Teil II
Objektorientierte Programmierung (OOP) mit Objektnetzen
20. Software-Entwicklung im V-Modell

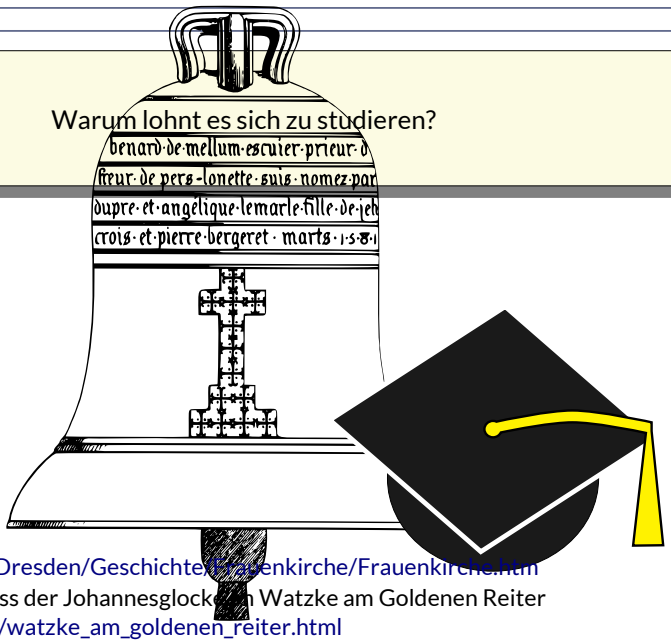
Prof. Dr. rer. nat. Uwe Aßmann
Institut für Software- und Multimediatechnik
Lehrstuhl Softwaretechnologie
Fakultät für Informatik
Technische Universität Dresden
Version 22-0.1, 30.04.22

Wichtige Literatur

- Java language spec
  - https://de.wikipedia.org/wiki/Java_Language_Specification
  - https://docs.oracle.com/javase/specs/jls/se8/html/index.html
- Diffs in Version 12
  - 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: Schöne und große Dinge erschaffen können

Warum lohnt es sich zu studieren?



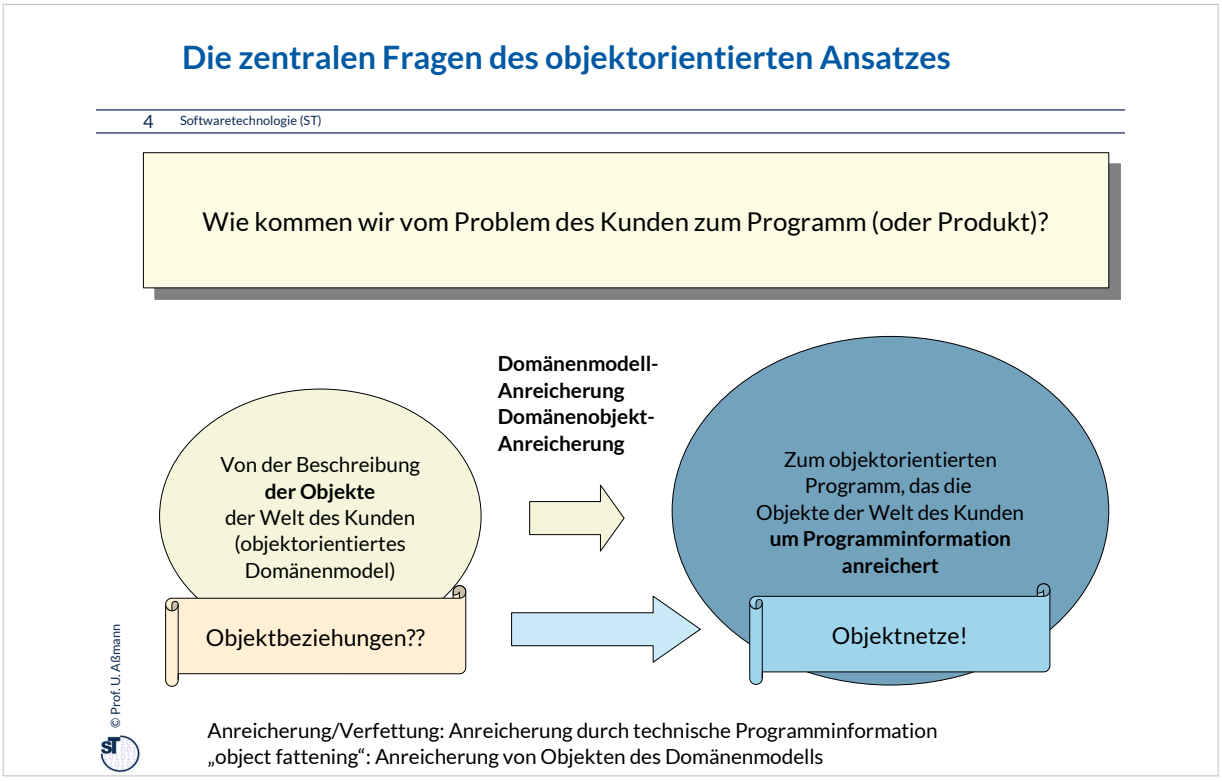
http://www.kprdd.de/Dresden/Geschichte/Frauenkirche/Frauenkirche.htm
Fehltönender Probeguss der Johannesglocke von Watzke am Goldenen Reiter
http://www.watzke.de/watzke_am_goldenen_reiter.html
Geschichte: http://www.md-pro.de/depot/mitteilungen/mit0401.pdf

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.



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:

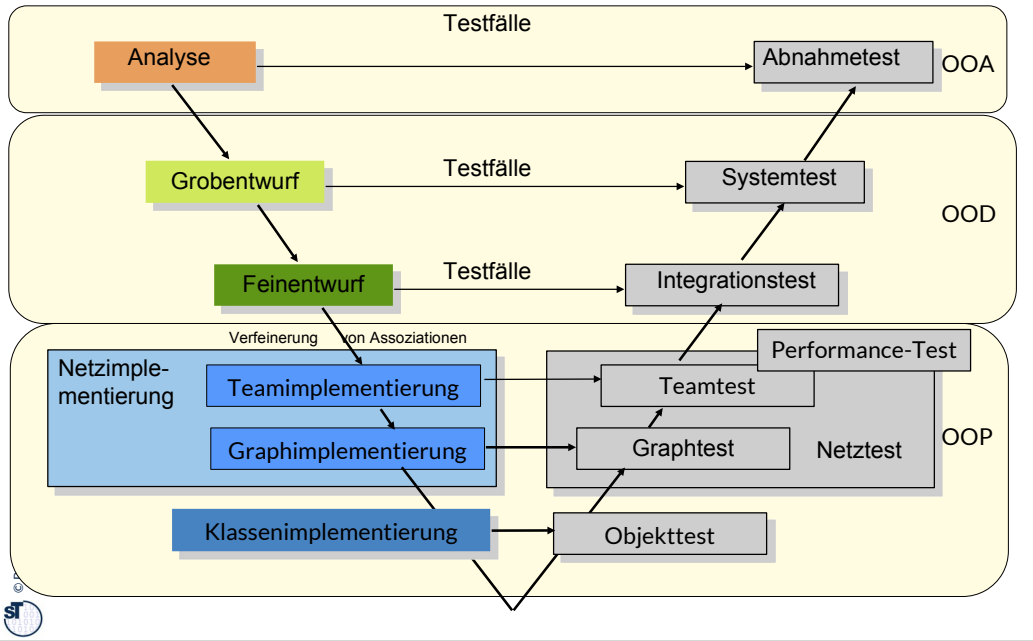
- “dangling pointers” (Luft-Zeiger) werden durch Typsierung vermieden oder durch Tests, die bei Frameworks für Netze mitgeliefert werden, schnell gefunden
- Frameworks verhindern dangling pointers von vorne herein durch sichere Konstruktion des Netzes (Build-Entwurfsmuster wie Fabrikmethoden, Fabriken)

Ziel von Teil II der Vorlesung

- Wie kann man **Objektnetze** (“Assoziationen”, 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

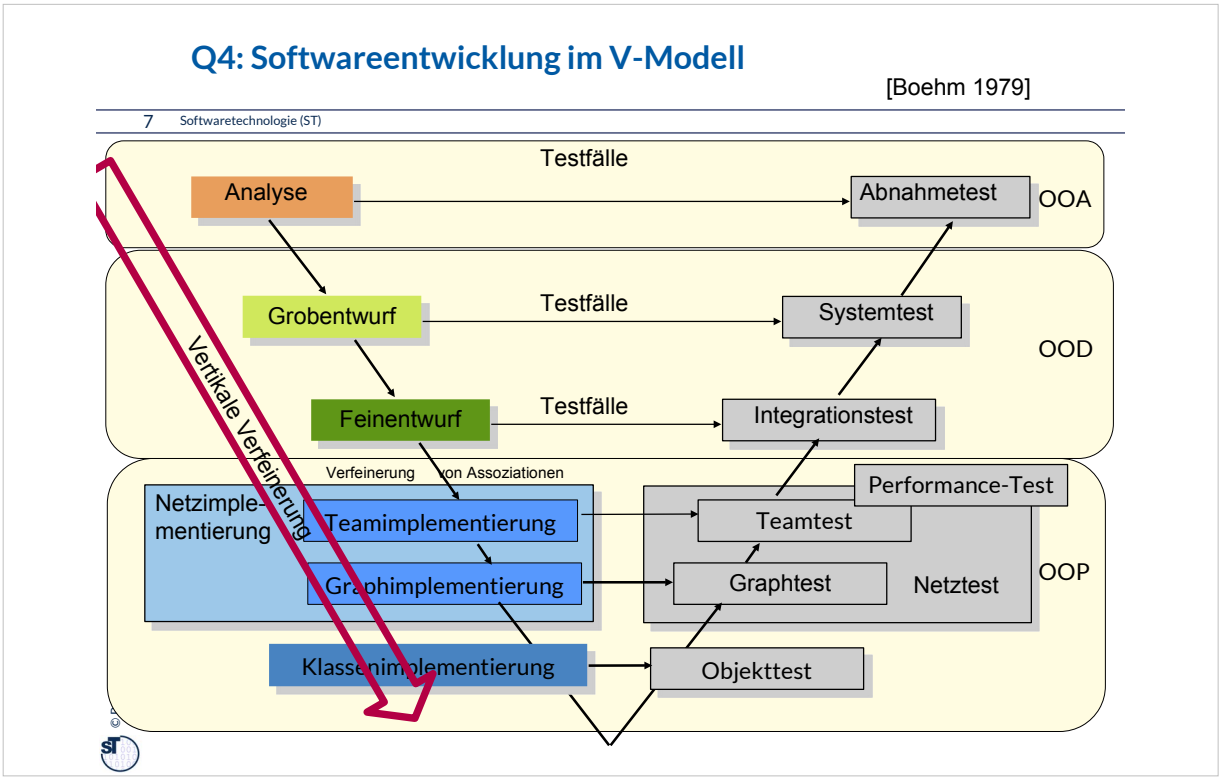
Q4: Softwareentwicklung im V-Modell

[Boehm 1979]



Quelle fuer das Boehm-Zitat: [Hesse/Merbeth/Frölich Software-Entwicklung, De Gruyter-Oldenbourg, S. 37] https://www.degruyter.com/document/doi/10.1515/9783110701951/html?lang=de

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



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

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

Modellierung im V-Modell

- Im V-Modell werden mehrere Sprachen gleichzeitig benutzt (hier aUML, dUML, jUML)
- Die Modelle werden durch verschiedene Verfeinerungsoperationen aus dem initalen Anforderungsmodell entwickelt

- Ein **Sprachkonstrukt (Sprachelement)** bezeichnet ein Konstrukt bzw. Konzept einer Sprache.
- Ein **Programm-/Modellelement** bezeichnet ein Element eines Programms bzw. eines Anforderungs- oder Entwufsmodells bezogen auf ein Sprachkonstrukt.
- 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"), der mit Füllseln gefüllt werden muss.
- 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, Verfeinerung)** ist das Anfügen von Details. Verfeinerung kann *horizontal* oder *vertikal* erfolgen.

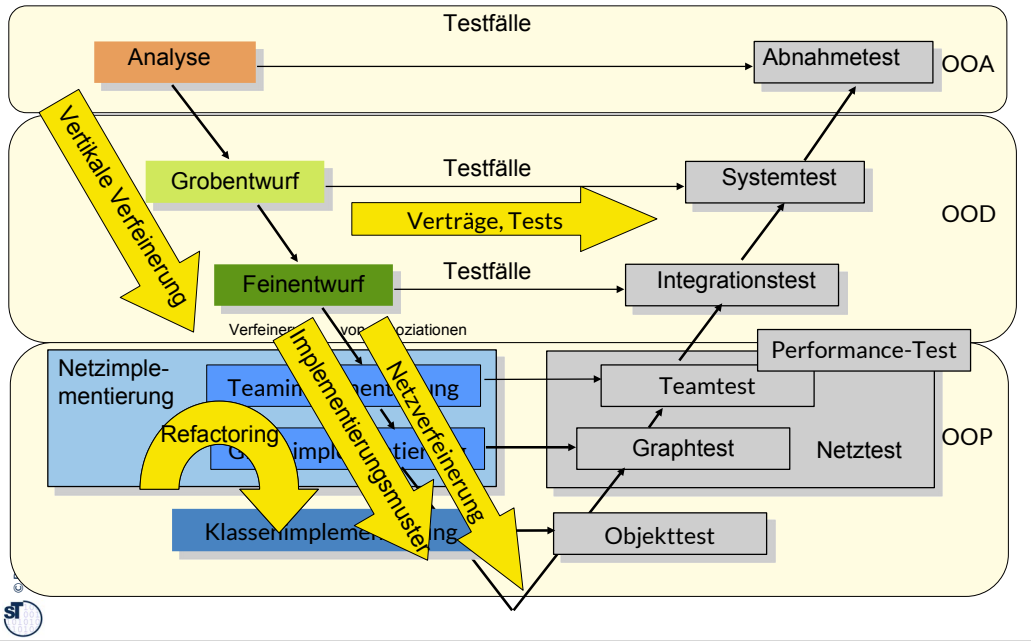
Software-Detaillierung (Verfeinerung) im V-Modell

- **Horizontale Verfeinerungsoperationen** ersetzen Fragmente und Fragmentgruppen 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) vereinfachen Fragmente und wechseln dabei die Sprache, z.B. von UML nach Java:
  - Abflachen von Fragmenten (Flachklopfen, Realisierung, lowering): Realisierung ersetzt ausdrucksstarke Konstrukte durch weniger ausdrucksstarke, implementierungsnähere
  - Einsatz von Implementierungsmustern

Verfeinerung heißt also meist “Detaillierung”.

Q4: Softwareentwicklung im V-Modell

[Boehm 1979]



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

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

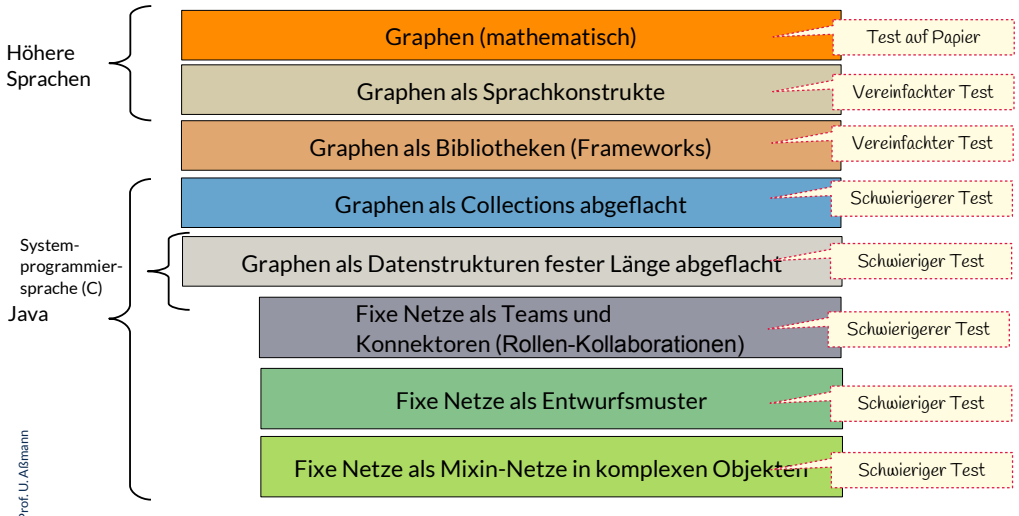
•Verfeinerung von Sprachkonstrukten (Realisierung, Abflachen, lowering): Viele der Sprachkonstrukte aus UML haben kein direktes Java-Äquivalent und müssen zu Java-Konstrukten verfeinert werden.

- 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 und fixen Objektnetzen als Datenstrukturen (Netzverfeinerung)



Auf Ebene der Anforderungen werden **flexible Objektnetze** als math. Graphen dargestellt. Im Grobentwurf und Feinentwurf werden sie repräsentiert durch (verfeinert zu)

- Graphen als **Sprachkonstrukte** (für Sprachen, die das eingebaut haben)
  - Rapid Application Development (RAD)
- Graphen aus Java-Graph-**Bibliotheken** (jgrapht)
- **Implementierungsmuster** wie **Collections**, nach dem Abflachen/Flachklopfen von bidirektionalen Assoziationen in gerichteten Links
- **maschinennahe Implementierungsmuster** wie **Datenstrukturen** fester Länge (Arrays, Matrizen) (speicher-bewusstes Programmieren)

Wohl dem, der eine gute Testsuite für flexible Objektnetze hat!

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

Fixe Netze mit statisch festem n, m (z.B. 1:1-Assoziationen) können durch **Teams** (Kollaborationen, Konnektoren) verfeinert werden durch

- Konnektorklassen realisieren Assoziationen und tragen Rollentypen als Assoziationsenden
- **Entwurfsmuster** wie Decorator, Chain, Composite
- interne Subobjektnetze großer Objekte (bobs), Endo-Assoziationen

Verständnisfragen

- Wieso kann man den Klang einer Glocke nicht testen?
- Wieso braucht man viel Erfahrungen, um die Parameter einer Glocke korrekt zu schätzen?
- Warum ist das Programmieren von Objektnetzen so schwierig?
- Welche Möglichkeiten gibt es fürs Testen, wenn man Objektnetz-Bibliotheken verwendet?
- Warum ist Performance-Test von Netz- und Objekttest verschieden?
- Warum ist Testautomatisierung für die Programmierung von Objektnetzen so wichtig?