HSLU Hochschule Luzern

Verteilte Systeme und Komponenten

Modularisierung und Schichtenarchitektur

Martin Bättig

Letzte Aktualisierung: 12. Oktober 2022

FH Zentralschweiz

Inhalt

- Modularisierung: Konzepte und Vorgehen
- Modularisierung mittels Schichten und Schichtenarchitektur

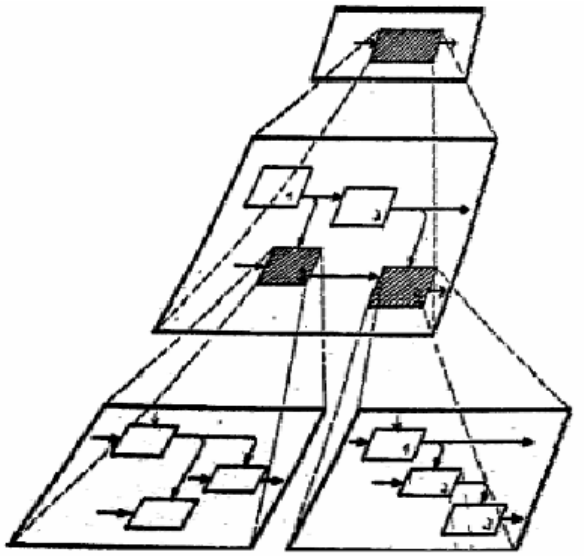
Lernziele

- Sie verstehen das Konzept der Software-Komponenten und kennen die Kriterien zur Modularisierung.
- Sie verstehen das Schichtenkonzept und wissen wie dieses Konzept bei der Schichtenarchitektur angewandt wird.

Modularisierung: Konzepte und Vorgehen

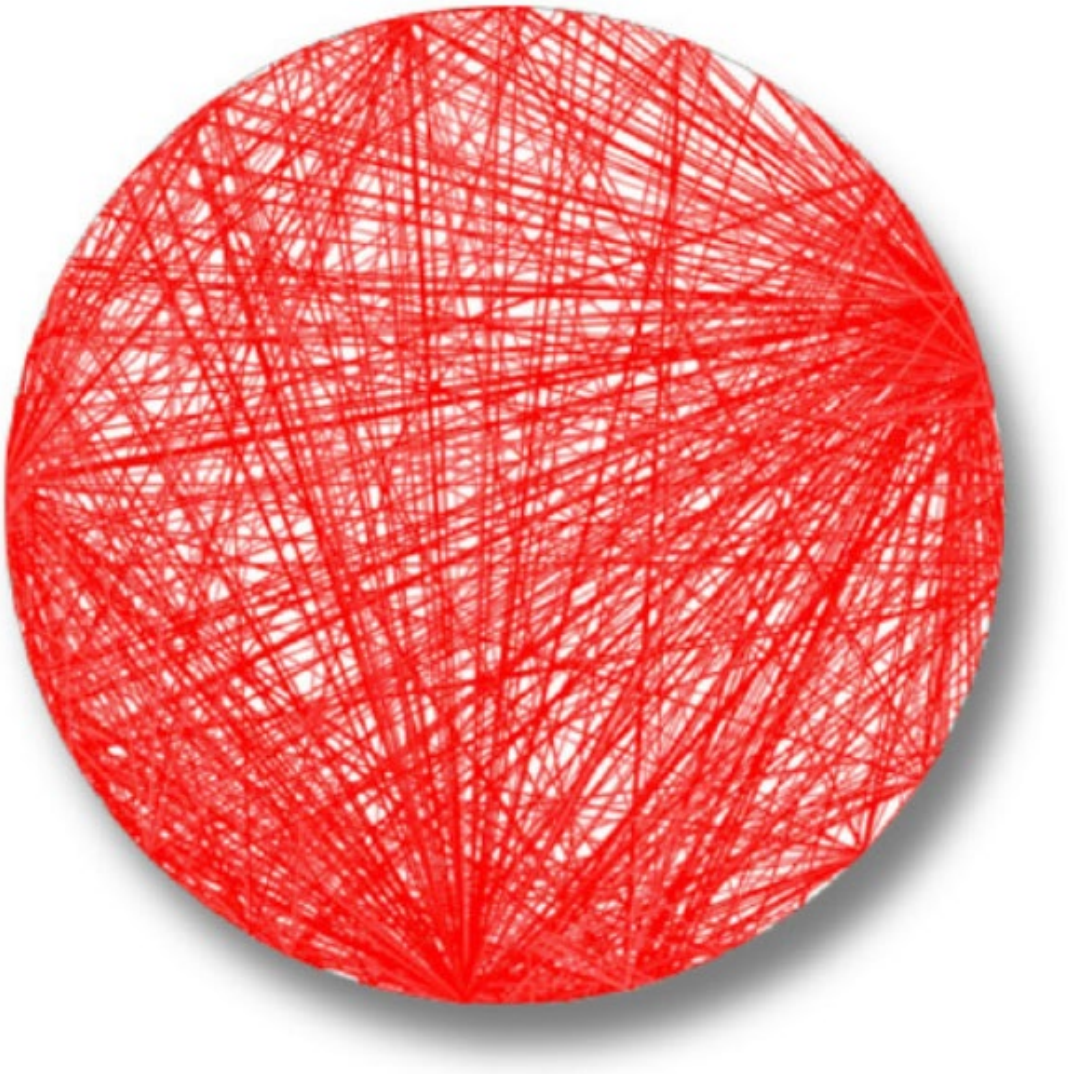
Modul: Begriff

- In sich abgeschlossener Teil des gesamten Programm-Codes, bestehend aus einer Folge von Verarbeitungsschritten und Datenstrukturen.
- Die Anwendung des Modulkonzepts im Software Engineering wurde bereits 1972 von David Parnas publiziert.



Big Ball of Mud

– Am Beispiel der Klassenbeziehungen einer realen Webapplikation (Java).

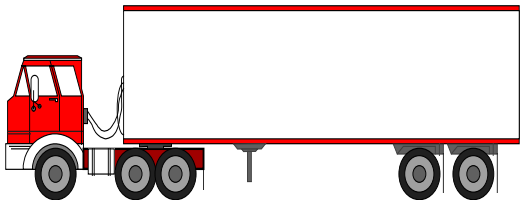


Quelle: Handbuch moderner Softwarearchitektur, Richards und Ford

Kopplung und Kohäsion: Konzept

hohe Kopplung

"Module":
- Frachtraum
- Führerkabine



geringe Kopplung

- **Kopplung** (Coupling):

  Ausmass der Kommunikation zwischen Modulen.

  - Unabhängigkeit der einzelnen Module.

  => Minimiere die Kopplung!

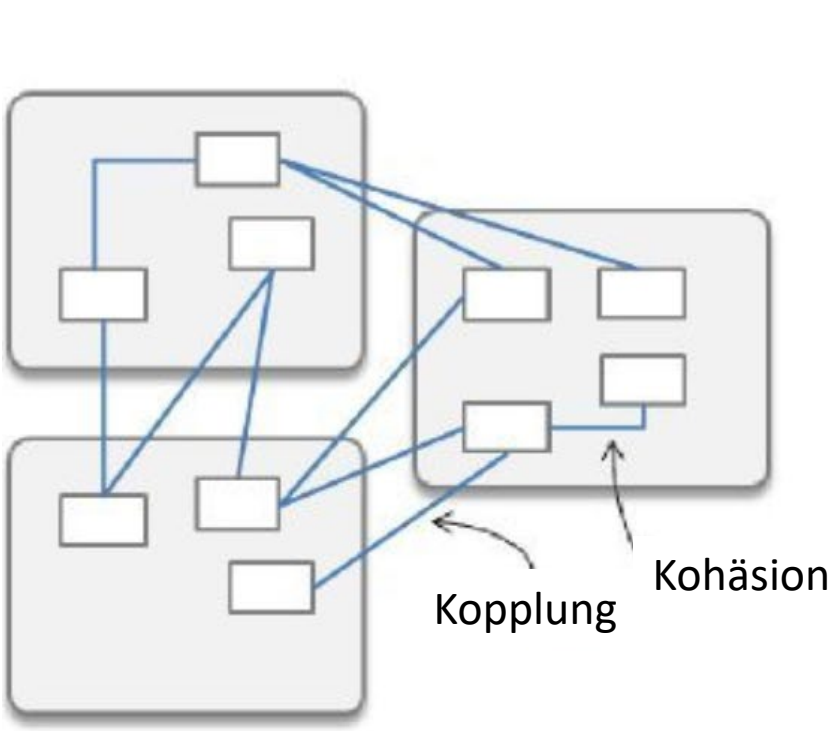
- **Kohäsion** (Cohesion):

  Ausmass der Kommunikation innerhalb eines Moduls.

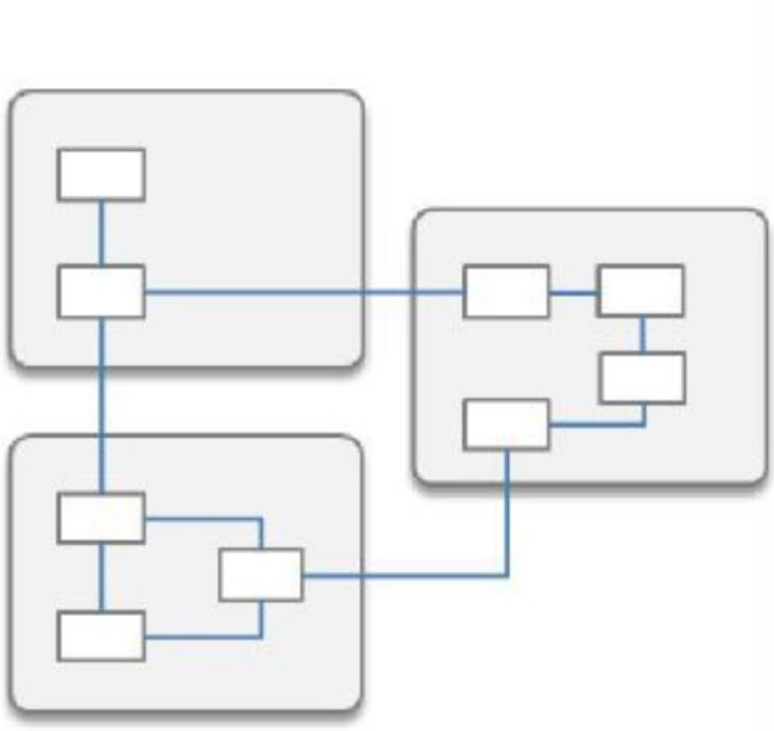
  - interner Zusammenhalt innerhalb eines Moduls.

  => Maximiere die Kohäsion!

Kopplung und Kohäsion: Beispiel (Abstrakt)



- hohe Kopplung
- geringe Kohäsion

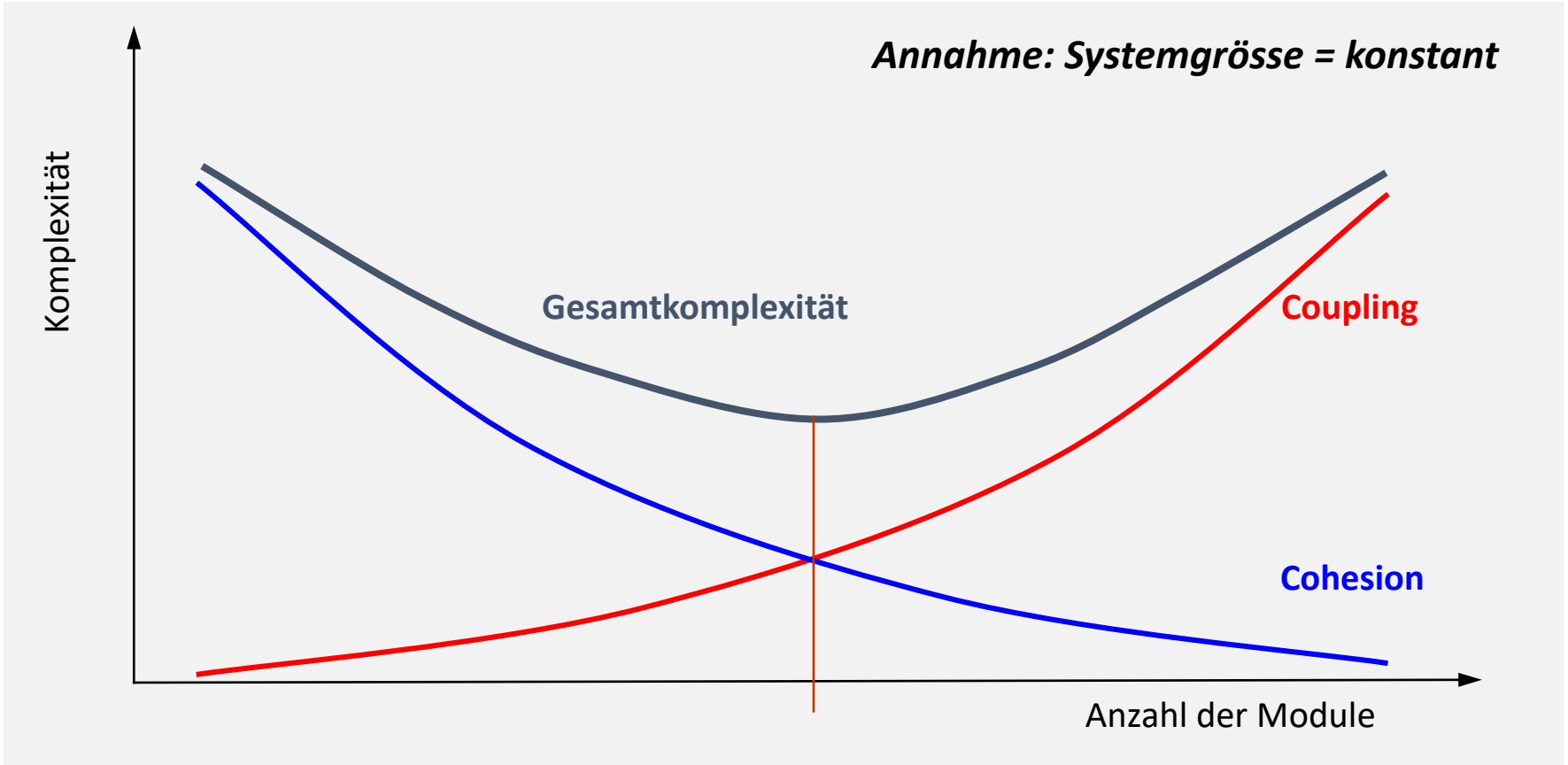


- geringe Kopplung
- hohe Kohäsion

Quelle: Modularisierung mit Java 9: Grundlagen und Techniken für langlebige Softwarearchitekturen von Guido Oelmann

Umgang mit Kopplung und Kohäsion

- Kohäsion maximieren und gleichzeitig Kopplung minimieren ist nicht immer möglich.
- Optimierungsaufgabe!

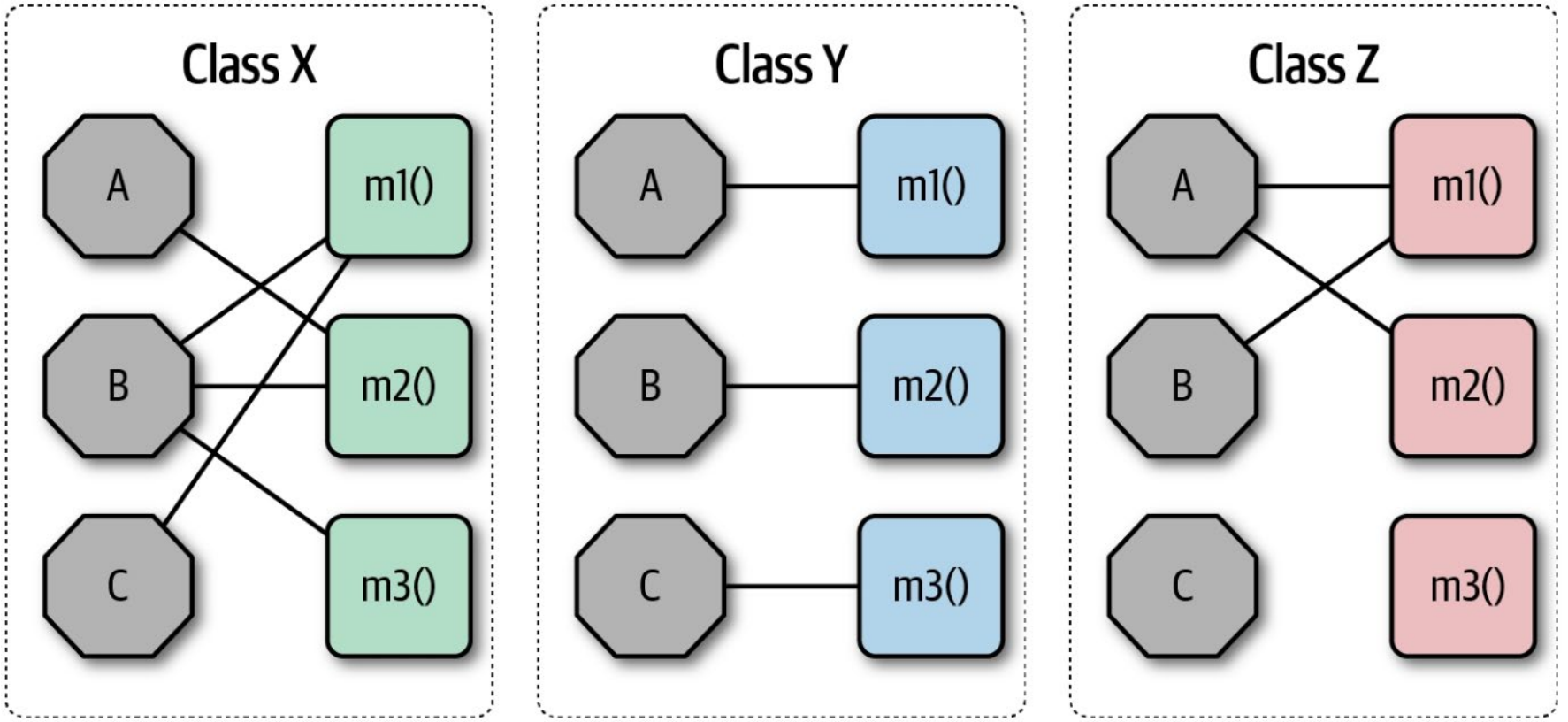


Arten der Kohäsion

- **Funktionale Kohäsion:** Alle Teile haben eine ineinandergreifende Beziehung zu einander.
- **Sequentielle Kohäsion:** Input/Output-Beziehungen.
- **Kommunikatorische Kohäsion:** Kommunikationsketten bzgl. Informationsverarbeitung.
- **Prozedurale Kohäsion:** Ausführung in bestimmter Reihenfolge nötig.
- **Temporale Kohäsion:** Zeitbezogene Abhängigkeit, z.B. beim Startup.
- **Logische Kohäsion:** Logische, aber nicht funktionale Beziehung.
- **Zufällige Kohäsion:** Einheiten sind zufällig im selben Modul.

Messung der Kohäsion

– LCOM (Lack of Cohesion in Methods): Summe der nicht gemeinsam genutzten Methodensätze, welche nicht auf geteilte Felder zugreifen.



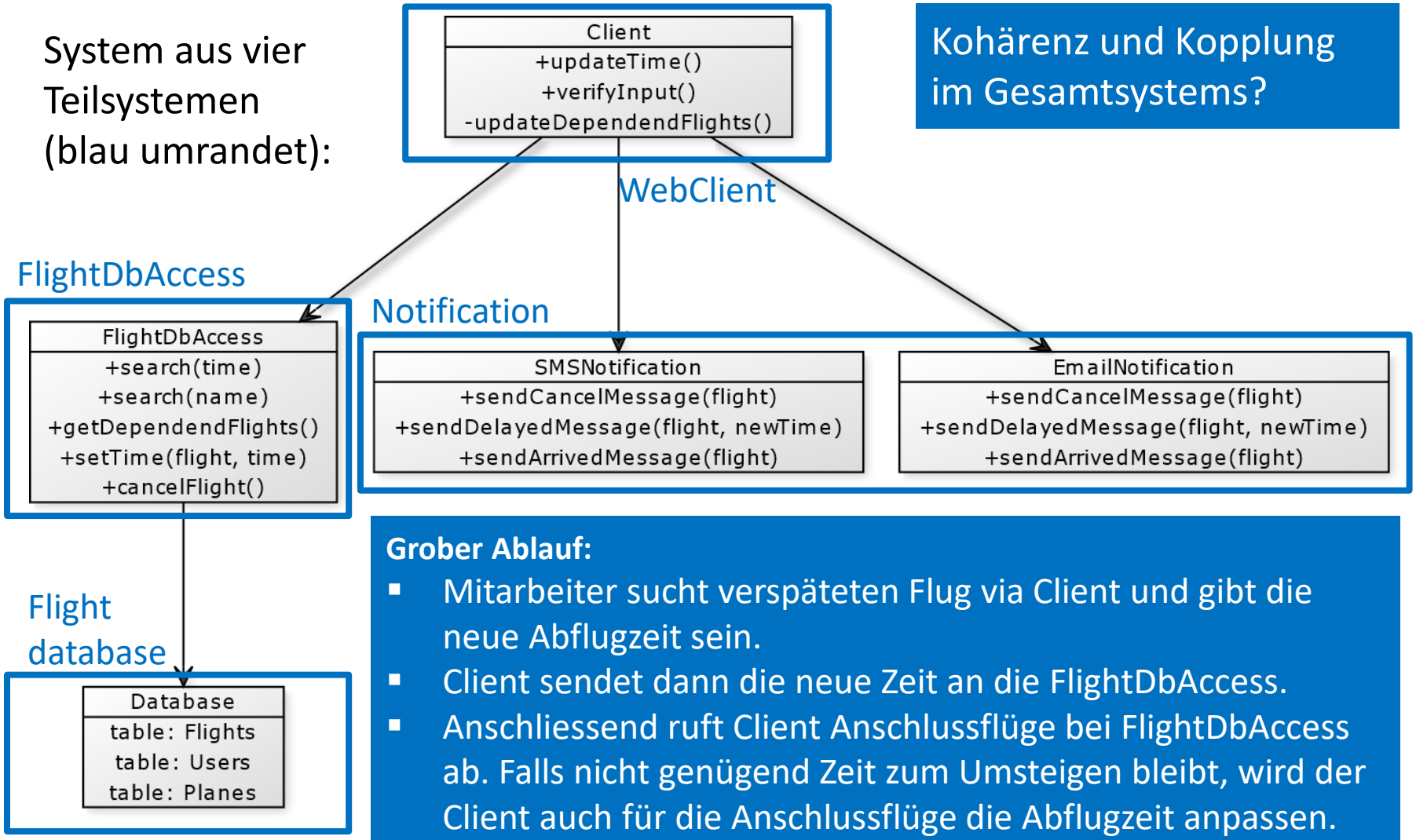
Quelle: Handbuch moderner Softwarearchitektur, Richards und Ford

Arten der Kopplung

- **Laufzeitumgebung, Ausführungsort:** Module müssen in der selben Laufzeitumgebung oder auf dem selben System ausgeführt werden.
- **Technologie:** Gekoppelte Module müssen (teilweise) dieselben Technologien verwenden.
- **Zeit:** Module müssen zur selben Zeit aktiv sein.
- **Daten und Formate:** Module müssen die selben Datenformate parsen und verstehen (z.B. Datum oder Headers).

Übung: Anpassung von Abflugzeiten.

System aus vier Teilsystemen (blau umrandet):



Kohärenz und Kopplung im Gesamtsystems?

Grober Ablauf:

- Mitarbeiter sucht verspäteten Flug via Client und gibt die neue Abflugzeit sein.
- Client sendet dann die neue Zeit an die FlightDbAccess.
- Anschliessend ruft Client Anschlussflüge bei FlightDbAccess ab. Falls nicht genügend Zeit zum Umsteigen bleibt, wird der Client auch für die Anschlussflüge die Abflugzeit anpassen.
- Anschliessend werden die Kunden über das bevorzugte Medium (SMS od. Email) informiert.

Arten von Modulen

- **Bibliotheken:** Sammlung von oft benötigten und thematisch zusammengehörenden Funktionen.
  Beispiele: Datum-Modul (Operationen auf Kalenderdaten), Trigonometrie-Modul (Winkelfunktionen), Systemschnittstellen-Modul.

- **Abstrakte Datentypen:** Modul implementiert einen neuen Datentyp und stellt die darauf definierten Operationen zur Verfügung.
  Beispiele: Mehrdimensionale Tabellen, komplexe Zahlen und Koordinaten.

Arten von Modulen (forts.)

- **Physische Systeme** insbesondere in technischen Anwendungen der Informatik.
  Beispiele: Sensorsystem, Gerätetreiber, Kommunikation, Anzeigetafel, usw.

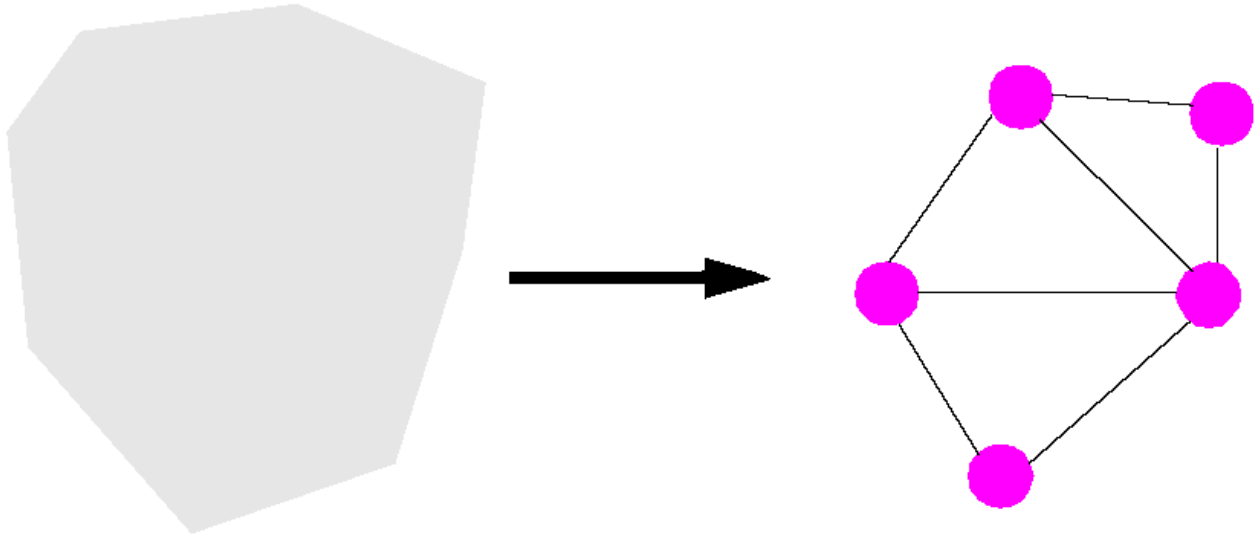
- **Logisch-konzeptionelle Systeme:** logisch-konzeptionelle Systeme modellieren und für andere Komponenten auf hoher Abstraktionsstufe nutzbar machen.
  Beispiele: Grafik, Datenbank, Messaging, GUIs, usw.

Wichtige Kriterien des modularen Entwurfs

- **Zerlegbarkeit (Top-Down)**
  Teilprobleme sind unabhängig voneinander entwerfbar.
- **Kombinierbarkeit (Bottom-Up)**
  Module sind unabhängig voneinander (wieder-)verwendbar.
- **Verständlichkeit**
  Module sind unabhängig voneinander zu verstehen.
- **Stetigkeit**
  Kleine Änderungen der Spezifikation führen nur zu kleinen Änderungen im Code.

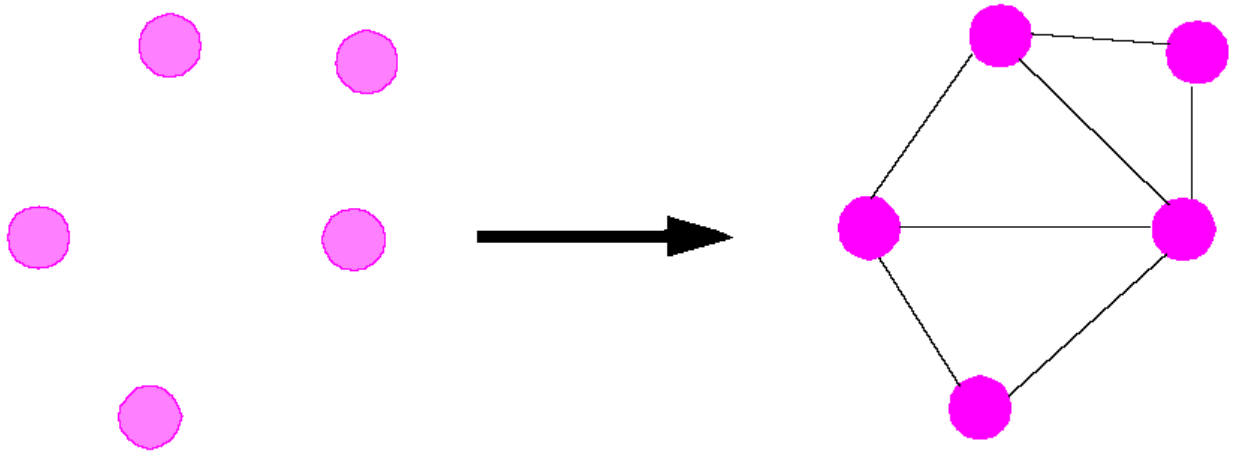
Zerlegbarkeit (Top-Down)

- Zerlege ein Softwareproblem in eine Anzahl weniger komplexe Teilprobleme und verknüpfe diese so, dass die Teile möglichst unabhängig voneinander bearbeitet werden können.
- Die Zerlegung wird häufig rekursiv angewendet: Teilprobleme können so komplex sein, dass sich eine weitere Zerlegung aufdrängt.



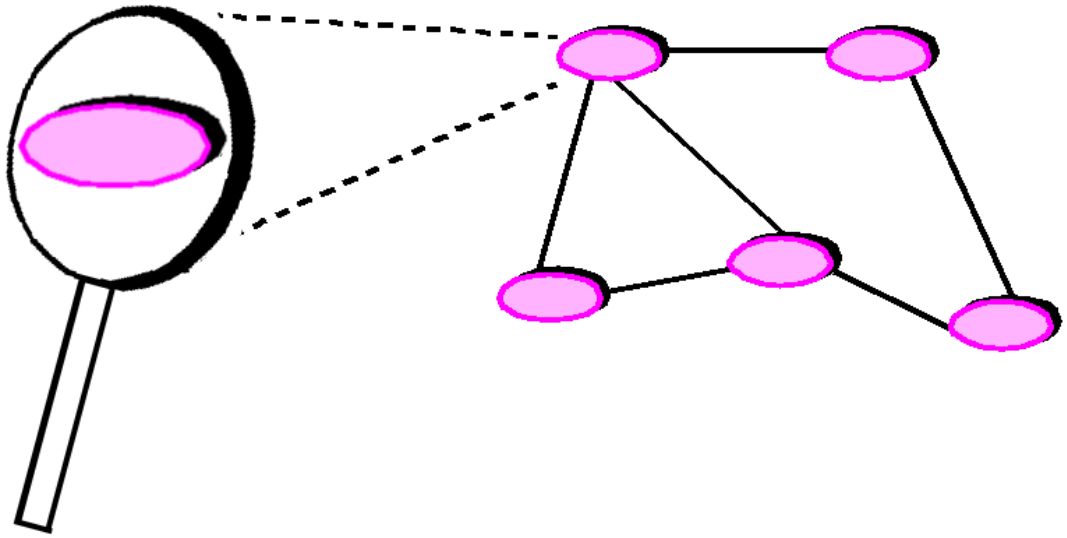
Kombinierbarkeit (Bottom-Up)

- Strebe möglichst frei kombinierbare Software-Elemente an, die sich auch in einem anderen Umfeld wieder einsetzen lassen.
- Kombinierbarkeit und Zerlegbarkeit sind voneinander unabhängige Eigenschaften.



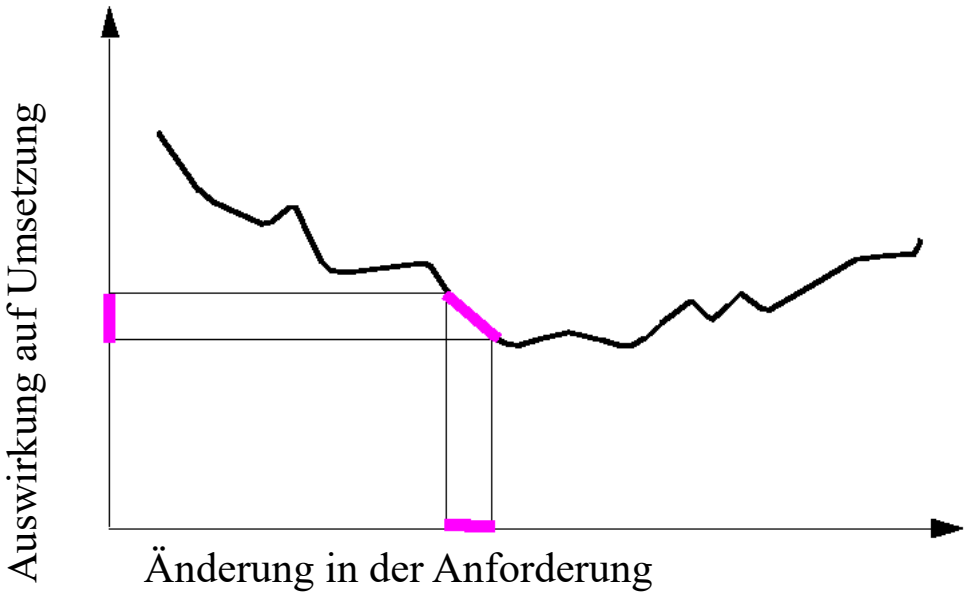
Verständlichkeit

- Der Quellcode eines Moduls soll auch verstehbar sein, ohne dass man die anderen Module des Systems kennt.
- Softwareunterhalt setzt voraus, dass die Teile eines Systems unabhängig von einander zu verstehen und zu warten sind.



Stetigkeit

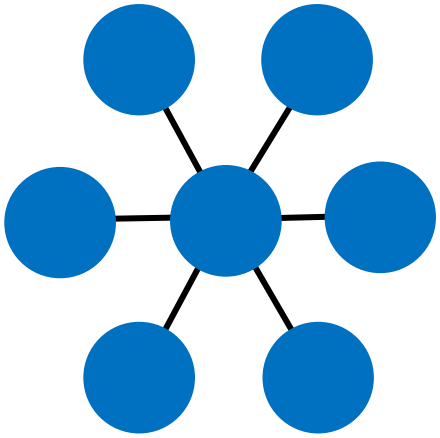
- Von einer kleinen Änderung der Anforderungen soll auch nur ein kleiner Teil der Module betroffen sein.
- Es ist oft unvermeidlich, dass sich im Laufe eines Projektes die Anforderungen ändern. Stetigkeit bedeutet, dass dies nicht die ganze Systemstruktur beeinflusst, sondern sich lediglich auf einzelne Module auswirkt.



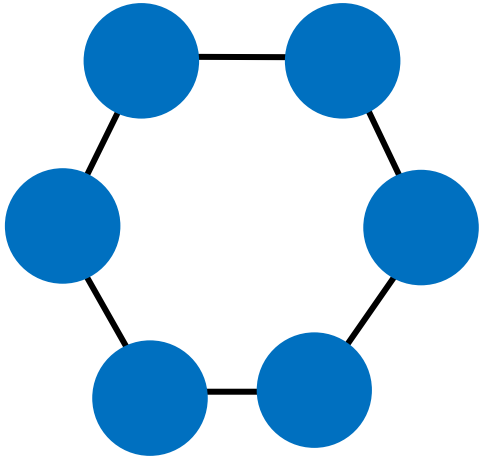
Beispiel: Auswirkung von Änderungen

– Wie wirkt sich eine Änderung an einem Modul aus?

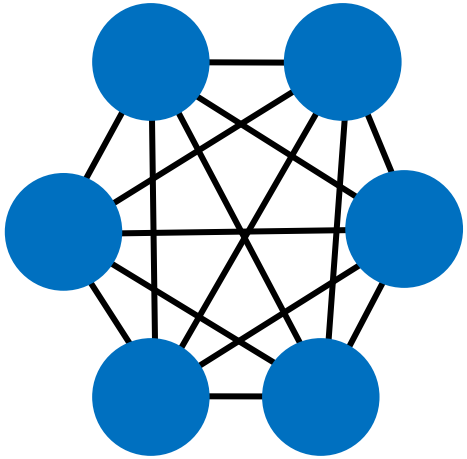
Aufteilung A:



Aufteilung B:



Aufteilung C:



Prinzipien des modularen Entwurfs

- **Lose Kopplung:** Schmale Schnittstellen um nur das wirklich Benötigte auszutauschen.
- **Starke Kohäsion:** Hoher Zusammenhalt innerhalb eines Moduls.
- **Information Hiding**: Modul ist nach aussen nur über seine Schnittstelle bekannt.
- **Wenige Schnittstellen:** minimale Anzahl Schnittstellen (Aufrufe, Daten).
- **Explizite Schnittstellen:** Aufrufe und gemeinsam genutzte Daten sind im Code ersichtlich.
- **Wenige Abhängigkeiten pro Modul:** Reduktion der Auswirkung von Änderungen auf andere Module.

Modularisierung: Iteratives Vorgehen

1. **Zerlegung (Top-Down oder Bottom-Up) unter Anwendung der Prinzipien:**
   - Wenig Kopplung und viel Kohäsion.
   - Information-Hiding.
   - Wenige und explizite Schnittstellen.
2. **Beurteilung hinsichtlich der Kriterien:**
   - Zerlegbarkeit: Module aufgeteilt und unabhängig bearbeitbar?
   - Kombinierbarkeit: Können Module wiederverwendet werden?
   - Verständlichkeit: Viel Kohärenz und wenig Kopplung?
   - Stetigkeit: Auswirkung von Änderungen?
   - Korrekte Modulart: Bibliothek, abstrakter Datentyp, physische Kapsel oder logische Kapsel.
3. **Falls Kriterien nicht zufriedenstellend: Zurück zu 1.**

Übung: Modularer Entwurf für Testautomationssystem

Ein System zum Test von Kommandozeilenprogrammen soll in Module zerlegt werden:

- Das System liest seine Konfiguration aus einer Datei. Die Datei enthält die Angabe über das auszuführende Programm und den Testfällen.
- Jeder Testfall besteht aus einer Eingabe und einer erwarteten Ausgabe.
- Das System muss das zu testende Programm für jeden Testfall isoliert in einem neuen Prozess ausführen.
- Anschliessend soll das System die Ausgabe des Programms mit der zu erwartenden Ausgabe des Testfalls vergleichen.
- Nach Ausführung aller Testfälle erstellt das System einen Bericht über den Testlauf. Dieser Bericht enthält für jeden ausgeführten Testfall: Eingabe, erwartete Ausgabe, effektive Ausgabe und das Resultat (Passed, Failed).

Dekomposition des Systems in Module? Top-Down? Bottom-Up?

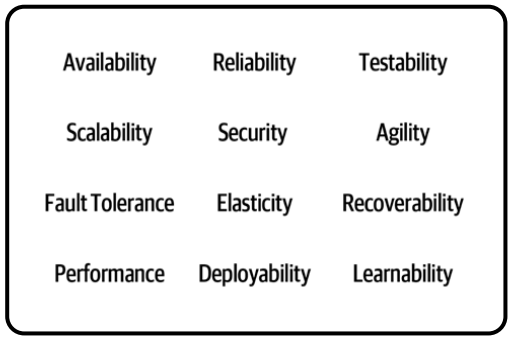
Modularisierung

Ein System sinnvoll in Module aufzuteilen ist eine der anspruchsvollsten Aufgaben in der Informatik.

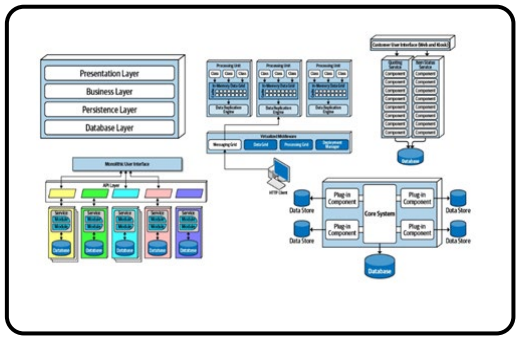
- Lesen Sie dazu den Klassiker **On the Criteria To Be Used in Decomposing Systems into Modules** von David L. Parnas. https://www.win.tue.nl/~wstomv/edu/2ip30/references/criteria_for_modularization.pdf
- Oder je nach Vorliebe folgenden Blog-Beitrag, welcher den Inhalt von Parnas mit der modernen Softwareentwicklung von Heute vergleicht: https://blog.acolyer.org/2016/09/05/on-the-criteria-to-be-used-in-decomposing-systems-into-modules/

Schichtenarchitektur

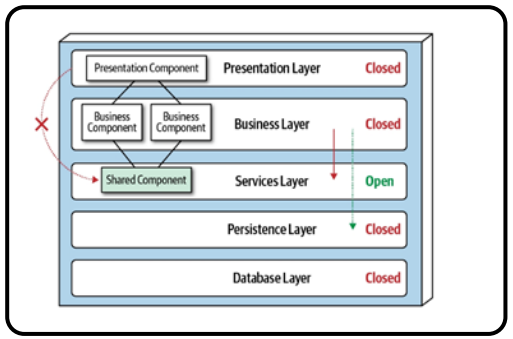
Was ist Softwarearchitektur?



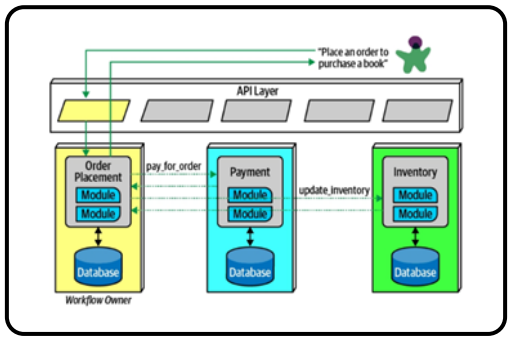
Architektonische Eigenschaften



Struktur



Architektonische Entscheidungen

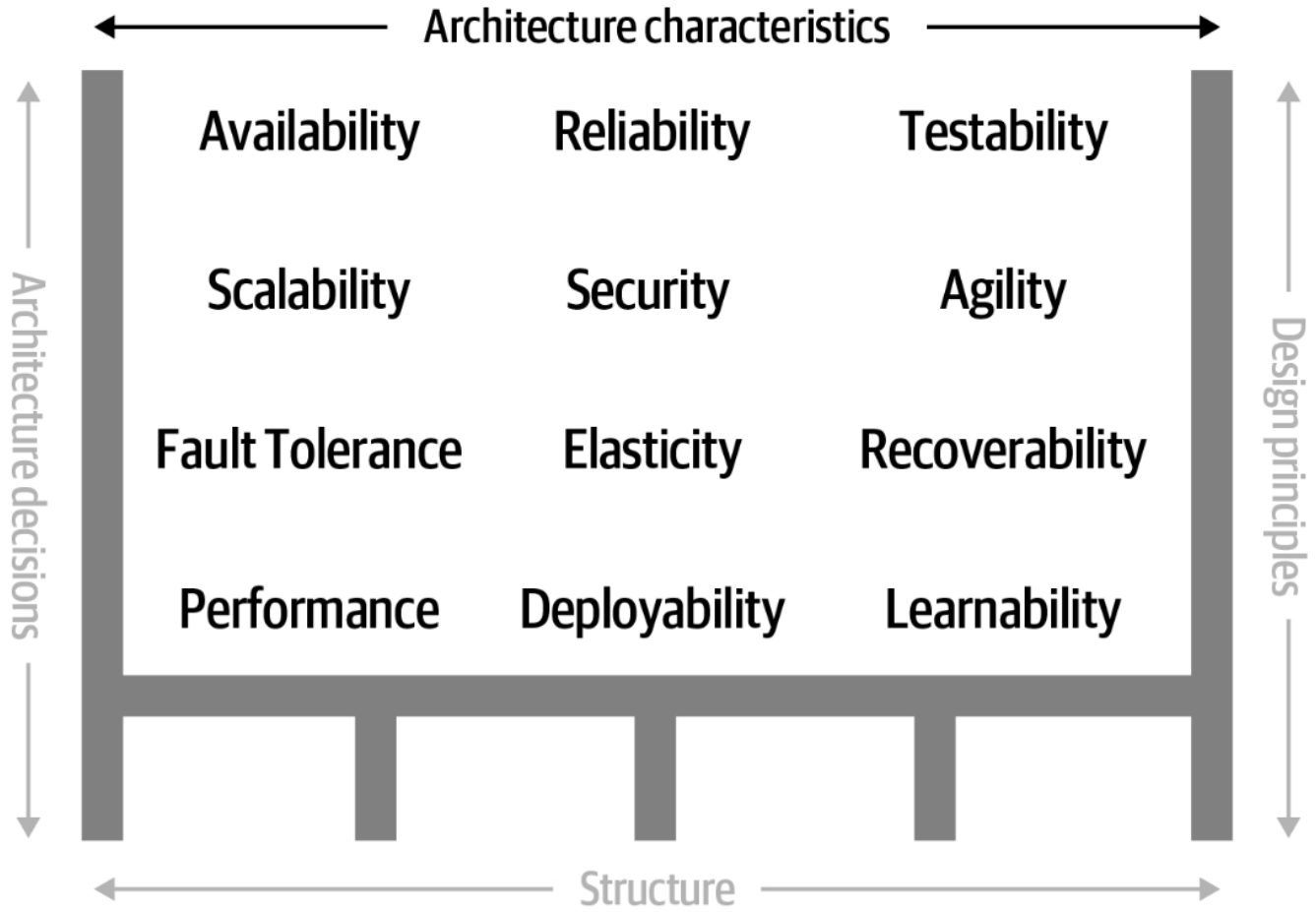


Entwurfsprinzipien

Quelle: Handbuch modernder Softwarearchitektur

Architektonische Eigenschaften

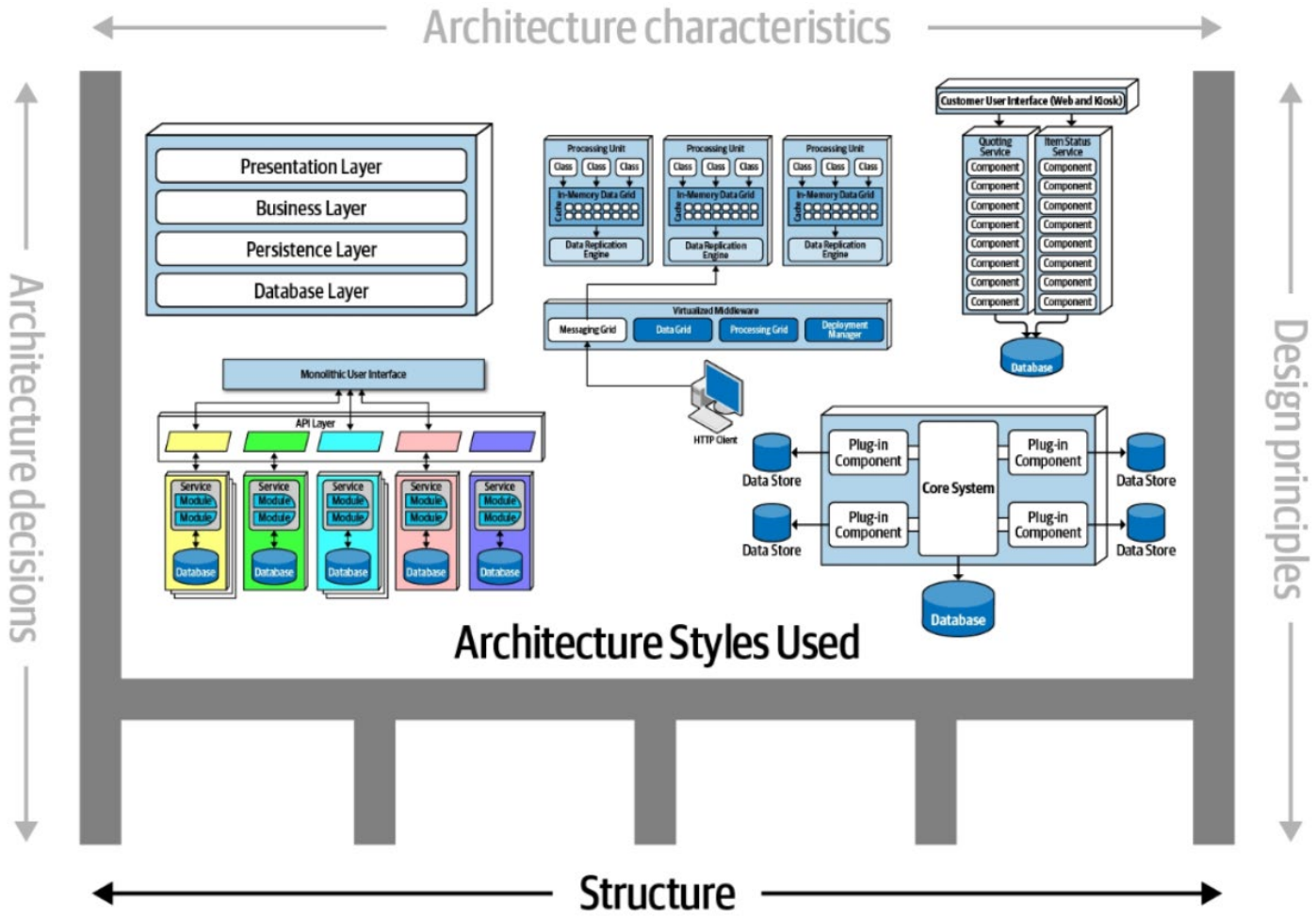
– Nichtfunktionale Eigenschaften eines Systems, welche zur ordentlichen Funktionsweise notwendig sind.



Quelle: Handbuch modernder Softwarearchitektur

Struktur

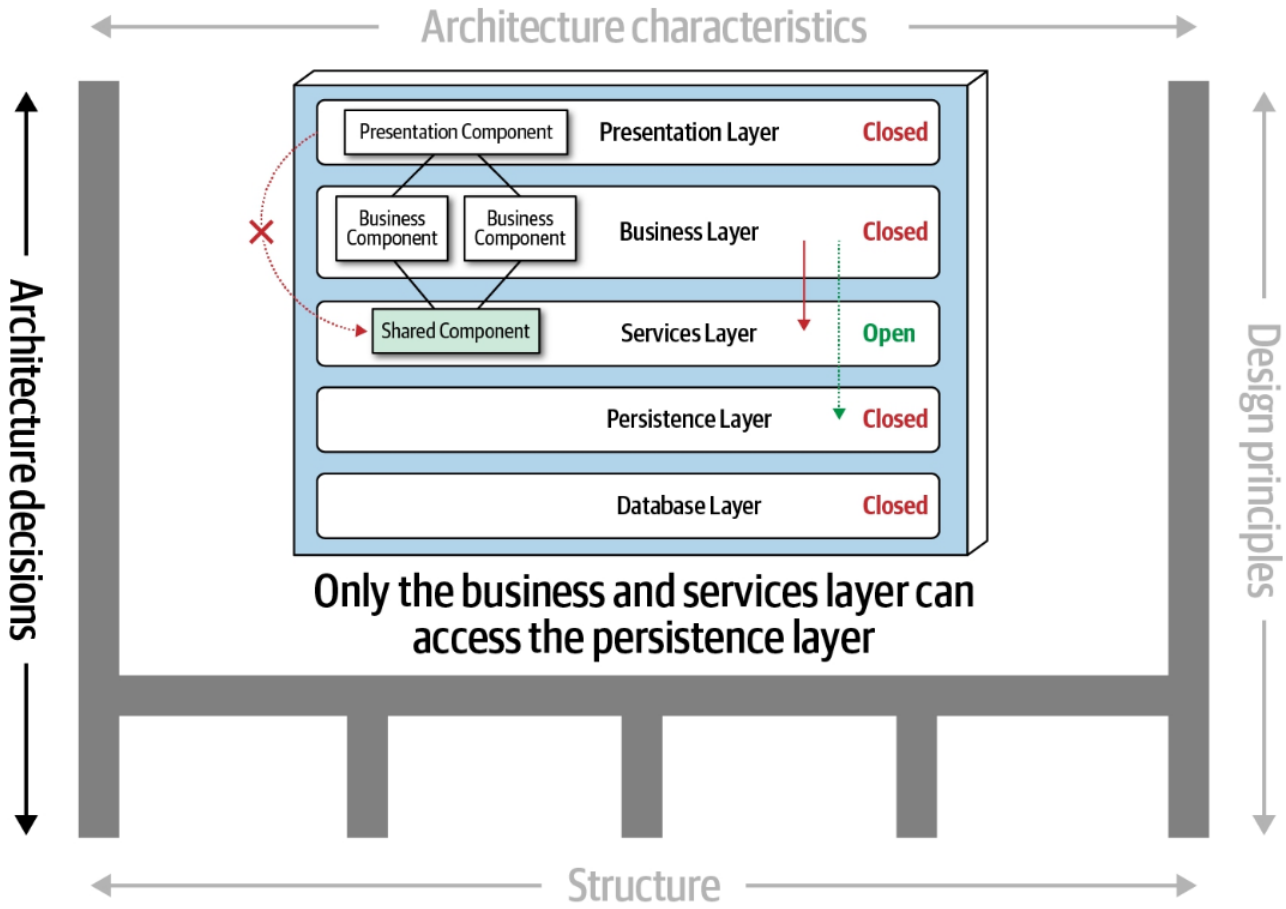
– Verwendete Architekturstile



Quelle: Handbuch modernder Softwarearchitektur

Architektonische Entscheidungen

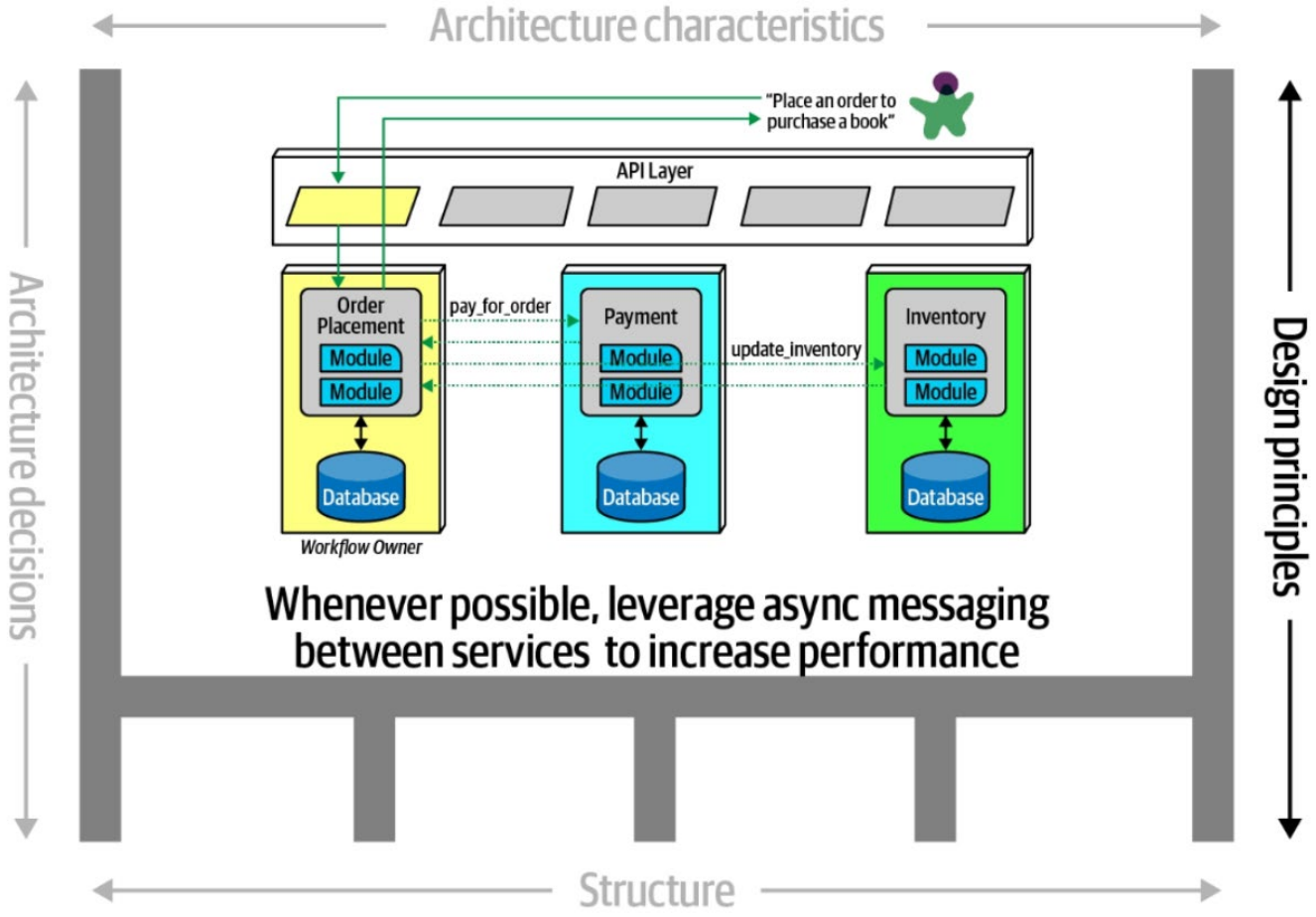
– Regeln, welche Entwickler beim Entwurf der Komponenten befolgen müssen.



Quelle: Handbuch modernder Softwarearchitektur

Entwurfsprinzipien

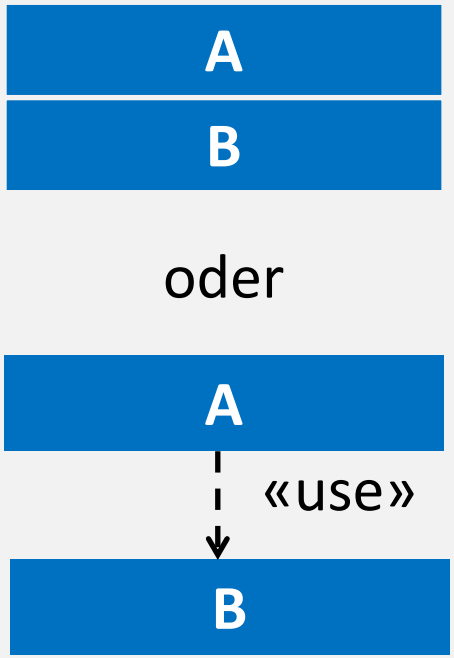
– Richtlinien, welche Entwickler bei Entwurf der Komponenten des Systems anwenden sollen.



Quelle: Handbuch modernder Softwarearchitektur

Was sind Schichten?

Darstellung:



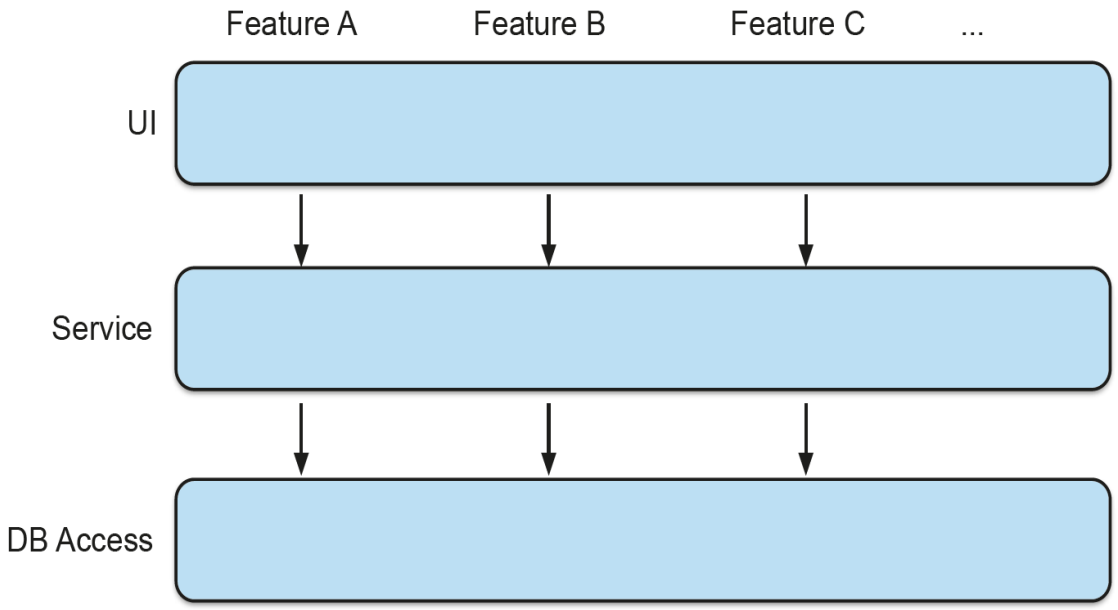
– – – –>

In diese Richtung ist die Verwendung erlaubt

- Modulhierarchie, in welcher öffentliche Methoden in Schicht B von der Software in Schicht A genutzt werden dürfen, aber nicht umgekehrt.
- Man spricht von einer use-Beziehung wenn das korrekte Funktionieren von A von einer korrekten Implementation von B abhängt.

Schichtenarchitektur

- Technische Modularisierung oft entlang organisatorischen Einheiten.
- Komponenten einzelnen Schichten zugeordnet:

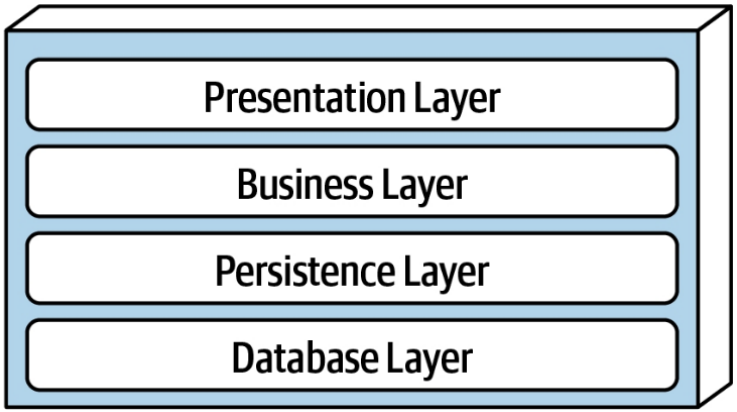


- Basis für komplexere Architekturen.
- Deployment-Monolith: Typischerweise müssen immer alle Schichten gleichzeitig angepasst und vollständig ausgeliefert / installiert werden.

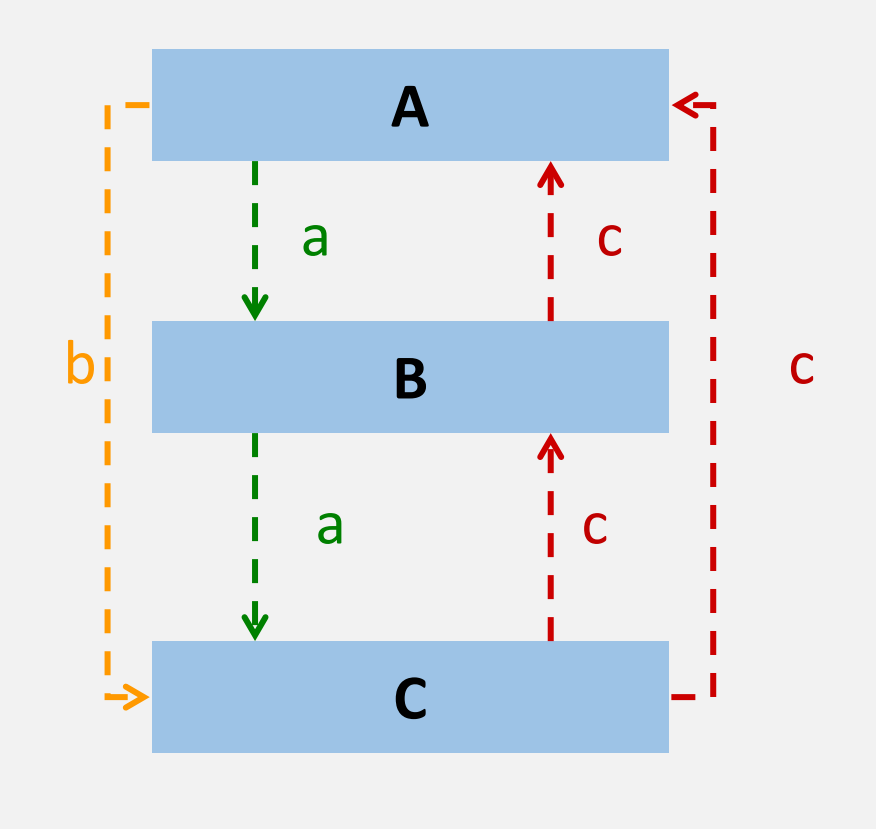
Quelle: Modulare Software Architektur, Herbert Dowalil

Typische Schichten

- **Presentation Layer (Anwendungen):** Darstellung und Benutzerinteraktion mit der Geschäftslogik einer Applikation.
- **API Layer (Services):** Bereitstellung des Zugriffs auf die Geschäftslogik.
- **Business Layer:** Geschäftslogik (Funktionalität und Datenstrukturen).
- **Service Layer:** Hilfsfunktionen für Komponenten einer darüberliegenden Schicht.
- **Persistence Layer:** Abstraktion des Datenzugriffs (z.B. möchten wir Kundendaten oder Kontoinformationen laden und nicht einfach Textfiles).
- **Database Layer:** Zugriff auf den Storage (z.B. Definition des Schemas bei relationalen Datenbanken).

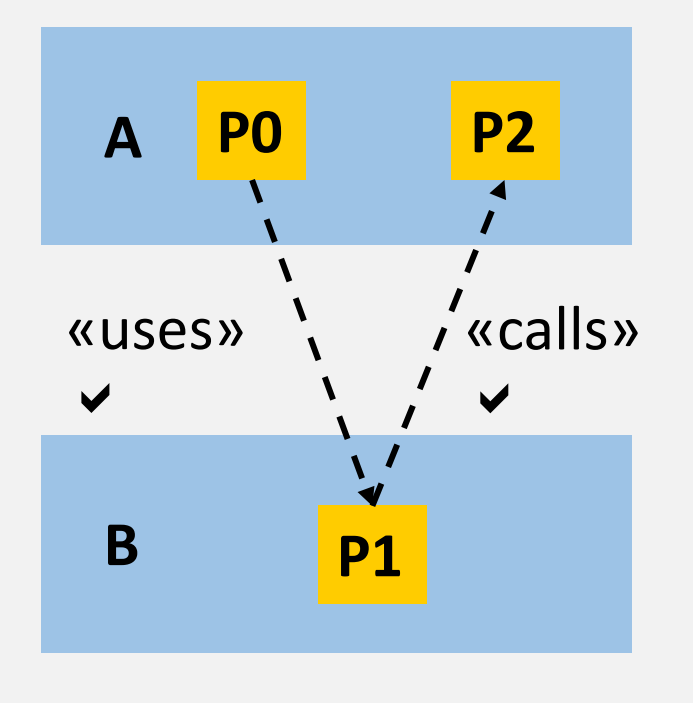


Schichtenbeziehungen: Zulässigkeit



- a: ok.
- b: gefährlich, falls nicht vorgesehen, wie z.B. bei offenen Schichtarchitekturen.
- c: nicht zulässig (keine zyklischen Abhängigkeiten zwischen Schichten)!

Weitere Schichtenbeziehung

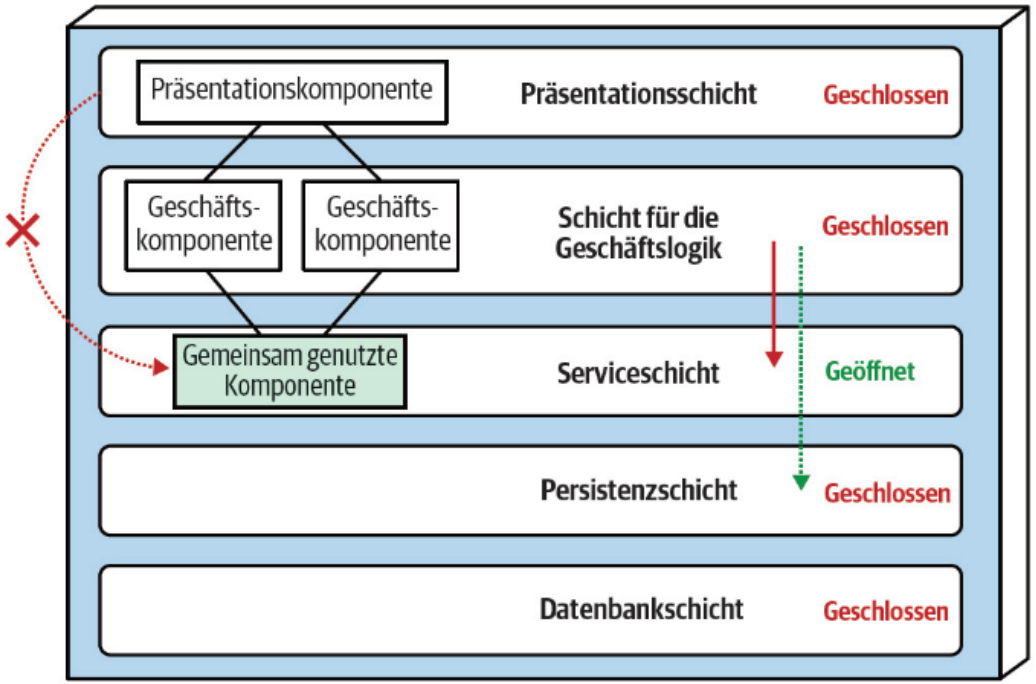


- Eine Klasse P1 kann P2 aufrufen ohne eine use-Beziehung mit P2 zu haben.

- **Beispiel:** P2 sei ein Errorhandler, dessen Referenz von P0 an P1 übergeben wurde. Die Referenz des Errorhandlers ist nicht in P1 festkodiert

Offene Schichtenarchitektur: Offene und geschlossene Schichten

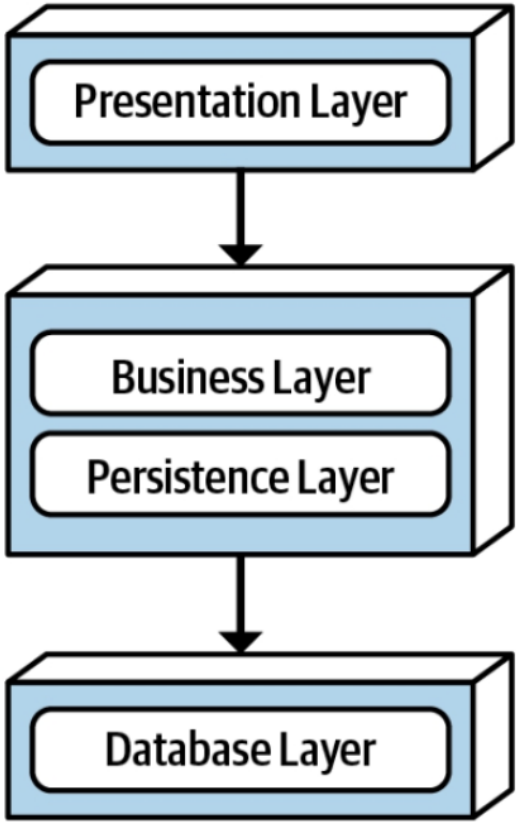
- Offene Schichtenarchitektur: Schichten können offen oder geschlossen sein.
- Offene Schichten können von direkt darüberliegender Schicht übersprungen werden:
  - Vermeidung horizontaler Abhängigkeiten.
  - Performancegewinn, falls eine Schicht Anfragen nur weiterreichen würde.



⇒ **Alternative:** Schichten rekursiv verschachteln.

Quelle: Handbuch moderner Softwarearchitektur, Richards und Ford

Schichten vs. Tier



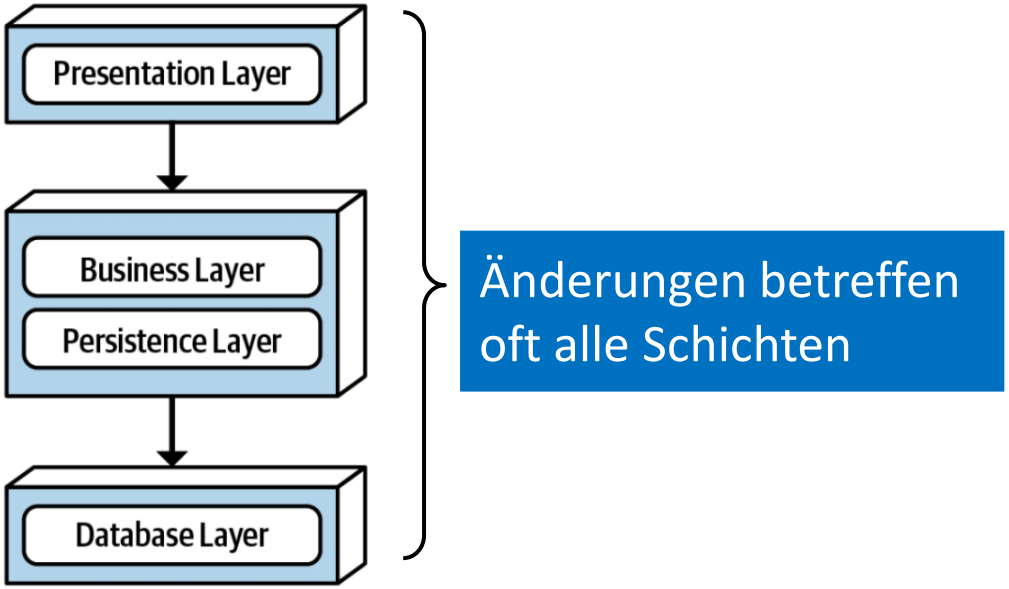
- **Schichten:** Logische Separierung.
- **Tier:** Zusätzlich eine physische Separierung oft kombiniert mit Verteilung (*).

(*) heutzutage oft als Service.

Beispiel (links): vier Schichten auf drei Tiers.

Quelle: Handbuch moderner Softwarearchitektur, Richards und Ford

Bewertung der Schichtbasierte Architektur

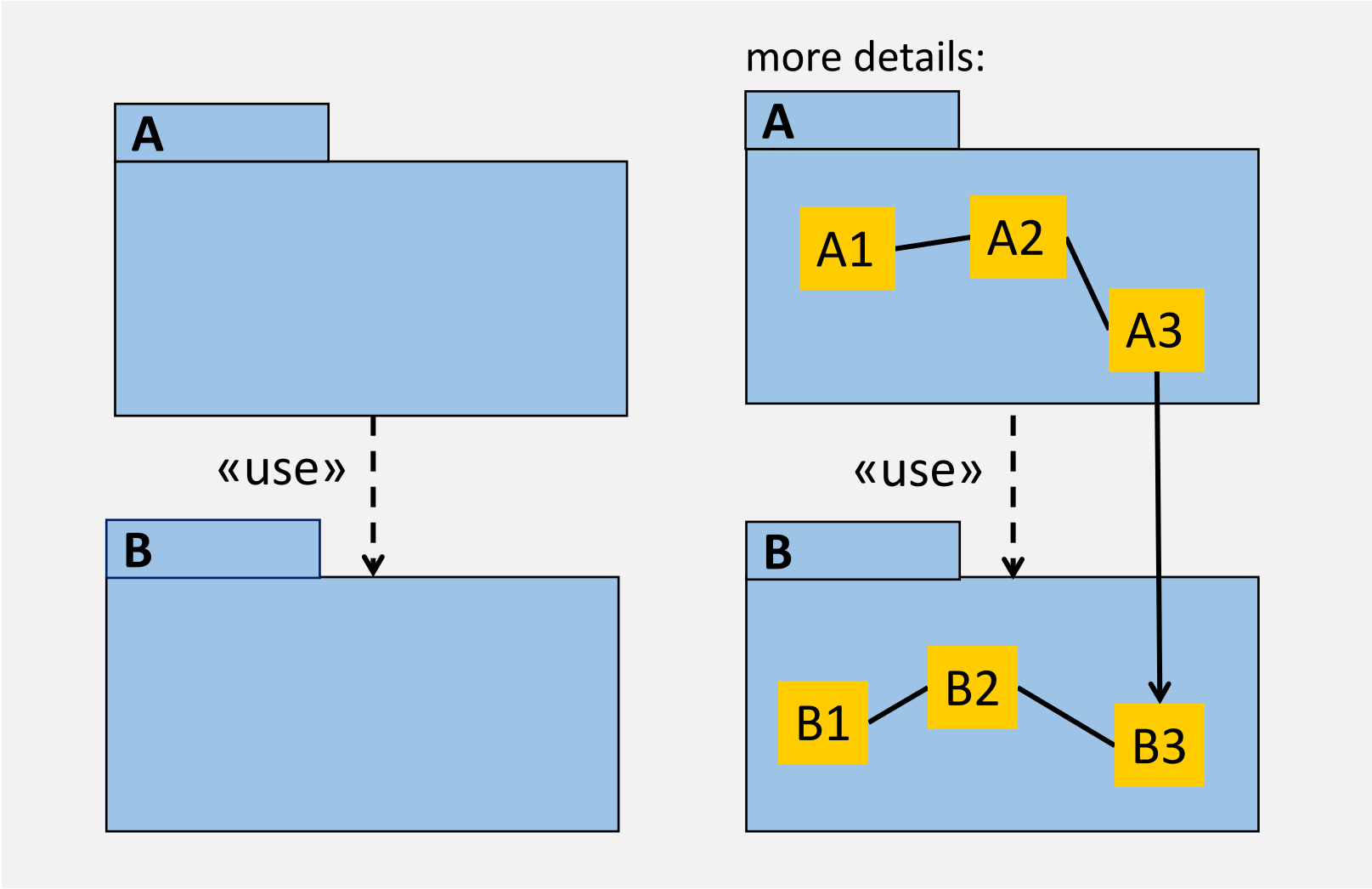


Architektonische Eigenschaft	Bewertung
Partitionierungstyp	Technisch
Anzahl der Quanten	1
Bereitstellbarkeit	★
Elastizität	★
Entwicklungsfähigkeit	★
Fehlertoleranz	★
Modularität	★
Gesamtkosten	★★★★★
Performance	★★
Verlässlichkeit	★★★
Skalierbarkeit	★
Einfachheit	★★★★★
Testbarkeit	★★

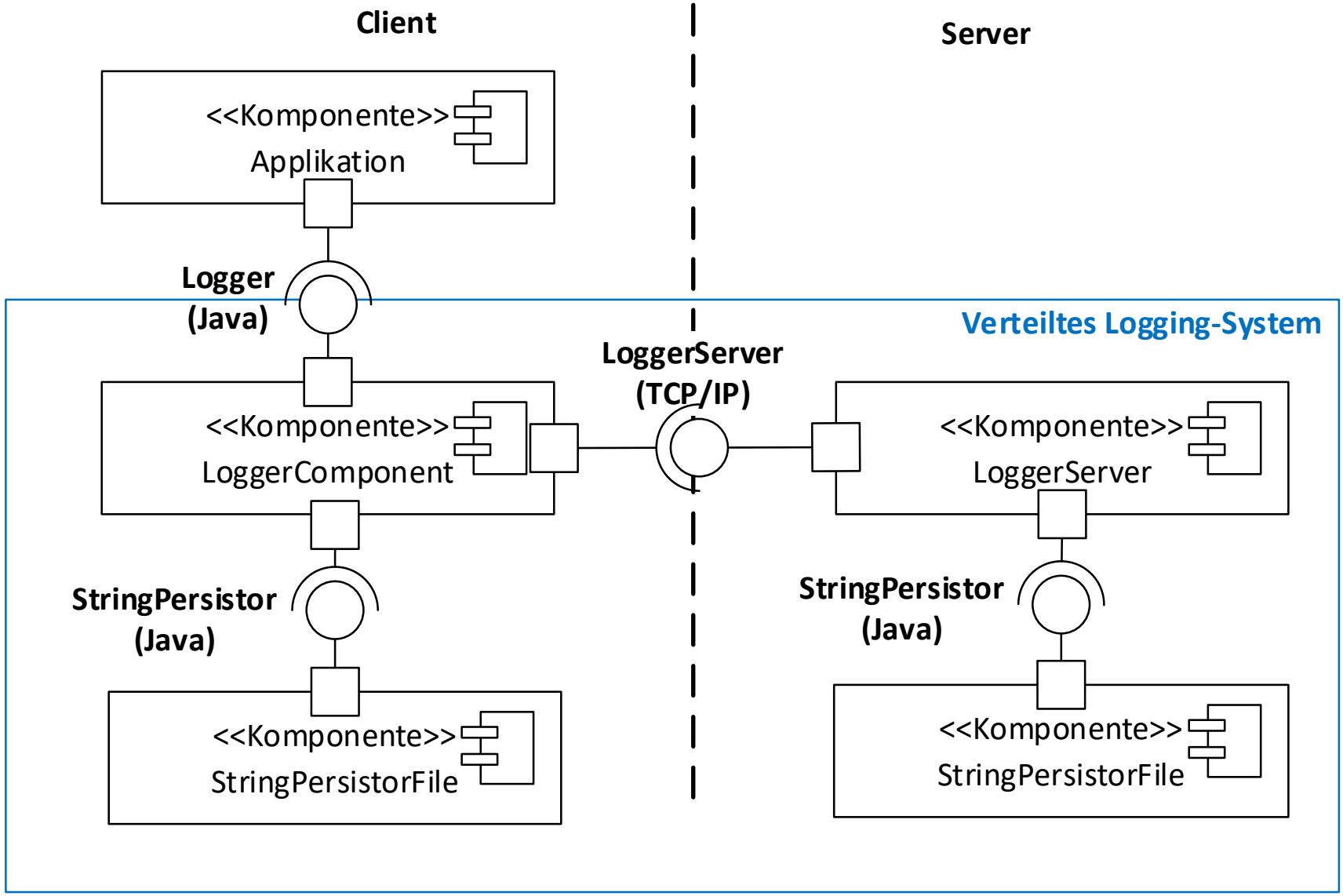
- Einfacher und kostengünstiger Architekturstil.
- Auslieferung als eine Einheit.
- Verteilung des Business-Layers i.d.R. nicht (einfach) möglich -> Skalierung nur innerhalb eines Systems.

Quelle: Handbuch modernder Softwarearchitektur

Schichten in UML



Modularität im Logger?



Zusammenfassung

- Modul: in sich abgeschlossener Teil des gesamten Programmcodes.
- Modulkonzept 1972 David Parnas.
- Kopplung und Kohäsion optimieren!
- Entwurfskriterien: Zerlegbarkeit / Kombinierbarkeit / Verständlichkeit / Stetigkeit.
- Entwurfsprinzipien: lose Kopplung / starke Kohäsion / Information Hiding / wenige & explizite Schnittstellen.
- Entwurfsvorgehen: anspruchsvolle Aufgabe.
- Schichtenarchitektur: Architekturstil basierend auf dem Schichtenkonzept.
- Eigenschaften der Schichtenarchitektur: Günstig, verständlich, aber wenig flexibel.

Fragen?

Literatur und Quellen

- Modulare Software Architektur, Herbert Dowalil, 2020, Carl Hanser Verlag.
- Handbuch moderner Softwarearchitektur, Mark Richards und Neal Ford, 2021, O'Reilly / dpunkt.verlang GmbH.
- Grundlagen des modularen Softwareentwurfs, Herbert Dowalil, 2018, Carl Hanser Verlag.
- Modularisierung mit Java 9: Grundlagen und Techniken für langlebige Softwarearchitekturen von Guido Oelmann, 2017, dpunkt.verlag GmbH.
- Object-Oriented Software Construction (second edition) von Bertrand Meyer, 1997, Prentice Hall.