

# 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, 05.04.19

- ▶ 21) Verfeinern von Assoziationen mit dem Java-2 Collection Framework
- ▶ 22) Netze mit Datenfluss: Iteratoren, Kanäle
- ▶ 23) Teams und Kanäle
- ▶ 24) Graphen in Java
- ▶ 25) Entwurfsmuster



**DRESDEN  
concept**  
Exzellenz aus  
Wissenschaft  
und Kultur

- ▶ 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/>

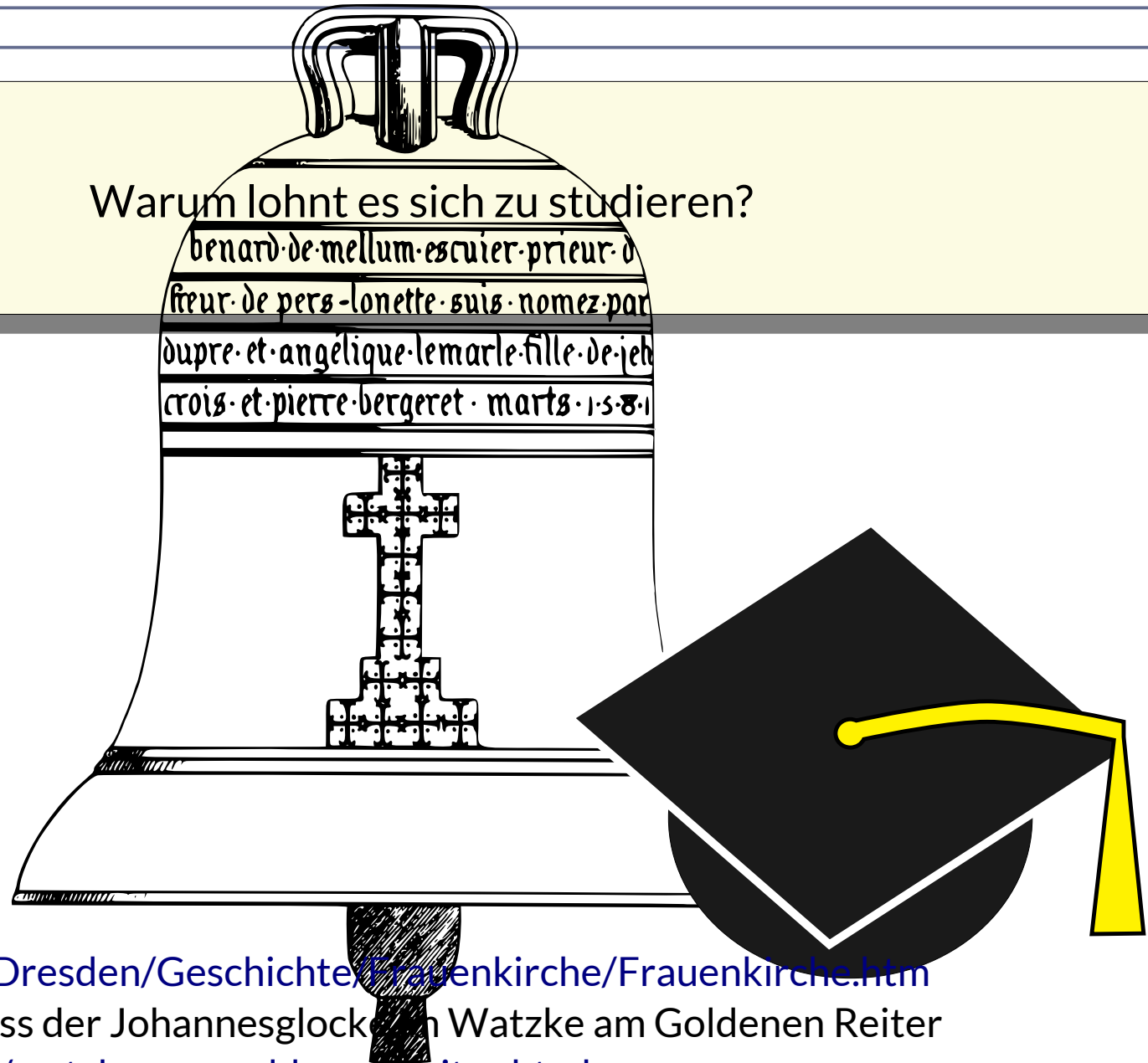
# Das Ziel des Studiums: Als Meister große Dinge erschaffen können

3

Softwaretechnologie (ST)

Warum lohnt es sich zu studieren?

benard·de·mellum·escuier·prieur·d  
feur·de·pers·lonette·suis·nomez·par  
dupre·et·angélique·lemarle·fille·de·jeh  
crois·et·pierre·bergeret·marts·1·5·8·1



<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](http://www.watzke.de/watzke_am_goldenen_reiter.html)

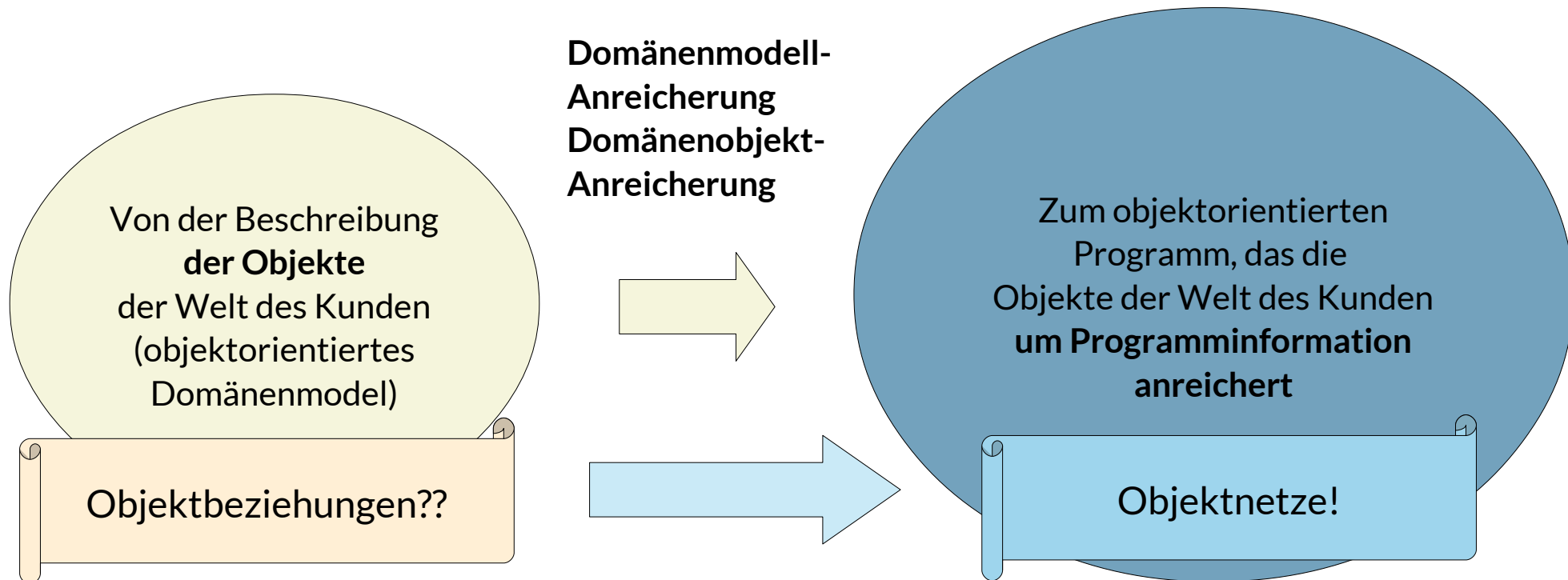
Geschichte: <http://www.md-pro.de/depot/mitteilungen/mit0401.pdf>

# Die zentralen Fragen des objektorientierten Ansatzes

4

Softwaretechnologie (ST)

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



Anreicherung/Verfettung: Anreicherung durch technische Programminformation

„object fattening“: Anreicherung von Objekten des Domänenmodells

# Ziel von Teil II der Vorlesung

5

Softwaretechnologie (ST)

- ▶ 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



OUR GOALS

# Q4: Softwareentwicklung im V-Modell

[Boehm 1979]

6 Softwaretechnologie (ST)

Testfälle

Analyse

Abnahmetest

OOA

Grobentwurf

Testfälle

Systemtest

OOD

Feinentwurf

Testfälle

Integrationstest

Verfeinerung von Assoziationen

Netzimple-  
mentierung

Teamimplementierung

Graphimplementierung

Klassenimplementierung

Teamtest

Graphtest

Objekttest

Netztest

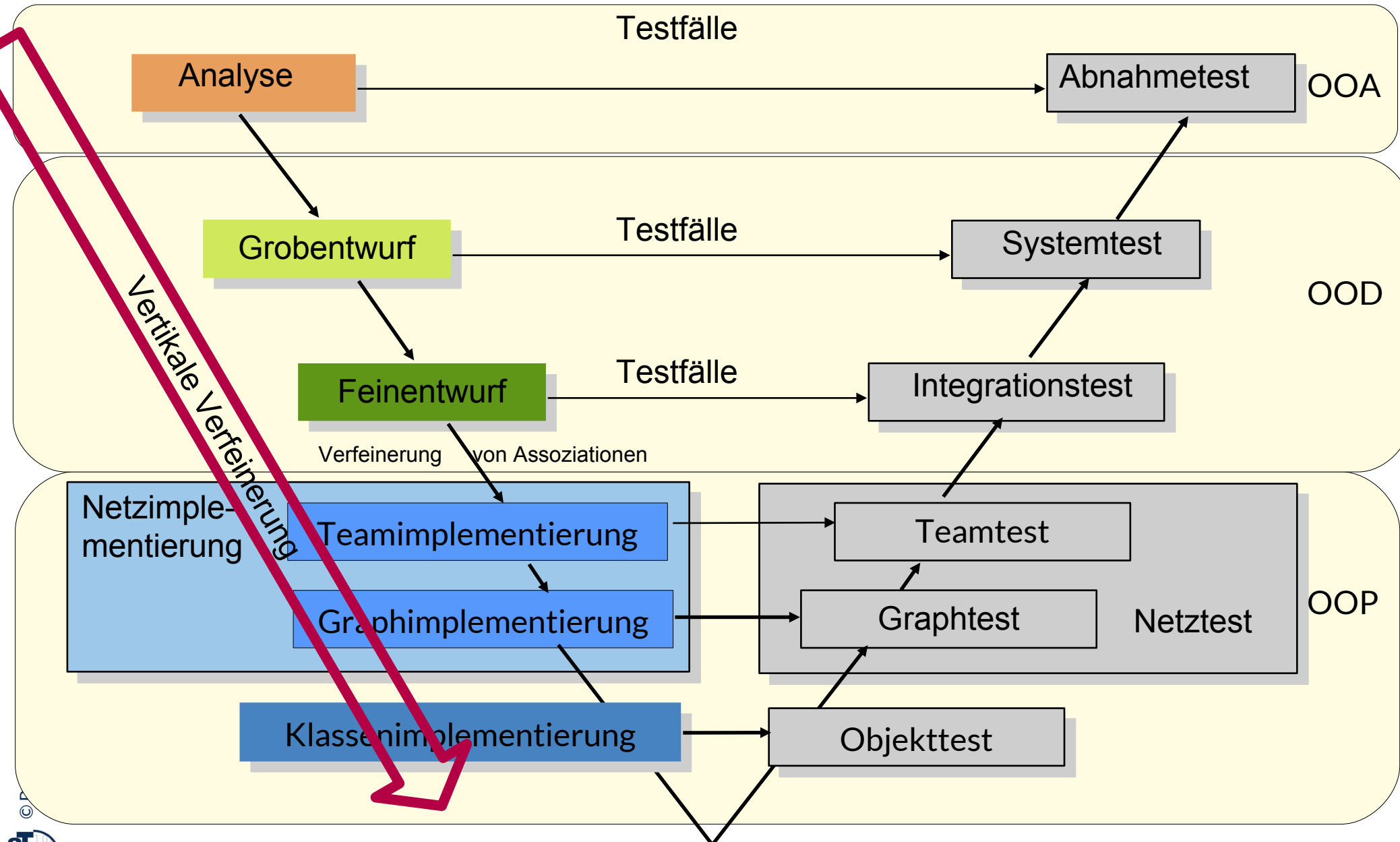
OOP

# Q4: Softwareentwicklung im V-Modell

[Boehm 1979]

7

Softwaretechnologie (ST)



- ▶ 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

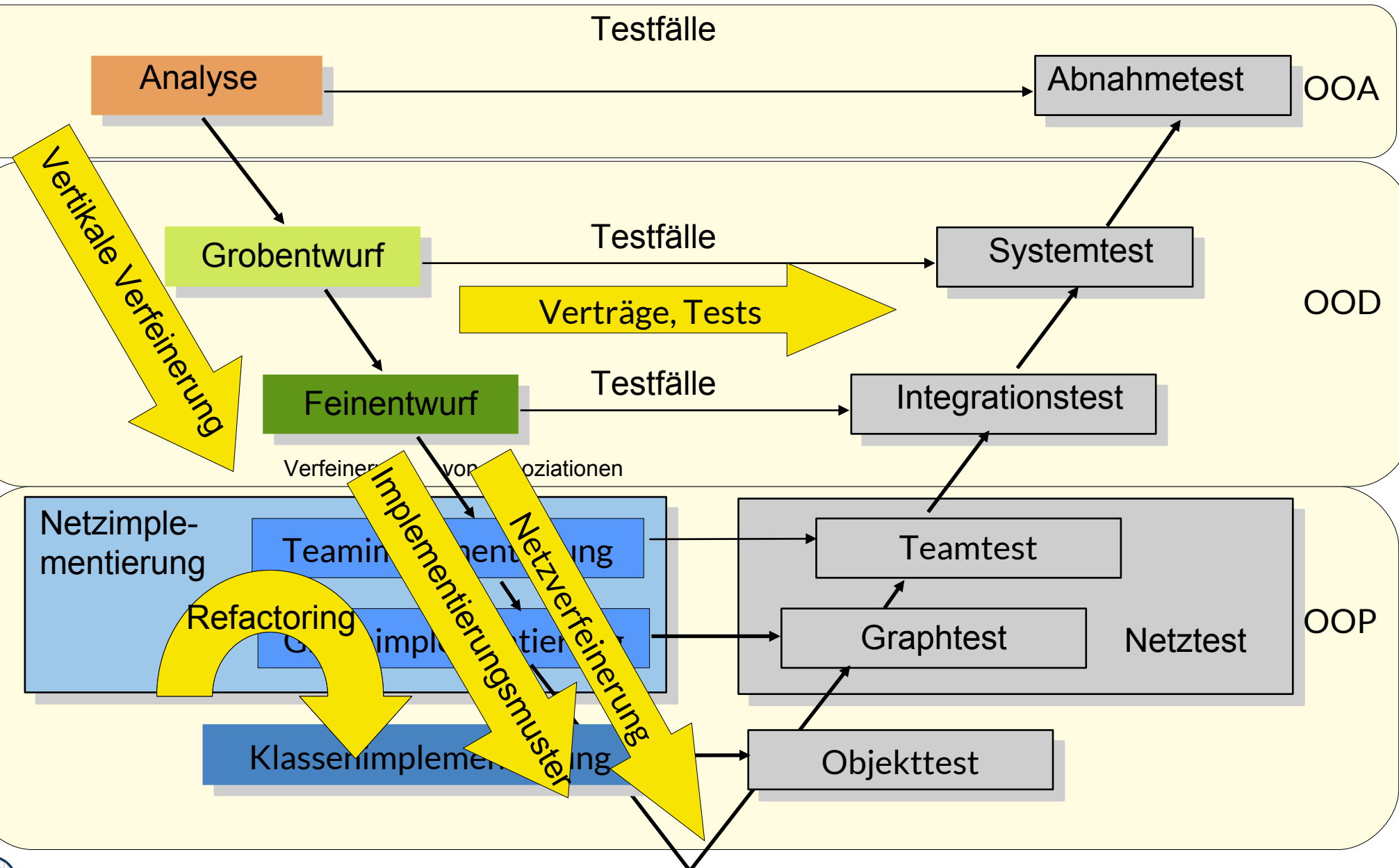


- ▶ **Horizontale Verfeinerungsoperationen** ersetzen Fragmente auf gleicher Sprachebene:
  - **Detaillierung (Anreicherung):** Ergänzung von Einzelheiten
    - **Vervollständigung (Elaboration)** von Fragmenten zu Sätzen der Modellierungssprache
  - **Erhöhung Zuverlässigkeit:** Ergänzung von qualitätssteigernden 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

# Q4: Softwareentwicklung im V-Modell

[Boehm 1979]

10 Softwaretechnologie (ST)



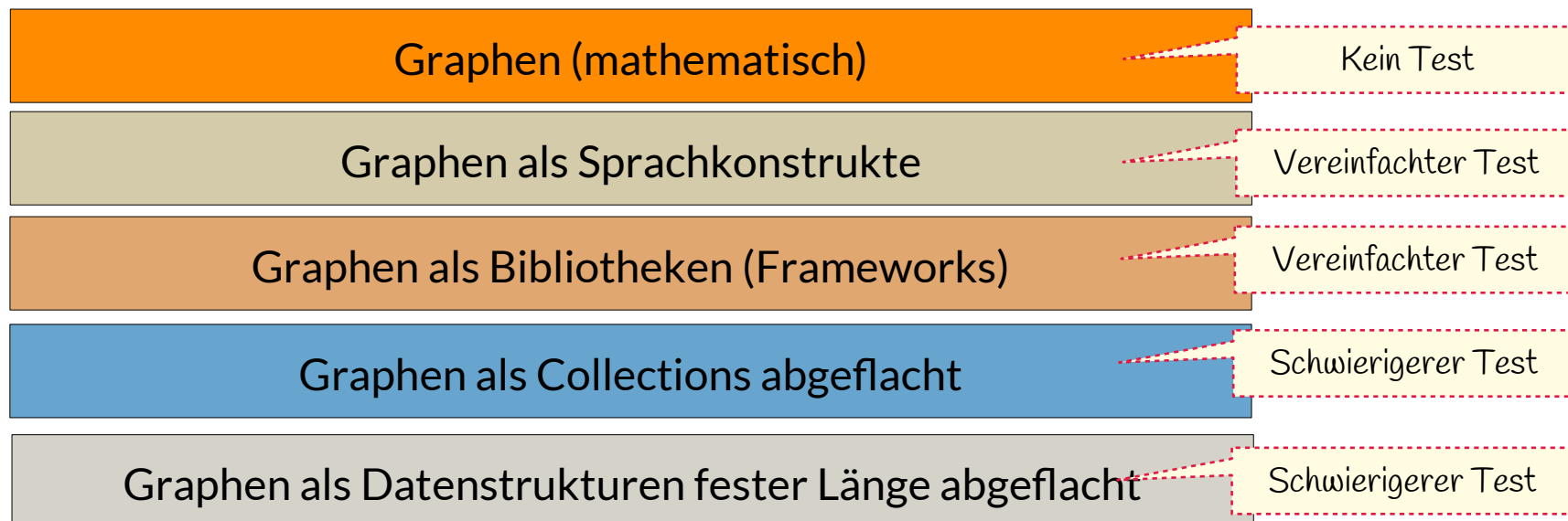
# Verfeinerung: Schritte von UML zur Implementierung

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

# 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)



# Netzverfeinerung von fixen Netzen als Konnektoren (Kollaborationen)

13

Softwaretechnologie (ST)

- ▶ 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

Graphen als Konnektoren (Rollen-Kollaborationen)

Vereinfachter Test

Graphen als Entwurfsmuster

Schwierigerer Test

Graphen als Subobjektnetze in Bobs

Schwierigerer Test

# Überblick Teil II

14 Softwaretechnologie (ST)

Höhere  
Sprachen

Graphen als Sprachkonstrukte

Teams mit Konnektoren (Rollen-Kollaborationen)

Teams (Fixe Netze) als Entwurfsmuster

Java

Teams (Fixe Netze) mit Datenfluss mit Konnektoren

Graphen als Bibliotheken (Java)

Graphen als Collections abgeflacht (Java)

Graphen als Endoassoziationen:  
Netze von Unterobjekten in komplexen Objekten (Java)

System-  
programmier-  
sprache (C)

Graphen als Datenstrukturen fester Länge abgeflacht

- ▶ 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?

## Teil II

# Objektorientierte Programmierung (OOP)

### 20. Objektnetze

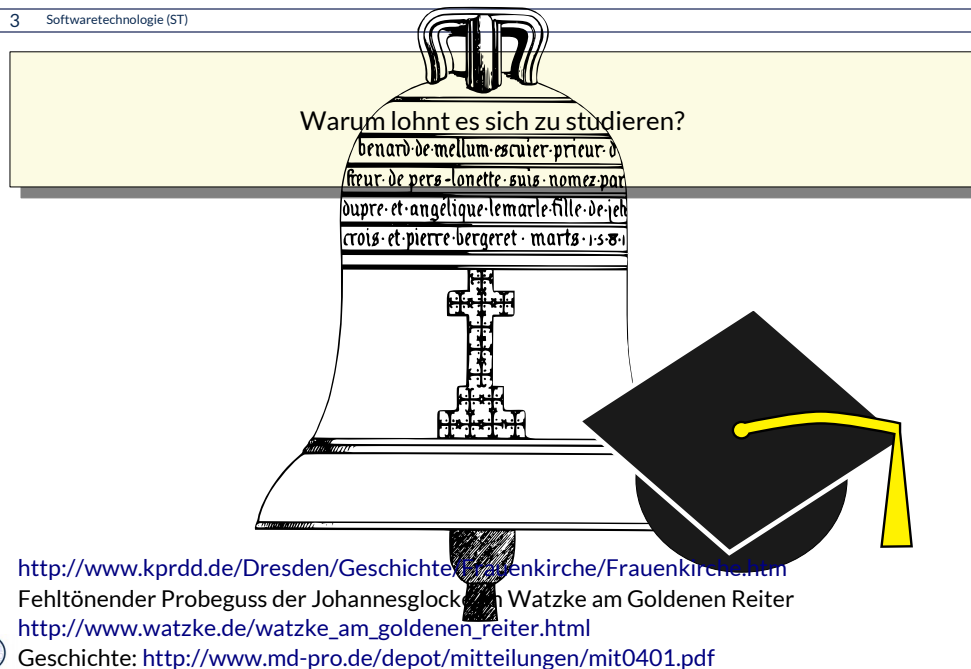
Prof. Dr. rer. nat. Uwe Aßmann  
Institut für Software- und  
Multimediatechnik  
Lehrstuhl Softwaretechnologie  
Fakultät für Informatik  
Technische Universität Dresden  
Version 19-0.2, 05.04.19

- ▶ 21) Verfeinern von Assoziationen mit dem Java-2 Collection Framework
- ▶ 22) Netze mit Datenfluss: Iteratoren, Kanäle
- ▶ 23) Teams und Kanäle
- ▶ 24) Graphen in Java
- ▶ 25) Entwurfsmuster





- ▶ 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/>



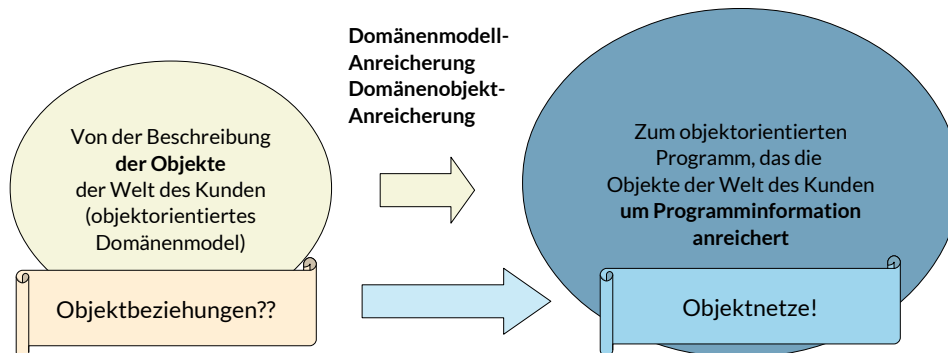
## Ziele des Studiums:

- Das Gefühl, dass man jedes Buch der Welt hernehmen und in 6 Wochen lernen und verstehen kann
- Meisterschaft: ein Software-Problem selbständig lösen

Allerdings kann man Glockengießen nicht testen. Man braucht viel Erfahrung, um zu “fühlen”, wie eine Glocke gelingt. Oft, wie bei der Johannes-Glocke der Frauenkirche, mißlingt es sogar.

Daher muss man sich zum Meister seines Fachs weiterentwickeln, wenn man Glockengießen will.

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



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

Das Programmieren von Objektnetzen ist äußerst schwierig.

Die Objektorientierung startet von den Beziehungen der Objekte in der Welt des Kunden aus – und übersetzt und verfeinert diese in Objektnetze im Programm. Allerdings ist dieser Schritt äußerst fehleranfällig und führt oft zu tagelangem Suchen von Fehlern (teuer, frustrierend).

Hier liegt die Stärke einer modernen, objektorientierten Sprache mit gutem Typsystem:

- “dangling pointers” (Luft-Zeiger) werden durch Typisierung vermieden oder durch Tests, die bei Frameworks für Netze mitgeliefert werden, schnell gefunden
- Frameworks verhindern dangling pointers von vorne herein durch sichere Konstruktion des Netzes (Build-Entwurfsmuster wie Fabrikmethoden, Fabriken)

## Ziel von Teil II der Vorlesung

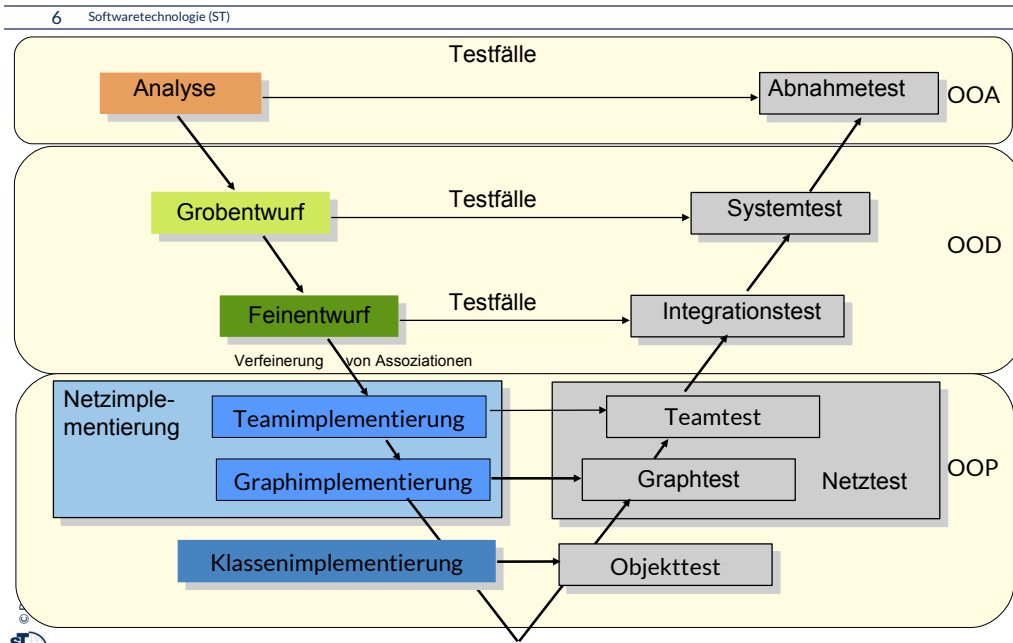
- ▶ Wie kann man Objektnetze (Graphen, Dags, Bäumen, Listen) abstrakt und ausdrucksstark beschreiben?
- ▶ Wie kann man deren Test vereinfachen? (Das Programmieren von Objektnetzen ist sehr fehleranfällig)
- ▶ Wie kann man den Aufbau von Objektnetzen durch Verfeinerung von Assoziationen konkret auf den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



OUR GOALS

## Q4: Softwareentwicklung im V-Modell

[Boehm 1979]



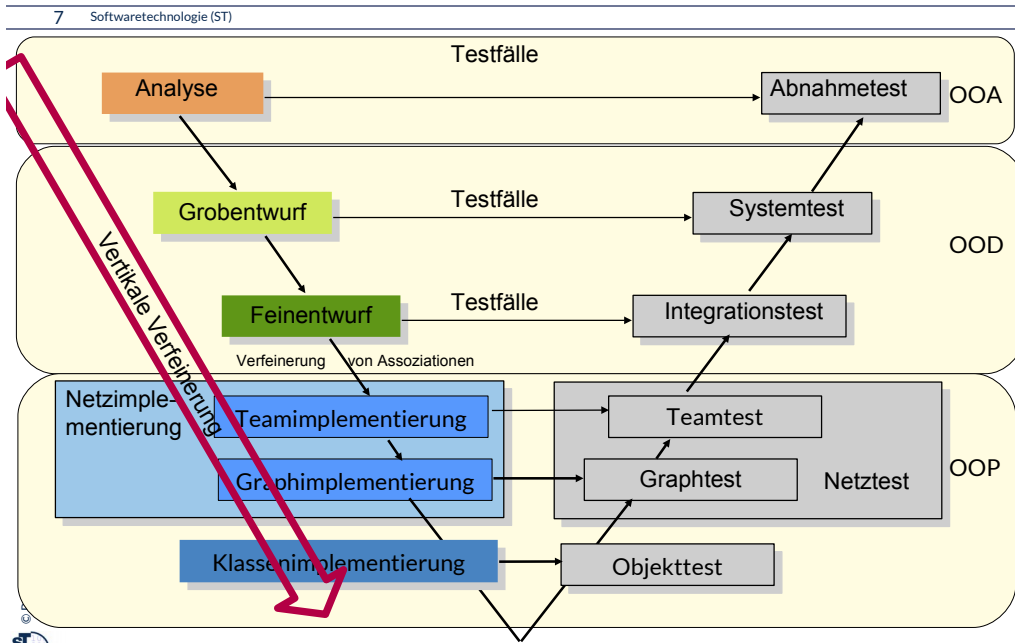
Quelle fuer das Boehm-Zitat: [Hesse/Merbeth/Fröhlich S. 37]

Netzimplementierung und Teamimplementierung bilden zwei separate Phasen der Implementierung, jeweils mit eigener Testsuite.

Die Testsuite kann manchmal vom Framework wiederverwendet werden, oder mit seiner Hilfe schnell implementiert werden.

## Q4: Softwareentwicklung im V-Modell

[Boehm 1979]



Quelle fuer das Boehm-Zitat: [Hesse/Merbeth/Frölich S. 37]

Netzimplementierung und Teamimplementierung bilden zwei separate Phasen der Implementierung, jeweils mit eigener Testsuite.

- ▶ 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 Programms oder Modells.
- ▶ Eine **Fragmentkomponente** ist eine Fragmentgruppe zur Wiederverwendung.
- **Abstraktion** ist das Vernachlässigen von Details
- **Detaillierung (Anreicherung)** ist das Anfügen von Details

► **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):

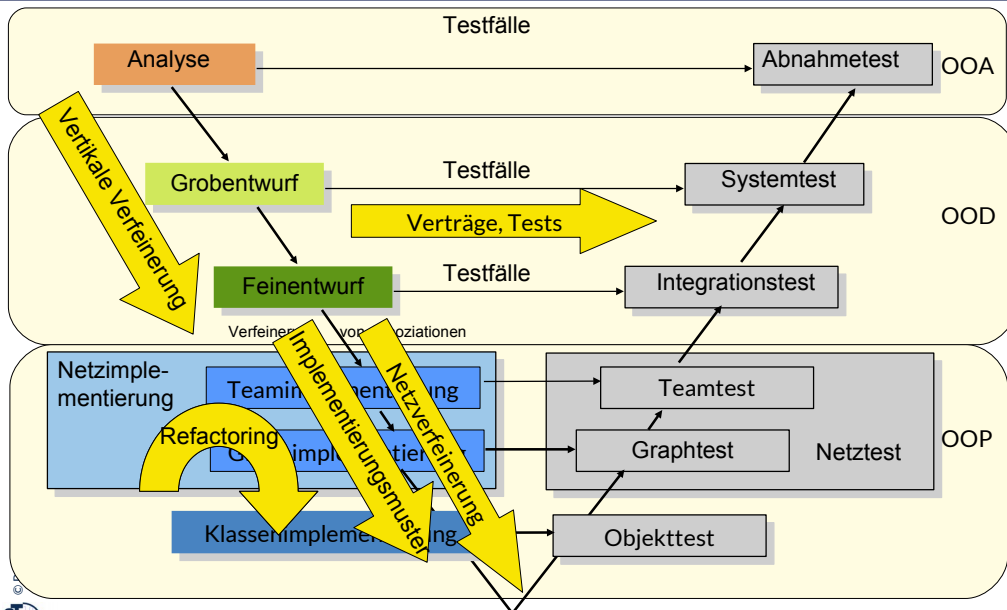
- **Abflachen von Fragmenten** (Flachklopfen, Realisierung, lowering):
- **Realisierung** ersetzt ausdrucksstarke Konstrukte durch weniger ausdrucksstarke, implementierungsnähere



## Q4: Softwareentwicklung im V-Modell

[Boehm 1979]

10 Softwaretechnologie (ST)



Die stufenweise Verfeinerung von dem Pflichtenheft, dem Ergebnis der Analyse aus, hin zur Implementierung beinhaltet Netzverfeinerung und Teamverfeinerung.

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



• Verfeinerung von Sprachkonstrukten (Realisierung, Abflachen, lowering):  
Viele der Sprachkonstrukte aus UML haben kein direktes Java-Äquivalent und müssen zu Java-Konstrukten verfeinert werden.

- Netzentwurf: Verfeinerung von Assoziationen
- Implementierung von Methoden (von Statecharts und Aktivitätsdiagrammen)

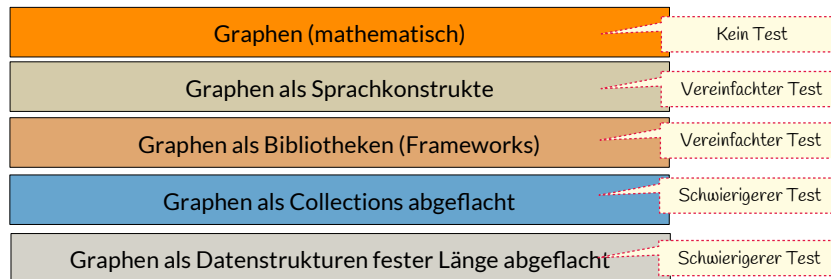
• Verfeinerungsschritte vom Analysemodell zum Entwurfsmodell

- **Syntaktische Verfeinerung** ersetzt nur die Syntax
  - **Datenverfeinerung** verfeinert Datenstrukturen
  - **Kontrollverfeinerung** verfeinert Kontrollstrukturen (Verhalten)
- **Semantische Verfeinerung** erhält die Semantik des Modells oder Programms
  - Dazu ist ein mathematischer Beweis nötig

## Repräsentation von flexiblen Objektnetzen als Datenstrukturen (Netzverfeinerung)

12 Softwaretechnologie (ST)

- ▶ 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)



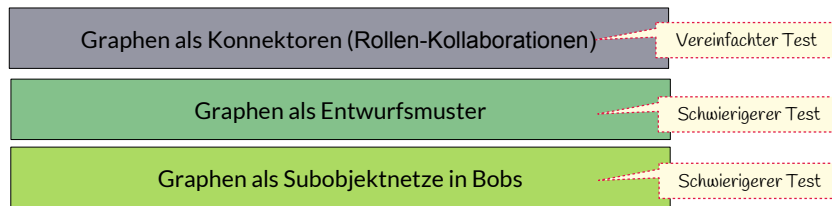
Wohl dem, der eine gute Testsuite für Objektnetze hat!

Jgrapht, unser Beispiel-Framework für Graphen, hat Generatoren für Graphen, die die Konstruktion von Testsuiten unterstützen.

## Netzverfeinerung von fixen Netzen als Konnektoren (Kollaborationen)

13 Softwaretechnologie (ST)

- ▶ 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



Wohl dem, der eine gute Testsuite für Objektnetze hat!

## Überblick Teil II

14 Softwaretechnologie (ST)

Höhere  
Sprachen

Graphen als Sprachkonstrukte

Teams mit Konnektoren (Rollen-Kollaborationen)

Teams (Fixe Netze) als Entwurfsmuster

Java

Teams (Fixe Netze) mit Datenfluss mit Konnektoren

Graphen als Bibliotheken (Java)

Graphen als Collections abgeflacht (Java)

Graphen als Endoassoziationen:  
Netze von Unterobjekten in komplexen Objekten (Java)

System-  
programmier-  
sprache (C)

Graphen als Datenstrukturen fester Länge abgeflacht

## 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?