Fachhochschule Wiener Neustadt
Wirtschaft • Technik • Gesundheit • Sicherheit • Sport

Softwarearchitektur und Design

Block 1 – Grundlagen, Basiswissen; Arch. Ordnungsrahmen

SS2014

DI Dr. Gottfried Bauer

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

Inhalt – Block 1

- **1** – Begriffe und Einordnung
- **2** – SW Engineering, SW Architektur, SW Entwicklung
- **3** – OO Grundlagen - Überblick
- **4** – UML Grundlagen - Überblick
- **5** – OOA & OOD Grundlagen – Überblick
- **6** – Architektonischer Ordnungsrahmen

MEIN Zugang ...

MEIN Zugang zu SW-Architektur

MEIN Zugang zur SW-Architektur

SAD
Analyse, Entwurf

Was sind meine ..... zu SW-Architektur:

- Assoziationen ->
- Kenntnisse ->
- Erfahrungen ->
- Zugänge (bekannt / gewünscht) ->
- Erwartungen (insbesondere an die LVA) ->

SW Engineering ...

SW-Engineering

Software Engineering - Historie

Anfänge der Software-Entwicklung

- Verwendung keiner Methodik
- Systementwicklung durch direkte Implementierung

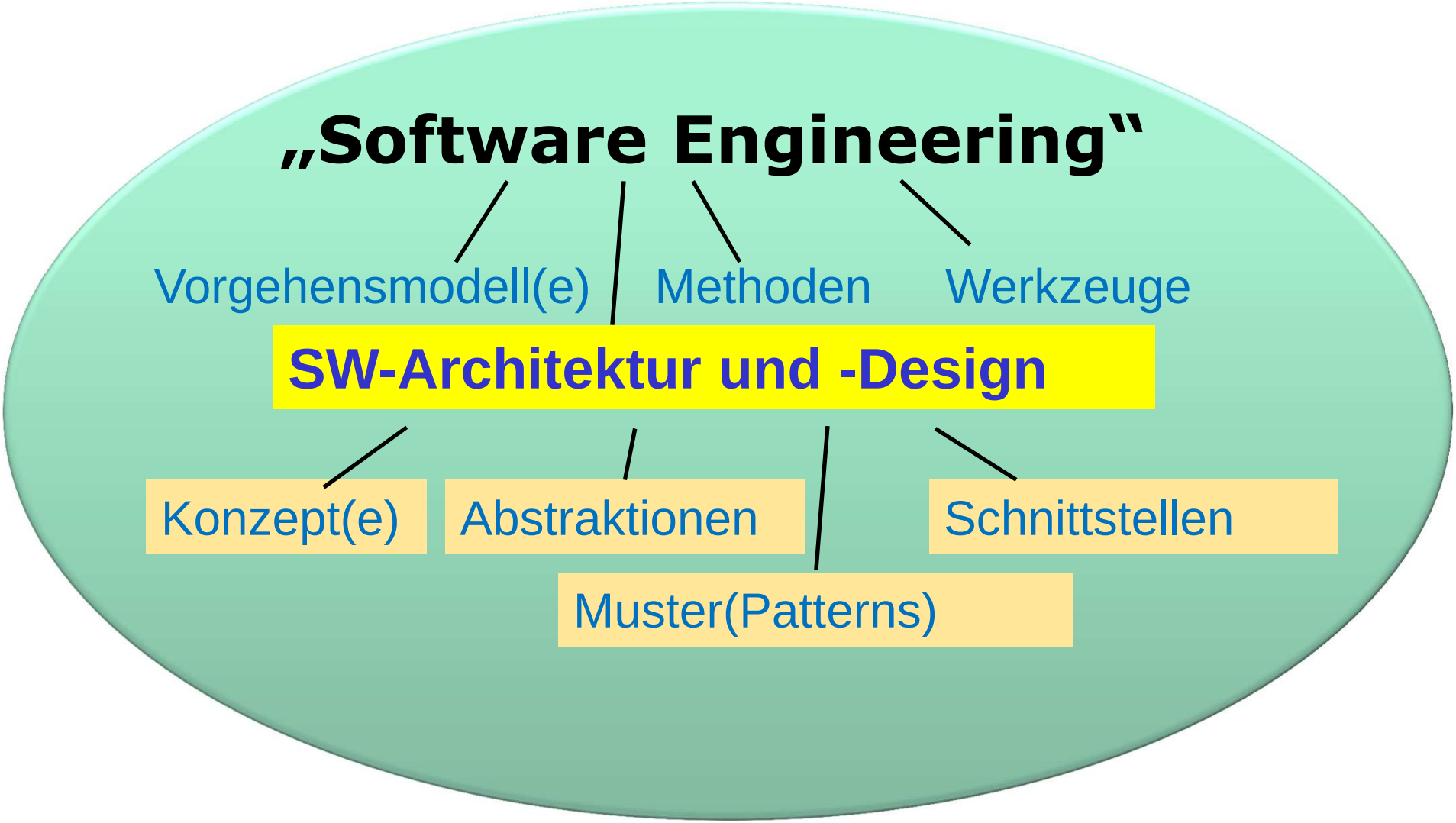
Software-Krise (60-er / 70 Jahre)

- Software immer komplexer
- Zeit- und Kostenüberschreitungen
- mangelnde Software Qualität

-> Entstehung von SW Engineering

- systematische Vorgehensweise
- Vorgehensmodelle
- Mittel zur Qualitätssicherung

Software Engineering - Umfang



Software Engineering - Legende

- **Vorgehensmodelle**
  - Prozessmodell - Ablauf der Phasen in der SW Entwicklung
  - erprobte Vorgehensweise
- **Methoden und Modelle**
  - Vorschrift zur Durchführung (z.B. OO Analyse)
  - Modell um die Idee darzustellen (analog Bauarchitektur)
- **Architekturen**
  - Zerlegung eines Systems
  - Beschreibung des dynamischen Zusammenwirkens
- **Werkzeuge**
  - Unterstützung der SW Entwicklung (CASE Tools, …)

Vorgehensmodelle - Übersicht

- **Traditionell**
  - Wasserfallmodell
  - V-Modell
  - Spiralmodell
  - Rational Unified Process (RUP)
- **Agil**
  - Scrum
  - eXtreme Programming
- **Hybride Modelle**
  - Traditionell versus Agil

Zugang zu SW-Architektur ...

Zugang zu SW-Architektur

Zugang zur SW-Architektur - 2

Warum SW-Architektur ?

- immer komplexere Anforderungen sollen immer schneller und kostengünstiger bei hoher SW-Qualität gefordert -> geht nicht (mehr) mit unsystematischer und unstrukturierter Vorgangsweise

Jedes IT-System besitzt eine Architektur ...

- bewusst geplante Architektur -> Ziel ist **tragfähige Architektur**
- unbewusst und zufällige Architektur („im Kopf", „Bierdeckel")

Problematik und Indikatoren ...

- Zunehmende Divergenz von „Konzept" und Quell-Code Realität
- (permanentes) Flickwerk (um Anforderungen zu erfüllen)
- SW ist ein **„big ball of mud"** (*)
- man weiss nicht mehr genau, warum das SW-System funktioniert
- Umsetzungen und Wartung: sehr schwierig

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

Zugang zur SW-Architektur - 3

Mangelhafte Architektur ...

- Verzögerungen (Termine); Kosten; unzufriedene Kunden; SW nicht einsatzfähig; …
- Indikatoren und Auswirkungen:
  Gesamtüberblick fehlt, Komplexität ufert aus, Komplexität nicht mehr beherrschbar, Planbarkeit ist erschwert, schwer Risiken zu erkennen, Wiederverwendbarkeit erschwert, Wartbarkeit erschwert, Flexibilität eingeschränkt, Integration erschwert, Performanz schlecht, Dokumentation unzureichend, redundanter bzw. mehrfacher Code, „schlechte" Klassendefinitionen, …
- Für jeden stakeholder die richtige Aufbereitung:
  - z.B. für Management: z.B. Pseudoarchitektur
  - z.B. für Entwickler: Architektur bis ins Detail

Zugang zur SW-Architektur - 4

Was ist SW-Architektur ...

- relativ „junge" Disziplin (ca. 30 Jahre)
- widersprüchliche Vorstellungen und Definitionen
- lässt sich nicht scharf definieren (ist „schwer greifbar")
- im Spannungsverhältnis zu Kunden und Management
- Argumente: Konsequenzen in Kosten umrechnen
- nicht nur technische Anforderungen und Bedingungen – auch organisatorische und soziale
- Architekturerfolg -> Projekterfolg
- Architektur hilft, Komplexität überschaubar und handhabbar zu machen
- Abhängigkeiten von jeweiligem Kontext wichtig
- ...

Zugang zur SW-Architektur - 5

SW-Architektur – Definition …

Definition nach IEEE 1471-2000:
„The fundamental organization of a system embodied in ist components, their relationships to each other, and to the environment, and the principles guiding ist design and evolution."

Architekturtreiber …

„Ein Architekturtreiber ist eine Anforderung, welche die Architektur eines Systems beeinflusst bzw. bestimmt.
Architekturtreiber sind:

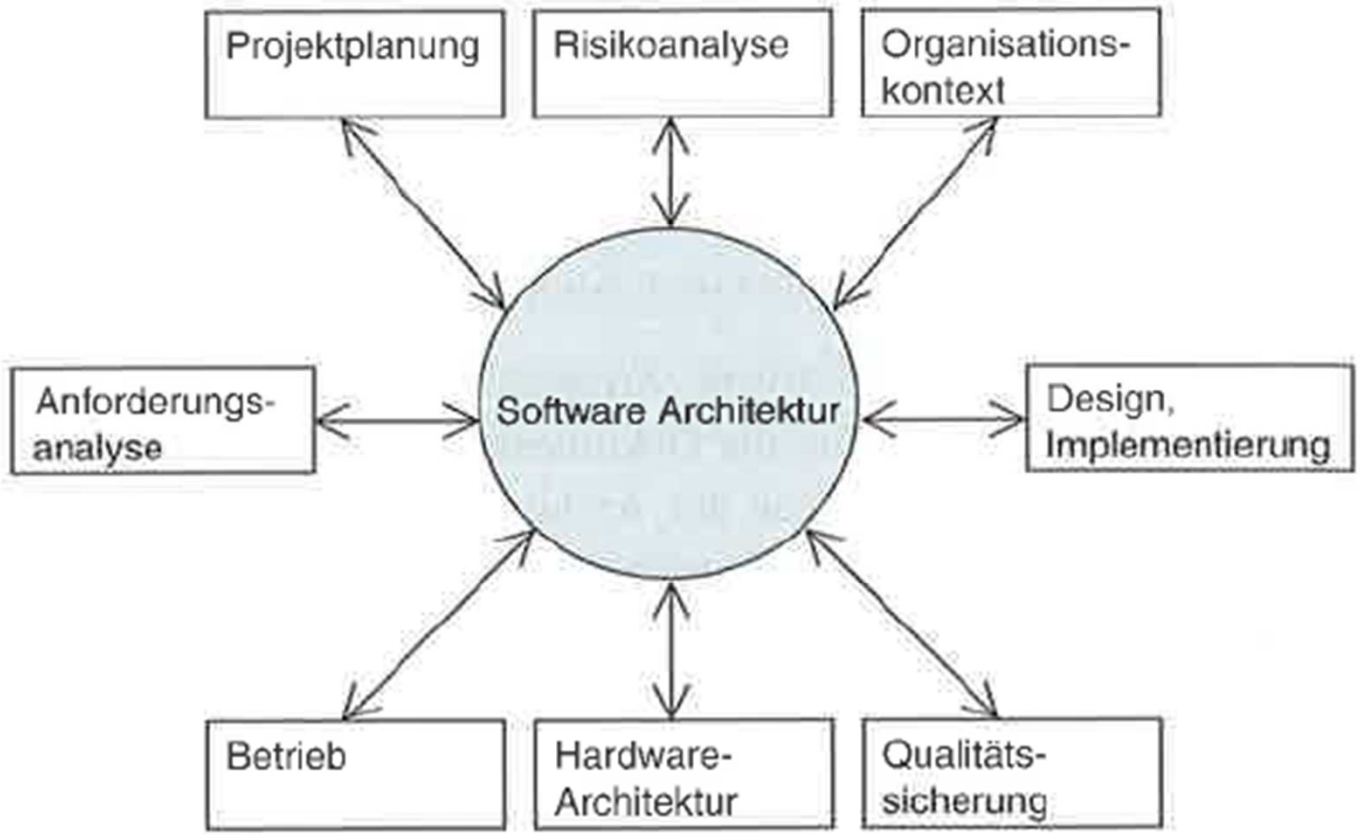
- Funktionale und nichtfunktionale Anforderungen
- Technische Anforderungen und Rahmenbedingungen
- Qualitätsanforderungen
- Wirtschaftliche Rahmenbedingungen
- Rahmenbedingungen in der Organisation

Zugang zur SW-Architektur - 6

Wichtige Fragen betreffend des zu erstellenden System ...

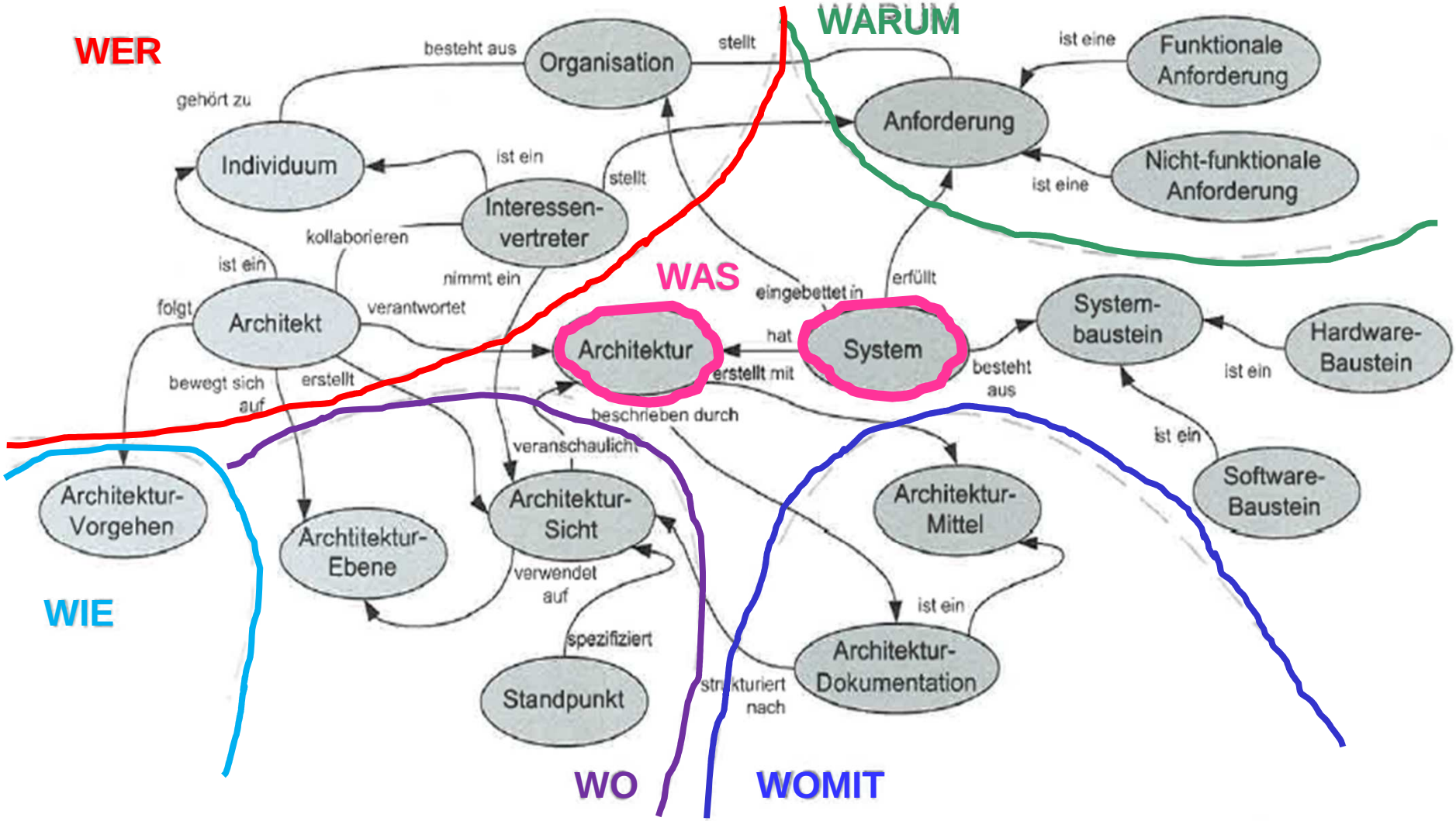
- Auf welche Anforderungen sind die Strukturierung und Entscheidungen zurückzuführen ?
- Welches sind die wesentlichen logischen und physikalischen Systembausteine ?
- Wie stehen die Systembausteine in Beziehung zueinander ?
- Welche Verantwortlichkeiten haben die Systembausteine ?
- Welche Schnittstellen besitzen die Systembausteine ?
- Wie sind die Systembausteine gruppiert bzw. geschichtet ?
- Was sind die Festlegungen und Kriterien, nach denen das System in Bausteine aufgeteilt wird ?

Architektur als Drehscheibe



A.Schatten et.al., Best Practice Software-Engineering, Spektrum, 2010

Mindmap zur SW-Architektur

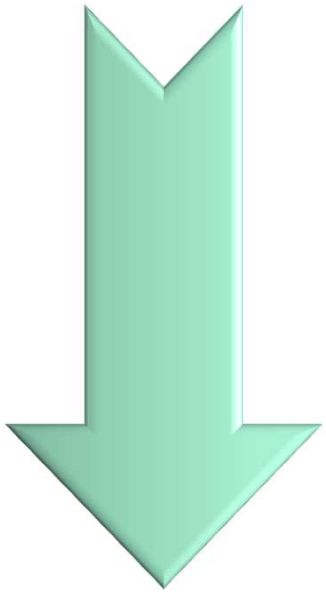


O.Vogel et.al., Software-Architektur, Grundlagen-Konzepte-Praxis, Spektrum, 2009

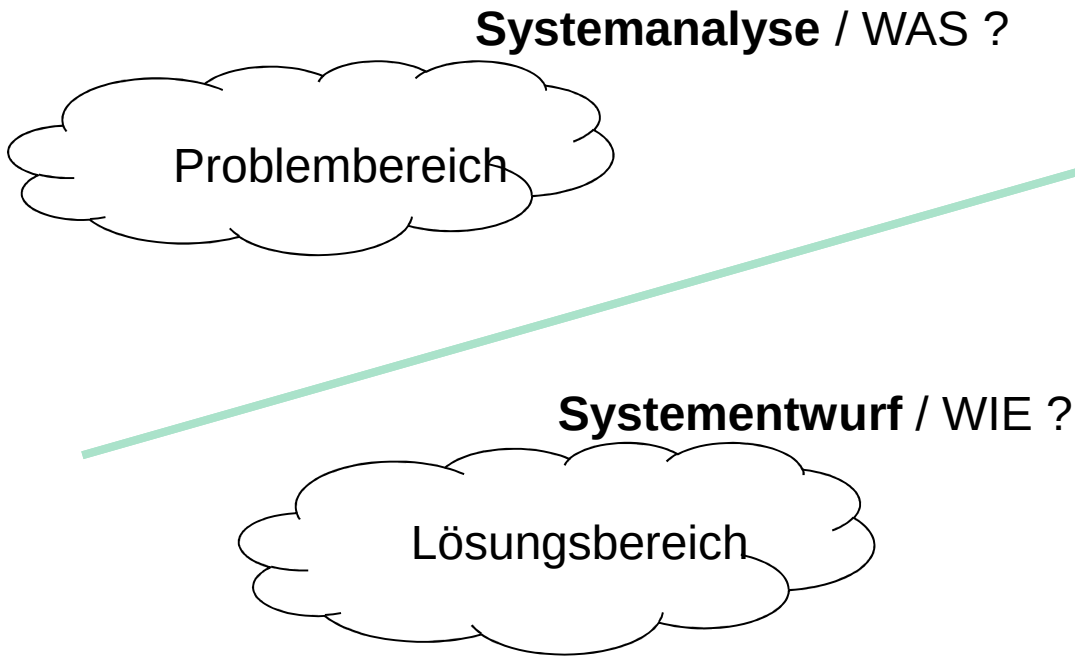
Anforderungen->Architektur->SW

SAD
SW Zyklus

- **Kundenanforderungen**



systematische Vorgehensweise



- **Software an den Kunden übergeben**

Systemanalyse und Systementwurf

Analyse: Erfassen der Kundenanforderungen

- Mit dem Problembereich des Kunden befassen (WAS)
- Welt wird als ideal betrachtet, noch ohne (technische) Einschränkungen
- Sprache des Kunden / Domäne

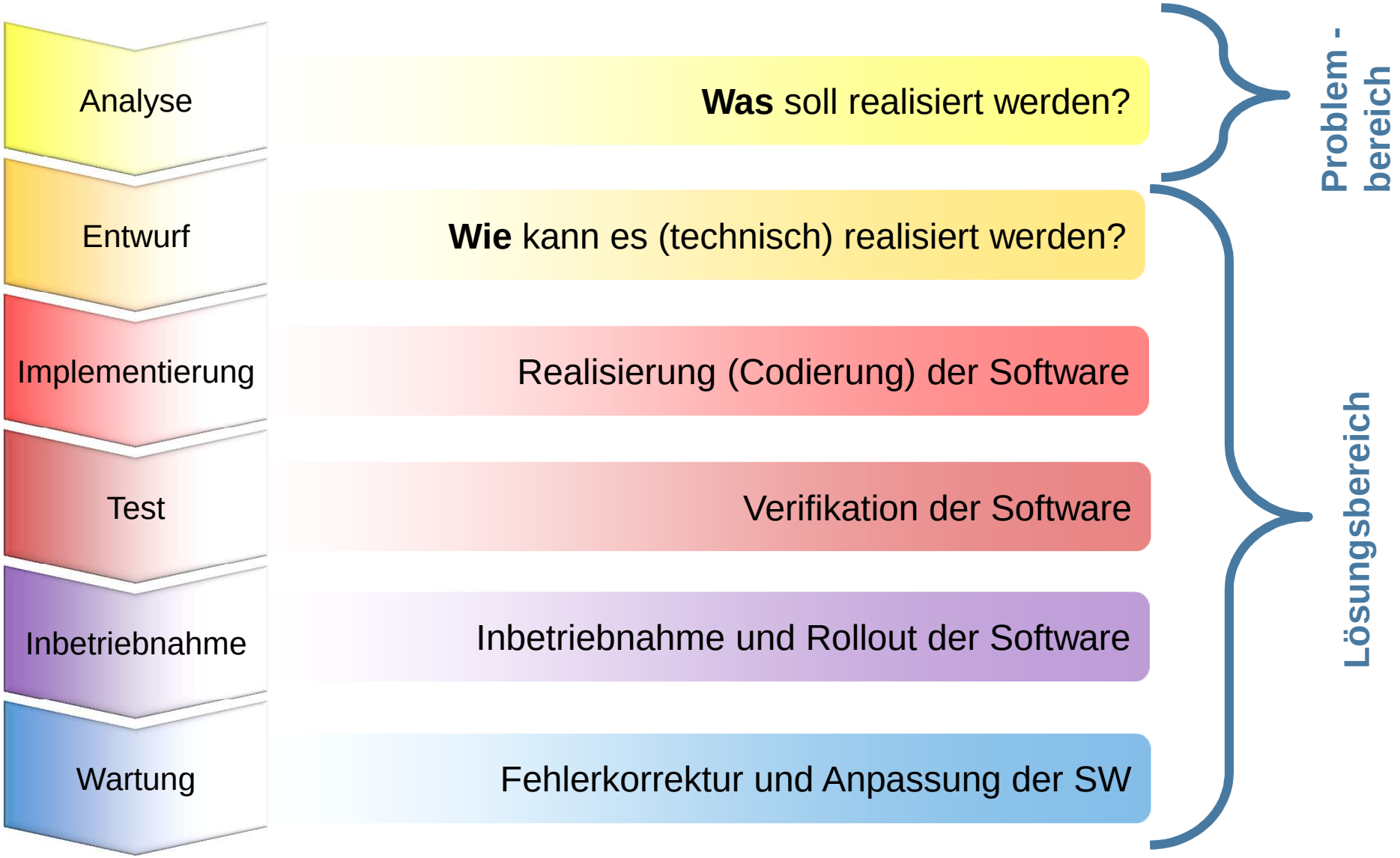
Entwurf: Erarbeitung des Lösungskonzepts

(Basis für die Umsetzung in eine technische Lösung)

- Arbeiten im Lösungsbereich (WIE)
- Welt wird als „ideal" betrachtet, d.h. noch ohne (technische) Einschränkungen
- Sprache des Kunden / der Domäne

Phasen der SW-Entwicklung

SAD SW Entwicklung



Paradigmen der SW-Entwicklung OOx; UML

Paradigmen der SW Entwicklung-1

- **Funktionsorientiert**
  - Funktionale Untergliederung des Systems
  - Schrittweise Verfeinerung - Top Down
  - Methode:
    - Strukturierte Analyse (SA)
    - Strukturiertes Design (SD)

- **Datenorientiert**
  - Modellierung von Daten und deren Beziehungen
  - Funktionen nicht relevant
  - Methode:
    - ERM (Entity-Relationship-Modell)

Paradigmen der SW Entwicklung-2

- **Objektorientiert**
  - Ganzheitliche Sicht –
    keine Trennung von Funktion und Daten
  - Modellierung von interagierenden Objekten
  - Methode:
    - Objektorientierte Analyse (OOA)
    - Objektorientiertes Design (OOD)
- **Aspektorientiert**
  - Weiterentwicklung der Objektorientierung
  - Themen die in vielen Anwendungsfällen vorkommen als Aspekte zentral formulieren und bereitstellen (zB. Logging)
  - Methode: OOA, OOD (erweitert)

Vorteile: objektorient. Methodik

- **Schutz der Daten**
  - Keine Trennung Funktion und Daten
  - Daten sind gekapselt
  - Zugriff über eigene Methoden

- **Verständlichkeit gegenüber dem Kunden**
  - Objekt entspricht Entität der realen Welt

- **Durchgängige Methodik**
  - OOA -> OOD -> OOP
  - Kein Paradigmenwechsel
  - Qualität und Konsistenz wird gesteigert

Vorteile: objektorient. Methodik

- **Übersicht bei großen Systemen**
  - Zusammenfassung in Pakete
- **Wiederverwendbarkeit**
  - Vererbung, Komposition, Aggregation, Polymorphie
- **Stabilität eines Programms**
  - Änderungen lokal begrenzt
  - Vereinfacht Wartung und Erweiterung

Prinzipien der Objektorientierung-1

- **Abstraktions-Prinzip**
  - Abstraktion der Wirklichkeit durch Objekte
- **Objekt-Klassen-Prinzip**
  - „Bauplan" (Klasse) und konkretes Exemplar zur Laufzeit (Objekt)
- **Kapselungsprinzip**
  - Definition der Sichtbarkeit
- **Objekt-Identitäts-Prinzip**
  - eindeutige Identifikation von Objekten

Prinzipien der Objektorientierung-2

- **Vererbungsprinzip**
  - Übertragung von Eigenschaften
  - Generalisierung und Spezialisierung
- **Assoziationen**
  - Objektbeziehungen
- **Polymorphie-Prinzip**
  - Gleichnamige Operationen mit unterschiedlichem Verhalten

Weitere OO-Begriffe

- **Klasse und Objekt (Instanz)**
  - Definition(en) und Unterscheidung
- **UML Notation (für):**
  - Klassen, Objekte
  - Attribute, Methoden, Sichtbarkeit
  - Vererbung, Generalisierung, Spezialisierung
  - Assoziation, Multiplizität, Aggregation, Komposition
  - Polymorphie
  - …

Warum Modellierung

- **Ausgangslage:**
  - Die Vorstellungen des Kunden und das erstellte Programm unterscheiden sich oft.
  - Kommunikationsproblem zwischen Kunden und Softwareentwickler, weil Programmiersprachen ungeeignet für derartige Kommunikation sind.

- **Idee:**
  - Ein nicht technisches und trotzdem präzises Kommunikationsmittel verwenden.

- **Lösung:**
  - Sprache, die das System beschreiben lässt ohne vorerst auf Implementierungsdetails eingehen zu müssen.

Modellierung und UML

Ein Modell …

- ist eine konsistente Vereinfachung der Realität, welche es ermöglicht ein komplexes System (besser) zu verstehen.
- hilft die Struktur und das Verhalten eines Systems zu spezifizieren.
- hilft wichtige Entscheidungen zu dokumentieren.

Mit UML kann …

- die Funktionsweise und das Design eines Software-Systems in einem Modell beschrieben werden, bevor es implementiert wird.
- die System-Modellierung durch eine Menge von miteinander interagierender Objekte erfolgen.

Modellierung von SW - Vorteile

- **Zur Kommunikation zwischen Projektbeteiligten**
  - Ideen und Vorstellungen besser kommunizieren
- **Reduzierung der Systemkomplexität durch Abstraktion**
  - Modelle sind notwendig um komplexe Abläufe zu verstehen
- **Methodisches Problemlösungsvorgehen durch Softwaremodellierung**
  - Problembereich (Was ?) – Lösungsbereich (Wie ?)
- **Beurteilung möglicher Lösungsvarianten im Modell**
  - Form der Qualitätssicherung in frühen Projektphasen

UML Begriffe

- **UML (>=) 2.5**
  - standardisierte **Notation**ssprache (ISO / IEC 19501)
  - Standardisierung durch die OMG (Object Management Group) http://www.omg.org/
  - Sprache zur Beschreibung der OO-Modellierung von Systemen
  - Nicht an ein spezielles Vorgehensmodell, Plattform oder Programmiersprache gebunden
  - Unterstützt OOA und OOD

Achtung: die UML ist keine Methode und kein Vorgehensmodell

Wozu dient die UML ?

Die UML dient zur …

- Modellierung
- Spezifizierung
- Dokumentation und
- Visualisierung

komplexer Softwaresysteme, unabhängig von …

- deren Fach- und Realisierungsgebiet sowie
- einem bestimmten Vorgehensmodell

Vorteile der UML

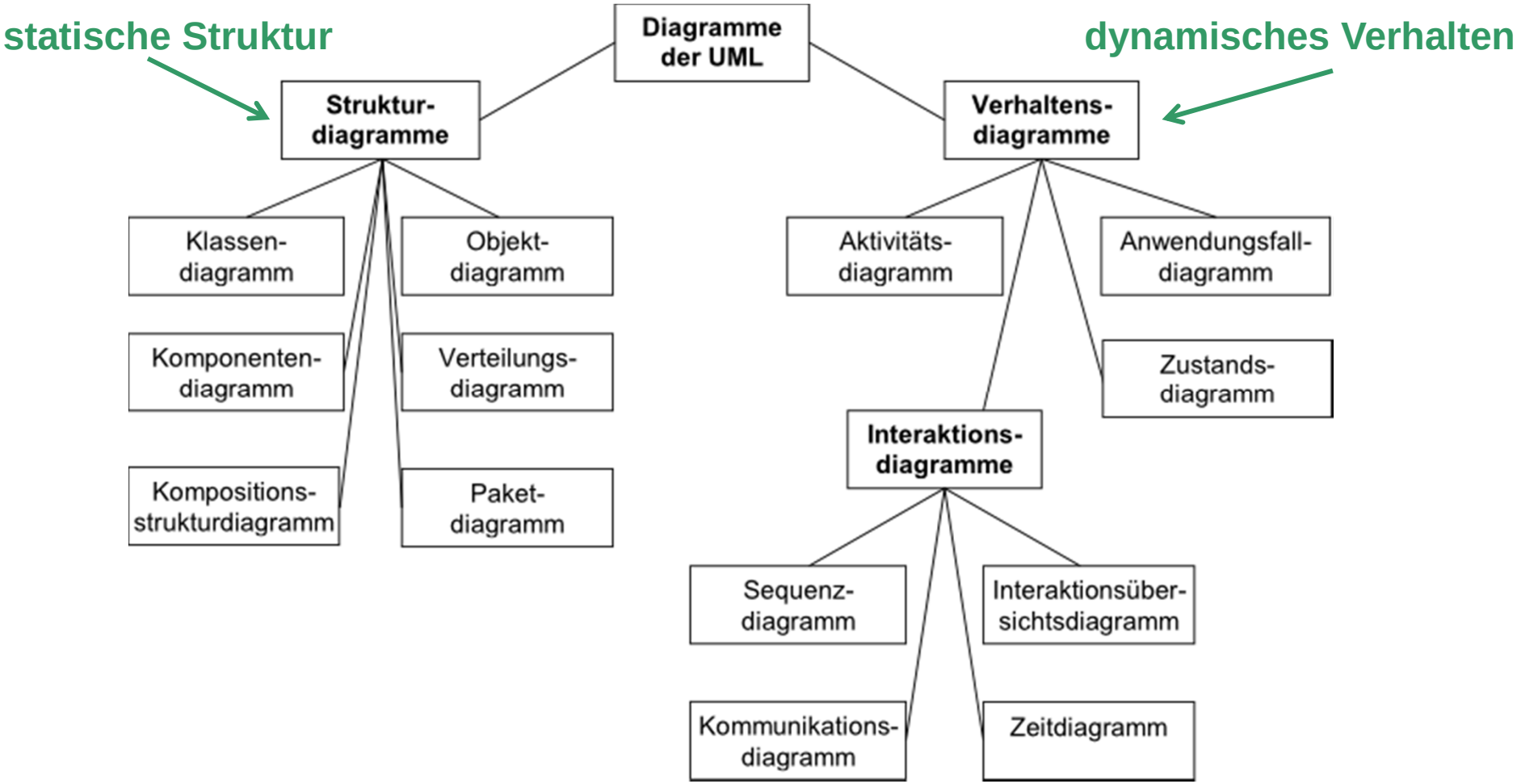
- **Kommunikation mit Anwendern**
  - Einfache graphische Notation
  - Konzentration auf das Wesentliche
- **Kommunikation mit Kollegen**
  - Abstraktion von Konzepten (Architektur)
  - Austausch von Entwürfen
  - Schutz vor übereilter Implementierungssicht
- **Unterstützung der OO Methodik**
  - Unterstützung der OOA und OOD

UML Werkzeuge

- **Werkzeugunterstützung für …**
  - die Erstellung von UML Diagrammen
  - Code-Generierung aus Diagrammen (Forward Engineering)
  - Diagramm-Generierung aus Code (Reverse Engineering)
  - Nahtlose Synchronisation von Diagrammen und Code (Round-Trip Engineering)

- **Liste von UML Werkzeugen**
  siehe http://www.oose.de/nuetzliches/fachliches/uml-werkzeuge/

UML Diagrammtypen



OOx - Übersicht

- **OO-Analyse (OOA)**
  - Erstellen eines Modells des Fachkonzepts
- **OO-Design (OOD)**
  - Ergänzen der Strukturen aufgrund technischer Randbedingungen (Plattform)
- **OO-Programmierung (OOP)**
  - Umsetzen in Programmiersprache und Anpassen an Sprachspezifika

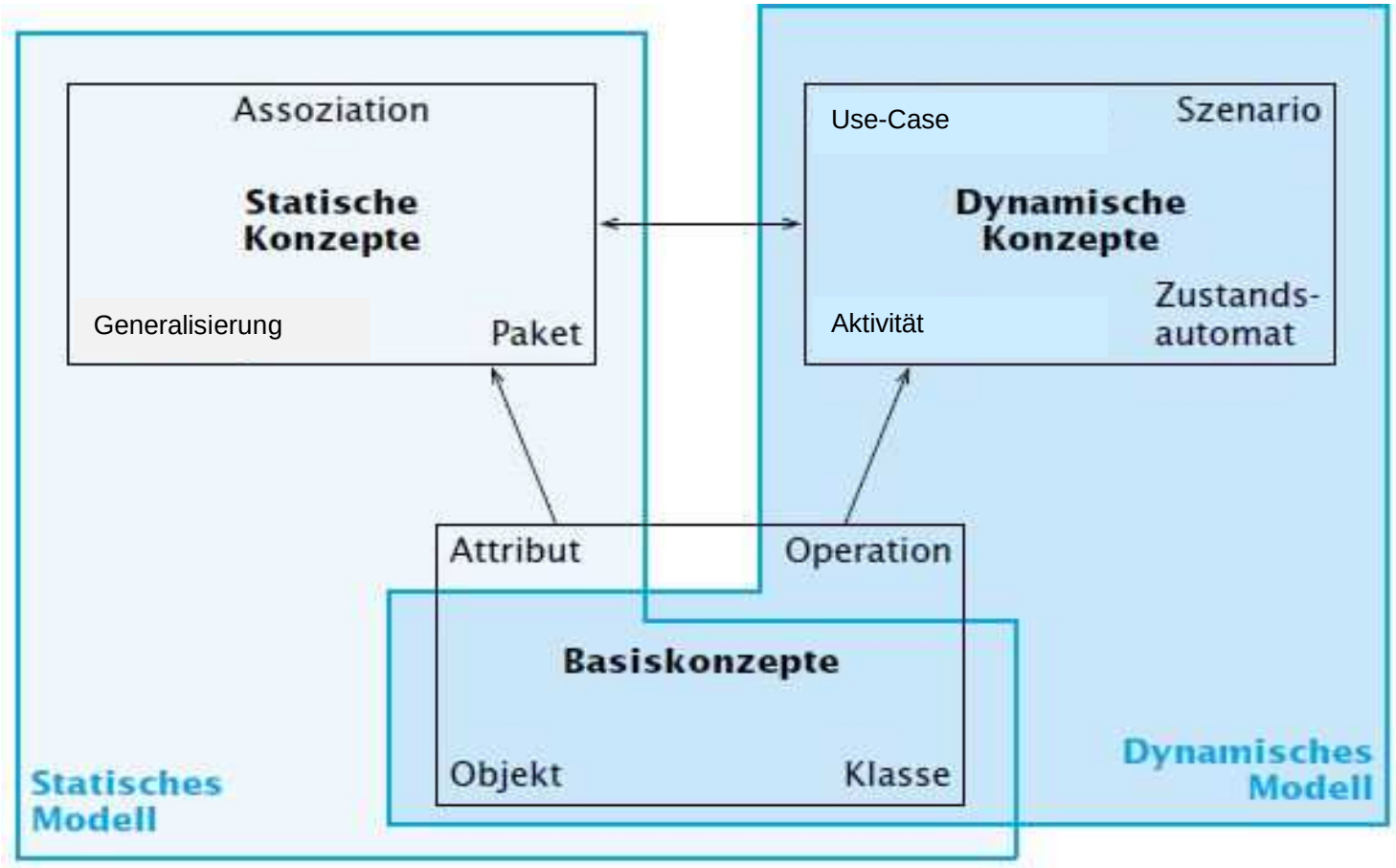
OOA – Ziele, Aufgaben, Modelle

- **Ziele:**
  Anforderungen und Wünsche eines Auftraggebers (bzw. Kunden) an ein neues SW-System ermitteln und beschreiben.
- **Aufgaben (und Artefakte):**
  Es muss ein Modell des Fachkonzepts (**= OOA-Modell**) erstellt werden, das konsistent, vollständig, eindeutig und realisierbar ist. Dieses Modell soll die essentielle Struktur und Semantik des Problems (der Aufgabenstellung) beschreiben.
- **Vorgehen und Prinzipien:**
  Aspekte der Implementierung (Umsetzung) und (techn.) Einschränkungen, Optimierungen etc. ausklammern - rein fachlich
- **Anspruchsvolle und oft schwierige Tätigkeit:**
  Anforderungen sind in der Regel: unklar, widersprüchlich, fall- und kontextorientiert, liegen auf unterschiedlicher Abstraktionsebene und in unterschiedlicher Detailierung vor
- **Systemanalyse als kontinuierlicher Prozess:**
  Informationen: sammeln, filtern, zusammenfassen, dokumentieren
- **Lasten/Pflichtenheft, Anforderungsspezifikation(en):** als Grundlage (Bezugsquelle) zur Erstellung des **OOA-Modells** …

OOA – Modell und Unterteilung

- **Statisches Modell:**
  - Klassen des Systems, Assoziationen und Generalisierung
  - Zusammenfassung in Pakete
- **Dynamische Modell:**
  - Aufgaben des Systems als Anwendungsfälle (Use Cases)
  - Aktivität(en) als Ausführung der Aufgaben
  - Szenarien zeigen die Interaktion von Objekten
  - Zustandsautomaten beschreiben Reaktionen des Systems auf Ereignisse, Botschaften
- **Erstellung und Beschreibung von Artefakten:**
  - z.B. in UML Notation und mit UML Diagrammen
  - Zusammenfassung in Pakete
- **Ersteller:**
  - Systemanalytiker; „Business Analyst“
  - Fachexperten, Domänenexperten (haben eigene Sprache …)
  - Benutzer bzw. Repräsentanten

OOA - Grundlagen



Heide Balzert et.al., Lehrbuch der Objektmodellierung:Analyse und Entwurf mit der UML 2, Spektrum, 2011

OOA – Vorgehen und Schritte

1. Systemkontext beschreiben

- Abgrenzung des Systems zur Umgebung

2. Use-Case Modell erstellen

- Use Cases identifizieren und beschreiben ➔*Anwendungsfalldiagramm*
- Fokus auf Verarbeitung ➔*Aktivitätsdiagramm*
- Fokus auf Zustände ➔ *Zustandsdiagramm*

3. Pakete bilden

- Bilden von Teilsystemen von Modellelementen ➔*Paketdiagramm*

OOA – Phasen/Modelle

Analyse im Großen => **Use Case Modell**

6 Schritte zum **statischen Modell**

4 Schritte zum **dynamischen Modell**

OOA - Statisches Modell

1. Klassen identifizieren

Für jede Klasse nur so viele Attribute und Operationen identifizieren, wie für das Problemverständnis und das einwandfreie Erkennen der Klasse notwendig sind. ➔*Klassendiagramm*➔*Kurzbeschreibung Klassen*

2. Assoziationen identifizieren

Zunächst nur die reinen Verbindungen eintragen, d. h. noch keine genaueren Angaben, z. B. zur Multiplizität oder zur Art der Assoziation, machen. ➔*Klassendiagramm*

3. Attribute identifizieren

Identifizieren aller Attribute des Fachkonzepts. ➔*Klassendiagramm*

4. Generalisierungsstrukturen identifizieren

Aufgrund der identifizierten Attribute Vererbungsstrukturen erstellen. ➔ *Klassendiagramm*

5. Assoziationen vervollständigen

Endgültig festlegen, ob eine »normale« Assoziation, Aggregation oder Komposition vorliegt sowie Festlegung der Multiplizitäten, Rollen und Assoziationsnamen.➔*Klassendiagramm* ➔*Objektdiagramm*

6. Attribute vollständig spezifizieren

Alle Attribute vollständig (Typ, Multiplizität, …) ➔*Klassendiagramm*

OOA - Dynamisches Modell

1. Use Cases präzisieren

Den Ablauf der Use Cases zusätzlich zur textuellen Beschreibung auch graphisch visualisieren (Standardablauf und etwaige Alternativen)

➔ *Aktivitätsdiagramm / Zustandsdiagramm*

Wesentliche dynamische Abläufe von Use Cases als Szenarien beschreiben. ➔ *Sequenzdiagramm / Kommunikationsdiagramm*

2. Zustandsdiagramme erstellen

Für jede Klasse prüfen, ob das dynamische Verhalten durch ein Zustandsdiagramm präzisiert werden muss. ➔ *Zustandsdiagramm*

3. Operationen eintragen

Tragen sie alle Operationen, die beim Erstellen von Szenarien und Zustandsdiagrammen identifiziert wurden, ein.
➔ *Klassendiagramm*

4. Operationen beschreiben

Überlegen, ob eine Beschreibung notwendig ist. Wenn ja, dann ist je nach Komplexitätsgrad die entsprechende Form zu wählen.
➔ *Beschreibung der Operationen in Textform*
➔ *Zustandsdiagramm*
➔ *Aktivitätsdiagramm*

OOD - Grundlagen

- **Ziele:**
  Spezifizierte Anwendung unter den geforderten (technischen) Rahmenbedingungen realisieren (auf höherem, abstrakten Level, als die Implementierung …).
- **Artefakte:**
  Objektorientierter **Entwurf/Design (= OOD) – das OOD Modell**
  - OO unterstützt: kein Paradigmenwechsel von Analyse->Design
  - OOD und Implementierung sind verzahnt (z.B. Klassen aus Entwurf können direkt implementiert werden)
- **Vorgehen und Schritte:**
  - **Randbedingungen für den Entwurf analysieren**
  - **Softwarearchitektur definieren** *(Details folgen laufend)**:***
    - Zerlegung in Teilsysteme
    - Verteilung der Teilsystem auf die HW
  - **Analysemodell verfeinern und präzisieren:**
    - OOD statisches und dynamisches Modell erstellen
    - Berücksichtigung von techn. Randbedingungen (z.B. Effizienz, Wiederverwendbarkeit, Wartbarkeit, Technologien, …

OOD – Architektur festlegen

Architektur über und als Sichten erarbeiten und festlegen:

- **Kontextabgrenzung**
  - Wie ist das System in seine Umgebung eingebettet ?
    ➔ Komponenten- , Paketdiagramm
- **Bausteinsicht**
  - Wie ist das System intern aufgebaut (inkl. SS) ?
    ➔ Komponenten-, Paket-, Klassendiagramm
- **Laufzeitsicht**
  - Wie läuft das System ab ? Wie wirken die einzelnen Bausteine zur Laufzeit zusammen ?
    ➔ Sequenz-, Komm.-, Aktivitätsdiagramm
- **Verteilungssicht**
  - Welche SW läuft auf welcher HW ?
    ➔ Verteildiagramm

OOD – Teil - Statisches Modell

- **OOD - Klassendiagramm**
  - Verfeinerung und Präzisierung
  - „Spiegelbild" des Programms
  - Syntax der Programmiersprache berücksichtigt
- **Paketdiagramm**
  - Logische Zusammenfassung von Systemteilen (Namensräume bilden)
- **Komponentendiagramm**
  - Softwarekomponenten mit definierten Schnittstellen und Zugriffsschutz
- **Verteilungsdiagramm**
  - Verteilung der SW auf physikalische HW

OOD – Teil - Dynamisches Modell

- **Aktivitätsdiagramme**
  - Modellierung interner Systemprozesse
  - Beschreibung komplexer Operationen (Feinentwurf)
  - „Programmierung auf grafischer Ebene“
- **Sequenz- und Kommunikationsdiagramme**
  - Interaktion zwischen einzelnen Architekturschichten / Systembestandteilen
  - Verfeinerung und Präzisierung
- **Zustandsautomaten**
  - Komplexe Lebenszyklen aus der Analyse in OOD Klassendiagramm transformieren

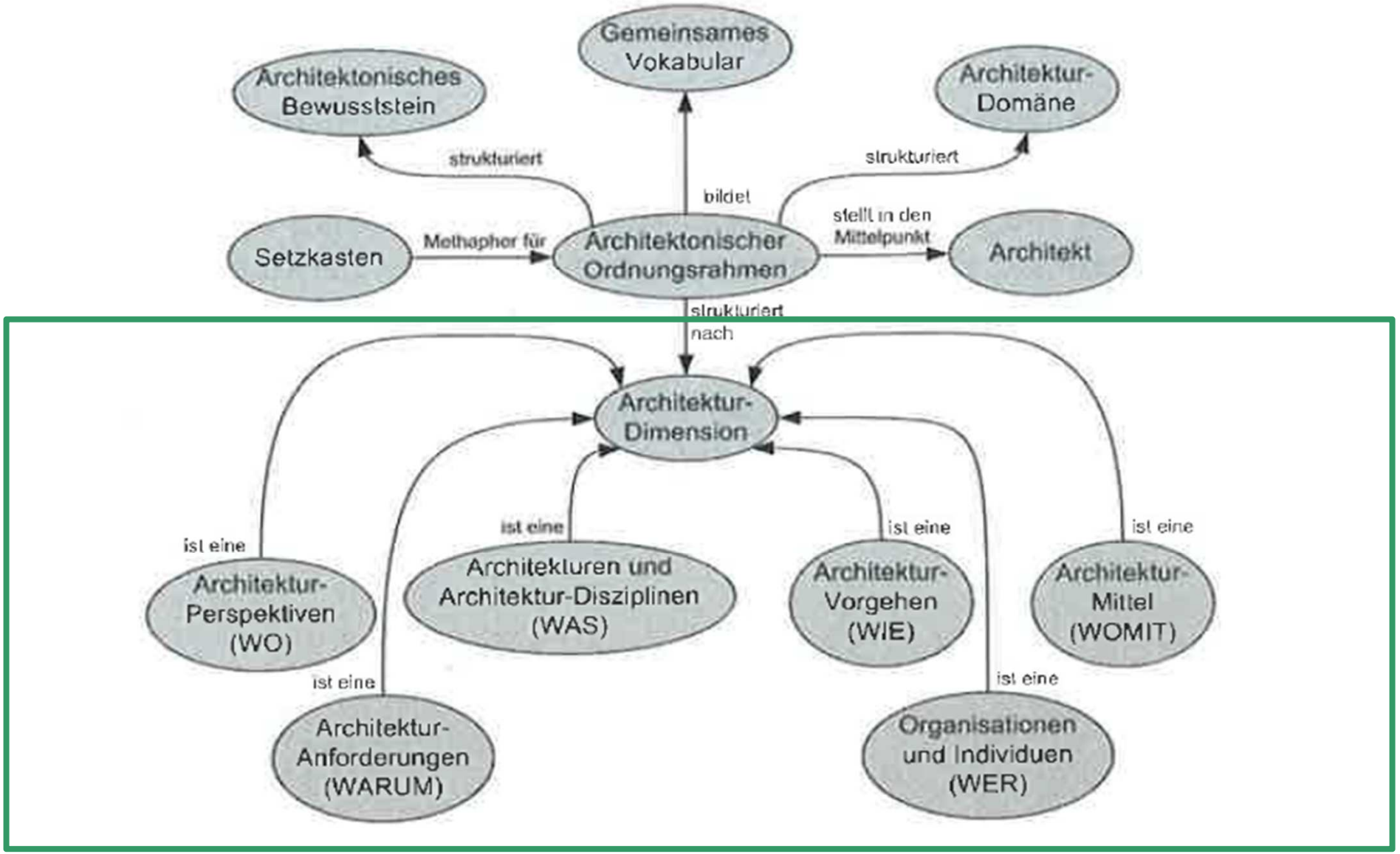
Architekton. Ordnungsrahmen ...

Architektonischer Ordnungsrahmen

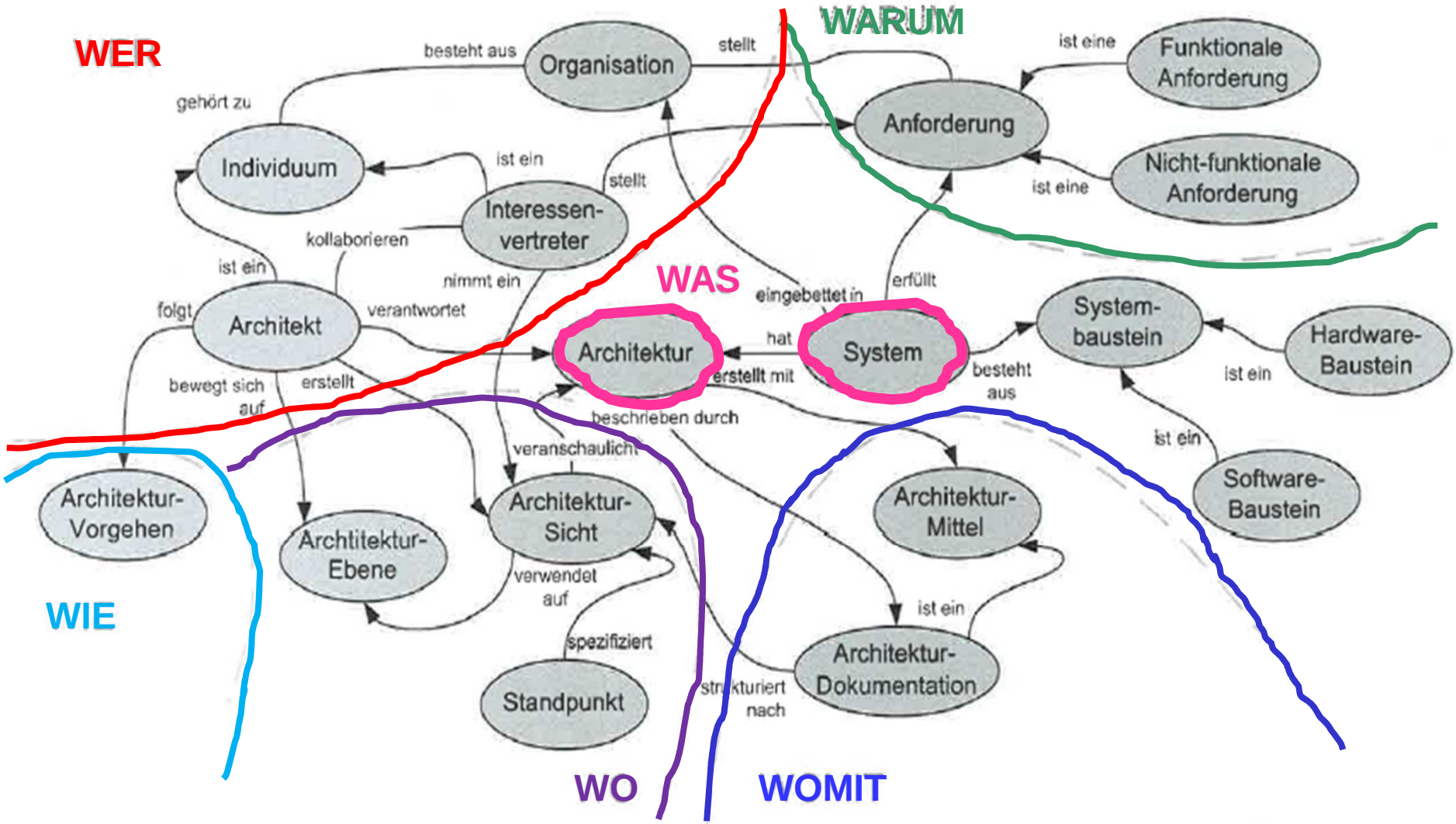
Architekton. Ordnungsrahmen - 1

- **Motivation:**
  - vielfältiges und dynamisches Umfeld (Technologien, Tools, Methoden, Ansätze: SOA, aspektorientiert, ..)
  - Architekt gefordert:
    - Umgang mit Informationsflut (Spreu von Weizen trennen)
    - Treffen von architektonisches Entscheidungen
    - Vorgeben von Richtlinien
    - fachliche Führung des Teams
    - Kundenbedürfnisse aufnehmen und analysieren
    - tragfähige Architektur entwerfen
- **Architekt und (sein) Bewusstsein:**
  - Entwicklung eines Architektonischen Bewusstseins = Setzkasten =
    Metapher für Architektonischen Ordnungsrahmen

Architekton. Ordnungsrahmen - 2



SW-Architektur-Dimensionen ...



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

Architekturdimensionen ...