University of Basel

Software Engineering - Design

M. Lüthi, Universität Basel, 11. August 2022

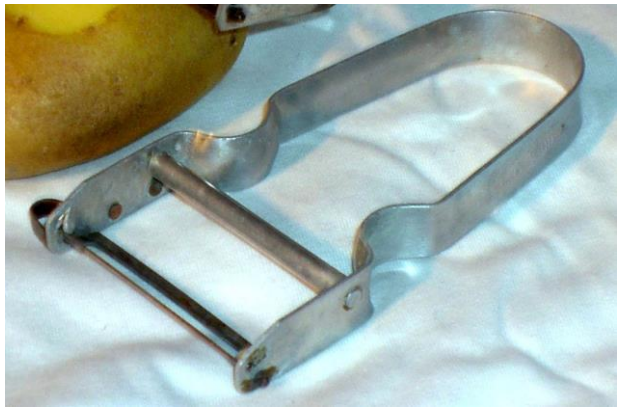
Agenda 11. August 2022

1	Softwaredesign und Software Architektur (45’)
2	Objektorientiertes design (30’)
3	Übung 4: Unittests und Continuous Integration (80’)

Was ist design

Formgebende und funktionale Gestaltgebung eines Produkts.

- Strukturiert ein Artefakt
- Zerlegen eines Systems in (einfachere) Komponenten
  - Zuweisen von Verantwortlichkeiten
  - Sicherstellen, dass Gesamtsystem Anforderungen erfüllt



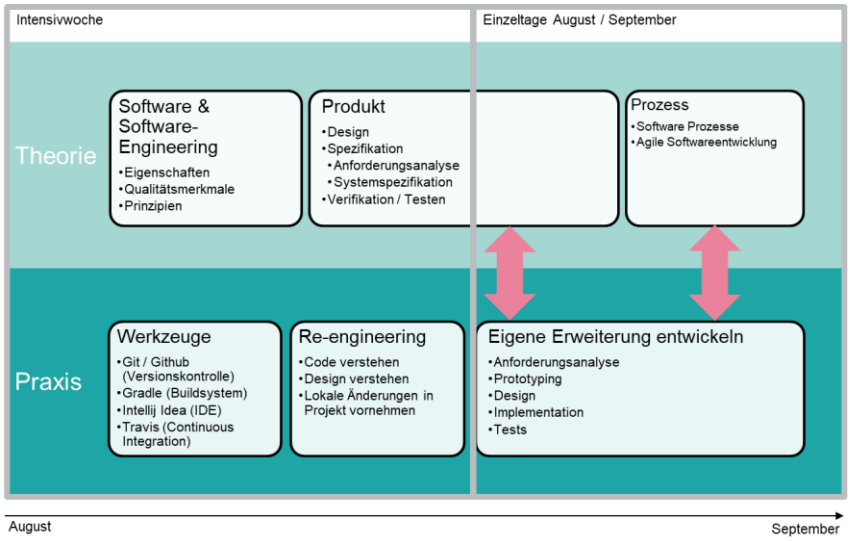
Ästhetik ist nicht die primäre Aufgabe!

Designbeispiel 1: Entwurf einer Vorlesung

Teile

- Vorlesungen
- Übungen
- Selbstlernphase
- Leistungsnachweise
- ...

Auch die Interaktion der Teile muss designed werden



Designbeispiel 2: Enwurf eines Einkaufszentrums

Teile

- Verkaufsfläche
- Laufwege
- Heizungs- und Lüftungssystem
- Verkabelung
- Parkplatz
- ...

Design im Software Engineering

Zwei Bedeutungen

1. Schritt zwischen Anforderungsanalyse und Implementation
   - Erstellen der Softwarearchitektur / Design

2. Strukturierung eines Artefakts
   - Design einer Klasse in einem OO-System
   - Design des Anforderungsdokuments

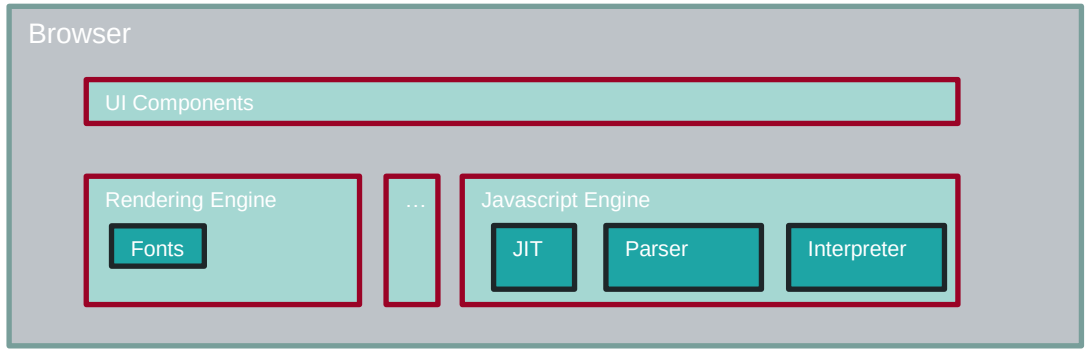
Softwaredesign

- Struktur und Entwurf der Module

Architektur vs. Design

Softwarearchitektur: Struktur der Module (Design des gesamten Systems)
Software oder Moduldesign: Entwurf individueller Module (Design der Teile)

Module enthalten selbst wieder Module ⇒ Keine strikte Trennung möglich



Ziel eines Designs

Struktur so festzulegen, dass hohe Software-qualität erreicht werden kann.

Wichtigstes Prinizip: Design for Change

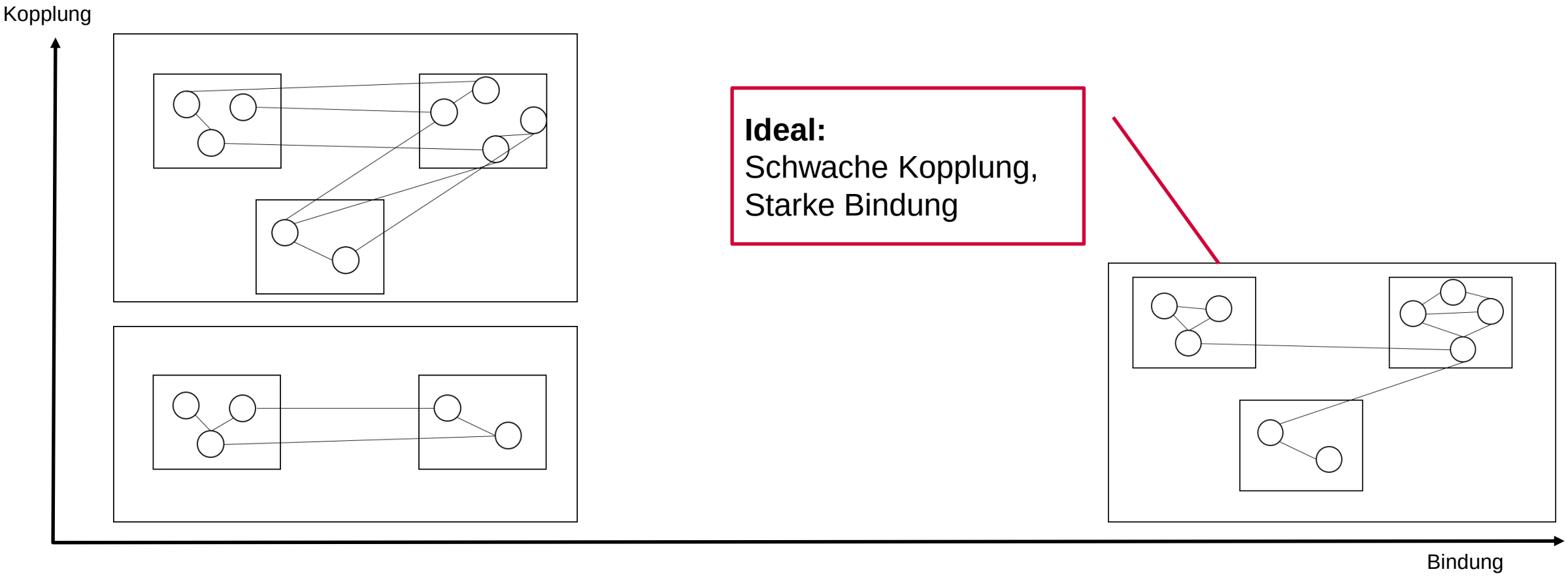
- Wahrscheinliche Änderungen sollten einfach zu implementieren sein.

"There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult.“

Tony Hoare - 1980

Wichtige Terminologie: Kopplung und Bindung

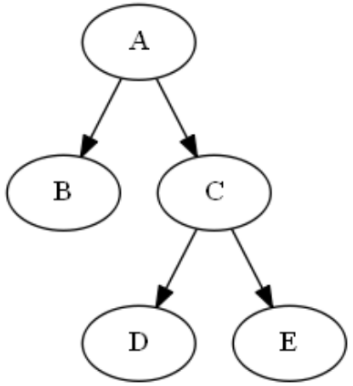
- **Kopplung**: Wie stark sind die Module verknüpft?
- **Bindung**: Wie gut bilden die Module eine logische Einheit?



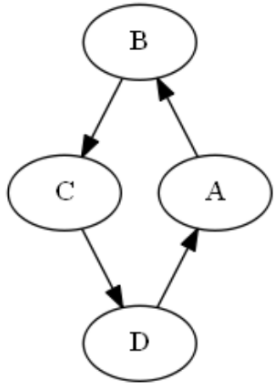
Anhaltspunkte für einfaches Design (1)

Einfaches Design: Module sind in Hierarchien organisiert

- Einfach zu verstehen (Bottom up)
- Funktionalität kann isoliert getestet werden



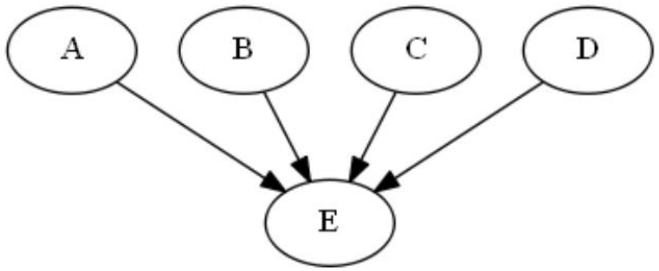
Einfach

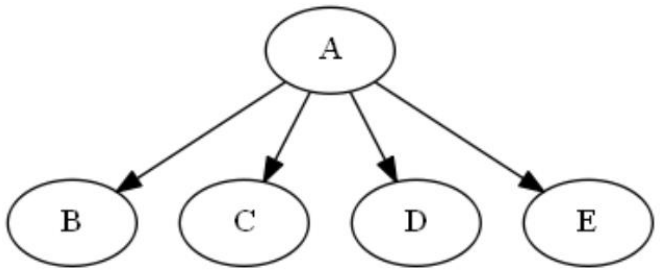


Schwierig

Anhaltspunkte für einfaches Design (2)

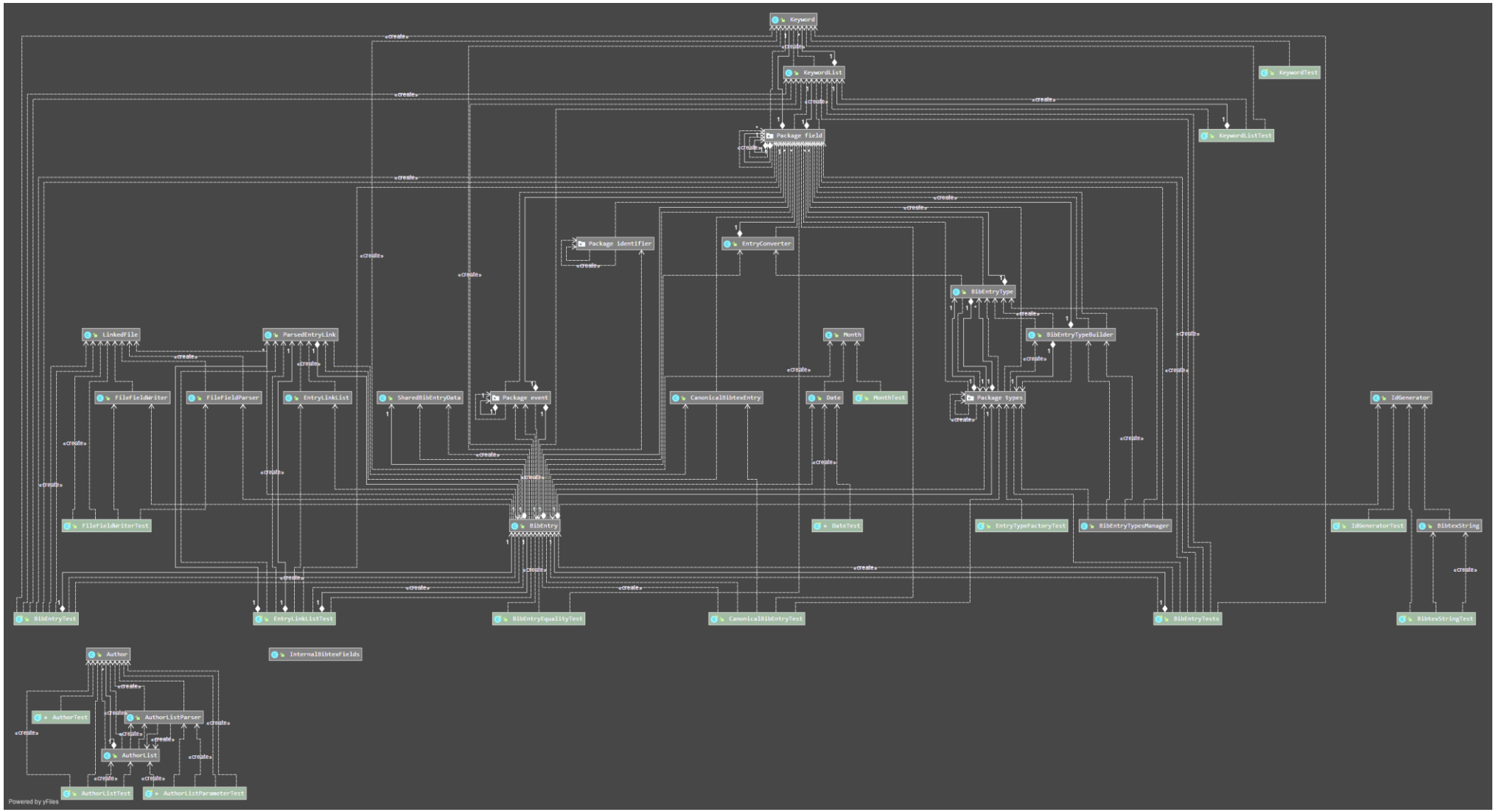
Fan-in: Anzahl einkommende Beziehungen
Fan-out: Anzahl ausgehende Beziehungen





Hoher Fan-out: Modul macht zu viel
Hoher Fan-in: Modul bietet nützlichen Service an

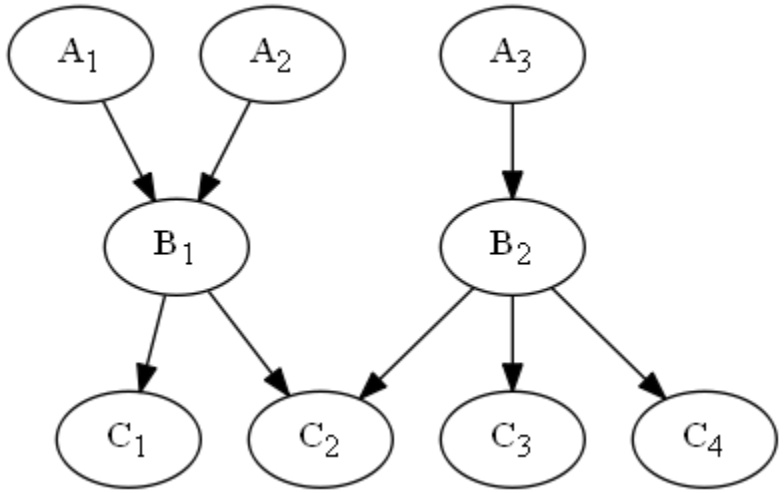
Modulgraph von JabRef (Ausschnitt)



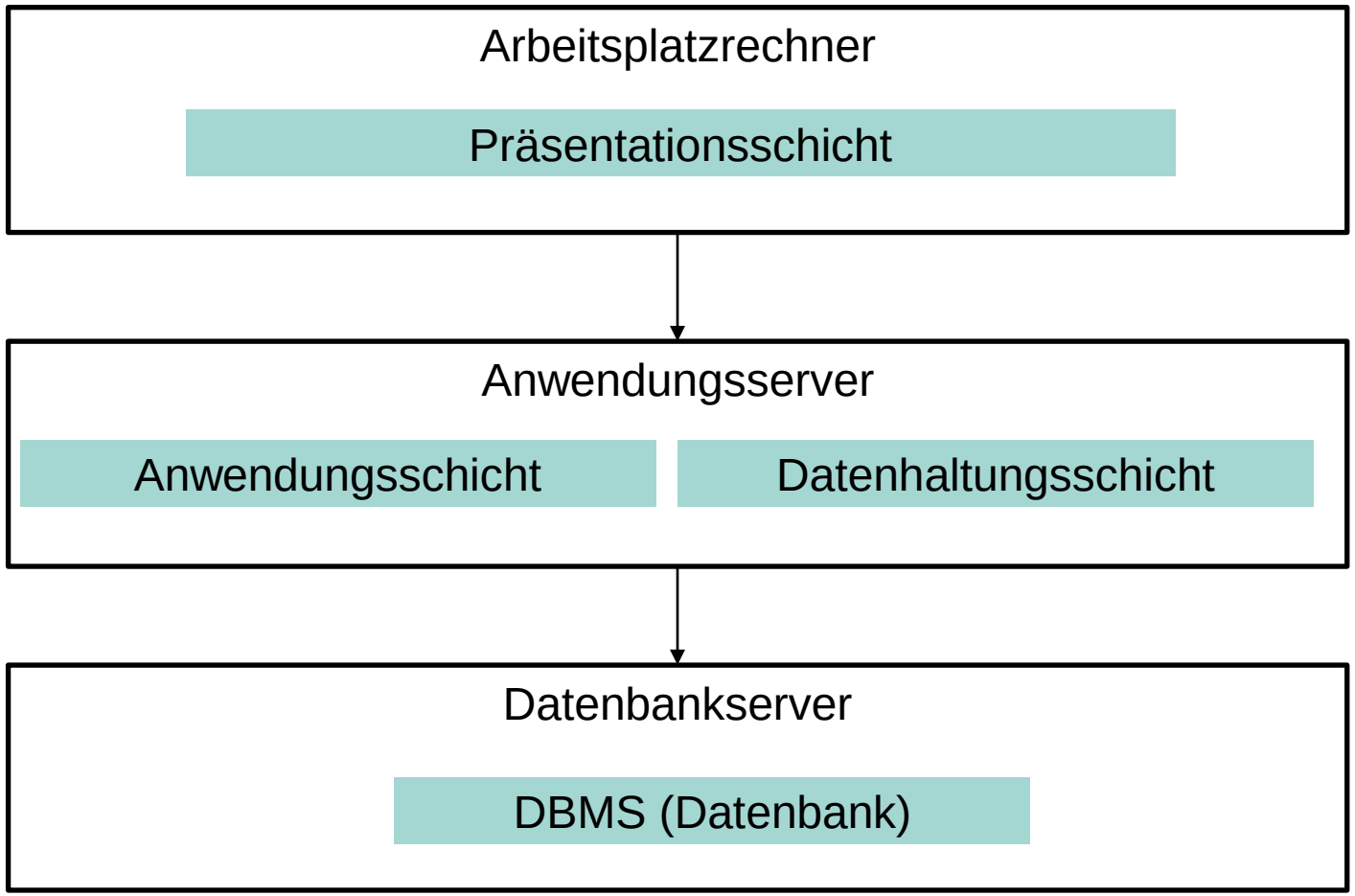
Architekturmuster: Layering

Module sind in Ebenen angeordnet

- Jedes Modul nutzt nur Module in tieferer (oder gleichen) Ebene



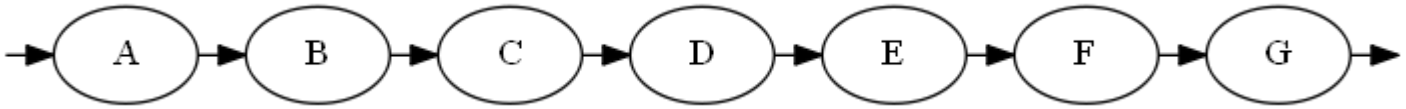
Beispiel: 3-Schichten Architektur



Architekturmuster: Pipelining

Virtuelles Fliessband

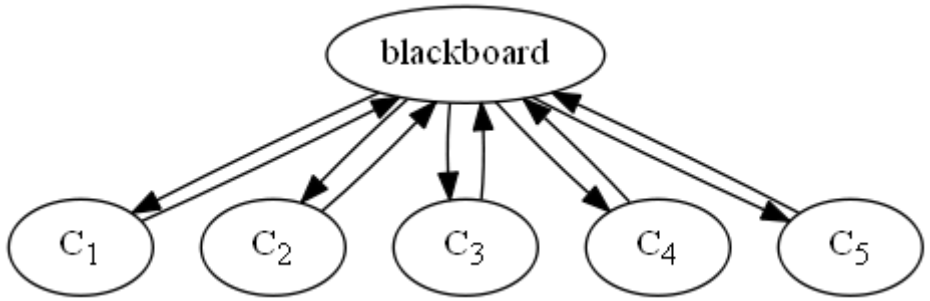
- Einzelne Verarbeitungsschritte werden hintereinander ausgeführt
- Jedes Modul nutzt nur ein weiteres Modul



Architekturmuster: Blackboard / Datarepository

Organisation wenn viele Module miteinander kommunizieren müssen

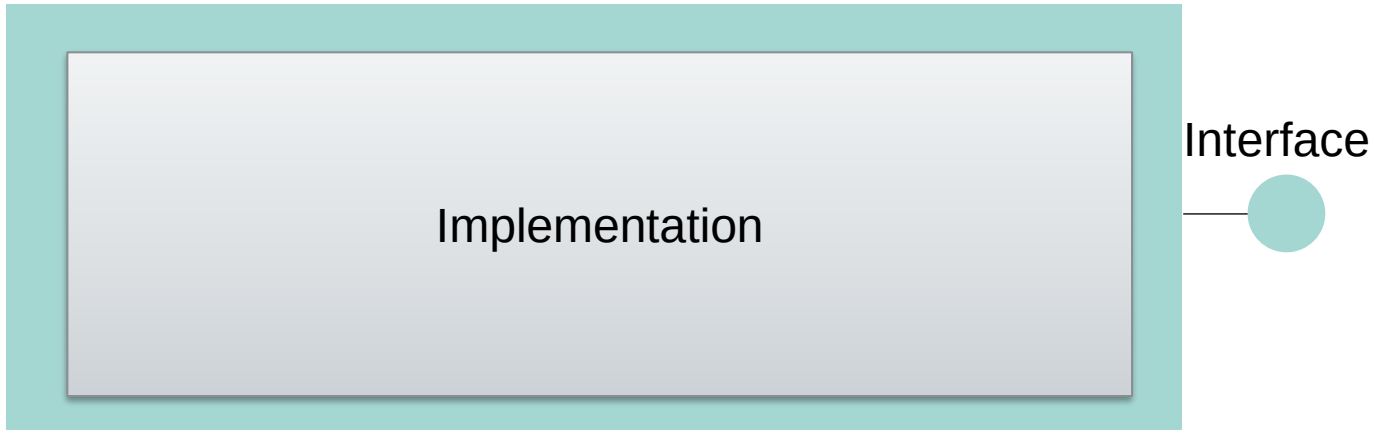
- Zentrales Modul dient als globaler Speicher und hält Information



Aufbau eines Moduls: Interface und Implementation

Interface und Implementation sollen getrennt werden!

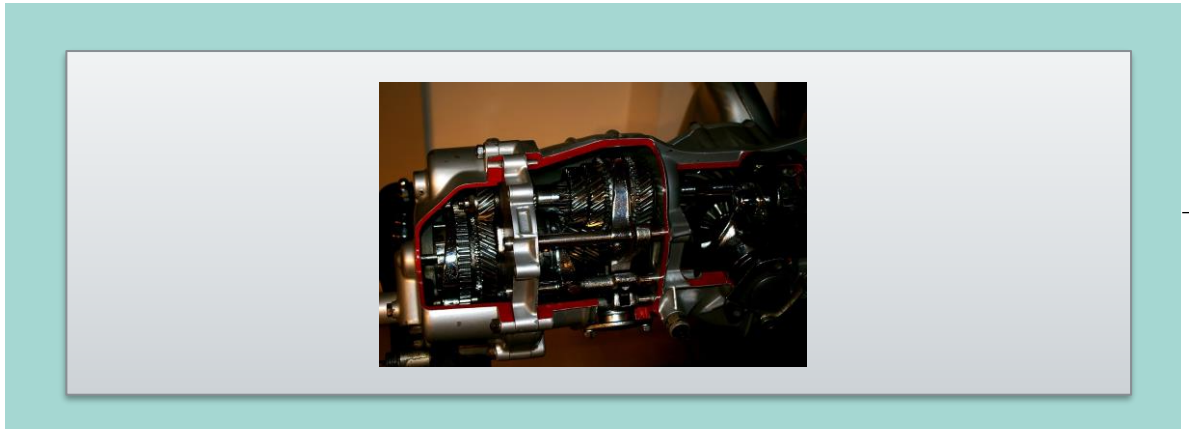
- Ermöglicht Trennung der Verantwortlichkeiten
  - Client benutzt Funktionalität
  - Server sorgt für richtige Implementation



Aufbau eines Moduls: Interface und Implementation

- Interface abstrahiert Funktionalität
  - beschreibt nur was Clients wissen müssen
  - Details der Implementation werden versteckt

Implementation sollte geändert werde können ohne dass Client verändert werden muss.



Information hiding

Komponenten die sich (wahrscheinlich) ändern werden sollten immer hinter einem Interface vor dem Client versteckt werden

Umsetzung in OO durch Klassen:

```
class Employees {
    private List<Person> employees;
    bool hire(Person p) { employees.add(p); }
    bool fire(Person p) { employees.remove(p); }
    bool isEmplyee(Person p) { employee.contains(p); }
}
```

- Information Hiding ist ein Prinzip
- Kapselung ist die Methode um das Prinzip zu erreichen.

Übung 4: Unittests und Continuous Integration

Übung 4: Unittests und Continuous Integration