5 Design

5.1 Grundlagen

Aufgaben des Designs

- Ziel: **Strukturierung des Systems** → klare Vorgabe für Implementierung
- Subsysteme > Komponenten > dessen Komponenten > usw. → hierarchische Struktur

- **Systemarchitektur:**
  a) Definition von **Subsystemen (= Teilsystemen)**
     – Spezifikation von **Komponenten** mit ihren **Schnittstellen**
     – **Verteilung** der **Komponenten auf HW** und **Systemsoftware**
     – Interaktion der Subsysteme
     ⇒ liefert abstrakte Systemsicht (abstrakter als Klassendiagramm)

  b) Implementierungsmodell
     – **Klassenmodell** mit vollständig spezifizierten (Kern-)Klassen
     – beschreibt neben Fachlichkeit auch **technische Aspekte** (Kommunikation, GUI, Persistenz, ...)
     ⇒ liefert unmittelbare Vorgabe für die Realisierung

Software-Architektur

Menge der grundsätzlichen Entwurfsentscheidungen – unter Berücksichtigung nichtfunktionaler Anforderungen

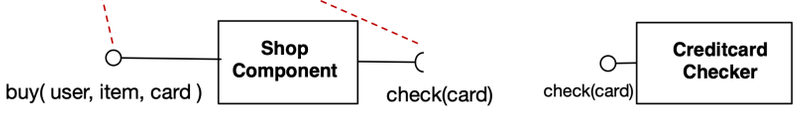
Komponenten

Komponenten = zentrale **Bausteine** der Software-Architektur

- **Wohldefinierte Schnittstelle:** man kann die Komponente von außen nutzten, ohne in dieser reinzuschauen, weil die Schnittstelle klar definiert ist.

- Definition **Komponente** (component) (analog UML):
  Medium der Komposition von Systemen mit wohldefinierten Schnittstellen:
  – **provided interface** definiert die angebotenen Dienste,
  – **required interface** definiert die benötigten Dienste

kapseln eine **Teilmenge** der **Systemfunktionalität / Daten**, z.B. Menge von Klassen, Menge von Packages



- Komponenten **können hierarchisch aus anderen Komponenten zusammengesetzt sein** (= **Komposition**)
- Komponenten besitzen bestimmte **Qualitätsattribute**
  - nichtfunktionale-Anforderungen die sie erfüllt wie z.B. Antwortzeit

Entwurfsprinzipien

Entwurfsprinzipien – Modularisierung

- Gliederung der Architektur eines Software-Systems in überschaubare Bestandteile/ SW-Bausteine

Vorteile von Modularität

- strukturierte Zerlegung in kleinere Einheiten macht **die Komplexität einfacher**
- **Verständlichkeit**

zu viel Funktionalität:

- zu komplex und fehleranfällig, schlecht wiederzuverwenden

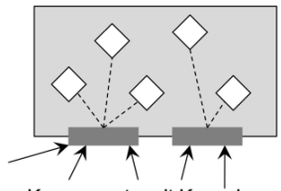
zu wenig Funktionalität:

- Komplexität durch viele kleine Komponenten und unübersichtliche Interaktionen

---

Entwurfsprinzipien – Kapselung

- Komponenten mit Schnittstellen verbergen Implementierungsdetails
- Außenwelt kennt nur die Schnittstelle; interne Struktur einer Komponente von außen nicht sichtbar oder zugreifbar



Vorteile von Kapselung:

- interne Änderungen in einer Komponente haben keine Auswirkungen nach außen (**Änderbarkeit**)
- zur Nutzung der Komponente muss man nur Schnittstelle kennen (**Verständlichkeit**)

Entwurfsprinzipien – Trennung von Zuständigkeiten

- eine Komponente hart nur einen Aufgabenbereich!

Vorteil von Trennung von Zuständigkeiten:

- besseres Verständnis, leichtere Wartbarkeit

Komponente, die mehrere Aufgabenbereiche abdeckt, ist unnötig komplex → **schwer verständlich, schlechter wartbar**

Kriterien zur Trennung von Zuständigkeiten, u.a.

a) trenne fachliche und technische Komponenten
b) grenze unterschiedliche fachliche Gebiete ab
c) trenne Funktionalität und Interaktion (z.B. GUI)
z.B. umgesetzt im **Model-View-Controller-Pattern**
d) ...

Entwurfsprinzipien – Kohäsion

starke Kohäsion (innerhalb Komponenten):

- Maß für inneren Zusammenhalt eines Elements
- SW-Komponente hat eine genaue Aufgabe

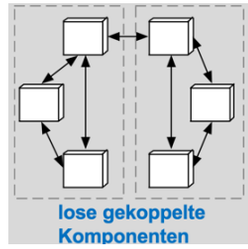
Vorteil von Trennung von starker Kohäsion:

- kohärente Komponenten sind **einfacher zu verstehen und zu implementieren** (Verständlichkeit)
- kohärente Komponenten sind **seltener Änderungen** unterworfen (Robustheit), **weil sie nicht viele verschiedene Dinge realisieren**
- **Redundanzfreiheit**
- bessere **Teambildung**

Entwurfsprinzipien – Kopplung

lose Kopplung (zwischen Komponenten):

- Komponenten kennen nur wenig andere Komponenten



Vorteil von Trennung von loser Kopplung:

- Änderungen einer Komponente hat wenig Einfluss auf andere Komponenten (**Wartbarkeit, Änderbarkeit**)
- lose gekoppelte Komponenten kennen weniger von ihrer Außenwelt → **stärkeren Kohäsion**

5.2 Design – Ergebnisse

Ergebnisse des Designs im Überblick

1. **Deployment - Modell(Verteilungsmodell)**
   - physikalische Struktur des Systems

2. **Zerlegung in Subsysteme(=Teilsysteme)/Komponenten**
   - logische Struktur des Systems

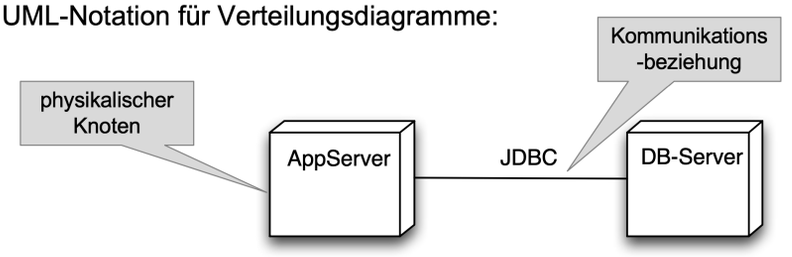
Deployment-Modell

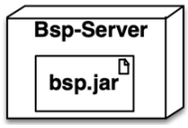
- **Deployment-Modell** beschreibt **physikalische Struktur** des Systems
  - welche SW-Bausteine laufen auf welchen Ausführungseinheiten (Hardware-Komponenten mit Systemsoftware (= Anwendungsserver, RDBMS, ...))

1. Deployment-Modell: UML-Verteilungsdiagramm

- Verteilung einer SW-Anwendung erfolgt in UML auf Knoten
  - jeder Knoten repräsentiert Ausführungsumgebung, z.B. PC, Anwendungsserver, Betriebssystem, Smartphone
  - Beziehungen repräsentieren eine Kommunikationsverbindung

UML-Notation für Verteilungsdiagramme:

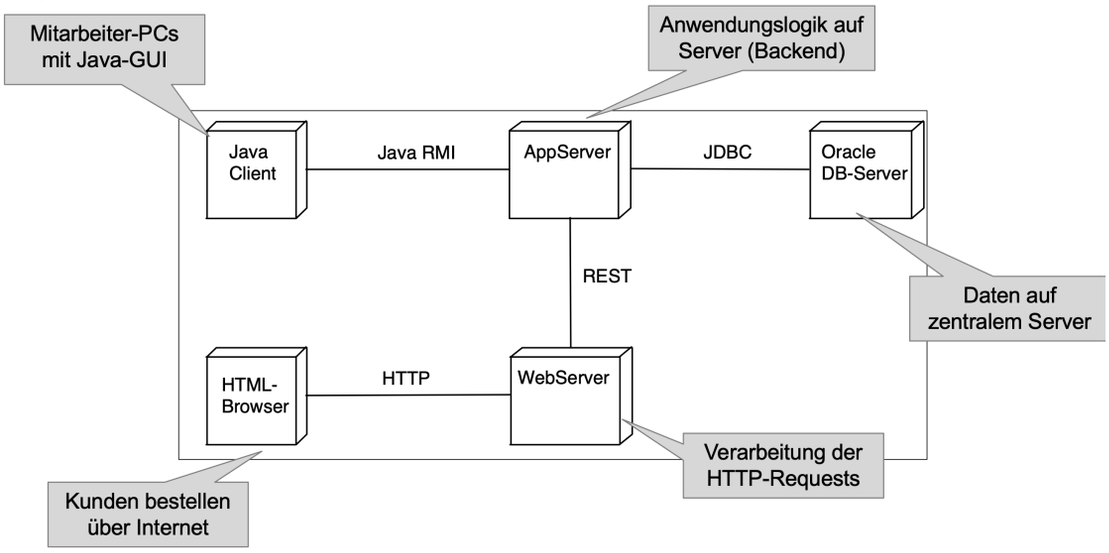




- Knoten können ineinander geschachtelt sein
- die Artefakte einer Anwendung werden auf die Knoten verteilt

Bsp. Verteilungsdiagramm für **Internet-Buchhandlung**

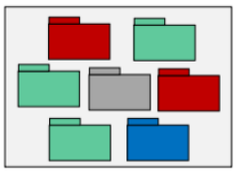
• physikalische Knoten (= Hardware)



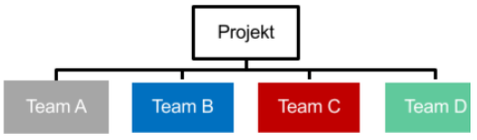
2. Zerlegung in Subsysteme

- Strukturierung des Softwaresystems in handhabbare Teile (⇒ Modularisierung)
  - **Verantwortung** jedes Subsystems muss definiert sein (⇒ separation of concerns)
  - Subsysteme kommunizieren über definierte **Schnittstellen** (⇒ information hiding)
  - **Schnittstellen** zwischen Subsystemen müssen exakt beschrieben sein (⇒ Kapselung)

- logische Systemstruktur bestimmt die Projektorganisation

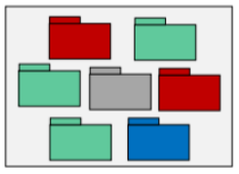


Systemstruktur
7 logische Komponenten

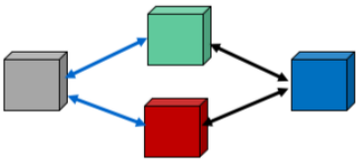


Projektorganisation
4 Teilteams

- physikalische Struktur erfordert passende logische Struktur



Systemstruktur
7 logische Komponenten

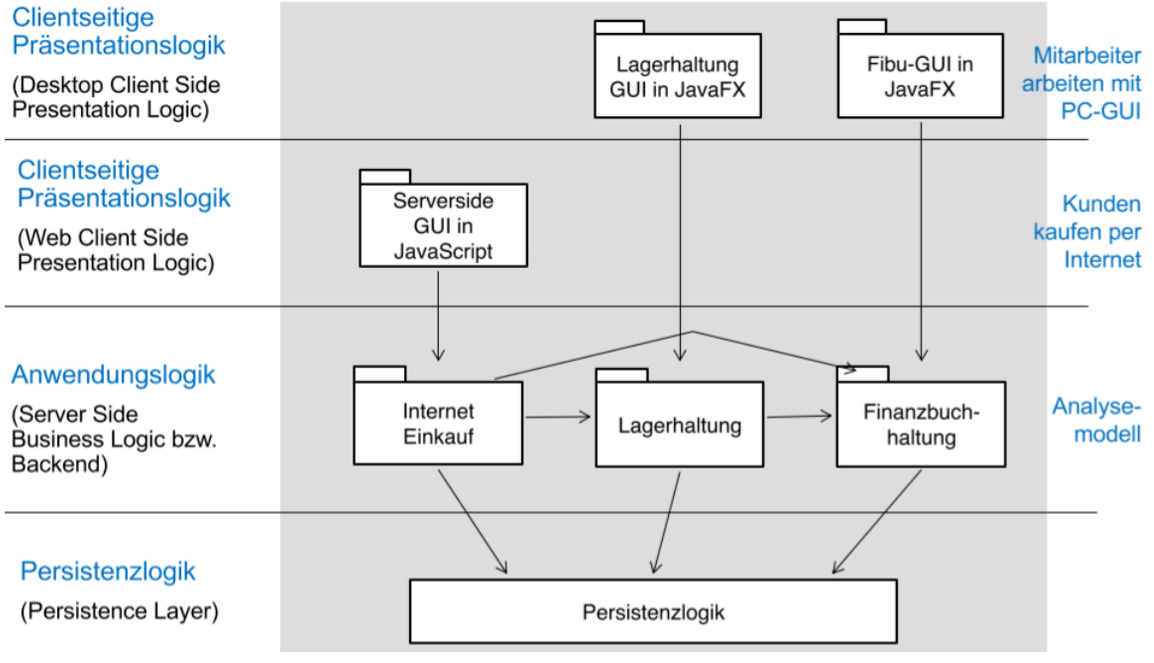


physikalisches Rechnernetz
4 verknüpfte Rechner

logische Struktur

- Clientseitig: Software liegt beim Client
- Anwendungslogik sind die in der Analyse erstellen Entity-, Control- und Boundary-Klassen

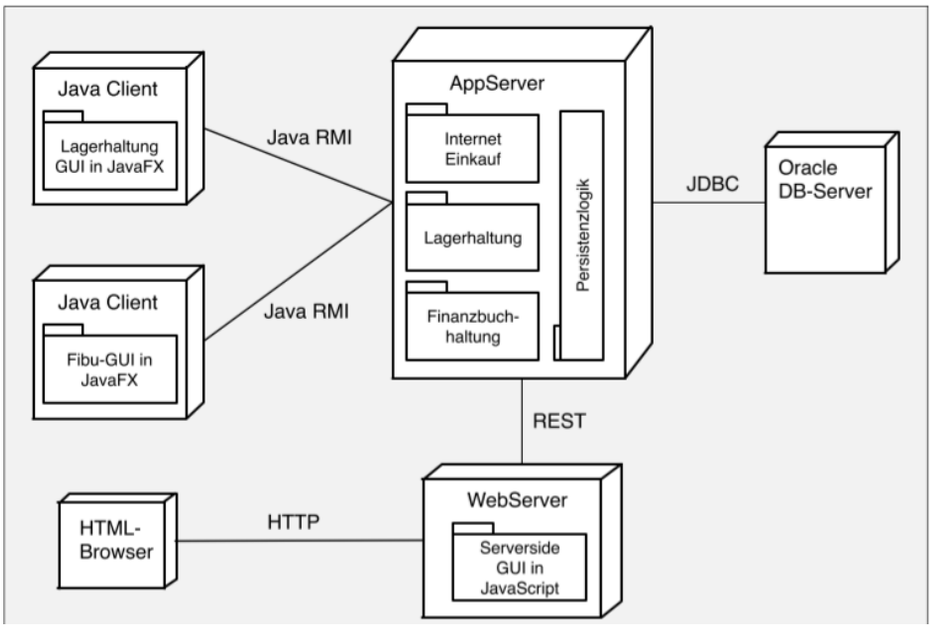
2. Subsysteme – logische Struktur (Bsp. Internet-Buchhandlung Pakete aus Folie 122)



- Die Boxen sind Subsysteme und innerhalb der Subsysteme Komponenten, die die Zusammenarbeit zwischen den Subsystemen ermöglicht

2. Subsysteme – physikalische Struktur (Bsp. Internet-Buchhandlung)

Verteilung der logischen Komponenten auf physikalische Struktur



5.3 Design – Vorgehen

5.3.1 Schritte der Design-Phase

Schritte der Design-Phase (I)

1. Modelliere **physikalische Struktur** durch **Deployment-Diagramm**
   - (virtuelle) Hardware und Netzwerkkonfiguration festlegen
   - Auswahl von Middleware (z.B. Application Server), Kommunikations-technologie (z.B. REST, MOM), GUI-Konzept und Systemsoftware (z.B. Betriebssystem, RDBMS)

2. logische (fachliche u. technische) **Subsysteme/ Komponenten** identifizieren
   - in ihren **Verantwortlichkeiten** und **Schnittstellen** beschreiben
   - Standard-Komponenten festlegen (Entscheidung „make or buy“)
   - komplexe Subsysteme in weitere Subsysteme bzw. Komponenten unterteilen (schrittweise Verfeinerung)

3. bei Bedarf **technische Komponenten** weiter verfeinern
   - Erweiterung der Klassendiagramme um **technische Klassen**
   - Durchspielen technischer Abläufe mittels Sequenzdiagrammen
   - bei komplexem dynamischen Verhalten: Zustands- oder Aktivitätsdiagramme erstellen

4. Entwurf der zentralen (Querschnitts-)Dienste

⇒ Vorgehen entspricht **schrittweiser Verfeinerung (top down)**

physikalische/ logische Architektur
→ Subsysteme
→ Komponenten
→ Klassen
→ Datenstrukturen/Algorithmen