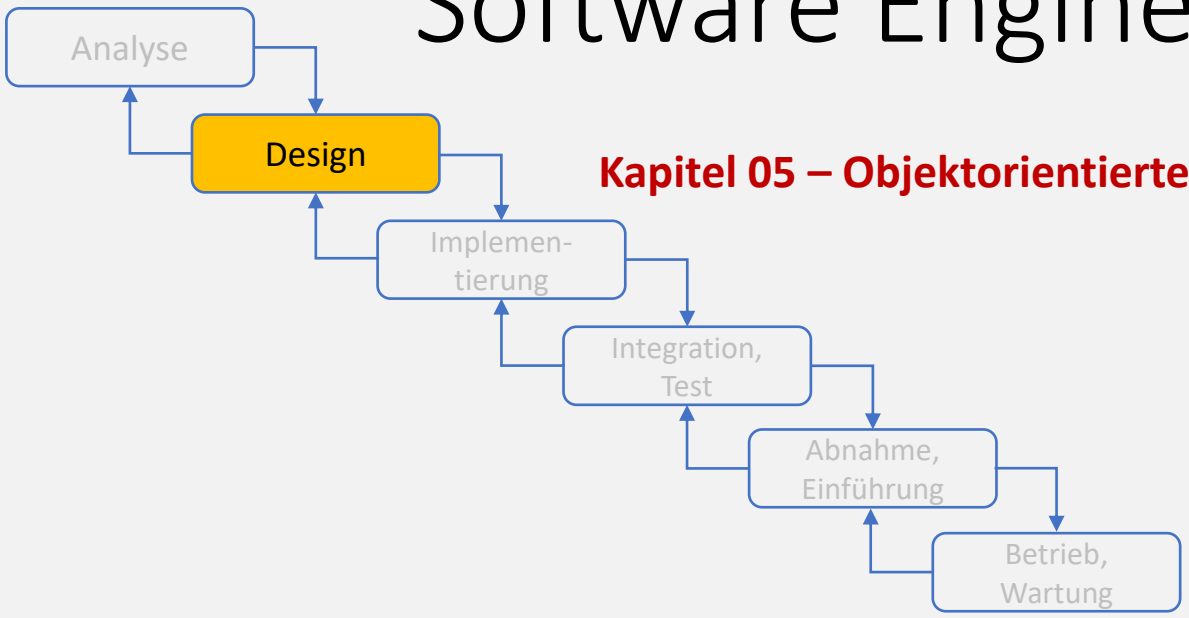
Software Engineering



Kapitel 05 – Objektorientiertes Design (OOD)

Architekturmuster

OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG
INFORMATIK UND MATHEMATIK

Gliederung

1. Motivation
2. Softwarearchitektur
3. Architektursichten
   - nach Kruchten
   - nach Starke
4. Architekturprinzipien (Kopplung, Kohäsion, DRY, OCP, …)
5. Architekturmuster (Schichten, Pipes&Filters, Blackboard, …)

Architekturmuster

Architekturmuster: Motivation

Bewährte Architekturmuster:

- Helfen bei der Strukturierung von Systemen und Anwendungen
- Dienen als Anhaltspunkt beim Systementwurf
- Müssen während des Designs weiter verfeinert werden

Beispiele:

- Schichtenarchitektur
- Pipes und Filter
- Blackboard, …

} **Chaos zu Struktur**

- Peer-to-Peer
- Client-Server, …

} **Verteilte Systeme**

- Model View Controller (MVC)

…

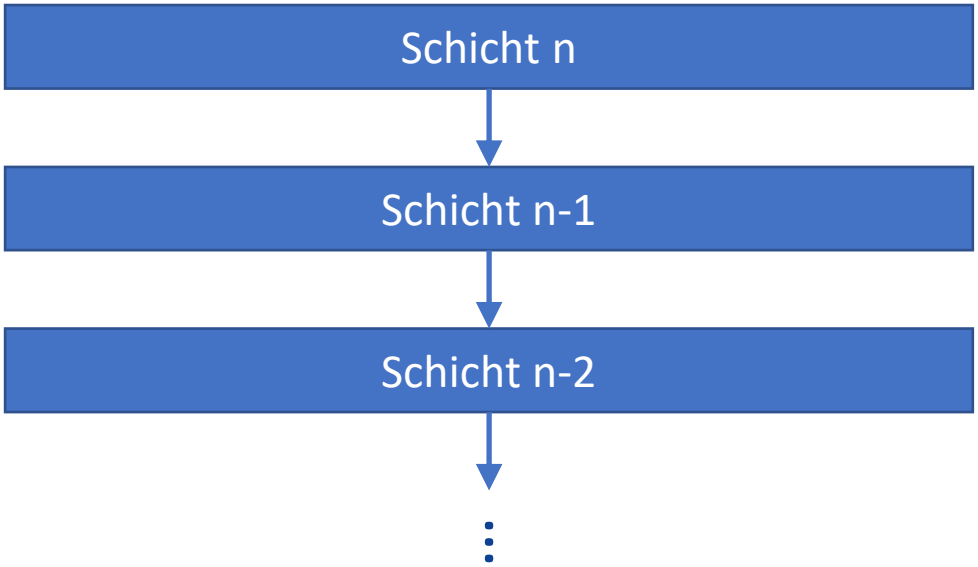
} **Interaktive Systeme**

Architekturmuster: Schichtenarchitektur (1)

Idee:

Beispiele?

- System in mehrere Schichten aufteilen
- Jede Schicht fasst logisch zusammengehörige Komponenten zusammen
- Jede Schicht stellt Dienstleistungen über Schnittstellen zur Verfügung
- Jede Schicht darf nur auf jeweilige Vorgängerschicht zugreifen (strikte Architektur)



Architekturmuster: Schichtenarchitektur (2)

Eigenschaften:

- Schichten sind nur gekoppelt, wenn sie benachbart sind
- Koppelung auf Schnittstellenebene (vertretbare Kopplung)
- Änderungen wirken sich meist nur lokal aus
- Eine Schicht kann aus mehreren entkoppelten Teilen mit intern hohem Zusammenhalt (hohe Kohäsion) bestehen

Unterscheidung von Schichtenarchitekturen:

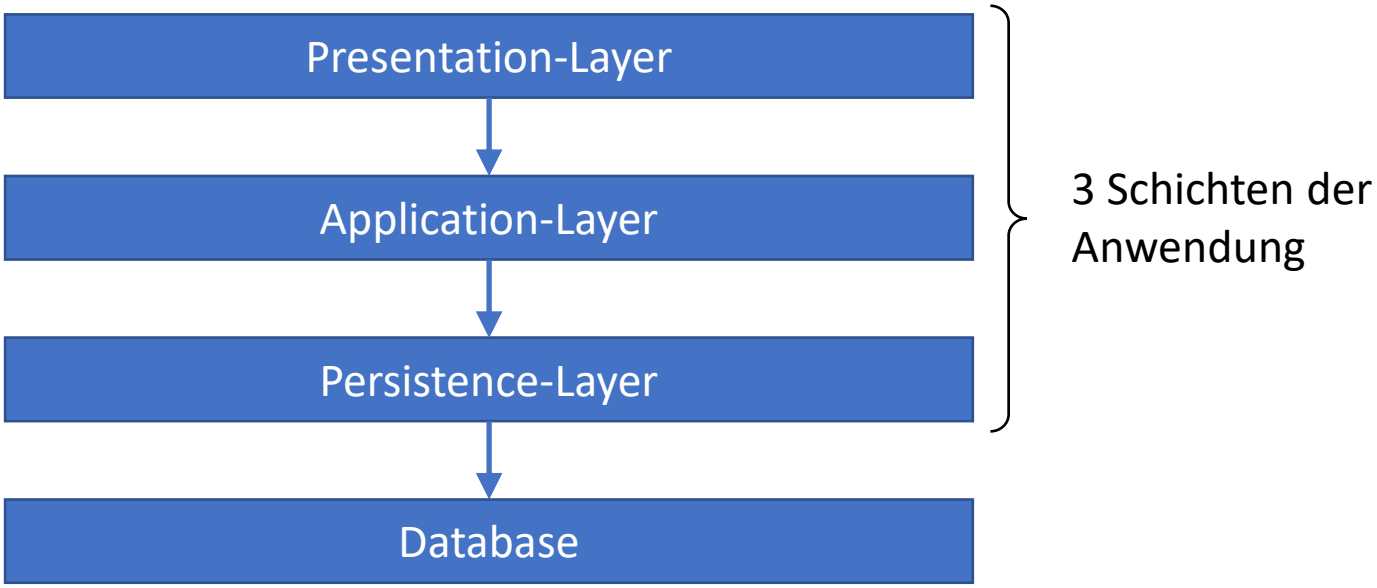
- Strikte Schichtenarchitektur/Protokollbasierte Schichten:
  - Zugriff ausschließlich auf die nächst niedrigere Schicht
- Objektorientierte Schicht:
  - Zugriff auf alle tieferen Schichten

Welche Architekturprinzipien treffen auf eine gute Schichtenarchitektur zu?

Architekturmuster: Schichtenarchitektur (3)

Die klassische 3-Schichten-Architektur:

- Häufig eingesetzt für interaktive Systeme
- Protokollbasierte Schichtung



Architekturmuster: Schichtenarchitektur (4)

Die klassische 3-Schichten-Architektur:

Presentation-Layer

- Realisiert die Bedienoberfläche
- Stellt Informationen dar
- Steuert die Interaktion mit dem Benutzer.
- Greift auf die Application-Layer zur Erfüllung der Aufgaben zu.

Austausch der Presentation-Layer bedeutet:

- Mehrere (verschiedene) Oberflächen können gleiche Applikationslogik benutzen
- Wiederverwendung auf Komponentenebene

Architekturmuster: Schichtenarchitektur (5)

Die klassische 3-Schichten-Architektur:

Application-Layer

- Realisiert die fachliche Funktion der Anwendung
- Kennt keine Information über den Presentation-Layer.
- Verwaltet alle Objekte und Klassen des Domänen-/Produktmodells
- Greift auf die Dienste der Persistence-Layer zu
- Kennt keine technischen Details der Persistence-Layers
- Datenaustausch erfolgt im Modell der Anwendung (Domainmodell)

Architekturmuster: Schichtenarchitektur (6)

Die klassische 3-Schichten-Architektur:

Persistence-Layer

- Abstrahiert die Art des Speichermediums von der Applikation
- Speichert die Objekte des Domänenmodells persistent ab (z.B. Festplatte, DB, …)
- Übersetzt die Objekte des Domänenmodells in ein Format welches für die Ablage notwendig ist. (Objekte → Relationen)

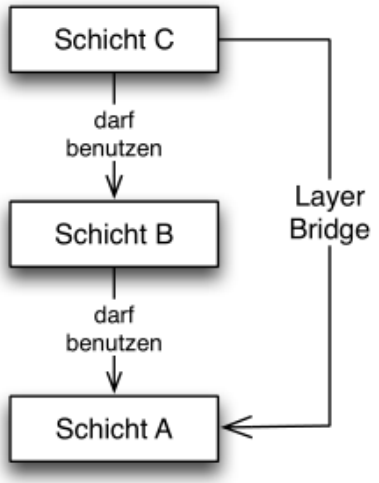
Architekturmuster: Schichtenarchitektur (7)

Vorteile von Schichtenarchitekturen:

- Schichten voneinander unabhängig (sowohl bei Erstellung als auch im Betrieb)
- Schichten über verschiedene Rechnerknoten verteilbar
- Implementierung austauschbar (sofern die gleichen Dienste angeboten werden)
- Schichtenbildung minimiert Abhängigkeiten zwischen Komponenten
- Leicht verständliches Strukturkonzept

Nachteile von Schichtenarchitekturen:

- Kann Performance beeinträchtigen
  (z.B. bei Anfragen durch mehrere Schichten)
- Schicht-übergreifende Änderungen
  werden schlecht unterstützt
  (z.B. neues Datenfeld, das sowohl gespeichert als auch in Nutzeroberfläche angezeigt werden soll zieht Änderungen in allen Schichten mit sich)



Architekturmuster: Schichtenarchitektur (8)

Hinweise:

- Vermeide Aspekte der Fachdomäne in die Präsentationsschicht zu verlagern:
  - Resultat: schwer wartbare Systeme
  - Reduziert Möglichkeit der Wiederverwendung innerhalb der Fachdomäne
- Wenn viele externe Ressourcen (Fremdsysteme) integriert werden:
  - Aufteilung der Infrastrukturschicht in Integrationsschicht und Ressourcenschicht

Beispiel: OSI-Modell (7 Schichten) Referenzmodell für Netzwerkprotokolle

Schicht	Aufgabe
Anwendungsschicht	- Schnittstelle zwischen Rechner und Anwendungsprogrammen
Darstellungsschicht	- Methoden zur Datei-Ein- und -Ausgabe; - Anzeige von Anweisungen und Fehlermeldungen; - Festlegung von Übertragungskonventionen und Bildschirmdarstellungen
Sitzungsschicht	- Verbindungsaufbau, Abfangen und Auswertung von Fehlern bei der Übertragung - Verbindungssynchronisation - Wiederaufbau von Sitzungen nach einem Ausfall in den unteren vier Ebenen
Transportschicht	- Flusskontrolle; - Aufteilung und Zusammenführung des Datenstroms in Datenpakete; - Fehlerkontrolle; - Kontrolle von Datenverlust.
Vermittlungsschicht	- Adressierung der Datenpakete und Routing durch das Netz
Sicherungsschicht	- Fehlererkennung, Synchronisation der Datenübertragung; - Realisierung der verschiedenen Zugriffsverfahren.
Physikalische Schicht	- Definition und Spezifikation für das Übertragungsmedium.

Architekturmuster: Pipes und Filter (1)

Idee:

Beispiele?

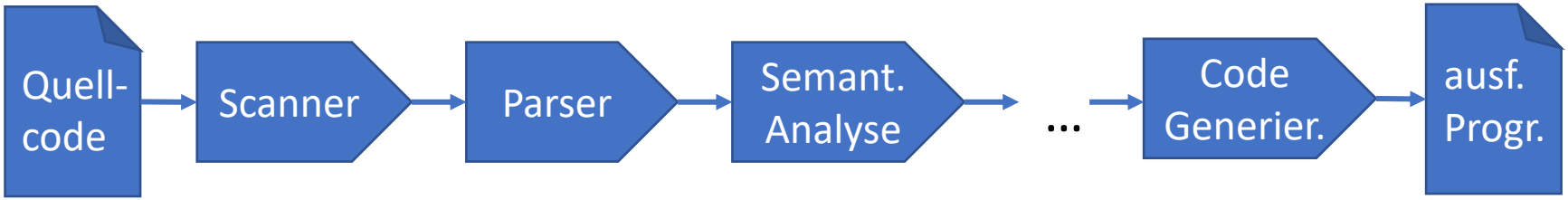
- Struktur für Systeme, die Datenstrom verarbeiten
- Verarbeitungsschritte in Filter gekapselt (Vorteil: getrennt entwickelbar)
- Daten werden von einem Subsystem zum nächsten transportiert (pipe)
- Jedes Subsystem transformiert (filtert) die Daten

Herkunft:

- Pipes in der Unix-Shell (einfache Aufgaben zu komplexeren verknüpfen)

Beispiel:

- Compiler als Pipe:



Architekturmuster: Pipes und Filter (2)

Arten von Filtern:

- Aktiv: Eigenständig laufende Prozesse oder Threads (von übergeordnetem Programm aufgerufen, ab dann laufen sie selbständig)
- Passiv: Durch Aufruf eines benachbarten Filterelementes aktiviert
  - Push-Mechanismus: Filter wird aktiv, wenn ihm Daten zugeschoben werden
  - Pull-Mechanismus: Filter wird aktiv, wenn nachfolgender Filter Daten anfordert

Zusammenarbeit zwischen Pipes und Filtern:

- Filter stellt Verarbeitung vollständig fertig, übergibt aktiv das Ergebnis an Pipe, die es zum nachfolgenden Filter transportiert
- Pipe erfragt das nächste Ergebnis bei Eingangsfilter und übergibt es an Ausgangsfilter
- Zentrale Steuerung koordiniert das Zusammenwirken der Filter
- Filter übergeben Ergebnisse “stückchenweise” über Pipes an nachfolgenden Filter; viele Pipes und Filter sind gleichzeitig aktiv (parallele Verarbeitung)

Architekturmuster: Pipes und Filter (3)

Pipes entkoppeln Filter:

- Pipes können direkt oder zeitversetzt Daten transportieren
- Pipes können entscheiden, an welche Instanz eines Filters sie die aktuellen Daten weiterreichen
- Pipes können kapseln, welche Filter als nächstes in Verarbeitungskette folgen

Fazit:

- Pipes dienen als flexible Puffer zwischen Filtern
- Pipes sollen Kopplung im System verringern

Architekturmuster: Pipes und Filter (4) Diskussion

Vorteile:

- Leicht verständliches Muster
  (→ wir sind mit solchen Abfolgen von Arbeitsschritten vertraut)
- Existierender Filter leicht durch neue Komponente austauschbar
- Einfache, definierte Schnittstellen zwischen Subsystemen
- Filter evtl. zu komplexeren Verarbeitungseinheiten kombinierbar

Nachteile:

- Fehlerbehandlung: Filter kennen sich nicht gegenseitig, Folgefehler schwer behebbar
- Konfiguration: über zentrale Steuerung oder Intelligenz in die Filter verlagern
- Zustand: Filter kennen keinen globalen Zustand; Verarbeitungsinformationen müssen also in Daten übertragen oder zentraler Steuerung mitgegeben werden

Architekturmuster: Pipes und Filter (5) Beispiel

- Pipes in Unix: “|” beispielhaft mit:
  - cat: Liest Strings von Standardeingabe
  - sort: Sortiert alphabetisch
  - uniq: Entfernt Duplikate
  - nl: Fügt Zeilennummern hinzu

Beispiele?

Input:

```
cat | sort| uniq | nl
Carsten
Thomas
Andreas
Tim
Bernd
Jakob
Thomas
Tessa
Wolfgang
Michael
Andreas
```

Ergebnis:

```
1 Andreas
2 Bernd
3 Carsten
4 Jakob
5 Michael
6 Tessa
7 Thomas
8 Tim
9 Wolfgang
```

Architekturmuster: Blackboard (1)

Idee:

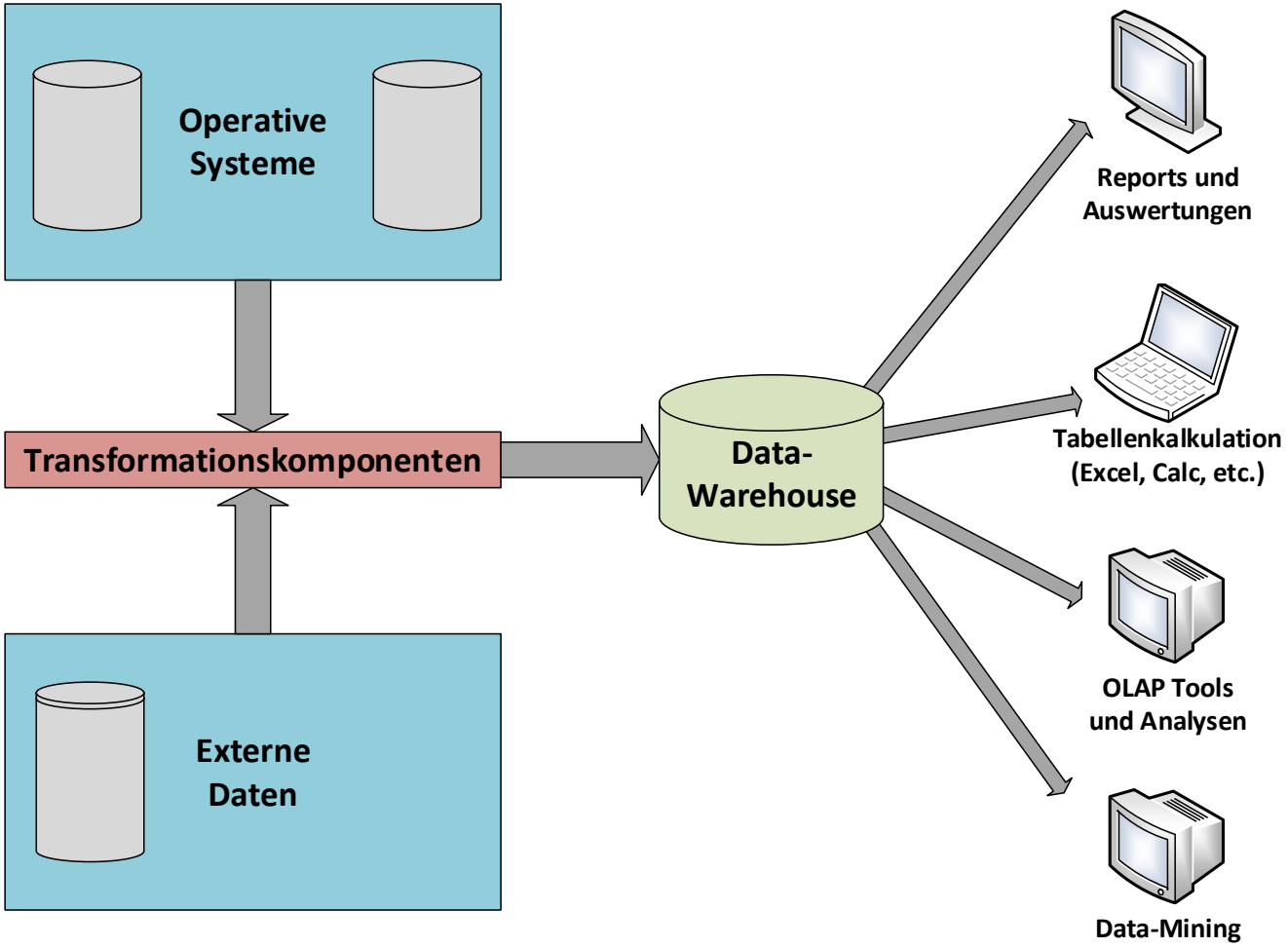
Beispiele?

- System ist Ansammlung unabhängiger Subsysteme
- Jedes Subsystem ist auf Teilaufgabe spezialisiert (Wissensquelle)
- Alle Subsysteme greifen auf ein gemeinsames Blackboard (Speicher/Datenbank/Repository) zu
- Datenaustausch untereinander über das Blackboard
- Zentrale Komponente bewertet Zustand und koordiniert Subsysteme

Beispiele:

- KI-Anwendungen (z.B. Bild- und Spracherkennung)
- Integrierte Entwicklungsumgebungen
- Management-Informationssysteme, Data Warehousing

Architekturmuster: Blackboard (2)
Beispiel: Data-Warehouse



Architekturmuster: Blackboard (3) Diskussion

Vorteile:

- Effiziente Methode für gemeinsame Datennutzung (ohne direkte Kommunikation)
- Datengenerierende Subsysteme brauchen keinerlei Rücksicht darauf zu nehmen, wie Daten von anderen Subsystemen verarbeitet werden (→ schwache Kopplung)
- Zentralisierung von Aufgaben (Datensicherung, Schutz von Daten, Zugriffskontrolle) (→ hohe Kohäsion)

Nachteile:

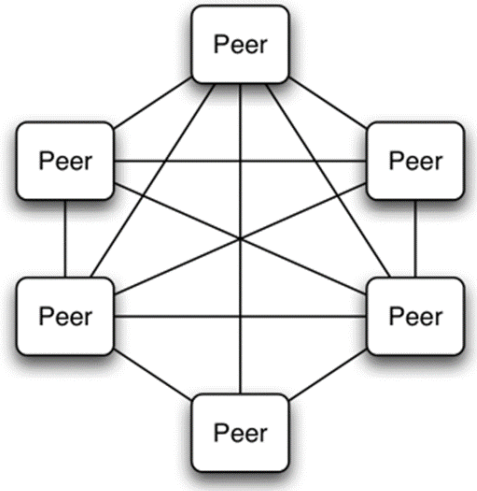
- Subsysteme müssen dasselbe Datenformat nutzen
- Integrieren neuer Subsysteme kann so schwierig werden
- Blackboard-Komponente als Flaschenhals

Architekturmuster: Peer-to-Peer (1)

Idee:

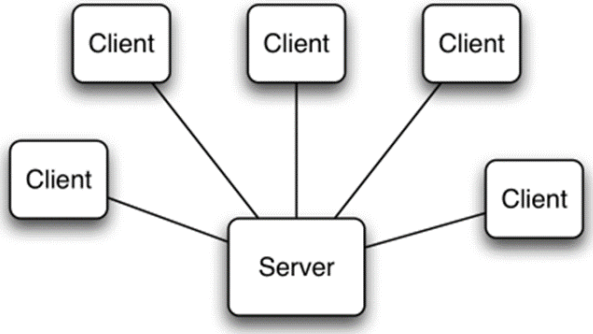
Beispiele?

- Gleichberechtigte Komponenten (Peers), die über Netzwerk verbunden sind
- Komponenten nehmen gleichzeitig Rolle von Server und Client wahr
- Teilen Ressourcen (CPU, Speicher, Dateien, etc.)



Peer-to-Peer-Architektur

vs



Client-Server-Architektur

Architekturmuster: Peer-to-Peer (2) Diskussion

Zweck: Ausfallsichere Datenverteilung:

- Filesharing
- Digitale Telefonie, Instant Messanging
- Verteiltes Rechnen: komplexe Rechenaufgaben (Spektralanalyse, Primfaktorzerlegung) auf Peers verteilt (Beispiel: Seti@Home, Folding@Home)

Vorteile:

- Hohe Ausfallsicherheit und Parallelität
- Kein Flaschenhals

Nachteile:

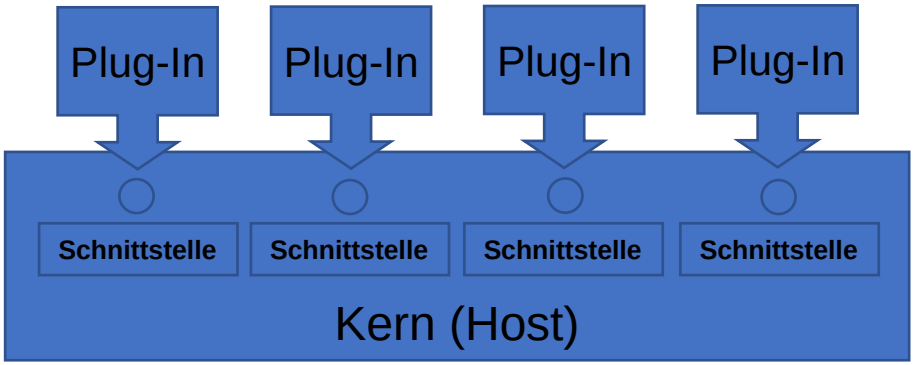
- Auffinden von Peers in großen Netzen schwierig
- Fehlerbehandlung schwierig

Architekturmuster: Plug In

Idee:

Beispiele?

- System dynamisch um neue Funktionen erweitern, ohne Kernsystem zu modifizieren
- System bietet Plug-In-Mechanismus an, über den externe Module hinzugefügt werden können
- System bietet Schnittstellen, die durch die Plug-Ins implementiert werden

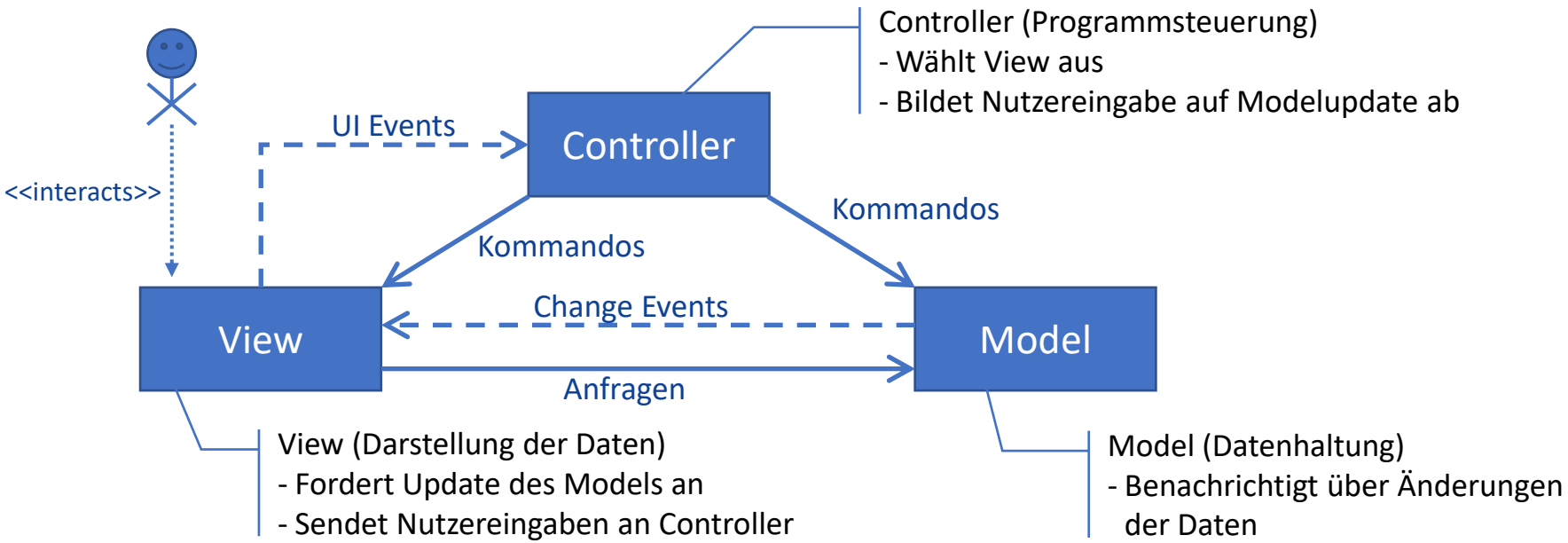


Architekturmuster: Model View Controller (MVC)

Idee:

Beispiele?

- Aufteilung eines Programms in Komponenten:
  - Model (Darzustellende Daten, evtl. auch Geschäftslogik)
  - View (Präsentation der Daten, Entgegennahme vom Benutzerinteraktionen)
  - Controller (Steuerung)



Architekturmuster: Thin vs. Fat Client

- **Thin Client:**
  - Nur die Benutzerschnittstelle ist auf dem Client vorhanden

Beispiele?

- **Fat Client:**
  - Teile oder die gesamte Fachlogik auf dem Client vorhanden
  - Hauptfunktion des Servers: Datenhaltung
  - Dadurch Entlastung des Servers
  - Aber zusätzliche Anforderungen an die Clients

Literatur

- Effektive Software-Architekturen, G. Starke
- The 4+1 view model of architecture, P. Kruchten, IEEE Software, November 1995, 12(6), pp. 42-50 [PK95]
- Software Engineering, I. Sommerville, Pearson, 2012