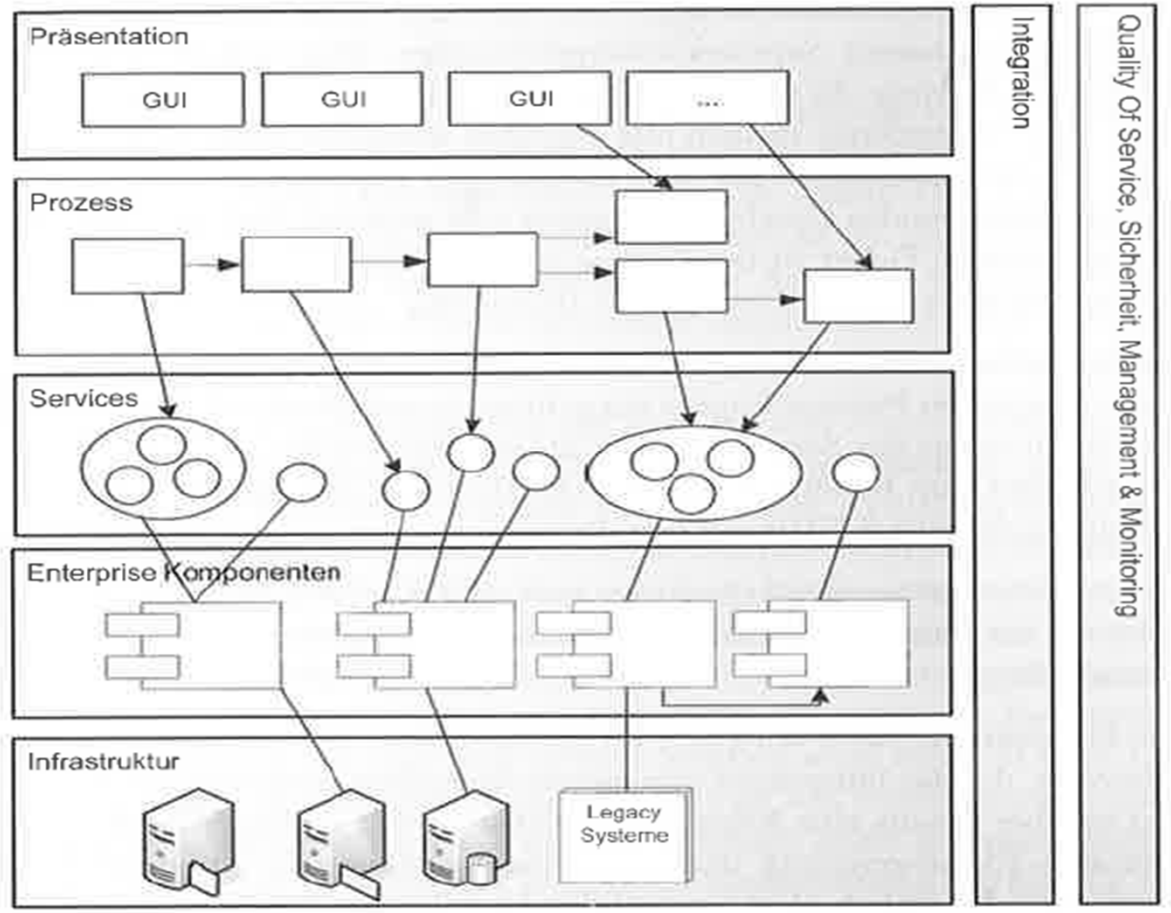
Basisarchitekturen:

- **Service-Orientierte-Architektur (SOA):**

SOA und Schichten:

Schichten in SOA anders verteilt.

Schichten sind oft verteilt auf viele unterschiedliche Rechner und eventuell auch Plattformen.



„Best Practice Software-Engineering", Alexander Schatten et.al, 2010

Basisarchitekturen:

Sicherheitsarchitekturen:

- Sicherheit: ist ein "crosscutting concern" – durchdringender Belang
- Sicherheit ist ein hochgradig verteilter Aspekt (über Baustein hinweg)
- Sicherheitsarchitektur bezieht sich auf:
  - eine zu schützende bzw. sichernde Anwendung
  - Systembausteine, die einer Sicherheitsinfrastruktur zuzurechnen sind
- Voraussetzungen bzw. Vorhandensein von:
  - Sicherheitssysteme auf allen Netzwerkebenen
  - Benutzer+Identitätsverwaltung
  - Authentifizierungs-System
  - Autorisierungs-System (Rollen, Rechte)
  - System zur geschützten Informationsübermittlung (privacy)
  - System zur Gewährleistung von Unbestreitbarkeit (Signaturen)
  - System zur Erkennung von Angriffen und Betriebsüberwachung
- Organisatorische Rahmenbedingungen:
  - Informationssicherheit (im Unternehmen): Schutzklassen; Umgang mit Schützenswertem; Organisationen im Unternehmen betreff Sicherheit
- Übergreifenden Systeme: IAM (Identity Access Management)
  "Sicherheitssysteme": bestehen nicht nur aus SW (alleine)

Basisarchitekturen:

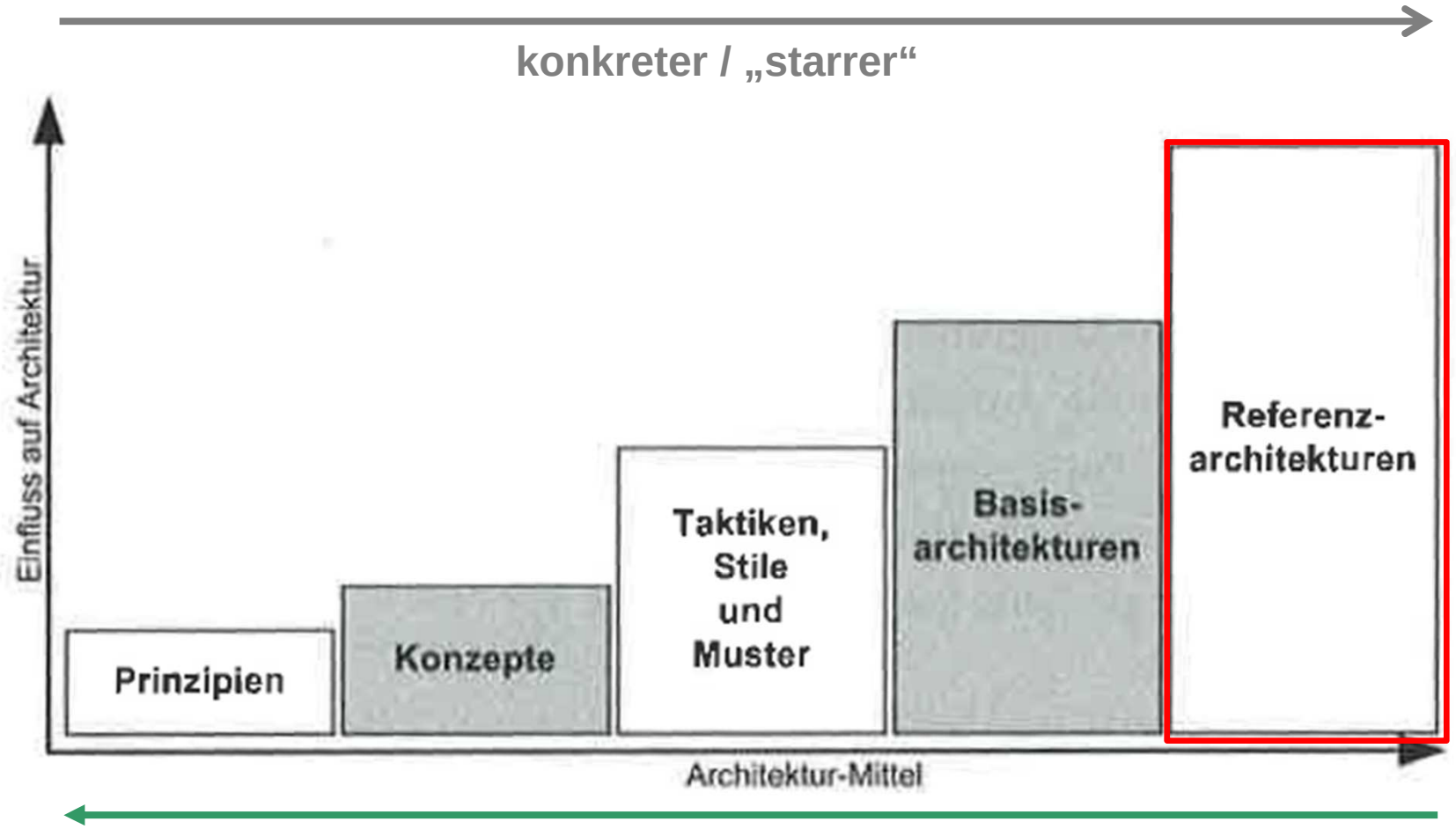
Sicherheitsarchitekturen – Merkmale und Aspekte:

- Leistungsmerkmale von Sicherheitsarchitekturen bzw. -Systemen: privacy, integrity, authentication, authorization, non-repudiation (unleugbar), intrusion protection
- generell: Umsetzungen von Leistungsmerkmalen sollten übergreifend und effizient sein
- Ansätze der Realisierung von Sicherheitssystemen:
  - "Eigenbau": schwerer integrierbar, nicht standardisiert, weniger erprobt
  - "Standarddienste-basiert": Aufsetzen und Verwedung von standardisierten (und bewährten) Lösungen/Infrastrukturen (z.B. LDAP; PKI) – Vernetzung sicherheitsrelevanter Informationen findet aber immer noch auf Anwendungsebene statt
  - Komponentenplattform-basiert": Sicherheitsfunktionalität wird "breit" bereitgestellt; vorteilhaft: wenn auch noch leicht erweiterbar
- Weitere Ansätze und Umsetzungen:
  - Klientseitige Sicherheitsarchitekturen (zentraler Sicherheitsbaustein)
  - Web-zentrische sicherheitsarchitekturen (Schutz über/mittels proxy)
  - Single-Sign-On - Bausteine

Architektur-Mittel - WOMIT-71

SAD Mittel

Architekturmittel und Einfluss auf die Architektur:



O.Vogel et.al., Software-Architektur, Grundlagen-Konzepte-Praxis, Spektrum, 2009

Referenzarchitekturen:

- Architektur soll elegant sein, vordringlich müssen fachliche Anforderungen befriedigt werden …
- „Verbund": Zusammenführung von Architektur-Expertise und industriespezifischer Kenntnisse erforderlich
- Referenzarchitekturen repräsentieren diesen „Verbund"
- Referenzmodell („Klasse") -> Referenzarchitektur („Objekt")
- Vorteile:
  - Aufbauen auf Wissen und Erfahrung
  - Senkung des Risikos von nicht tragfähigen Architekturen
  - Steigerung der Qualität durch Aufsetzen auff /Verwendung von Bewährtem
  - Senkung der Kosten für den konkreten Entwurf (nicht alles neu)
  - Schnellere Entwicklung und schnelleres Time-To-Market
- Anforderungen:
  - müssen basieren auf: Prinzipien, Konzepten, Taktiken, Stilen, Mustern
  - müssen bewährt = erfolgreich eingesetzt worden sein
  - müssen an konkrete Bedürfnisse anpassbar sein
  - müssen (umfassend) dokumentiert sein

Referenzarchitekturen - Einsatz:

- z.B. Muster bieten keine Lösung für einen gesamten Entwurf eines Softwaresystems -> Referenzarchitekturen bieten das …
- Arten von Referenzarchitekturen:
  - Plattformbezogene (Architektur + Bausteine, Code, …)
  - Industriespezifische (zugeschnitten auf Domänen, Unternehmen)
  - Industrieübergreifende (z.B. gemeinsame Architektur ähnlicher SW-Produkte – Produktlinienarchitekturen)
- z.B. für (Industrie-)Unternehmen individuell verfügbar:
  - Anwender mit Individualsoftware (Deutsche Bank, Allianz, BMW, etc.)
  - Technologieanbieter (IBM, SUN, Microsoft, etc.)
  - Lösungsanbieter und Softwarehäuser (Accenture, etc.)

  bieten einheitliche Architekturen für **Softwaresysteme** viele Vorteile …
- Referenzmodelle und –architekturen:
  NGOSS (Next Generation-Operation-Support-Systems-Initiative)
  OSS/J als NGOSS – Implementierung

K.Bergner et. al, Vorlesung "Softwarearchitektur verteilter Systeme", 2002/3, TU-München

Arc.Mittel/Referenzar.-WOMIT-74

SAD Mittel

Referenzarchitekturen:

- Referenzarchitekturen beschreiben eine Vorlage (Blueprint) für die Software- oder / und Systemarchitektur von Softwaresystemen.
- Für die Entwicklung von Softwaresystemen wird dann diese Architektur „kopiert" und gegebenenfalls modifiziert.
- Referenzarchitekturen legen insbesondere fest:
  - Funktionale Aufteilung des Systems unter technischen Gesichtspunkten
  - Technische Trägersysteme, die Rahmen für Applikationsentwicklung darstellen (Middleware, Datenbanken, Standardschnittstellen, etc.)
  - Prozesskommunikationsgrenzen und Ausführungsort von Systemteilen
  - Verwendete Systemsoftware, Netzwerke und Hardware
- Für die Entwicklung von Softwaresystemen wird dann diese Architektur „kopiert" und gegebenenfalls modifiziert

**Oftmals sind Referenzarchitekturen sehr umfangreich.
Es empfiehlt sich daher oftmals nur gewisse Bestandteile umzusetzen, um die Komplexität möglichst gering zu halten.**

Arc.Mittel/Referenzar.-WOMIT-75

SAD Mittel

Referenzarchitekturen:

- ☺ Reduzierung der Entwicklungs- und Produktionskosten
- ☺ Wiederverwendung von Entwurfswissen
- ☺ Geringere Lizenzkosten durch beispielsweise einheitliches Datenbanksystem
- ☺ Geringere Wartungs- und Einarbeitungskosten da einheitliche Umgebung
- ☺ Erhöhung der anwendungsübergreifenden Interoperabilität
- ☺ Anwendungsübergreifende Wiederverwendung von Komponenten
- 😐 Abhängigkeit von Technologie- oder Lösungsanbieter: für Technologie- und Lösungsanbieter ☺ , für Anwender ☹
- ☹ Referenzarchitekturen beschränken sich meist auf technische Aspekte

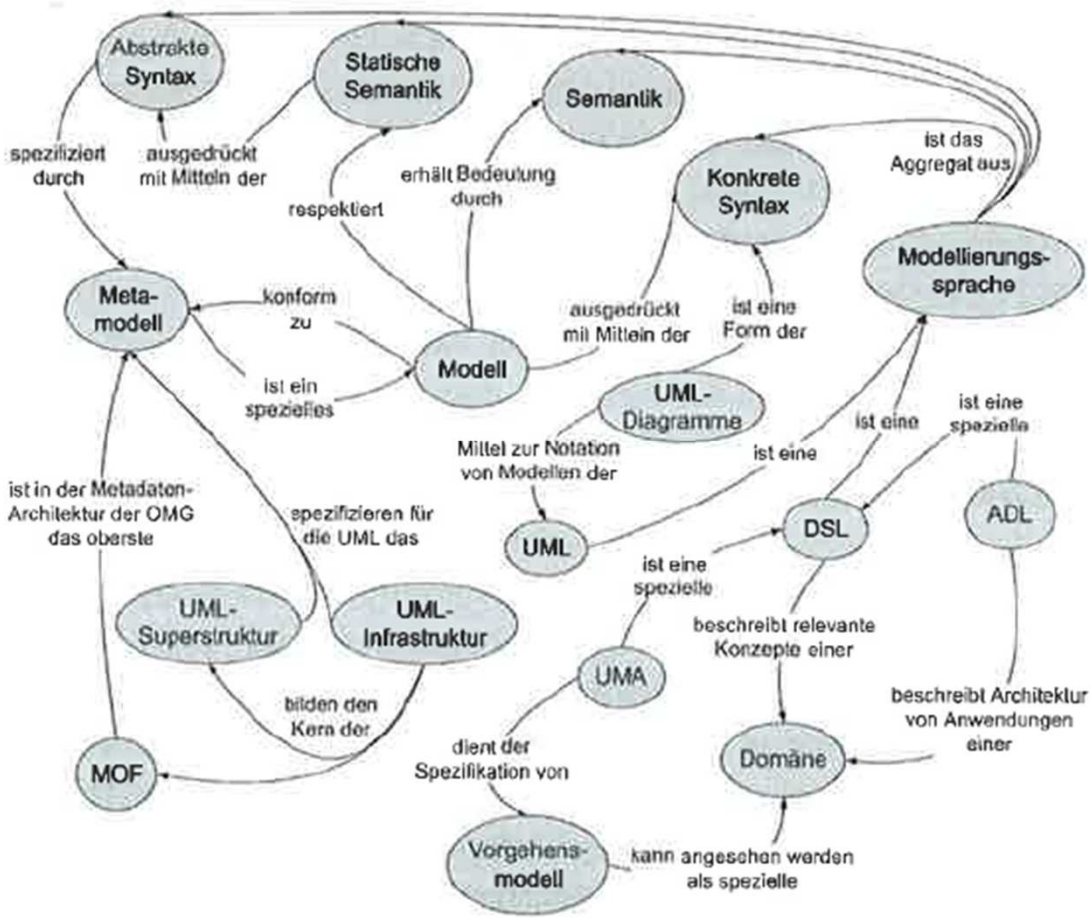
K.Bergner et. al, Vorlesung "Softwarearchitektur verteilter Systeme", 2002/3, TU-München

Arc.Mittel/Modellierg.-WOMIT-76

SAD Mittel

Modellierungsmittel:

- **Konzepte der Modellierung und Modellierungsmittel:**



O.Vogel et.al., Software-Architektur, Grundlagen-Konzepte-Praxis, Spektrum, 2009

Arc.Mittel/Modellierg.-WOMIT-77

Modellierungsmittel:

Grundlagen der Modellierung - Modelle:

- **Modelle** sind ein bewährtes Mittel , um komplexe Sachverhalte übersichtlich darzustellen – werden vielfach eingesetzt
- **Modellbasierte Entwicklung:**
  Anforderungen -> Systementwurf -> Implementierung -> Test
- **Modellierungssprachen** und standardisierte **Notationen** unterstützen geeignete und effiziente Beschreibung von Modelle
- Ein **Modell** ist eine Abstraktion eines beliebigen Originals.
  Es beschreibt die für den Einsatzzweck des Modells relevanten Aspekte des Originals (welche Aspekte hängt von Zielsetzung ab).
  Mithilfe des Modells können bestimmte strukturelle und dynamische Eigenschaften des Originals nachgebildet, analysiert oder simmuliert werden.
- **Beispiel:** U-Bahn-Netzplan ist ein Modell des U-Bahnnetzes – enthält die Kerninformationen über alle Linien und Stationen
  (z.B. reale Position und Abstände zwischen den Stationen ist nicht genau abgebildet)

Ulrike Hammerschal, Gerd Beneken, Software Requirements, Pearson Verlag, 2013

Arc.Mittel/Modellierg.-WOMIT-78

SAD Mittel

Modellierungsmittel:

- **Grundlagen der Modellierung – Sichten:**
  - Eine **Sicht** ist ein (Teil-)Modell, welches das Original aus einer bestimmten Perspektive heraus beschreibt (Unterteilung meist in Fällen von komplexen Systemen). In der Sicht werden nur die Aspekte berücksichtigt, die für die Perspektive relevant sind. Beispiel: spezieller Plan (Planart) zu einer Wohnung (z.B. Elektro).
  - Alle Sichten zusammen bilden das **(Gesamt-)Modell**. Beispiel: alle relevanten Pläne (Planarten) zu einer Wohnung.
- **Modellierungssprachen und Notationen:**
  - **Programmiersprachen:** sind formal (haben formale Semantik)
  - **Modellierungssprachen** können mehr oder weniger formal sein.
- **Arten von Modellierungssprachen:**

Informelle Notationen: z.B. freie Diagramme, Grafiken
„oft nicht exakt genug …"

in der Regel verständlich und einfach anwendbar, gut für frühe „Phasen" des Entwicklungsprojekts: Sprache verstehen (noch) alle stakeholder, Nachteile: Fehlen einer eindeutigen Semantik -> unterschiedliche Interpretationen möglich, Beispiele: Kontextdiagramme, Datenflussdiagramme, Funktionsbaum, …

Ulrike Hammerschal, Gerd Beneken, Software Requirements, Pearson Verlag, 2013

Modellierungsmittel:

Arten von Modellierungssprachen:

Formale Sprachen: bieten formale Spezifikation ihrer Semantik
„oft zu wenig flexibel und komplex…"

- Eindeutige Definition der: *Bedeutung jedes Notationselementes, Verknüpfungsregeln, Ableitungsregeln*
- Mit formaler Sprache entwickelte Modelle sind ihrerseits formal
- Vorteile: erlaubt automatisierte Analyse, Verifikation; Generierungen
- Notation kann textuell und/oder grafisch sein
- Beispiele: Entity-Relationship-Diagramme (ER-Diagramme); Petri-Netze; Z-Notation (ISO/IEC 13568); Standardisierungs-Organisation: FME (Formal Method Europe)
- Verwendet eher in Forschung-für Praxis zu komplex– Einarbeitung aufwendig …

Semiformale Sprachen: zwischen informell und formal …
„guter Kompromiss für die Praxis …"

- es gibt textuelle und grafische Varianten
- Menge der Darstellungselemente der Notation ist vorgegeben + Konzepte und Grammatik + Semantik (nicht vollständig)

Ulrike Hammerschal, Gerd Beneken, Software Requirements, Pearson Verlag, 2013

Arc.Mittel/Modellierg.-WOMIT-80

SAD Mittel

Modellierungsmittel:

Arten von Modellierungssprachen:

Semiformal Sprache: UML

„guter Kompromiss für die Praxis …"

- Beispiele: am bekanntesten ist die UML (Unified Modelling Language) – grafische Notation (13 Diagrammtypen) – zusätzlich: zur Formulierung von Bedingungen und Abhängigkeiten, die mit grafischen Mitteln dabei nicht ausgedrückt werden können, dient eine textuell semiformale Sprache, die OCL (Object Constraint Language)
- UML ist weitverbreitet – dient ihrerseits wieder zur Spezifikation neuer semiformaler Sprachen, wie etwa domänenspezifischer Modellierungssprachen (DSL: Domain Specific Language)
- 1990er Jahre – Objektorientierung (auch) für frühere Phasen im Entwicklungsprozess wie die Anforderungsanalyse:
  federführend: Grady Booch, James Rumbaugh, Ivar Jacobsen
  jeder der Drei brachte wichtiges ein -> Standardisierung der UML Mitte der 90er Jahre
- Ergebnis in Form der UML mit Inhalten:
  Sammlung an grafischen, diagrammartigen Notationen mit gemeinsamer objektorientierter Semantik –
  ist aber streng genommen keine Methode (gibt nicht vor,
  wie die Diagramme einzusetzen sind) – ist stabil und flexibel
- UML: getrieben bzw. unter der Obhut der OMG (Object Management Group)

Ulrike Hammerschal, Gerd Beneken, Software Requirements, Pearson Verlag, 2013

Arc.Mittel/Modellierg.-WOMIT-81

Modellierungsmittel:

- **Modellbildung:**
  Prozeß zur Ableitung eines Modells aus einem Orig.
  - Analyse: das Original ist die Anwendungsdomäne, Entwurf: das Original ist das zu entwickelnde System)
  - Schritte „guter" Modellbildung:
    - Identifikation und Abgrenzung
    - Festlegung des Einsatzzwecks für das Modell
    - Auswahl geeigneter Sichten und Notationen (*)
    - Festlegung der Abbildungsvorschrift
  - (*) SW-Entwicklung - 2 grundsätzliche Arten von Sichten etabliert:
    - dynamische Aspekte (Ablauf, Verhalten, Kommunikation, …)
    - Statische Aspekte (Typen, Klassen, Beziehungen, …)
  - Modelle enstehen durch die Anwendung von Operationen auf bzw. bezüglich der Informationen des Originals – das sind vor allem:
    - Reduktion: unwichtige Details weglassen
    - Abstraktion: relevante Informationen (im Original) verallgemeinern

Ulrike Hammerschal, Gerd Beneken, Software Requirements, Pearson Verlag, 2013

Arc.Mittel/Modellierg.-WOMIT-82

SAD Mittel

Modellierungsmittel:

Modellierungstechniken:

- **Objektorientierte Modellierungstechniken:**
  Objekte: z.B. Person, Auto; Vertrag, Semester; …
  Objekte interagieren:
  „Kunde schliesst mit Gesellschaft Handy-Vertrag"
  … ist schon abstrahierte, allgemeine Formulierung
  Objekte/Klassen haben:
  - Eigenschaften (Attribute)
  - Zustände (Werte)
  - Verhalten (Methoden)
- **UML als bekannteste**, objektorientierte Modellierungssprache: unterstützt die Modellierung von Objekten und Klassen.
  UML unterstützt den ganzen Entwicklungsprozeß – bestimmte Diagrammtypen sind jeweils vorteilhaft. Versionen – UML: >= 2.5
- Alternativ gibt es zu objektorientierten Modellierungstechniken z.B. die Techniken der **Strukturierten Analyse:**
  Funktionsbaum, Datenflussdiagramm, ER (Entity Relationship)-Diagramm, Petri-Netze (Bedingungs-Ereignis-Netze)

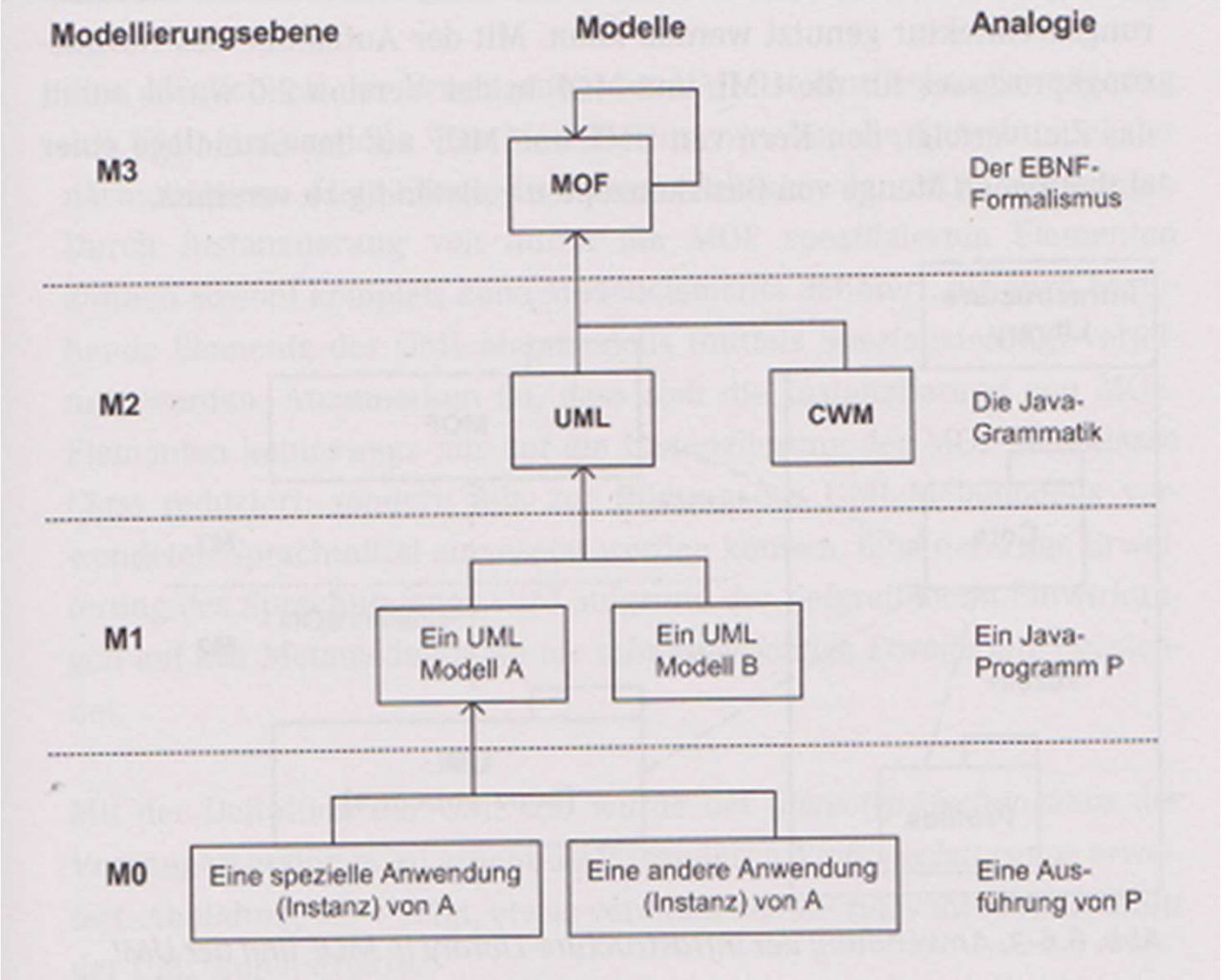
Ulrike Hammerschal, Gerd Beneken, Software Requirements, Pearson Verlag, 2013

Arc.Mittel/Modellierg.-WOMIT-83

SAD Mittel

Modellierungsmittel:

- **Modellierungsarchitektur der OMG - MOF:**



Arc.Mittel/Modellierg.-WOMIT-84

SAD Mittel

Modellierungsmittel:

- **Modellierungsebenen und -Hierarchien:**
  - Architektur-Modell ist KEIN Vorgehensmodell
  - Metamodellierung: Beschreibung von Modellen durch Modelle …
    Ein Metamodell definiert Typen – Modelle sind Instanzen (der Typen) …
  - Mehrschichtige Modellierungsarchitekturen: Meta Object Facility (MOF); ist Basis der Modellierungsarchitektur der OMG (Object Management Group)
- **Weitere Begriffe:**
  - **DSL: Domain Specific Languages (spezailisiert)**; Domänenanalyse; DSL als Mittel zur Formalisierung von Architekturwissen; **UML: „nicht spezialisiert"**
  - ADL: Architecture Description Language (Domäne ist die SW-Architektur)
  - UMA: Unified Method Architecture („Architektonisches Vorgehensmodell")
- **UML – Leitidee; Diagramme; Einsatz:**
  - Leitidee der UML: EIN Modell – verschiedene Sichten:
    verschieden Aspekt werden durch verschiedene Diagrammarten dargestellt
  - Generell: in ein Diagramm nicht zu viele Aspekte reinpacken –>
    Übersichtlichkeit und Aussagekraft gehen verloren

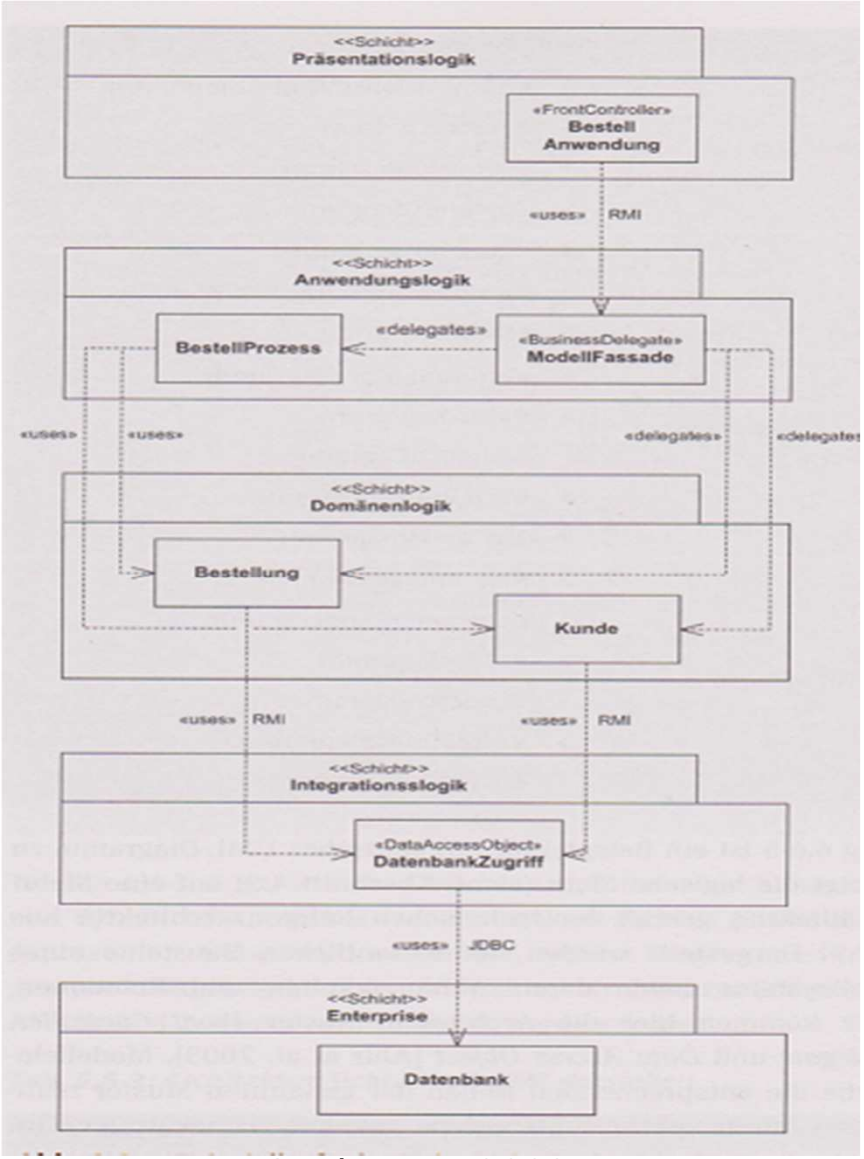
O.Vogel et.al., Software-Architektur, Grundlagen-Konzepte-Praxis, Spektrum, 2009

Arc.Mittel/Modellierg.-WOMIT-85

SAD Mittel

Modellierungsmittel:

- **Logische Sicht – ein Beispiel:**
  Online-Bestell-System
  Verwendung der Muster:
  - Front Controller
  - Business Delegate
  - Data Access Object (DAO)



O.Vogel et.al., Software-Architektur, Grundlagen-Konzepte-Praxis, Spektrum, 2009

Arc.Mittel/Modellierg.-WOMIT-86

SAD Mittel

Modellierungsmittel:

- **Architektonische Bedeutung der UML-Diagramme:**

Diagramm	zeigt	statisch/ dynamisch
Aktivitätsdiagramm	Schritte, die innerhalb eines Systems ablaufen, um eine bestimmte Aufgabe zu erfüllen. Unter Angabe der beteiligten Bausteine.	dynamisch
Anwendungsfall-diagramm	Anwendungsfälle eines geplanten oder existierenden Systems und den daran beteiligten Parteien.	dynamisch

Diagramm	zeigt	statisch/ dynamisch
Interaktionsübersichtsdiagramm	Wann welche Interaktion zwischen Bausteinen abläuft.	dynamisch
Klassendiagramm / Komponentendiagramm	Schnittstellen und Beziehungen von Bausteinen.	statisch
Kommunikationsdiagramm	Bausteine, die zusammenarbeiten bzw. kommunizieren.	dynamisch
Kompositionsstrukturdiagramm	Bausteine hinsichtlich ihrer Schnittstellen und Beziehungen sowie ihres Innenlebens.	statisch
Objektdiagramm	Innere Struktur eines Bausteins zu einem bestimmten Zeitpunkt zur Laufzeit.	statisch
Paketdiagramm	Logische Zusammenfassung von kohäsiven Bausteinen.	statisch
Sequenzdiagramm	Kommunikationsabläufe zwischen Bausteinen.	dynamisch
Timing-Diagramm	Zustände von Bausteinen in Abhängigkeit von der Zeit.	dynamisch
Verteilungsdiagramm	Physikalische Verteilung von Bausteinen zur Laufzeit.	statisch
Zustandsdiagramm	Zustände eines Bausteins und Ereignisse, welche diese Zustände bewirken.	dynamisch

Arc.Mittel/Modellierg.-WOMIT-87

SAD Mittel

Modellierungsmittel:

- **Sichten und UML-Diagramme:**

Architektur-Sicht	UML-Diagramm
Anforderungssicht	> Aktivitätsdiagramm > Anwendungsfalldiagramm > Klassendiagramm > Paketdiagramm > Sequenzdiagramm > Zustandsdiagramm
Logische Sicht	> Aktivitätsdiagramm > Klassendiagramm > Komponentendiagramm
Logische Sicht (Forts.)	> Kompositionsstrukturdiagramm > Paketdiagramm > Sequenzdiagramm > Zustandsdiagramm

Architektur-Sicht	UML-Diagramm
Datensicht	> Klassendiagramm > Komponentendiagramm > Paketdiagramm
Verteilungssicht	> Komponentendiagramm > Paketdiagramm > Sequenzdiagramm > Verteilungsdiagramm > Zustandsdiagramm
Umsetzungssicht	> Klassendiagramm > Komponentendiagramm > Paketdiagramm > Sequenzdiagramm > Verteilungsdiagramm > Zustandsdiagramm

O.Vogel et.al., Software-Architektur, Grundlagen-Konzepte-Praxis, Spektrum, 2009

Arc.Mittel/Modellierg.-WOMIT-88

SAD Mittel

Modellierungsmittel:

- **Anforderungssichten im Detail und UML-Diagramme:**

Sicht	Beschreibung	Notationen
Kontextsicht	Modelliert die Einbettung des Systems in seine Umgebung.	Kontextdiagramm
Funktionssicht	Modelliert die funktionale Einbettung des Systems in die existierende Systemlandschaft.	Funktionsbaum
Struktursicht	Modelliert strukturelle Zusammenhänge der Anwendungsdomäne. Dies sind fachliche Konzepte mit ihren Eigenschaften und Beziehungen.	UML-Klassendiagramm ER-Diagramm
Verhaltenssicht	Modelliert das Verhalten fachlicher Objekte der Anwendungsdomäne.	UML-Zustandsdiagramm Petri-Netz Entscheidungstabellen
Schnittstellen-sicht	Modelliert das Schnittstellenverhalten des Systems.	Use-Case-Diagramm UML-Aktivitätsdiagramm UML-Sequenzdiagramm UML-Zustandsdiagramm

Ulrike Hammerschal, Gerd Beneken, Software Requirements, Pearson Verlag, 2013

Arc.-Mittel/Technolog.-WOMIT-89

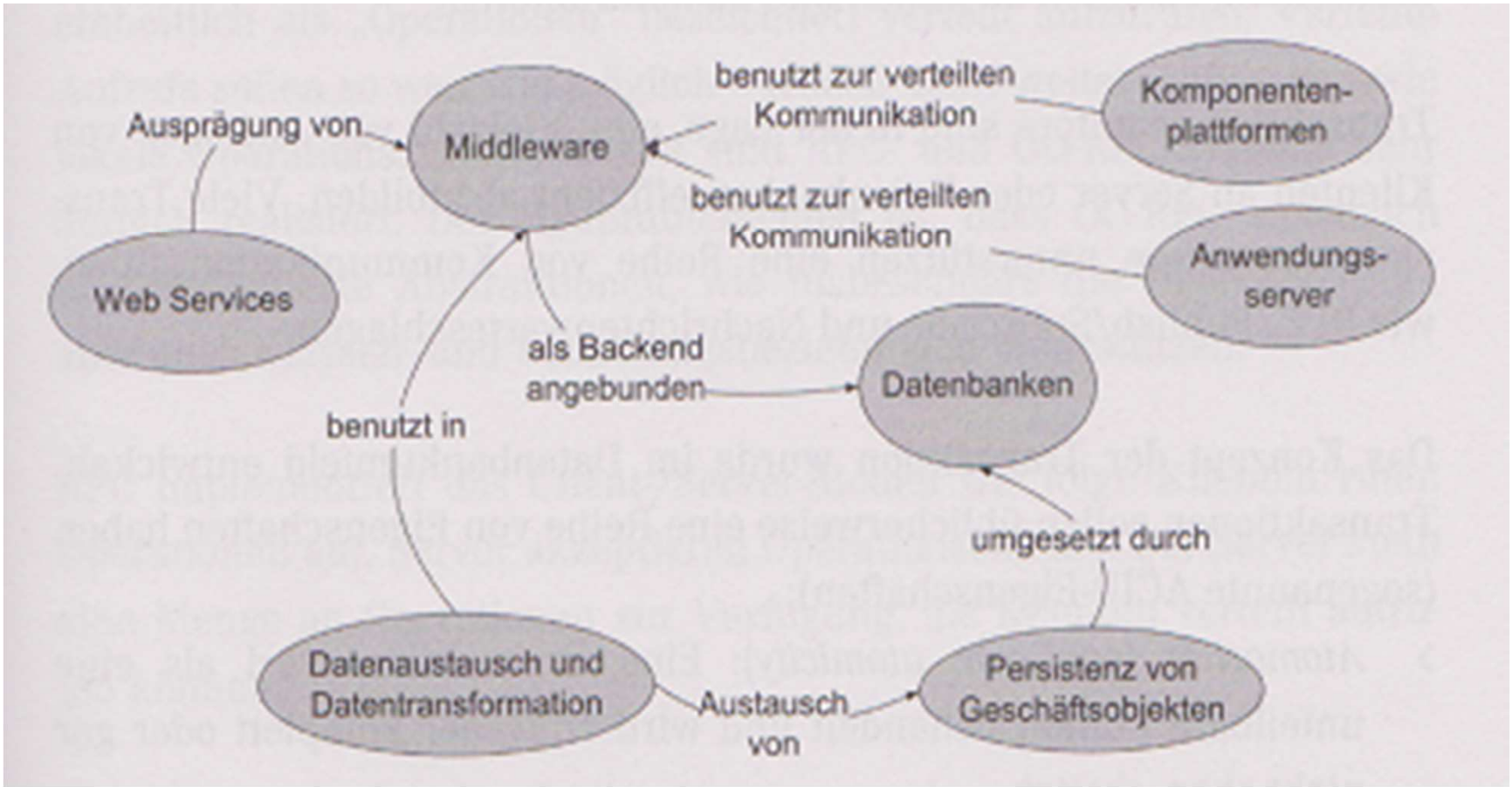
Architekturrelevante Technologien:

- Grundsätzliche Technologien:
  - Compiler
  - Virtuelle Maschinen
  - …
- Spezifische Technologien:
  - Content-Management-Systeme
  - Enterprise-Resource-Planning-Systeme (ERP)
  - …

- Im folgenden kurz betrachtete Technologien:
  - Middlewaresysteme
  - Datenbanken und Persistenz von Geschäftsobjekten
  - Komponentenplattformen
  - Web Services

Architekturrelevante Technologien:

- **Überblick - Technologien:**



O.Vogel et.al., Software-Architektur, Grundlagen-Konzepte-Praxis, Spektrum, 2009

Arc.-Mittel/Technolog.-WOMIT-91

Architekturrelevante Technologien:

- **Middleware-Systeme:**

  Middleware-Systeme sind eine Basisarchitektur …

  - **Beispiele:**

    - Transaktionsmonitore: sind in der Lage, eine Vielzahl von Anfragen von Klienten an Server oder Datenbanken effizient abzubilden; unterstützen Kommunikationsstile (z.B. RPC); Transaktionen sollen sein: atomar (komplett oder gar nicht) – konsistent (Zustand des Systems nach der Transaktion) isoliert (keine Beeinflussung durch andere Transaktionen dauerhaft (Änderungen nach der Transaktion sind permanent); kommerzielle Transaktionsmonitore: z.B. Tuxedo von BEA

    - RPC- und OO-RPC-Middleware: basierend auf RPC + objektorientiert; es gibt synchrones und asynchrones RPC; zusätzlich eigene Fehlerfälle/Stati betreff „remote" (z.B. bei Netzwerkausfall);
    OO-RPC-Systeme: z.B. CORBA, .NET Remoting, Java RMI

Arc.-Mittel/Technolog.-WOMIT-92

Architekturrelevante Technologien:

- **Middleware-Systeme:**
  - **Beispiele:**

- Message-oriented Middleware (MOM Systeme):
realisieren (asynchrone) Kommunikation über Nachrichten-Queues (Warteschlangen): dabei haben Clients und Server Queues; Resultate werden geliefert durch Callbacks oder Polling der Clients; der asynchrone Charakter erfordert die Benutzung von eindeutigen Identifikatoren (Identifier) für die richtige Klammerung von Anfragen und Antworten (Resultaten);
MOM Systeme unterstützen meist mehrere Nachrichtenkanäle mit jeweils eigenen Queues (die Anwendungen interagieren dabei meist über sogenannte Endpoints); darüberhinaus noch weitere wichtige Eigenschaften: Nachrichten zuverlässig übermitteln–Reihenfolge der Nachrichten garantieren-automatische Clearing von Queues – richtige Behandlung von Mehrfachnachrichten
MOM-Systeme: z.B. IBM WebSphere MQ, JMS, MSMQ

- Peer-To-Peer-Syst., spontane Netzwerke, Grid Computing, Mobiler Code

Arc.-Mittel/Technolog.-WOMIT-93

Architekturrelevante Technologien:

- **Datenbanken und Persistierung von Geschäftsobjekten:**
  - Persistenzanforderung architektonisch relevant
  - Persistierung: meist erforderlich und notwendig – bei kleineren und temp. Daten kann „Halten" in RAM auch ausreichend sein
  - Zugriff auf Daten: Relevanz von Performanz, Verfügbarkeit, Skalierbarkeit, Einfachheit, Sicherheit; transaktionsunterstützt vorteilhaft
  - Objektorientierung und Datenbanken (meist relationale): möglicher Strukturbruch – wie kann man das Anwendungsmodell auf das Datenbankmodell abbilden (wie gut ist das unterstützt …) – OODBMS-Systeme leisten das …
  - Relationale Datenbanken vielfach im Einsatz: Object-Relational-Mapping (ORM); eine einfache aber nicht optimale Lösung ist: SQL-Code in die Anwendungslogik einbetten …
  - Meist eigene Datenbankzugriffsschichten:
    - z.B. JDBC (Entwickler muss SQL „kennen")
    - ORM: Entwickler konfiguriert Persistenz nur mehr

Arc.-Mittel/Technolog.-WOMIT-94

Architekturrelevante Technologien:

- **Datenaustausch mit XML:**
  - Austausch (unterschiedlich) strukturierter Daten(sätze)
  - Meistgewählter Ansatz: XML – die Verwendung von XML erlaubt es XML-basierte Austauschformate und Austauschstandards zu definieren; auch „Altdaten" können leicht in XML-Formate transformiert werden; grosser Vorteil von XML: sehr verbreitet (eingesetzt)
  - Document Type Definition (DTD): Spezifikation des Aufbaus von XML-Dokumenten möglich
  - Umfassendere Lösung mit XML-Schema-Standard(s): jedes Schema ist ein gültiges XML-Dokument - selbstdefinierte Datentypen möglich
  - Weitere XML-Standards: XML Namespaces, XHTML, Xlink, Xpath, XSLT (Transformer), XQuery (SQL-artig)
  - Neben XML gibt es viele andere Datenaustauschsprachen – allg. Ansätze wie EDI (Electronic Data Interchange) komplex … Praxis: zumindest mal „lediglich" branchespezifische Lösungen

Arc.-Mittel/Technolog.-WOMIT-95

Architekturrelevante Technologien:

- **Web-Anwendungsserver:**
  - **Web-Seiten:** heutzutage meist dynamisch (Datenaustausch mit einem Backend, HTML-Aufbereitung, …): Web-Anwendungsserver.
    Architekturen für serverseitige Programmmodule:
    - *CGI-Schnittstelle:* dynamisches Starten von Scripts, eher nur für kleinere Sachen einsetzbar, schwer zu verstehen und warten
    - *Template Sprachen:* PHP, ASP, JSP; schon komfortabler, aber: Fehlen von einfachen Mitteln für Kommunikation, Programmlogik in der jeweiligen Template-Sprache kann oft nicht für andere Zwecke (wieder)verwendet werden
    - *Anwendungsserver:* Apache Tomcat, Jboss, BEA Web Logic, IBM WebSphere; sind oft Teil von größeren Standardarchitekturen wie .NET oder JEE
    - *Web-Content-Management und –Community-Systeme:* Web-Anwendungsserver+Erweiterungsmodule (Foren, Wikis, …)
    - *Agile Web Frameworks:*
    basierend auf Prinzipien: „Don‘t Repeat Yourself“, „Convention over Configuration“ - Programmierkonventionen werden über Anwendungs-konventionen gestellt -> raschere Umsetzung von Anforderungen; Frameworks: Ruby on Rails, seaside, Mason, Grails, …

Arc.-Mittel/Technolog.-WOMIT-96

Architekturrelevante Technologien:

Komponentenplattformen:

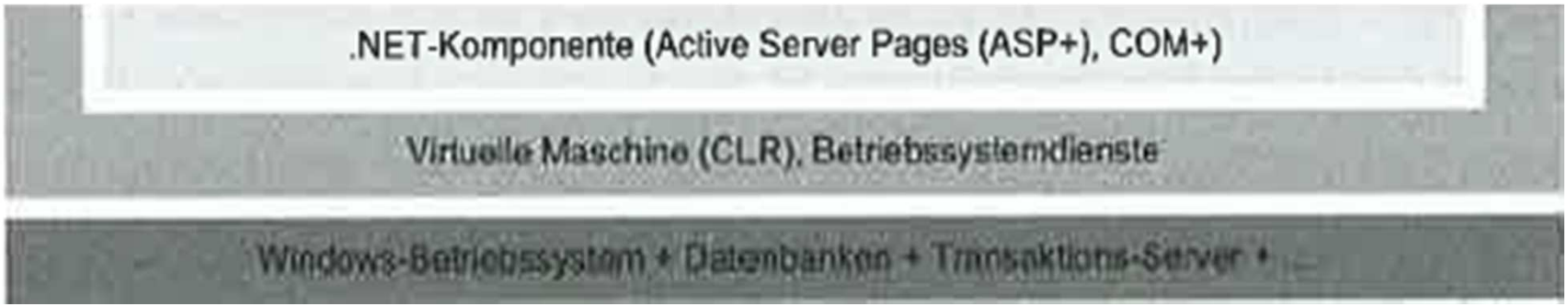
- Beispiel: JEE ist auf Java-Technologie basierende Komponentenplattform in der Programmiersprache **Java**:
  ist plattformunabhängig – nur bedingt herstellerunabhängig (SUN);
  es gibt viele JEE-Implementierungen unterschiedlicher Hersteller (inkl. Open Source) –> Entscheidung abhängig von Anforderungen und Budget …
- JEE sollte „in den architektonischen Rahmen gesetzt" werden –
  JEE schränkt die architektonische Freiheit bis zu einem gewissen Grad ein – falscher Einsatz kann zu Problemen führen (z.B. schlechte Performanz) -> bewährte JEE-Entwurfstechniken verwenden …
- JEE - Komponentenplattform – Überblick:

Arc.-Mittel/Technolog.-WOMIT-97

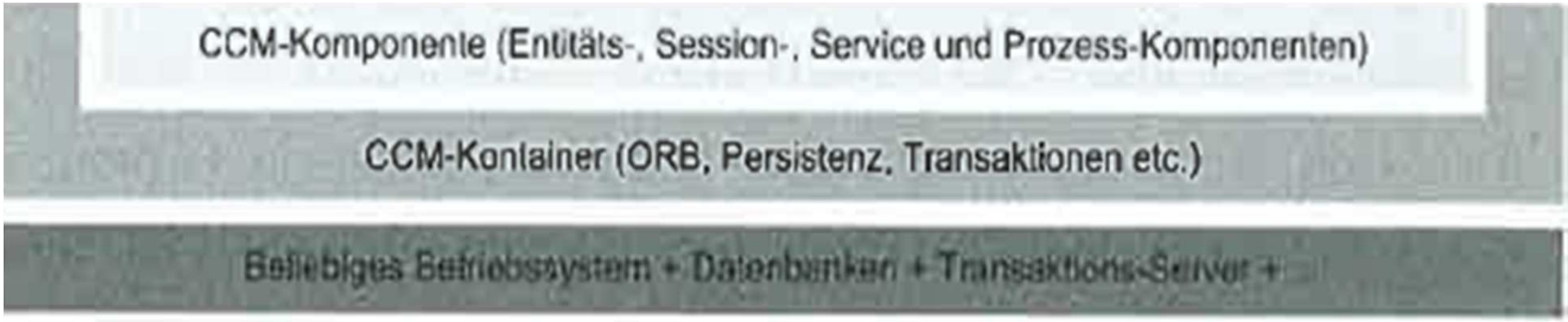
SAD Mittel

Architekturrelevante Technologien:

- **Komponentenplattformen:**
  - .NET - Komponentenplattform – Überblick (für/in **C#**):



  - CORBA-Component-Model(CCM) – Überblick(**+Prozesskomponenten**):



Arc.-Mittel/Technolog.-WOMIT-98

SAD Mittel

Architekturrelevante Technologien:

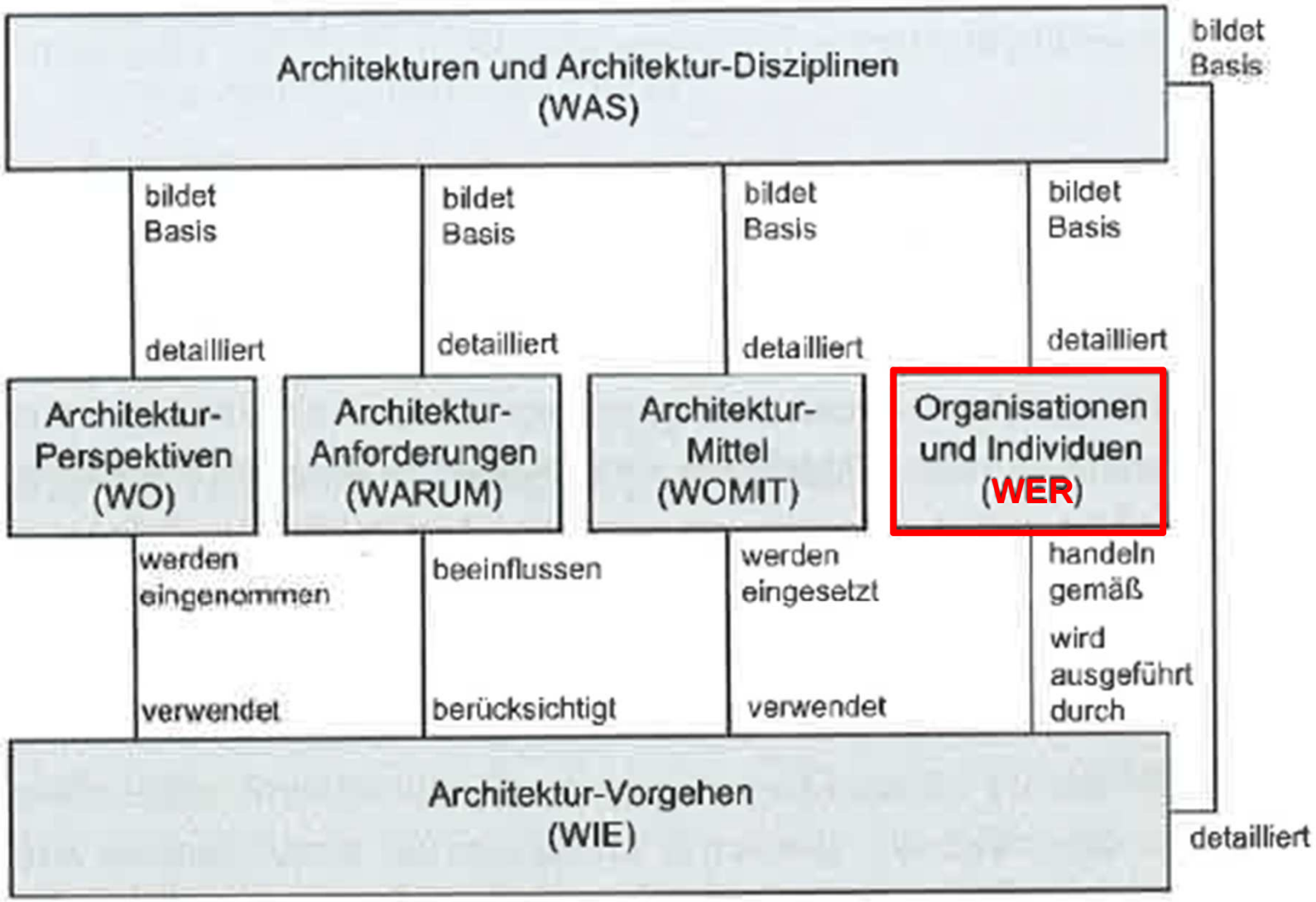
- **Web Services:**
  - Web Services setzen im Prinzip die SOA-Basisarchitektur um
  - Web Services heute basieren auf einer Schichtenarchitektur aus mehreren, standardisierten Protokollen: SOAP/REST(„schlanker"; für „kleine" Daten); WSDL (Schnittstellenbeschreibungssprache)
  - Web Services benötigen kein spezielles Kommunikationsprotokoll – zumeist http(s) – weitere Protokolle meist als Plug Ins hinzufügbar
  - Komposition von Web Services: z.B. Business Process Execution Language für Web Services (BPEL4WS): erlaubt es (ganze) Geschäftsprozesse zu beschreiben
  - …

Architektur-Organisation - WER

Architektur - Organisation
WER

Architektur-Organisation - WER-1

SAD Organisation



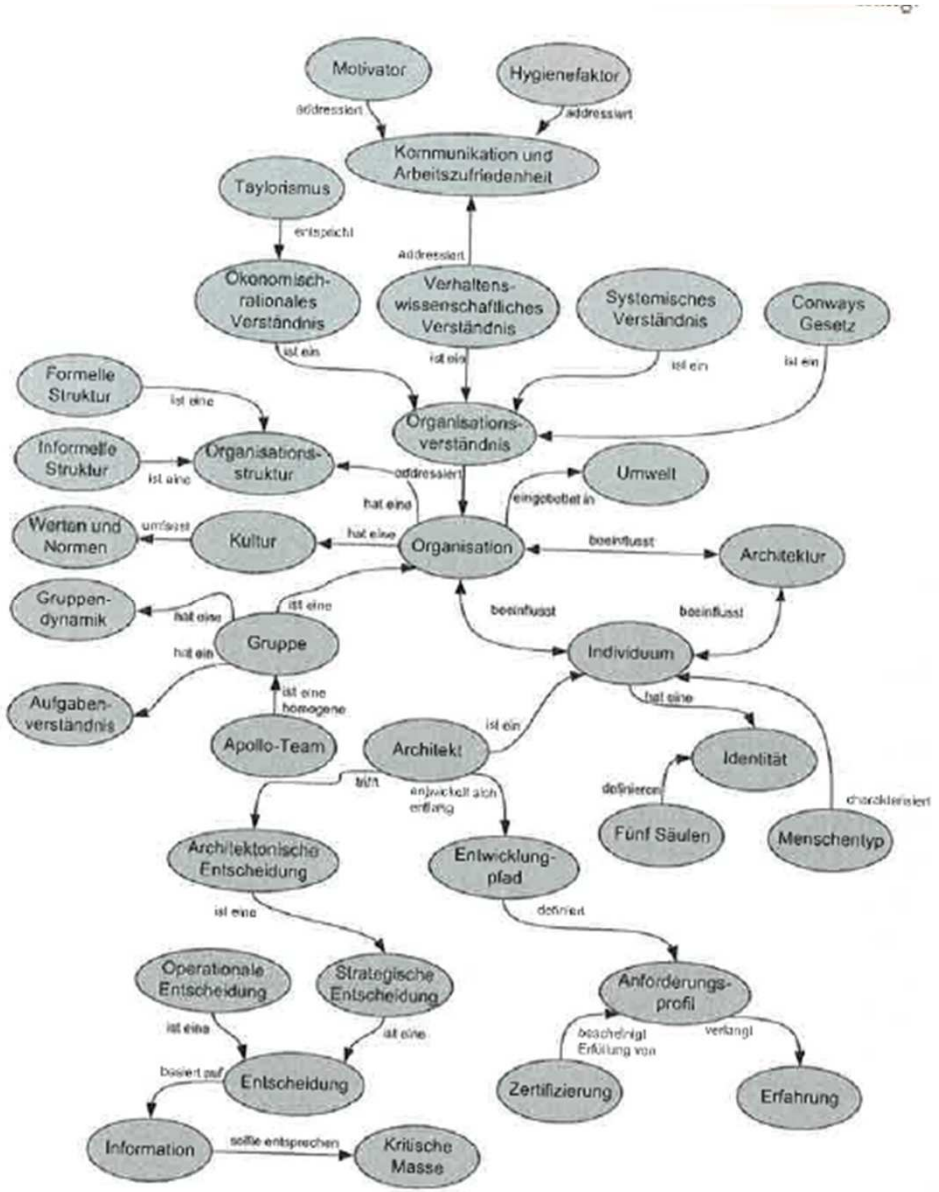
Architektur-Organisation - WER-2

Was meint Mittel und WER:

- Auf welchen Abstraktionsstufen bewegt sich ein Architekt im Rahmen seiner Tätigkeit
- Wie manifestiert sich die Architektur auf den Abstraktionsstufen
- Welche architektonischen Sichten (auf die Architektur) können verwendet bzw. angewandt werden, um die Architektur entsprechend unterschiedlicher Aspekt zu betrachten, zu beschreiben und zu bearbeiten

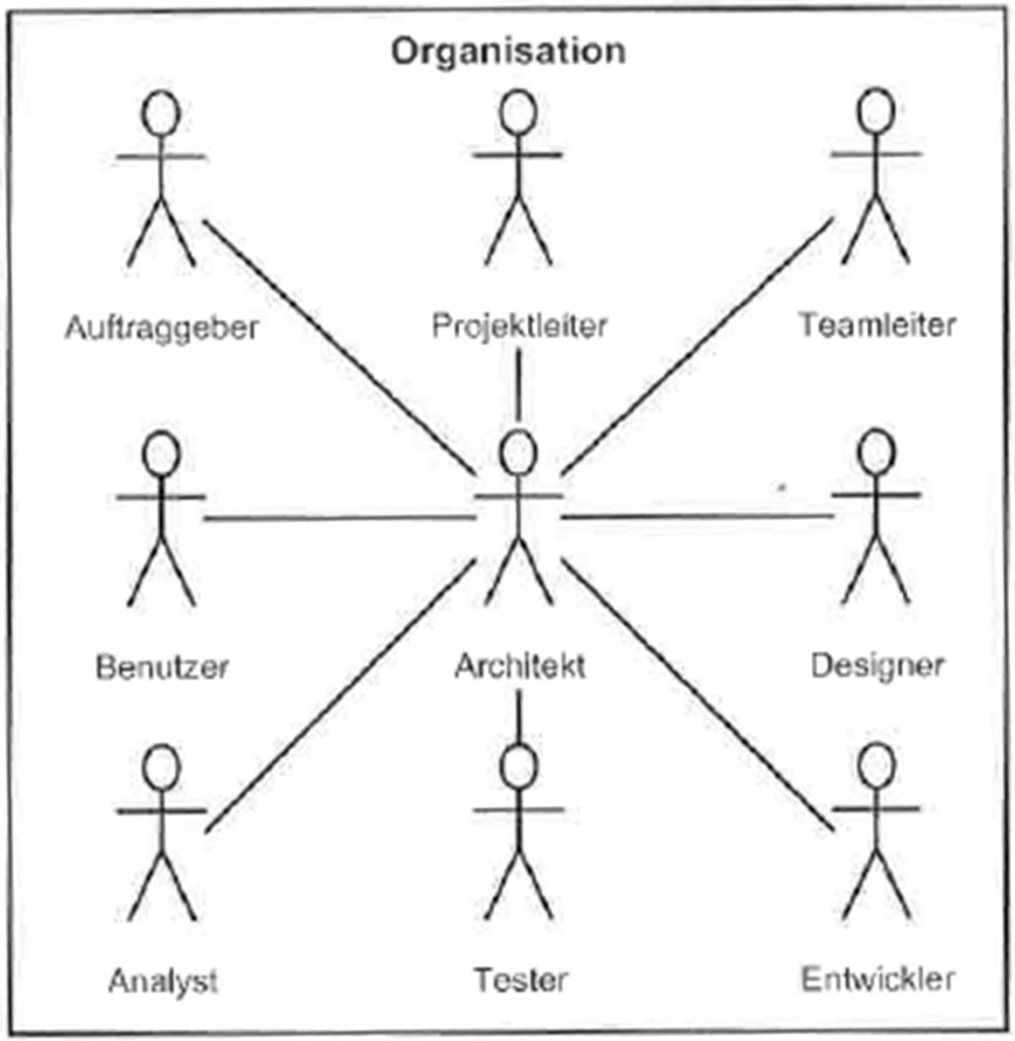
Architektur-Organisation - WER-3

SAD Organisation



Architektur-Organisation - WER-4

SAD Organisation



Arch.Org.-Architektenrollen-WER-5

SAD Organisation

- **Uneinheitlich, dennoch richtungsweisende Abgrenzungen**
  - Je nach Aufgabengebiet und der zu erstellenden Artefakte können unterschiedliche Architektenrollen unterschieden werden
  - **Software Architekt**
    - Bearbeitet Struktur und Design von SW Systemen
    - Funktionale Anforderungen und Qualitätsattribute
    - Rolle idR einem Projekt zugeordnet
  - **Solution Architekt**
    - Fokus auf Business Requirements und auf Wiederverwendung von IT Capabilities und Services
    - Einzelprojekt unter Einhaltung übergreifender Architekturprinzipien
    - Bedeutsam in grösseren Projekten mit unterschiedlichen Systemen
  - **Enterprise Architekt**
    - Fokus auf Unterstützung der Geschäftsstrategie durch die IT Strategie
    - Strategische Ausrichtung der IT Capabilities unter wirtschaftlichen Aspekten
    - Etablierung von Standards und unternehmensweite Governance

Arch.Org.-Architektenrollen-WER-6

SAD Organisation

- **Die unterschiedlichen Architektenrollen können sich unterscheiden durch:**
  - **Qualifikation**
    - z.B. SW-Architekten eher technische Qualifikation (technische Frameworks etc.), Enterprise Architekten hingegen IT Management
  - **Domänenwissen**
    - SW-Architekt fokussiert auf einzelne Lösungsdomäne, Solution Architekt auf Integration, EA auf organisationsweite IT Gestaltung
  - **Reichweite der Verantwortung**
    - SW-Architekt trägt technische Verantwortung einzelner Lösungen, EA verantwortlich für gesamte IT Bebauung einer Organisation (in Zusammenarbeit mit IT Management)
  - **Erzeugte Artefakte**
    - z.B. konkrete SW-Entwürfe (SW-A) versus aggregierte Bebauungspläne (EA)
  - **Erfahrung**
    - z.B. steigen Senior Developer häufig in die Rolle eines SW-Architekten ein, Solution Architekten und EA verfügen hingegen über langjährige Erfahrung mit komplexen Systemen und Organisationen

Arch.Org.-Architektenrollen-WER-7

SAD Organisation

Aufgaben:

- **Anknüpfen an fachliche Anforderungen**
  - Machbarkeit, Erfüllung
- **Entwurf von SW-Architektur**
  - Berücksichtigung des gesamten Lebenszyklus einer Lösung: Entwicklung, Betrieb und Wartung
  - Komponenten und deren Verantwortlichkeiten
  - Schnittstellen und deren Verträge
  - Strukturen auf Basis der Komponenten auf unterschiedlichen Ebenen (statisch und dynamisch)
- **Fortlaufendes Treffen von Entwurfsentscheidungen**
  - Framworks, Entwurfsmuster, Schnittstellen, Make or buy etc.
  - Begründung, Kommunikation und Dokumentation der Entscheidungen!
  - Eingrenzung von Komplexität durch Prototypen, iteratives Vorgehen etc.

Arch.Org.-Architektenrollen-WER-8

SAD Organisation

Aufgaben:

- **Beratung und Kommunikation**
  - Unterstützung u.a. bei Qualifikation von Vorhaben, Machbarkeit, Risikomanagement, Konzeption von Betriebsumgebungen, Testbarkeit
  - Adressieren unterschiedlicher Stakeholder, Information über Strukturen und Schnittstellen, Kommunikation von Entscheidungen, Vermittlung von Know-How
- **Bewertung**
  - Güte von Architekturen, Messen der Zielerreichung
  - Erfüllung von NFRs
  - Maßnahmen zur Optimierung
- **Dokumentation**
  - Zweckorientiert und angemessen
  - Auf Aktualität achten!

Arch.Org.-Architektenrollen-WER-9

SW-Architekt – Rolle und Aufgaben im Überblick:

- ***Architekt:*** eigene Ingenieurdisziplin/eigenes Berufsbild.
- ***Architekt:*** ist zentrale Drehscheibe zwischen Kunden, Projektmanager, Designer, Entwickler(n).
- ***Architekt:*** argumentiert und begründet Entscheidungen gegenüber Kunden und Management.
- ***Architekt:*** enge Zusammenarbeit mit Projektmanagement (oft Einbindung in Planungen - Iterationen, Arbeitspakete, Meilensteine, …).
- ***Architekt:*** stellt während der Implementierung sicher, daß die definierte Architektur umgesetzt wird und Abweichungen erkannt werden.
- ***Architekt:*** zuständig für Abbildung und Aufbereitung der funktionalen und nichtfunktionalen Anforderungen.
- ***Architekt:*** dokumentiert Architektur und zugehörige Architekturentscheidungen adequat.