

# Systementwurf

- Systementwurf allgemein
- Wann ist ein Systementwurf gut?
- Paketdiagramme
- UML Komponentendiagramme
- UML Verteilungsdiagramme

# Systementwurf allgemein

Auswirkungen von **Entscheidungen** verstehen, z.B. Virtualisierung und Cloud-Infrastrukturen, Entwurfsmuster...

→ mit **Module** | **Kohäsion** (cohesion) | **Kopplung** (coupling)

**Entwurfsziele festlegen:** Einschränkung möglicher Alternativen im Entwurf ermöglichen Entwurfsentscheidungen (**Quality Tree**)

→ durch Nichtfunktionale **Anforderungen** | **Termine** mit dem Auftraggeber | **Studium** der Anwendungsdomäne

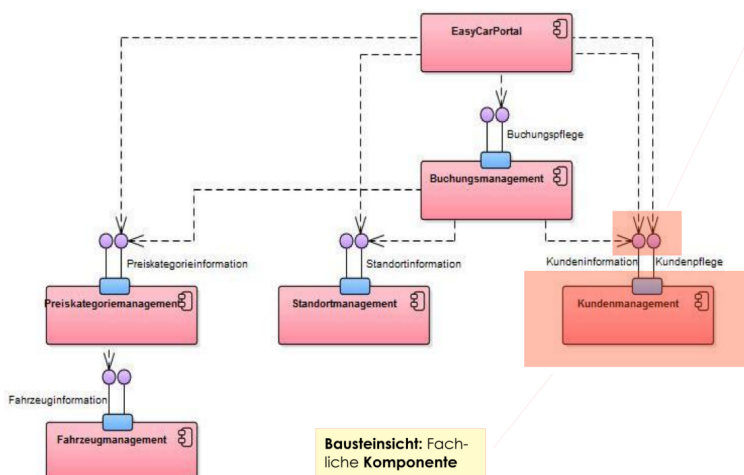
## Subsysteme definieren

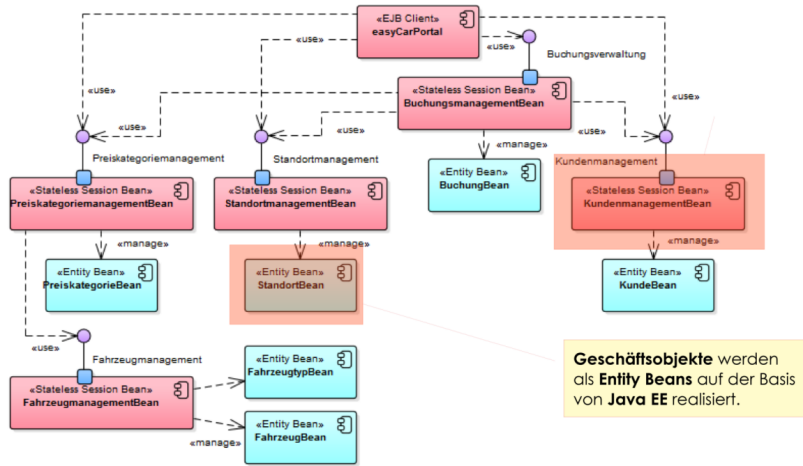
**Softwarearchitektur** – Festlegung der Organisation eines Softwaresystems fest durch Komponenten und Verbindungen

→ Spezifikation der wesentlichen Strukturen eines Anwendungssystems, dh. der Komponenten und Schnittstellen (Bausteinsicht)

**Fachliche Architektur** – Architektur des Softwaresystems aus fachlicher Sicht, d.h. aus Sicht der Fachabteilung

**Technische Architektur** – Realisierung der logischen Komponenten auf der Grundlage der technischen Infrastruktur





**Architekturmuster** – Lösungsansatz, der für Element der Architektur durchgängig und ausnahmslos angewandt wird

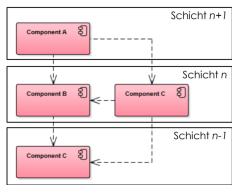
Beispiele: **Datenzentrierte** Architektur | **Schichtenarchitektur** | **Ereignisbasierte** Architektur | **Service-orientierte** Architektur

**Datenzentrierte Architektur:** Komponenten kommunizieren über eine **Datenbank**

→Etablierung eines einheitlichen Datenmodells ist organisatorisch und technisch höchst anspruchsvoll



**Schichtenarchitektur:** *Komponenten mit Zusammenhang werden einer Schicht zugewiesen.*



- **Komponenten** einer Schicht dürfen nur die Komponenten dieser und der Schicht nutzen
- **Aufrufe** passieren Schichten *von oben nach unten*.
- **Ergebnisse** werden *von unten nach oben* durchgereicht.

**Verteilung** – Festlegung, auf welchen *Rechnern* die *Komponenten* einer *verteilten Anwendung* installiert werden

**Datenhaltung | Geschäftslogik | Ergebnisdarstellung** | verschiedene Architekturen

- **Ein-Ebenen-Architektur:** Komponenten der Anwendung werden auf einem Rechner installiert. (Zentralrechner) (ERP-Systeme)
- **Zwei-Ebenen-Architektur:** Komponenten der Anwendung werden auf zwei Rechnern installiert (Client und Server) (Datenbank)
- **Drei-Ebenen-Architektur:** Komponenten auf drei Rechnern (Datenbankserver, Applikationsserver, Client) (ERP-Systeme)
- **Mehrebenen-Architektur:** mehrere Rechnern (Datenbankserver, Applikationsserver, Webserver, Client-Rechner) (Onlineshops)

**Zwei-Ebenen-Architektur**

**Thin Client**

**Smart Client**

# Wann ist ein Systementwurf gut?

## Korrektheit

Ein **Systementwurf** ist korrekt, wenn das **Analysemodell** dem **Systementwurf** zugeordnet werden kann.

Kann jedes **Subsystem** auf eine **funktionale oder nicht-funktionale Anforderung** zurückverfolgt werden?

Kann jedes **Entwurfsziel** auf eine **nicht-funktionale Anforderung** zurückverfolgt werden?

Gibt es für jeden Akteur eine **Zugangskontrolle**?

---

## Vollständigkeit

Ein **Systementwurf** ist vollständig, wenn jede **Anforderung** für den **Systementwurf** berücksichtigt wird.

Wurde das **Anwendungsfalldiagramm** verwendet, um **fehlende Funktionen im Systementwurf** zu entdecken?

Wurden alle **Anwendungsfälle** jeweils einem **Kontrollobjekt** zugewiesen?

Wurden alle **Aspekte** des **Systementwurfs** (Plattform, Datenspeicherung, Zugangskontrolle, Komponenten) behandelt?

Sind **alle Subsysteme** definiert?

---

## Konsistenz

Ein **Systementwurf** ist konsistent, wenn er **keine Widersprüche** in sich trägt.

Sind gegenüberstehende **Entwurfsziele** priorisiert?

Verletzt ein **Entwurfsentscheidung** eine **nicht-funktionale** Anforderung?

---

## Implementierbarkeit

Ein Systementwurf ist realistisch, wenn das System **implementiert werden kann**

Sind **Leistungs- und Zuverlässigkeitsanforderungen** im Rahmen der Zerlegung in Subsysteme überprüft?

Können **Probleme** bei nebenläufigen **Zugriff** auf **Objekte** oder **Subsysteme** ausgeschlossen werden (Deadlocks)?

---

## **Lesbarkeit**

Ein Systementwurf ist lesbar, wenn nicht am Systementwurf **beteiligte Entwickler den Entwurf verstehen** können:

Sind die Namen der Subsysteme (fachlich) nachvollziehbar?

---

**Entwurfsentscheidungen als Sensitivity Points, Trade-offs oder Risiken einordnen.**

# Paketdiagramme

Dienen der **Gliederung** eines Gesamtmodells in **kleinere**, überschaubare und miteinander interagierende **Einheiten**.

**Paket** – **Benennung** einer **Gruppe** von Elementen, die **semantisch verknüpft sind** und sich **zusammen ändern** können.

→ **Elemente** eines **package** können **innerhalb oder außerhalb** des package (mittels Zeichen) dargestellt werden.

→ **Elemente** eines **package** können **öffentlich** oder **privat** sein.

**Pakete sind Zusammenfassungen von Elementen beliebigen Typs**

→ Einzelne Modellelemente (Klasse, Akteur, etc.)

→ Teilmodelle (Klassendiagramm, Aktivitätsdiagramm, etc.)

**Elementimporte** erlauben das direkte **Referenzieren** eines Elements über seinen Namen.

**Paketimporte** erlauben das direkte **Referenzieren** mehrerer Elemente über ihren Namen.



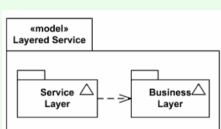
**Öffentlich** (engl. public): <<import>> (optional, da Standardwert)



Das importierte Element wird dem Namespace hinzugefügt und ist auch außerhalb des Namespace sichtbar.

**Privat** (engl. private): <<access>>

Das importierte Element wird ausschließlich dem Namespace hinzugefügt



**Paketzusammenführung**- Erweiterung eines Pakets durch den Inhalt

eines anderen Pakets

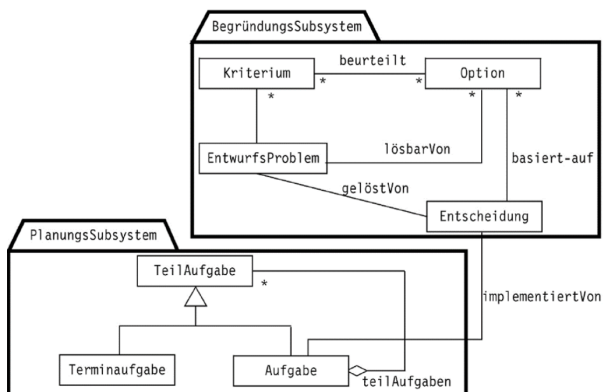
## Modell - Paket, das die Abstraktion eines gesamten Systems darstellt.

### Das Paket soll so strukturiert sein, dass es ...

- den Leser durch das Modell führt,
- in sich **abgeschlossenen Themenbereich** enthält,
- logisch **zusammengehörende Klassen** enthält,
- eine wohldefinierte Schnittstelle zur Umwelt enthält.

Dabei soll die Paket-Schnittstelle ...

- Vererbungsstrukturen in **horizontaler** Richtung schneiden
- **keine Aggregation** durchtrennen und
- **wenige Assoziationen** enthalten



1. Die zu Anwendungsfall Elemente sollten zum Subsystem gehören.
2. Elemente, die zwischen Subsystemen Daten darstellen, sollten in einem **separaten Subsystem** untergebracht werden.
3. wenig **Assoziationen** zwischen Subsystemen → **Lose Kopplung**
4. Elemente eines Subsystems sollten in funktionalen Zusammenhang stehen → **Hohe Kohäsion**

**Kopplung** - Anzahl der Abhängigkeiten zwischen Subsystemen

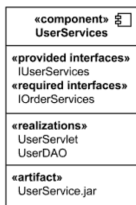
**Kohäsion** - Anzahl der Abhängigkeiten innerhalb eines Subsystems



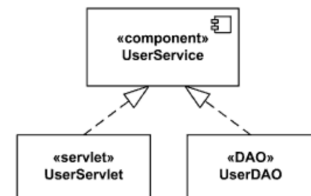
# UML

## Komponentendiagramme

Modellierung von Komponenten, deren bereitgestellte und benötigte Schnittstellen, Ports und Beziehungen zwischen ihnen



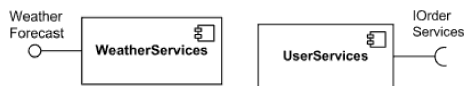
**Logische Komponenten** (z.B. Geschäftskomponenten)



**Physische Komponenten** (Programme, Maven-Packages)

Realisieren **interne Bestandteile** (z.B. Klassen) eine Komponente, können diese innerhalb Komponente in einem separaten **Abschnitt** ( <realization> compartment) notiert werden

### Schnittstellen

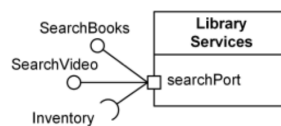


**Bereitgestellte** Schnittstelle wird **durch die Komponente** ealisiert.

**Benötigte** Schnittstelle definiert durch **Verwendungsabhängigkeit** ( <use> dependency) .

### Ports

Über Port kann Element mit Umgebung, mit anderen Objekten oder mit seinen internen Bestandteilen kommunizieren.

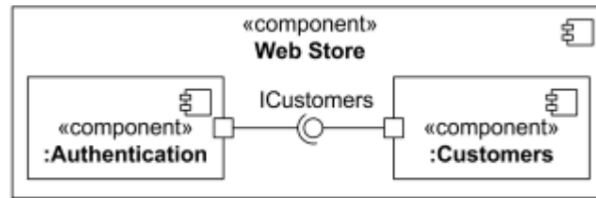


→ Standardsichtbarkeit: öffentlich (public)

→ Angabe von Multiplizitäten optional, z.B. [1..6]

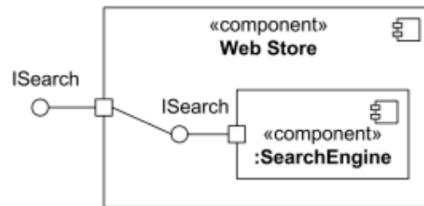
→ Simple port: Port mit nur einer Schnittstelle

## Assembly Connector



**Verbindung** zwischen min. zwei Komponenten, die Dienstbereitstellung und -nutzung zwischen Komponenten darstellt.

## Delegationskonnector



**Verbindung** eines externen Ports bzw. Schnittstelle einer Komponente mit ihren internen Bestandteilen

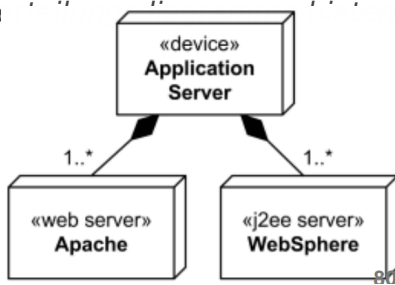
Komponenten sind Paketen ähnlich: Sie definieren Grenzen und gruppieren und gliedern Modellelemente

- Aber Komponenten definieren die Laufzeitsicht:
- Kapselung der enthaltenen Modellelemente
- Expliziter Export von Eigenschaften und Fähigkeiten der enthaltenen Komponenten
- Können Binärcode, Bibliotheken oder ausführbare Programme darstellen
- Können direkt auf Hardware oder Virtualisierungsinfrastruktur ausgeführt werden

Pakete: Logische Sicht auf Codestruktur zur Entwicklungszeit | Komponenten: Physische Sicht zur Laufzeit

# UML Verteilungsdiagramme

Verteilungsdiagramm: Sichten auf ein Anwendungssystem zur Laufzeit



Zeigen, welche **Kommunikationsbeziehungen** zwischen

## Komponenten, Prozessen, Objekten

**Knoten** - Ziel der Verteilung für Laufzeitobjekte und sonstige Artefakte (evtl. hierarchisch)

→ datenverarbeitenden Anlagen (Prozessoren, Computer) einer Systemarchitektur modellieren

### Knoten: Geräte



Eine physische Rechenressource, auf der Artefakte zur Ausführung bereitgestellt werden können

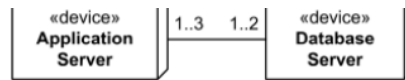
Standard-Stereotypen: **device**, **application server**, **client**, **mobile**

### Spezielle Knoten: Ausführungsumgebung

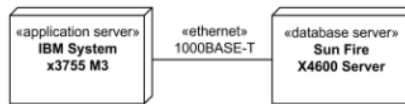


Software für Typen von **Komponenten**, die darauf als ausführbare **Artefakte** bereitgestellt werden

Standard-Stereotyp: **executionEnvironment**, **OS**, **workflow**, **web browser**



## Kommunikationsverbindungen



Zwischen Knoten eines Systems können Kommunikationsbeziehungen bestehen

Zwischen **Geräten**: physische Verbindungen, z.B. <ethernet>

Zwischen **Ausführungsumgebungen**: Kommunikationsprotokolle, z.B. <protocol> TCP/IP

## Artefakte und Laufzeitobjekte



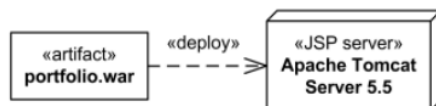
Auf Knoten können Laufzeitobjekte und sonstige Artefakte bereitgestellt und ausgeführt werden.

Standard-Stereotyp: <artefact, file, ...>

→ Konkrete physische Darstellung eines oder mehrerer Modellelemente durch ein Artefakt

→ Artefakte können von anderen Artefakten **abhängig** sein

## Deployment



Bereitstellung (engl. deployment) - Zuordnung Artefakts zu einem Verteilungsziel

→ Darstellung als Abhängigkeit (dependency) mit Stereotyp <deploy>

→ Kann auch als in einem Knoten enthalten dargestellt werden Kann auch auf Instanzebene dargestellt werden

**Relevante Fragestellungen:**

→ Welche Kommunikationsbeziehungen bestehen zwischen diesen Komponenten, Prozessen und Objekten?

→ Arten der Modellierung: Können auf Typebene (Spezifikationsebene) und auf Instanzebene verwendet werden!