5.5 Architekturmuster

- Entwurfsmuster auf der Ebene von Klassem, also konkreter als Architekturmuster
- generelles Ziel: Teile bestehender Lösungen wiederverwenden: sowohl bewährte Entwurfsmodelle als auch implementierte Softwarekomponenten
- etablierte Muster existieren für unterschiedliche Abstraktionsebenen

1. **Architekturmuster** (architecture pattern, Basisarchitekturen, Architekturstile)
   - beschreiben Systemstrukturen, die die Gesamtarchitektur eines Systems festlegen
   - spezifizieren, wie Subsysteme zusammenarbeiten

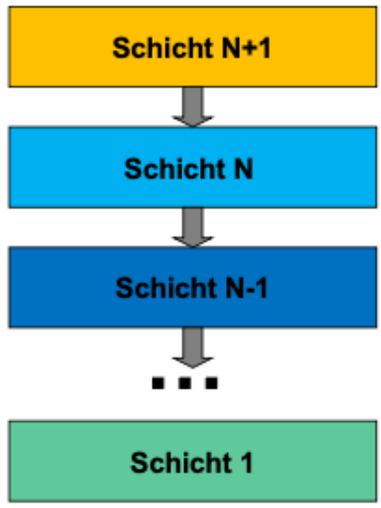
2. **Entwurfsmuster** (design patterns) (⇒ Kap. 5.6)
   - stellen bewährte generische Lösungen für oft wiederkehrende Entwurfsprobleme dar, die in bestimmten Situationen auftreten
   - sind häufig Teil eines Architekturmusters

5.5.1 Logische Softwareschichten

Softwareschichten (Layers)

- Komplexität verbergen erreicht man durch Anwendung von Entwurfsprinzipien wie Modularisierung, Kapselung und hohe Kohäsion
- **Architekturmuster** zur Strukturierung eines (Anwendungs-)**Software-Systems**
- **Schicht (layer)** fasst **logisch** zusammengehörige Komponenten zusammen

- **Verantwortung der Schicht *N*:**
  - eine Schicht *N* stellt Dienste (services) zur Verfügung, die **nur** von der darüber liegenden Schicht *N+1* genutzt werden können
  - eine Schicht *N* nutzt **ausschließlich** Dienste der darunter liegenden Schicht *N-1*
  - Schicht *N* kennt Schicht *N+1* **nicht**



- Schichten bilden **logische Struktur** des Software-Systems
- **Ziel: Komplexität in Schichten verbergen**

Eigenschaften der logischen Schichtentrennung

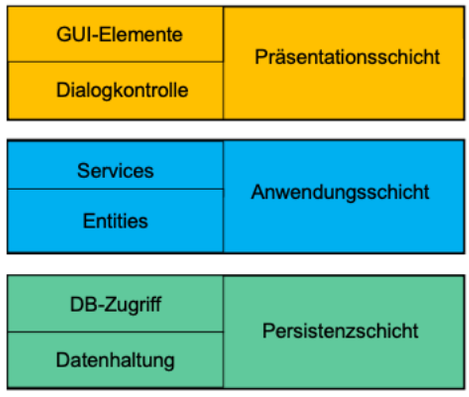
- **lose Kopplung:** nur benachbarte Schichten kennen sich
- robust gegen über Änderungen:
  - Änderungen wirken sich nur lokal aus (**starke Kohäsion**)
  - Änderungen schlagen sich nur auf die darüberliegende Schicht durch

- **Schichten werden für physikalische Verteilung genutzt**

3-Schichtenarchitektur

- Software-System besteht i.d.R. aus 3 logischen Schichten
- **drei logische Schichten getrennt implementiert**
- Die **Standardarchitektur** für logische Softwarestrukturierung

1. **Präsentationsschicht** (bzw. GUI-Schicht)
2. **Anwendungsschicht** (bzw. Applikationsschicht)
3. **Persistenzschicht** (bzw. Zugriffsschicht)



Vorteile:

- **hohe Kohäsion**
  - klare Verteilung der Verantwortlichkeiten auf unterschiedliche Schichten (separation of concerns)
- **Redundanzfreiheit**
  - fachliche Plausibilitäten und Funktionen nur in fachlichen Objekten
  - DB-Zugriffscode nur in Persistenzschicht
  - GUI-Code nur in Präsentationsschicht
- **gute Änderbarkeit/ Wartbarkeit**
  - durch lose Kopplung schlagen Änderungen nicht auf andere Schichten
  - durch (insbesondere Änderungen der GUI und des Datenbank-Schemas)

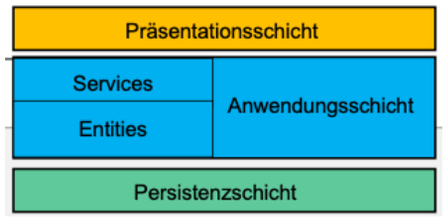
Nachteile

- Softwarearchitektur wird komplexer

A. Anwendungsschicht

(1) Entities (Entitätsklassen)
- Enthalten **fachliche** Daten (→ zentrale fachliche Objekte)
- Beziehungen zu anderen Objekten

(2) Services (Control-Klassen)
- Realisieren **fachliche** Abläufe (→ Geschäftslogik)



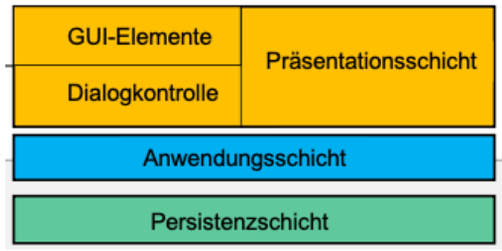
(3) Schnittstellen
- Bietet der Präsentationsschicht Schnittstellenklasse/n (Fassadenklasse/n)
- Nutzt Persistenz-Framework (Middleware) zum Speichern/Laden von Objekten in/ aus DB

Anwendungsschicht kennt weder GUI-Fenster noch DB-Tabellen!

B. Präsentationsschicht

(1) Benutzungsoberfläche (GUI = graphical user interface)
- Präsentation fachlicher Datenobjekte mittels GUI-Elemente (Textfelder, Radiobuttons, ...)
- Interaktion mit Benutzer

(2) Dialogkontrolle
- sendet Daten an die Anwendungsschicht (bzw. ruft fachliche Dienste auf) (ausgelöst durch Benutzerereignis)
- empfängt Daten aus der Anwendungsschicht und bereitet sie auf



(3) Schnittstellen
- Präsentationsschicht weiß möglichst wenig über die Anwendungsschicht
  - kennt Anwendungsschnittstelle (Fassadeklasse(n) → später)
  - nutzt Datenobjekte zum Datenaustausch

C. Persistenzschicht

(1) DB-Zugriff
- (SQL-)Code für den Zugriff auf die Datenbank (Suchen, Speichern von Objekten, Transaktionen, ...)
- insbesondere komplexe Suchanfragen zum Auffinden von Objektmengen

(2) Datenhaltung
- persistente Verwaltung fachlicher Daten in einem Datenbanksystem (RDBMS, NoSQL-Datenbank, Dateisystem, ...)
- liegt **außerhalb** der eigentlichen Persistenzschicht (nämlich im Datenbanksystem)

(3) Schnittstellen
- bietet der Anwendungsschicht API zum Laden und Speichern von Objekten
- Persistenzschicht kennt nur wenig Strukturen der Anwendungsschicht (meist die Geschäftsobjekte/ Entity-Klassen)
- Persistenzschicht kennt das DB-Schema

- Aufgabe: fachliche Objekte der Anwendungsschicht in einem Datenbanksystem verwalten
- Anwendungsschicht kennt Datenbankschema nicht

5.5.2 Physikalische Softwareschichten

Physikalische Verteilung der Schichten

Schichten werden auf verschiedene Rechnersysteme/-knoten **verteilt**

- Zweischichten (2-Tier) -Architektur
- Dreischichten (3-Tier) -Architektur
- Mehrschichten (N-Tier) -Architektur

Motivation für physikalische Verteilung

a.) Mehrbenutzerfähigkeit:

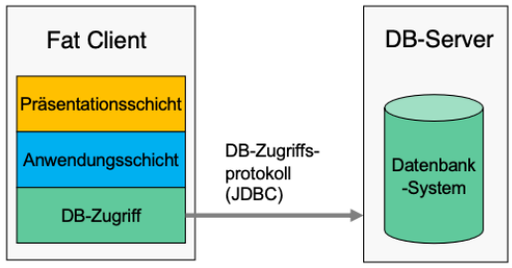
- mehrere Benutzer*innen sollen gleichzeitig die selben Daten verwenden oder die selbe Funktionalität benutzen können

b.) Skalierbarkeit:

- steigende Leistungsanforderungen (Zahl der Benutzer, Zahl der Dienstaufrufe, größere Datenmengen) an Softwaresystem müssen erfüllt werden, ohne das System zu modifizieren

2-Tier-Architektur (Fat Client)

- **Fat-Client**: enthält alle 3 logischen Schichten der Anwendung (entspricht Remote Database auf Folie 189)
- **Datenbank-Server**: verwaltet die Daten zentral
- **Protokoll**: DB-Zugriff (über JDBC)



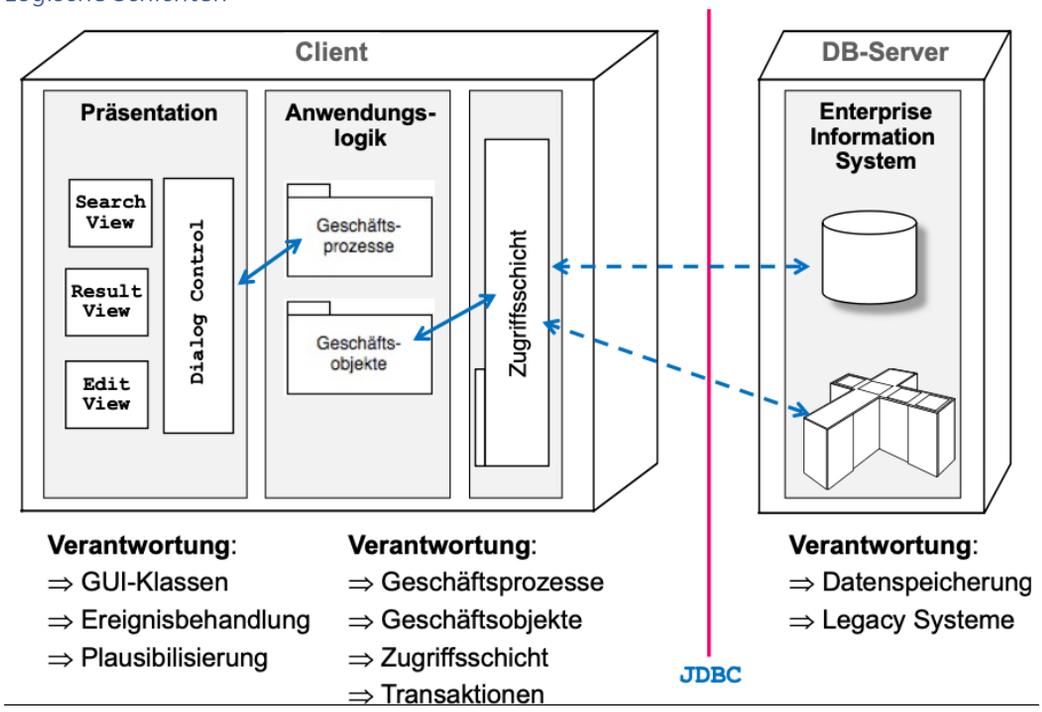
Vorteile

- einfache Architektur
- nur ein Kommunikationsmechanismus (z.B. JDBC)

Nachteile

- keine Skalierbarkeit, d.h. Anpassung an erhöhte Leistungsanforderungen
- Client erfordert hohe Rechenleistung
- Software-Verteilung bei Änderungen

Logische Schichten



Verantwortung:
⇒ GUI-Klassen
⇒ Ereignisbehandlung
⇒ Plausibilisierung

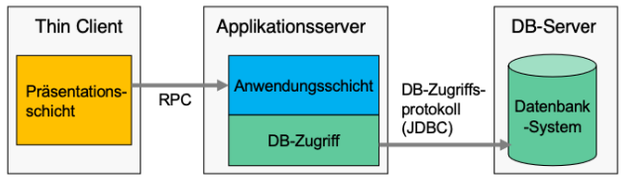
Verantwortung:
⇒ Geschäftsprozesse
⇒ Geschäftsobjekte
⇒ Zugriffsschicht
⇒ Transaktionen

Verantwortung:
⇒ Datenspeicherung
⇒ Legacy Systeme

3-Tier-Architektur

- Hier muss der Client auch Software installieren und bei Änderungen dementsprechend angepasst verteilt werden

- **Thin Client**: enthält nur Präsentationsschicht (z.B. JavaFX)
- **Applikationsserver**: enthält Geschäftslogik und DB-Zugriffsschicht
- **Datenbank-Server**: verwaltet die Daten
- **Protokolle**: DB-Zugriff (über JDBC) und



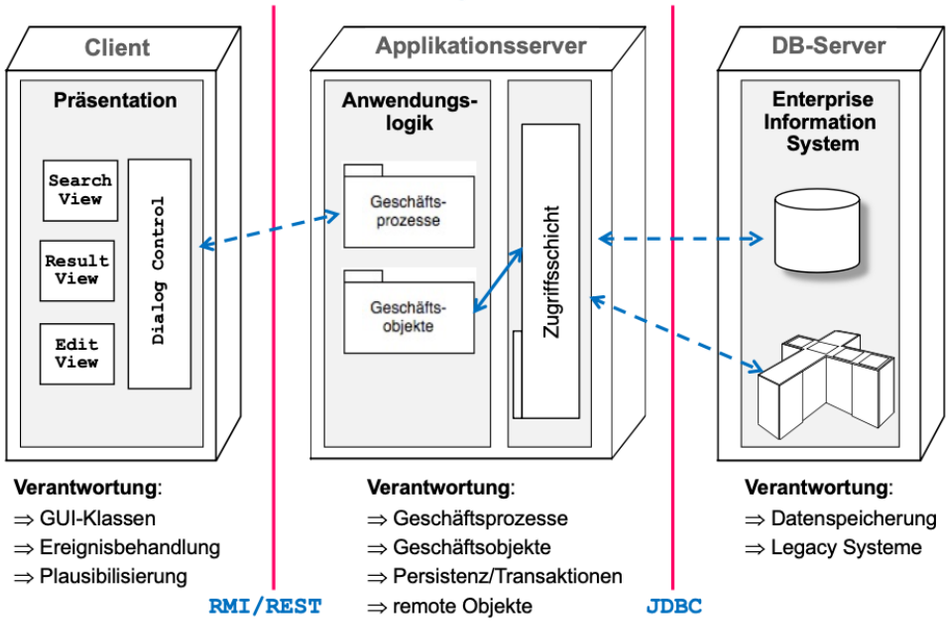
Vorteile

- gute Skalierbarkeit
- wenig Rechenleistung für Client erforderlich (ggf. Smartphones)

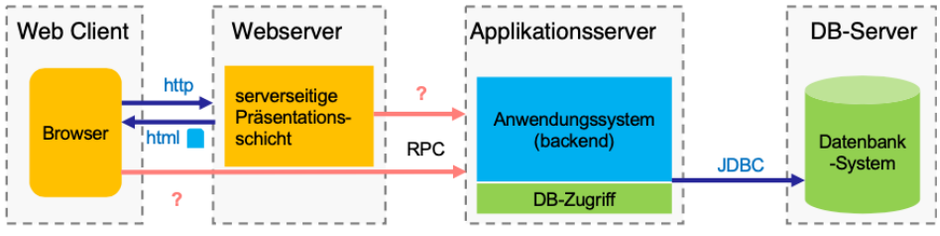
Nachteile

- komplexe Architektur

3-Tier-Architektur: Client/ Server-System



N-Tier-Architektur: Websysteme



- Websysteme besitzen eine **HTML-basierte Präsentationsschicht**
  - **Web-Client**: Browser zeigt HTML-Webseiten
  - **Webserver**: stellt HTML-Webseiten zentral (serverseitig) zur Verfügung
- **Applikationsserver:** enthält Geschäftslogik und DB-Zugriffsschicht
- **Datenbank-Server:** verwaltet die Daten
- **Protokolle:** DB-Zugriff (über JDBC);
  Zugriff auf die Anwendungsschicht (über RPC);
  Kommunikation Web-Client mit Webserver (über HTTP)

Vorteile von Websystemen:

- Client ist plattformunabhängig
- einfache Softwareverteilung (keine Installation auf Client notwendig)
- gute Skalierbarkeit

Nachteile von Websystemen:

- Webseiten können weniger als Desktops:
  - werden in Sandbox ausgeführt
  - kein bzw. eingeschränkter Zugriff auf Hardware (Sensoren, Speicher,..)