Softwarearchitektur und Design

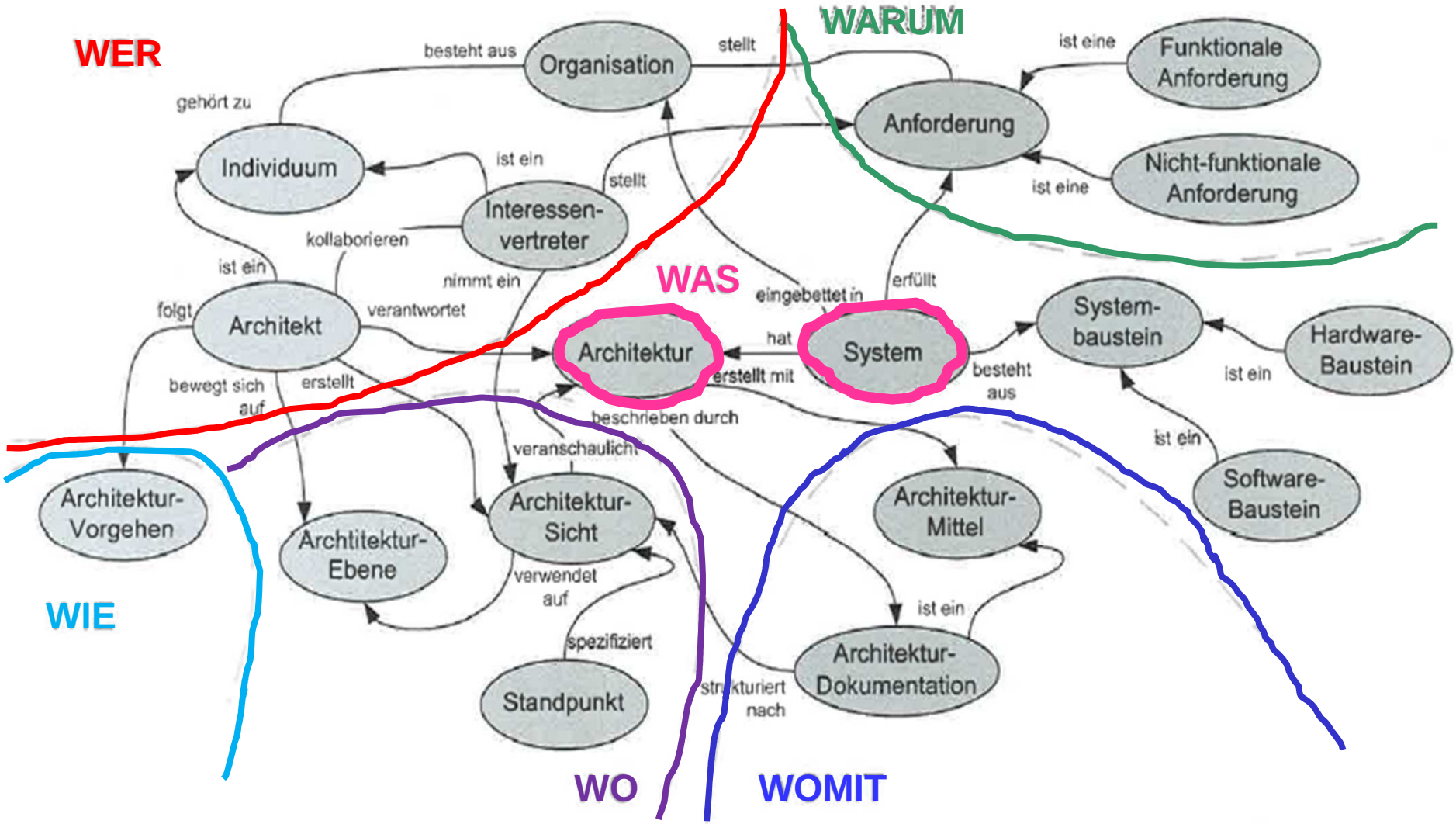
Block 3a – Mittel

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 3a

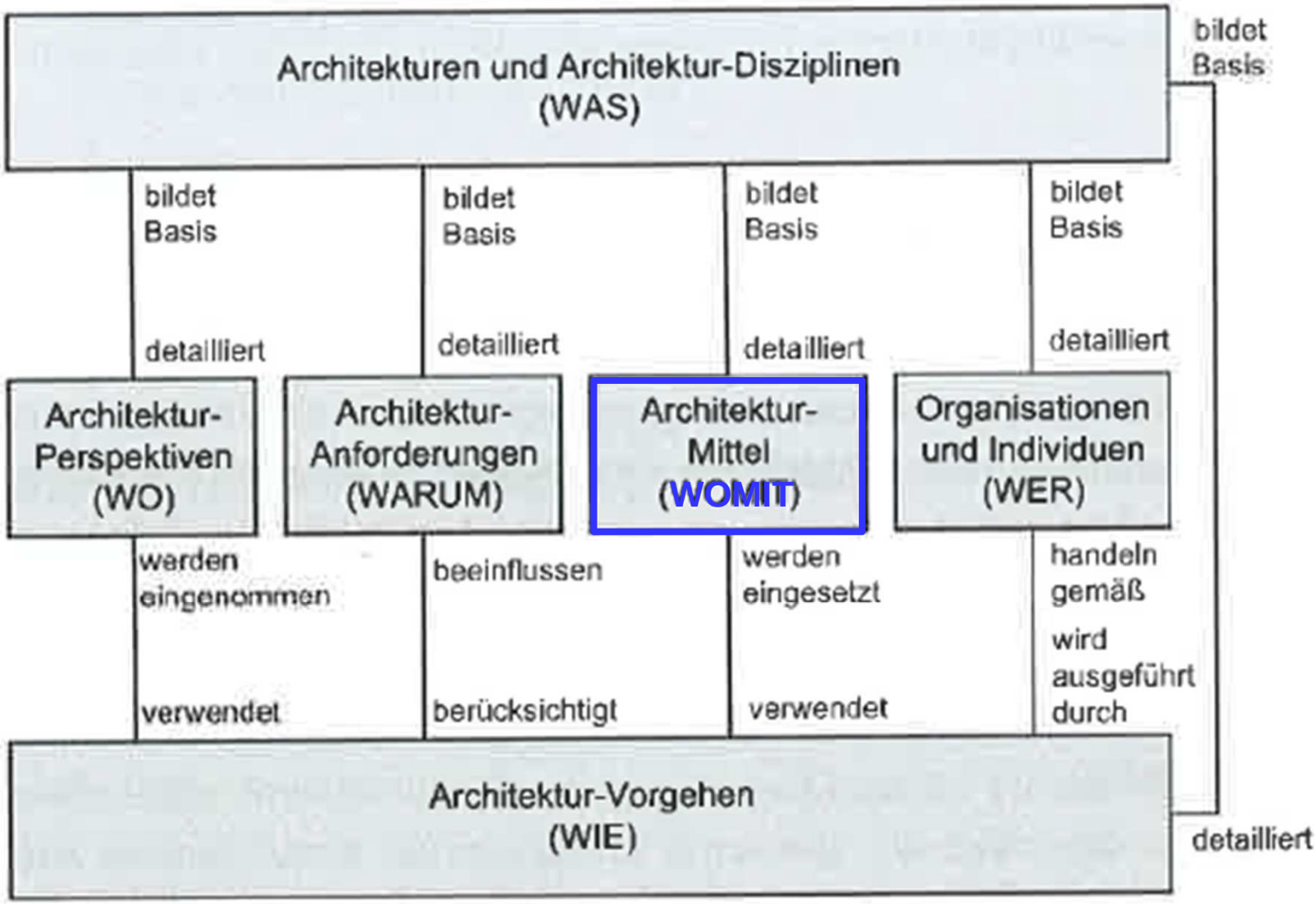
- **4 – WOMIT**
  Architektur-Mittel
  Prinzipien, Taktiken, Stile, Muster(Patterns);
  Modellierung(smittel)

Architektur-Mittel - WOMIT

Architektur - Mittel
WOMIT

Architektur-Mittel - WOMIT-1

SAD Mittel



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

Architektur-Mittel - WOMIT-2

Was meint Mittel und WOMIT:

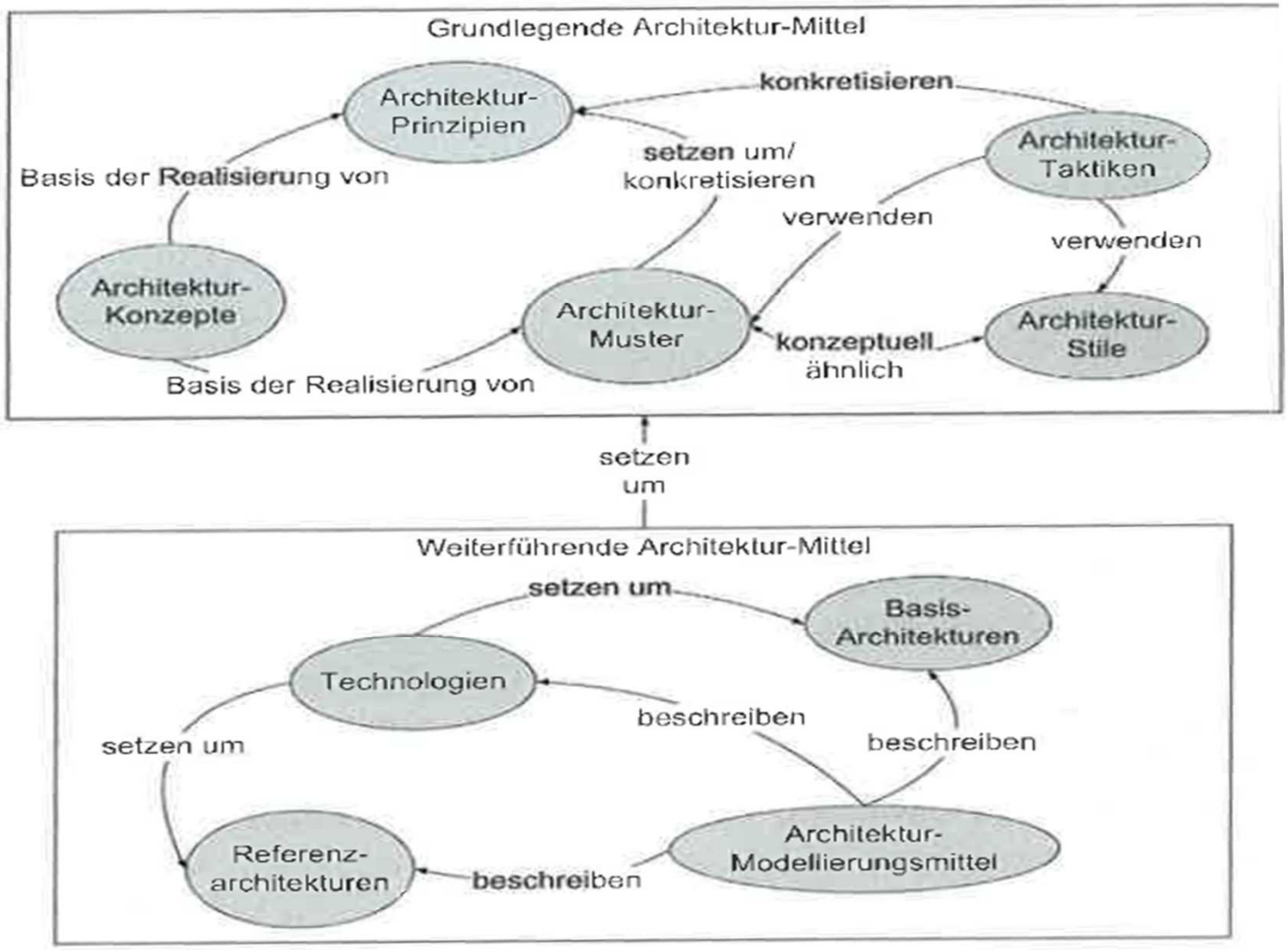
- Werkzeugkasten des SW-Architekten
- Werkzeuge: Konzepte, Techniken, …

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-3

SAD Mittel

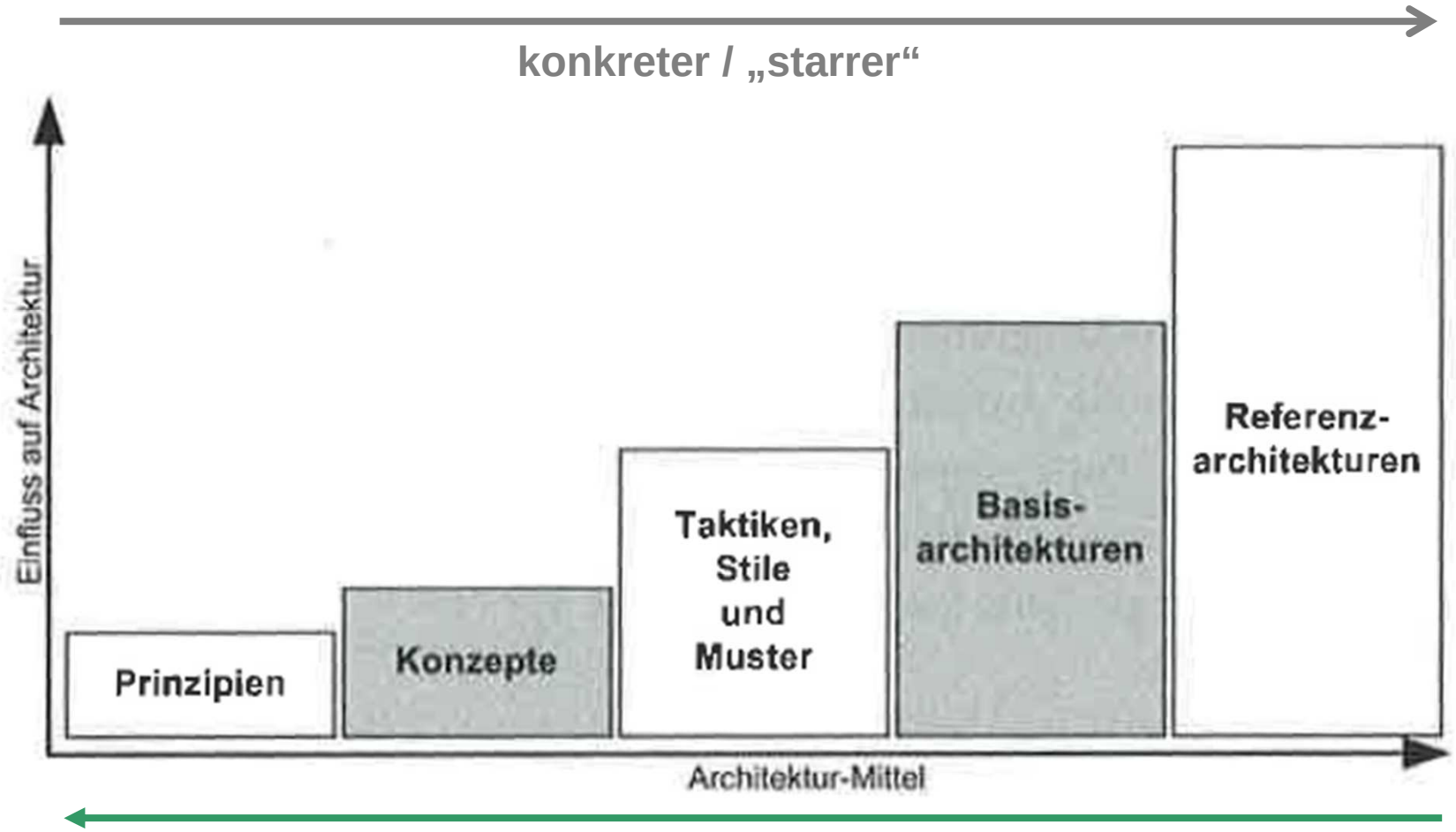


O.Vogel et.al., Software-Architektur, Grundlagen-Konzepte-Praxis, Spektrum, 2009

Architektur-Mittel - WOMIT-4

SAD Mittel

Architekturmittel und Einfluss auf die Architektur:



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

Arc.-Mittel/Prinzipien - WOMIT-5

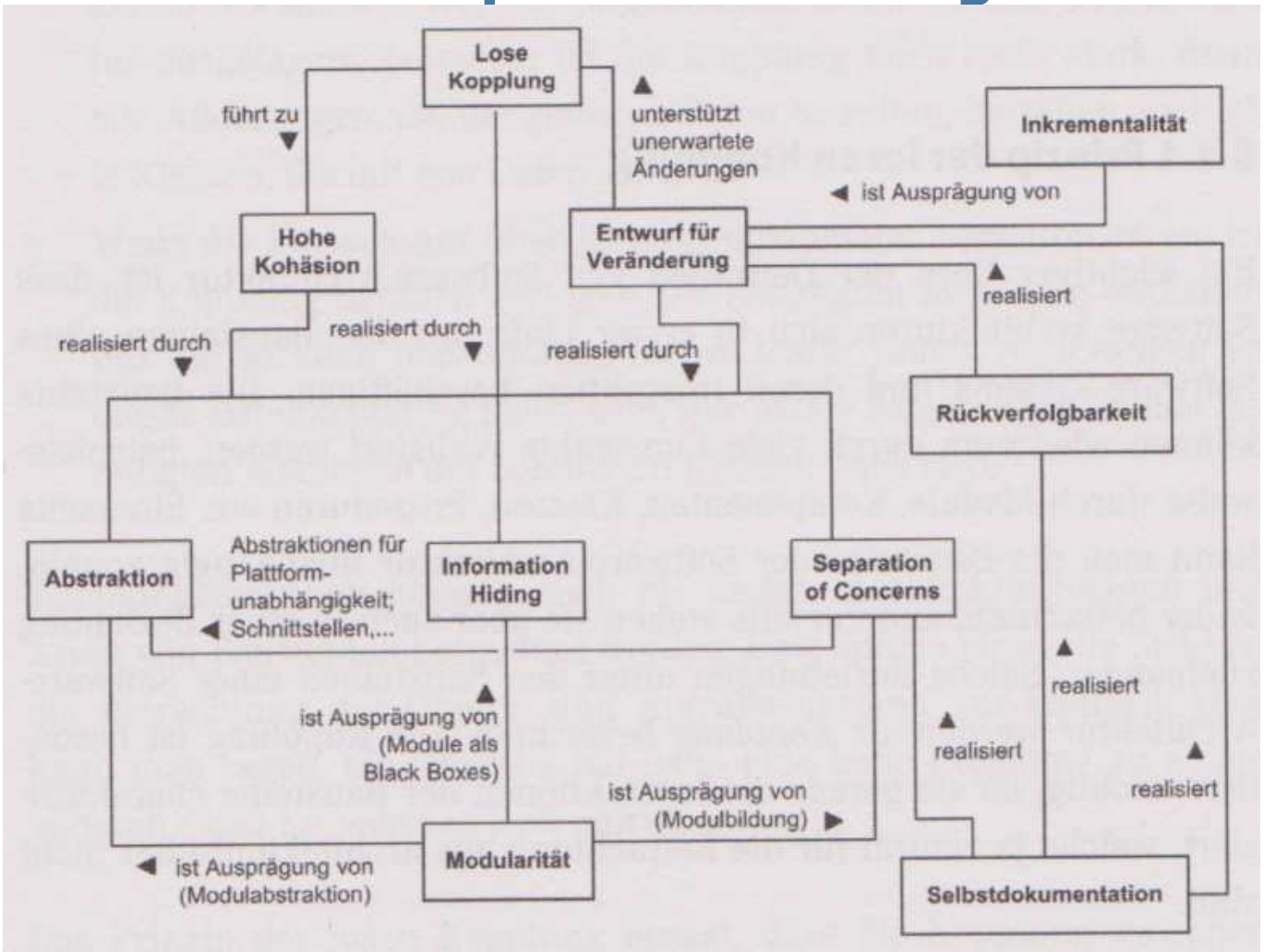
Architektur-Prinzipien:

- Sind bewährte (allgemeine) Grundsätze zur Gestaltung der Architektur (keine Konkretisierungen)
- Architekturen unterscheiden sich zumeist betreffend die Ausprägung der Qualitätsmerkmale (Erfüllungsgrad der nichtfunktionalen Anforderungen) - angepasst
- Es gibt trotzdem allgemeine Prinzipien, die bei Erstellung einer SW-Architektur beachtet werden sollten
- 2 Probleme/Kräfte und Prinzipien (oft gegenläufig):
  - Reduktion der Komplexität
  - Erhöhung der Flexibilität
- Man unterscheidet grundsätzlich zwischen:
  - Grundlegenden Architekturprinzipien
  - Abgeleiteten Architekturprinzipien

Arc.-Mittel/Prinzipien - WOMIT-6

SAD Mittel

Architektur-Prinzipien - Grundlegende:

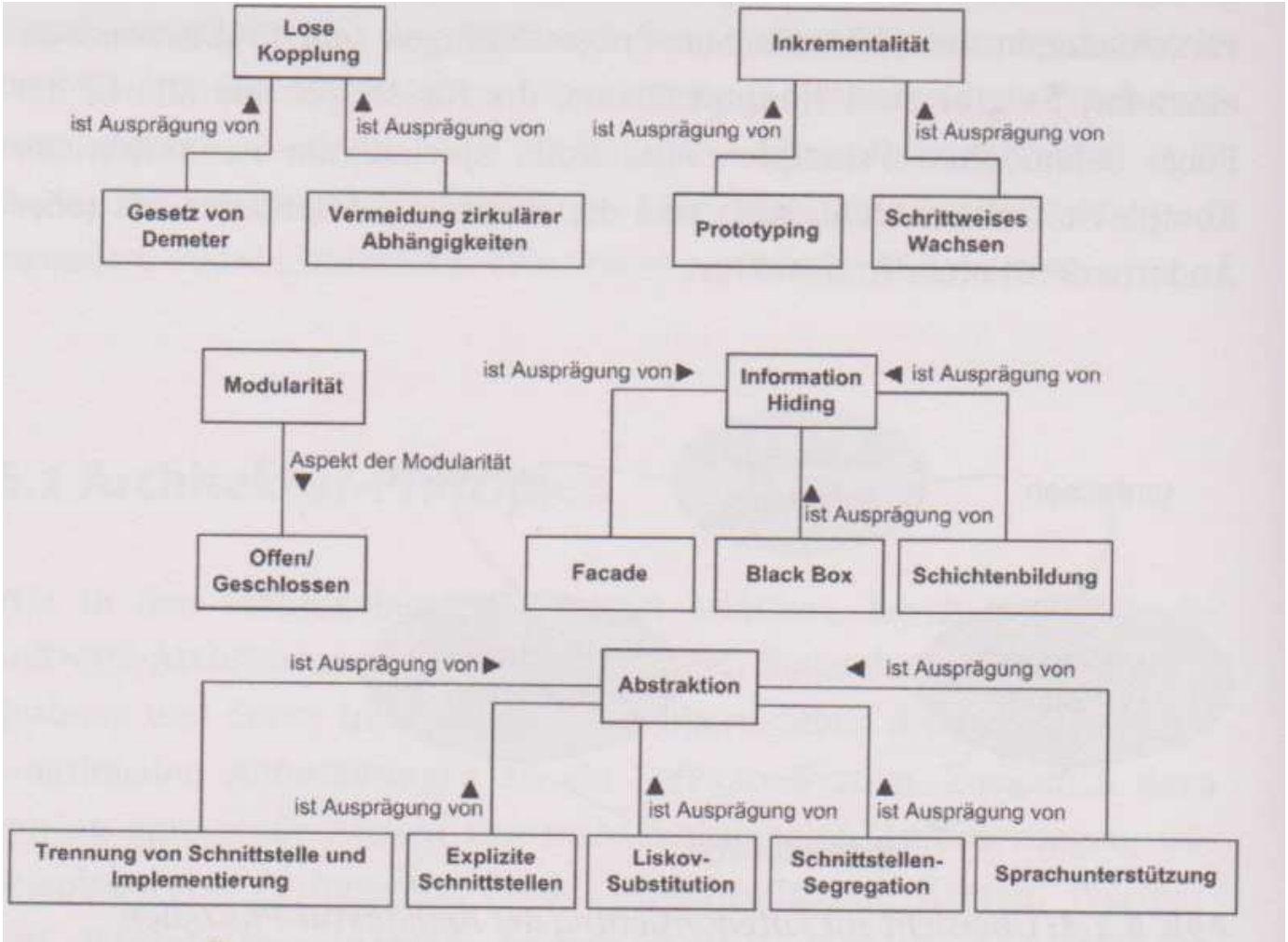


O.Vogel et.al., Software-Architektur, Grundlagen-Konzepte-Praxis, Spektrum, 2009

Arc.-Mittel/Prinzipien - WOMIT-7

SAD Mittel

Architektur-Prinzipien - Abgeleitete:



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

Arc.-Mittel/Prinzipien - WOMIT-8

Architektur-Prinzipien – „Lose Kopplung“:

Kopplung charakterisiert die Interaktion von Bausteinen …

Geringe Kopplung = Lose Kopplung

- Bausteine von SW-Systemen und ihre Interaktionen können durch unterschiedliche Konstrukte realisiert werden:
  Module, Komponenten, Klassen, Prozeduren, ...
- Lose Kopplung bringt: Verringerung der Komplexität (betreff Verstehen, Änderungen und ihre Auswirkungen, Abhängigkeiten)
- Eventuelle Nachteile: z.B. Komplexität, hoher Ressourcenbedarf bei falschem Einsatz

Beispiel - Klassen:

- Wie stehen Klassen untereinander in Beziehung + in welchem Ausmass: Anzahl ist z.B. eine Metrik für Kopplung der Klassen (**Kopplungsmetrik**)
  Anmerkung: es gibt andere Kopplungsmetriken: z.B. wie oft ruft Objekt A Objekt B zur Laufzeit auf
- *Stärkere Kopplung* bei Klassen z.B. betreff Daten(-austausch): gegenseitiges Zugreifen auf (private) Daten, globale Datenstrukturen
  Losere Kopplung: Klassen kommunizieren nur über Methoden(parameter)

Arc.-Mittel/Prinzipien - WOMIT-9

Architektur-Prinzipien – „Lose Kopplung“:

- Lose Kopplung kann erreicht werden durch:
  Abstraktion, Separation of Concerns, Information Hiding, Schnittstellenabstraktion: Belange von „Schnittstelle“ und „Implementierung“ sauber trennen + einfache Schnittstelle
- Verwandtes Prinzip zu loser Kopplung ist das Gesetz von Demeter: „Systembaustein soll nur mit eng verwandten/benachbarten Bausteinen zusammenarbeiten“ (man vergleiche: Neuronen, neuronale Netze ..,)
- Weitere Teilprinzipen der losen Kopplung:
  - **Vermeidung zirkularer Abhängigkeiten** von Bausteinen:
    bekannt: „Deadlocks; Probleme enstehen im Zusammenspiel der Bausteine und im gesamten Zyklus – Arbeitsteiligkeit verloren)
  - **Hollywood Prinzip oder Inversion of Control:**
    „Don‘t call us, we call You !“
  - **Dependency Inversion, Injection (Anwendg. des Hollywood Prinzips):**
    Inversion: Baustein definiert Schnittstelle – anderer Baustein realisiert diese;
    Injection: Verantwortung für Erzeugen und Verknüpfen von Bausteinen wird an ein extern konfigurierbares Framework übergeben

Arc.-Mittel/Prinzipien - WOMIT-10

Architektur-Prinzipien – „Hohe Kohäsion“:

Kohäsion bezeichnet die Abhängigkeiten (von Teilen) innerhalb eines Systembausteins

(z.B: Klassen haben Attribute und Methoden; Objekt ruft sich zur Laufzeit selbst oft auf)

- Hohe Kohäsion: vorteilhaft bezüglich Änderungen (nur der eine Systembaustein betroffen) und Wartbarkeit; Komplexität: meist auch verringert (alles in einen Systembaustein verpackt); Austauschbarkeit von Bausteinen leichter
- Hohe Kohäsion: Systembausteine als Black Boxes …

Zusammenhang zu anderen Prinzipien und Teilprinzipien:

- Kopplung und Kohäsion stehen in Wechselbeziehung: je höher die Kohäsion der Bausteine -> desto geringer in der Regel die Kopplung (und umgekehrt)
- Hohe Kohäsion kann erreicht werden durch Umsetzung folgender Prinzipien bzw. Teilprinzipien:
  - Abstraktion
  - Separation of Concerns
  - Information Hiding

Arc.-Mittel/Prinzipien - WOMIT-11

Architektur-Prinzipien – „Entwurf für Veränderung“:

Motivation und Hintergrund: in der Realität ändert sich SW oft ständig (aus unterschiedl. Gründen)

- Prinzipiell kann man zwischen „vorhersehbaren/erwartbaren“ und „unvorhersehbaren/nichterwartbare“ Änderungen unterscheiden
- **„Entwurf der Veränderung“** liegt die Idee zugrunde: **vorhersehbare Änderungen architektonisch berücksichtigen und vorausplanen** (vorhersehen: meist Erfahrungssache) **...**

- Hinweise und Ansatzpunkte:
  - Unklare Anforderungen; weitergehende Anforderungen
  - Erweiterungen (die bereits von Anfang an offensichtlich sind)
  - Momentaner Kostenzwang (bzw. -engpass), aber Anforderung wichtig
  - Erfahrungen aus Entwurf zu „ähnlichen“ Systemen
- Vorgangsweise bezüglich nichterwartbarer Änderungen:
  - Man kann nicht alles in Betracht ziehen und hochflexibel entwerfen -> kann Nachteile bedeuten, sowie dann Mehraufwand und Mehrkosten

Arc.-Mittel/Prinzipien - WOMIT-12

SAD Mittel

Architektur-Prinzipien – „Entwurf für Veränderung“:

Zusammenhang zu anderen Prinzipien und Teilprinzipien:

- Kopplung und Entwurf für Veränderung: je loser die Kopplung der Bausteine -> desto besser meist „gewappnet“ für Änderungen
- Hohe Änderungsfähigkeit wird unterstützt durch Ansätze wie: SOA, Aspektorientierte Arch.
- Identifikation von „Hot Spots“: wo sind schon oft Änderungen aufgetreten oder wo sind viele Aspekte zusammengekommen -> dort Änderungsfähigkeit vorsehen (vorausplanen)

  Beispiele für Hotspots:
  - Aufrufverteilung bei Middleware wie CORBA oder Web-Service-Implementierungen mit Broker-Architektur
  - Konfiguration von Komponenten: ändert sich generell oft (-> XML)
- Hohe Änderbarkeit kann weiters erreicht werden durch Umsetzung folgender Prinzipien bzw. Teilprinzipien:
  - Abstraktion
  - Modularität
  - Separation of Concerns
  - Information Hiding

Arc.-Mittel/Prinzipien - WOMIT-13

SAD Mittel

Architektur-Prinzipien – „Separation of Concern“:

Trennung von Aufgabenbereich und Belangen = Teile und Herrsche (Divide and Conquer)

- Verschiedene Aspekte werden voneinander getrennt und als separate Teilprobleme behandelt
- Prinzip unterstützt und erlaubt:
  - Reduktion von Komplexität
  - Arbeitsteilige Bearbeitung
- Ist ein generelles Prinzip des SW-Engineerings über die SW-Architektur hinaus
- Anwendung des Prinzips betreffend SW-Architektur:
  - Zerlegung von Anforderungen
  - Zerlegung von Architekturbeschreibungen entsprechend Sichten
  - …
- Wichtigster Einsatz des Prinzips in der SW-Architektur (+Entw.): **Modularisierung:** Kapselung von Systembausteinen entsprechend ihrer Aufgabe, Angelegenheit, Aspekt, … bedeutet: lose Kopplung

Arc.-Mittel/Prinzipien - WOMIT-14

SAD Mittel

Architektur-Prinzipien – „Separation of Concern“:

- Sinnvolle Zer- bzw. Aufteilung des Gesamtsystems in verständliche und handhabbare, möglichst unabhängige Einzelteile unterstützt die parallele Realisierung der SW
- Kriterien für Dekomposition von SW-Systemen:
  - Funktionalitätsanforderungen (fachlich orientierte Dekomposition)
  - Wiederverwendbarkeit
  - …
- **Grundsätzliche Vorgangsweisen:**
  - Trennung von fachlichen und technischen Teilen ist anzustreben
- **Mehrdimensionalität** von Sep. of Concerns
  - Einbeziehung weiterer Dimensionen bzw. Aspekten wie: Transaktionsmanagement, Sicherheit, Logging …
  - Aspekt jeweils separiert von den fachlichen Belangen des Systems
  - Umsetzung: Aspektorientierte Architekturen

Arc.-Mittel/Prinzipien - WOMIT-15

SAD Mittel

Architektur-Prinzipien – „Information Hiding“:

Klienten (von Systembausteinen) sehen nur (bzw. sollen nur sehen) notwendige Teilausschnitte von Informationen

- Teilaspekt von Information Hiding: Data Hiding
  Beispiel–Objektorientierung: public und private Attribute bzw- Daten (z.B. private Daten können bestenfalls über Methoden abgegriffen werden)
- Information Hiding ist nicht beschränkt auf einzelne Bausteine:
  z.B. Einsatz des Facade-Musters zum „Abschotten“ ganzer Subsysteme – Zugriff nur über eine gemeinsame Schnittstelle möglich
- Information Hiding durch Schichtenbildung (Schichtenarchitektur): jede Schicht sieht nur die jeweils darunterliegende Schicht – Schichten können nicht beliebig aufeinander zugreifen und nur über definierte Schnittstellen (zwischen den Schichten)
  z.B. Darstellungschicht / Business Logic Schicht / Datenbankschicht …
- Black-Box-Prinzip als Ausprägung des Information Hiding Prinzips: Klienten sehen / wissen nichts über die Interna eines Systembausteins, sondern kennen und nutzen nur (bereitgestellte) Schnittstellen

Arc.-Mittel/Prinzipien - WOMIT-16

Architektur-Prinzip – „der Abstraktion“:

Abstraktion bedeutet grob: wichtige Aspekte identifizieren und vereinfachen, Details weglassen

- ist eigentlich ein Spezialfall des Separation of Concern: wichtiges von unwichtigem separieren …
- Anwendung in vielen Bereichen (ausserhalb von SW-Architektur): Ingenieurdisziplinen; SW: Entwurfsmethoden, Architekturbeschreibungen; Prozesse; Programmiersprachen
- Schnittstellenabstraktion als wichtiges Teilprinzip in der SW-Arch.:
  - Explizite Schnittstellen
  - Trennung – Schnittstelle und Implementierung
  - Liskov-Substitutions-Prinzip
  - Schnittstelle-Segregations-Prinzip
  - Sprachunterstützung für Abstraktionen
  - Design-by-Contract

Arc.-Mittel/Prinzipien - WOMIT-17

Architektur-Prinzip – „der Abstraktion“:

Zusammenhang zu anderen Prinzipien und Teilprinzipien:

- Abstraktionsprinzip (insbesondere Schnittstellenabstraktionen) dienen oft der Realisierung von Loser Kopplung
- Abstraktionsprinzip unterstützt Umsetzung des Prinzips der Modularisierung
- Abstraktionsprinzip und Information Hiding:
  Beispiel Portabilität:
  - Architektur und Systembausteine sollen in verschiedenen Umgebungen verwendbar sein
  - Verwendung von Abstraktionen zum/mit Information Hiding zu Plattform-Details
  - Beispiele: virtuelle Maschinen; Datenbankzugriffsschichten

Arc.-Mittel/Prinzipien - WOMIT-18

SAD Mittel

Architektur-Prinzip – „der Modularität“:

Klar abgegrenzte Systembausteine mit den prim. Vorteilen:

- Änderbarkeit
- Erweiterbarkeit
- Wiederverwendbarkeit

Modularität ergibt sich eigentlich „organisch“ bei Anwendung und Kombination der Prinzipien Abstraktion, Separation of Concerns und Information Hiding.

Ansätze zur (Erzielung) von Modularität sind z.B.:

- Objektorientierung
- Komponentenansatz
- Schichten-Architekturen; n-tier – Architekturen

Kriterien zu/für Modularität sind:

- Entwurf (und Dekomposition) des/durch Architekten entscheidend
- Disziplin der Entwickler
- z.B. objektorientiert: gute Klassendefinitionen (keine Alleskönnerklassen)

Offen/Geschlossen – Prinzip (bezogen auf Systembausteine):

- Offen für Änderungen / Geschlossen betreffend interne Details

Arc.-Mittel/Prinzipien - WOMIT-19

SAD Mittel

Architektur-Prinzip – „der Rückverfolgbarkeit“:

- Rückverfolgbarkeit = Traceability
- Forward/Backward Traceability quer durch; RQ-Keys, Kommentare, …

Architektur-Prinzip – „der Selbstdokumentation“:

- Jede Information zu Systembaustein sollte Teil des Systembausteins sein: gute Dokumentation in Code, Architektur-Beschreibungen, …
  Oftmals: kleine Änderungen werden nicht konsequent dokumentiert bzw-. aktualisiert
- Automatische Generierungen: z.B. HTML-Beschreibung zu API mit JavaDoc

Architektur-Prinzip – „der Inkrementalität“:

- SW-Architekturen meist hochkomplex -> können nur iterativ in Inkrementen erstellt werden (Anmerkung: agiler Ansatz …)
- Hinweise und Hints: erste Version relativ rasch bereitstellen, früh Meinung realer Nutzer einholen, neue Funktionalität nur schrittweise einführen
- Schrittweises Wachsen besondere Variante des Prinzips:
  (Anmerkung: agiler Ansatz …, organisches Wachstum - nur begrenzte Vorausplanung)
- Prototyping als Variante des Prinzips

Arc.-Mittel/Prinzipien - WOMIT-20

Weitere Architektur-Prinzipien:

- **Bezug zu Anwendungsfällen:**
  Unbedingte Orientierung an den (geforderten) Anwendungsfällen

- **Vermeidung überflüssiger Komplexität:**
  Weniger ist mehr; komplexes ist fehleranfälliger

- **Konsistenz:**
  durchgängig und konsistent betreff: Namensgebungen, Kommunikation, Schnittstellen, Dokumentation etc.

- **Konventionen statt Konfigurationen:**
  Sinnvolle Standardannahmen (z.B. Parameter setzen und vorversorgen) - nur notwendige Anpassungen müssen (für den Betrieb) konfiguriert werden

Arc.-Mittel/Konzepte - WOMIT-21

Architektur-Konzepte (grundlegende architektonische Konzepte):

Ansätze:

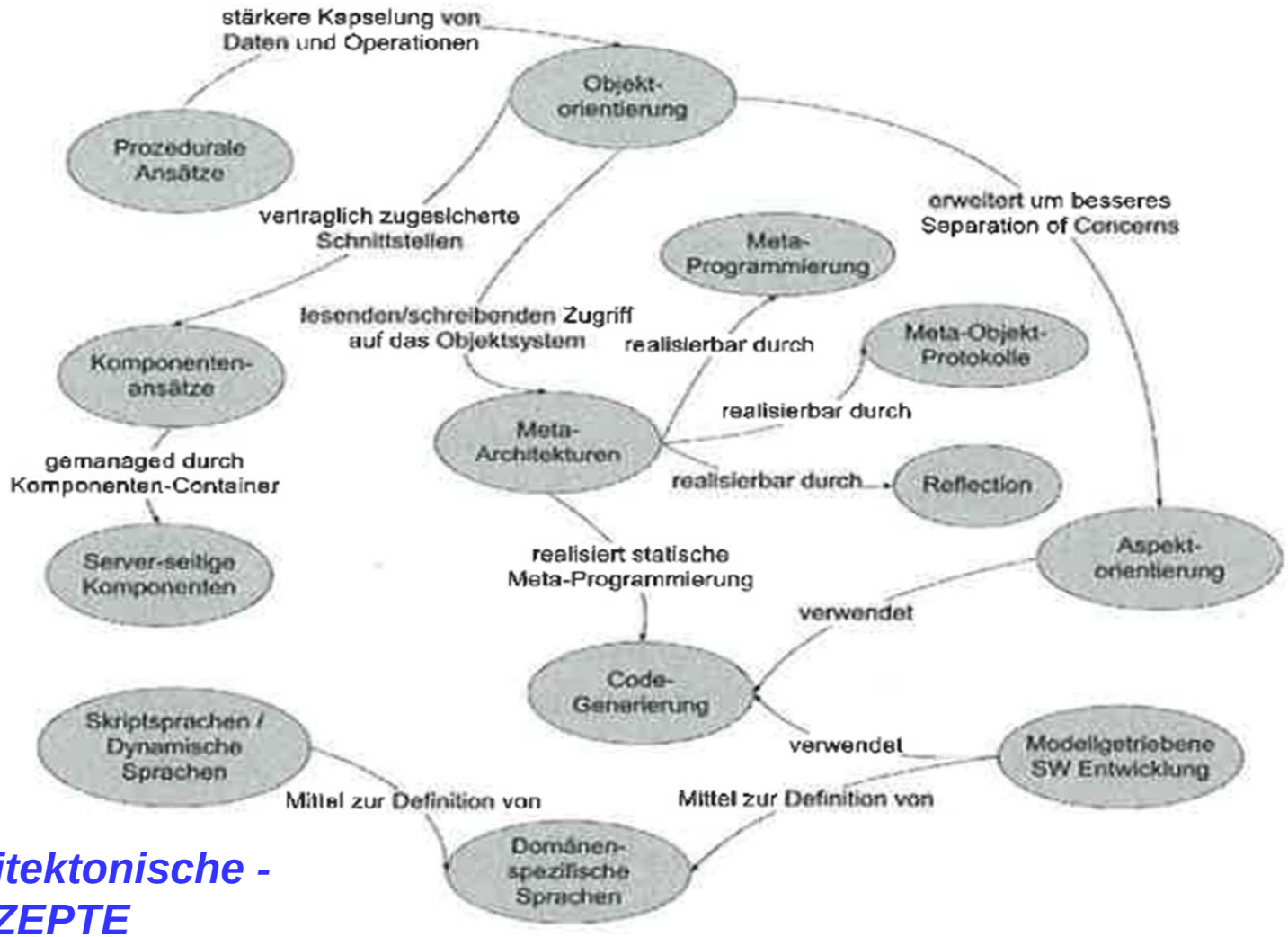
- Prozedurale Ansätze
- …
- Aspektorientierte Ansätze
- Komponentenorientierte Ansätze
- …

Sichtweisen:

- System wird neu erstellt
- System besteht und soll verbessert werden (strukturverbessernde Maßnahmen und zugeh. Mittel)

Arc.-Mittel/Konzepte - WOMIT-22

SAD Mittel



Architektonische - KONZEPTE

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

Arc.-Mittel/Konzepte - WOMIT-24

Architektur-Konzepte – Prozedurale Ansätze:

Prozedural: Funktion, Prozedur, Unterprogramm, Operation, …

- Prozedurale Programmiersprachen: C, COBOL
- Prozedur ist definiert durch: Name der Prozedur, Parameter der Prozedur, Rückgabewert, Prozedurkörper = Algorithmus
- Prozeduren können mit unterschiedlichen Daten und in unterschiedlichen Kontexten durchlaufen werden …
- Prozeduren: lokale Variablen, globale Variablen (Parameter) -> Prinzipien werden verletzt (z.B. globale Variablen: Verletzung von Information Hiding)
- Eigene Prozedur-Bibliotheken: libraries …

Fazit:
„Es ist mühsam und bedarf eines wohldurchdachten Entwurfs, um möglichst alle Architektur-Prinzipien gut in einer prozeduralen Programmiersprache umzusetzen“

=> Objektorientierung

Arc.-Mittel/Konzepte - WOMIT-25

Architektur-Konzepte – Objektorientierung:

Objektorientierung: unmittelbare Umsetzung der Prinzipien Modularisierung, Information Hiding, Abstraktion, ...

- Klassen, Objekte:
  Klassifizierung, Objektidentität, Klasse abstrahiert Objekt, …
- Beziehungsabstraktionen:
  Assoziation, Aggregation, Vererbung, Schnittstellen und abstrakte Klassen
- Polymorphie:
  Objekte und Methoden können (trotz Typisierungen von Klassen) flexibel gehalten werden; Kompilationszeitpolymorphie, Laufzeitpolymorphie
- (OO)Framework:
  definiert eine Architektur für eine Familie von (Sub-)Systemen und stellt elementare Bausteine zur Verfügung; Framework steuert …; Frameworks können einengend sein
- OO und Weiterentwicklungen: Muster, Komponenten, Aspekte, …

Objektorientierung unterstützt im Prinzip besser (als prozedural): Modularisierung, Information Hiding, Abstraktion, ...

-> trotzdem ist wieder der Architekt gefragt

-> nicht alles wird mit OO unterstützt (z.B. nichtfunkt. Anford.)

Arc.-Mittel/Konzepte - WOMIT-26

SAD Mittel

Architektur-Konzepte – Komponentenorientierung:

Komponente: wiederverwendbare, geschlossene Bausteine

- Komponente soll auch nicht-funktionale Anforderungen abdecken, die nicht durch den objektorientierten Ansatz gegeben bzw. abhandelbar sind
- Es gibt auch viele „alte" Komponenten, die nicht objektorientiert sind
- Hauptzweck von Komponenten: Zusammenfügen und Interaktion mit anderen Komponenten über definierte Schnittstellen
- Komponenten sind in sich geschlossen und können unabhängig von einer speziellen Umgebung eingesetzt werden
- Komponenten können von „Dritten" stammen:
  DLLs, JavaBeans, …, CORBA-Komponenten, .NET-Komponenten; Komponentenplattformen
- Komponenten setzen folgende Prinzipien (stärker als Objekte) um:
  Modularisierung, Lose Kopplung (aufgrund von Unabhängigkeit), Separation of Concerns (Trennung fachlicher und technischer Belange)
- Komponenten (bzw. Teile) meist gut wiederverwendbar

Arc.-Mittel/Konzepte - WOMIT-27

Architektur-Konzepte – Meta-Architekturen:

Meta-Programmierung: Erreichung höherer Flexibilität und Kontrolle durch Einführung zusätzlicher Abstraktionsebenen

- klassisch: Programm und Daten; Programme können auf Daten zugreifen, aber nicht auf sich selbst
- (nur) lesender Zugriff: z.B. Java Reflection
- schreibender Zugriff: z.B. Smalltalk
  Zugriff/Änderungen von: Klassendefinitionen, Klassen dazu/wegnehmen
- wird meist als komplex empfunden

Architektur-Konzepte – Baustein-Generierung:

Generierung (ähnlicher) Bausteine unter Einsatz eines Generators („Rahmen ist gleich, nur gewisse Teile sind anzupassen") …

- Schablone/Template für die gleichartigen Teile (den Rahmen)
- Generator kann erzeugen: (**möglichst viele**) gleiche Rahmenteile
- Steuerung des Inputs (etwa anhand von Mustern) und Outputs (anhand von Regeln): Beispiel XLS als Transformation von XML-Dokumenten (kann komplex sein und komplexe Generierungen erlauben)
- Weitere Schlagworte: Frames; „Compiler als Generatoren"; Präprozessoren; Code-Weaving (aspektorientierte Programmierung): Code-Mischungen

Arc.-Mittel/Konzepte - WOMIT-28

SAD Mittel

Architektur-Konzepte – Modellgetriebene Arch.:

MDSD: Model Driven Software Development

- Anwendungslogik ist nicht in einer Programmiersprache ausformuliert, sondern in Modellen
- Modelle müssen so exakt sein, daß sie durch die SW zu erbringende Funktionalität beschreiben können (->eindeutiges Verhalten zur Laufzeit)
- Ausgangspunkt: Anwendungsdomänen und domänenspezifische Sprachen (DSL) – im Mittelpunkt steht das Modell
- Modellplattformen: besteht aus wiederverwendbaren, domänenspezifischen Bausteinen und Frameworks
- Vom Modell zur ausfürbaren Applikation:
  - direkt vom Modell aus (direkte Interpretation)-z.B. Executable UML (OMG)
  - über eine (oder mehrere) Transformation(en) - erfolgt meist über Templ.
- Gewinnbringender Einsatz in der Praxis:
  zusätzlich zu manuellem und generiertem Qellcode und Systembausteinen
- MDSD: fachlich zentrierte, architekturzentrierte (Domäne ist architektonisch motiviert, Transformation erzeugt Implementierungsrahmen mit architektonischem Infrastrukturcode, fachliches wird manuell implementiert

Arc.-Mittel/Konzepte - WOMIT-29

SAD Mittel

Architektur-Konzepte – Modellgetriebene Arch.:

MDA: Model Driven Architecture (OMG)

- ist eine Spezialisierung von MDSD
- Spezialisierung der Transformation liegt in der Standardisierung:
  - der zu verwendenden Modellierungssprache und –architektur
  - eines mehrstufigen Prozesses um über eine Reihe von Transformationen zu einer lauffähigen Anwendung zu gelangen
  - der zu verwendenden Mittel zur Beschreibung der Transform. und Regeln
- Ziele: Interoperabilität und Standardisierung von Modellen für populäre Anwendungsdomänen; z.B.: die gleiche Anwendungslogik mittels verschiedener Transformationen auf verschiedenen Plattformen ablauffähig zu machen; Fehlerhaftigkeit der SW durch den Einsatz verringern; (teilweise) automatisierte Generierungen
- Kosten/Nutzen: ist aufwendig; braucht DSL (Domain Specific Lang.), Modellierungswerkzeuge, Generatoren, Plattformen … ; Analyse der Domäne, DSL => rechnet sich erst bei mehrmaligem Einsatz
- Vorteile von MDSD: Entwicklungseffizienz, Integration der Fachexperten, Änderbarkeit der SW leichter, gute Umsetzung der Architektur, einfache Portierungen möglich; notwendig: saubere Verwaltung in CM-System

Arc.-Mittel/Konzepte - WOMIT-30

Architektur-Konzepte – Aspektorientierte Arch.:

Aspektorientierung-Motivation: Vermeidung von über den Entwurf und den Code verstreute Lösungen für „Crosscutting Concerns“ (z.B. Logging, Sicherheit, Synchronisation, …)

- Aspekt: ein Belang übergreifend/querliegend zur Anwendungslogik
- Aspektorientierung realisiert das Prinzip Separation of Concerns für die „Crosscutting Concerns“
- Der Aspekt wird automatisch in das System eingewoben
- Aspekte sind „nicht-invasiv“: Systeme (Kernprogramm) müssen den Aspekt nicht beachten
- AOP: Aspektorientierte Programmierung (Tools: z.B. AspectJ)
- Beispiel: (nachträglicher) Einbau von Logging – mit Trigger(Join)Points für jeden Methodeneintritt (nicht für jede Methode extra Code-Zeilen für Logging)

- zu AOP: siehe auch Literaturliste zur LVA (z.B. Buch von J.Goll)

Architektur-Konzepte – Skriptsprachen:

Ausgangspunkt: (eigene) Programmiersprachen zur Kontrolle und Steuerung von SW-Systemen

- Kernsystem in einer Programmiersprache implementiert
- Skriptsprache übernimmt nur das Zusammenfügen der Systembausteine in ein lauffähiges System
- Vorteile: Systemnutzer können die Systembausteine flexibel komponieren; für Entwicklung: kann auf hohem Abstraktionsniveau erfolgen; Nachteile: in der Regel weniger performant
- Beispiele für Skriptsprachen: Perl, Python, Ruby, …
- Skriptsprachen sind vollständige Programmiersprachen (meint nicht nur einfache shell scripts); werden oft für „rapid prototyping" verwendet; eingebettet in die Programmiersprache des Kernsystems
- oft auch als dynamische oder agile Sprachen bezeichnet bzw. positioniert
- Skriptsprachen sind gut geeignet um DSLs zu implementieren
- Problem: Pflege der Sprachen; „Breite" der verfügbaren Tools

Arc.-Mittel/Konzepte - WOMIT-32

Architektur-Konzepte – Wartung von Architektur:

Wartung: Verbesserungen schon existierender Systeme – Reengineering (neue Systeme: Forward Engineering)

(alle zuvor für Forward Engineering genannten Prinzipien und Konzepte können auch für Reengineering von „Legacy Systemen"=Altsystemen angewandt werden

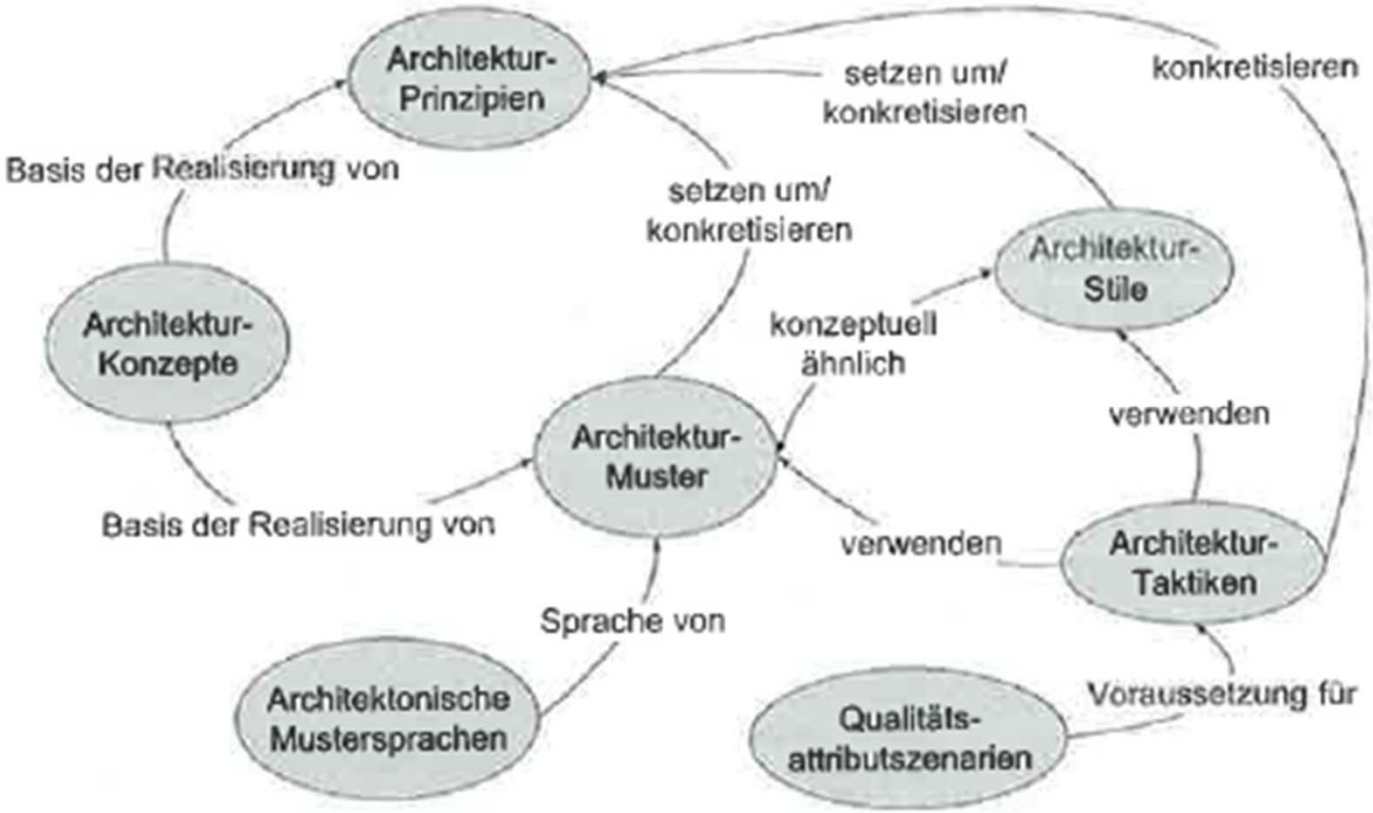
- **Interessante Statistik zur Bedeutung von Reengineering:**
  20% Aufwände für Neuentwicklungen
  40% Aufwände für Verstehen von SW-Systemen
  40% Aufwände für Änderungen von SW-Systemen
- **Gesetzmäßigkeiten und Realitäten betreff SW und ihren Einsatz:**
  Gesetz des ständigen Wandels (Änderungen)
  Gesetz der wachsenden Komplexität
  + „Schlampigkeiten" (z.B. Dokumentation) in der Hektik des täglichen Geschäfts + fluktuierende Personen, Experten, Architekten, Entwickler, …
- **Disziplinen bzw. Aufgaben des Reengineerings:**
  - Reverse Engineering (… verlorene Information zurückholen/generieren)
  - Restrukturierung (… Strukturverbesserungen: Code, Entwurf; Dokum.)
  - SW-Evolution (… Implementierung von Änderungen)
- **Reengineering: nicht nur für Verbesserungen von „Legacy Systeme"**, sondern auch zur bewussten „Modernisierung" von Systemen eingesetzt
- **Einsatz von Reeingineering in der SW-Architektur:** eher gering, da aufwendig und nicht unmittelbar als Mehrwert für die Anwender „verkaufbar"; es gibt Tools …

Arc.Mittel/Stile,Muster-WOMIT-33

SAD Mittel

Architektur – Taktiken, Stile, Muster:

Wesen: Wiederverwendung bewährter Entwurfsentscheidungen (die nachweislich zu erfolgreichen SW-Architekturen geführt haben)



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

Arc.Mittel/Stile,Muster-WOMIT-34

SAD Mittel

Architektur – Taktiken, Stile, Muster:

- Bevor Taktiken, Stile und Muster diskutiert und angedacht werden, sollten **Qualitätsattributszenarien** erstellt und durchgespielt werden – dazu gehören Szenarien betreff z.B.: Verfügbarkeit, Änderbarkeit, Performanz, Sicherheit, Testbarkeit, …
- Funktionale Anforderungen
  -> Modellierung in Anwendungsfällen (Use Cases)
  Nichtfunktionale Anforderungen
  -> Modellierung in Qualitätsattributsszenarien
  Konzentration auf architekturrelevante …
  Verbindungen von Anwendungsfällen und Szenarien berücksichtigen …

Architektur-Taktiken:

- Ist eine Hilfe für den Architekten eine erste Idee zu einem Entwurfsproblem auszuarbeiten. Dazu kann er dann in folge etwa Stile und Muster als weitergehende Mittel verwenden.
- Beispiel: „Lokalisierung von Änderungen" wichtig
  -> Entscheid für eine Schichtenarchitektur

Arc.Mittel/Stile,Muster-WOMIT-35

SAD Mittel

Architektur – Taktiken, Stile, Muster:

Architektur-Stile:

- Ein **Architektur-Stil** gibt primär die fundamentale Struktur eines SW-Systems und dessen Eigenschaften wieder
- **Unterscheidung zu Mustern** nicht einfach (Muster beinhalten zusätzlich Begründungen für di Entscheidung für das Muster …)
  Beispiel: Pipes and Filters als Stil und als Muster (Architekturmuster)

Architektur-Muster (Patterns):

- **Muster sind sehr generisch** – Wurzeln in Definitionen aus der Bau-Architektur – Anwendungen gibt es für viele Bereiche – bezüglich SW-Systeme: **Analyse**muster, **Architektur**muster, **Entwurf**smuster, **Idiome** (Reihenfolge in abnehmender Abstraktion)
- **Muster sind bewährte Lösungen** von wiederkehrenden Problemen
- „Ein Muster ist allgemein eine dreiteilige Regel, die die Beziehungen zwischen einem bestimmten Kontext, einem Problem und einer Lösung ausdrückt".
- **Beispiel – Broker:**
  Kontext: ein verteiltes Objektsystem (Clients greifen über Netzwerk zu)
  Problem: Zugriff über Kommunikation über Netzwerk
  Lösung: Verlagerung aller Kommunikationsaufgaben in einen Broker
- **Muster als „Sprache" von SW-Architekten …**

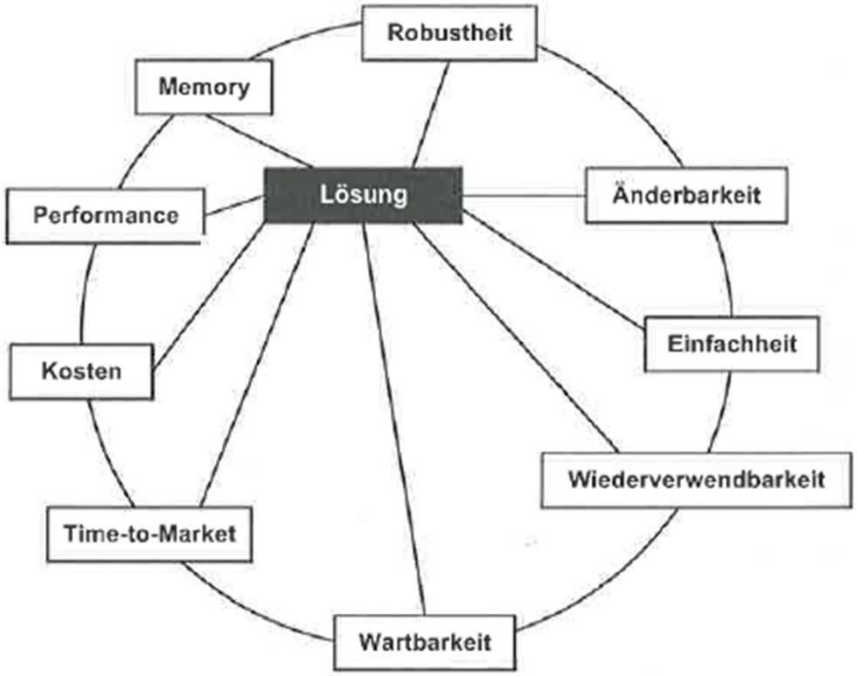
Arc.Mittel/Stile,Muster-WOMIT-36

SAD Mittel

Architektur – Taktiken, Stile, Muster:

Architektur-Muster (Patterns):

- weiterer wichtiger Bestandteil jedes Musters ist ein System von Kräften – „Spannungen" sollen durch die Lösung aufgelöst werden (dazu das optimale Muster unter ähnlichen Mustern eruieren)
- Wichtige Kräfte sind oft Qualitätsattribute:



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

Architektur – Taktiken, Stile, Muster:

Architektur-Muster (Patterns):

Muster – „Schablone" zur umfassenden Beschreibung:

- **Zweck:** wozu dient das Muster
- **Szenario:** Beispielszenario für das Muster
- **Kontext:** technischer/struktureller Kontext indem Problem auftritt
- **Problem:** Problemstellung
- **Lösung:** erprobte Lösung des Problems (inkl. Struktur)
- **Randbedingungen:** eventuelle Einschränkungen/Vorauss.
- **Vorteile:** welche Vorteile entstehen aus der Verwendung
- **Nachteile:** eventuelle Nachteile (Performance; Komplexitäten)
- **Konsequenzen:** Auswirkungen des Einsatzes (siehe auch Vort./Nacht.)
- **Verwendung:** Fälle in denen Vorteile zum Tragen kommen
- **Varianten:** Abwandlungen und Varianten
- **Verweise:** Verweise auf verwandte Muster und Quellen
- **Implementierungen:** programmiersprachenspez. Ref.-Implement.

Architektur – Taktiken, Stile, Muster:

Mustersprachen, -Systeme, Sequenzen, -Kombinationen:

Mustersprachen:
Verwandte Muster werden „verwoben" – bilden ein Vokabular für die Problemstellung UND einen Prozeß zur Lösung
(Muster zu Netzwerken verweben; Sprache: Vollständigkeit)
Einsatz/Umsetzung von Mustern in (kommerzieller) Middleware

Anstrengungen um Ordnung in die (vielen Muster) zu bringen:

- Mustersysteme (nicht vollständig – Sammlung)
- Mustersprachen (nur streng genommen Sprache wenn vollständig bezogen auf einen Kontext: muss Lösungen für ALLE Probleme eines Kontextes enthalten bzw. bereitstellen)
- Musterlandkarten (> 300 Muster !)
- Metamuster

„Gute" Mustersysteme:

- erfüllen bestimmte Kriterien
- bieten gute Ordnung der Muster
- sind offen für Weiterentwicklungen

Architektur – Taktiken, Stile, Muster:

Muster (Patterns) und Anwendungsgebiete - generell:

- Patterns als Mittel des Wissenstransfers
- Patterns und Agile Software Development
- Patterns in Analyse -> Patterns zu Requirementsengineering
- Patterns für Re-Engineering
- Patterns für Dokumentation
- Patterns für Testen
- Patterns für Konfigurations- und Versionsmanagement
- Security and Safety Patterns
- Software Management Patterns
- Software Quality Patterns
- Software Performance Patterns
- Organisational Patterns
- …

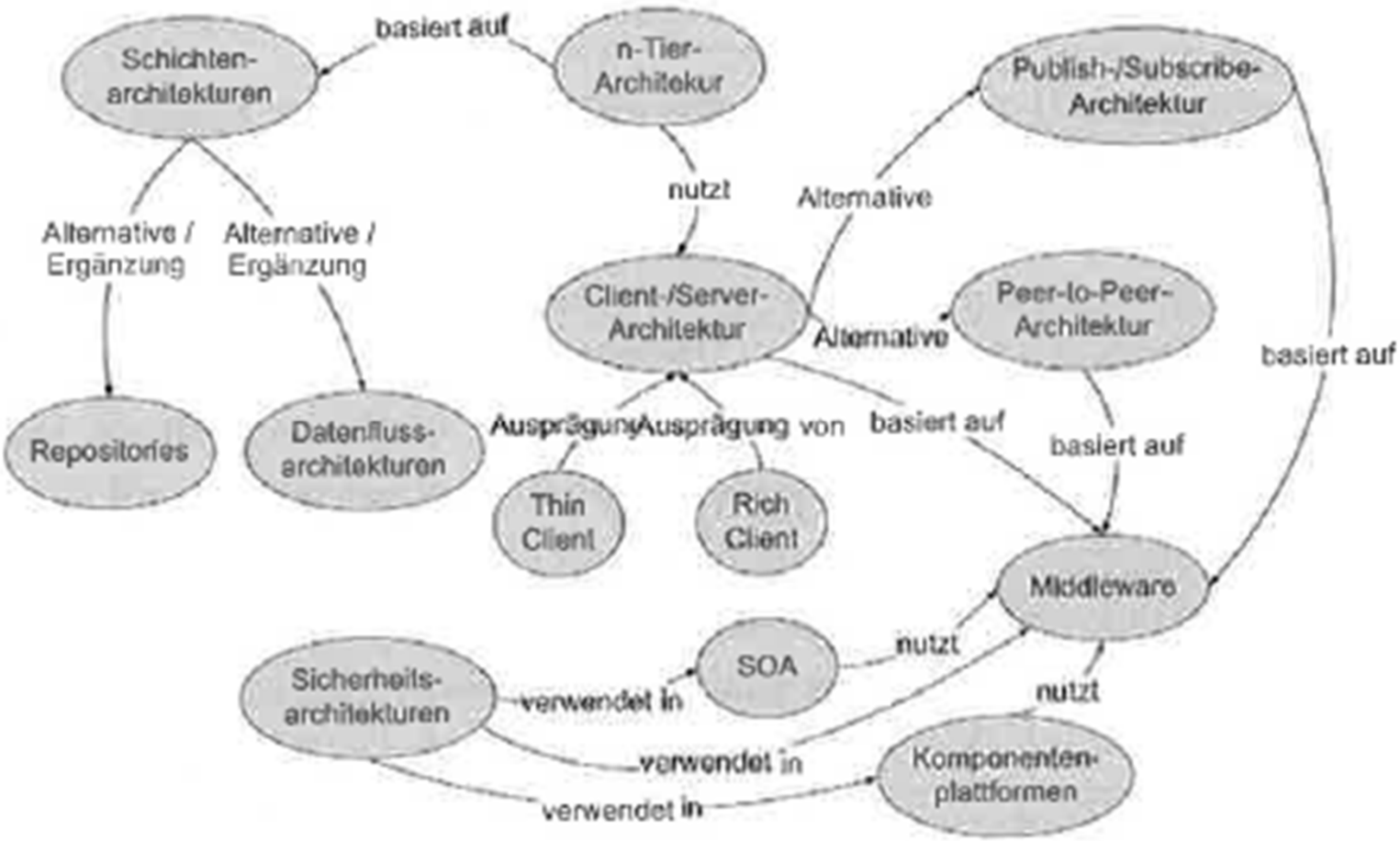
Basisarchitekturen – Überblick und Vorausschau:

- 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

Arc.Mittel/Basisarchit.-WOMIT-41

SAD Mittel

Basisarchitekturen - Übersicht:



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