Teil II
Objektorientierte Programmierung (OOP)
20. Objektnetze

Prof. Dr. rer. nat. Uwe Aßmann
Institut für Software- und Multimediatechnik
Lehrstuhl Softwaretechnologie
Fakultät für Informatik
Technische Universität Dresden
Version 19-0.2, 08.04.19

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

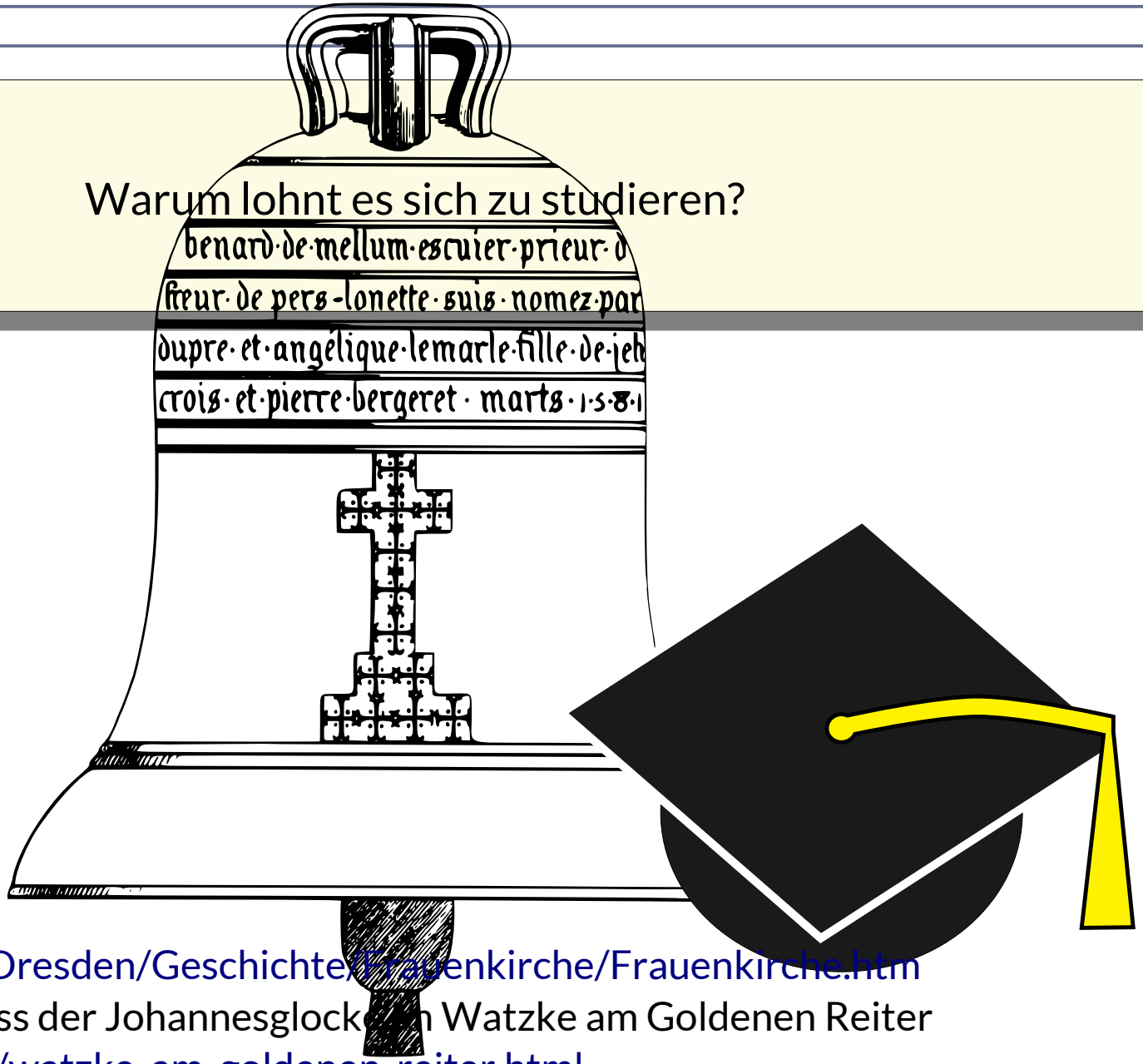
Obligatorische Literatur

- Java language spec
  https://cr.openjdk.java.net/~iris/se/12/latestSpec/java-se-12-annex-3.html
- JDK Tutorial für J2SE oder J2EE, https://docs.oracle.com/javase/tutorial/

© Prof. U. Aßmann

Das Ziel des Studiums: Als Meister große Dinge erschaffen können

Warum lohnt es sich zu studieren?

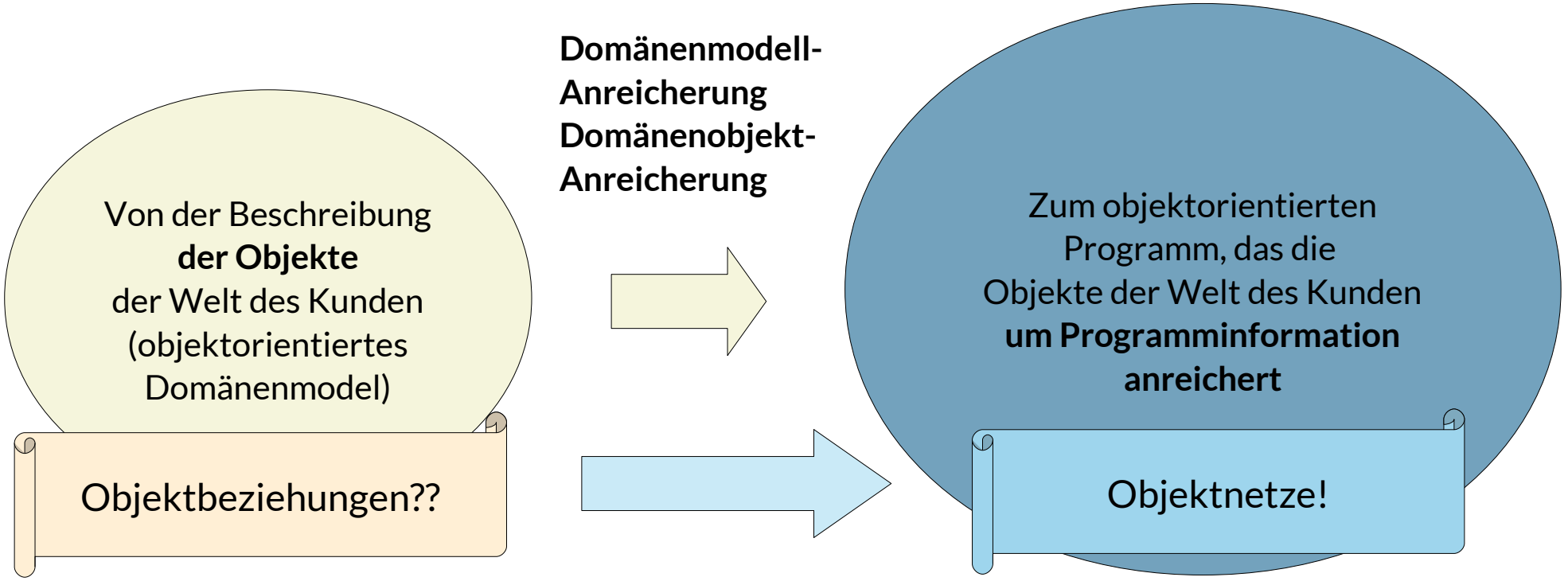


http://www.kprdd.de/Dresden/Geschichte/Frauenkirche/Frauenkirche.htm
Fehltönender Probeguss der Johannesglocke von Watzke am Goldenen Reiter
http://www.watzke.de/watzke_am_goldenen_reiter.html
Geschichte: http://www.md-pro.de/depot/mitteilungen/mit0401.pdf

© Prof. U. Aßmann

Die zentralen Fragen des objektorientierten Ansatzes

Wie kommen wir vom Problem des Kunden zum Programm (oder Produkt)?



Anreicherung/Verfettung: Anreicherung durch technische Programminformation
„object fattening“: Anreicherung von Objekten des Domänenmodells

© Prof. U. Aßmann

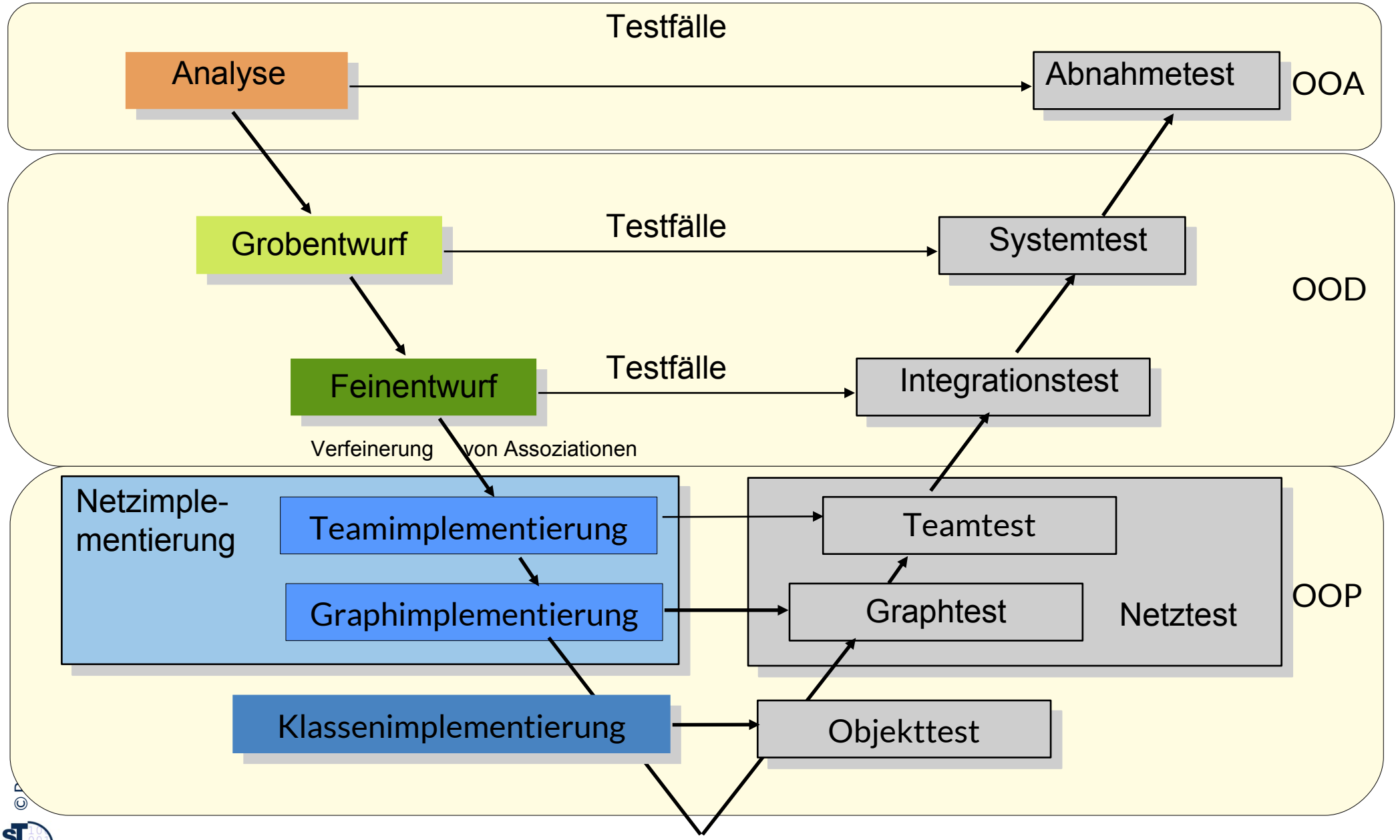
Ziel von Teil II der Vorlesung

- Wie kann man Objektnetze (Graphen, Dags, Bäumen, Listen) abstrakt und ausdrucksstark beschreiben?
- Wie kann man deren Test vereinfachen? (Das Programmieren von Objektnetzen ist sehr fehleranfällig)
- Wie kann man den Aufbau von Objektnetzen durch Verfeinerung von Assoziationen konkret auf den Rechner zuschneiden?
  - Graphen, Iteratormethoden, Iteratoren, und Streams
  - Große Objekte (Bobs) mit internen Netzen
  - Endo- und Exoassoziationen
  - Wie man Graphen erweitert

© Prof. U. Aßmann

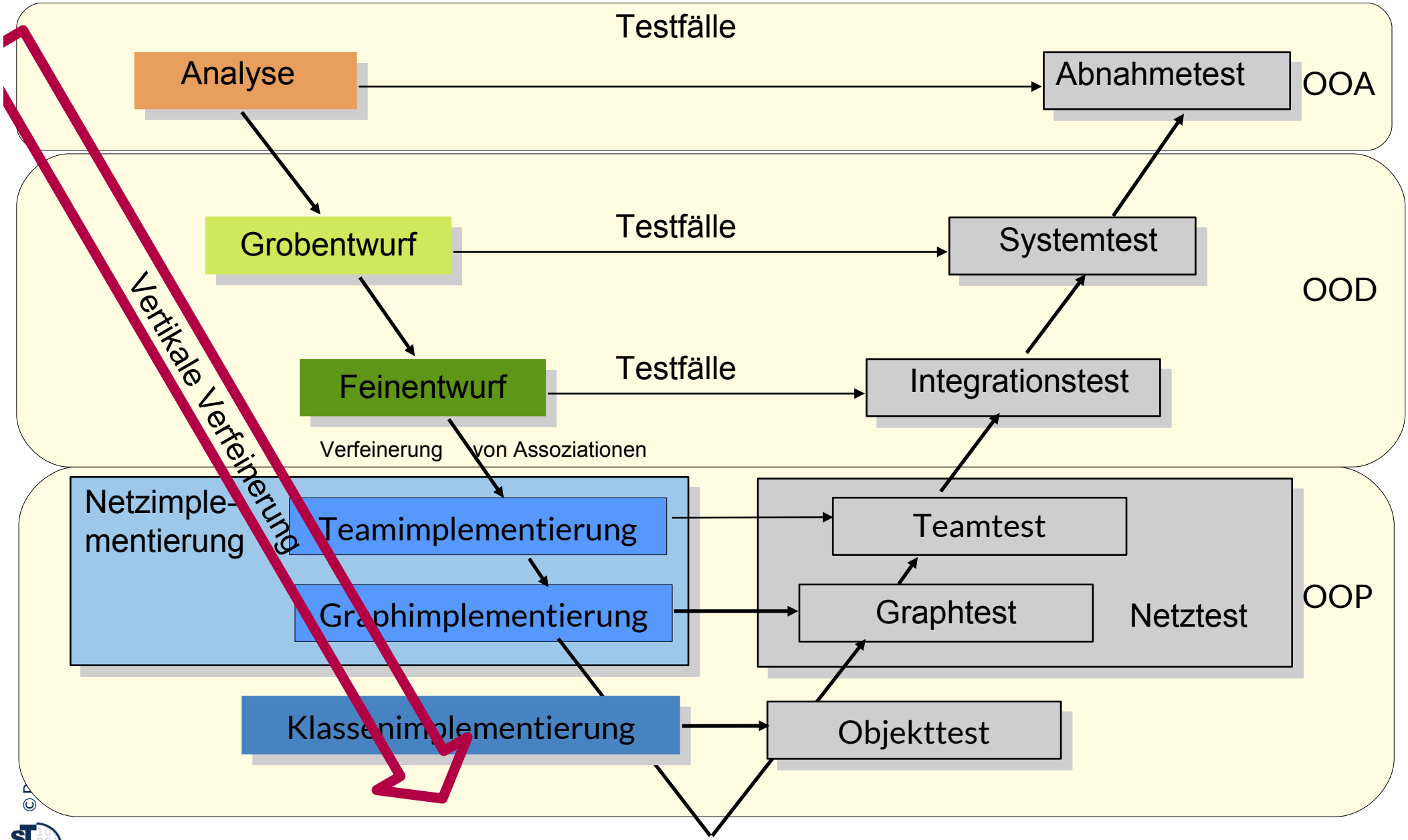
Q4: Softwareentwicklung im V-Modell

[Boehm 1979]



Q4: Softwareentwicklung im V-Modell

[Boehm 1979]



Grundlegende Begriffe des Modellierens

- Ein **Sprachkonstrukt (Sprachelement)** bezeichnet ein Konstrukt bzw. Konzept einer Sprache.
- Ein **Programm-/Modellelement** bezeichnet ein Element eines Programms/ Modells.
- Ein **Fragment (Snippet)** eines Programms oder Modells ist ein partieller Satz der Sprache, d.h. ein Netz aus Programm- oder Modellelementen.
- Ein **generisches Fragment (generisches Snippet, Fragmentformular)** eines Programms oder Modells ist ein partieller Satz der Sprache mit Platzhaltern ("Lücken").
- Eine **Fragmentgruppe** ist eine Menge von (ggf.generischen) Fragmenten eines Programs oder Modells.
- Eine **Fragmentkomponente** ist eine Fragmentgruppe zur Wiederverwendung.
- **Abstraktion** ist das Vernachlässigen von Details
- **Detaillierung (Anreicherung)** ist das Anfügen von Details

© Prof. U. Aßmann

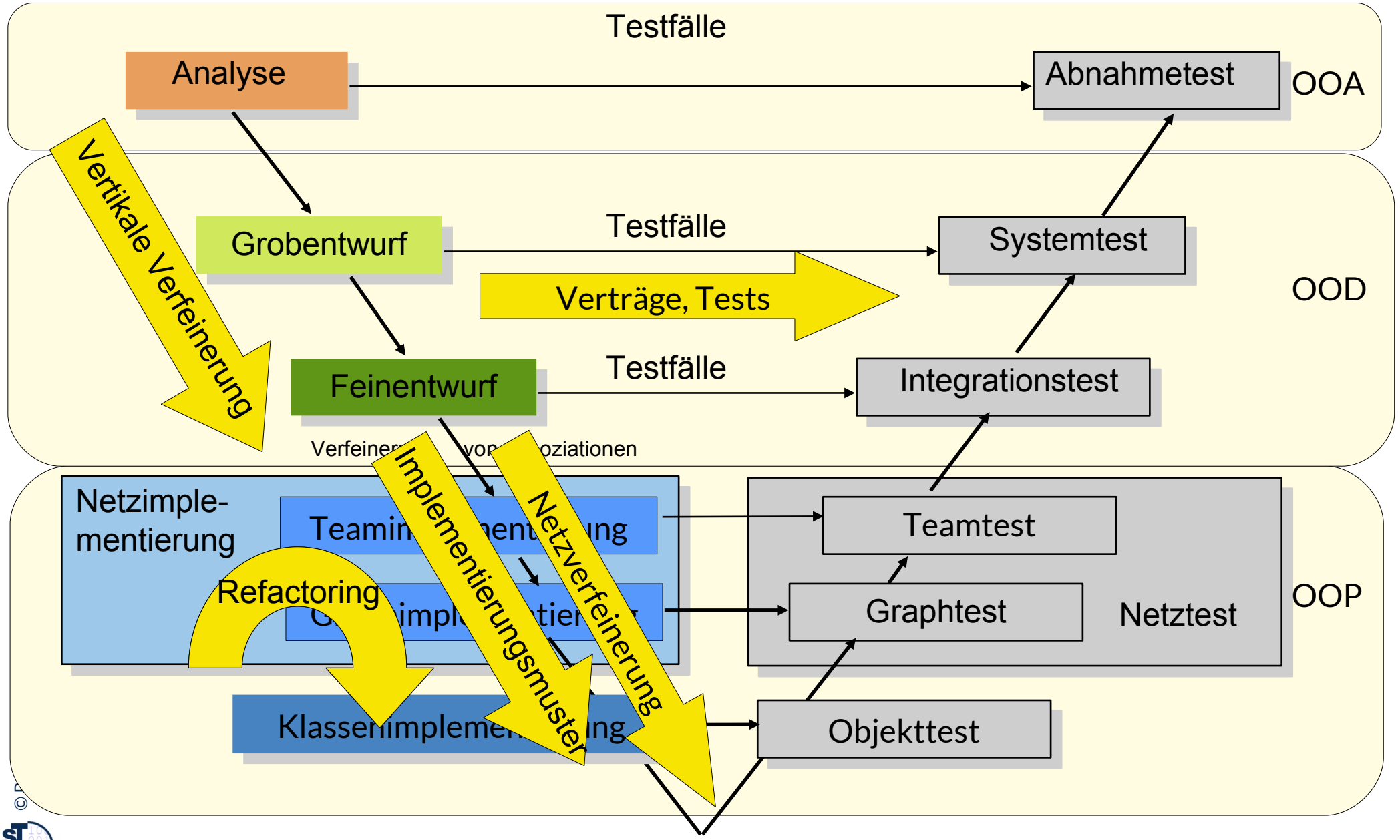
Grundlegende Begriffe des Software-Entwurfs

- **Horizontale Verfeinerungsoperationen** ersetzen Fragmente auf gleicher Sprachebene:
  - **Detaillierung (Anreicherung)**: Ergänzung von Einzelheiten
    - **Vervollständigung** (**Elaboration**) von Fragmenten zu Sätzen der Modellierungssprache
  - **Erhöhung Zuverlässigkeit**: Ergänzung von qualitätssteigernden Fragmenten (Typisierung, Verträge, Tests)
  - Einführung des **Architektur-Aspektes** des Systems
    - **Strukturierung** und **Restrukturierung**
    - **Refaktorisierung (Refactoring)** ist semantische Restrukturierung
- **Vertikale Verfeinerungsoperationen** (von abstrakter Ebene zu konkreter Ebene):
  - **Abflachen von Fragmenten** (Flachklopfen, Realisierung, lowering):
  - **Realisierung** ersetzt ausdrucksstarke Konstrukte durch weniger ausdrucksstarke, implementierungsnähere

© Prof. U. Aßmann

Q4: Softwareentwicklung im V-Modell

[Boehm 1979]



Verfeinerung: Schritte von UML zur Implementierung

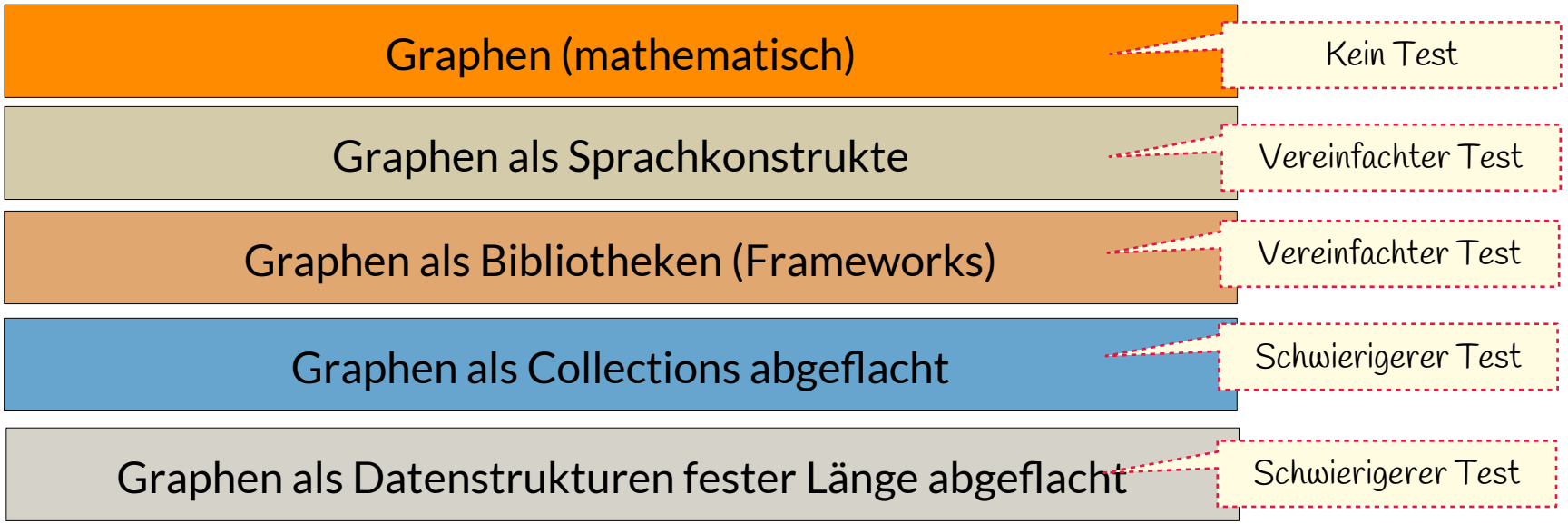
Ein ***Implementierungsmuster*** (*workaround, Idiom)* beschreibt die vertikale Verfeinerung eines Sprachkonstruktes einer Modellierungs- oder Spezifikationssprache durch ein Fragment einer Implementierungssprache

- Verfeinerung von Sprachkonstrukten (Realisierung, Abflachen, lowering)
  - Netzentwurf
  - Implementierung von Methoden (von Statecharts und Aktivitätsdiagrammen)
  - Datenverfeinerung
  - Kontrollverfeinerung
  - Syntaktische Verfeinerung
  - Semantische Verfeinerung

© Prof. U. Aßmann

Repräsentation von flexiblen Objektnetzen als Datenstrukturen (Netzverfeinerung)

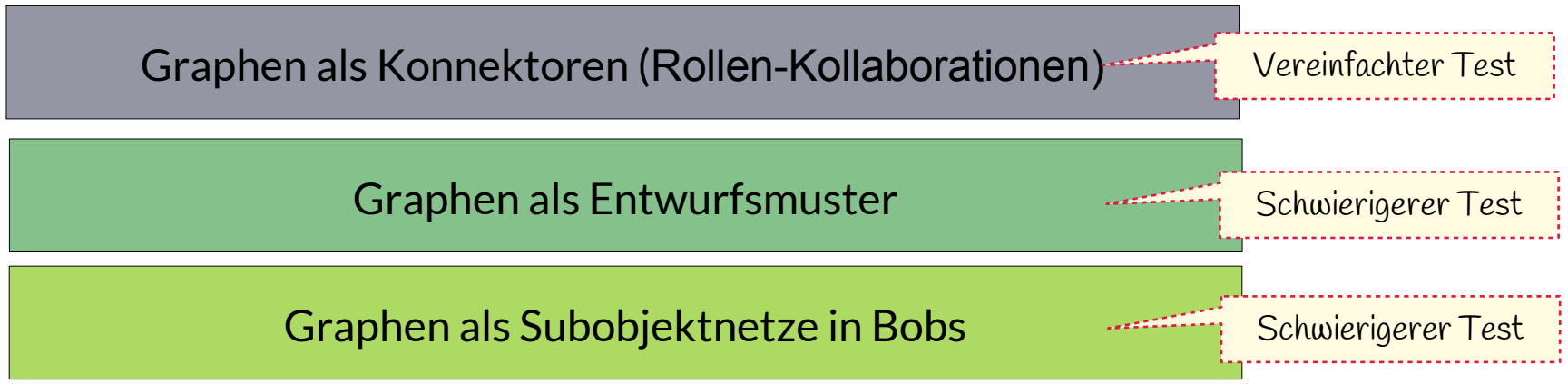
- Auf Ebene der Anforderungen werden flexible Objektnetze als math. Graphen dargestellt
- Im Grobentwurf und Feinentwurf werden sie repräsentiert (verfeinert zu)
  - durch Graphen als **Sprachkonstrukte** (für Sprachen, die das eingebaut haben)
    - Rapid Application Development (RAD)
  - durch **Graphen** aus Java-Graph-Bibliotheken
  - durch **Implementierungsmuster** wie **Collections**, nach dem Abflachen/Flachklopfen von bidirektionalen Assoziationen in gerichteten Links
  - durch **maschinennahe Implementierungsmuster** wie **Datenstrukturen** fester Länge (Arrays, Matrizen) (speicher-bewusstes Programmieren)



© Prof. U. Aßmann

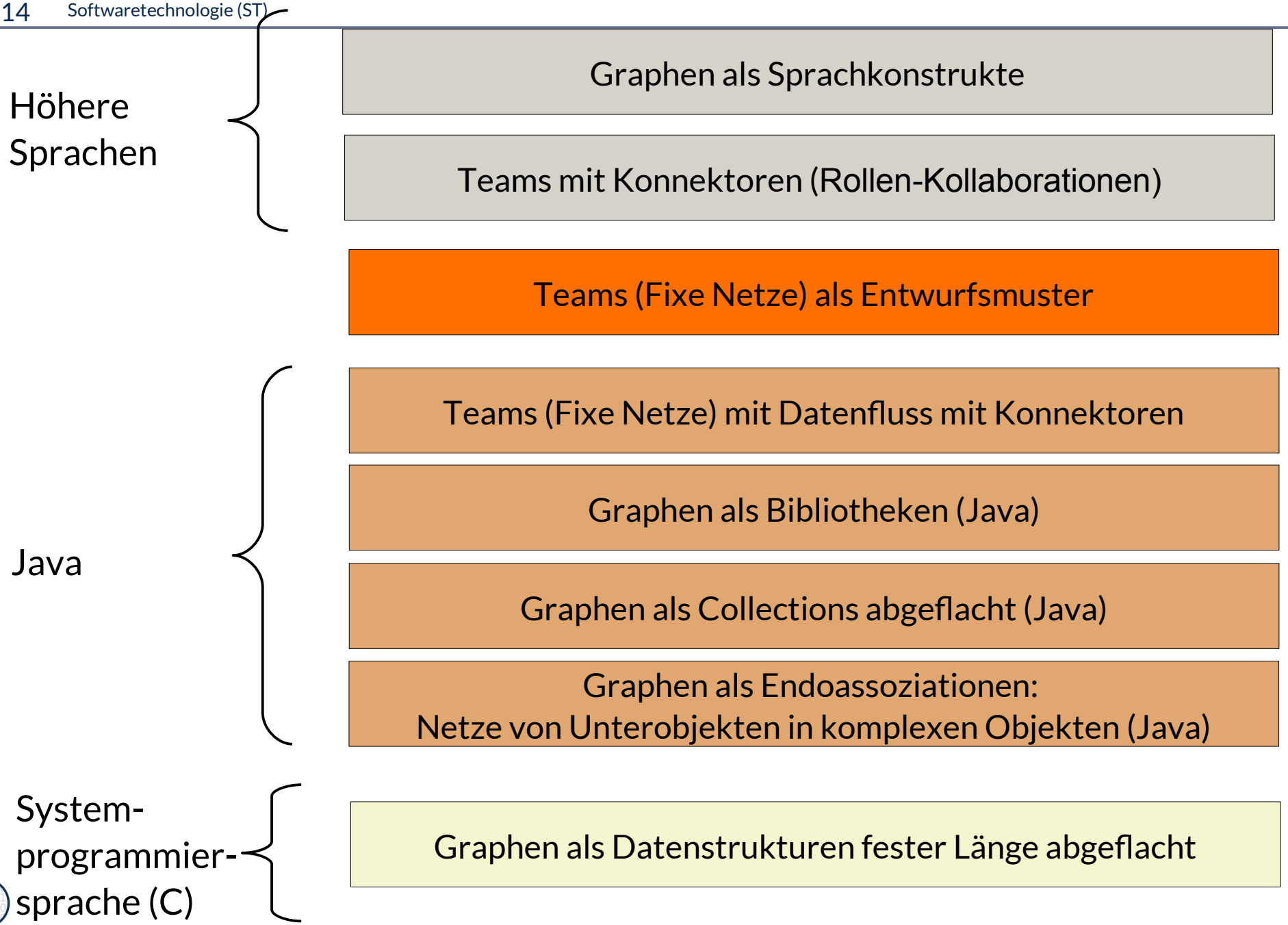
Netzverfeinerung von fixen Netzen als Konnektoren (Kollaborationen)

- Fixe Netze mit statisch festem n, m (z.B. 1:1-Assoziationen) können durch **Konnektoren** (Kollaborationen, Teams) verfeinert werden
  - Assoziationen tragen Rollentypen als Assoziationsenden
- Einfache Objektnetze werden durch **Entwurfsmuster** beschrieben
  - Listen: Decorator, Chain
  - Bäume: Composite
  - Dags, Graphen: Modifiziertes Composite
  - Konnektoren: Observer, Mediator, Visitor
- Große Objekte (Bobs) haben oft interne Subobjektnetze
  - Subobjektnetze beruhen auf Endo-Assoziationen



© Prof. U. Aßmann

Überblick Teil II



© Prof. U. Aßmann

Verständnisfragen

- Wieso kann man den Klang einer Glocke nicht testen?
- Warum ist das Programmieren von Objektnetzen so schwierig?
- Welche Möglichkeiten gibt es fürs Testen, wenn man Objektnetz-Bibliotheken verwendet?
- Warum ist Testautomatisierung für die Programmierung von Objektnetzen so wichtig?

© Prof. U. Aßmann