

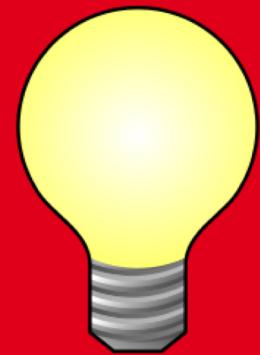


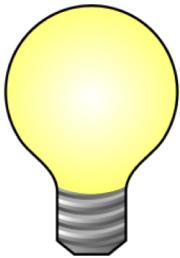
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

05.11.2020

# Einführung in die Vorlesung SWT

Allgemeine Informationen zum Ablauf der  
Vorlesung





Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# AGENDA

Willkommen

Allgemeine Bemerkungen über den Ablauf

Vorstellung der Themengebiete

Literatur

Packen wir's an!



# WILLKOMMEN ZURÜCK

- Schön, dass Sie wieder hier sind!
- Ich hoffe Sie hatten eine schöne vorlesungsfreie Zeit!
- Nun geht's wieder los
  - Hoffentlich haben Sie nicht alles vergessen
  - Leider steigen wir auch wieder relativ schnell ein



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# 01

## Willkommen

Ziel:  
Bestandsaufnahme  
Was fanden Sie bisher gut?  
Was könnte man verbessern?

# ÜBER MICH



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim



Manuel Dudda (M. Sc.)

- Freiberuflicher Software-Entwickler (<https://duddaweb.de>)
- 2015 – heute: selbstständig
- 2016: Abschluss Master of Science (Informatik), an der HSRM
- 2010 – 2015: Angestellter bei Web-Agentur sowie Start-Up
- 2010: Abschluss Bachelor of Science (Allg. Informatik), an der HSRM
  
- E-Mail: **Manuel.Dudda@hs-rm.de**
- Thema der Master-Thesis: „Ein Model-Finder für UML-Klassendiagramme“
- Honorar-Tätigkeit: Weiterentwicklung der Anwendung „UMoFi“ (UML-Model-Finder)
- Leitung von Softwaretechnik-Praktika seit WS 2016/17

# ZENTRALE ANLAUFSTELLEN



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Videos: <https://video.cs.hs-rm.de/>,  
Passwort: SWT2020
- Vorlesung (BBB): <https://zapp.mi.hs-rm.de/>
- Praktikum (BBB): <https://zapp.mi.hs-rm.de/>
- Übungsblätter: <https://studip.hs-rm.de/>
- E-Mail: [Manuel.Dudda@hs-rm.de](mailto:Manuel.Dudda@hs-rm.de)



# WILLKOMMEN ZURÜCK

- Schön, dass Sie wieder hier sind!
- Ich hoffe Sie hatten eine schöne vorlesungsfreie Zeit!
- Nun geht's wieder los
  - Hoffentlich haben Sie nicht alles vergessen
  - Leider steigen wir auch wieder relativ schnell ein

# BESTANDSAUFGNAHME



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Was wurde bisher behandelt?
  - Programmieren
    - OOSE
    - PM
  - Steigerung der Softwarequalität
    - PM
      - Selbstdokumentierender Code
      - Design By Contract
      - Unittesting

} Schon ein  
Einstieg was  
wir hier machen

# ZIELE DIESER VERANSTALTUNG



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Vorbereitung auf die professionelle Software-Entwicklung
  - Entwicklung großer Softwaresysteme
    - Arbeitsteilig
    - Systematisch
  - Modellierung
  - Qualitätsmanagement
  - Vorgehensmodelle

# IHRE ERWARTUNGSHALTUNG



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Was fanden Sie bisher gut?
- Was kann verbessert werden?
- Gibt es ein Thema aus der vorherigen Vorlesung, das generell noch unter den Nägeln brennt?

# EINSCHUB: WIESO GIBT ES KEINE EINDEUTIGEN ANTWORTEN?



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Kritik aus vorherigen Feedbackbögen:  
„deutlichere Aussagen bzgl. bestimmter Fragen („kann so sein,  
oder so...“, „manchmal so ...“)“
- Leider gibt es keine einfachen Antworten
- Modellierung ist viel schwammiger als Programmieren
  - Es gibt immer mehrere Lösungen
  - Je nach Problemkonstellation kann eine Lösung auch besser oder schlechter sein
- Somit auch keine einfachen Rezepte möglich

# EINSCHUB: WIESO GIBT ES KEINE EINDEUTIGEN ANTWORTEN?



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Kritik aus vorherigen Feedbackbögen:  
„nur ein kleiner Teil des geforderten Stoffs wird in der Vorlesung behandelt“
  - Bezieht sich wohl auf die Recherchefragen?
- Vorlesung des 4. Semesters und nicht mehr 1. oder 2. Semester
  - Andere Lernziele → Auch Vorbereitung auf die Bachelorarbeit
    - In der Vorlesung lernen wir den groben Überblick
    - Im Praktikum lernen Sie
      - deren Anwendung
      - aber auch: selbständige Analyse und Recherche von wichtigem Wissen
  - Sie wollen irgendwann ja mal “Ingenieur” sein

# EINSCHUB: UNTERSCHÄTZEN SIE SWT NICHT!



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Vieles mag vielleicht zunächst einfach klingen
- Und vielleicht ist es nicht so spannend wie etwa Programmieren
- ABER: Sehr viele Projekte scheitern oder werden viel teurer, obwohl gute Leute dabei sind
  - Daraus wurden viele Schlüsse gezogen, wie es besser geht
  - Hier lernen Sie die Antworten und Konzepte dazu
- Wichtiger Faktor: Viele Personen an einem Projekt beteiligt
  - Vieles in SWT dreht sich deshalb um Kommunikation und Zusammenarbeit zwischen Personen in SW-Projekten



02

## Allgemeine Bemerkungen über den Ablauf

Ziel:

Meine Vorstellungen wie die Vorlesung verlaufen soll

Strategie („Kriegsplan“)

# ÜBERSICHT ÜBER INHALT (STOFF)



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- UML
- „Programmierung im Großen“
  - Vorgehensmodelle
  - Analyse
  - Entwurf einer Gesamtsoftware (Architektur)
  - Qualitätskontrolle (Test)
  - Muster
- „Programmieren im Kleinen“
  - Entwurf von Detailaspekten (Detailed Design)
  - Implementierung
  - Qualitätskontrolle (Test)
  - Objektorientierung

# AUFBAU



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Die Veranstaltung besteht aus **vier/fünf** Teilen:
  - **Vorlesung: 4h/Woche**
    - 2h/Woche „Flipped Classroom“! (AMIGO, SS2020, WI)
    - 2h/Woche Vorlesungs-Sprechstunde + Zusatzstoff
    - Start: 11:30 Uhr (ab 12.11.20)
  - Praktikum: 2h/Woche
  - Vor-/Nachbereitung der Vorlesung: 2h/Woche
  - Bearbeitung der Praktikumsaufgaben: 2h/Woche



## Vorsicht:

- Praktikum: Anwesenheitspflicht
  - **ABER: Auch das andere beachten!!**
- Berechnungsgrundlage
  - „normale/r“ Informatik-Student/in
  - Vollzeit-Studium  $\leq$  40h/Woche



# ANFORDERUNGEN

- Anwesenheitspflicht?
  - Besteht nur in den Praktika (auch in diesem Semester!)
- Praktische Leistungen → müssen bestanden werden
  - Übungsblätter
- Klausur am Ende (100%)
  - Corona: Höchstwahrscheinlich Klausur, **Februar 2021**

# ORGANISATORISCHES ALLGEMEIN



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Vorlesung: **Donnerstag, 10:00 – 13:15, B002**
  - Corona: Per Video + BBB-Session am Do **ab 11:30 Uhr**
- Praktikum
  - Beginn: Ab heute
  - Termin: Je nach Ihrer Anmeldung
  - Corona: BBB-Session jeweils zu den angegebenen Zeiten
  - Achtung: Praktikum am **Do.** nach der Vorl. wird **nicht** aufgezeichnet
- Zugang zu virtuellen Räumen
  - <https://zapp.mi.hs-rm.de> (mit HDS-Account anmelden)
    - Die jeweiligen Räume nutzen!

# ZUGANG ZU MATERIALIEN, ...



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Stud-IP: Nur für die Mailliste
- Folien, Aufgabenblätter, Alte SWT-Filme → **Stud-IP**
- Vorlesungsfilme über Amigo (Passwort: SWT2020)
  - Vorher selbst ansehen → Inverted Classroom

# ZEITPLAN (derzeitige Planung)



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

Flipped Classroom ( <a href="https://video.cs.hs-rm.de/course/77/">https://video.cs.hs-rm.de/course/77/</a> ), PW: SWT2020	Länge	Praktikum Übungsblätter	Abgabe/Vorstellung Übungsblatt
05. Nov		Praktikum 1 Einführung & Motivation	12.11 – 18.11
12. Nov UML – Einführung und Objektdiagramm	01:02:42	Praktikum 2 Objektdiagramme	19.11 – 25.11
19. Nov UML – Klassendiagramm	01:10:31	Praktikum 3 Klassendiagramme	26.11 – 02.12
26. Nov UML – Zustandsdiagramm	01:01:30	Praktikum 4 Zustandsdiagramme	03.12 – 09.12
03. Dez		Praktikum 5 GIT	10.12 – 16.12
10. Dez UML – Aktivitätsdiagramme	00:42:15	Praktikum 6 Aktivitätsdiagramme	17.12 – 06.01
17. Dez UML – Interaktionsdiagramme	00:51:34	kein neues Übungsblatt	
24. Dez			
31. Dez			
07. Jan PiG – Einstieg & Analyse	01:48:29	Praktikum 7 Interaktionsdiagramme	07.01 – 13.01
14. Jan PiG – Grobentwurf	01:21:02	Praktikum 8 Einstieg ins Fachmodell	14.01 – 20.01
21. Jan PiG – Feinentwurf	00:43:34	Praktikum 9 Analyse	21.01 – 27.01
28. Jan PiG – Muster	01:59:43	Praktikum 10 Entwurf	28.01 – 03.02
04. Feb PiG – Testen	01:49:58	Praktikum 11 Testen	04.02 – 10.02
11. Feb PiG – Durchführen von Projekten	01:35:31	Praktikum 12 noch offen ...	11.02 – 17.02
18. Feb		kein neues Übungsblatt	

D.h.:

- **Filme “Einführ. In SWT” & “UML - ... Objektdiagramm” bis 12.11. VOR der Vorlesung sehen**
- **Film “UML - Klassendiagramm” bis 19.11. VOR der Vorlesung sehen**
- ...

# PRAKTIKUM – VORBEREITUNG & DURCHFÜHRUNG



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Jede Woche neues Aufgabenblatt
- zu Hause / in der Lerngruppe lösen
  - Nur Digital zählt!
  - Lösung dem Betreuer zusenden!
- Sie stellen ihre Lösungen im Praktikum vor
  - Auswahl abhängig von Praktikumsleitung (z.B. nach Zufallsprinzip)
- Empfehlung: Zeichnungen mit Kugelschreiber zeichnen
  - Aber kein Karopapier!

# PRAKTIKUM – BESTEHENSKRITERIEN:



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- 75% Anwesenheit → Prüfung am Anfang des Praktikums
  - Wichtig wegen des Seminarcharakters & Bestehen
  - (Mögl. Alternativregel: 0%, wenn aufgerufen & nicht anwesend)
- Zwei Teile, die Sie beide bestehen müssen (je  $\geq 50\%$ )
  - I. Teil: UML allgemein
  - II. Teil: Programmieren im Großen (PiG)
  - Wenn man in einem Teil  $< 50\%$  hat, muss man im anderen Teil das Delta \*2 haben, um noch zu bestehen
  - BSP:
    - Teil I (UML): nur 30% geschafft → Delta 20%  
→ Teil II (PiG):  $50\% + 2*20\% = 90\%$

# PRAKTIKUM – BEWERTUNG

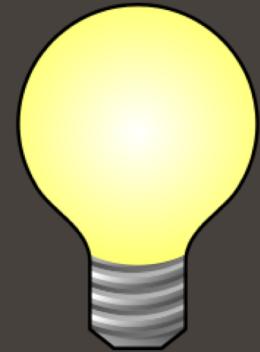


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Vorgestellte Lösung → Note [%]
- Nur vorhandene Lösungen in digitaler Form zählen
  - (Zeichnungen als Scan oder digitales Foto)
- jede/r Student/in: mind. 1x (eher 2x) pro Teil
- **1. Aufgabenblatt: kein Freischuss für den Teil II.  
(weil eher allgemein)**



## 03 Vorstellung der geplanten Themengebiete



Ziel:  
Überblick gewinnen über was wir -hoffentlich-  
sprechen werden.

# VORLESUNG 1 - HEUTE EINSTIEG FINDEN



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Einführung in die Vorlesung (→ gerade)
- Den ersten Einstieg finden
  - Einführung in die Softwaretechnik (SWT)

# VORLESUNGEN 2 – 5: UML



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

**ZIEL:** Die Modellierungssprache UML lernen

2. Einführung in UML & Objektdiagramm
3. Klassendiagramm
- 4-5. Zustands-, Aktivitäts- und Interaktionsdiagramme

# VORLESUNGEN 6 - 12: PROGRAMMIEREN IM GROSSEN



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

**Ziel:** Entwicklung großer Softwaresysteme

→ große Programme, viele Entwickler, echtes Produkt, ...

6. Einführung – Worum geht's genau? (eher kurz mit 7.)

7. Anforderungsanalyse

8. Grobentwurf (Architektur)

9. Feinentwurf (Detailed Design)

10. Die Musteridee umfassend kennenlernen

11. Testen – Wie kann man systematisch testen?

12. Durchführen von Projekten

# VORLESUNG 13:

## LETZTE VORLESUNG



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

**Ziel:** Rekapitulation und Klausurvorbereitung

### 13. Klausurvorbereitung (Online über BBB)

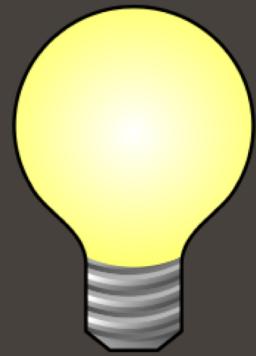
- Besprechen die Details zur Klausur
- Probeklausur besprechen



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## 04 Literatur

Ziel:  
Literatur



# WO KANN MAN SELBST NACHRECHERCHIEREN?



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Bücher
  - allgemein
  - speziell
- Infos auf der Homepage
  - Vorlesungsfolien
  - ggfs. Dossiers
- Dossiers
  - Unterstützung bei Vor-/Nachbereitung der Vorlesung
  - teilweise Verweis auf andere Lektüre
  - ggfs. detaillierter/ausführlicher als Vorlesung
  - Pflicht!

# BÜCHER – SOFTWARETECHNIK ALLGEMEIN



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- S. Kleuker: Grundkurs Software-Engineering mit UML.
  - <http://dx.doi.org/10.1007/978-3-8348-9843-2>
- Zuser et al: Software-Engineering mit UML und dem Unified Process.
  - [BF 500 92]
- van Vliet: Software Engineering: Principles and Practice.
- ...

# BÜCHER – SPEZIELL UML



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- H. Störrle: UML 2 für Studenten.  
→ Als Einführung gut geeignet – Leider teilw. nicht mehr auf akt. Stand
- H.J. van Randen; et al.: Einführung in UML  
Gratis in der Bib. verfügbar:  
<https://hds.hebis.de/hsrm/Record/HEB386572232>
- Ch. Rupp et al: UML 2 glasklar [BF 500 91]  
Onlineexemplare: <https://hds.hebis.de/hsrm/Record/HEB46314144X>  
→ Für Fortgeschrittene geeignet. Hilft auch bei kniffligen Fragen
- Spezifikationen im Original auf [www.uml.org](http://www.uml.org)  
→ Man muss wissen, dass es das gibt und wo man das findet.  
– Jedoch eher Nicht zum Lernen geeignet

Mein Tip  
(ist auch so das  
Standardwerk)

# ELEKTRONISCHE BÜCHER IN UNSERER BIBLIOTHEK



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Zugriff auf eBooks
  - vom Hochschul-(W)LAN aus
  - von überall über Hochschul-VPN

Siehe: <https://www.hs-rm.de/de/service/hochschul-und-landesbibliothek/a-z/informationen-a-z/hinweise-zum-vpn>

- Lesen über kostenfreie Reader „Adobe Digital Editions“
- Lesen von eBooks
  - Online
  - Download PDF-Datei → offline

# EMPFEHLUNGEN ZU DER VORLESUNG



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Besorgen Sie sich ein Buch zu jedem Themenbereich
  - Sie brauchen das für die Aufgaben
    - Recherche-Aufgaben etc.
  - Ein gutes Buch (zu jedem Thema) genügt
- Wie finden?
  - Erst ausführlich testlesen
  - Am besten ausleihen

- ⚠ Vorsicht:**
- Es genügt **nicht**, ein gutes Buch zu **besitzen**.  
→ Lesen!
  - Frage zu Vorlesung / Aufgaben?  
→ Was sagt das Buch dazu?

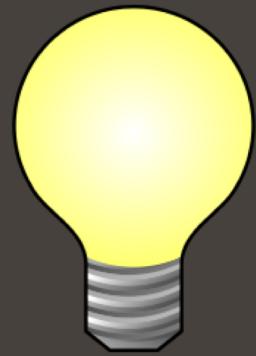


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## 05

# Packen wir's an!

Ziel:  
Was will ich damit sagen?



# PACKEN WIR'S WIEDER AN!



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Programmieren ist nicht alles
  - Wichtig ja!
  - Und Sie werden das brauchen
- ABER:
  - Bisher haben Sie gelernt wie man Beton anröhrt, mauert, ...
  - Jetzt: Wie man den Bau plant (Architektur), die Baustelle am Laufen hält (Vorgehensmodelle) und solide Arbeit abliefert (Qualität) ...
- Es gibt so viel mehr
  - Diese Vorlesung wird Ihren Blick verändern
- Ruhnen Sie sich Nicht auf dem Erreichten aus!
  - Steigen Sie jetzt gleich wieder ein





Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

AUF GEHT'S!!

SELBER MACHEN UND LERNEN!!



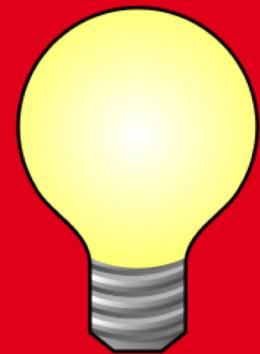


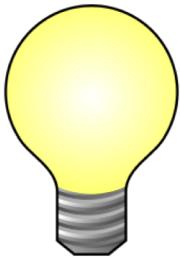
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

05.11.2020

# Softwaretechnik – Was ist das?

Einführung in die Softwaretechnik-Vorlesung





Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# AGENDA

Einführung ins Thema

Was ist Softwaretechnik?

Der Softwareentwicklungszyklus

Vorgehensmodelle

Fazit

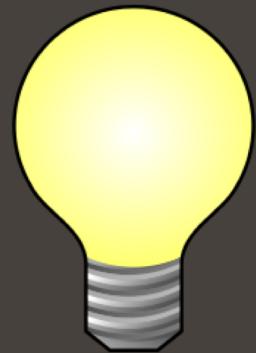


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# 01

## EINFÜHRUNG INS THEMA

Ziel:  
Die Eckpunkte des Themas kennenlernen



# GRUNDLEGENDE FRAGEN ZU SWT



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

Sie werden sich zu Softwaretechnik fragen:

- Was ist das?
- Wozu braucht man das?

→ Siehe Film „clip\_1\_final.mp4“



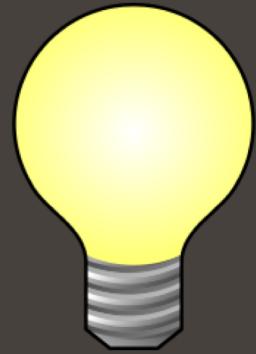
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

02

## Was ist Softwaretechnik?

Ziel:

Was versteht man unter Softwaretechnik?



# AUSGANGSPUNKT



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## Typische Probleme bei der Software-Entwicklung:

- viele IT-Projekte scheitern
- Konsequenzen fehlerhafter Software
- Kosten und Dauer von SW-Entwicklungsprojekten geraten aus dem Ruder
- ...

# ZIEL:



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

→ Günstige Software von hoher Qualität

- Hohe Qualität:
  - fehlerfrei
  - schnell
  - schnell/leicht veränderbar
  - leicht testbar
  - benutzerfreundlich
  - sicher

# ANSATZ I: INDUSTRIALISIERUNG



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Entwicklungsstufen:
  - Laie → Zufallstreffer
  - Handwerker → Teure Einzelanfertigung
  - Ingenieur → Qualität am Fließband

→ Software-Entwicklung als Ingenieursdisziplin:  
    → Softwaretechnik (Software Engineering)

- Erfolgsfaktoren:
  - erprobte Methoden
  - erprobte Werkzeuge



# DEFINITIONEN

- Softwaretechnik (Software Engineering) =
  - Methoden + Werkzeuge + Hilfsmittel,
  - die Softwareentwickler dabei unterstützen,
  - für Geld reproduzierbar
  - Software hoher Qualität herzustellen.
- Software =
  - Programm
  - Konfigurationsdateien
  - Dokumentation
  - . . .

# HINWEISE ZU DEN BEGRIFFEN



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Vielleicht spreche ich auch öfters von Software Engineering
  - Der englische Begriff
  - Abkürzung: SE
  - Auch im deutschsprachigen Raum
- Weiterer Hinweis: Es gibt auch Systems Engineering
  - Hier geht es darum ganze Systeme zu entwickeln
    - HW, Software, Mechanik, Elektrotechnik, Chemie, ...
  - Auch hier ist Software involviert
  - Diesen Aspekt klammern wir hier aus
    - Hier geht es nur um reine Softwaresysteme
    - Näheres zu SysEng können Sie in der Anforderungsmanagement-Vorlesung erfahren

# ANSATZ II: SOFTWARE CRAFTMANSHIP



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## Ansatz I – Industrialisierung (traditionell)

- SE als Ingenieursdisziplin, die zur „Industrialisierung“ führt
  - Z.B. durch Strukturierte Methoden
- Große Fortschritte, aber nicht so durchschlagende Erfolge wie ursprünglich erwartet

## Ansatz II – Meisterliches Handwerk (Software Craftmanship)

- Man ist bescheidener geworden
  - SW ist anders als z.B. Bauingenieurwesen
    - SW ist z.B. sehr abstrakt und es sind andere Lösungen mögl.
- Agile Methoden

# ANSATZ II: SOFTWARE CRAFTMANSHIP



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Ansatz I – Industrialisierung (traditionell)
- Ansatz II – Meisterliches Handwerk (Software Craftmanship)  
→ z.B.: Agile Methoden
  
- Erfolgsfaktoren für Ansatz II:
  - erprobte Methoden
  - erprobte Werkzeuge

} NUR: Etwas andere Methoden und Werkzeuge

## Was lernen wir hier?

- Hauptsächlich Ansatz I (evtl. ein paar Bemerk. zu Ansatz II)
- Ansatz I ist nicht verkehrt → wird noch sehr oft benutzt
- Bildet die absolute Basis für weiterführende Überlegungen
- ABER: Es gibt eben wesentlich mehr als wir hier lernen können

# KERNTHEMEN FÜR UNS



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

Das sind für uns die wichtigsten Themen in diesem Semester:

- Strukturierte Vorgehensweise: hier OOAD  
  („Object-Oriented Analysis and Design“)
- Einheitliche Notation für SW-Modelle
  - Hier: UML (Unified Modeling Language)
  - FMC (Fundamental Modeling Concepts)
- Konzepte und Abstraktionen: z.B. Muster, SW-Architektur
- Qualitätssicherung – z.B. Testen
- CASE (Computer Aided Software Engineering)

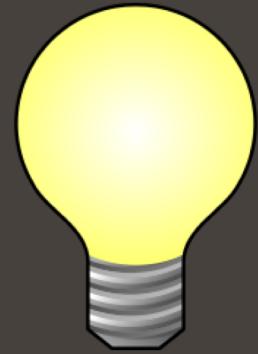


03

## Der Softwareentwicklungszyklus

Ziel:

Den grundlegenden Entwicklungszyklus  
für Software kennenlernen



# WICHTIGE BEGRIFFE – ARTEFAKT



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Beschreibt -ganz abstrakt- ein von Menschen künstlich (artifiziell) erstelltes Objekt.
- Ein Artefakt ist nicht notwendigerweise ein Dokument
  - Ein Art. kann durch eines oder mehrere Dok. ausgedrückt werden
  - Ein Dok. kann aber auch mehrere Art. beinhalten

## Beispiel:

- Das Artefakt „Quellcode“ besteht meist aus mehreren Codefiles (=mehrere Dokumente).
- Gut sich selbst dokumentierender Code (z.B. dokumentierte Methodenköpfe) enthält auch schon Teile des Artefaktes „Dokumentation“.
  - Können (z.B. über JavaDoc) noch in einen explizites Dokumentationsdokument transformiert werden.

# WICHTIGE BEGRIFFE – STAKEHOLDER



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Englisch für Interessenvertreter, Akteur
  - „Stabhalter“ - Staffellauf
  - Something is at stake == etwas steht auf dem Spiel

**Def nach Pohl und Rupp [PR15; S.4]:**

Ein Stakeholder eines Systems ist:

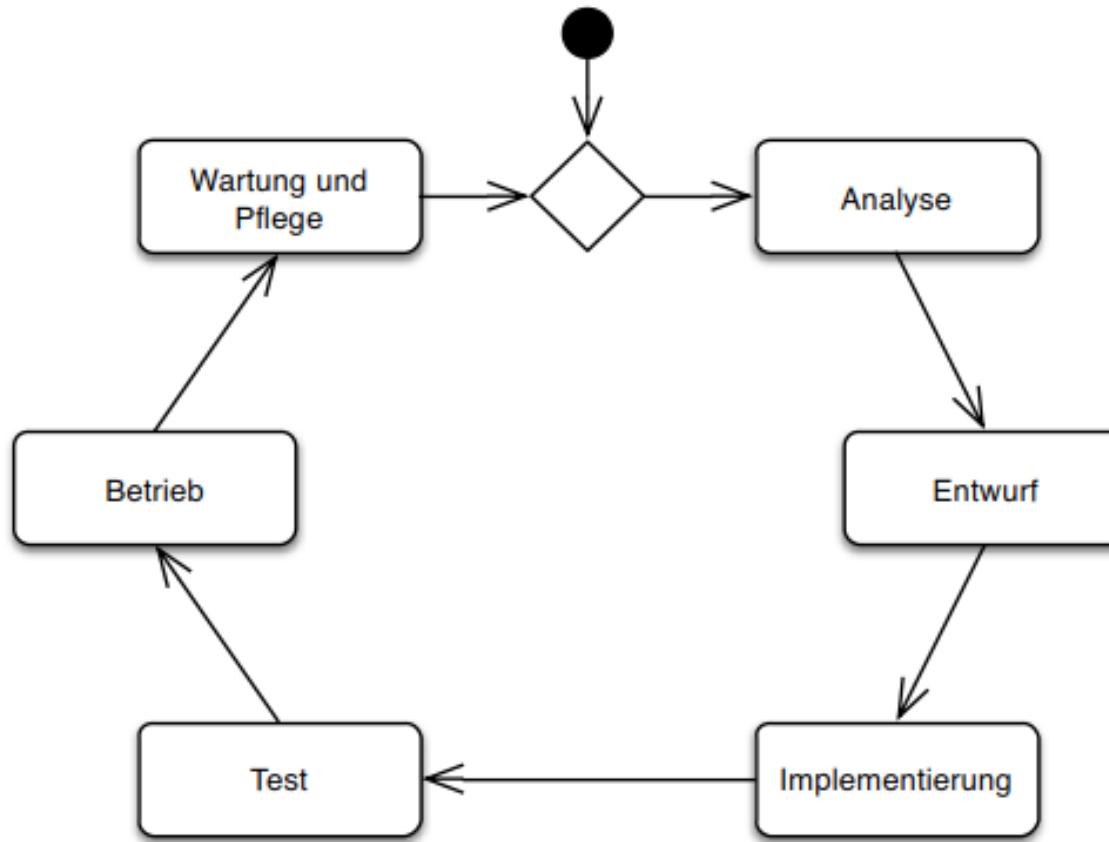
- eine Person oder Organisation, die
- direkten oder indirekten Einfluss auf das Projekt (v.a. die Anforderungen) des betrachteten Systems hat.



# LEBENSZYKLUS VON SOFTWARE



→ Beschreibt die typischen Tätigkeiten bei der SW-Entwicklung

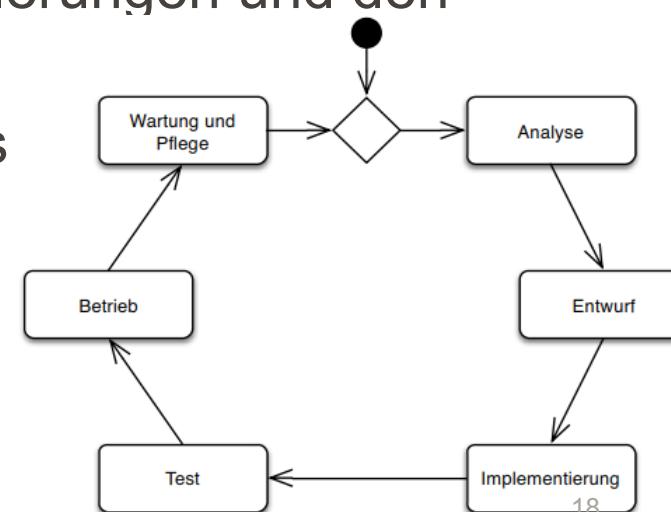


# LEBENSZYKLUS VON SOFTWARE



→ Beschreibt die typischen Tätigkeiten bei der SW-Entwicklung

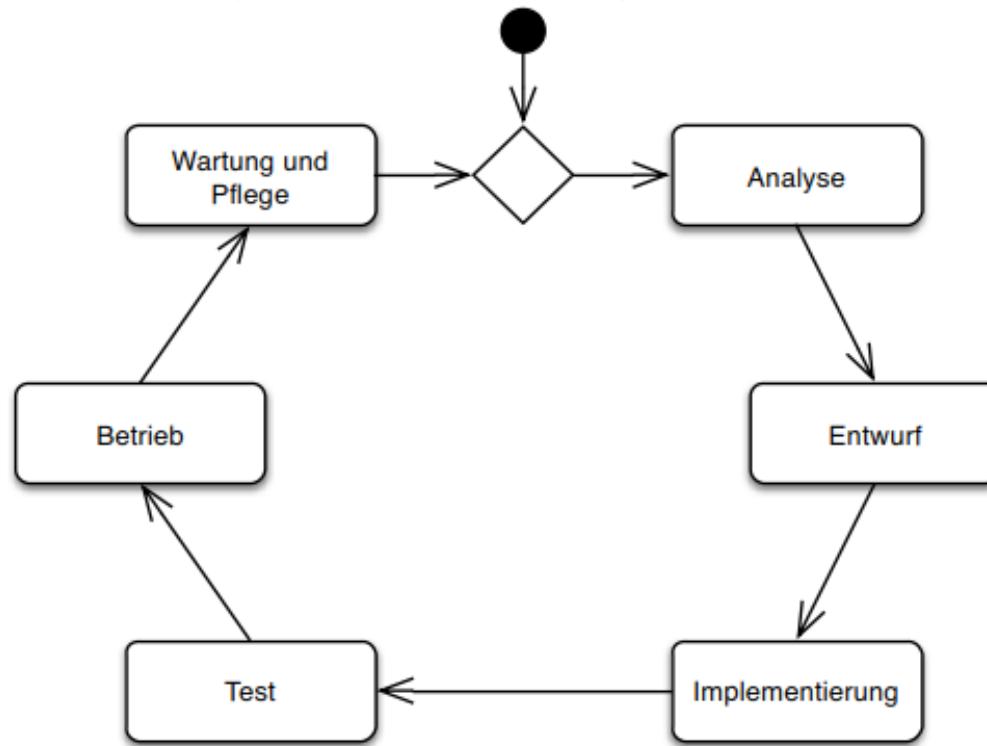
1. Analyse: Was will der Kunde? (= Anforderungen)
2. Entwurf: Wie soll das zu bauende System sein?
  - grob: Grobentwurf (Architektur/Architecture)
  - detailliert: Feinentwurf (Detailed Design)
3. Implementierung: Entwurf → Programm
4. Test: Erfüllt das Programm die Anforderungen und den Entwurf?
5. Betrieb: Verwendung des Programms
6. Wartung und Pflege
  - Änderungswünsche/Fehler
  - Was will der Kunde?
  - ...



# LEBENSZYKLUS VON SOFTWARE



→ Beschreibt die typischen Tätigkeiten bei der SW-Entwicklung



Vorsicht: Ist eine Idealisierung!

→ In der Praxis kann auch mal von Implementierung wieder zur Analyse zurückgesprungen werden, ...

# MEHR VORGABEN SIND NÖTIG



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Typische Fragen bei der Software-Entwicklung:
  - Wie fangen wir an?
  - Was sollen wir tun?
  - Wie verteilen wir die Aufgaben?
  - Wie machen wir's richtig?
  - . . .

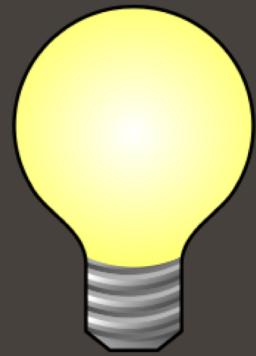
→ Hier sind mehr Vorgaben nötig



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## 04 Vorgehensmodelle

Ziel:  
Vorgehensmodelle kennenlernen



# MEHR VORGABEN SIND NÖTIG



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## → Vorgehensmodelle

- = Bestimmte Vorgaben für die Durchführung von Software-Entwicklungs-Projekten
- Typische Vorgaben:
  - Abfolge von Phasen/Tätigkeiten
  - Artefakte = Resultate von Phasen/Tätigkeiten, z.B.
    - Beschreibung der Anforderungen in bestimmter Form
    - Testfallbeschreibungen in bestimmter Form
    - Quellcode-Dateien gemäß Codier-Richtlinien
  - Zusammenhänge zwischen den Phasen/Tätigkeiten
  - Andere organisatorische Aspekte

# VORGEHENSMODELLE – BEISPIELE



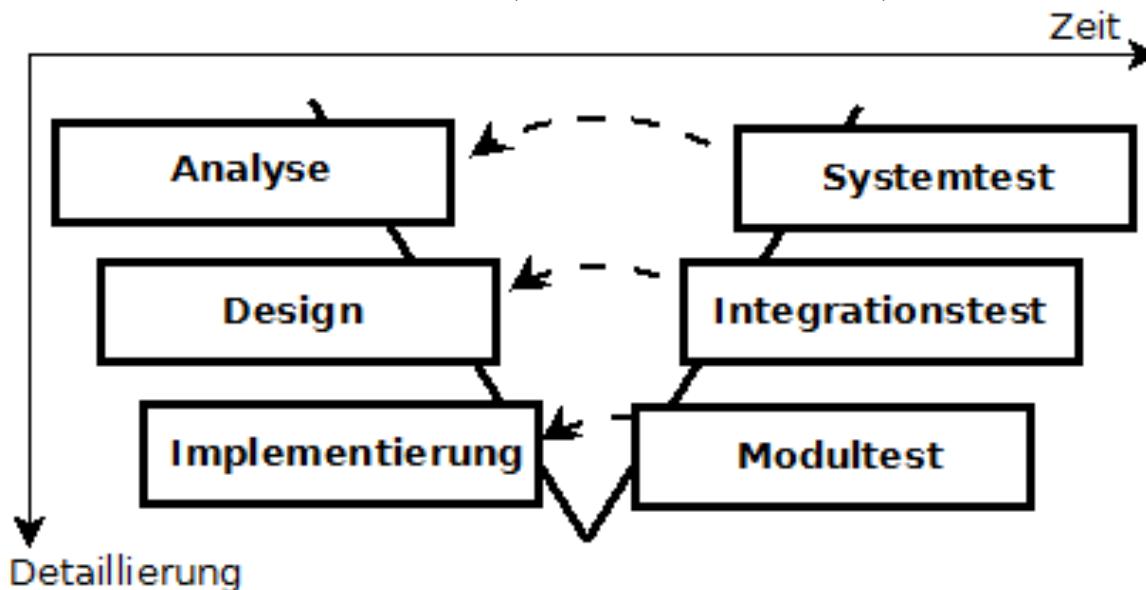
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Frühe Modelle
  - Wasserfall
  - V-Modell
  - (Deutsche Erfindung – oft benutzt, z.B. Behörden, Automotive)
- Objektorientierte Modelle
  - Spiralmodell (von Barry Boehm)
  - Rational Unified Process (RUP)
- Agile Methoden
  - eXtreme Programming
  - SCRUM



# V-MODELL

- Abwandlung des Wasserfall-Modells
  - Deutsche Erfindung (TU München → siehe Manfred Broy)
  - oft in Deutschland benutzt, z.B. Behörden, Automotive, ...

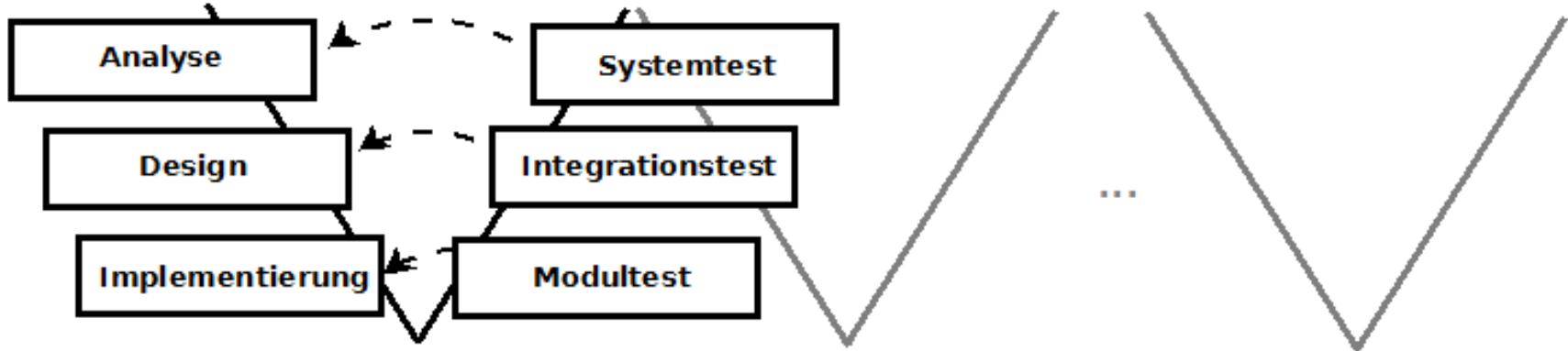


- Leider sehr starr
- Zunächst nicht iterativ geplant → Nur ein Zyklus



# V-MODELL

- Weiterentwicklung für iterative Entwicklung:



- Redesign (seit 2005): V-Modell XT
  - Iterativ, inkrementell, wesentlich flexibler und besser skalierbar (Reaktion auf RUP, Spiralmodell & Agile Methoden)



# SPIRALMODELL

- Entwickelt von Barry Boehm
  - Iterativ

## 1. Festlegen der Ziele

## 2. Beurteilen von Alternativen, Risikoanalyse

Zustimmung durch Überprüfung

Kosten

Fortschritte

## 4. Planung des nächsten Zyklus

## 3. Entwicklung und Test

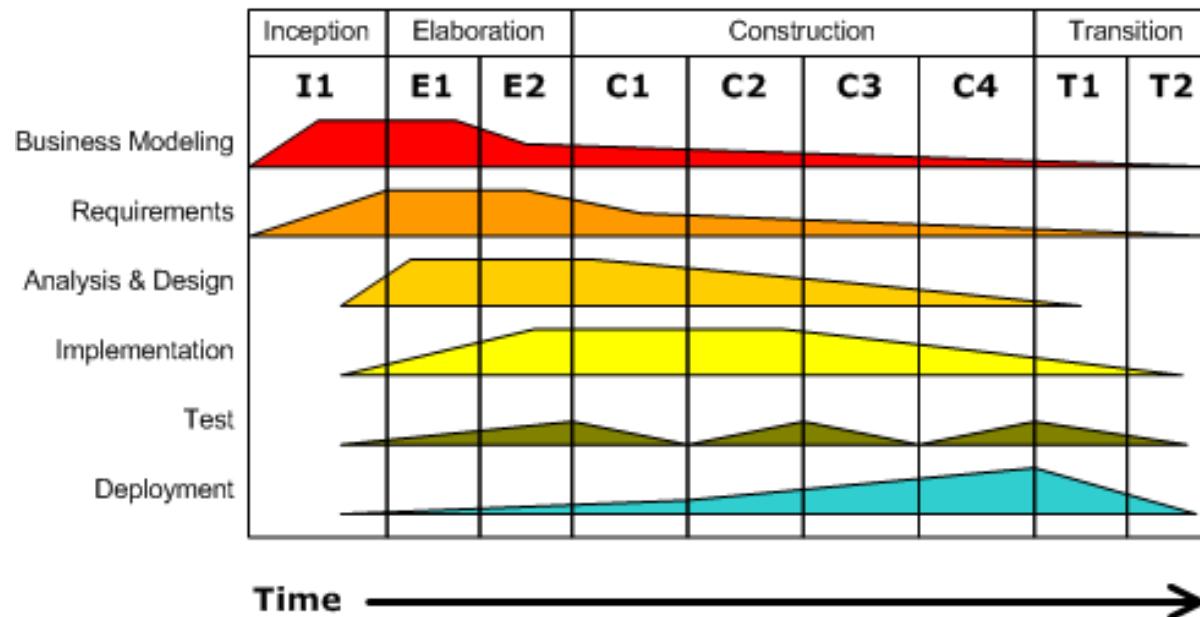


# RATIONAL UNIFIED PROCESS (RUP)

- Entwickelt parallel zur UML
  - Von der Firma Rational (jetzt IBM)

## Iterative Development

Business value is delivered incrementally in time-boxed cross-discipline iterations.





# AGILE METHODEN

- Relativ neu
  - Abkehr von starren Prozessen
  - Keine richtigen Vorgehensmodelle in klassischen Sinn
  - Eher Sammlung von Prinzipien (Best Practices)
    - Muster-Idee → Prozessmuster
    - (siehe Vorlesungseinheit zu Mustern)
- Typische Beispiele:
  - eXtreme Programming (XP) → Für Entwicklung
    - Nach Win Vista-Katastrophe hat Microsoft auf XP gesetzt → Win 7
  - SCRUM
    - Eigentlich eher eine Projektmanagementmethode
    - Für Entwicklung kann z.B. XP genutzt werden oder anderes
    - Derzeit richtig „IN“

# WAS MACHEN WIR JETZT



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Das war jetzt nur mal zur Orientierung
  - In Vorlesung 12\_ProgrammierenImGrossen\_VI\_Durchfuehren\_von\_Projekten werden wir darauf nochmals genauer zurückkommen
- Woran sollten Sie sich jetzt orientieren?
  - Wir benötigen für das Praktikum & spätere Projekte:
    - einen Rahmen für OOAD
    - solide
    - erprobt
  - V-Modell und RUP
  - V-Modell, weil es sehr eingängig von den Phasen her ist
  - RUP, weil es für Objektorientierte Entwicklung spezielle viele gute Sachen vorstellt
  - Beides ist dazu gut kompatibel

# WELCHE TÄTIGKEITEN BETRACHTEN WIR?



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Wir betrachten folgende Tätigkeiten (RUP: „disciplines“)
  - Anforderungs-Analyse (RUP: „Requirements“)
    - Was will der Kunde?
  - Analyse und Entwurf (RUP: „Analysis and Design“)
    - Wie soll das zu bauende System sein?
  - Implementierung (RUP: „Implementation“)
    - Das System programmieren
  - Testen (RUP: „Test“):
    - Wie stelle ich sicher, dass das System das tut, was es tun soll?
- Wir gehen nicht ein auf:
  - Geschäftsprozessmodellierung (RUP: „Business Modeling“)
  - Inbetriebnahme (RUP: „Deployment“)

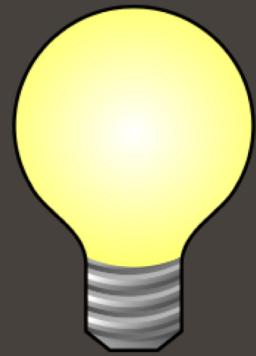


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# 05

## Fazit

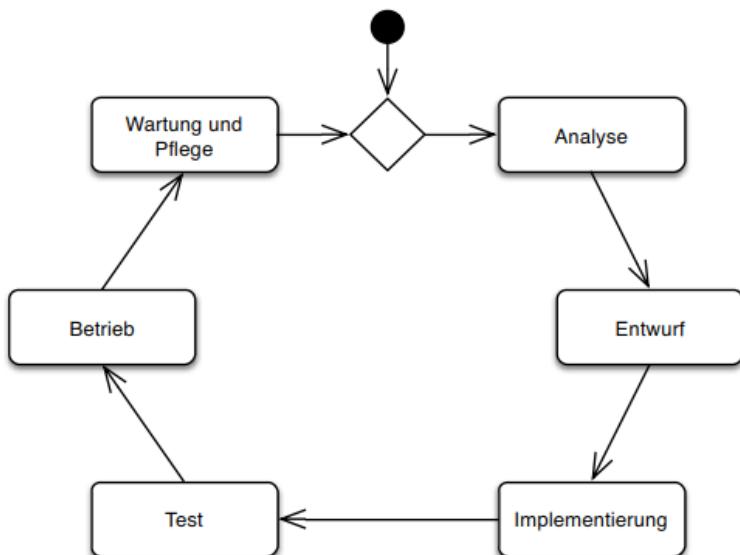
Ziel:  
Was haben wir damit gewonnen?





# WAS HABEN WIR GELERNT?

- Den Begriff Software Engineering kennenzulernen
  - Lehre von Methoden + Werkzeugen + Hilfsmitteln zur systematischen Entwicklung von Software hoher Qualität
- Grundlegende Entwicklungszyklus für Software
- Vorgehensmodelle
  - Weiteres, genaueres Vorschriftengerüst



# WORUM GEHT ES IN DEN KOMMENDEN VORLESUNGEN?



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Modellierung von Software
  - Mittels Zeichnungen die Eigenschaften einer Software herausarbeiten  
→ Wir lernen die Modellierungssprache UML kennen
- Später (ab Mitte des Semesters):
  - Einsatz dieser Zeichnungen in einen typischen Projekt  
→ Anhand des Vorgehensmodells RUP
- + Weitere wichtige Sachen jeweils zu diesen Aktivitäten

# RECHERCHE



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Das Recherchieren ist ein elementarer Bestandteil von wissenschaftlichen Arbeiten.
- Insbesondere bei Hausarbeiten, Abschlussarbeiten, wissenschaftlichen Papern ist es wichtig, Grundlagen und Folgerungen durch ordentliche Quellenangaben zu belegen und dokumentieren.
- Die häufigsten Fragestellungen dabei sind:
  - welche Quellen sind relevant?
  - was ist eine wissenschaftliche Quelle?
  - wie suche und finde ich Informationen im Internet?

# SUCHBEGRIFF



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Wonach suche ich?
- Verwende wissenschaftlich anerkannt und gebräuchlichen Suchbegriff
- Dieser ändert/erweitert/etabliert sich ggf. mit der Zeit
- Verwende ggf. Synonyme

# WISSENSCHAFTLICHKEIT DER QUELLE



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Wer ist der Autor?
- Wo wurde der Text veröffentlicht? Herausgeber, Website, ...?
- Verfügt der Artikel/Text über weitere Quellen/Literaturverzeichnis?
- Faustregel: Wenn kein Literaturverzeichnis => keine wissenschaftliche Quelle
- Aber nicht zwangsläufig umgekehrt: Wikipedia hat idR Literaturverzeichnis => dennoch kein wissenschaftlicher Text



# TEXTARTEN

👎 Zeitungsartikel

*dienen meistens der mediale Aufbereitung*

👎 Ratgeberliteratur

*behandeln meist spezielle Situationen*

👍 Artikel aus Fachzeitschriften

*zusammengefasste Forschungsergebnisse mit wissenschaftlicher Relevanz und Aktualität*

👍 Wissenschaftliche Monografien und Herausgeberbände

*„Sachbücher“, „Fachliteratur“*

👎 Diplom-, Magister-, Bachelor- und Masterarbeiten

*bedingt einsetzbar, da Qualität schwer beurteilbar*

👎 Handbuchartikel

*guter Einstieg für weiterführende Recherche*

👍 Offizielle Dokumente

*Untersuchungsbericht, Spezifikationen, „Ur-Dokument“*

# RECHERCHEORTE



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- 👍 Fachportale  
„Springer Verlag“
- 👍 Archive von Fachzeitschriften
- 👍 Literaturverzeichnisse
- 👎 Google usw.
- 👎 Wikipedia, Foreneinträge und Blogs, ...

Quelle: <https://www.ewi-psy.fu-berlin.de/einrichtungen/arbeitsbereiche/sozialpaedagogik/lehre/Materialien/Leitfaden-fuer-wissenschaftliche-Recherchen- neu .pdf>

# RECHERCHE-BEISPIEL: ARIANE

## 5



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Suchbegriffe: „Ariane 5“, „Ariane V 88“, „Raketenfehlstart“, „Rakete Software-Fehler“, „Rakete Explosion“
- Ergebnisse:
  - Wikipedia  
*Autor? Literaturverzeichnis guter Einstieg*
  - ESA Website  
*„Ur-Dokument“*
  - Zeitungsartikel  
*medial aufbereitet, tlw. populistisch*
  - Videos  
*Als Informationsaufnahme geeignet, als Quelle nicht, Informationen nachträglich validieren!*
  - Deutsches Zentrum für Luft- und Raumfahrt (DLR)  
*Literaturverzeichnis guter Einstieg*
  - ...



# QUELLEN-NOTATIONEN

- Wie werden Quellenangaben notiert?
- Fußnote, Endnote
- Sollte beinhalten: Autor, Titel, [Kapitel], Jahr, [ISBN]
- Beispiele:  
Die Grundlagen für die ganzheitliche Betrachtung  
grammen bilden [UML2.5]

Braga, Portugal: Springer-Verlag, 2007, S. 632–647. ISBN: 978-3-540-71208-4. URL: <http://dl.acm.org/citation.cfm?id=1763507.1763571>.

[UML2.5] Object Management Group. *Unified Modelling Language*. Nov. 2015.  
URL: <http://www.omg.org/spec/UML/>.

welchen Programme aus der Entwicklungsumgebung auf das Board übertragen und debuggt werden können. Zur schnellen Inbetriebnahme des Boards und einfachen Verwendung des Codecs werden die mit [1] zur Verfügung gestellten Dateien verwendet

- [1] CHASSAING, Rulph: *Digital Signal Processing and Applications with the C6713 and C6416 DSK*. Hoboken, NJ, USA : John Wiley & Sons, 2005. – ISBN 978-0-471-70406-5
- [2] ELLIOTT, S. J. ; GARCIA-BONITO, J.: Active cancellation of pressure and pressure gradient in a diffuse sound field. In: *Journal of Sound and Vibration*



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

AUF GEHT'S!!

SELBER MACHEN UND LERNEN!!



# Softwaretechnik WS19/20

Allgemeine Informationen zum Ablauf der Vorlesung

# DB ade, SWT juuccheee

Naja fast ...

Time Overruns	% of Responses
Under 20%	13.9%
21 - 50%	18.3%
51 - 100%	20.0%
101 - 200%	35.5%
201 - 400%	11.2%
Over 400%	1.1%

Cost Overruns	% of Responses
Under 20%	15.5%
21 - 50%	31.5%
51 - 100%	29.6%
101 - 200%	10.2%
201 - 400%	8.8%
Over 400%	4.4%

% of Features/Functions	% of Responses
Less Than 25%	4.6%
25 - 49%	27.2%
50 - 74%	21.8%
75 - 99%	39.1%
100%	7.3%

# Chaos Report (1)

# Flughafen Berlin Brandenburg

Baubeginn 2006

Eröffnungstermine:

- November 2011
- Juni 2012
- Herbst 2012
- August 2013
- Frühjahr 2014 - Teilweise
- Möglicherweise 2018
- Oktober 2020

Eigentlicher Plan: 2007

# A2LL

- ▶ T-Systems und Herten (später ausgestiegen)
- ▶ Typische Webanwendung mit Applikationsservern
- ▶ Probleme:
  - ▶ Überlastung -> Schichtarbeit
  - ▶ Kontonummern wurden mit 0en aufgefüllt)
  - ▶ Straßennamen abgekürzt
  - ▶ Anmeldungen, Abmeldungen und Veränderungsmitteilungen zur Krankenversicherung aus unbekannten Gründen storniert
  - ▶ 25 Millionen Euro pro Monat zu viel an die Krankenkassen überwiesen
  - ▶ Berechnung des Zuschlags → Excel Tabelle

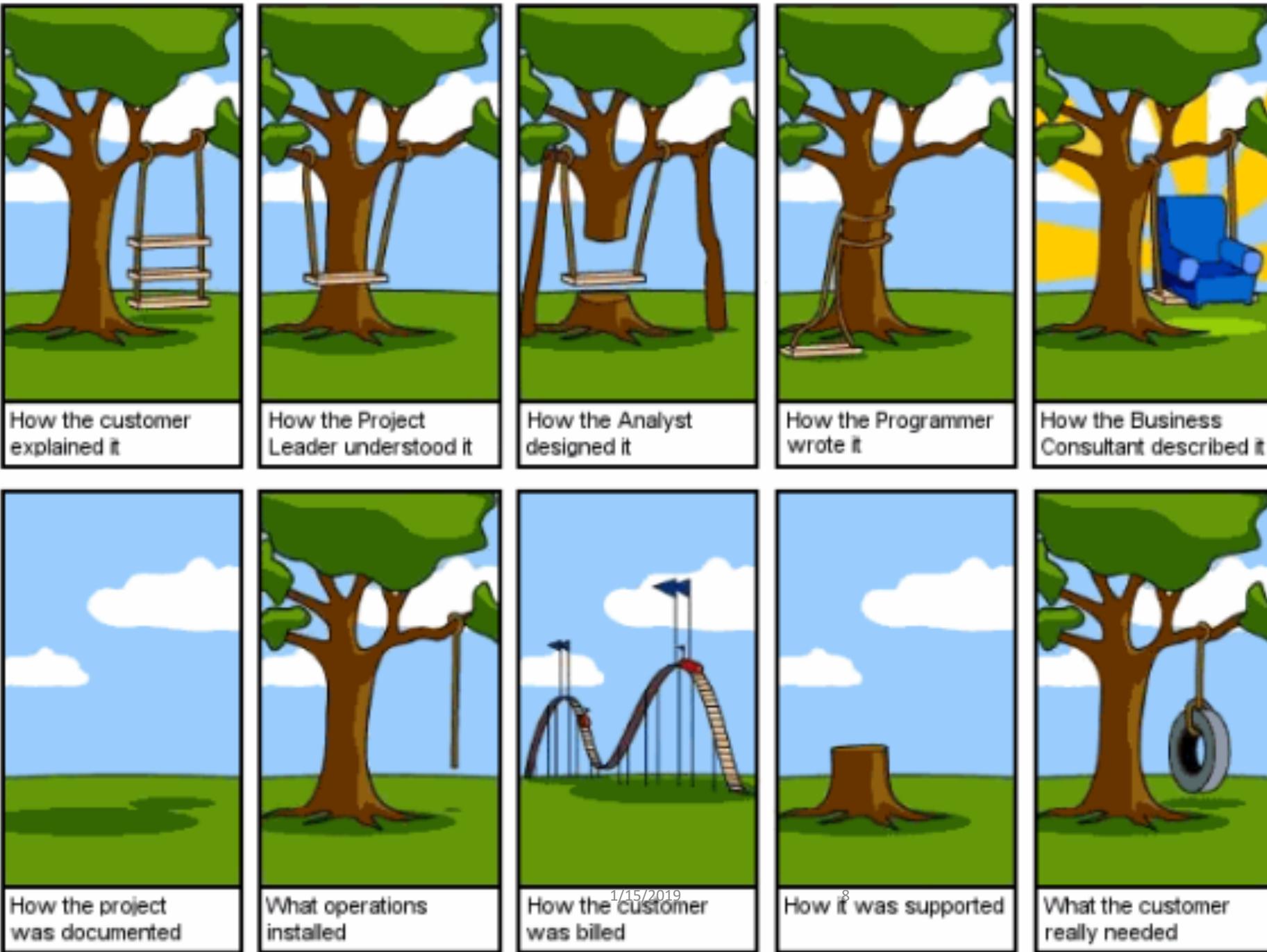
# Chaos Report (2)

Project Impaired Factors	% of Responses
1. Incomplete Requirements	13.1%
2. Lack of User Involvement	12.4%
3. Lack of Resources	10.6%
4. Unrealistic Expectations	9.9%
5. Lack of Executive Support	9.3%
6. Changing Requirements & Specifications	8.7%
7. Lack of Planning	8.1%
8. Didn't Need It Any Longer	7.5%
9. Lack of IT Management	6.2%
10. Technology Illiteracy	4.3%
Other	9.9%

	Than 5 Years Ago	Than 10 Years Ago
<b>Significantly More Failures</b>	27%	17%
<b>Somewhat More Failures</b>	21%	29%
<b>No Change</b>	11%	23%
<b>Somewhat Fewer Failures</b>	19%	23%
<b>Significantly Fewer Failures</b>	22%	8%

# Chaos Report (3)

# Softwaretechnik



# Ziele dieser Veranstaltung

- Vorbereitung auf die professionelle Software-Entwicklung
  - Entwicklung großer Softwaresysteme
    - Arbeitsteilig
    - Systematisch
  - Modellierung
  - Qualitätsmanagement
  - Vorgehensmodelle

# Softwaretechnik



Die Kunst erfolgreiche Softwareprojekte durchzuführen



Best Practice lernen, um Zeit für eigene Innovationen zu haben



Vom Hacker zum Informatiker

# Vorlesungsinhalte

# Inhalte

- ▶ Software-Entwicklungsprozesse
  - ▶ Traditionell
  - ▶ Agil
- ▶ Analyse
  - ▶ Anforderungsanalyse
  - ▶ UML @ Anforderungsanalyse
  - ▶ Story Card
- ▶ Design
  - ▶ UML @ Design
  - ▶ Design Pattern
  - ▶ Architektur Pattern
- ▶ Entwicklung
  - ▶ Clean Code
  - ▶ Testbarer Code
- ▶ Test
  - ▶ Testverfahren
  - ▶ Teststrategien

# Klausur

# Klausur

- ▶ Notizen: 1 A4 Blatt, beidseitig per Hand beschrieben, keine Kopie
- ▶ Elektronischen Geräte sind nicht erlaubt
- ▶ Zettel werden gestellt
- ▶ Bitte keine Bleistifte, Holzstifte und rote Stifte nutzen
- ▶ Probeklausuren werden im Laufe des Semester in studIP zur Verfügung gestellt.

# Literatur

# Allgemeine SWT Bücher



Software Engineering von Ian Sommerville  
10th Edition



Lehrbuch der Softwaretechnik von Helmut  
Balzert  
3. Auflage

## Bücher – Softwaretechnik Allgemein

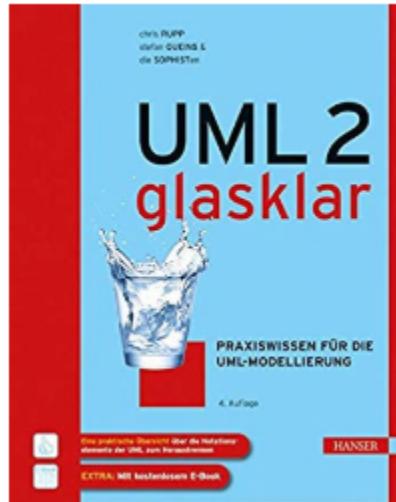


S. Kleuker: Grundkurs Software-Engineering mit UML

Download: <http://dx.doi.org/10.1007/978-3-8348-9843-2>

- van Vliet: Software Engineering: Principles and Practice.
- ...

# Bücher - Speziell UML



Viele Exemplare in der Bibliothek [BF 500 91]



# Bücher - Speziell UML

- ▶ H. Störrle: UML 2 für Studenten.  
→ Als Einführung geeignet - Leider teilw. nicht auf akt. Stand
- ▶ H.J. van Randen; et al.: Einführung in UML  
Gratis in der Bib. verfügbar:  
<https://hds.hebis.de/hsrm/Record/HEB386572232>
- ▶ <https://www.uml-diagrams.org>

# UML Spezifikation

► [www.uml.org](http://www.uml.org)

An OMG® Unified Modeling Language® Publication



## OMG® Unified Modeling Language® (OMG UML®)

Version 2.5.1

---

OMG Document Number: formal/2017-12-05

Date: December 2017

Normative Reference: <http://www.omg.org/spec/UML/2.5.1>

Machine readable files: <http://www.omg.org/spec/UML/20161101>

Normative:  
<http://www.omg.org/spec/UML/20161101/PrimitiveTypes.xmi>  
<http://www.omg.org/spec/UML/20161101/UML.xmi>  
<http://www.omg.org/spec/UML/20161101/StandardProfile.xmi>  
<http://www.omg.org/spec/UML/20161101/UMLDI.xmi>

---

# Elektronische Bücher in Unserer Bibliothek

- Zugriff auf eBooks
  - vom Hochschul-(W)LAN aus
  - von überall über Hochschul-VPN
- Lesen von eBooks
  - Online
  - Download PDF-Datei → offline

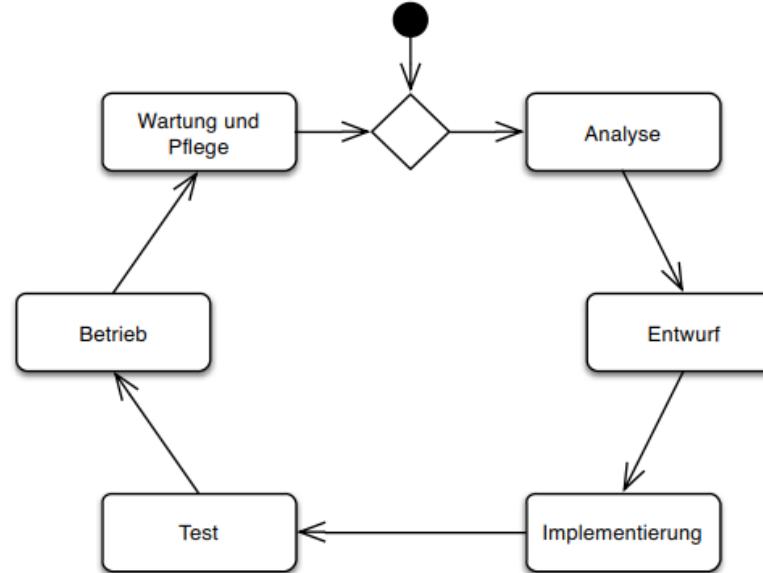
# Der Anfang ...

Die erste Entscheidung ist maßgeblich für den Erfolg des Projektes

## Typische Frage zu Anfang eines Projekts

- Wie gehe ich bei der Software-Entwicklung am besten vor?  
→ Suche und nutze ein geeignetes Vorgehensmodell
- Muss ich bei der Software-Entwicklung alles mühsam von Hand machen?  
→ Nein, es gibt eine Menge nützlicher Werkzeuge, die einen unterstützen  
→ In dieser Vorlesungseinheit besprechen wir:
  - Vorgehensmodelle nochmals im Detail
  - Geeignete Werkzeuge zur Projektunterstützung

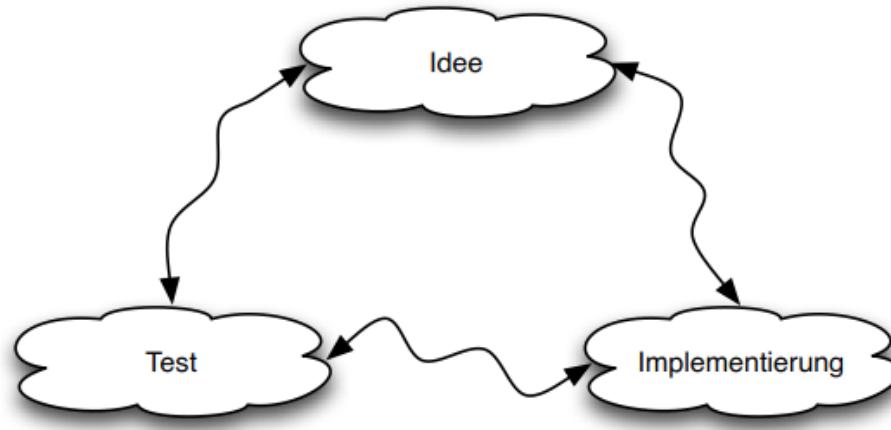
## BSp für Vorgehensmodelle – Lebenszyklus von Software



→ Beschreibt die typischen Tätigkeiten bei der SW-Entwicklung

- einfache, klare Struktur
- sehr schlichtes Vorgehensmodell
- wichtigste Aussagen:
  - Software-Entwicklung umfasst 6 grundlegende Tätigkeiten
  - zyklisch

## Bis jetzt arbeiten Sie so:



→ Sehr beliebt aber problematisch:

- Streng genommen kein Vorgehensmodell (zu allgemein)
- Verwandte Ansätze/Andere Namen:
  - Versuch und Irrtum (Trial and Error)
  - Hacken
- Sehr beliebt
- Manchmal sinnvoll: Neues ausprobieren, Prototypen, ...
- Sonst: problematisch
  - wächst einem schnell über den Kopf

# Technische Infrastruktur

Ziel:

Geeignete Werkzeuge zur Unterstützung in Projekten  
kennen und nutzen

# Technische Infrastruktur – Wofür brauchen wir das?

- Was meint technische Infrastruktur?
  - Sammlung an Werkzeugen (Programmen), die bei der SW-entwicklung helfen
  - Arbeiten hoffentlich so zusammen, dass Sie eine wirkliche Infrastruktur bilden
    - Möglichst keine Brüche
- Wofür ist das wichtig?
  - ▶ Effizientes Arbeiten & Strukturierung
  - ▶ Kollaboratives Arbeiten im Team
  - Bewältigung der aus den verschiedenen Artefakten und der daran arbeitenden Menschen entstehenden Komplexität

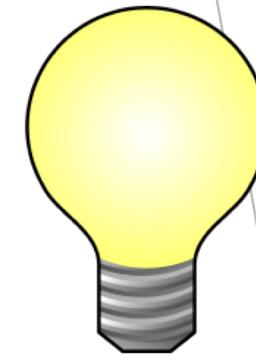
# Technische Infrastruktur

- Die wichtigsten Werkzeuge in einem Entwicklungsprojekt:
  - Integrierte Entwicklungsumgebung (Eclipse IDE, NetBeans, ...)
  - Testautomatisierung (JUnit, NUnit, PHPUnit, . . . )
  - Versionsverwaltung (Subversion (svn), git (derzeit sehr populär))
    - Wichtig für Konfigurationsmanagement (siehe folgendes Kap.)
  - Build-Automatisierung (make, ant, gradle, ...)
  - Continuous Integration Server (Cruise Control, Trac)
  - Issue-Tracking (Bugzilla, Mantis, . . . )
  - Projektmanagement (Open Workbench, MS Project, . . . )
- Empfehlung:
  - Nicht zu viele **neue** Werkzeuge auf ein Mal
  - Lieber nach und nach einführen

# Versionsverwaltung

Ziel:

Nutzen von Versionsverwaltungssystemen kennenlernen  
→ Konfigurationsmanagement kennenlernen



# Versionsverwaltungswerkzeuge

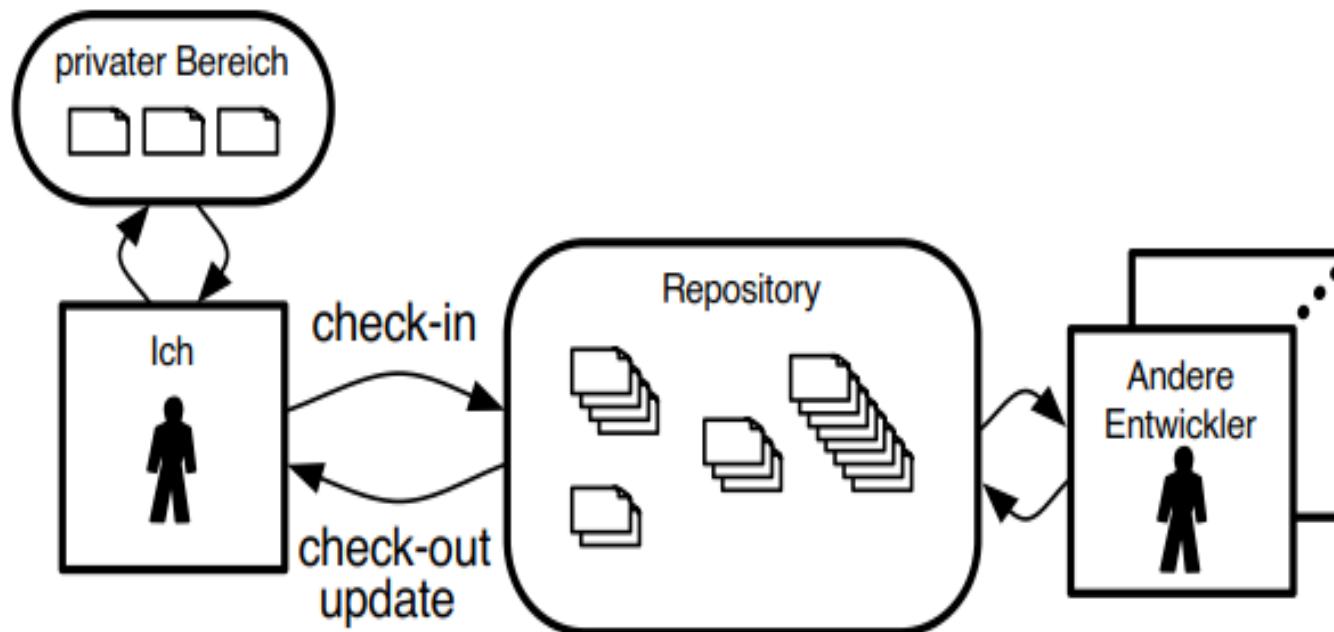
- Wofür brauchen wir eine Versionsverwaltung?
  - Backup-Restore
    - Wiederherstellen eines Standes, falls etwas kaputt gegangen ist
  - Nachvollziehbarkeit von Veränderungen
    - Stand der Datei X vor dem letzten Release
  - Parallele Entwicklung ermöglichen
  - Rekonstruktion vorheriger Konfigurationen
    - z.B. Quellcode-Dateien für Binaries, die mit Release XY geliefert wurden.

→ Versionsverwaltung ist **essentieller Bestandteil einer professionellen Softwareentwicklung**

- ▶ Nur Amateure & Bastler arbeiten ohne!!

# VersionsVerwaltungswerkzeuge

- Bekannte Versionsverwaltungen:
  - Subversion (SVN) – Open Source, immer noch oft verwendet
  - Git – Open Source + neuere -revolutionäre- Konzepte
- Grober Aufbau einer Versionsverwaltung:

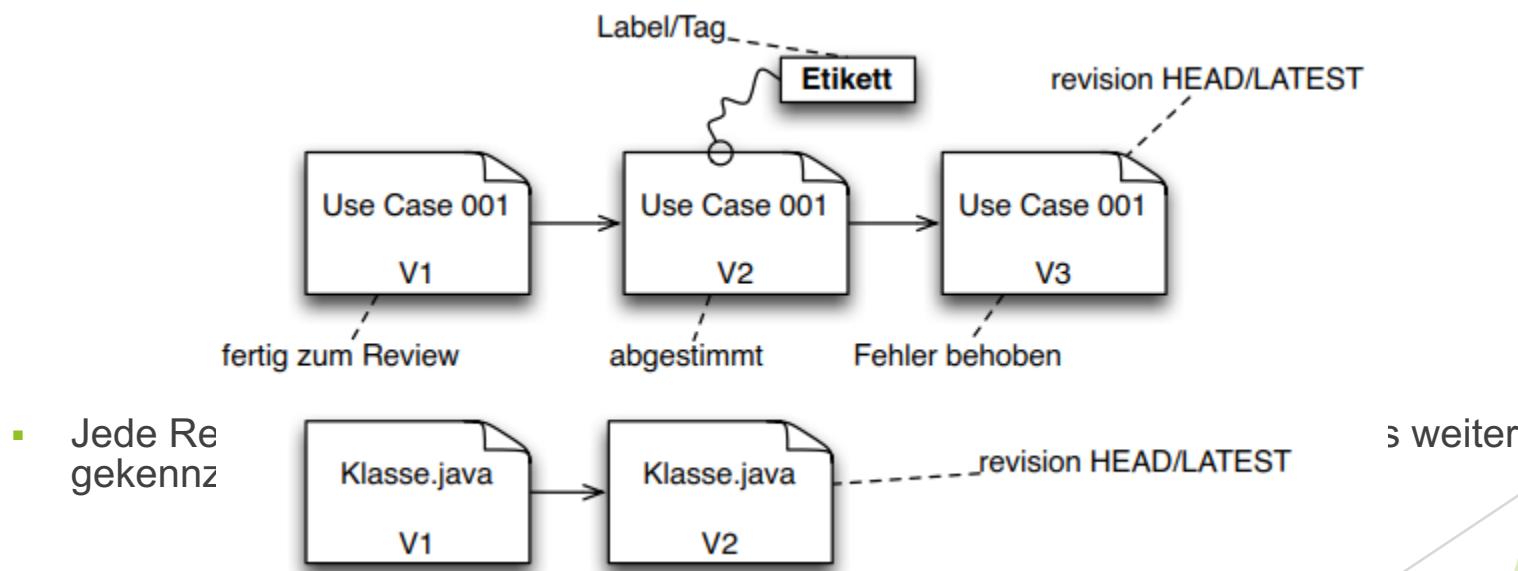


# VersionsVerwaltungs werkzeuge

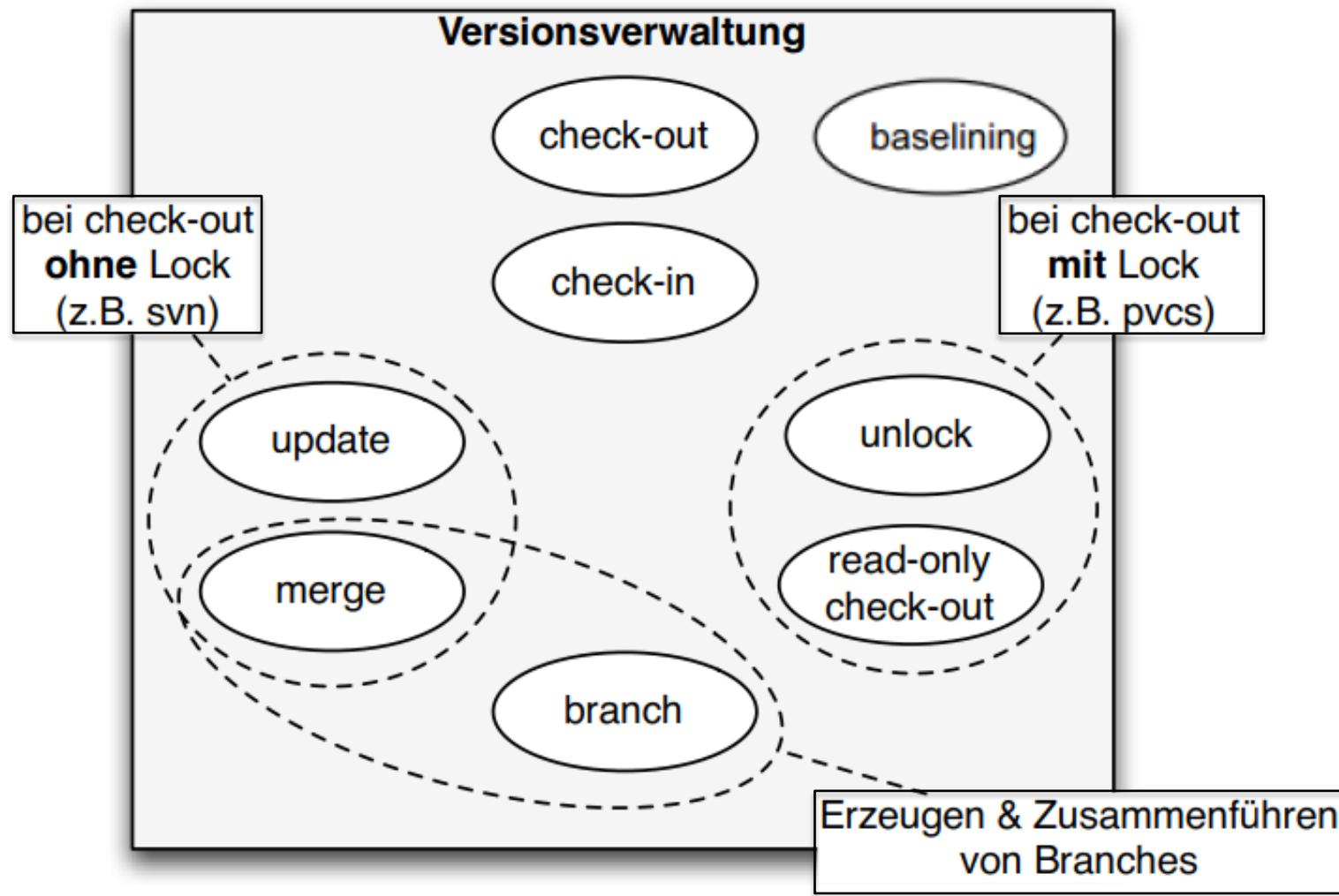
- Grundlegende Anforderungen:
  - Transparenz der Änderungen
    - Veränderung: Wer? Wann? Was?
  - Simultaner Zugriff + Konfliktlösung
  - Rekonstruktion
    - jederzeit beliebige Revision eines beliebigen Artefakts
    - jederzeit beliebige Konfiguration
  - Baselining / Labeling
    - Einfrieren von Ständen verschiedener Dateien zueinander (z.B. alle Stände der Dateien, die bei einem Release betroffen sind)  
→ Bekommen ein sog. Label

# Wie funktioniert die Versionierung von Dateien?

- Für jede Datei / Verz. wird ein Repräsentant (Head) angelegt
- Für jede Version der Datei /V wird eine sog. Revision erzeugt
  - Die Revision enthält die Datei in der jeweiligen Version
  - Daraus entsteht dann jeweils ein Historienbaum für jede Datei:



# VersionsVerwaltung – Grundlegende Use Cases



# Konfigurationsmanagement

- Was sollte versioniert werden?
  - Alle Dateien, die mit einem Projekt zu tun haben
    - ▶ Code, Tests, Anforderungen, Designdokumente, Sonstige Texte, Möglichst alle Entwicklungswerkzeuge (haben auch Versionen!)
- Zu bestimmten Zeiten müssen alle Dateistände zueinander eingefroren werden → Baselines
  - ▶ Zu definierten Meilensteinen (meist vor und nach dem Review, ...)
  - ▶ Zu bestimmten Meilensteinen müssen evtl. nur bestimmte Dateien zueinander versioniert werden (sog. Konfiguration)
- Zu freigegebenen Releases müssen evtl. noch Bugfixes, Patches, Service Packs nachgereicht werden
  - Sog. Branching erforderlich → Verzweigung der Entwicklung
  - Branches müssen später wieder in den Hauptzweig gemergt werden

# Konfigurationsmanagement



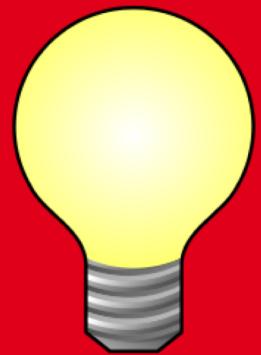
- Puh, das ist ja doch ganz schön kompliziert!
- Deshalb gibt es in größeren Projekten meist mindestens einen Verantwortlichen, der hier den Überblick behält
  - Prozess wird als Konfigurationsmanagement bezeichnet
- Typische Tätigkeiten des Konfigurationsmanagements:
  - Konfigurationsmanagementplan entwickeln
    - Welche Dateien müssen versioniert werden?
    - Wann müssen welche Baselines gezogen werden?
    - Wie verläuft ein Branching Prozess?
    - ...
  - Baselining durchführen, Probleme in der Versionsverwaltung lösen
  - Bei Branches den Rückmerge sicherstellen
  - ...

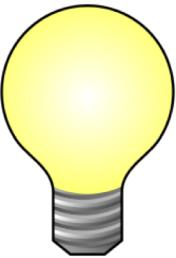


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

12.11.2020  
UML allgemein

Einführung in die UML





Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# AGENDA

Einführung ins Thema

Was ist UML?

Wie geht's jetzt mit der UML weiter?

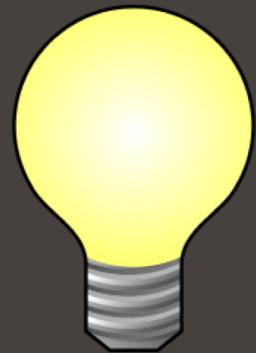


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# 01

## EINFÜHRUNG INS THEMA

Ziel:  
Die Eckpunkte des Themas kennenlernen



# MODELLIERUNGSSPRACHEN



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Bei der Softwareentwicklung haben wir es häufig mit
    - einem großen Anwendungssystem
    - und vielen KollegInnen zu tun
  - Wichtige Fragen bei der praktischen Durchführung:
    - Wie können wir uns auch über verzwickte Sachverhalte austauschen?
    - Wie können uns möglichst viele verstehen?
    - . . .
- Grafische Modellierungssprachen, z.B. UML
- **Verständliche** Darstellung auch **komplexer/komplizierter** Sachverhalte durch **Grafiken (Zeichnungen)**
  - **standardisiert**



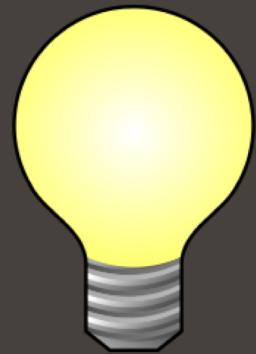
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

02

## Was ist UML?

Ziel:

Die UML grundsätzlich kennenlernen

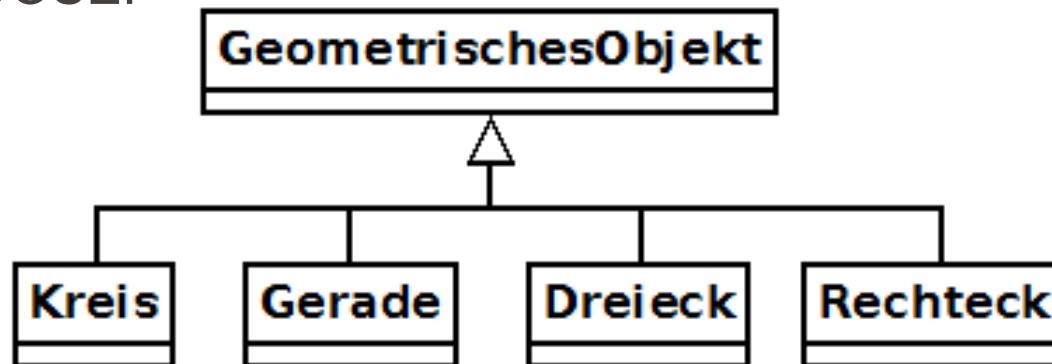




# WAS IST UML?

= Unified Modeling Language

- Standard der OMG
  - OMG = Object Management Group - <http://www.omg.org/>
  - Aktuelle Version: UML 2.5
  - > 10 Diagrammarten
- Ziel: Graphischen Modellierung objektorientierter Softwaresysteme  
→ Mittlerweile de-facto Standard für Modellierung von Software aller Art
- BSP aus OOSE:



# WAS IST UML?

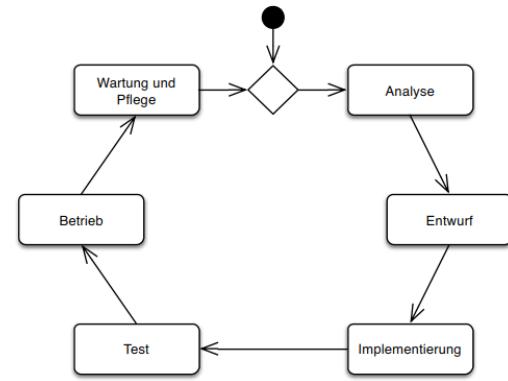


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

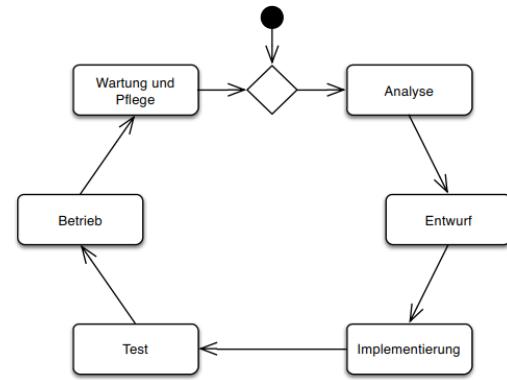
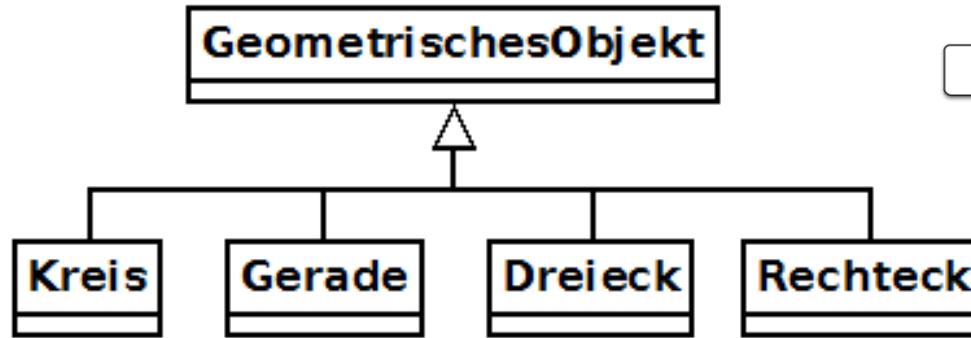
- Für uns:
  - Grundlegende Modellierungssprache
  - Später lernen wir jedoch auch noch FMC kennen
    - Hat gewisse Vorzüge in der Architekturmodellierung
  - Aber mit UML kann man das eigentlich auch machen

# UML-DIAGRAMME SIND VIELSEITIG VERWENDBAR

- Typische Verwendungsarten:
  - Anforderungsspezifikation
  - Analyse
  - Entwurf
  - Implementierung (Detailed Design, Nachträgliche Dokumentation)
  - Test (Design eines Testszenarios)
- Das kann verwirrend sein:
  - Diagramm-Syntax ist unabhängig von der Verwendungsart
    - Syntax (= Regelsystem für Form/Aussehen)
  - Diagramm-Semantik ist abhängig von der Verwendungsart
    - Semantik (= Bedeutung)



# BEISPIEL: ARTEN VON KLASSENDIAGRAMMEN



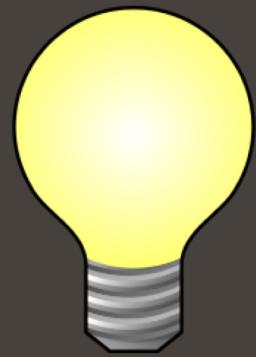
- Implementierungs-Klassendiagramm
    - (fast) gleichwertig zu Quellcode
  - Entwurfs-Klassendiagramm
    - 1:1-Beziehung zum Quellcode (hoffentlich!)
    - Aber: programmiersprachenunabh., oder unabh. von Details
  - Analyse-Klassendiagramm
    - Beschreibt fachliche und **keine** technischen Konzepte
- Gleiche Syntax für alle Arten von Klassendiagrammen
- Semantik (Bedeutung) abhängig von der Verwendungsart



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## 03 Wie geht's weiter

Ziel:  
Wie geht es jetzt weiter?



# UML-BÜCHER



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- H. Störrle: UML 2 für Studenten.  
→ Als Einführung geeignet – Leider teilw. nicht auf akt. Stand
- H.J. van Randen; et al.: Einführung in UML  
Gratis in Bib. verfügbar: <https://hds.hebis.de/hsrm/Record/HEB386572232>
- Ch. Rupp et al: UML 2 glasklar [BF 500 91]  
→ Für Fortgeschrittene geeignet – hilft auch bei kniffligen Fragen
- Spezifikationen im Original auf [www.uml.org](http://www.uml.org)  
→ Man muss wissen, dass es das gibt und wo man das findet.  
– Jedoch eher Nicht zum Lernen geeignet

Mein Tip

→ **Besorgen Sie sich ein Buch!**

# WAS MACHEN WIR JETZT IN DEN NÄCHSTEN WOCHEN?



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Heute: Objektdiagramm
  - Objekte und deren Beziehungen (Links)
- Nächste Woche: Klassendiagramm
  - Klassen und deren Beziehungen (Assoziationen)
  - Vererbung, ...
- Danach:
  - Verhaltensdiagramme
    - Zustandsdiagramme
    - Aktivitätsdiagramme
  - Interaktionsdiagramme
    - Sequenz- und Kommunikationsdiagramme
- Später:
  - UseCase-Diagramm



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

AUF GEHT'S!!

SELBER MACHEN UND LERNEN!!



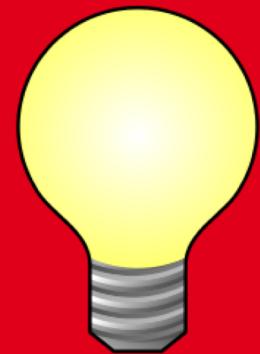


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

12.11.2020

# Das Objektdiagramm

Objektdiagramme nutzen





# AGENDA

Einführung ins Thema

Darstellung von Objekten

Verbindungen (Links)

Spezialfall: Konstanten

Weitere Hinweise

Fazit

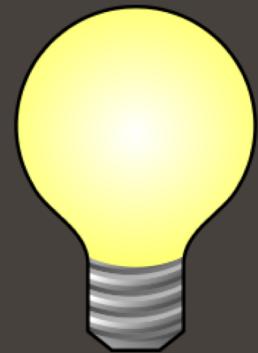


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# 01

## EINFÜHRUNG INS THEMA

Ziel:  
Die Eckpunkte des Themas kennenlernen





# WAS SIND OBJEKTE?

1. Laut Duden: „Objekt“ ist aus dem Lat.  $\triangleq$  „Gegenstand“
  - Objekt  $\triangleq$  Ding mit Eigenschaften und Fähigkeiten

2. Beim Programmieren (z.B. Java):

Klasse:

$\leftrightarrow$  Objekt:

```
public class Person {  
  
    String nachname;  
    int groesse;  
  
    public Person(String nname  
                 , int groess){  
        nachname= nname;  
        groesse=groess;  
    }  
}
```

- Definitionscode unveränderlich
- Im statischen Speicher

```
Person t= new Person("Turban", 185);
```

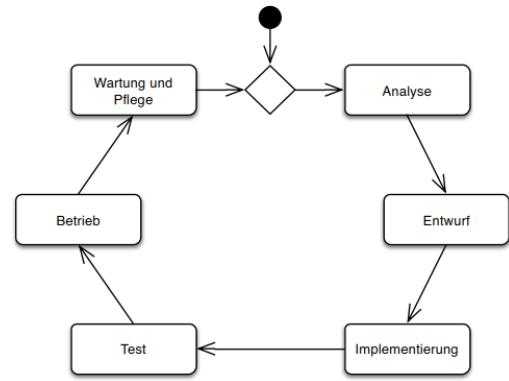
→ Konkrete Instanzen einer Klasse

→ Objektwerte, Anzahl Objekte, ...  
sind veränderlich

→ Im dynamischen Speicher

„Turban“  
185

# WAS SIND OBJEKTE IN DER UML?



1. Laut Duden: „Objekt“ ist aus dem Lat.  $\triangleq$  „Gegenstand“
  - Objekt  $\triangleq$  Ding mit Eigenschaften und Fähigkeiten

→ Es gibt eine Person mit den Eigenschaften

Nachname == “Turban” und

Grösse == 185 cm

} Fachliches Konzept  
( $\triangleq$  Business Object)  
z.B. in **Anforderungsbeschreibung** oder in der **Analyse**

→ Unabhängig von Programmierung oder einer sonstigen technischen Umsetzung

→ **Fachliche Sicht**

# WAS SIND OBJEKTE IN DER UML?

2. Beim Programmieren (z.B. Java):

Klasse:                           ↔      Objekt:

→ Ein Objekt der Klasse

```
public class Person {  
    String nachname;  
    int groesse;  
    ...  
}
```

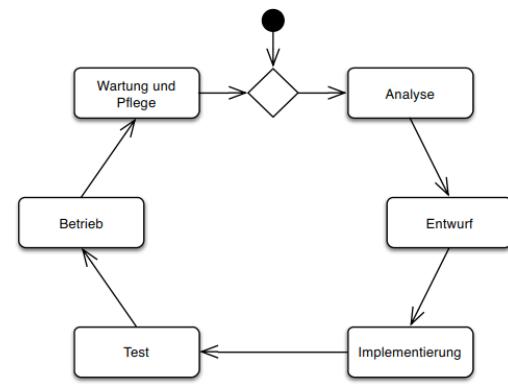
mit Attributbelegungen

„Turban“  
185

Konkretes  
Technisches Konzept  
z.B. in **Design**  
oder  
**Implementierung**

→ Abhängig von Programmierung oder einer sonstigen  
technischen Umsetzung

→ **Technische Sicht (Implementierung)**

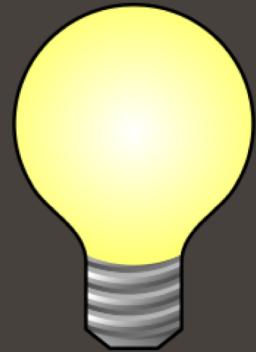




## 02

# Darstellung von Objekten

Ziel:  
Eine (mögliche) Darstellung für **beide Sichten**



# (MÖGLICHE) DARSTELLUNG FÜR BEIDE SICHTEN



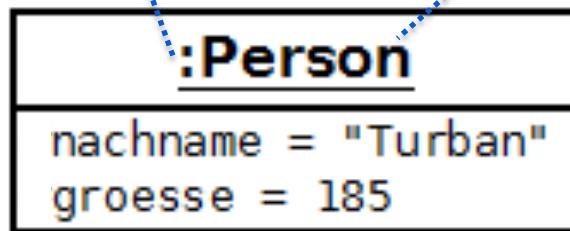
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

Immer mit : und unterstrichen!

→ Kennzeichnet Objekt  
im Gegensatz zu Klasse

Art des Objekts  
(Classifier, Class Type)

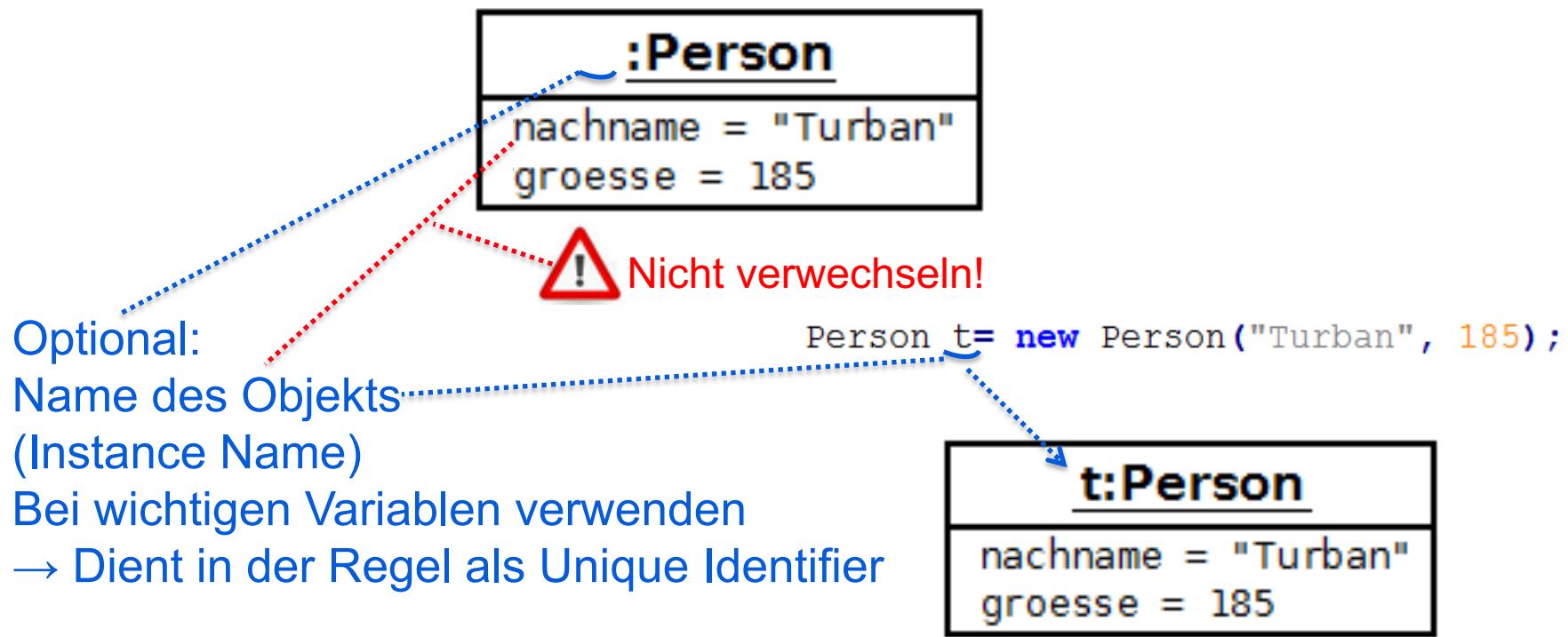
Objekt (Instanz)



# (MÖGLICHE) DARSTELLUNG FÜR BEIDE SICHTEN



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim



**BEM:**

Bis auf **:** ist alles optional

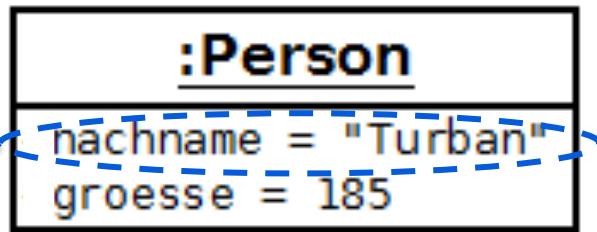


Minimum, aber macht das Sinn?

# (MÖGLICHE) DARSTELLUNG FÜR BEIDE SICHTEN



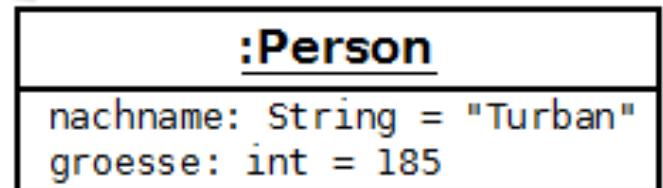
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim



Konkrete Eigenschaft (Slot)

→ Besteht aus:

- i) Name der Eigenschaft (Feature Name)
- ii) Optional: Typ der Eigenschaft (z.B. :String)
- iii) Belegung / Wert (Value Specification)

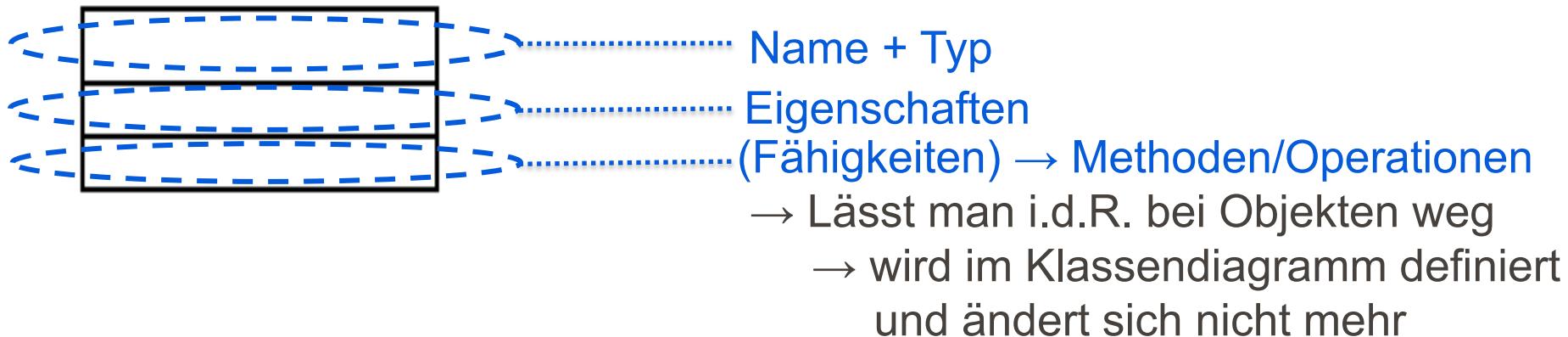


# WEITERE BEMERKUNG ZUR DARSTELLUNG



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Ein Objekt kann maximal drei Abschnitte haben:
  - (vgl. auch Klassendiagramm kommende Woche)



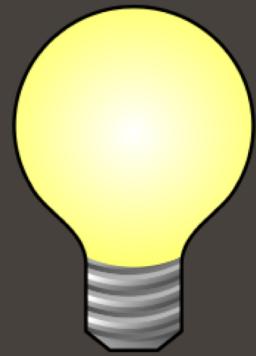


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# 03

## Verbindungen (Links)

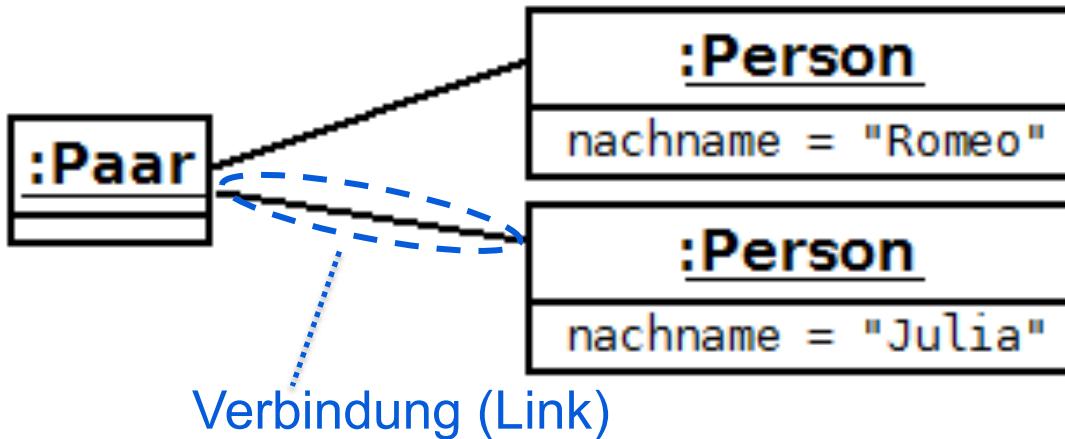
Ziel:  
Verbindungen zwischen Objekten darstellen





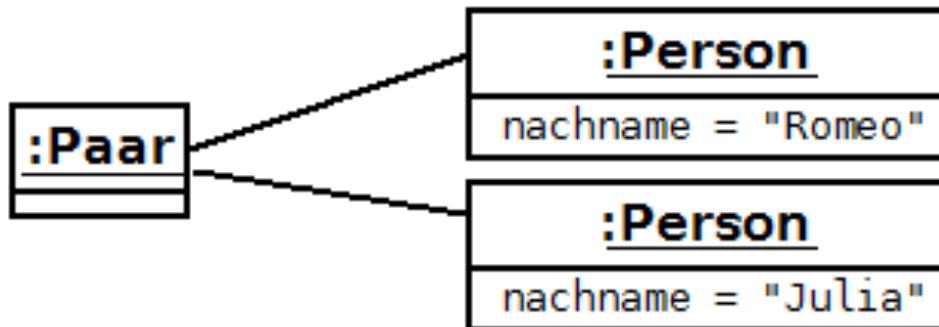
# VERBINDUNGEN (LINKS)

- Objekte haben miteinander zu tun
  - Es gibt Verbindungen, die wir modellieren können müssen





# VERBINDUNGEN (LINKS)



Mögliche Bedeutungen:

## 1. Fachliche Sicht:

- Wir haben ein Paar, das aus der Person mit Namen „Romeo“
- und der Person mit Namen „Julia“ besteht

## 2. Technische Sicht:

```
public class Person { ... }  
  
public class Pair {  
    Person pers1;  
    Person pers2;  
}
```

Oder:

```
public class Pair {  
    Person [] pers= Person[2];  
}
```

Oder:

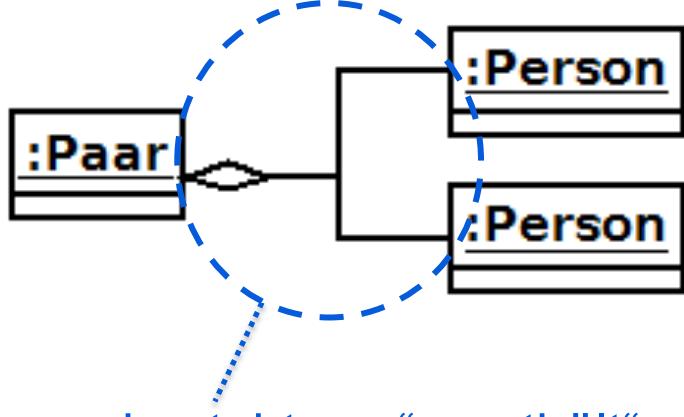
...

# VERBINDUNGEN (LINKS) – AGGREGATION & KOMPOSITION

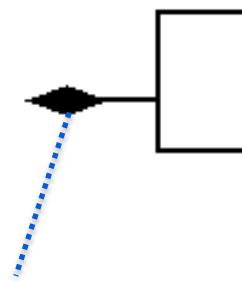


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Betonung des Aspekts „besteht aus“:  
(v.a. in Analysediagrammen\* nützlich)



„besteht aus“, „enthält“  
(Aggregation)



„besteht aus“, „enthält“  
(Komposition)  
Stärker als

\* Diagramme, die im Zuge der Analysephase gemacht werden  
→ Siehe Fachmodell in Vorl. 07 Anforderungsanalyse

# VERBINDUNGEN (LINKS) – AGGREGATION & KOMPOSITION



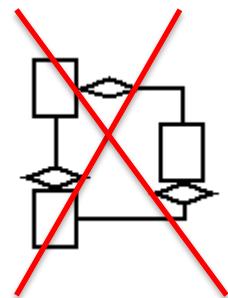
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim



Vorsicht: UML definiert keine präzise Bedeutung für



- Lediglich:
  - Für beide sind keine zyklischen Abhängigkeiten erlaubt
  - ist irgendwie stärker als
    - Inoffizielle Unterscheidung:  
  $\triangleq$  Gesamtteil kann nicht ohne Einzelteile existieren



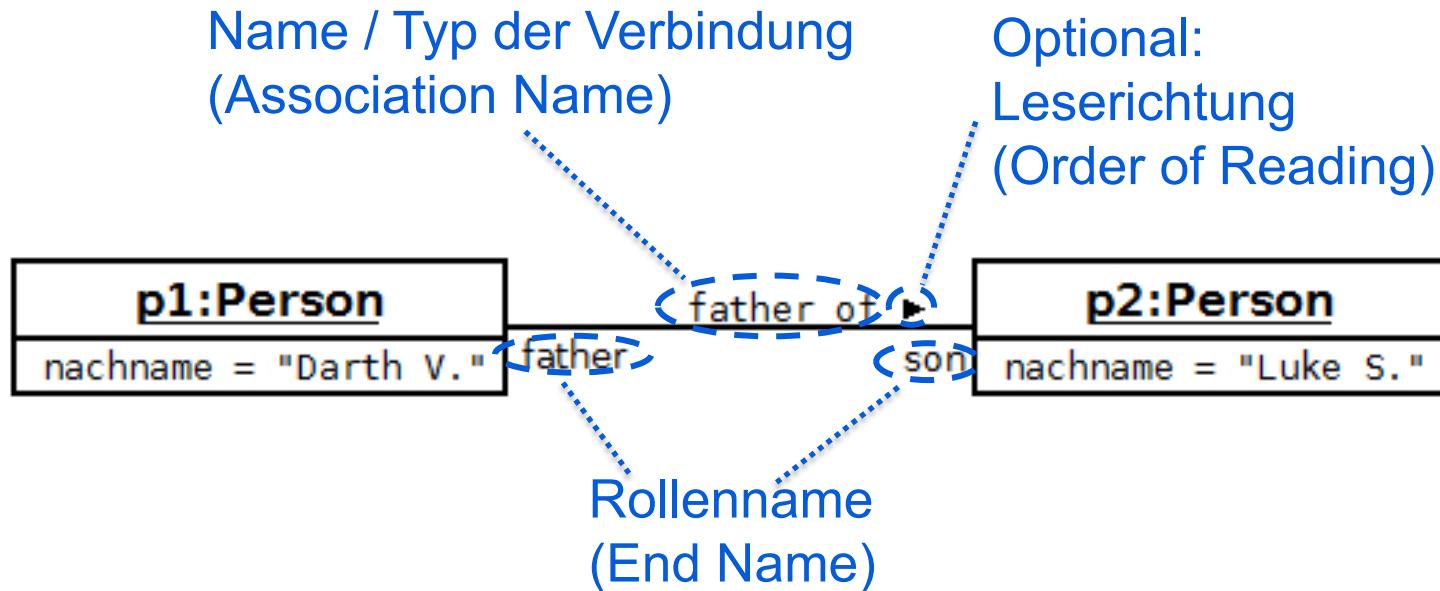
→ TIPP: Nur verwenden, wenn WIRKLICH notwendig

# WEITERE INFORMATIONEN ZU OBJEKTVERBINDUNGEN (LINKS)

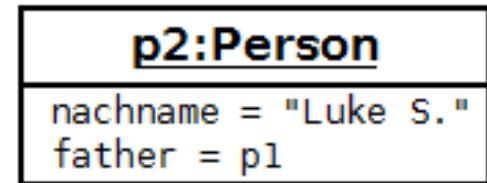
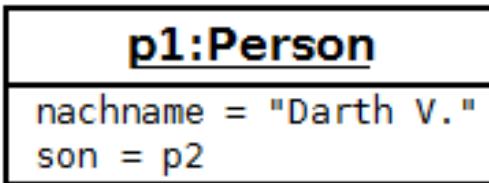


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Zu Objektverbindungen (Links) können weitere Infos hinterlegt werden:



Auch möglich, aber nicht besonders schön:



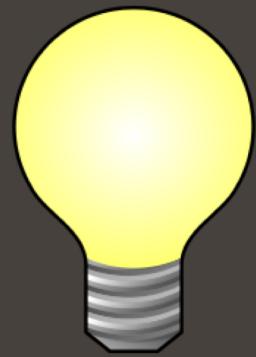


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## 04

# Spezialfall: Konstanten

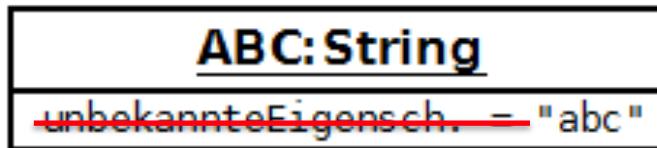
Ziel:  
Konstanten darstellen



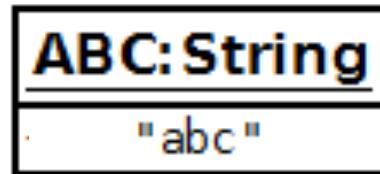
# SPEZIALFALL: KONSTANTEN UND PRIMITIVE OBJEKTE



- Konstanten (Werte), oder primitive Objekte, oder berechenbare Werte darstellen:
  - Objekt mit einer Eigenschaft, deren Name nicht bekannt ist



- Da es nur einen Wert gibt, kann man dann auch den Eigenschaftsnamen weglassen:





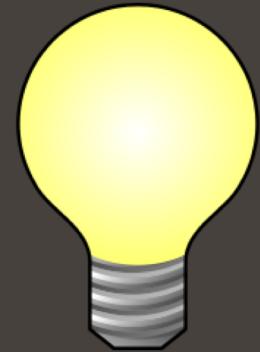
# 05

## Weitere Hinweise

Ziel:

Die Bedeutung des Objektdiagramms

Wie hängen fachliche und technische Sicht zusammen?



# DIE BEDEUTUNG DES OBJEKTDIAGRAMMS



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Das Objektdiagramm stellt nur einen bestimmten Schnapschuss (Momentaufnahme) dar
  - Wenig später ist vielleicht folgendes passiert:
    - Neue Objekte erzeugt
    - Neue Verbindungen zwischen Objekt entstanden
    - Es wurden einige Objekte gelöscht
    - Einige Verbindungen zwischen den Objekten wurden gelöst
  - Die neue Situation ist ganz anders
  - Man müsste ein neues Diagramm zeichnen
- Mit dem Klassendiagramm kann ein allgemeingültiges Diagramm aller möglichen Objektkonstellationen zeichnen
- Definiert alle möglichen Objektdiagramme

# WIR ERINNERN UNS AN DEN ANFANG:



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## Programmieren:

### Klasse:

```
public class Person { ... }
```

- Definitionscode unveränderlich
- Im **statischen** Speicher



### Objekt:

```
Person t = new Person("Turban", 185);
```

- Konkrete Instanzen einer Klasse
- Objektwerte, Anzahl Objekte, ... sind veränderlich
  - Im **dynamischen** Speicher

„Turban“ 185

### UML:

- Klassendiagramm
- Eine **statische** Sicht
- UML ermöglicht viele verschiedene Sichten (sog. Sichtenkonzept)

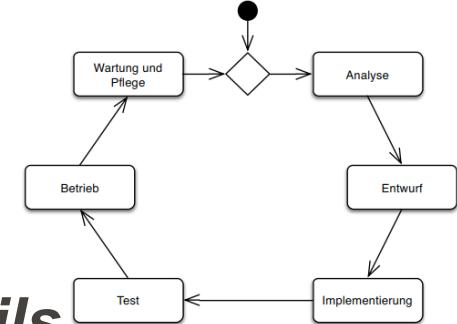
- Objektdiagramm
- Eine **dynamische** Sicht



## WEITERE SICHTEN:

1. Laut Duden: „Objekt“ ist aus dem Lat.  $\triangleq$  „Gegenstand“
  - Objekt  $\triangleq$  Ding mit Eigenschaften und Fähigkeiten
    - ***Fachliche Sicht***
2. Beim Programmieren (z.B. Java)
  - ***Technische Sicht***

# FACHLICHE ↔ TECHNISCHE SICHT



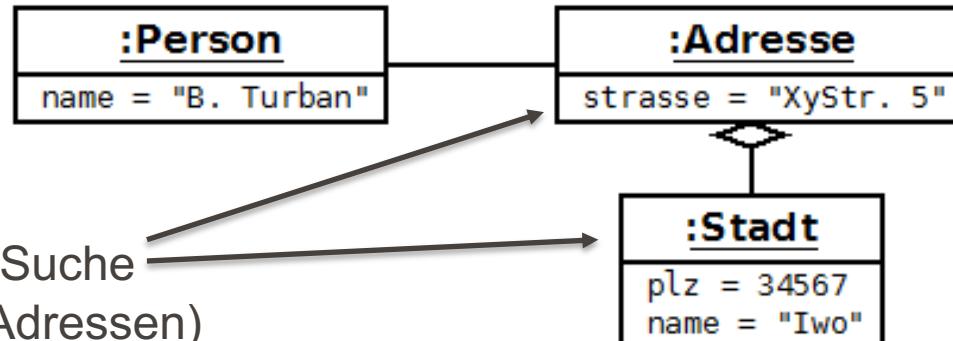
→ ***Fachliche Sicht: alles ohne technische Details***

→ Wird in der Analyse verwendet, um **NUR** die fachlichen Anforderungen zu ermitteln

:Person	
nachname	= "Turban"
groesse	= 185
adresse	= "XyStr. 5, 34567 Iwo"

→ ***Technische Sichten***

- In Architektur, Detailed Design, Implementierungsdoku. verw.
- Es werden auch die für die Lösung **relevanten** technischen Details dargestellt:



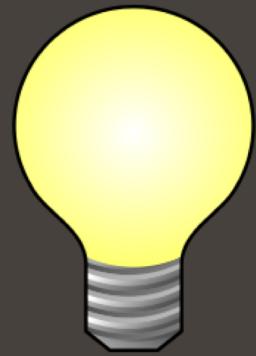
- + Technische Lösung  
(z.B. für spezifische Suche nach Städten oder Adressen)



# 06

# Fazit

Ziel:  
Was haben wir damit gewonnen?



# WAS HABEN WIR GELERNT?



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Verschiedene Sichten
  - Fachliche Sicht (Nur fachliche Zusammenhänge aus Sicht des Kunden)  
→ Anforderungsanalyse
  - Technische Sicht (Technische Umsetzung)  
→ Design & Implementierung
  - Beide können unterschiedlich zueinander sein
- Objektdiagramm
  - Objekte
    - Rechteckiger Kasten mit max. 3 Unterteilungen
    - Instanzname : Typname, Wertbelegungen
  - Verbindungen
    - Einfache Links, Aggregation, Komposition
    - Verbindungsname, Richtung, Rollennamen
  - Das Objektdiagramm stellt nur einen Schnappschuß dar



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

AUF GEHT'S!!

SELBER MACHEN UND LERNEN!!

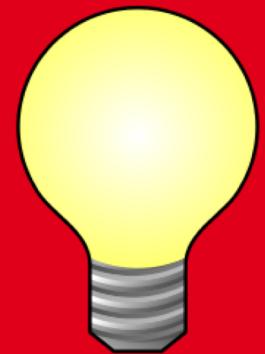


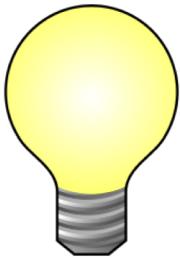


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# UML II - Klassendiagramm

Einführung in Klassendiagramme





Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# AGENDA

Einführung ins Thema

Grundlagen

Weitere Anmerkungen zu Assoziationen

Vererbung

Attribute, Methoden, Sichtbarkeiten

Zwei mögliche Bedeutungsarten

Fazit

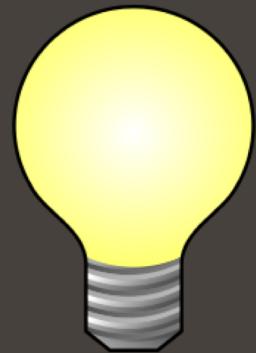


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# 01

## EINFÜHRUNG INS THEMA

Ziel:  
Die Eckpunkte des Themas kennenlernen



# DAS KLASSENDIAGRAMM



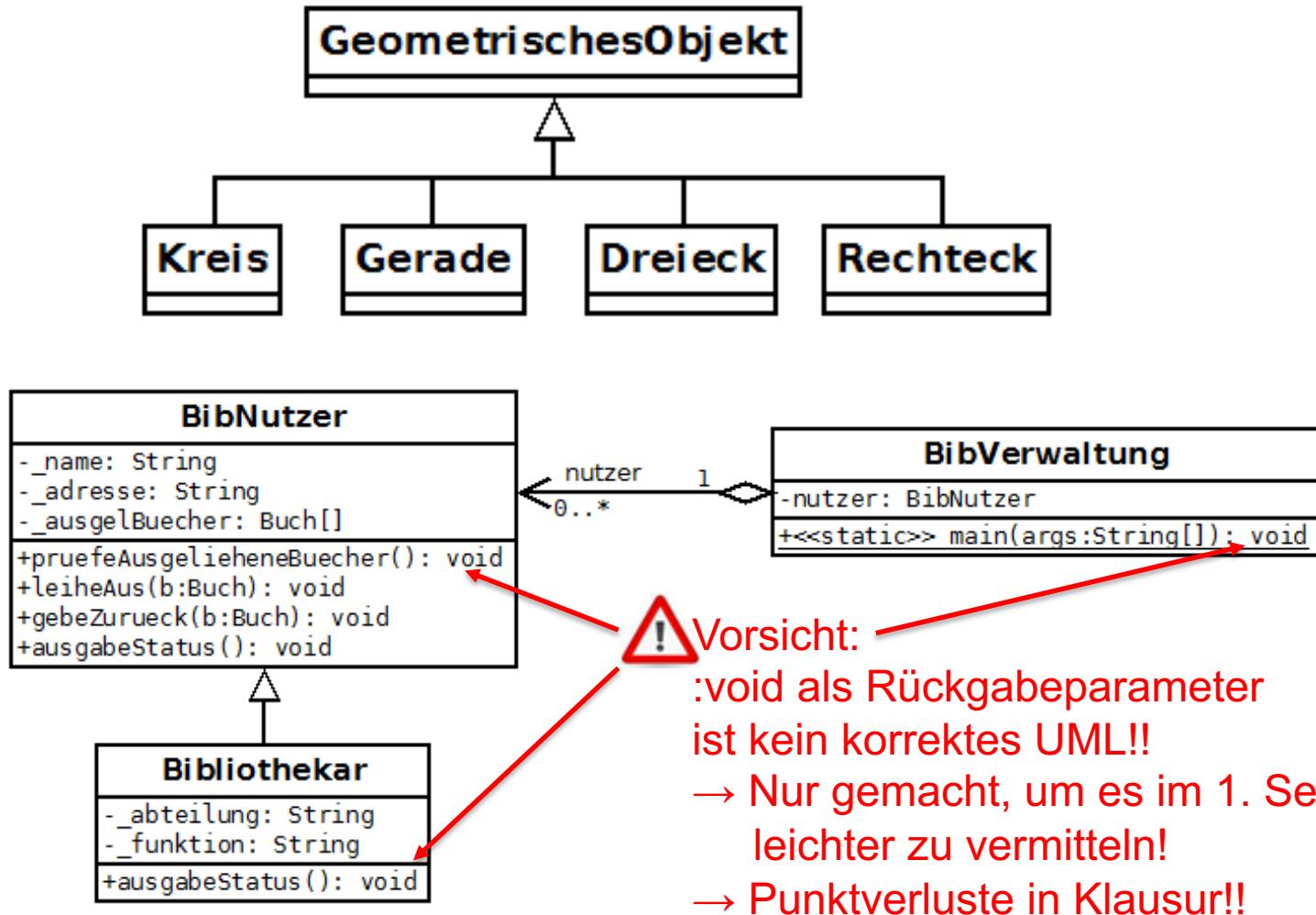
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Neben dem Objektdiagramm
  - Objekte und deren Beziehungen (Links)
  - Schnappschuß einer bestimmten Situation zu bestimmter Zeit
  - dynamischen Sicht
- Gibt es das Klassendiagramm
  - Klassen und deren Beziehungen (Associations)
  - Allgemeine Definition der Klassen und aller möglichen Beziehungskonfigurationen durch Assoziationen
  - statische Sicht

# BEISPIELE AUS DER OOSE – VORLESUNG



- Das Klassendiagramm habe ich in OOSE bereits verwendet:

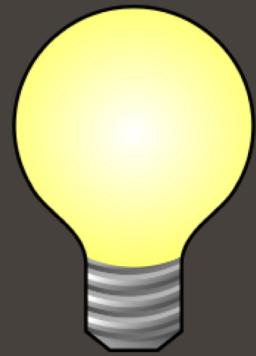




Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## 02 LOSLEGEN

Ziel:  
Die Grundlagen kennenlernen

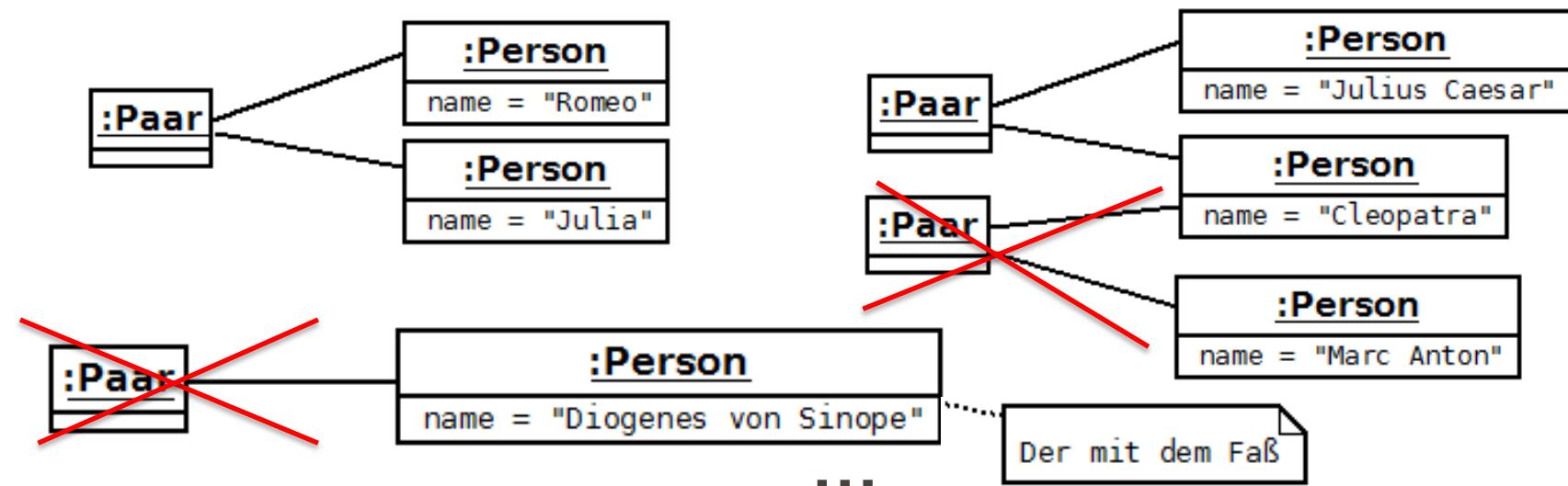


# BSP: DEFINITION PAARE UND DEREN BEZIEHUNGEN



1. Es gibt nur Paare, die mit 2 Personen verbunden sind
2. Eine Person kann höchstens mit einem Paar verbunden sein

→ Man kann nun einige Objektdiagramme zeichnen:

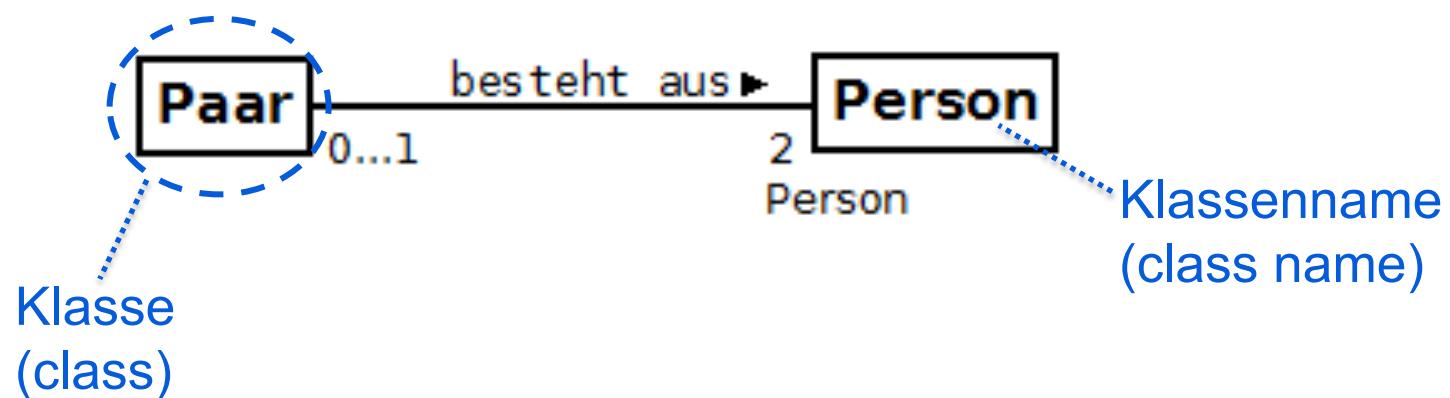


→ Nur unvollständige Auflistung von richtigen & falschen Beispielen möglich



# DAS KLASSENDIAGRAMM

- Wir brauchen eine vollständige Darstellung der Sachverhalte
- Klassendiagramm:



Vergleiche die Darstellung!

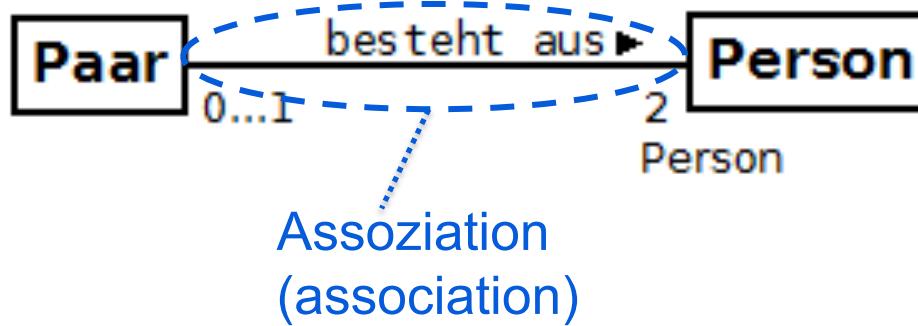
Klasse: **Person**  
Objekt: **:Person**

# DAS KLASSENDIAGRAMM – ASSOZIATIONEN



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Verbindungen zwischen Klassen → Assoziationen:

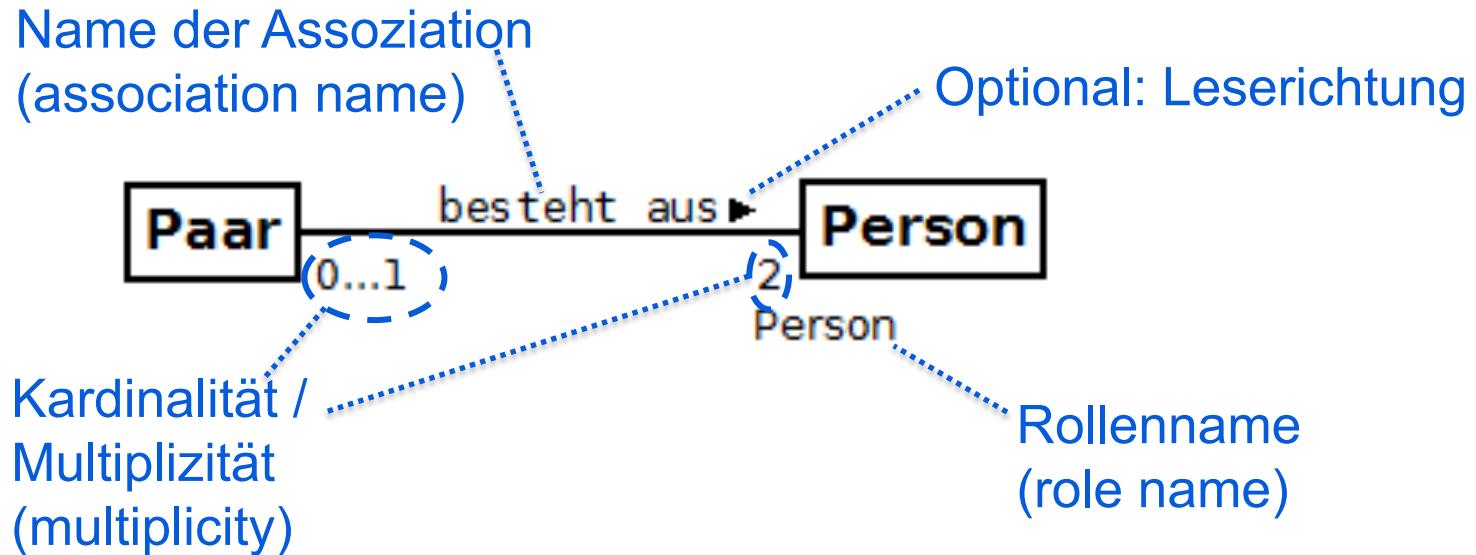


BEM: Assoziation  $\triangleq$  Relation im Entity-Relationship-Diagramm

# DAS KLASSENDIAGRAMM – ASSOZIATIONEN



- Weitere Infos zu einer Assoziation:



## Kardinalitäten:

- Allgemeine Form: Untergrenze...Obergrenze  
 $\in \mathbb{N}$        $\in \mathbb{N} \cup \{\ast\}$        $\hat{=} \infty$  / beliebig

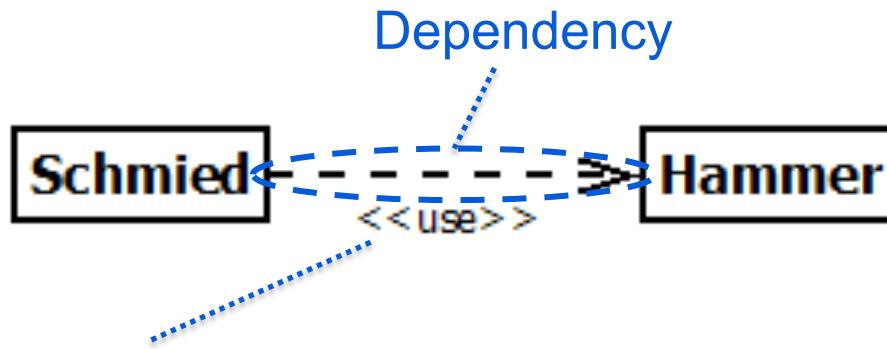
- Spezialformen:  
 $a \dots a$  |  $a$  → nur/genau a (Bsp: 7...7 → genau 7)  
<leer> → genau 1 !

# DAS KLASSENDIAGRAMM – DEPENDENCY\*



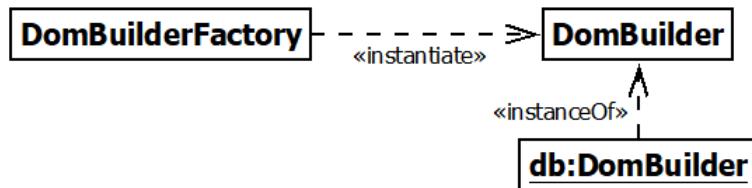
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Neben Assoziationen gibt es noch Dependency-Pfeile  
→ Für einfache Abhängigkeitsbeziehungen



Stereotyp (Stereotype) «...» = Erweiterungsmechanismus der UML

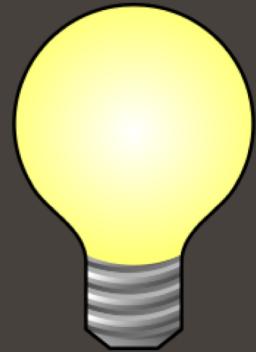
- Genauere Definition (Eingrenzung) der Abhängigkeitsbeziehung
- Typische Stereotypen für Dependencies\*: «use», «import», «create», «call», « abstraction», «instanciate», «instanceOf», ...
  - Eigene Stereotypen möglich





03

## Weitere Anmerkungen zu Assoziationen



Ziel:

Weitere Details zu Assoziationen besprechen

# KLASSENDIAGR.: ASSOZIATION ↔ OBJEKTDIAGR.: LINKS

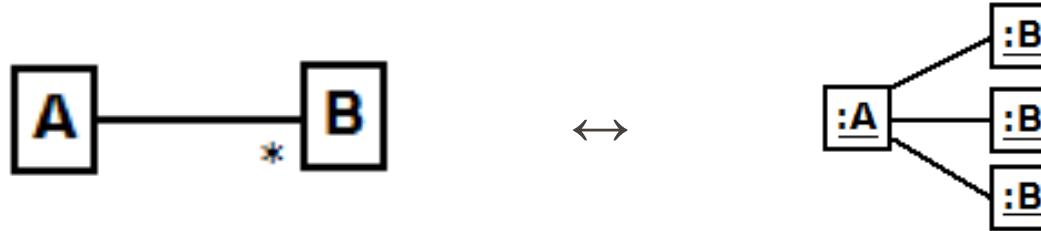


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Weitere Bemerkungen zu Assoziationen:

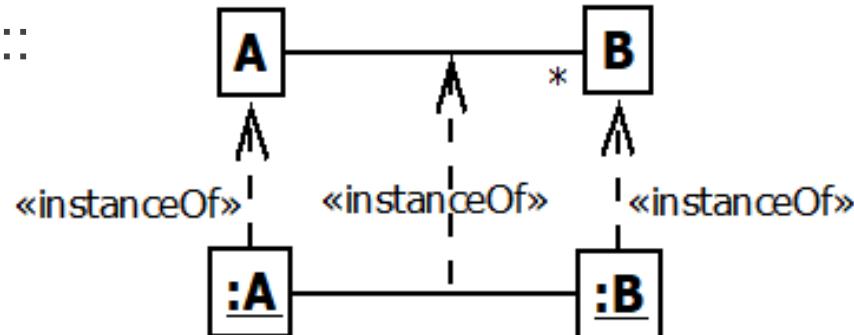
- Eine **Assoziation** im Klassendiag. (KD):

entspricht **evtl. n Links** in einem Objektdiag. (OD):



- Beziehung in der UML:

Klassendiag.::



Objektdiagr.::

# KLASSENDIAGR.: ASSOZIATION ↔ OBJEKTDIAGR.: LINKS

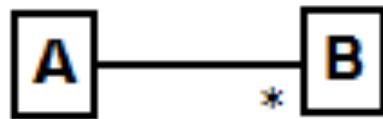


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

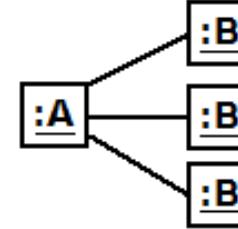
- Weitere Bemerkungen zu Assoziationen:

- Eine **Assoziation** im Klassendiag. (KD):

entspricht **evtl. n Links** in einem Objektdiag. (OD):



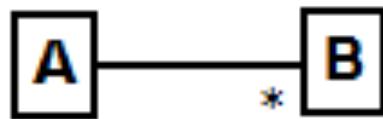
↔



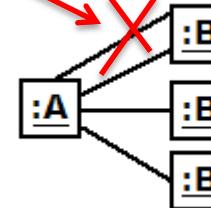
- 1 “normale” Assoziation ist auch nur 1 Link zwischen 2 Objekten



Es kann dann **nicht 2** geben!

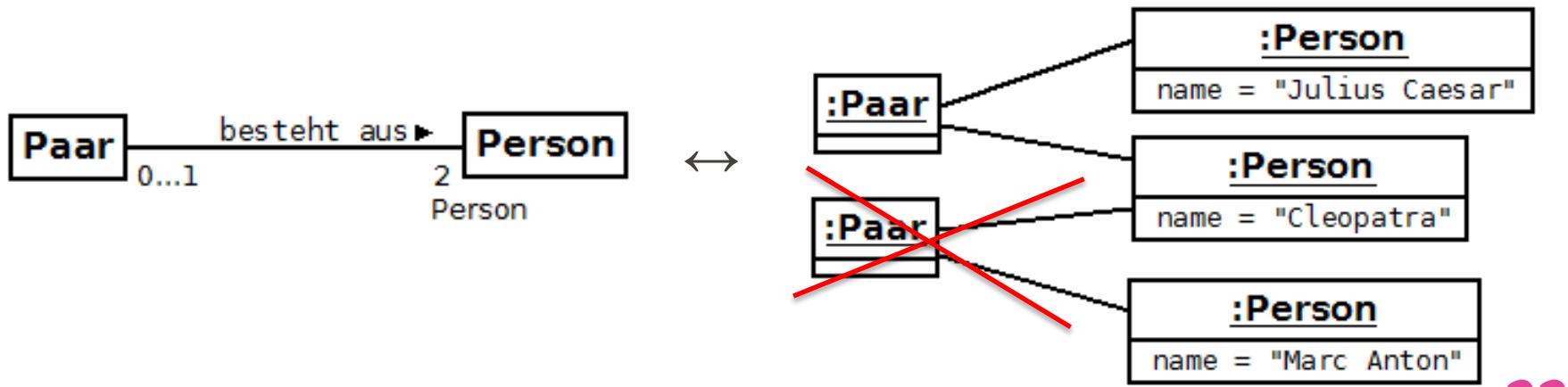


↔





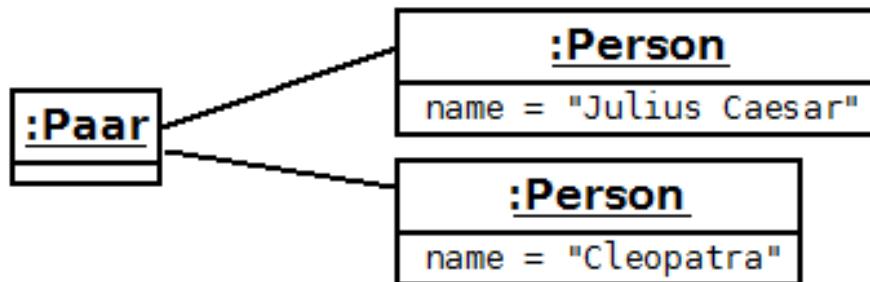
# WEITERE BEMERKUNG:



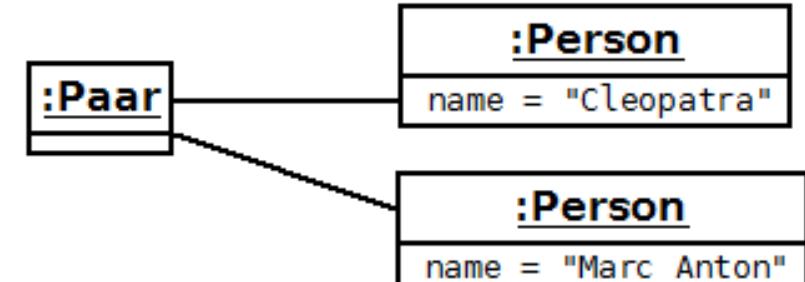
- Info. ist iwie richtig. → Wie gehört das zusammen?



Situation vor Cäsars Tod:



Einige Jahre nach Cäsars Tod:



→ Schnappschußcharakter der Objektdiagramme!

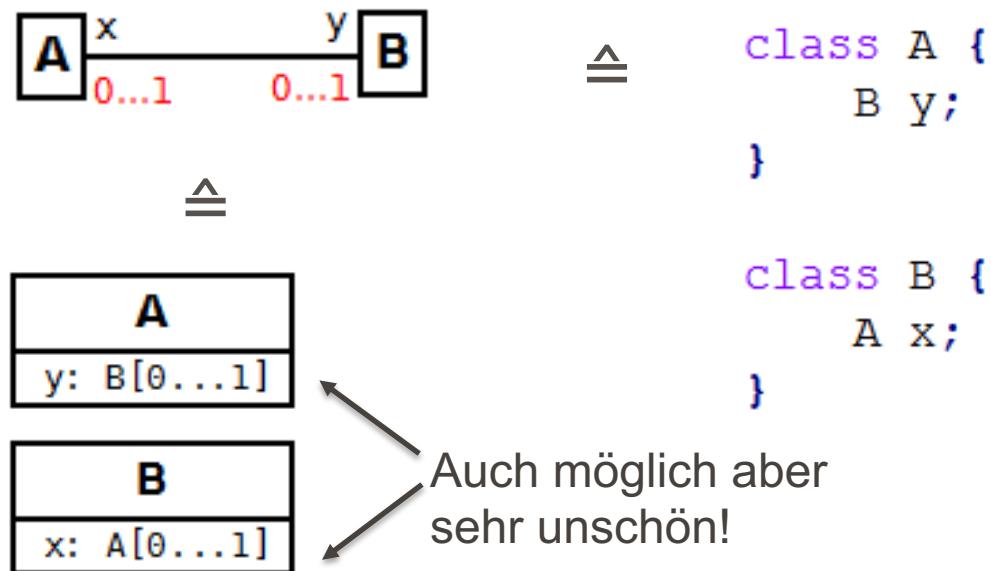
# WAS IST ZUSAMMENHANG ZW. ASSOZIATIONEN UND CODE?



In Programmiersprachen (Java, ...) gibt es keine direkte Entsprechung für das Konzept “Assoziation”

→ Wie umsetzen?

- Möglichkeit A:

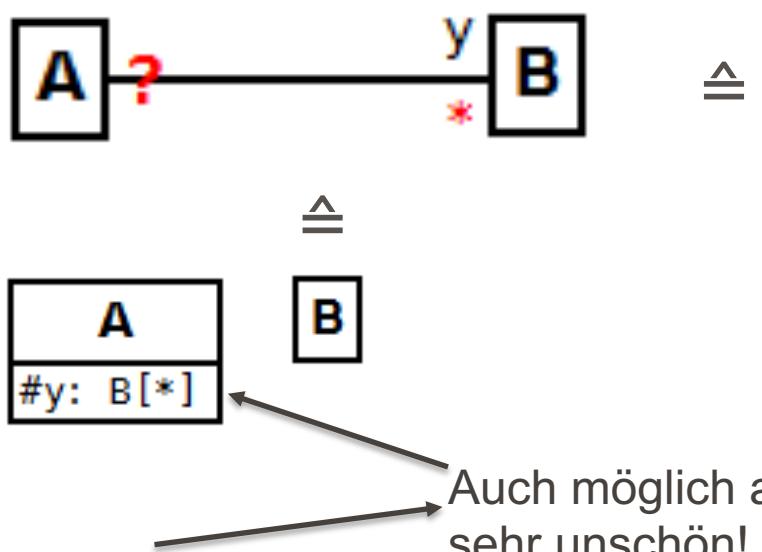


# WAS IST ZUSAMMENHANG ZW. ASSOZIATIONEN UND CODE?



In Programmiersprachen (Java, ...) gibt es keine direkte Entsprechung für das Konzept “Assoziation”

- Wie Umsetzen?
- Möglichkeit A:



```
class A {  
    B[] y;  
//ODER:  
    Collection<B> y;  
//ODER:  
    List<B> y;  
//ODER:  
    ...  
}
```

Auch möglich aber  
sehr unschön!

→ Empfehlung:

- Primitive Datentypen (Zahl, Bool, String, Enum)
- Sonst

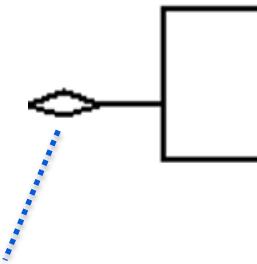
→ Attribut  
→ Assoziation

# AGGREGATION & KOMPOSITION IM KLASSEN- & OBJEKTDIAGRAMM

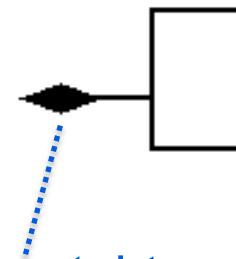


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Betonung des Aspekts „besteht aus“:  
(v.a. in Analysediagrammen\* nützlich)



„besteht aus“, „enthält“  
(Aggregation)



„besteht aus“, „enthält“  
(Komposition)  
Stärker als



Klassendiagr.: Aggregation & Komposition sind Untertypen von  
Assoziation

Objektdiagr.: Aggregation & Komposition sind Untertypen von  
Link

\* Diagramme, die im Zuge der Analysephase gemacht werden  
→ Siehe Fachmodell in Vorl. 07 Anforderungsanalyse

# (WIEDERHOLUNG) AGGREGATION & KOMPOSITION



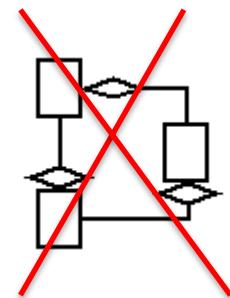
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim



Vorsicht: UML definiert keine präzise Bedeutung für



- Lediglich:
  - Für beide sind keine zyklischen Abhängigkeiten erlaubt
  -  ist irgendwie stärker als 
    - Inoffizielle Unterscheidung:  
  $\triangleq$  Gesamtteil kann nicht ohne Einzelteile existieren



→ TIPP: Nur verwenden, wenn WIRKLICH notwendig

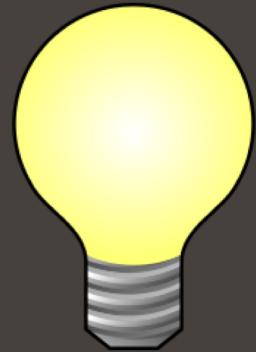


# 04

## Vererbung

Ziel:

Wie kann Vererbung dargestellt werden?



# VERERBUNG



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

Notation Vererbung in UML:



→ Drück aus: B erbt von A

- Hier gibt es auch eine extensionale und eine intensionale Bedeutungsart

# VERERBUNG – AUSWIRKUNGEN AUF OBJEKTE & OBJEKTDIAGRAMM



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

Mögliche Notation für B-Objekte:



→ Typ-Info wird unspezifischer



Auch möglich:



⚠ Vererbung ist in der Regel im Objektdiagramm  
nicht (direkt) sichtbar!

**Typregel:** Die B-Objekte müssen auch die gleichen

- Eigenschaften
- Fähigkeiten

wie die A-Objekte haben

Mit anderen Worten: Jedes B-Objekt ist auch A-Objekt  
(ABER: Nicht jedes A-Objekt ist auch B-Objekt)

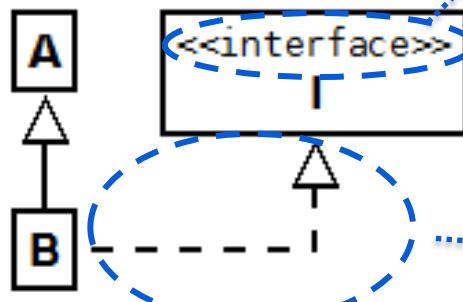


# MEHRFACHVERERBUNG



## UML erlaubt echte Mehrfachvererbung (wie C++)

- Damit UML auch kompatibel zu Sprachen wie C++ ist
- Man sollte Mehrfachvererbung in Designsichten vermeiden, wenn die Sprache das nicht unterstützt
  - Oder zumindest dann eine Beschreibung angeben wie diese realisiert wird
- “Erben” von Interfaces:



Stereotyp (Stereotype) «...»  
= Erweiterungsmechanismus  
der UML

Hier: gleiches Modellelement  
(rechteckiger Kasten) wie für Klasse,  
aber „Kasten“ hat mit «interface»  
andere Bedeutung: Interface

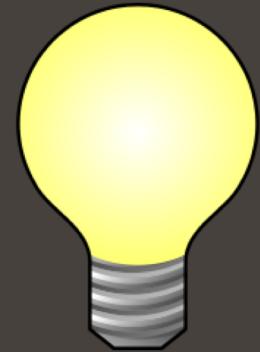
Implementierungspfeil /  
Realisierungspfeil





05

## Attribute, Methoden, Sichtbarkeiten

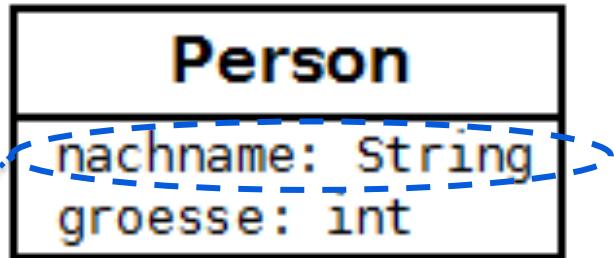


Ziel:

Den inneren Aufbau von Klassen detaillierter spezifizieren

# (MÖGLICHE) DARSTELLUNG FÜR BEIDE SICHTEN

- Analog wie in Objektdiagramm dargestellt:



Konkrete Eigenschaft (Attribut)

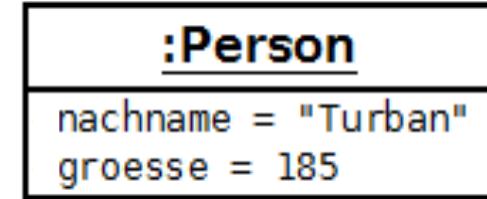
→ Besteht aus:

- Name der Eigenschaft (feature name)
- optional: Art der Eigenschaft (z.B. :String)
- optional: Initiale Belegung / Wert  
(Vorgabewert  $\triangleq$  default value specification)



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

Vgl. Objekt:



# WEITERE BEMERKUNG ZUR DARSTELLUNG



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Eine Klasse kann in der Regel\* maximal drei Abschnitte (Compartments) haben:
  - (vgl. auch Objektdiagramm vorletzte Woche)

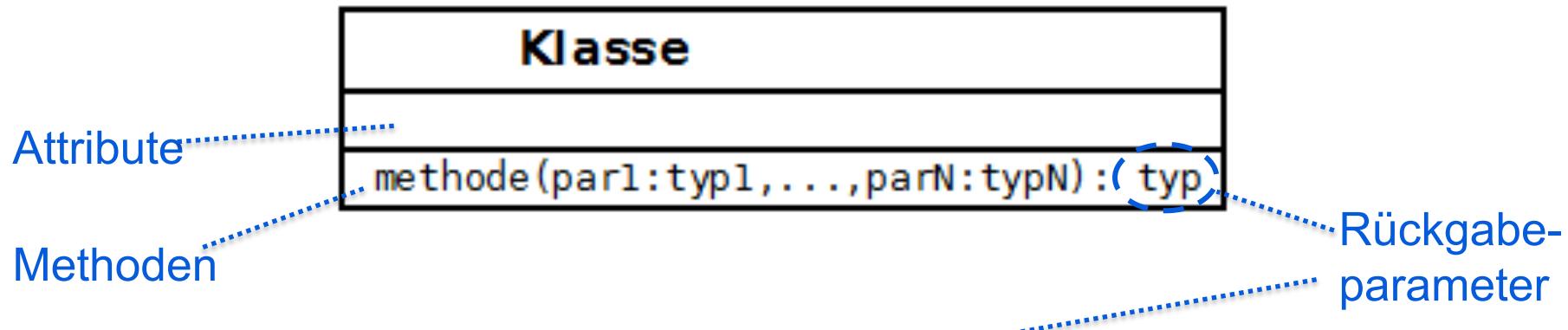


\* Theoretisch gibt die UML keine Vorgaben über Reihenfolge und Anzahl außer, dass oben der Name stehen muss → beliebige Anzahl und Reihenfolge  
Praktisch werden eigentlich nur diese 3 und eigentlich immer in dieser Reihenfolge genutzt



# METHODEN

- Übliche Namen: Methoden, Operationen, Member Functions
- Notation:



- BEM: void  $\notin$  UML  
typ == <leer> hat zwei Bedeutungen:  
void unbekannt
- BEM: Meth. v.a. bei Entwurfs-/Implementierungsdiagramm interessant



# SICHTBARKEITEN

- Für Attribute und Methoden können die Sichtbarkeiten definiert werden:

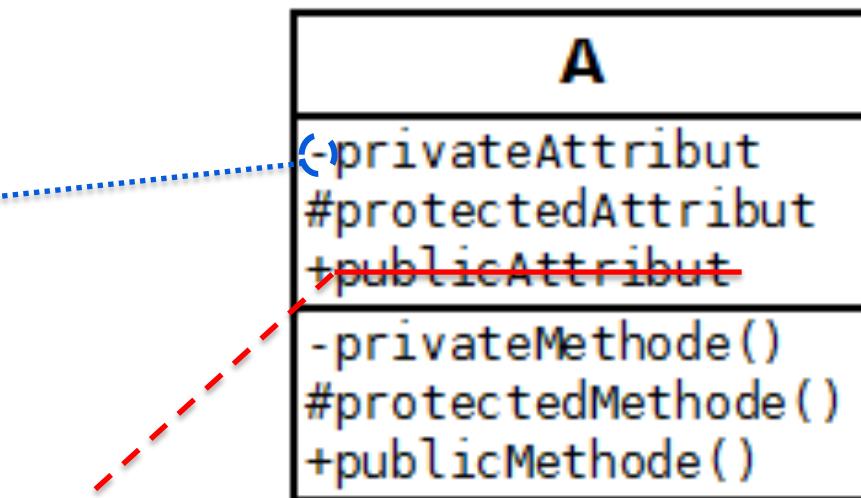
Sichtbarkeiten:

+ == public

# == protected

- == private

~ == package



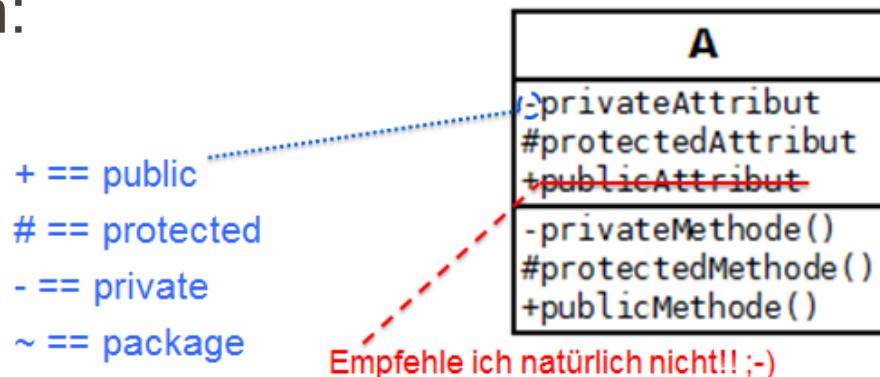
Empfehle ich natürlich nicht!! ;-)

- Ein unterstrichenes Attribut oder Methode bedeutet “static”



# SICHTBARKEITEN

- Für Attribute und Methoden können die Sichtbarkeiten definiert werden:



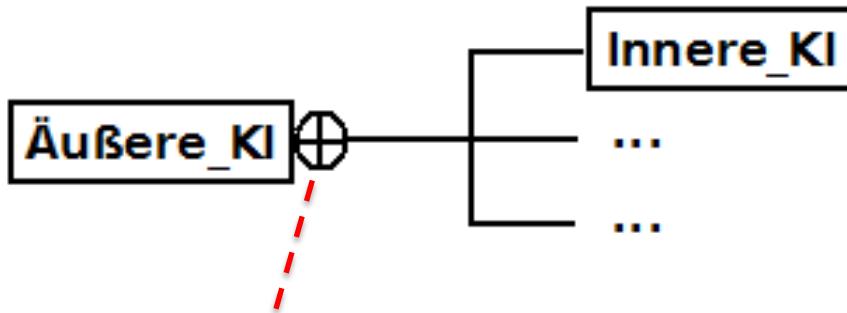
- Sichtbarkeiten können auch weggelassen werden
  - Wird in Objektdiagrammen üblicherweise weggelassen
  - Kann in Analysemodellen / Fachmodell auch weggelassen werden
  - In der Regel nur im Entwurfs-/Implementierungsdiagramm und im Zusammenhang mit Methoden interessant

**BEM:** Leider gibt es Werkzeuge, die Sichtbarkeiten immer anzeigen → Muss man dann einfach ignorieren ☹

# DARSTELLUNG INNERE KLASSEN



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

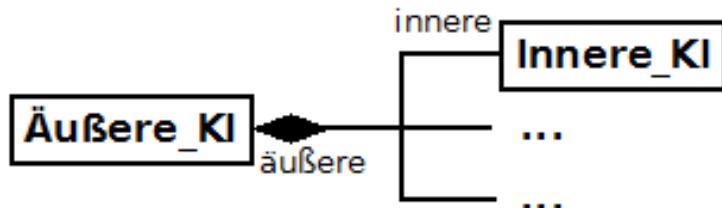


Nicht offiziell  
UML 2.x!

→ war Teil von UML 1.4, aber  
wird teilweise noch verwendet (auch im Standard)

```
class Äußere_Kl {  
    ...  
    class Innere_Kl {  
        ...  
    }  
}
```

- Ansonsten immer möglich (aber mehrdeutig!):

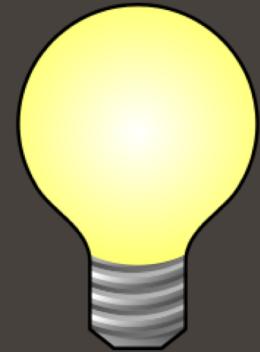


⚠ Komposition bedeutet nicht autom. innere Klasse!!  
(kann aber auf mögliche Lösung als innere Klasse hindeuten)



06

## Zwei mögliche Bedeutungsarten



Ziel:

Verstehen lernen, dass das Klassendiagramm verschiedene Bedeutungsarten annehmen kann

# GRUNDSÄTZLICH MÖGLICHE BEDEUTUNGSSARTEN



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Auf der Bedeutungsebene (Semantik) kann man mit einem Klassendiagramm zwei Bedeutungsarten abbilden:
  - I. Extensionale Bedeutungsart
    - △ Mengenorientiert
      - (Menge aller Objekte mit gleichen Eigenschaften)
  - II. Intensionale Bedeutungsart
    - △ Bauplanorientierte Bedeutung
      - (Gleicher Aufbau)

---

Hinweis: Beide Bedeutungsarten wurde von Aristoteles schon in der griech.

Antike zur Definition von Begriffen eingeführt und unterschieden.

Hierzu eine nähere Beschreibung:

[https://de.wikipedia.org/wiki/Extension\\_und\\_Intension](https://de.wikipedia.org/wiki/Extension_und_Intension)

# I. EXTENSIONALE BEDEUTUNGS-ART



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

Extensional:

Darstellung der

- Gemeinsamen Eigenschaften &
- Gemeinsamen Fähigkeiten

} = Gemeinsamkeiten

einer Menge an Objekten

→ 1 Klasse  $\hat{=}$  Menge gleichartiger Objekte

→ Mengenorientierte Darstellung

# I. EXTENSIONALE BEDART – BSP

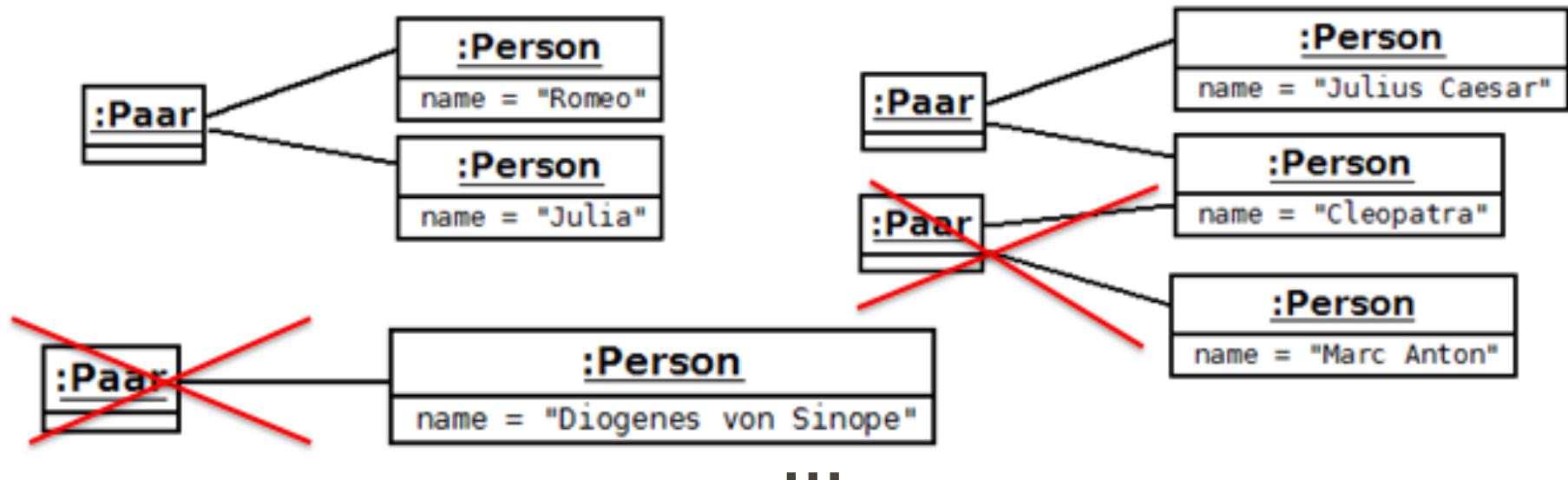
→ SIEHE BEISPIEL VON OBEN!



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

1. Es gibt nur Paare, die mit 2 Personen verbunden sind
2. Eine Person kann höchstens mit einem Paar verbunden sein

→ Man kann nun einige Objektdiagramme zeichnen:



→ Nur unvollständige Auflistung von richtigen & falschen Beispielen möglich

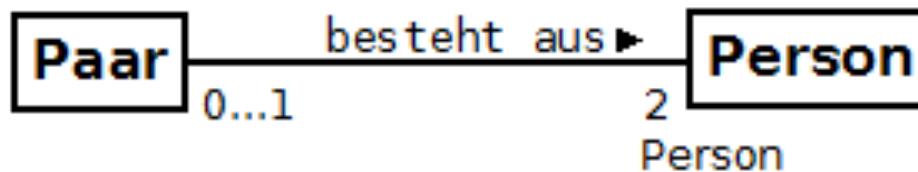
# I. EXTENSIONALE BEDART – BSP

## → DAS KLASSENDIAGRAMM



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Wir brauchen eine vollständige Darstellung der Sachverhalte
  - Klassendiagramm:



## II. INTENSIONALE BEDEUTUNGS- ART



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

Intensional:

Darstellung des Bauplans für ein(e) Art/Sorte/Typ/Klasse von Objekten

→ 1 Klasse  $\triangleq$  Bauplan für gleichartige Objekte

→ Bauplanorientierte Darstellung

## II. INTENSIONALE BEDEUTUNGS- ART – BSP



```
public class Person { ... }
```

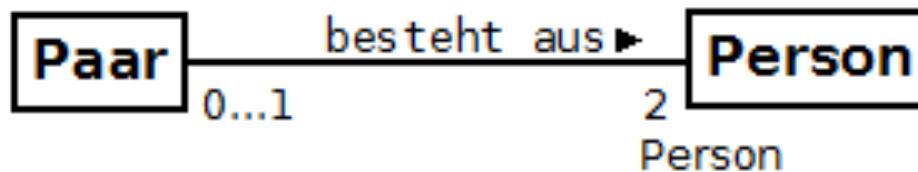
```
public class Paar { ... }
```

Gegebenenfalls weiterer Code

Quellcode stellt sicher, dass:

- Jedes Paar-Objekt genau mit 2 Person-Objekten verbunden ist.
- Jedes Person-Objekt mit höchstens einem Paar-Objekt verbunden ist

→ Klassendiagramm:



→ Selbes Diagramm wie in I. !!

Wie passt das jetzt zusammen?



# EXT. & INT. BEDEUTUNGSART – WIE PASST DAS JETZT ZUSAMMEN? \*

Hochschule RheinMain  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- UML legt nicht eindeutig fest welche Bedeutungsart ein Klassendiagramm hat
  - Die Bedeutungsarten können zusammenfallen
  - Müssen aber nicht
- Man kann sich zu jedem KD fragen
  - Ist es Extensional oder Intensional gemeint?
  - Oder beides?
- **Grundsätzliche Regel:**
  - Ein Diagramm sagt mehr als 1000 Worte
  - Kann aber auch tausenderlei verschieden interpretiert werden  
→ Auch textuelle Beschreibung eines Diagramms nötig

# VERERBUNG – I. EXTENSIONAL

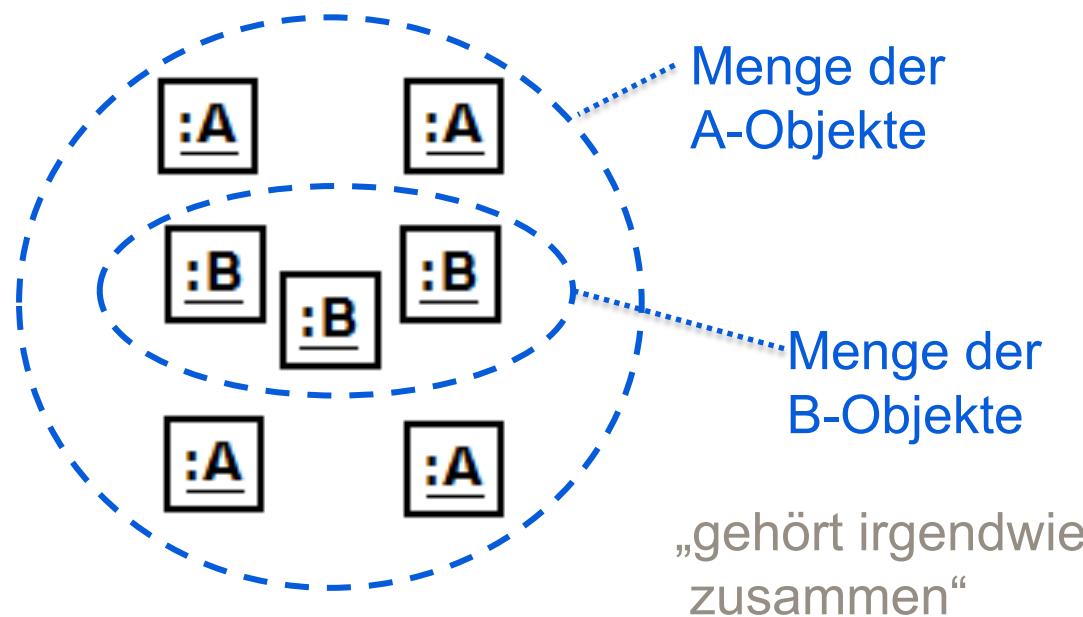
Meist im Analysediagramm

Bedeutungsart I. Extensional → Mengenorientiert:

A ist Oberklasse von B bedeutet:

Die Menge der B-Objekte ist eine  
Teilmenge der Menge der A-Objekte

Schematisch:

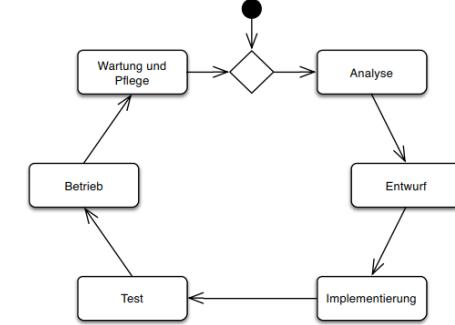


Notation  
Vererbung  
in UML:



# VERERBUNG – II. INTENSIONAL

Im Designdiagramm

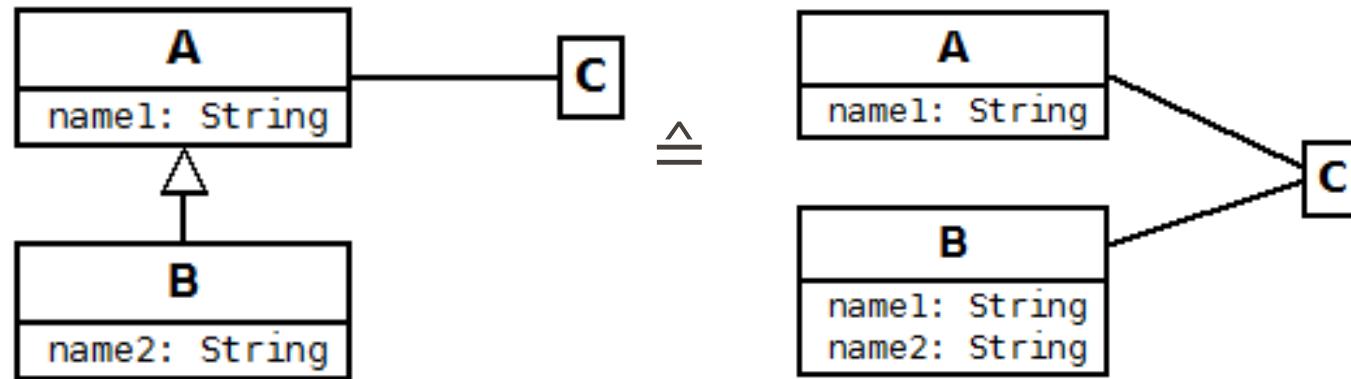


Bedeutungsart II. Intensional → Bauplanorientiert

A ist Oberklasse von B bedeutet:

Der Bauplan für B-Objekte umfasst auch den Bauplan  
für A-Objekte

Schematisch:



→ Nutzen von  $\uparrow$ :

Kurzschreibweise für die Wiederverwendung von Bauplänen

# EXT. & INT. BEDEUTUNGSART – BEISPIELE AUS DER REALEN WELT



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

1. Analyse: Aal und Schlange könnten beispielsw. als Beinlose zusammengefasst werden

 ABER: Aal ist ein Knochenfisch ↔ Schlange: ist ein Reptil!  
→ Interne Baupläne sind anders

→ Besseres Design: Aal erbt von Knochenfisch, Schlange von Reptil

2. Analyse: Delphine und Haie könnte man zu den Fischartigen zählen

 ABER: Delphine sind Säugetiere (näher mit Robben als mit Haien verwandt!)

→ Besseres Design: Delphin erbt von Säugetier, Hai von Knorpelfisch

(Biologie: Beispiele für Konvergente Evolution)

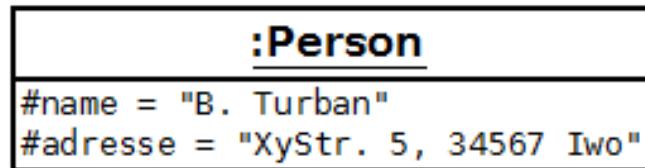
# LETZTE WOCHE: FACHLICHE ↔ TECHNISCHE SICHT



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

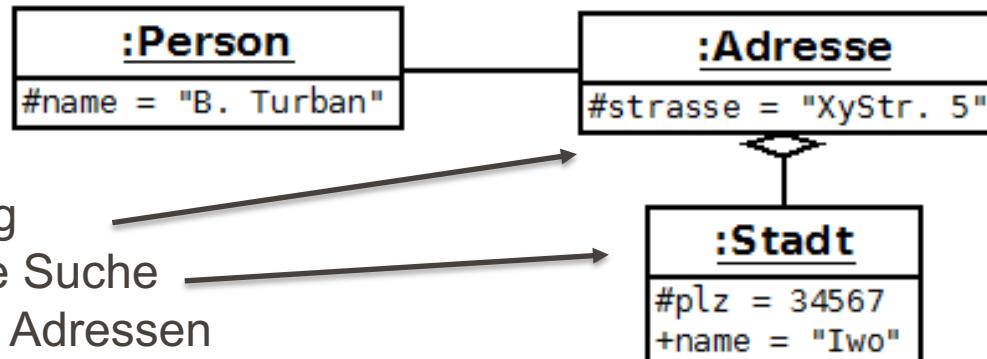
→ **Fachliche Sicht betrachtet alles ohne technische Details**

- Wird in der Analyse verwendet, um **NUR** die fachlichen Anforderungen zu ermitteln



→ **Technische Sichten**

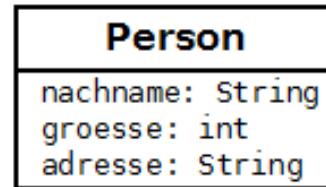
- In Architektur, Detailed Design, Implementierungsdoku verw.
- Es werden auch die für die Lösung **relevanten** technischen Details dargestellt:





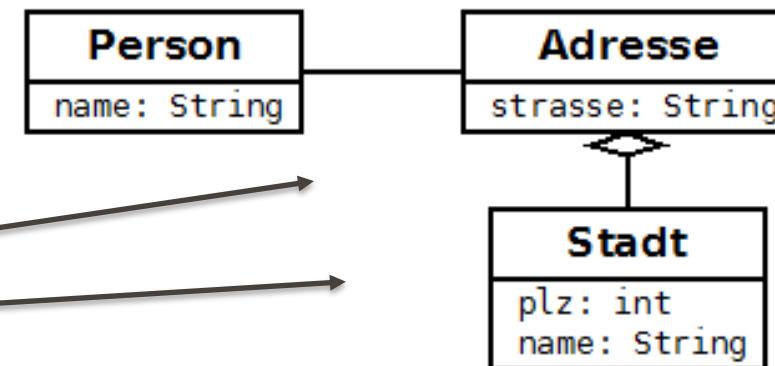
→ ***Fachliche Sicht betrachtet alles ohne technische Details***

- Wird in der Analyse verwendet, um **NUR** die fachlichen Anforderungen zu ermitteln



→ **Technische Sichten**

- In Architektur, Detailed Design, Implementierungsdoku verw.
- Es werden auch die für die Lösung **relevanten** technischen Details dargestellt:



Technische Lösung  
z.B. für spezifische Suche  
nach Städten oder Adressen

# EXTENSIONALE & INTENSIONALE SICHTWEISE



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

→ **Fachliche Sicht betrachtet alles ohne technische Details**

- Wird in der Analyse verwendet, um **NUR** die fachlichen Anforderungen zu ermitteln
- Eigentlich immer **Extensionale Bedeutungsart**

→ **Technische Sichten**

- Architektur (High-Level)
- Detailed Design
- Code / Diagramm mit exakter Darstellung des Codes
- “**Sollten**” beide **Bedeutungsarten** spätestens beim Implementierungsnahen Diagramm **zusammenfallen**
  - Nicht Zusammenfallen deutet auf möglichen Designfehler oder Architekturerosion hin

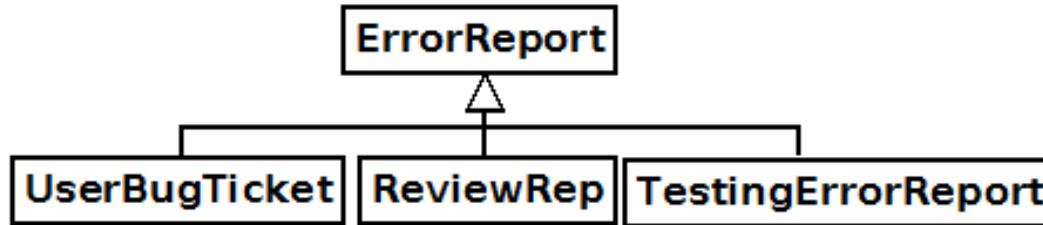
# EXTENSIONALE & INTENSIONALE BEDEUTUNGSARTEN



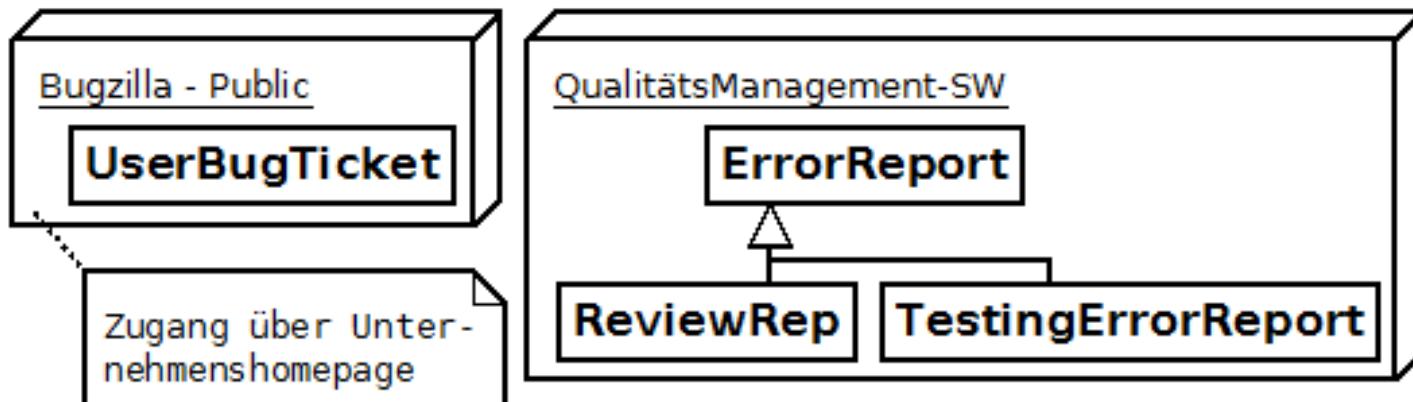
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## BSP - Mögliche Architekturerosion:

- Ursprünglich analysiert (Fachliche Sicht):



- Mapping auf IT-Systeme zunächst (Deploymentdiagramm):



Weil User UserBugTickets über die Homepage erstellen müssen  
wird Bugzilla dafür verwendet

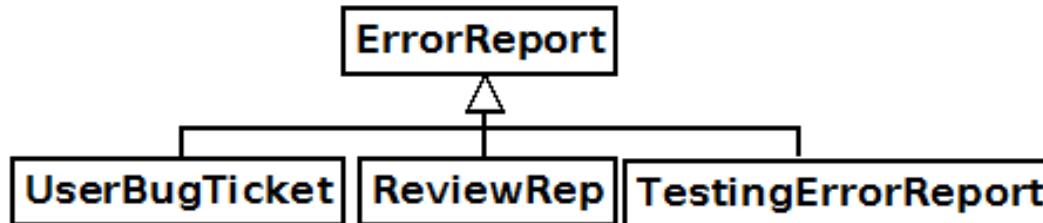
# EXTENSIONALE & INTENSIONALE BEDEUTUNGSARTEN



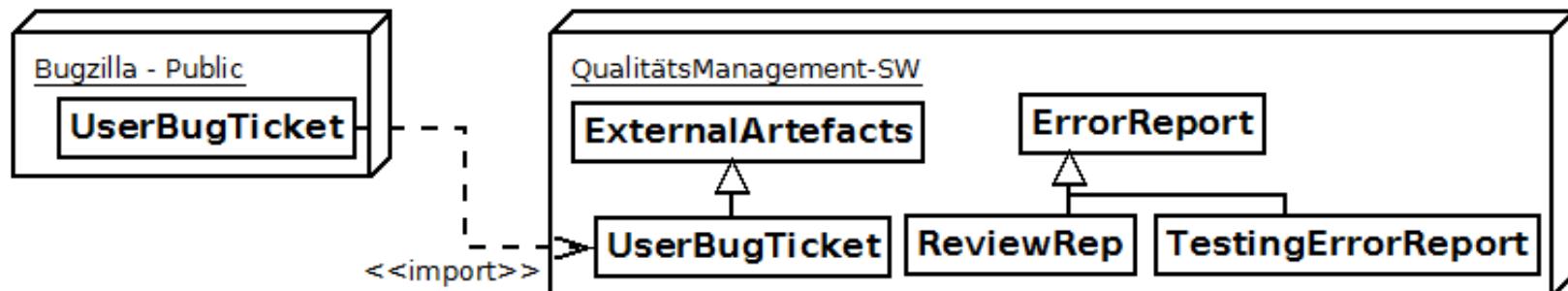
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## BSP Mögliche Architekturerosion:

- Eigentlich gilt:



- Einige Jahre später – import in das QM-System:



Aus anderen Gründen scheint es besser UserBugTicket anders zu behandeln  
ABER: Diese „Sonderlocke“ muss dann in Zukunft muss immer extra behandelt  
werden → macht dann später immer mal wieder Probleme

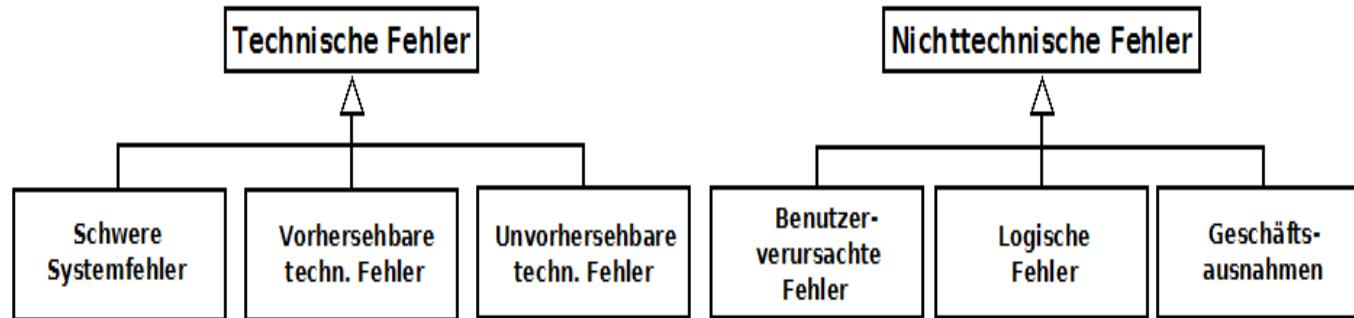
# EXTENSIONALE & INTENSIONALE BEDEUTUNGSARTEN



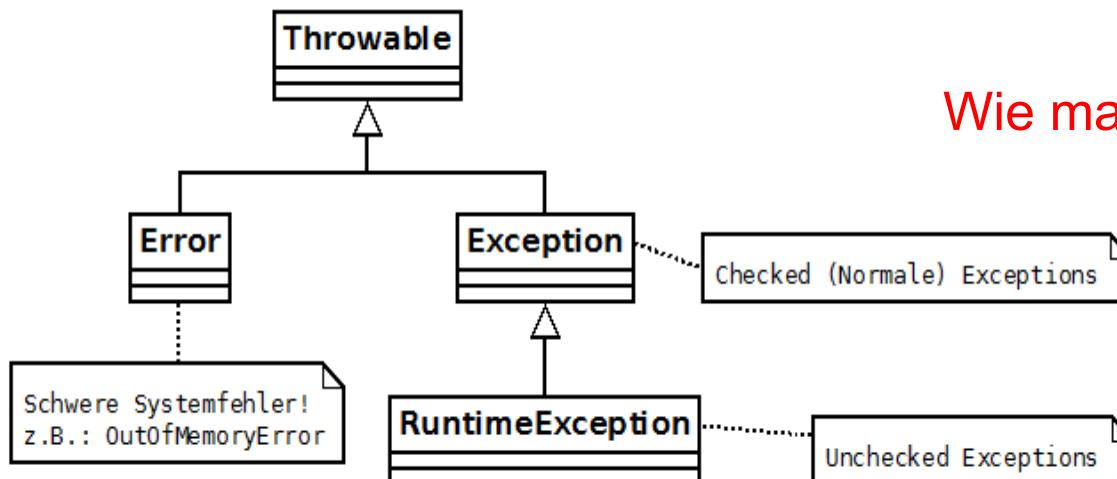
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## BSP Fehlerbehandlung:

- Analyse (→ Extensional):



- Technische Ebene JAVA Typsystem (→ Intensional):



Wie mappt sich das?

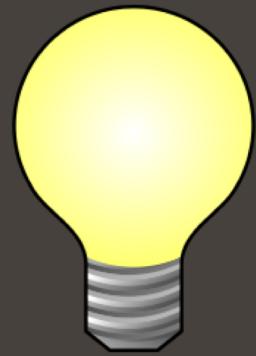




# 07

## Fazit

Ziel:  
Was haben wir damit gewonnen?





# WAS HABEN WIR GELERNT?

- Das Klassendiagramm
  - Klassen und Assoziationen, Dependency-Beziehungen
  - Vererbung, Attribute, Methoden, Sichtbarkeiten
  - Stereotypen dienen zur Bedeutungseinschränkung
- Verschiedene Bedeutungsarten möglich
  - **Extensional vs. Intensional**
    - Vielleicht eher eine akademische Unterscheidung
  - ABER: Wichtig zu Unterscheiden bei Analyse- und Designphase
  - ABER: Wichtig zu verstehen, dass die Bedeutung nicht vollständig eindeutig ist und variieren kann
    - Leider, die Welt ist halt nicht perfekt
    - Man sollte die Bedeutung auch textuell beschreiben!



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

AUF GEHT'S!!

SELBER MACHEN UND LERNEN!!



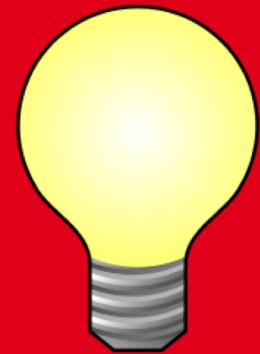


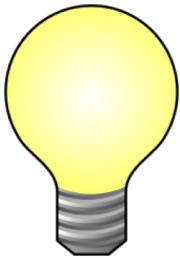
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

26.11.2020

# Zustandsdiagramme

Zustandsdiagramme





# AGENDA

Einführung ins Thema

Loslegen mit einem Beispiel

Modellelemente im Überblick

Weitere Anmerkungen

Fazit

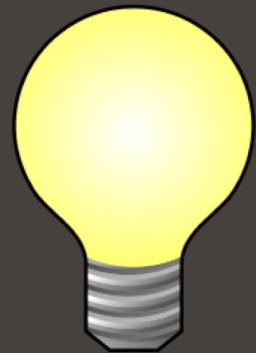


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# 01

## EINFÜHRUNG INS THEMA

Ziel:  
Die Eckpunkte des Themas kennenlernen



# ZUSTANDSDIAGRAMME



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

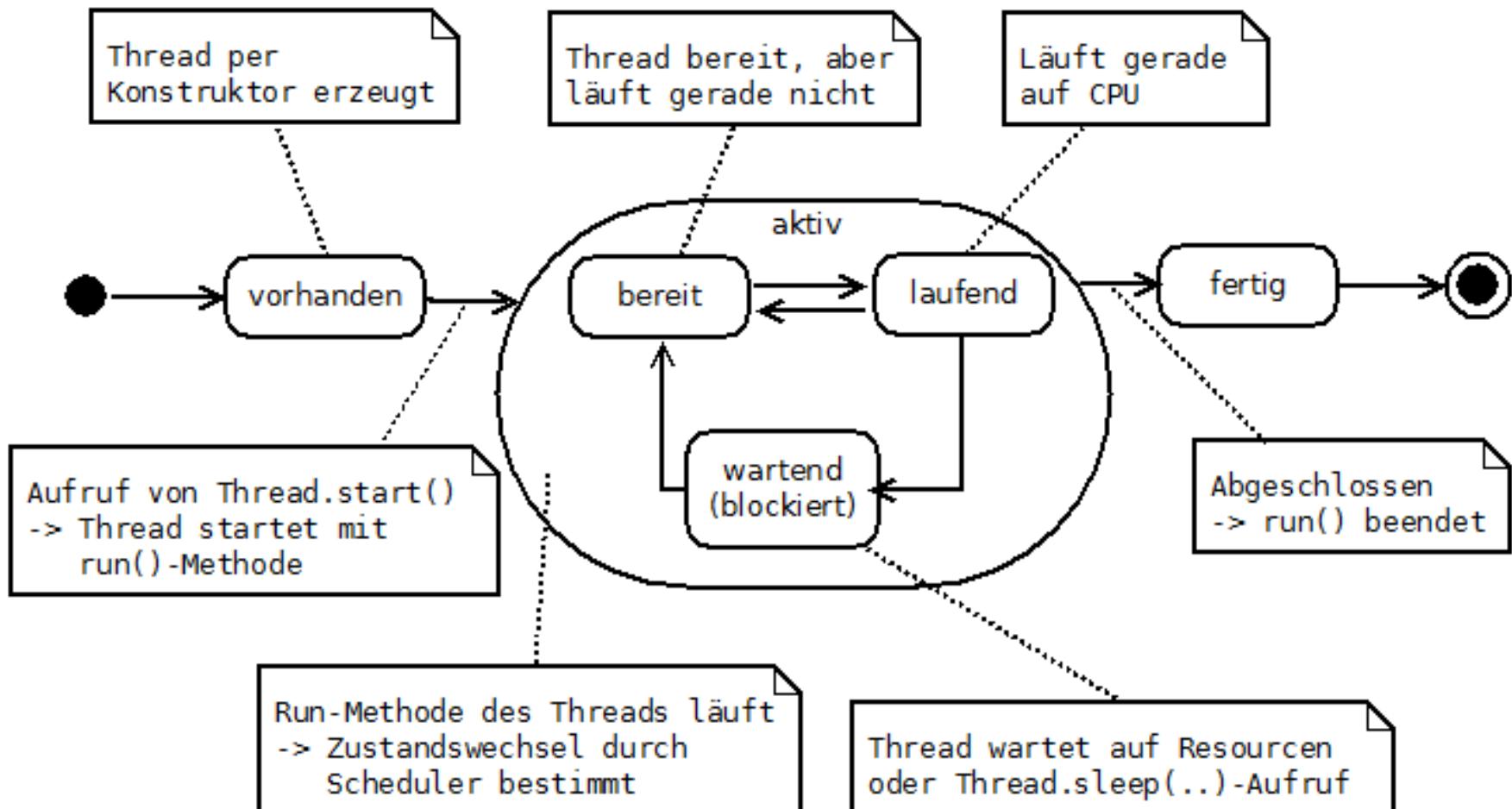
- Eignen sich zur Modellierung von Systemen/SW, die:
  - in Zuständen verharren
  - und diese meist durch ein Signal von außen wechseln
- Zustand: Unter Umständen verbleibt ein System sehr lange in einem bestimmten “Modus” → Zustand
- Wechsel: Meist durch ein Signal von außen
  - Es kann sehr lange dauern bis ein Wechsel eintritt



# ZUSTANDSDIAGRAMME

- Zustand: Unter Umständen verbleibt ein System sehr lange in einem bestimmten “Modus” → Zustand
- Wechsel: Meist durch ein Signal von außen
  - Es kann sehr lange dauern bis ein Wechsel eintritt
- Beispiele:
  - Getränkeautomat:
    - Zustände: Wartend, Geld eingeworfen, Getränkewahl, Bereitstellung Getränk, Wartend
  - Automobilsteuergeräte:
    - Zustände: Schlafen (23h am Tag) –(Tür öffnet sich)→ aufgewacht Zündschlüssel steckt, Motor gestartet, Motor aus, Zündschlüssel raus –(Tür zu)→ Schlafen

# 2. SEMESTER - PM: MÖGL. ZUSTÄNDE EINES THREADS



# ANDERE NAMEN FÜR ZUSTANDSDIAGRAMME



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- State Machine, Zustandsautomat (endlich)
- State Diagram
- State Chart
- Mealy-Automat (Aktion im Übergang)
- Moore-Automat (Aktion im Zustand)

→ Details lernen Sie hier noch in der Vorlesung  
“Automatentheorie und Formale Sprachen”



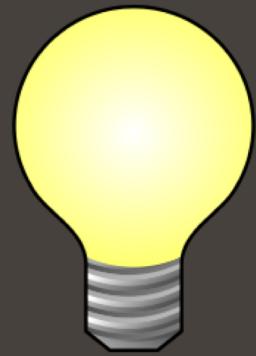
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

02

## Loslegen mit einem Beispiel

Ziel:

Erste Sachen kennenlernen

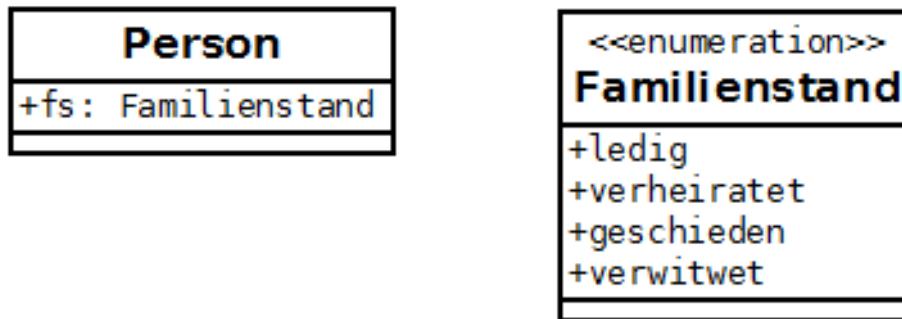


# BSP: PERSONEN AUS SICHT DES STANDESAMTS



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

Klassendiagramm:

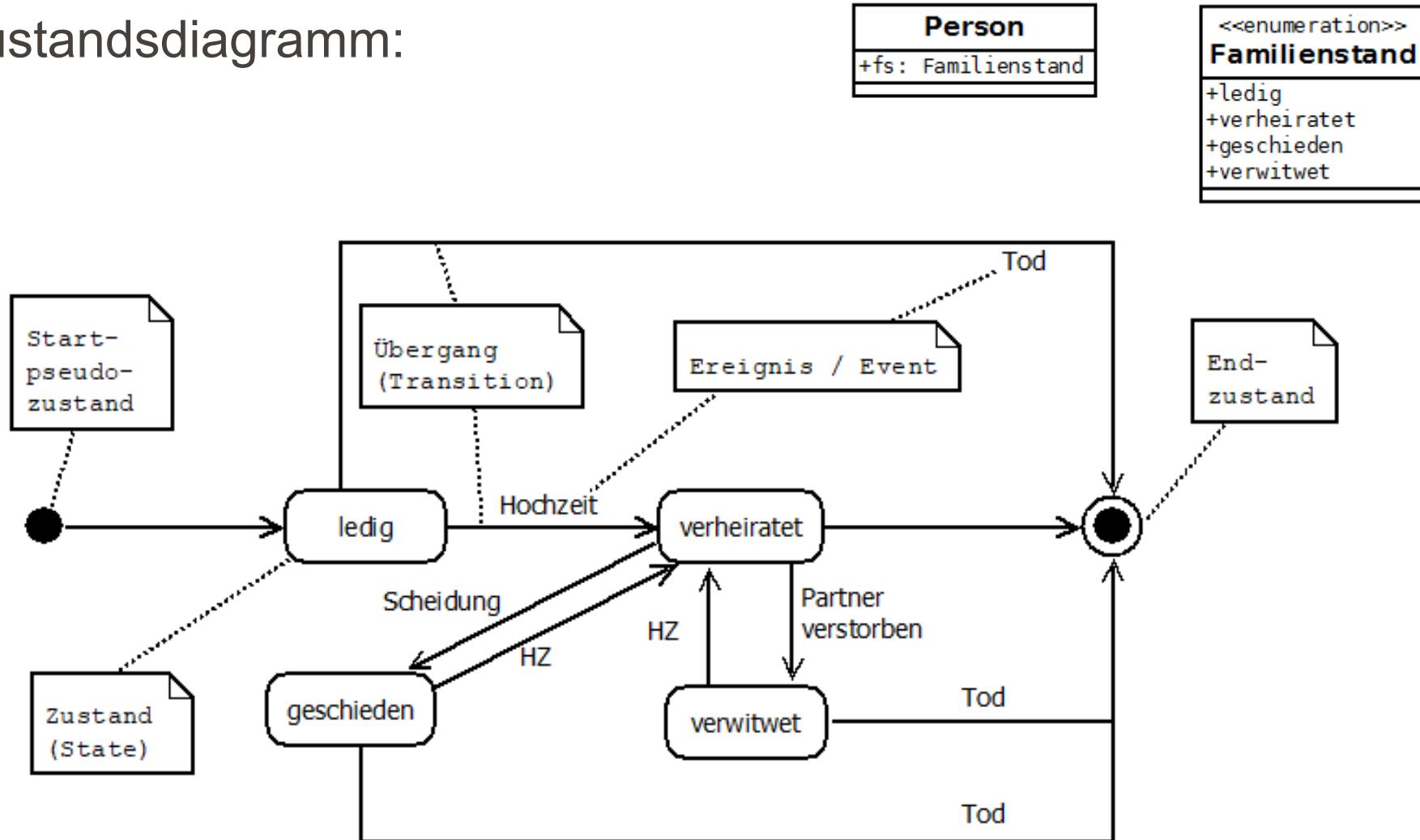


# BSP: PERSONEN AUS SICHT DES STANDESAMTS



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

Zustandsdiagramm:

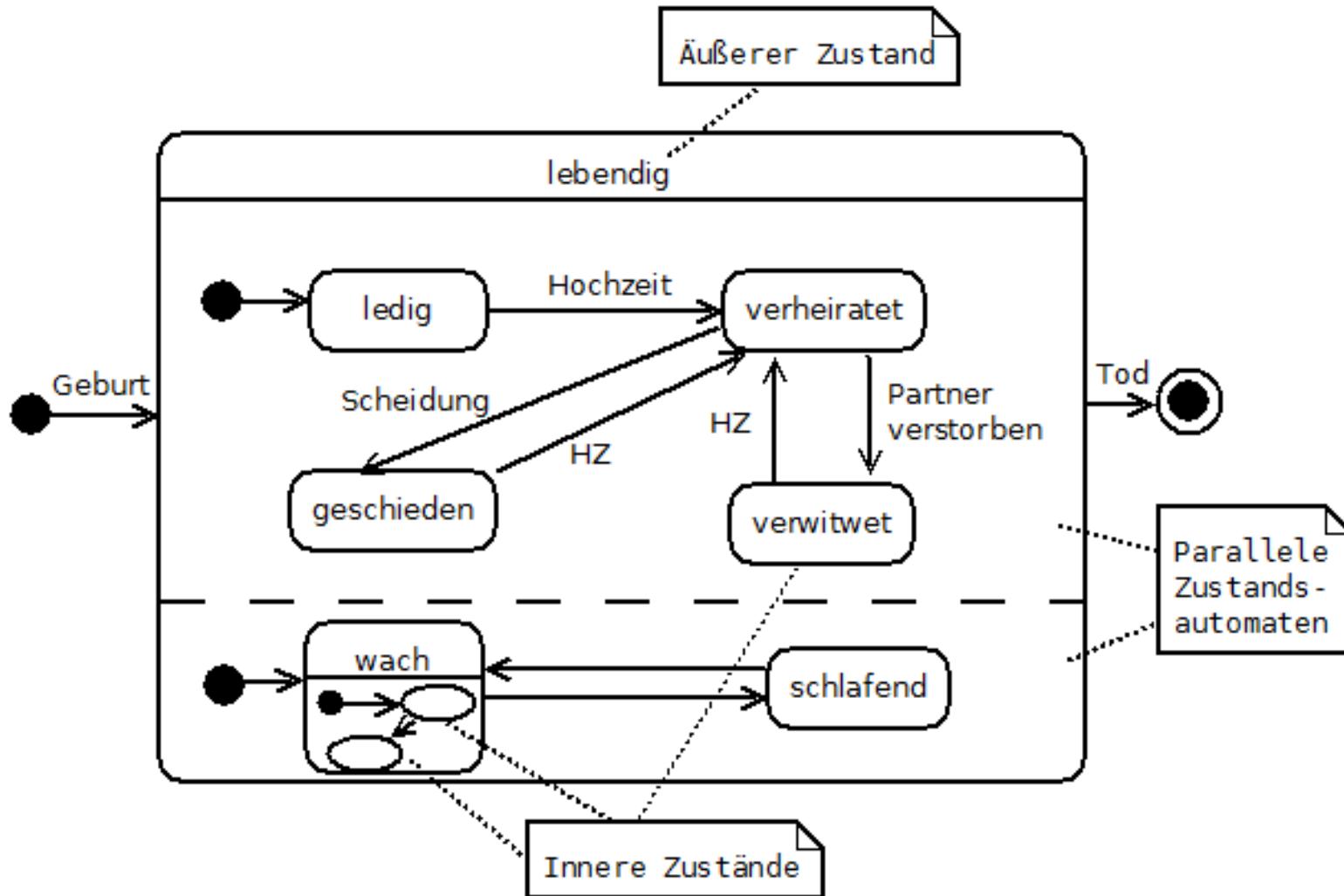


# BSP: PERSONEN AUS SICHT DES STANDESAMTS



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

Zustandsdiagramm – Nebenläufigkeit / Schachtelung :



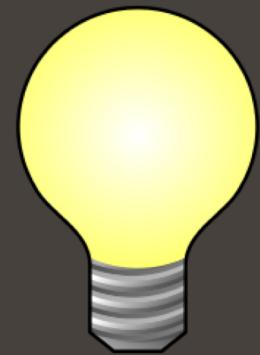


# 03

## Zustandsdiagramme

### - Modellelemente im Überblick

Ziel:  
Die Elemente im Überblick erfassen



# MODELLELEMENTE – ZUSTÄNDE



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- “Normaler Zustand” (State):



- Pseudozustände (Pseudostate):
  - Start
  - Einfache Historie
  - Tiefe Historie
- End-Zustand (FinalState) → Sonderfall, weil er beendet

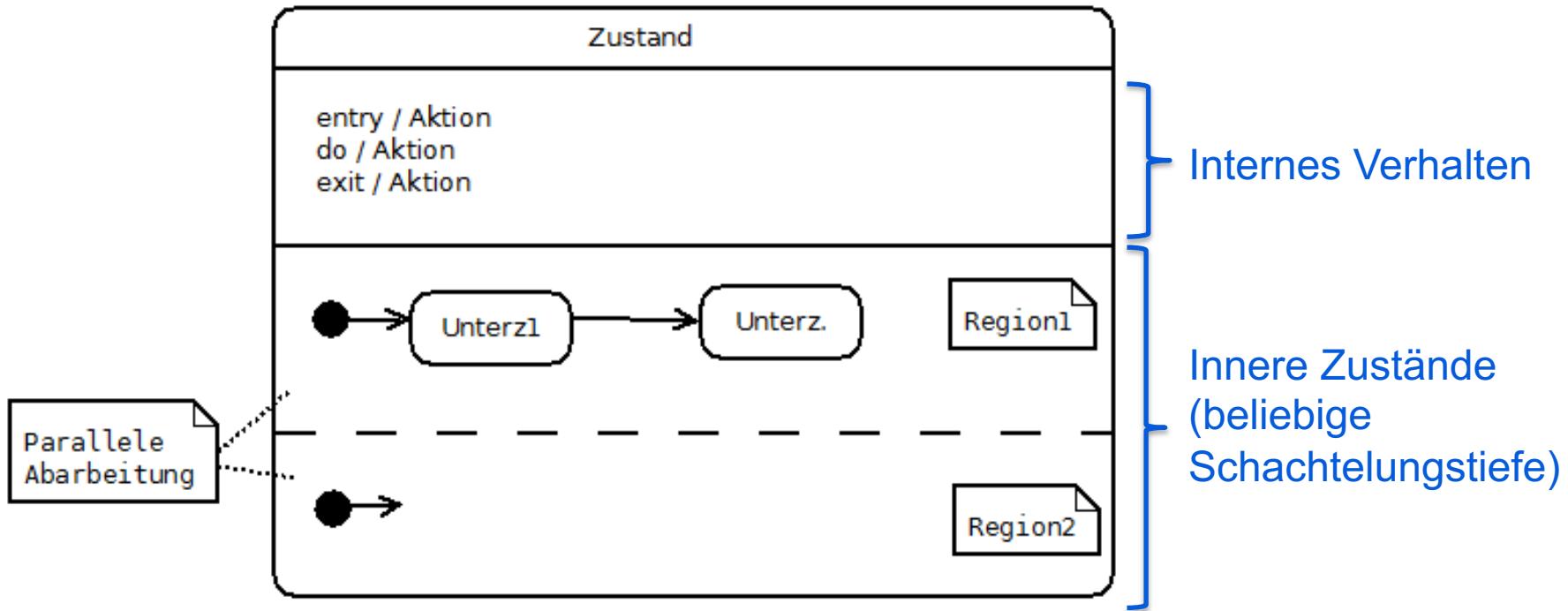


# MODELELEMENTE – ZUSTÄNDE

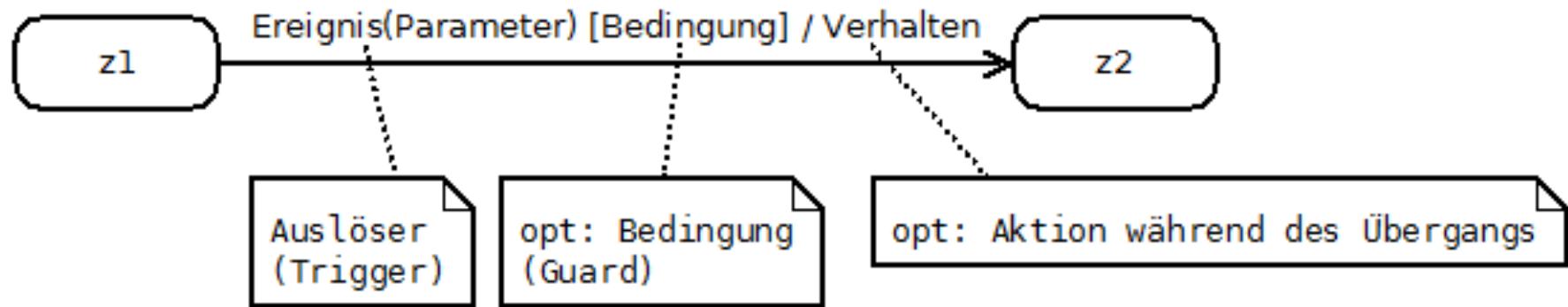


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Möglicher innerer Aufbau von Zuständen:



# MODELELEMENTE – ÜBERGÄNGE



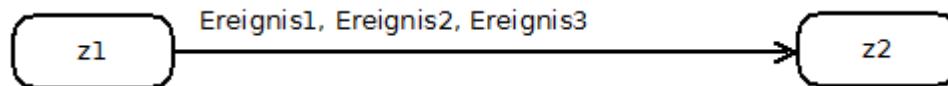
Mögliche Impl. in Java:

- Trigger  $\triangleq$  Methoden-Signatur
- Verhalten  $\triangleq$  Methoden-Körper
- + Guard

# MODELELEMENTE – ÜBERGÄNGE



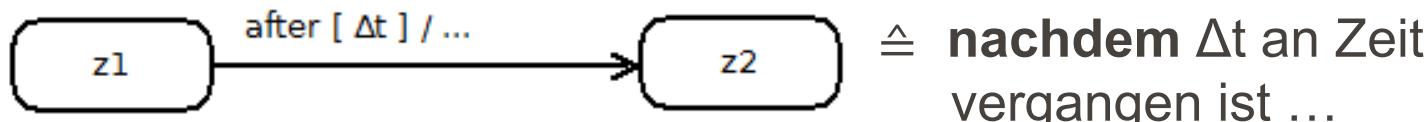
- Mehrere Ereignisse für einen Übergang:



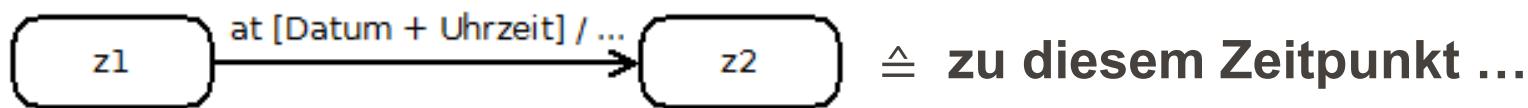
- Spezielle Auslöser (Trigger):**



△ **sobald** Bedingung eintritt ...



△ **nachdem** Δt an Zeit vergangen ist ...

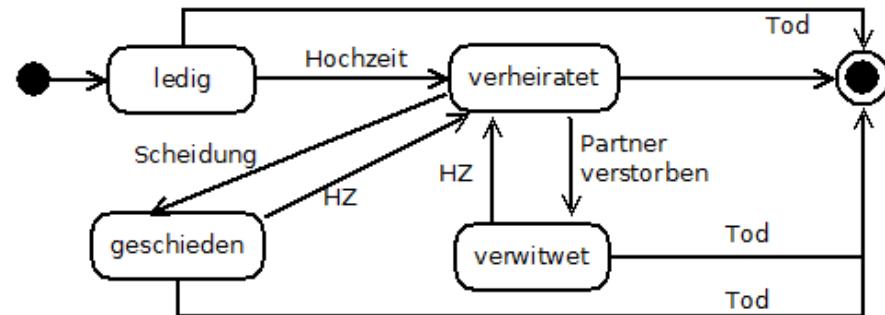


△ **zu diesem Zeitpunkt** ...

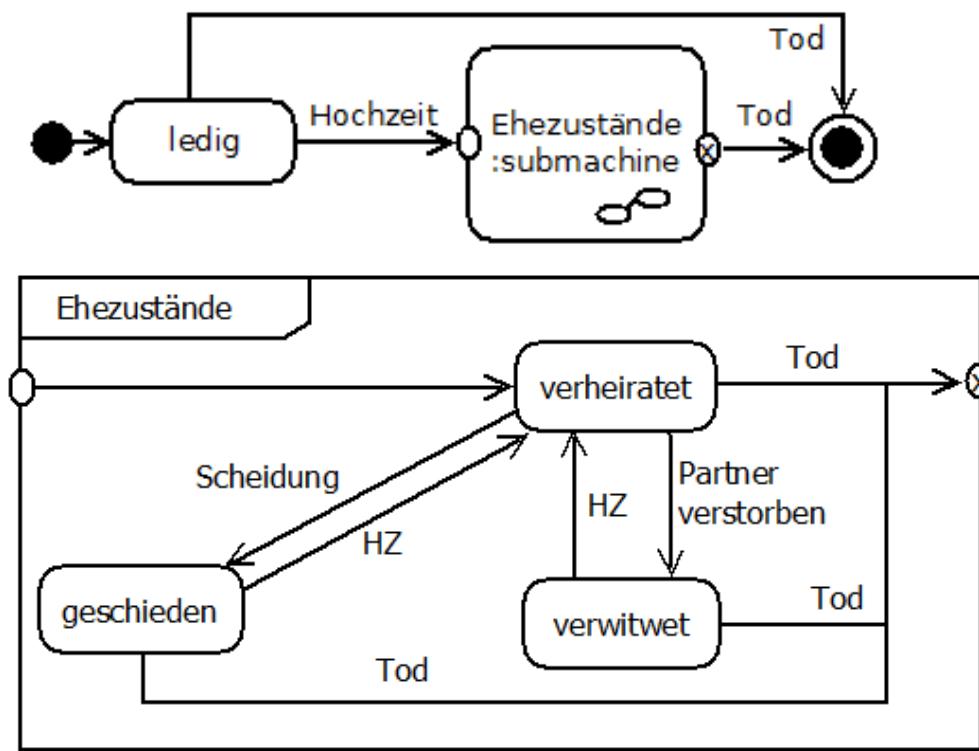


△ Erreichen des Endzust.

# ZERLEGGUNG IN UNTERDIAGRAMME



Zustandsdiagramme können auch in mehrere zerlegt werden:



# TIPPS ZUR MODELLIERUNG



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

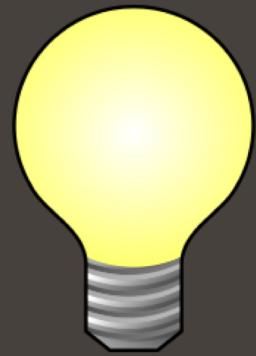
- Name eines Zustands == Adjektiv
  - (oder Substantiv+Adjekt – z.B.: LichtAn, LichtAus)
- Erst grob, dann Details
  - Zustände sammeln
  - Nach auslösenden Ereignissen suchen
  - Iterativ verfeinern
- Erst auf einer Ebene arbeiten
  - Später ggfs. Schachteln



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

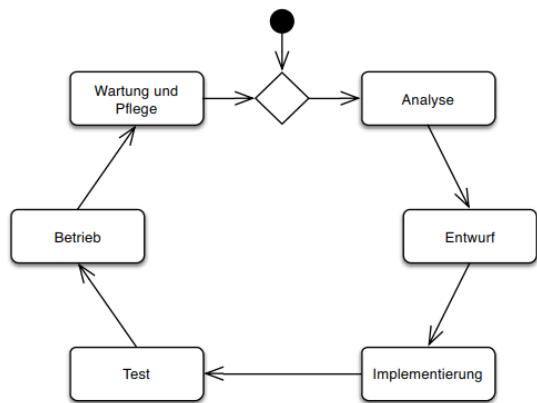
## 04 Weitere Anmerkungen

Ziel:  
Weitere Sachen



# WOFÜR EIGNEN SICH ZUSTANDSAUTOMATEN?

- Analyse
  - Anforderungen des Kunden analysieren
    - Potentielle Zustände des Zielsystems
    - Anderes noch unbestimmtes
  - Prozesse oder Systeme analysieren in die das Zielsystem eingebettet ist
- Design
  - Zustände des Zielsystems
  - Zustände einzelner Objekte
  - Lebenszyklus von Objekten
  - ...
- Implementierung → siehe Design



# BEVORZUGTE EINSATZGEBIETE



- Asynchrone Vorgänge:
  - Auf eine kurze Aktion folgt eine lange Wartezeit
- Lebenszyklen über mehrere Use Cases hinweg
- Ereignisgesteuerte Vorgänge: z.B. GUI (1 Klick  $\triangleq$  1 Ereignis)
- Verhalten hängt von Vorgeschichte/aktuuellem Zustand ab
  - Z.B. Einfügen nicht möglich, wenn Liste schon voll
- Codegenerierung\* von zustandbasiertem Verhalten
  - Besonders geeignet für ereignisbasierte Systeme,  
da Zustandsdiagramme eine deterministische und vollständige  
Modellierung erlauben
  - Erlaubt dann eine ausgiebigere Analyse mit Simulation, ...

\* Siehe z.B.: <https://www.embedded-software-engineering.de/codegenerierung-was-man-damit-nicht-machen-kann-a-555705/>

# ZUSTANDSAUTOMATEN ALS TABELLEN



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Die in Zustandsautomaten enthaltene Information kann auch in Tabellenform spezifiziert werden (kein UML!)

Ausgangszustand	Ereignis	Bedingung	Aktion	Zielzustand
●				ledig
ledig	Hochzeit			verheiratet
...				

# WIE ZUSTANDSAUTOMATEN IMPLEMENTIEREN?



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## Verschiedene Ansätze zur Implementierung:

- Zustandsattribut (z.B. Enum) – Zwei mögliche Implement.:
  1. 1 Trigger  $\triangleq$  1 Methode
  2. Switch – Case
- Tabelle
  - Array wo jede Zeile der Tabelle repräsentiert (Beliebt in C)
- Zustandsobjekte
  - Jeder Zustand  $\triangleq$  1 Objekt
  - Java: Enum, die Zustände repräsentiert
    - + Methoden, die Verhalten in jeweiligen Zustand repräs.
    - + Methoden, die in neuen Zustand wechseln

# WIE IMPLEMENTIEREN?

```
public class Person{
    FamilienStand _fs= FamilienStand.ledig;
    Person _ehePartner= null;

    public enum FamilienStand implements ... {
        ledig(...), verheiratet(...), geschieden(...),
        verwitwet(...), tot;
    }

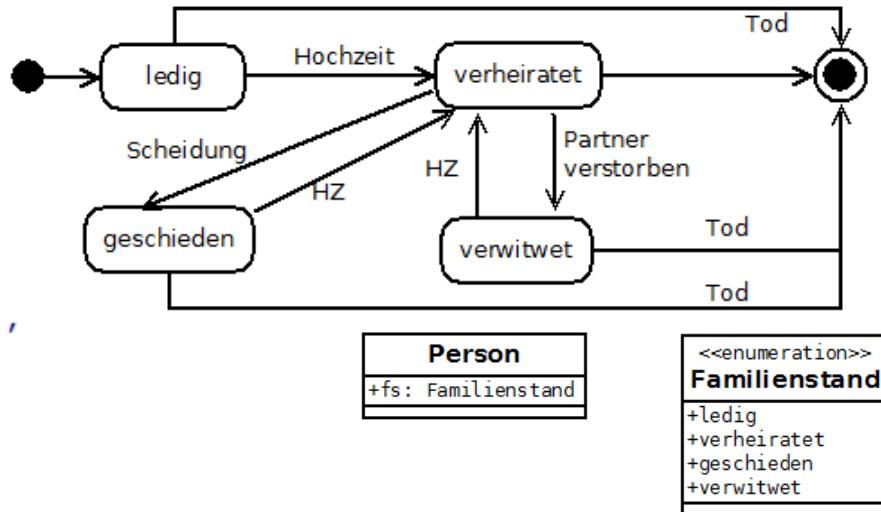
    private FamilienStand(...) { ... }

    public boolean kannHeiraten(){
        return this == ledig || this == geschieden || this == verwitwet;
    }

    public void heirate(Person partner){
        if(this.kannHeiraten() && partner._fs.kannHeiraten()){
            Person.this._ehePartner= partner;
            partner._ehePartner= Person.this;
            Person.this._fs= verheiratet;
            partner._fs= verheiratet;
        }
        throw new InvalidStateChangeException("....");
    }

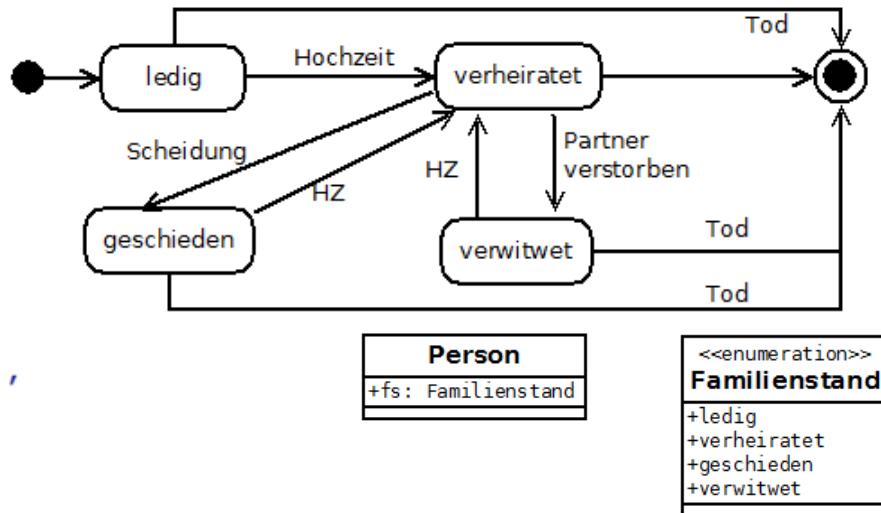
    public void scheide(){
        ErrorHandler.Assert(Person.this._fs == verheiratet, ... );
        ErrorHandler.Assert(Person.this._ehePartner._fs == verheiratet, ... );
        ErrorHandler.Assert(Person.this._ehePartner._ehePartner == Person.this, ... );

        Person.this._fs == geschieden;
        Person.this._ehePartner._fs == geschieden;
        Person.this._ehePartner._ehePartner= null;
        Person.this._ehePartner= null;
    }
}
```



# WIE IMPLEMENTIEREN?

```
public class Person{  
    FamilienStand _fs= FamilienStand.ledig;  
    Person _ehePartner= null;  
  
    public enum FamilienStand implements ... {  
        ledig(...), verheiratet(...), geschieden(...),  
        verwitwet(...), tot;  
  
        private FamilienStand(...) { ... }  
        ...  
  
        public void sterbe(){  
            Person.this._fs= tot;  
  
            if(Person.this._ehePartner != null){  
                ErrorHandler.Assert(Person.this._ehePartner._fs == verheiratet, ... );  
                Person.this._ehePartner._fs= verwitwet;  
            }  
        }  
  
        public void heirate(Person partner){  
            this._fs.heirate(partner);  
        }  
  
        public void scheide(){  
            this._fs.scheide();  
        }  
  
        public void sterbe(){  
            this._fs.sterbe();  
        }  
    }  
}
```



→ Ähnliche Wirkweise wie  
Method-Object-Pattern (PMT (2. Sem))

- Komplexität wird in innerer Klasse Gekapselt
- Hier aber die Komplexität des Zustandsautomaten

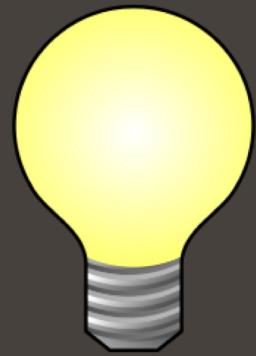


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# 05

## Fazit

Ziel:  
Was haben wir damit gewonnen?





# WAS HABEN WIR GELERNT?

- Zustandsdiagramm
- Wie man Zustände und dessen Übergänge modelliert
- Alternative Notation via Tabelle
- Sonstige Informationen



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

AUF GEHT'S!!

SELBER MACHEN UND LERNEN!!





Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# SOFTWARETECHNIK

## Teil 1

Letztes Update: 19. Oktober 2019

Dr. Eva-Maria Iwer

Fachbereich Design Informatik Medien (DCSM)  
Hochschule **RheinMain**



# GLIEDERUNG

1. Versionierung

2. Abkürzungen

# VERSION

## Versionierung

Version	Datum	Kommentare
1.0	April 2019	Initial
1.1	Mai 2019	Fehlerbehebung
1.2	Oktober 2019	Erweiterung Branches

# VERSIONIERUNG

# VERSIONIERUNG

## Was ist Versionskontrolle und warum es so wichtig ist!

- Versionsverwaltungssystem (VCS) ist ein System, welches die Änderungen an einer oder einer Reihe von Dateien über die Zeit hinweg protokolliert.
- Mit dem System kann man später auf eine bestimmte Version zurückgreifen kann

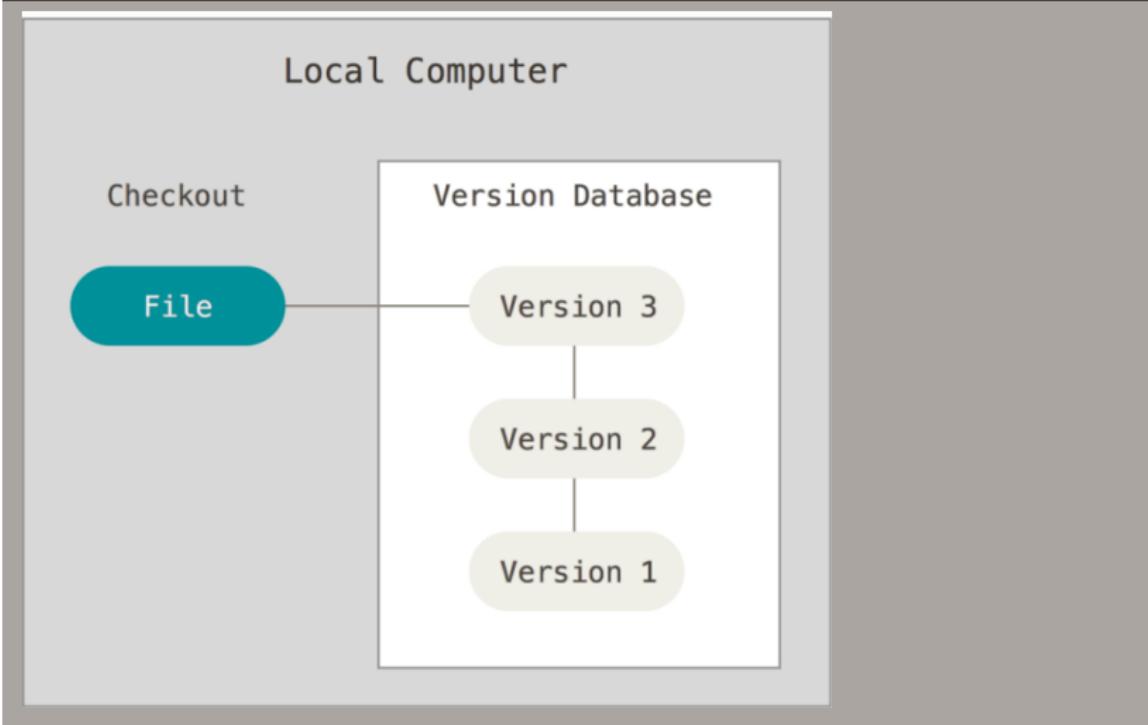
# VERSIONIERUNG

## Lokales VCS

- Eines der populäreren Versionsverwaltungssysteme war RCS
- RCS arbeitet nach dem Prinzip, dass für jede Änderung ein Patch (ein Patch umfasst alle Änderungen an einer oder mehreren Dateien) in einem speziellen Format auf der Festplatte gespeichert wird
- Um eine bestimmte Version einer Datei wiederherzustellen, wendet es alle Patches bis zur gewünschten Version an und rekonstruiert damit die Datei in der gewünschten Version.

# VERSIONIERUNG

## Lokales VCS



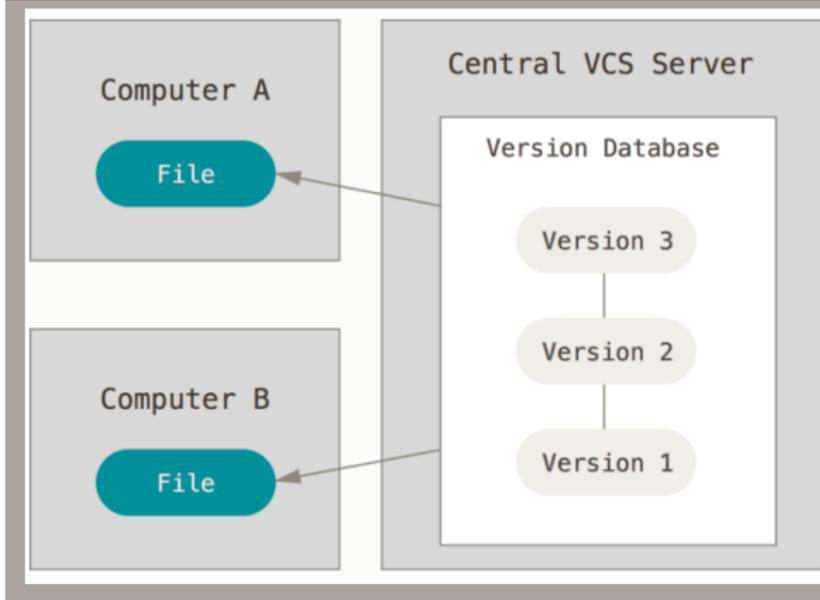
# VERSIONIERUNG

## Zentralisierte Versionskontrolle (CVCS)

- Beispielsysteme sind CVS, Subversion und Perforce
- basieren auf einem zentralen Server, der alle versionierten Dateien verwaltet
- Die Clients können die Dateien von diesem zentralen Ort abholen und auf ihren PC übertragen.
- Den Vorgang des Abholens nennt man Auschecken (engl. to check out).

# VERSIONIERUNG

## Zentralisiertes VCS



# VERSIONIERUNG

## Verteilte Versionsverwaltungssysteme (DVCS) - 1

- Beispielsysteme sind Git, Mercurial, Bazaar oder Darcs
- Anwender nicht einfach nur den jeweils letzten Zustand des Projektes von einem Server: Sie erhalten stattdessen eine vollständige Kopie des Repositorys.
- Jede Kopie, ein sogenannter Klon (engl. clone), ist ein vollständiges Backup der gesamten Projektdaten.

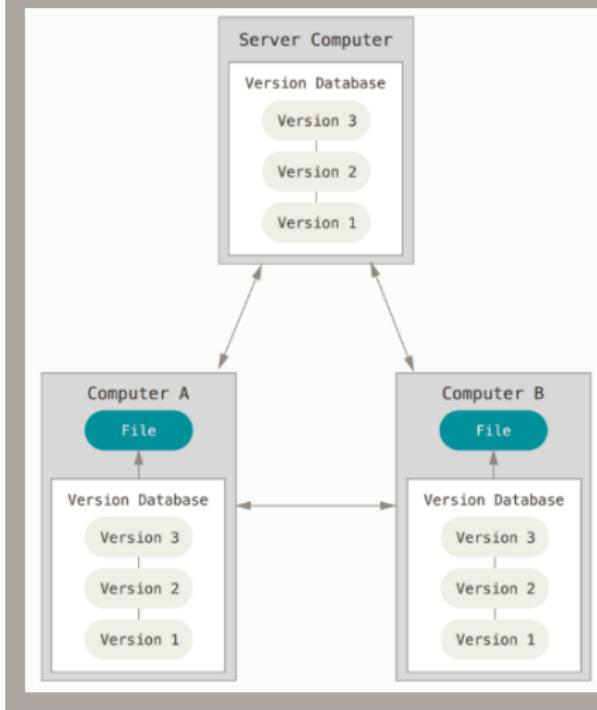
# VERSIONIERUNG

## Verteilte Versionsverwaltungssysteme (DVCS) - 2

- Darüber hinaus können derartige Systeme hervorragend mit verschiedenen externen Repositorys, sogenannten Remote-Repositorys, umgehen, sodass man mit verschiedenen Gruppen von Leuten simultan auf verschiedene Art und Weise, an einem Projekt zusammenarbeiten kann.
- Damit ist es möglich, verschiedene Arten von Arbeitsabläufen zu erstellen und anzuwenden, welche mit zentralisierten Systemen nicht möglich wären. Dazu gehören zum Beispiel hierarchische Arbeitsabläufe.

# VERSIONIERUNG

## Verteilte VCS



# VERSIONIERUNG

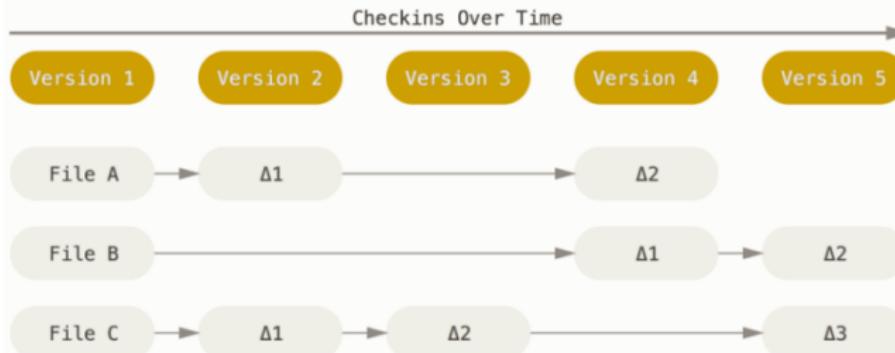
## GIT

- Geburt 2005
- Entstanden aus der Linux Entwickler Community
- Ziele:
  - Geschwindigkeit
  - Einfaches Design
  - Gute Unterstützung von nicht-linearer Entwicklung  
(tausende parallele Entwicklungszweige)
  - Vollständig dezentrale Struktur
  - Fähigkeit große Projekte, wie den Linux Kernel, effektiv zu verwalten (Geschwindigkeit und Datenumfang)

# GIT GRUNDLAGEN

# Snapshots und nicht die Unterschiede

- Die meisten anderen Systeme speichern Information, als eine fortlaufende Liste von Änderungen an Dateien.
  - Diese Systeme betrachten die Informationen, die sie verwalten, als eine Menge von Dateien und die Änderungen, die über die Zeit hinweg an einzelnen Dateien vorgenommen werden.



# GIT GRUNDLAGEN

# Snapshots und nicht die Unterschiede

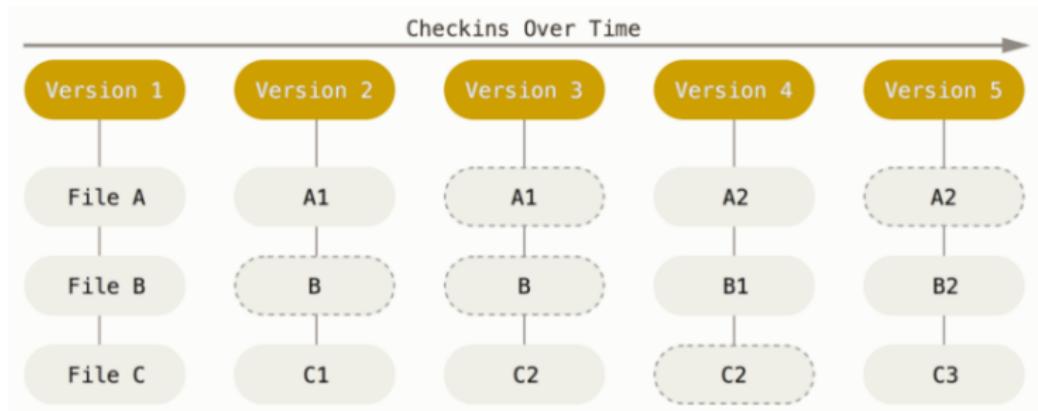
- Stattdessen betrachtet Git seine Daten eher als eine Reihe von Snapshots eines Mini-Dateisystems.
  - Der gegenwärtigen Status Ihres Projekts wird als eine Version in Git gespeichert
  - Sichert Git den Zustand sämtlicher Dateien in diesem Moment und macht sozusagen ein Schnappschuss (engl. Snapshot) von all Ihren Daten.
  - Zusätzlich speichert Git eine Referenz auf diesen Snapshot.
  - Um dies möglichst effizient und schnell tun zu können, kopiert Git unveränderte Dateien nicht, sondern legt lediglich eine Verknüpfung zu der vorherigen Version der Datei an.

# GIT GRUNDLAGEN

## Zustände

- Committet (engl. committed) - dass die Daten sicher in der lokalen Datenbank gespeichert sind
- Geändert (engl. modified) - dass eine Datei geändert, aber noch nicht in die lokale Datenbank eingecheckt wurde
- für Commit vorgemerkt (engl. staged) - dass eine geänderte Datei in ihrem gegenwärtigen Zustand für den nächsten Commit vorgemerkt ist.

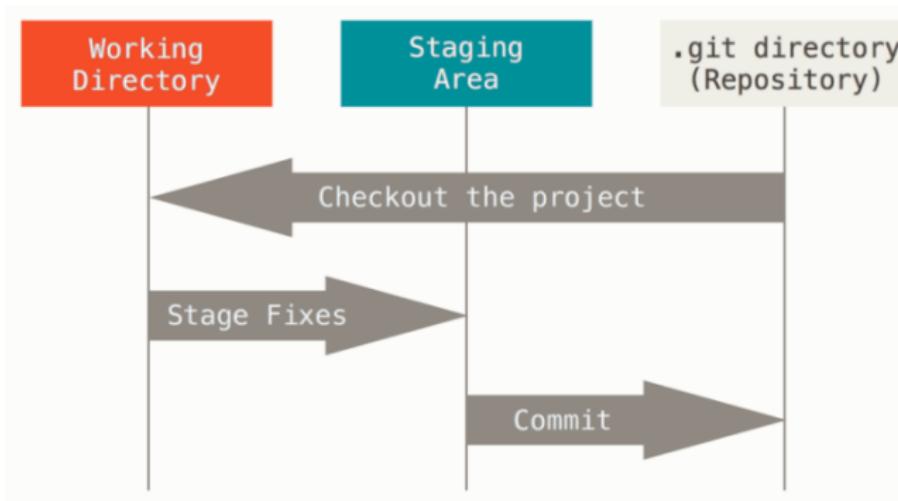
# GIT GRUNDLAGEN



# GIT GRUNDLAGEN

## Hauptbereiche

- Working Directory - Das Arbeitsverzeichnis
- Staging Area
- Repository - Das Git Verzeichnis



# GIT GRUNDLAGEN

## Hauptbereiche

- Working Directory - Das Arbeitsverzeichnis ist ein einzelnes Abbild einer spezifischen Version des Projektes. Die dort enthaltenen Dateien werden aus der komprimierten Datenbank geholt und auf der Festplatte in einer Form gespeichert, sodass man sie nutzen oder bearbeiten kann.

# GIT GRUNDLAGEN

## Hauptbereiche

- Staging Area - eine Datei, normalerweise befindet sich diese im Git Verzeichnis, in der vorgemerkt wird, welche Änderungen der nächste Commit umfassen soll.  
Manchmal wird dieser Ort auch als “Index” bezeichnet, aber der Begriff Staging-Area ist der gängigere.

# GIT GRUNDLAGEN

## Hauptbereiche

- Repository - Das Git Verzeichnis ist der Ort, an dem Git Metadaten und die lokale Datenbank für ein Projekt sichert. Dies ist der wichtigste Teil von Git, und dieser Teil wird kopiert, wenn man ein Repository von einem anderen Rechner klonnt.

# GIT GRUNDLAGEN

## **Arbeitsschritte beim Einchecken**

1. Man ändert die zu bearbeitenden Dateien im Arbeitsverzeichnis
  2. Man merkt die Dateien für einen Commit vor, fügt also einen Schnappschuss der Dateien der Staging-Area hinzu
  3. Man führt einen Commit aus, wodurch der in der Staging-Area vorgemerkte Schnappschuss dauerhaft im Git Verzeichnis gespeichert wird

# GIT GRUNDLAGEN

# Installation unter Linux

```
sudo yum install git-all
```

bzw.

```
sudo apt-get install git-all
```

# GIT GRUNDLAGEN

## Installation unter Windows

Eine offizielle Windows Version findet man direkt auf der Git Homepage.

# GIT BASIS-KONFIGURATION

## git config

Git umfasst das Werkzeug git config, welches die Möglichkeit bietet, Konfigurationswerte zu verändern. Die Konfiguration ist an drei verschiedenen Orten gespeichert:

- /etc/gitconfig enthält Parameter, die für jeden Anwender des Systems und alle Projekte gelten. Wenn man git config mit der Option –system verwendet, wird von dieser Datei gelesen bzw. in diese Datei geschrieben.
- /.gitconfig gelten ausschließlich für den jeweiligen Benutzer. Wenn man git config mit der Option –global verwendet, wird von dieser Datei gelesen bzw. in diese Datei geschrieben.
- config im Git Verzeichnis (also '.git/config') enthält Parameter, die für das jeweilige Projekt gelten.

# GIT BASIS-KONFIGURATION

## git config

Diese drei Dateien haben unterschiedliche Prioritäten. Die oberste Priorität haben die Werte aus `.git/config`, dann folgt `/.gitconfig` und zuletzt `/etc/gitconfig`. Ist zum Beispiel ein Parameter in `.git/config` und `/etc/gitconfig` mit unterschiedlichen Werten gesetzt, so gilt in diesem Fall der höherpriore Wert aus der Datei `.git/config`.

# GIT BASIS-KONFIGURATION

## Persönliche Konfiguration

→ Name angeben:

```
git config --global user.name "Eva-Maria Iwer"
```

→ eMail angeben: git config --global user.email „eva-maria.iwer@hs-rm.de“

→ Konfig anzeigen: git config --list

# ARBEITEN MIT GIT

## Ein existierendes Verzeichnis als Git Repository initialisieren

Wenn Sie ein bestehendes Projekt in Zukunft versionieren möchten, können Sie dazu in das Hauptverzeichnis des Projekts wechseln und den folgenden Befehl ausführen:

```
git init
```

### Achtung

Zu diesem Zeitpunkt werden noch keine Dateien in Git versiert.

# ARBEITEN MIT GIT

## initialen Commit

Mit dem Befehl `git add` legen Sie fest, welche Dateien versioniert werden sollen und mit dem Befehl `git commit` erzeugen Sie einen neuen Commit:

- `git add *.c`
- `git add LICENSE`
- `git commit -m 'initial project version'`

# ARBEITEN MIT GIT

## Ein existierendes Repository klonen

- Wenn Sie eine Kopie eines existierenden Git Repositorys anlegen wollen, können Sie den Befehl `git clone` verwenden.
- jede einzelne Version jeder einzelnen Datei, also die gesamte Historie eines Projekts auf den Rechner heruntergeladen.

# ARBEITEN MIT GIT

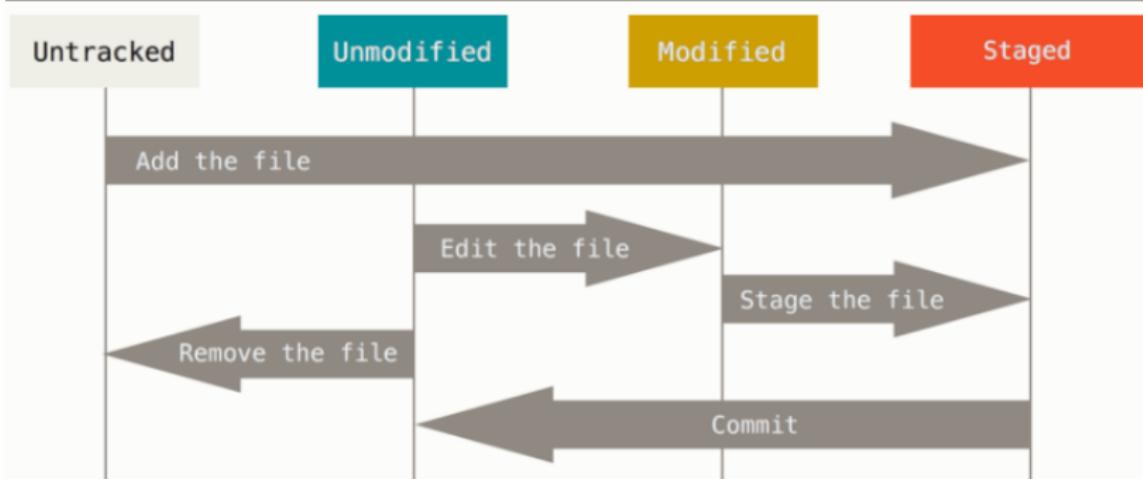
## Änderungen nachverfolgen und im Repository speichern

Ab jetzt können Dateien im Projekt bearbeitet und eingecheckt werden.

- Wenn Sie eine Kopie eines existierenden Git Repositorys anlegen wollen, können Sie den Befehl `git clone` verwenden.
- jede einzelne Version jeder einzelnen Datei, also die gesamte Historie eines Projekts auf den Rechner heruntergeladen.

# ARBEITEN MIT GIT

## Änderungen nachverfolgen und im Repository speichern



# ARBEITEN MIT GIT

## Änderungen nachverfolgen und im Repository speichern

- Sobald Sie anfangen, versionierte Dateien zu bearbeiten, erkennt Git diese als modifiziert, weil sie sich im Vergleich zum letzten Commit verändert haben.
- Die geänderten Dateien können Sie dann für den nächsten Commit vormerken und schließlich alle Änderungen, die sich in der Staging-Area befinden, einchecken.
- Danach geht der Vorgang wieder von vorne los.

# ARBEITEN MIT GIT

## Änderungen nachverfolgen und im Repository speichern

- Das wichtigste Hilfsmittel, um den Zustand zu überprüfen, in dem sich Ihre Dateien gerade befinden, ist der Befehl `git status`.
- Wenn Sie diesen Befehl unmittelbar nach dem Klonen eines Repositorys ausführen, sollte er in etwa folgende Ausgabe liefern:

```
git status
```

```
On branch master
```

```
Your branch is up-to-date with 'origin/master'.  
nothing to commit, working directory clean
```

# ARBEITEN MIT GIT

```
Untracked files:
```

```
(use "git add <file>..." to include in what will be committed)
```

```
Vorlesung/SW_Vorlesung/test
```

```
no changes added to commit (use "git add" and/or "git commit -a")
```

## Neue Dateien zur Versionsverwaltung hinzufügen

```
→ git add test
```

```
→ git status
```

```
Changes to be committed:
```

```
(use "git reset HEAD <file>..." to unstage)
```

```
new file:   Vorlesung/SW_Vorlesung/test
```

# ARBEITEN MIT GIT

## Neue Dateien zur Versionsverwaltung hinzufügen

- Dass die Datei für den nächsten Commit vorgemerkt ist, sehen Sie daran, dass sie im Abschnitt „Changes to be committed“ aufgelistet ist.
- Wenn Sie jetzt einen Commit anlegen, wird der Schnappschuss den Zustand der Datei beinhalten, den sie zum Zeitpunkt des Befehls `git add` hatte.
- Der `git add` Befehl akzeptiert einen Pfadnamen einer Datei oder eines Verzeichnisses. Wenn Sie ein Verzeichnis angeben, fügt `git add` alle Dateien in diesem Verzeichnis und allen Unterverzeichnissen rekursiv hinzu.

# ARBEITEN MIT GIT

```
Changes not staged for commit:  
(use "git add <file>..." to update what will be committed)  
(use "git checkout -- <file>..." to discard changes in working directory)  
  
modified: Vorlesung/SW_Vorlesung/SW_Vorlesung.pdf  
modified: Vorlesung/SW_Vorlesung/SW_Vorlesung.tex  
modified: Vorlesung/SW_Vorlesung/git.tex
```

## Geänderte Dateien zur Staging-Area hinzufügen

- „Changed but not staged for commit“ – das bedeutet, dass eine versionierte Datei im Arbeitsverzeichnis verändert worden ist, aber noch nicht für den Commit vorgemerkt wurde.
- Um sie vorzumerken, führen Sie den Befehl `git add` aus.
- Befehl `git add` - bestimmten Inhalt für den nächsten Commit vorbereiten.

# ARBEITEN MIT GIT

```
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   Vorlesung/SW_Vorlesung/SW_Vorlesung.pdf
    modified:   Vorlesung/SW_Vorlesung/SW_Vorlesung.tex
    modified:   Vorlesung/SW_Vorlesung/git.tex
    new file:   Vorlesung/SW_Vorlesung/images/gitadd.PNG
    new file:   Vorlesung/SW_Vorlesung/images/gitadd1.PNG
    new file:   Vorlesung/SW_Vorlesung/images/gitadd2.PNG
    new file:   Vorlesung/SW_Vorlesung/test
```

## Geänderte Dateien zur Staging-Area hinzufügen

- alle Dateien im Abschnitt „Changes to be committed“ sind für den nächsten Commit vorgemerkt
- Wenn Sie jetzt einen Commit anlegen, wird der Schnappschuss den Zustand der Datei beinhalten, den sie zum Zeitpunkt des Befehls git add hatte.

# ARBEITEN MIT GIT

## Dateien commit

Dateien in das Verzeichnis hochladen

```
git commit
```

```
[master 233ba0a] neue Version für gitadd mit beispiel test hinzufügen
7 files changed, 100 insertions(+), 15 deletions(-)
create mode 100644 Vorlesung/SW_Vorlesung/images/gitadd.PNG
create mode 100644 Vorlesung/SW_Vorlesung/images/gitadd1.PNG
create mode 100644 Vorlesung/SW_Vorlesung/images/gitadd2.PNG
create mode 100644 Vorlesung/SW_Vorlesung/test
```

# ARBEITEN MIT GIT

## Dateien löschen

Als erstes müssen Sie die Datei von ihren tracked Dateien entfernen und dann commiten. Das git rm Kommando unterstützt. Wenn nur die Datei entfernt wird, wird „Changed but not updated“ in ihrem Status.

```
git rm test
```

```
git status
```

# ARBEITEN MIT GIT

## Dateien bewegen

```
git mv file_from file_to
```

# ARBEITEN MIT GIT

## Dateien ignorieren

Erstellen einer .gitignore Datei

```
# no tex created files
*.log
*.tcp
*.toc
*.aux
*.nav
*.out
*.snm
*.bbt
*.blg
# ignore all files with grading relevant data
Klausur/
Praktikum/solution
```

# ARBEITEN MIT GIT

## Commit History

```
git log
```

```
commit 8f09fd3218d88fd9fa6e2109bc4d1b7fa80ff5e4 (HEAD -> master)
```

```
Author: Eva-Maria Iwer <eva-maria.iwer@hs-rm.de>
```

```
Date: Thu Jul 18 09:35:46 2019 +0200
```

```
    Beispieldatei wieder 1<C3><B6>schen
```

```
commit 233ba0aae4ccbc56d7f1dd524b87c78544d9d36e
```

```
Author: Eva-Maria Iwer <eva-maria.iwer@hs-rm.de>
```

```
Date: Thu Jul 18 09:32:20 2019 +0200
```

```
    neue Version f<C3><BC>r gitadd mit beispiel test hinzuf<C3><BC>gen
```

```
commit 9f60c5de8a2aa8051d770df5cf69e5d7b2edcfae
```

```
Author: Eva-Maria Iwer <eva-maria.iwer@hs-rm.de>
```

```
Date: Wed Jul 17 15:12:31 2019 +0200
```

```
zeusch
```

# VERTEILTES ARBEITEN MIT GIT

## Git-Server der HS-RM

- Im LDAP-Server muss eine Mailadresse hinterlegt sein, was normalerweise nicht der Fall ist.
- Um die Adresse zu setzen, bitte unter <https://ldap.local.cs.hs-rm.de/mail/index.php> einloggen und dort die Hochschul email-Adresse angeben.
- Unter <https://zenon.cs.hs-rm.de> kann jetzt das Gitlab Account angelegt werden.



## GitLab Community Edition

### Open source software to collaborate on code

Manage Git repositories with fine-grained access controls that keep your code secure. Perform code reviews and enhance collaboration with merge requests. Each project can also have an issue tracker and a wiki.

LDAP	Standard
LDAP Username	
jwer	
Password	

# VERTEILTES ARBEITEN MIT GIT

## Git-Server der HS-RM - neues Projekt hinzufügen



# VERTEILTES ARBEITEN MIT GIT

## Git-Server der HS-RM - neues Projekt hinzufügen

Blank project      Create from template      Import project

Projektname  
SWT-Unterlagen

Projekt-URL  
<https://zenon.cs.hs-rm.de/iwer/>

Projekt-Slug  
swt-unterlagen

Want to house several dependent projects under the same namespace? [Create a group](#).

Project description (optional)  
Vorlesungskript und Praktikumsblätter für SWT WS2019/2020

Visibility Level ?

 Privat  
Jedem/Jeder Benutzer(in) muss explizit der Zugriff auf das Projekt gewährt werden.

 Intern  
Auf das Projekt kann jede(r) angemeldete Nutzer(in) zugreifen.

 Öffentlich  
Auf das Projekt kann ohne Authentifizierung zugegriffen werden.

**Initialize repository with a README**  
Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

Create project Cancel

# VERTEILTES ARBEITEN MIT GIT

## Übersicht Befehle

### Git global setup

```
git config --global user.name "Eva Iwer"  
git config --global user.email "eva-maria.iwer@hs-rm.de"
```

### Create a new repository

```
git clone git@gitlab.cs.hs-rm.de:iwer/swt-unterlagen.git  
cd swt-unterlagen  
touch README.md  
git add README.md  
git commit -m "add README"  
git push -u origin master
```

### Push an existing folder

```
cd existing_folder  
git init  
git remote add origin git@gitlab.cs.hs-rm.de:iwer/swt-unterlagen.git  
git add .  
git commit -m "Initial commit"  
git push -u origin master
```

### Push an existing Git repository

```
cd existing_repo  
git remote rename origin old-origin  
git remote add origin git@gitlab.cs.hs-rm.de:iwer/swt-unterlagen.git  
git push -u origin --all  
git push -u origin --tags
```

# BRANCHES

## Einleitung

- Nahezu jedes VCS unterstützt eine Form von Branching
- Branching bedeutet, dass Sie von der Hauptlinie der Entwicklung abzweigen und Ihre Arbeit fortsetzen
- GIT ist unglaublich leichtgewichtig, wodurch Branch-Operationen nahezu verzögerungsfrei ausgeführt werden
- leichtes Hin- und Herschalten zwischen einzelnen Entwicklungszweigen

# BRANCHES

## Branches auf einen Blick

- Wenn Sie einen Commit durchführen, speichert Git ein Commit-Objekt, das einen Zeiger auf den Snapshot des von Ihnen bereitgestellten Inhalts enthält
- zeigt auf den Commit oder die Commits, die direkt vor diesem Commit stattfanden (zu seinem Vorgänger bzw. seinen Vorgängern)
- Besonderheiten: keine Vorgänger für den ersten Commit, einen Vorgänger für einen normalen Commit und mehrere Vorgänger für einen Commit, welcher aus dem Zusammenführen (engl. merge) von zwei oder mehr Branches resultiert.

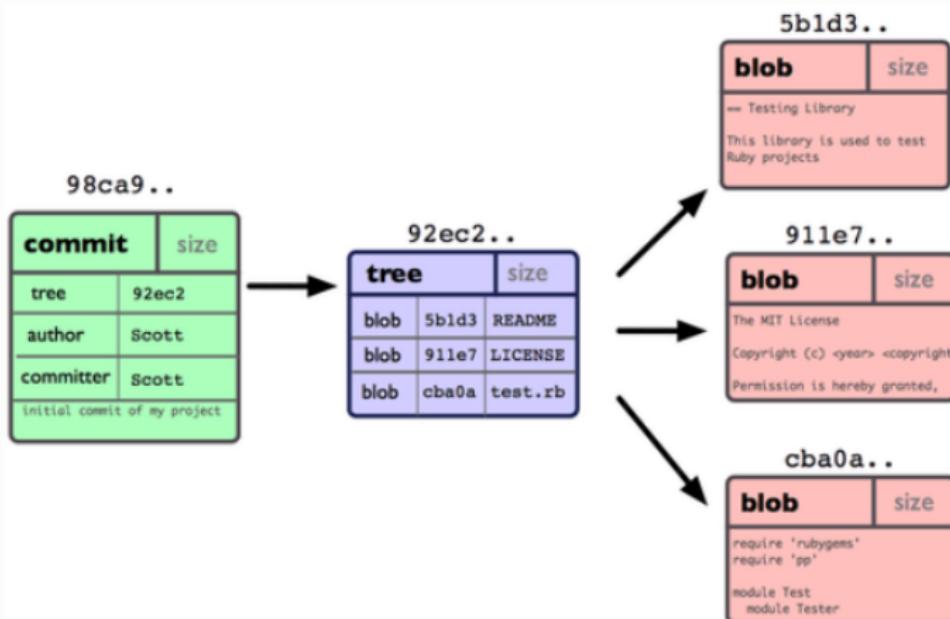
# BRANCHES

## Branches auf einen Blick

- Bei Commit wird Prüfsumme erstellt
- Speichert diese Version der Datei im Git-Repository (Git verweist auf diese als blobs) und fügt die Prüfsumme der Staging-Area hinzu
- Ihr Git-Repository enthält jetzt verschiedene Objekte: ihre blobs (die jeweils den Inhalt einer der drei Dateien repräsentieren), ein tree-Objekt, welches den Inhalt des Verzeichnisses auflistet und angibt, welcher Dateiname zu welchem Blob gehört, und ein commit-Objekt mit dem Zeiger, der auf die Root des Projektbaumes und die Metadaten des Commits verweist.

# BRANCHES

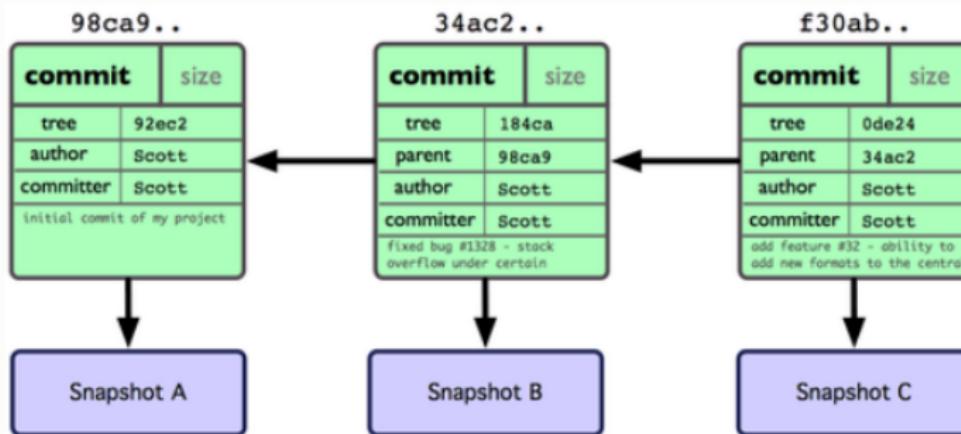
## Commit und sein Tree



# BRANCHES

## Commits und ihre Vorgänger

Wenn Sie einige Änderungen vornehmen und wieder einen Commit durchführen, speichert dieser einen Zeiger zu dem Commit, der unmittelbar davor gemacht wurde.



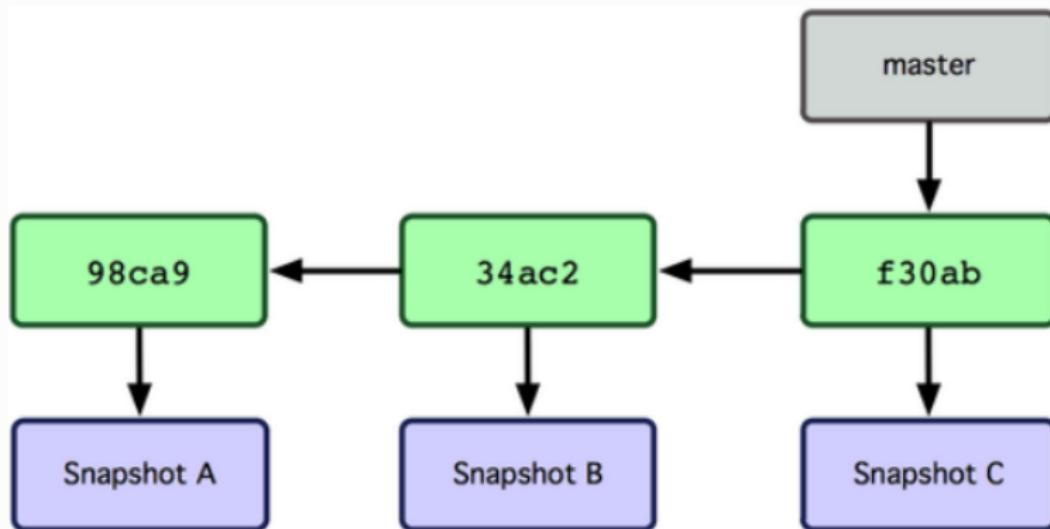
# BRANCHES

## Master

- Ein Branch in Git ist einfach ein leichter, beweglicher Zeiger auf einen dieser Commits.
- Die Standardbezeichnung für einen Branch bei Git lautet master.
- Wenn Sie damit beginnen, Commits durchzuführen, erhalten Sie einen master-Branch, der auf den letzten Commit zeigt, den Sie gemacht haben.
- Jedes Mal, wenn Sie einen Commit durchführen, bewegt er sich automatisch vorwärts.

# BRANCHES

## Ein Branch und sein Commit-Verlauf



# BRANCHES

## Erzeugen eines neuen Branches

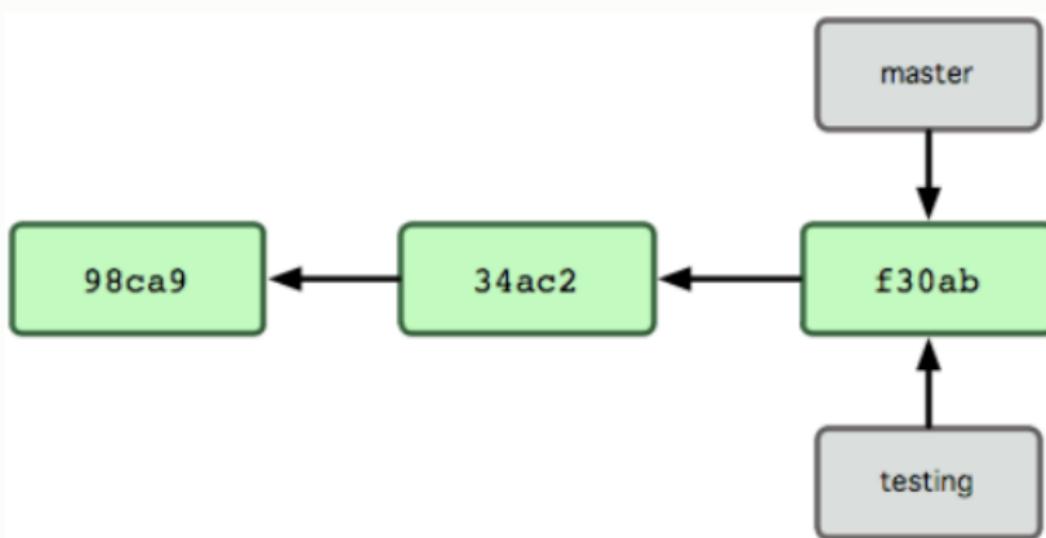
Was passiert, wenn Sie einen neuen Branch erzeugen?  
Ein neuer Zeiger erstellt, mit dem Sie sich in der Entwicklung  
fortbewegen können  
Bsp Brunch testing:

```
git branch testing
```

Dieser Befehl erzeugt einen neuen Zeiger, der auf den selben  
Commit zeigt, auf dem Sie sich gegenwärtig befinden.

# BRANCHES

Zwei Branches, die auf die selbe Serie von Commits zeigen



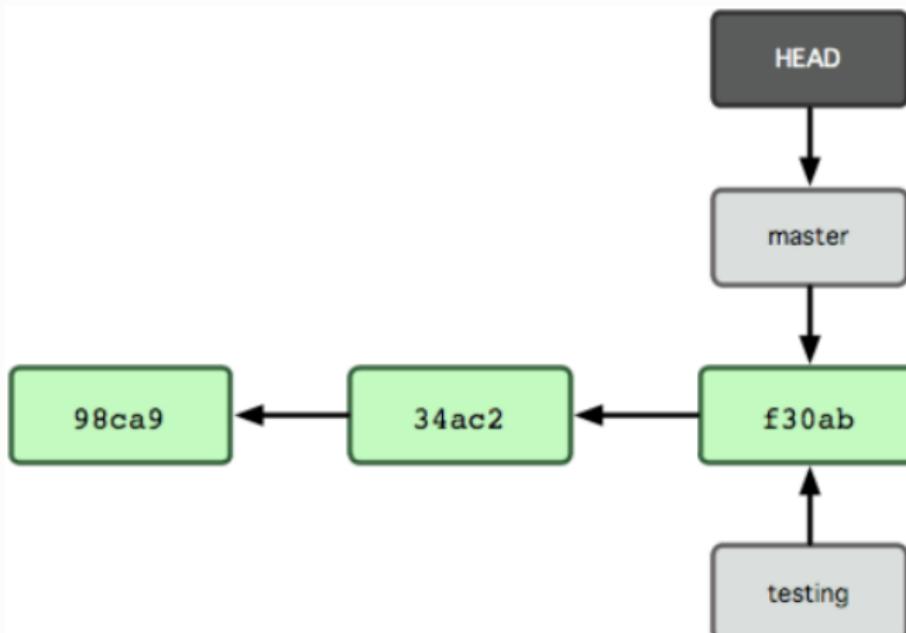
# BRANCHES

## Woher weiß Git, auf welchem Branch Sie gegenwärtig sind?

- speziellen Zeiger namens HEAD
- Zeiger auf den lokalen Branch, auf dem Sie sich gegenwärtig befinden
- Anweisung git branch hat den neuen Branch nur erzeugt, aber nicht zu diesem gewechselt.

# BRANCHES

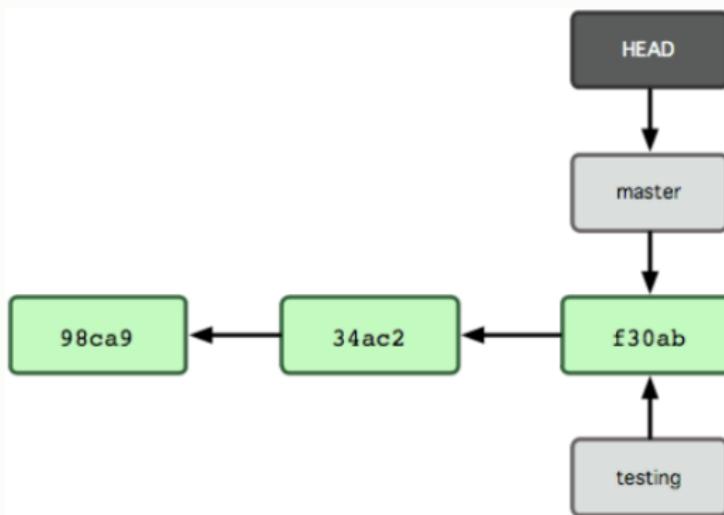
## Auf einen Branch zeigender HEAD



# BRANCHES

## Wechseln der Branches

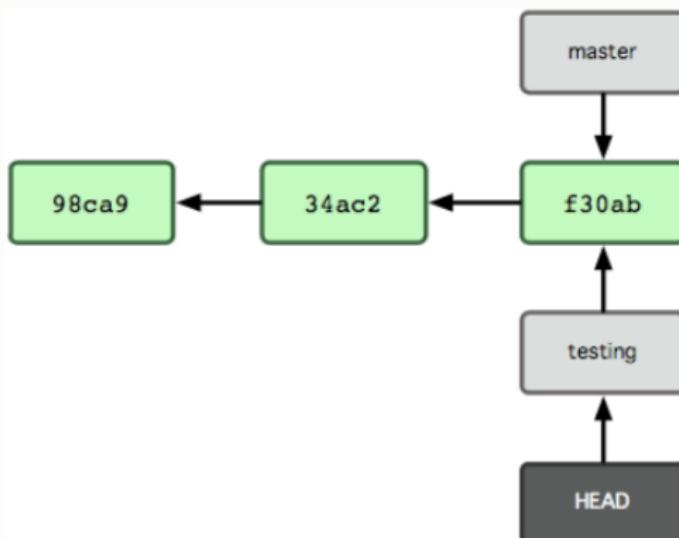
```
git checkout testing
```



# BRANCHES

**HEAD zeigt auf den aktuellen Branch**

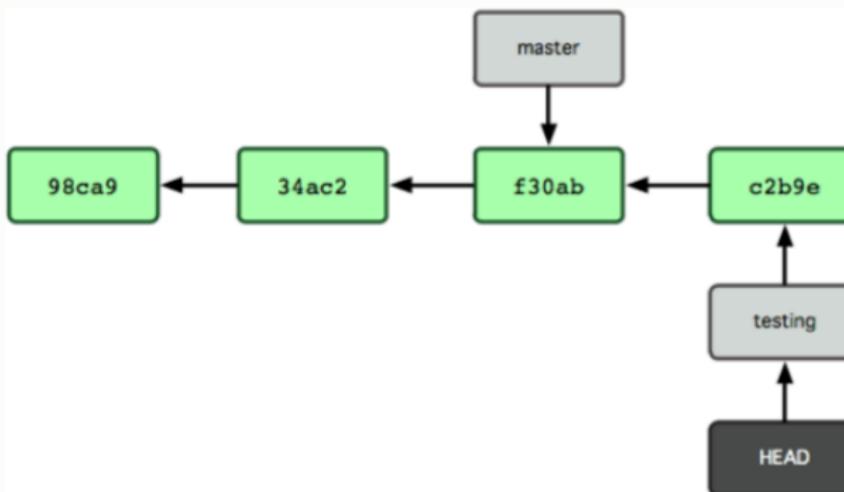
```
git checkout testing
```



# BRANCHES

**Der Branch, auf den HEAD zeigt, bewegt sich vorwärts, wenn ein Commit gemacht wird**

```
vim test.rb  
git commit -a -m 'made a change'
```



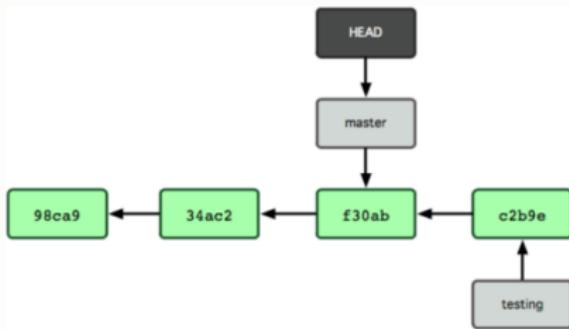
## BRANCHES

**HEAD bewegt sich, wenn Sie auschecken**

Der Branch, auf den HEAD zeigt, bewegt sich vorwärts, wenn ein Commit gemacht wird

Wechseln zum master-Branch.

```
git checkout master
```



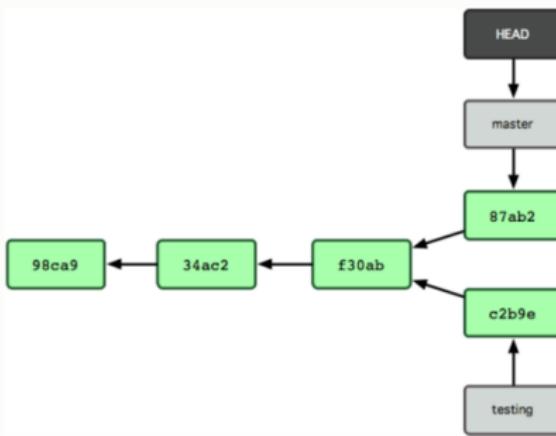
# BRANCHES

## Verzweigter Verlauf

Wenn Sie jetzt noch weitere Änderungen hinzufügen, entsteht ein verzweigter Verlauf

```
vim test.rb
```

```
git commit -a -m 'made other changes'
```



# MERGING?

Dazu mehr im Praktikum

# LITERATUR

## nützliche Bücher und Links

- <https://git-scm.com/book/en/v2>
- <http://gitbu.ch/pr01.html>

# ABKÜRZUNGEN

# VERSIONIERUNG

**VCS** Version Control System

**DVCS** Verteilte Versionsverwaltungssysteme

**CVCS** Zentralisierte Versionskontrolle

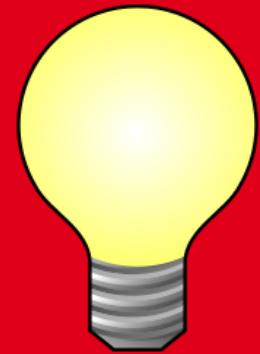


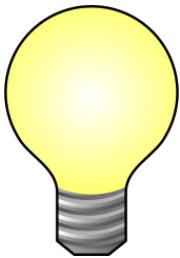
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# 11.12.2020

# Aktivitätsdiagramme

Aktivitätsdiagramme in UML





Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# AGENDA

Einführung ins Thema

Loslegen

Beispiel

Modellelemente im Überblick

Analyse von Nebenläufigkeit

Fazit

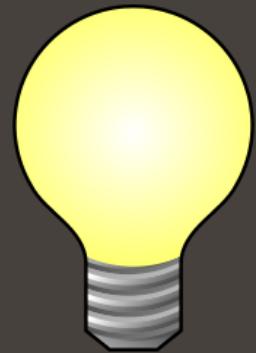


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

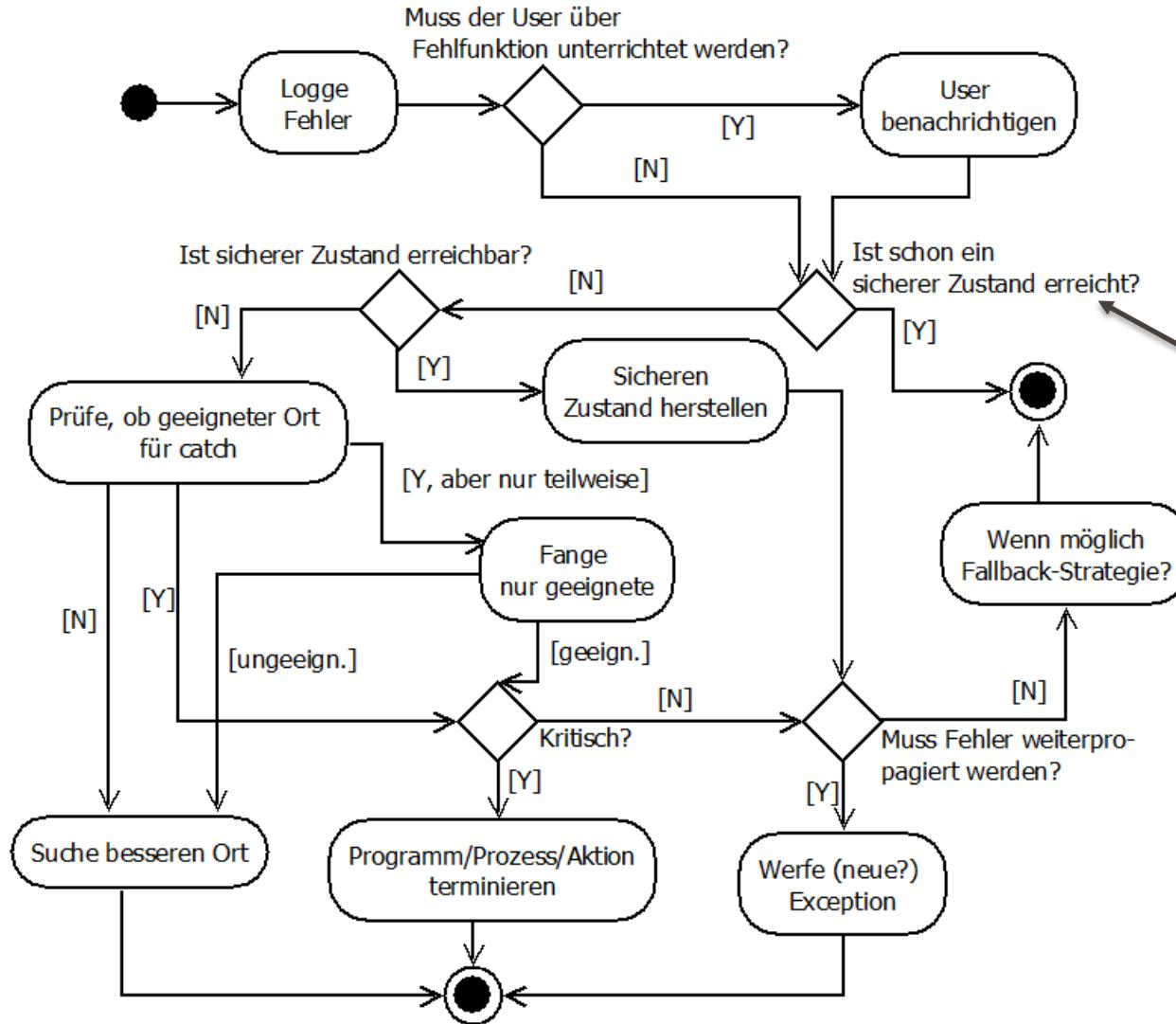
# 01

## EINFÜHRUNG INS THEMA

Ziel:  
Die Eckpunkte des Themas kennenlernen



# AUS DER PM-VORLESUNG: VORGEHEN BEHANDLUNGS- STRATEGIE IN REISSELINE



Bedingungsspezif.  
hier ist **KEIN**  
korrektes UML!  
→ Müsste so heißen:

```
<<decisionInput>>  
Ist schon ein sicherer  
Zustand erreicht?
```



# AKTIVITÄTSIDIAGRAMME



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

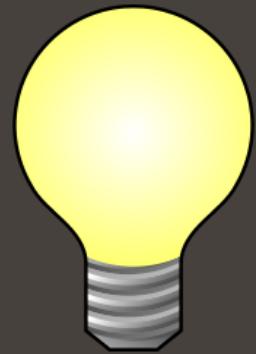
- Eignen sich zur Modellierung von:
    - Aktionen und deren Ablauf / Zusammenhang
      - Prozesse (Geschäftsprozesse und andere)
      - Algorithmen
- Darstellung komplizierter Abläufe mit Schleifen & Verzweigungen
- Gleichmäßiges Fließen  
(im Gegensatz zu stockender Abarbeitung bei Zustandsdiagrammen)



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## 02 Loslegen

Ziel:  
Erste Sachen kennenlernen





# VORSICHT:



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

Aktivitätsdiagramme haben:

- (Fast) gleiche Syntax (gleiches Aussehen) wie Zustandsdiagramme, aber komplementäre Semantik (Bedeutung)

Modellelement	Zustandsautomat	Aktivitätsdiagramm
	Zustand (oft: nichts passiert direkt)	Aktion
	Übergang (Aktion)	Verbindet Aktionen (im „→“ passiert nichts)

# BEVORZUGTE EINSATZGEBIETE



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Ablauf / Prozesse / Algorithmen
  - Siehe Flussdiagramm
- Zusammenarbeit bei Nebenläufigkeit
  - Siehe Petri-Netze

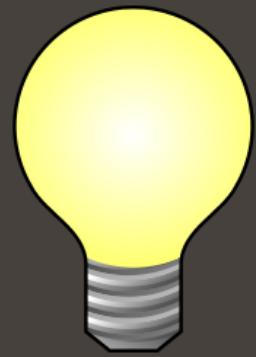


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## 03 Beispiel

Ziel:

Ein Beispiel





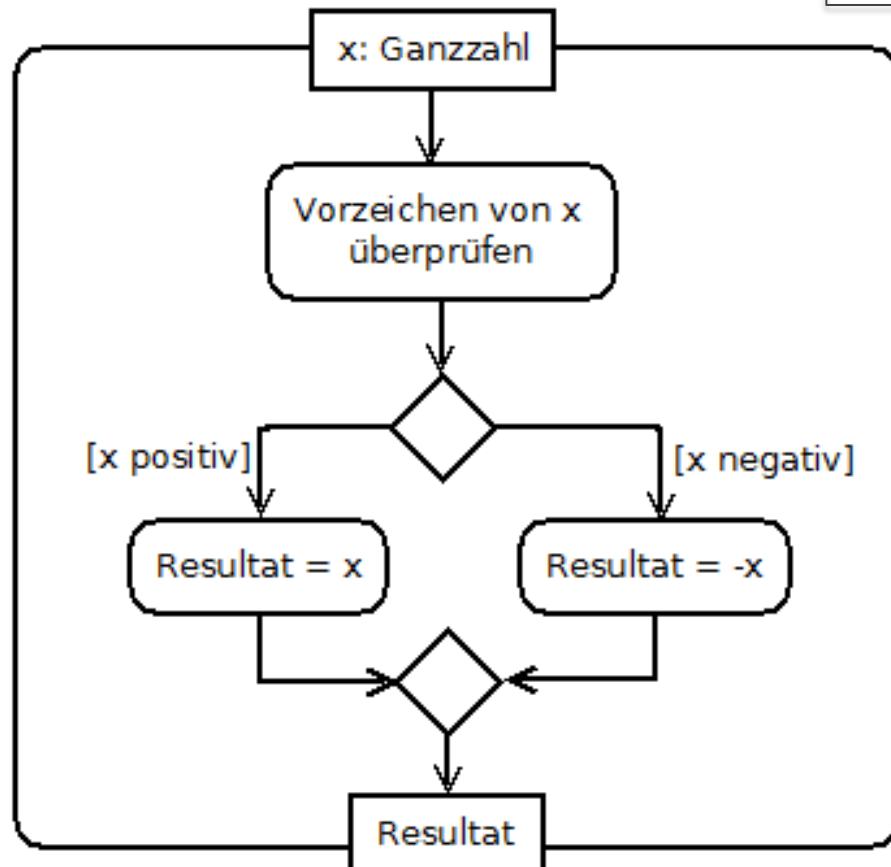
# BEISPIEL

- Code einer Betragsfunktion:

```
public static int betrag(int x) {  
    if (x>=0) {  
        return x;  
    }  
    else {  
        return -x;  
    }  
}
```

# BEISPIEL

- Betragsfunktion als Aktivitätsdiagramm:



```
public static int betrag(int x) {  
    if (x>=0) {  
        return x;  
    }  
    else {  
        return -x;  
    }  
}
```

→ Grafische Darstellung eines Algorithmus

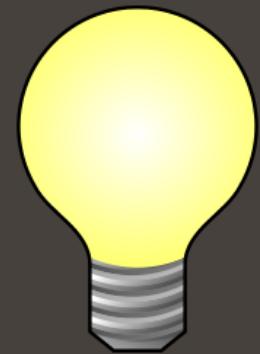


04

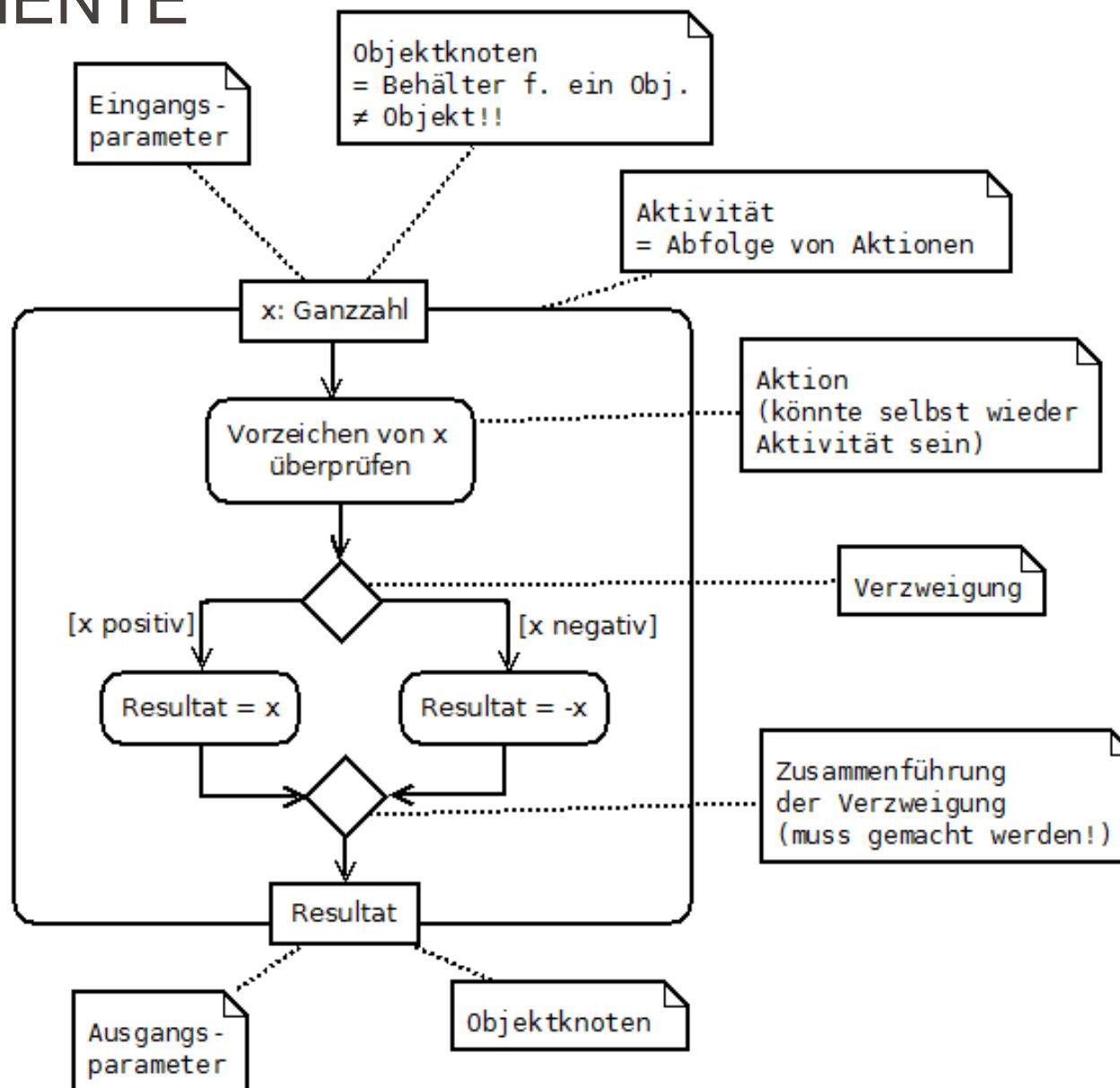
## Aktivitätsdiagramme - Modellelemente im Überblick

Ziel:

Die Elemente im Überblick erfassen



# DIE WESENTLICHEN ELEMENTE



# START- & ENDE- SYMBOLE



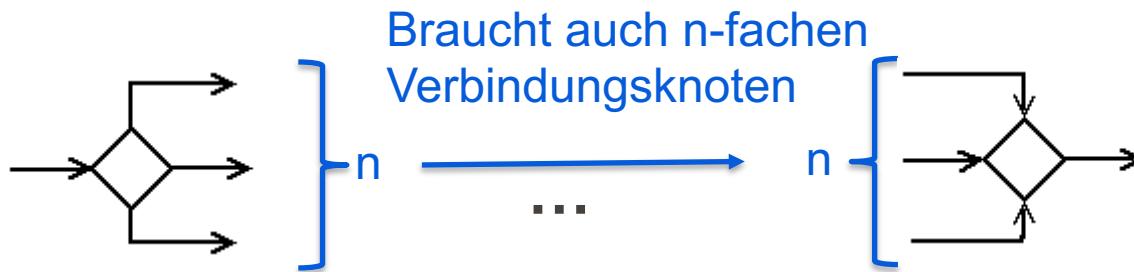
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Start:
  - Start des Aktivitätsdiagramms (Initial Node)
- Ende:
  - Komplettes Ende des Aktivitätsdiagramms  
(Activity Final Node)
  - Nur Ende des Pfads (Flow Final Node)

# VERZWEIGUNGEN



Einfache Verzweigungen (alternative Wege, keine Parallelität):



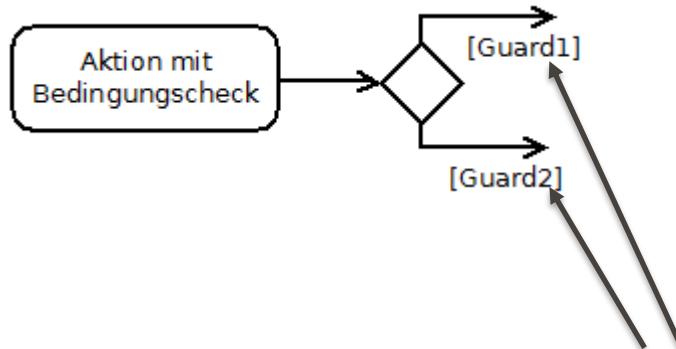
- Für jede öffnende Raute muss auch eine schließende Raute gemacht werden
  - Ansonsten Probleme mit checks bei Nebenläufigkeit
  - Token-Konzept (siehe folgendes Kap.) funktioniert dann nicht

# VERZWEIGUNGEN – WIE BEDING. FORMULIEREN?

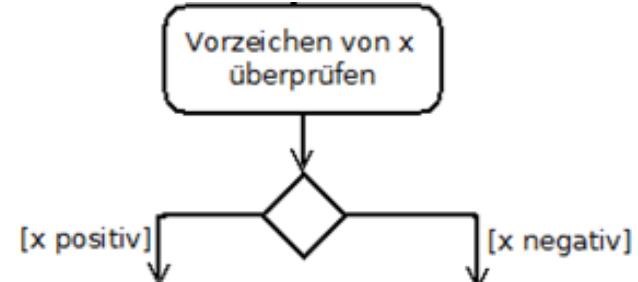


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

1. Möglichkeit: Die Aktion vorher liefert die Bedingung:

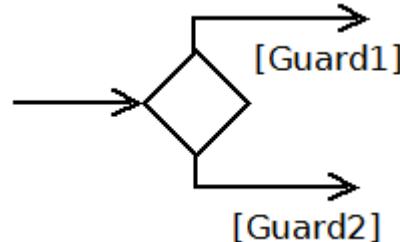


BSP (siehe vorher):

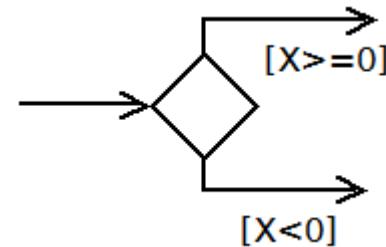


Guards nicht vergessen!!

2. Möglichkeit: Spezifikation am Entscheidungsknoten über  
Guards (Wächter)



BSP:



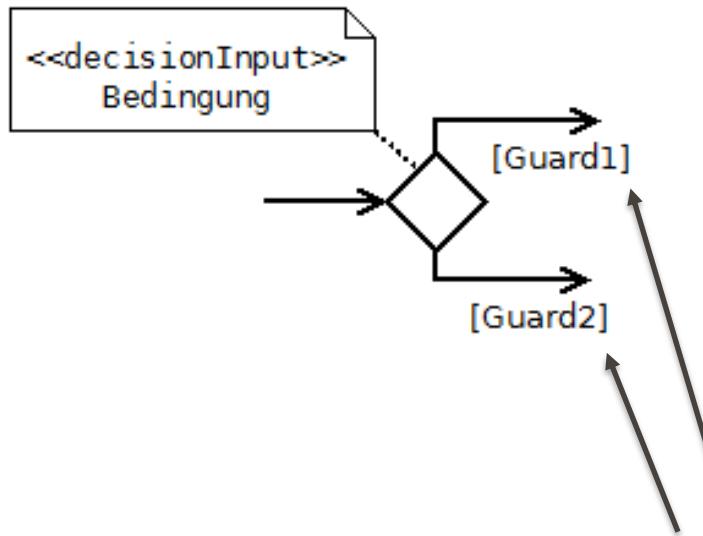
→ Eignet sich eher für einfache Bedingungen

# VERZWEIGUNGEN – WIE BEDING. FORMULIEREN?

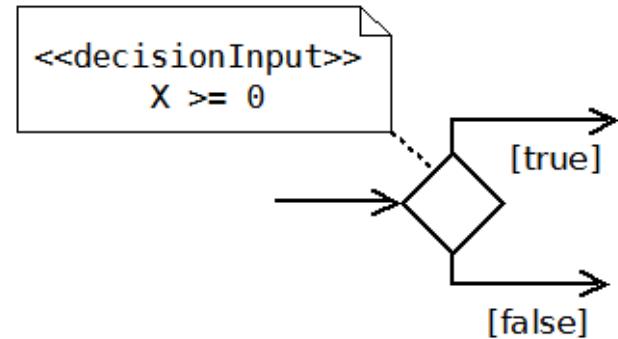


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

3. Möglichkeit: Spezifikation am Entscheidungsknoten über  
Kommentar mit <<decisionInput>>-Stereotypen



BSP:



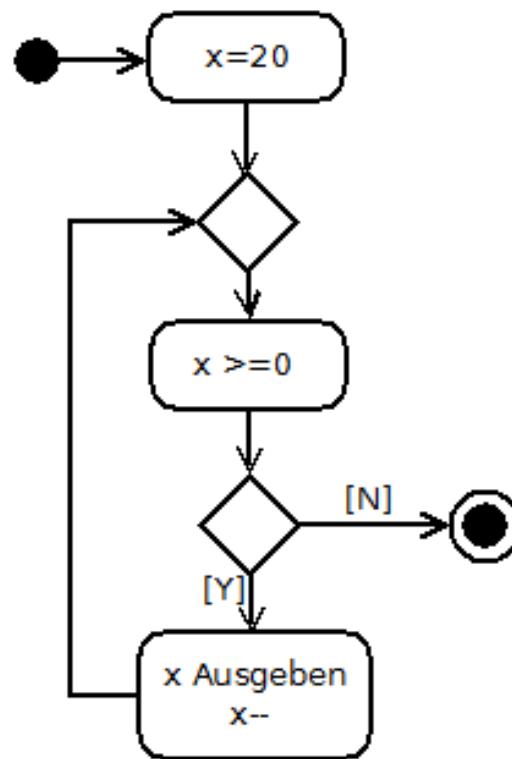
Guards nicht vergessen!!

→ Eignet sich eher für komplexere Bedingungen

# WIE SCHLEIFEN MODELLIEREN?



- Schleifen können über Verzweigungskomponente modelliert werden:



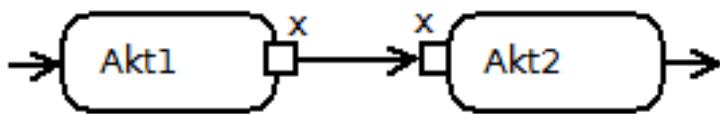
→ Alternativ: Schleifenknoten verwenden  
– Siehe [UML Glaskar; S. 317 ff].



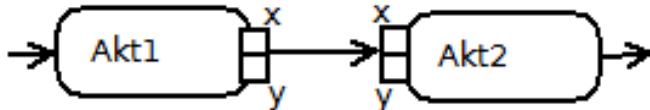
# DIE PIN-NOTATION

Pins zeigen Objektflüsse an:

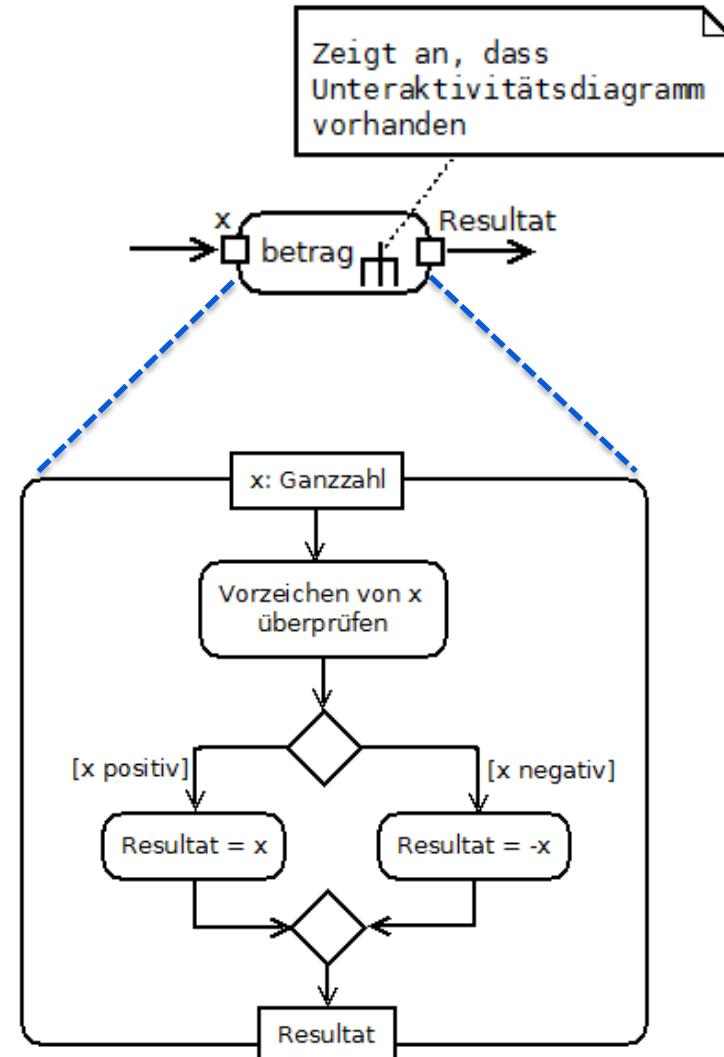
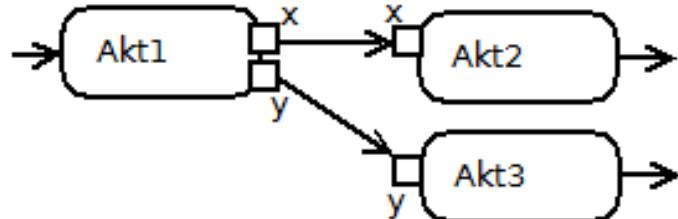
- Einzelne Objekte:



- Objektmengen:



- Objektfl. (UND-Semantik):

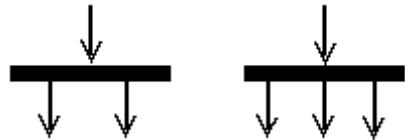


# NOTATION VON NEBENLÄUFIGKEIT

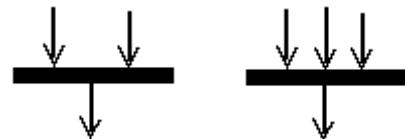


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Prozesse teilen (fork):



- Prozesse zusammenführen (join):



- Prozesse müssen immer auch wieder zusammengeführt werden, oder → muss verwendet werden

# WAS IST NEBENLÄUFIGKEIT?



- Auch Parallelität oder Concurrency genannt  
→ Es können Sachen parallel abgearbeitet werden
- Einfachstes Beispiel: Threads (siehe letztes Semester PM)

```
private void myAlgorithm() {  
    doTask1();  
  
    Thread th=new Thread(() -> {  
        doTask2();  
    }  
);  
th.start();  
  
doTask3();  
th.join();  
}
```

- ABER: Nicht nur, sondern auch bei Geschäftsprozessen, Client-Server-Abläufe, Verteilte Systeme (SOA, RPCs), ...

# WAS IST NEBENLÄUFIGKEIT?



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

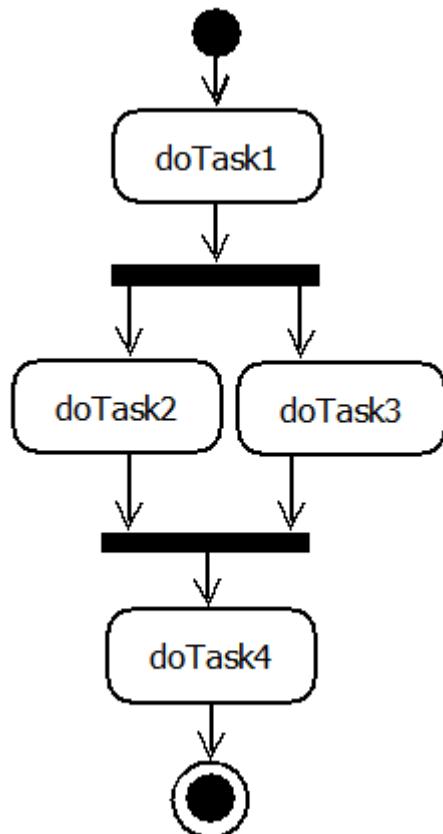
- Aufgabe: Das Bsp. modellieren

```
private void myAlgorithm() {  
    doTask1();  
  
    Thread th=new Thread(()-> {  
        doTask2();  
    });  
    th.start();  
  
    doTask3();  
  
    th.join();  
  
    doTask4();  
}
```



# WAS IST NEBENLÄUFIGKEIT?

- Aufgabe: Das Bsp. modellieren



```
private void myAlgorithm() {  
    doTask1();  
  
    Thread th=new Thread(()-> {  
        doTask2();  
    });  
    th.start();  
  
    doTask3();  
  
    th.join();  
  
    doTask4();  
}
```

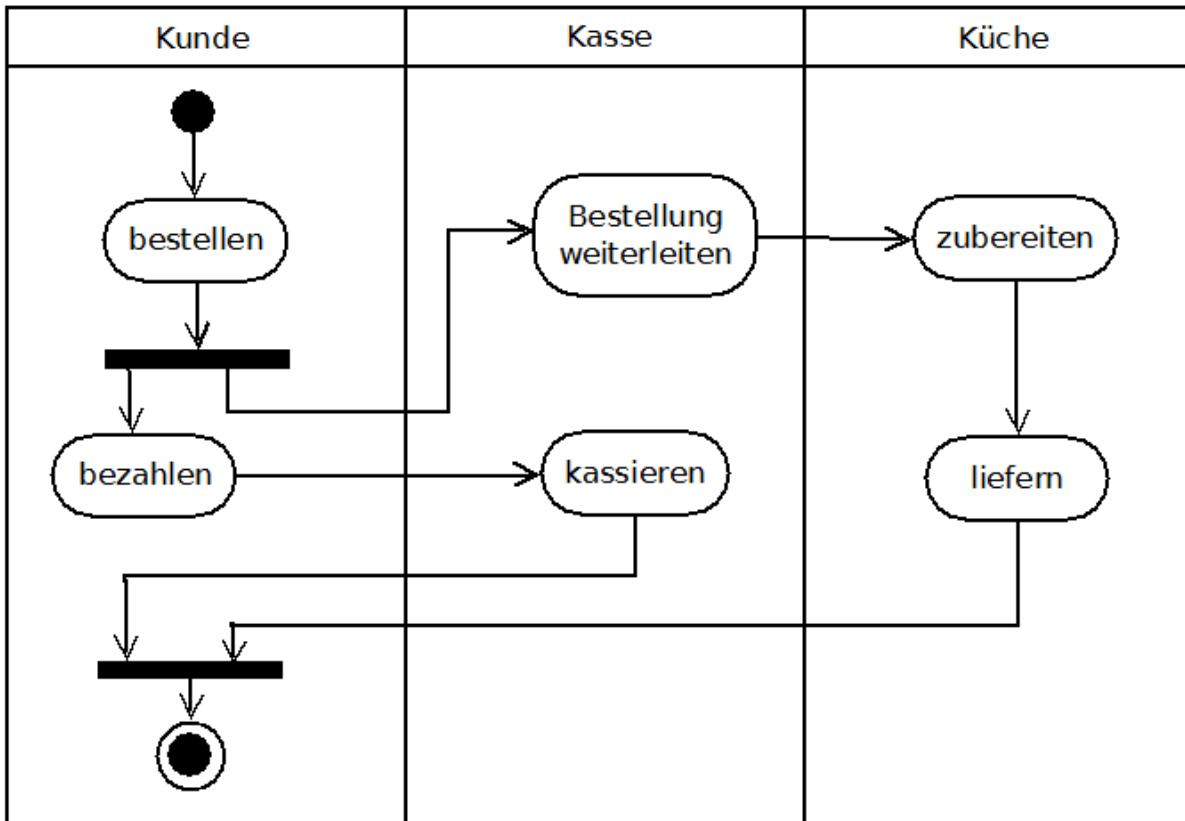
# AKTIVITÄTSBEREICHE – („ACTIVITY PARTITIONS“)

Name



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Inoffiziell: Schwimmbahnen (Swimlanes)
- Erlauben Mapping von Aktionen auf Systeme / Stakeholder:

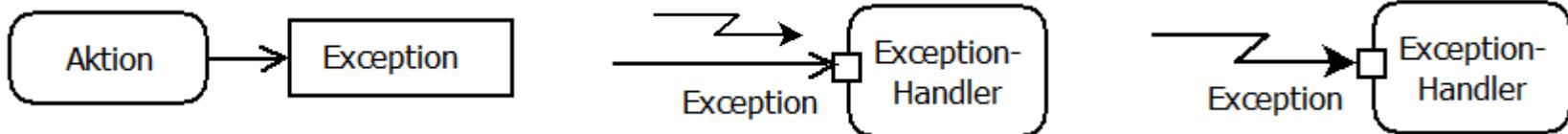


→ Beispiel eines Geschäftsprozesses

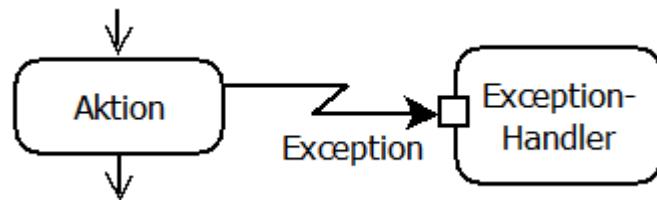
# WERFEN UND FANGEN VON AUSNAHMEN



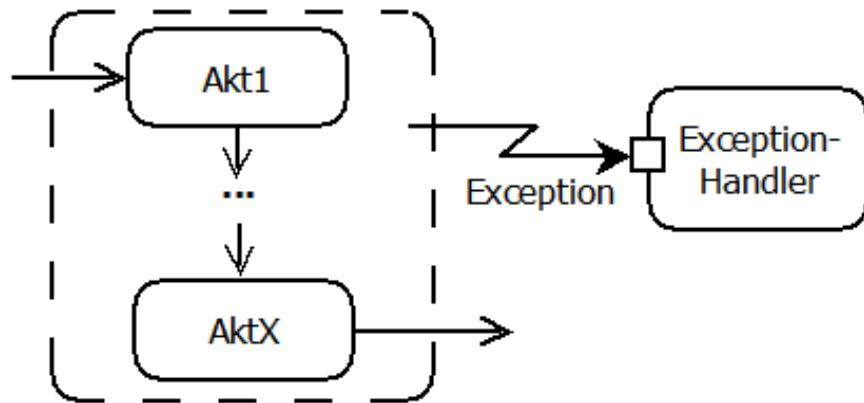
- Mehrere Möglichkeiten der Darstellung:



- Kann direkt aus einer Aktion/Aktivität kommen:



- Oder es wird ein Unterbrechungsbereich definiert:

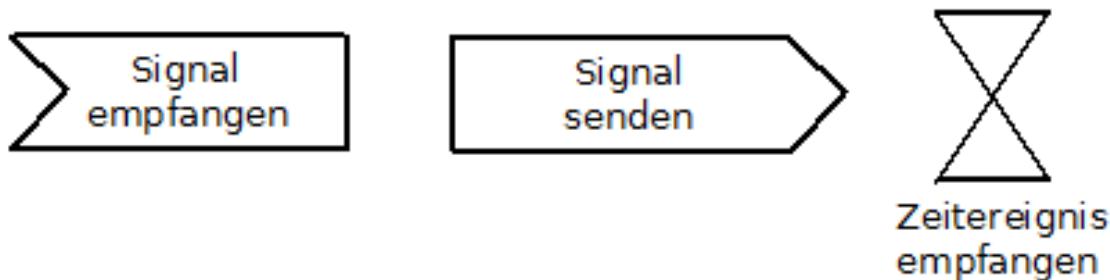


# WEITERE DARSTELLUNGS-MITTEL



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

Signale:



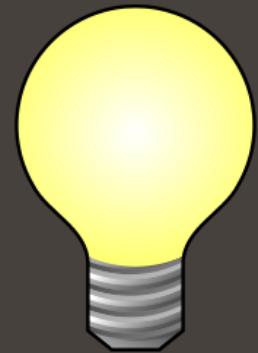


05

## Analyse von Nebenläufigkeit

Ziel:

Das formale Modell des Aktivitätsdiagramms  
erlaubt die korrekte Analyse der Nebenläufigkeit

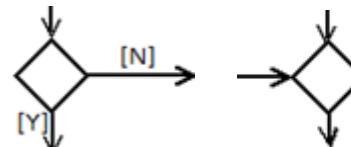




# DEF: KONTROLLKNOTEN

- Als Kontrollknoten werden alle Knoten genannt, die keine Aktionen oder Aktivitäten sind:

- Verzweigung und Merge:



- Start- und Endknoten:



- Fork and Join:  
(Nebenläufigkeit, z.B. Threads)



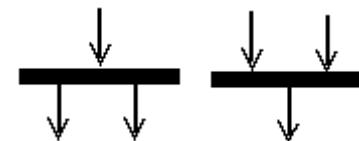
# DAS TOKEN-KONZEPT



- Das Token-Konzept bildet die Grundlage der Semantik von Aktivitätsdiagrammen
  - Das Token-Konzept eignet sich sog. Verklemmungen (Deadlocks) aufzudecken
  - Vgl. Petri-Netze

## IDEE:

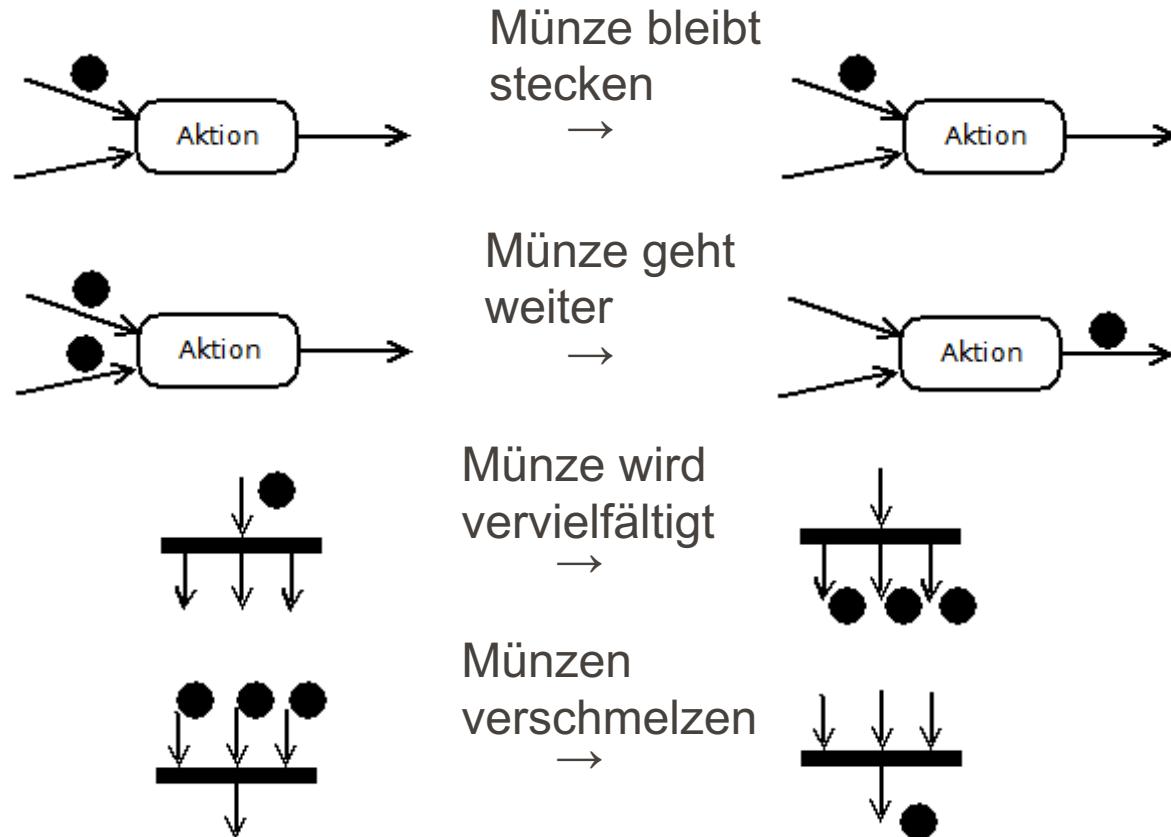
- Ein oder mehrere Token (Münzchips) zeigen / zeigen an in welchem Zustand sich die Verarbeitung gerade befindet.
- Token laufen durch das Diagramm → Kontroll-/Datenfluss
- Token wird erst weitergeleitet, wenn alle Kanten und Zielknoten bereit sind.
- Token darf an Kontrollknoten nicht warten
  - Ausnahme Join-Knoten:  
→ Synch: Warten bis alle Pfade da





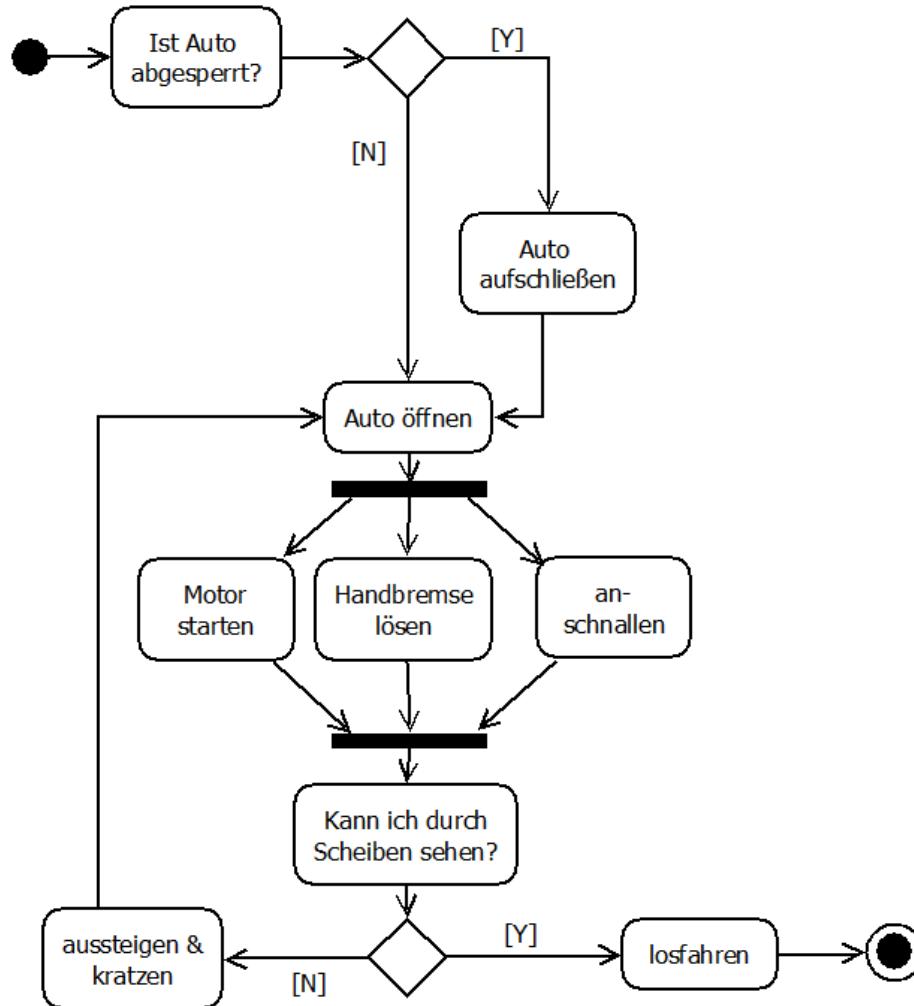
# DAS TOKEN-KONZEPT

- Münzen zirkulieren (reines Gedankenkonstrukt):
  - Eine Aktion schaltet die Münze(n) nur weiter, wenn an allen Eingängen Münzen anliegen:



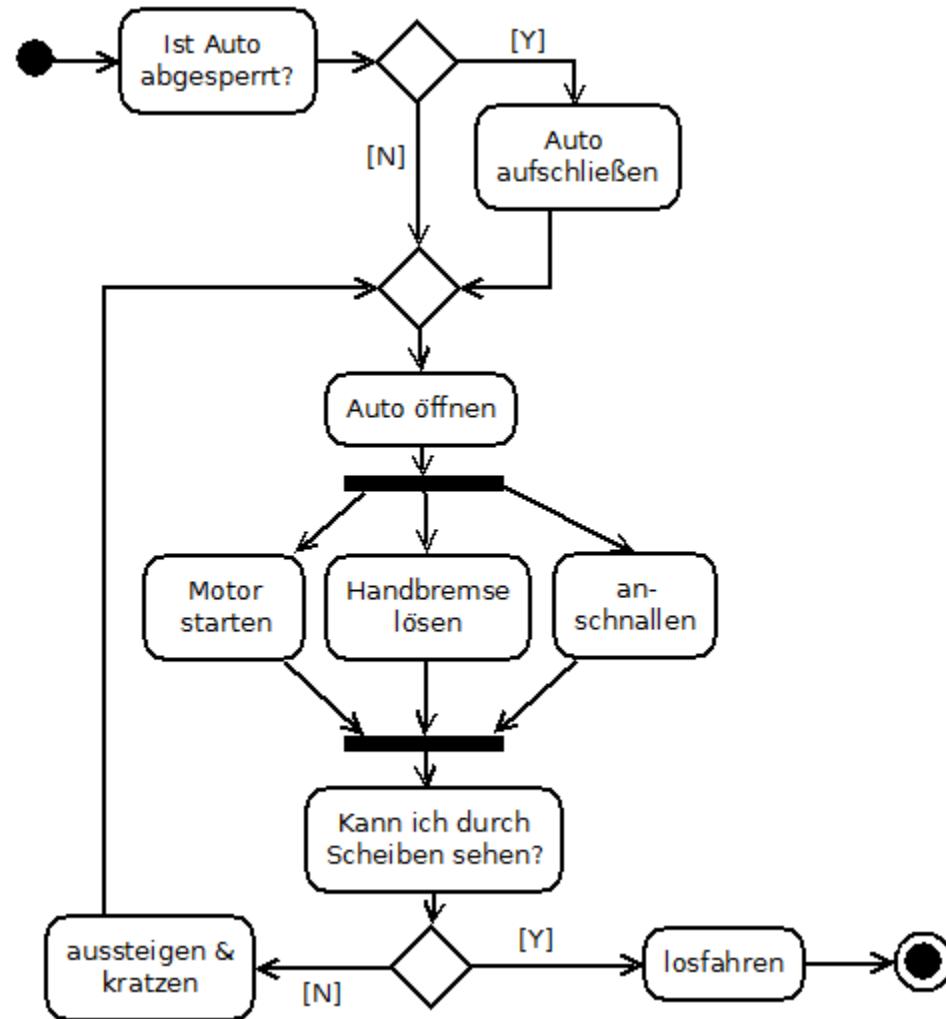
# DAS TOKEN-KONZEPT

- Beispiel:



# DAS TOKEN-KONZEPT

- Beispiel:



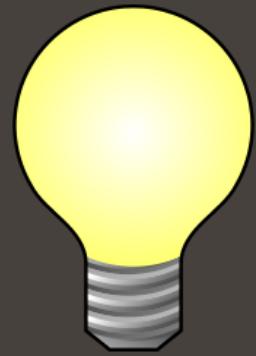


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# 06

# Fazit

Ziel:  
Was haben wir damit gewonnen?





# WAS HABEN WIR GELERNT?

- Das Aktivitätsdiagramm
  - Hilft Prozesse oder Algorithmen zu modellieren
  - Aktivität, Aktion, Pins, Verzweigungen und Fork-Join-Balken
- Token-Konzept hilft bei der Analyse von Nebenläufigkeiten
  - Deadlocks aufdecken



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

AUF GEHT'S!!

SELBER MACHEN UND LERNEN!!



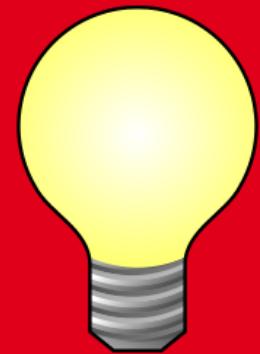


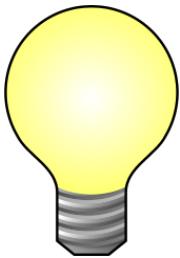
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

17.12.2020

# Interaktionsdiagramme

Interaktionsdiagramme nutzen





Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# AGENDA

Einführung ins Thema

Sequenzdiagramme

Sequenzdiagramme – 2 verschiedene Semantiken

Kommunikationsdiagramme

Weitere Interaktionsdiagramme

Fazit

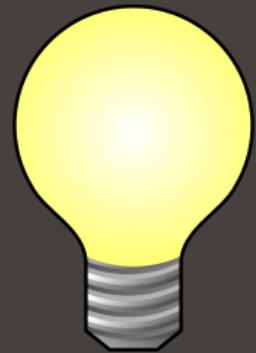


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# 01

## EINFÜHRUNG INS THEMA

Ziel:  
Die Eckpunkte des Themas kennenlernen

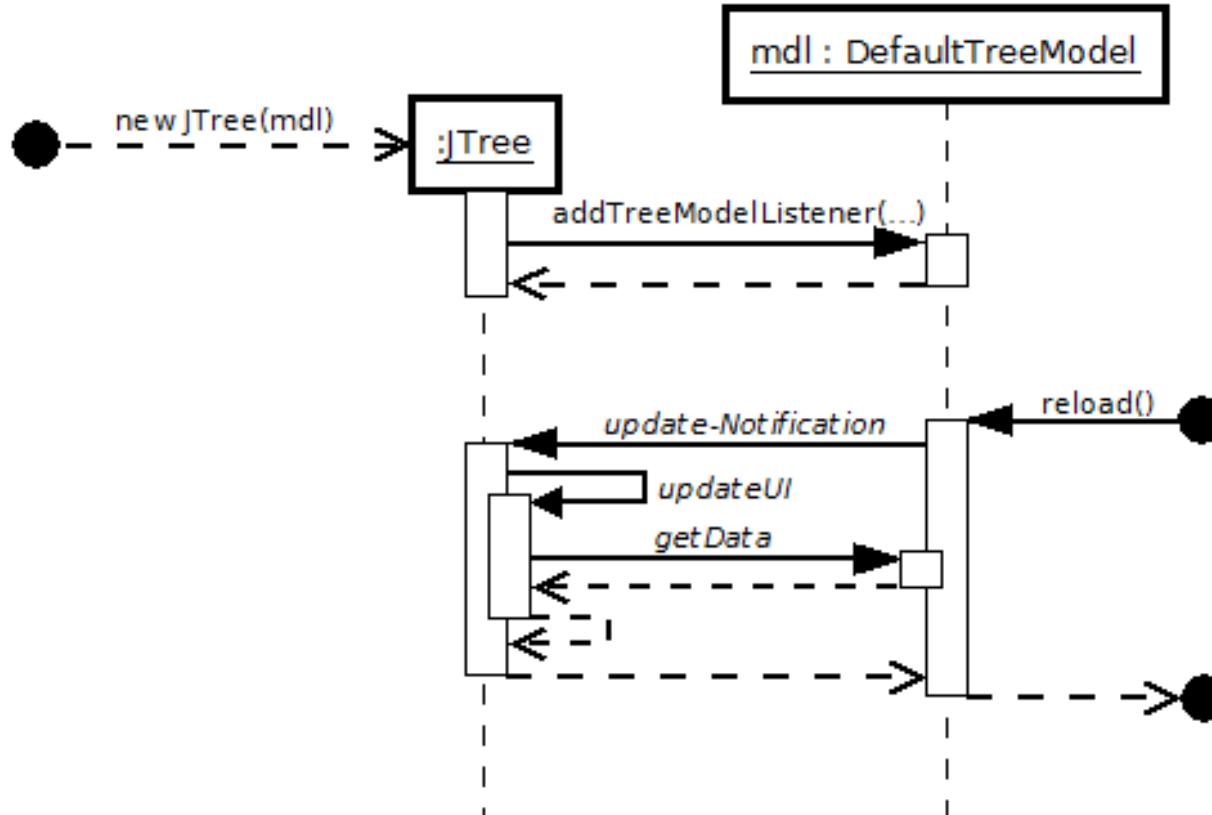


# 2. SEMESTER, PM: DEN JTREE VERÄNDERN – HINTERGRÜNDE VON RELOAD()



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

Sequenz-Diagramm über den groben Ablauf:



Hinweis:

Kursiv gesetzten Ausdrücke `update-Notification`, `updateUI` und `getData` zeigen nur ungefähr den Ablauf, die wahren Namen der Funktionen sind Javainternas!

# INTERAKTIONSDIAGRAMME



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Zeigen die Interaktionen zwischen Objekten
  - Interaktion = Abfolge von Nachrichten, die zwischen Objekten ausgetauscht werden
- Grundsätzlich gibt es 4 Interaktionsdiagramme in UML:
  - Sequenzdiagramm (wichtigste)
  - Kommunikationsdiagramm
  - Interaktionsübersichtsdiagramm
    - Metadiagramm, das die Zusammenhänge zw. verschied. Interaktionsdiagrammen zeigt
  - Timing Diagramm:
    - Zeigt Nachrichtenaustausch und Zustandwechsel verschiedener Objekte zu bestimmten Zeitpunkten an.

} Diese besprechen wir  
im Detail

# BEVORZUGTE EINSATZGEBIETE



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Man hat mehrere Einheiten (Objekte), die Nachrichten austauschen
- Zeitliche Abfolge der Nachrichten soll dargestellt werden
- Snapshot-Charakter (ähnlich wie Objektdiagramme)

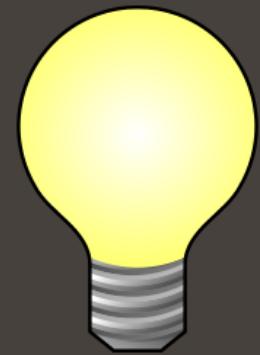


## 02

# Sequenzdiagramme

## - Modellelemente im Überblick

Ziel:  
Elemente im Überblick erfassen

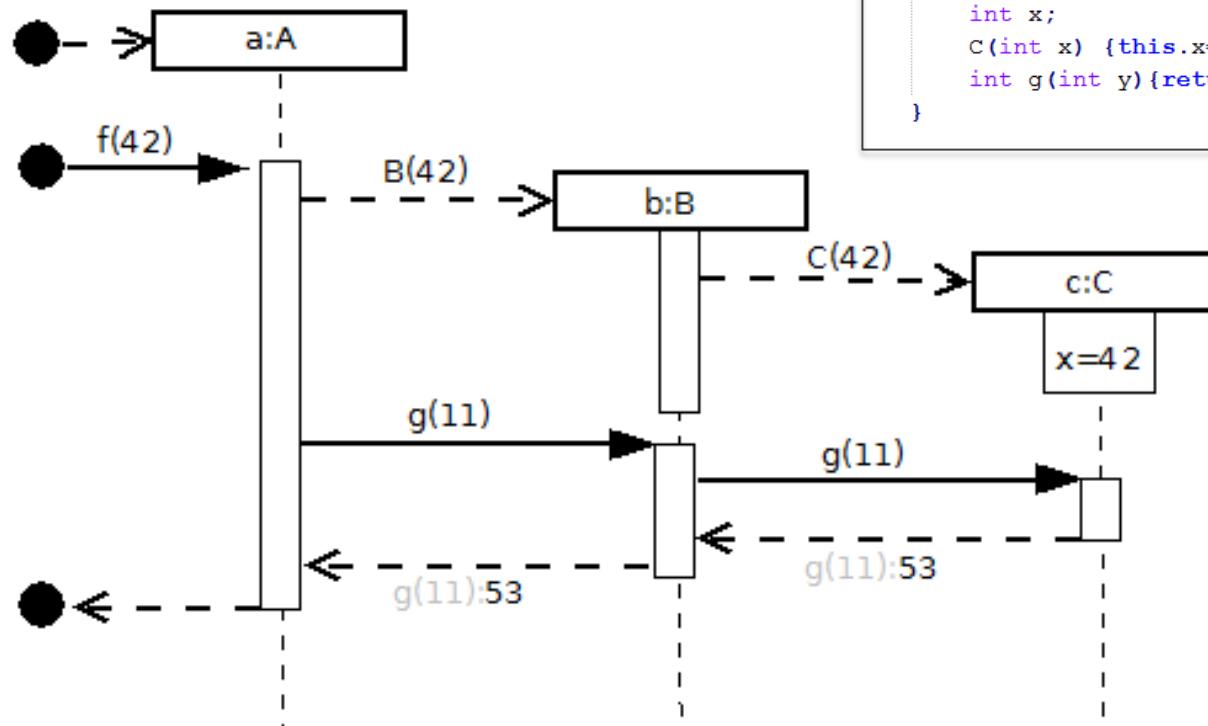


# CODE-BEISPIEL:



```
class A {  
    void f(int x){  
        B b = new B(x);  
        int z;  
        z=b.g(11);  
    }  
  
    public static void main(String [] args){  
        A a=new A();  
        a.f(42);  
    }  
}  
  
class B{  
    C c;  
    B (int x) { c=new C(x);}  
    int g(int y){return c.g(y);}  
}  
  
class C {  
    int x;  
    C(int x) {this.x=x;}  
    int g(int y){return x+y;}  
}
```

# DER CODE ALS SEQUENZDIAGRAMM:

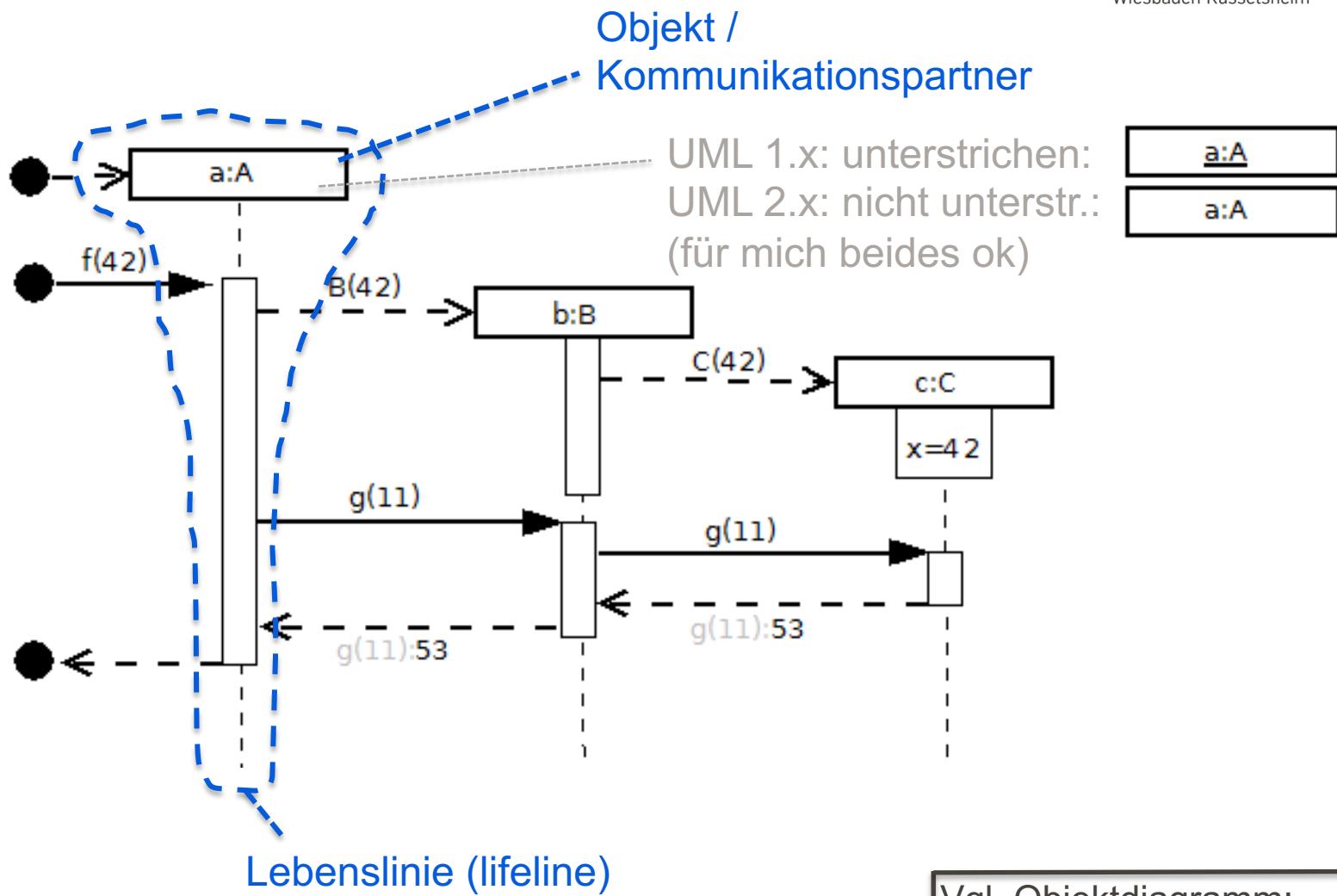


```
class A {  
    void f(int x){  
        B b = new B(x);  
        int z;  
        z=b.g(11);  
    }  
  
    public static void main(String [] args){  
        A a=new A();  
        a.f(42);  
    }  
}  
  
class B{  
    C c;  
    B (int x) { c=new C(x); }  
    int g(int y){return c.g(y);}  
}  
  
class C {  
    int x;  
    C(int x) {this.x=x;}  
    int g(int y){return x+y;}  
}
```

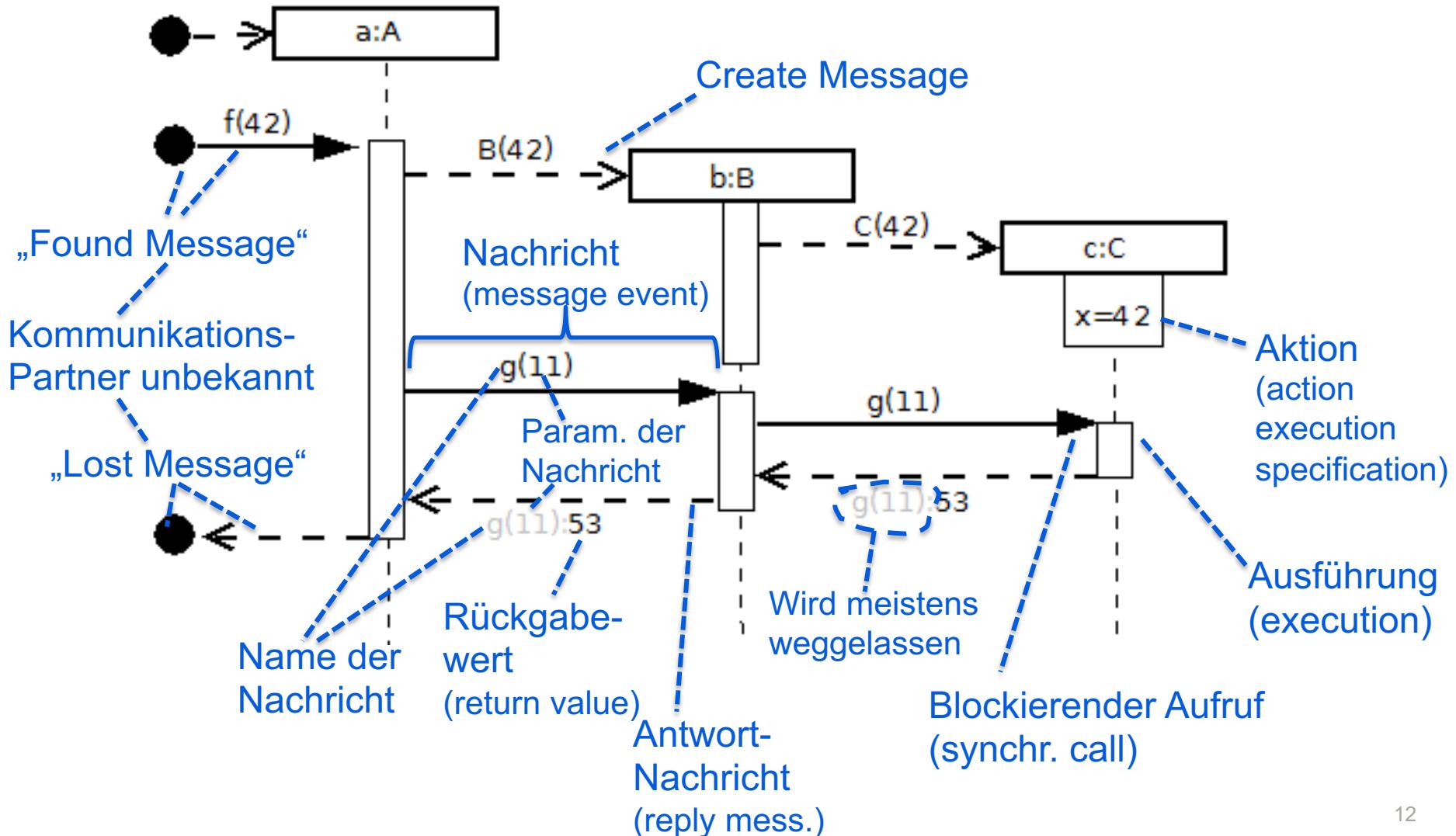
# DIE ELEMENTE IM DETAIL:



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

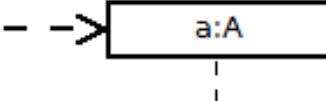


# DIE ELEMENTE IM DETAIL:



# GENERELL MÖGLICHE NACHRICHTENTYPEN:



1. Erzeugungsnachricht (Create Message): 

2. Synchrone Nachricht (blockiert): 

- Sender wartet, bis Empfänger die Nachricht abgearbeitet hat
- Gestrichelter Pfeil für Rücksprung zum Sender nötig!

3. Asynchrone Nachricht (blockiert nicht): 

- Sender wartet nicht auf Empfänger und arbeitet unmittelbar weiter
  - Sender und Empfänger befinden sich in unterschiedlichen Ausführungsprozessen
- Kein gestrichelter Rückgabepfeil!!

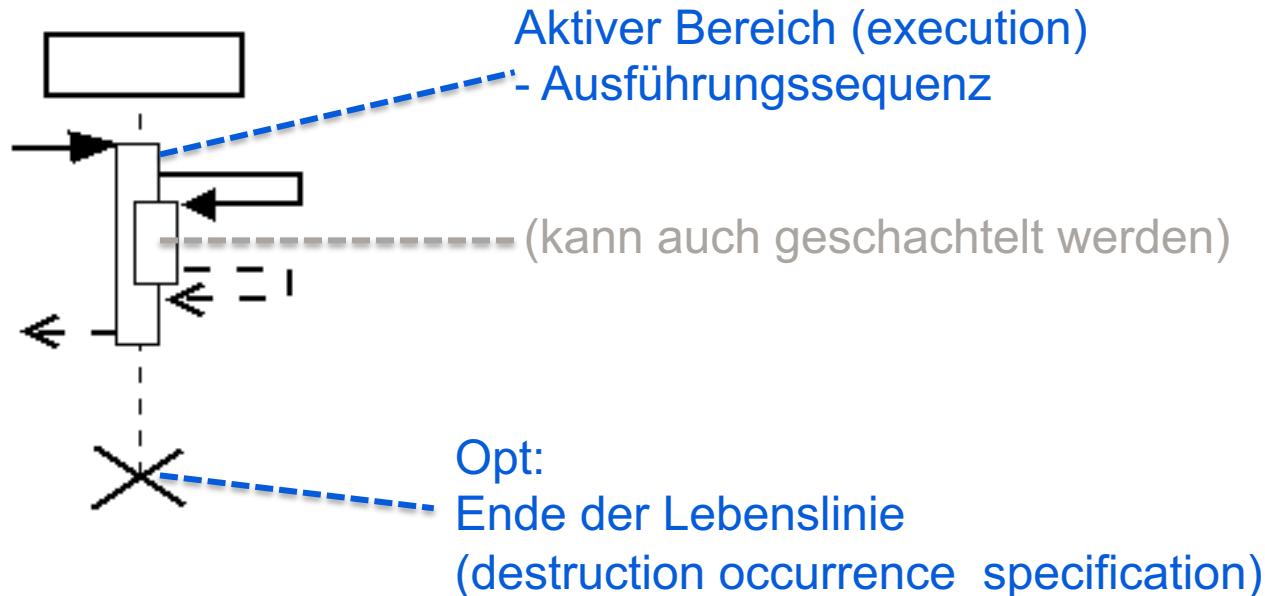
**BEM:** Ausnahmen (Exceptions) werden auch “nur” als normale Nachrichten behandelt.

# DIE ELEMENTE IM DETAIL:



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## Die Lebenslinie im Detail:

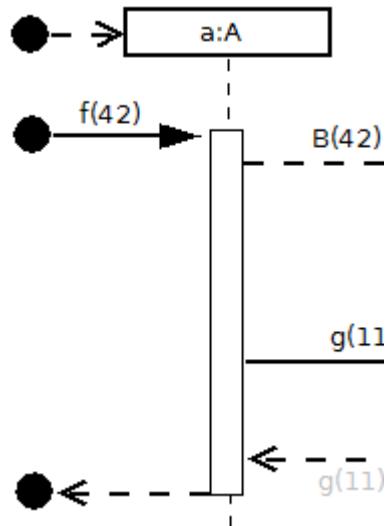


# LOST-FOUND MESSAGES VS SOG. GATES



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

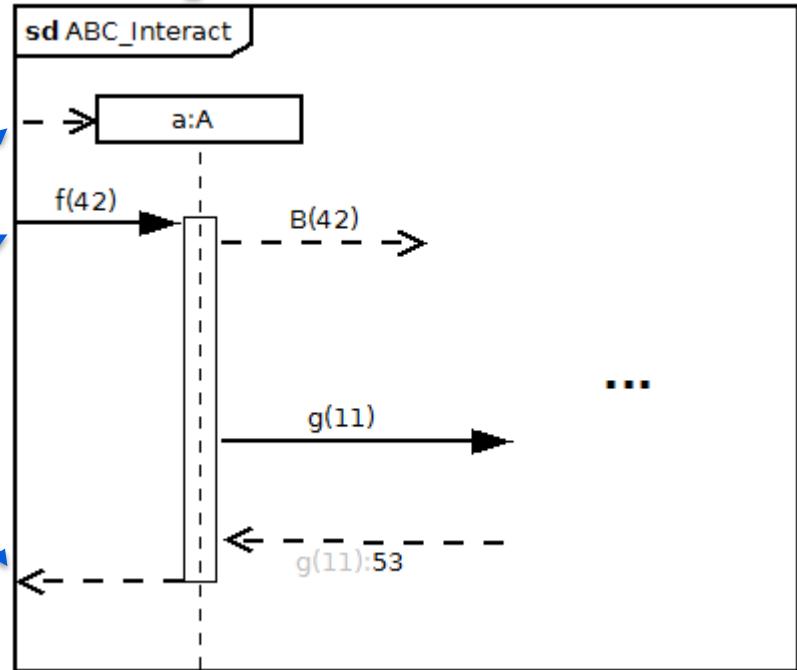
## Lost-Found-Messages:



## Gates:

Zeigt Name des SDs an

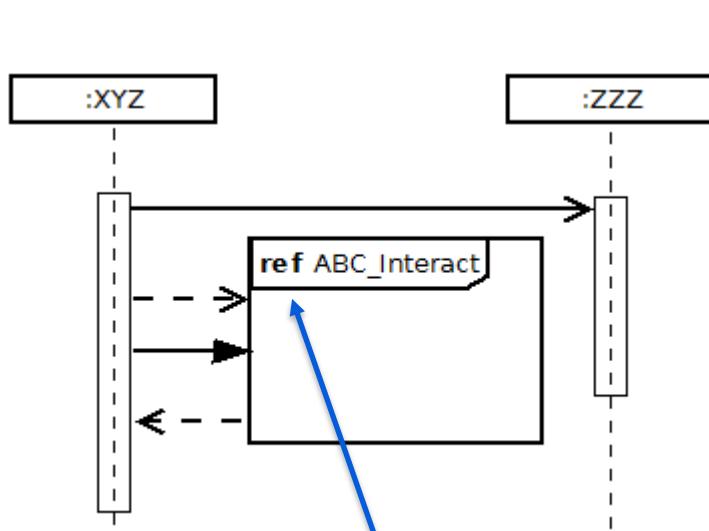
Gates



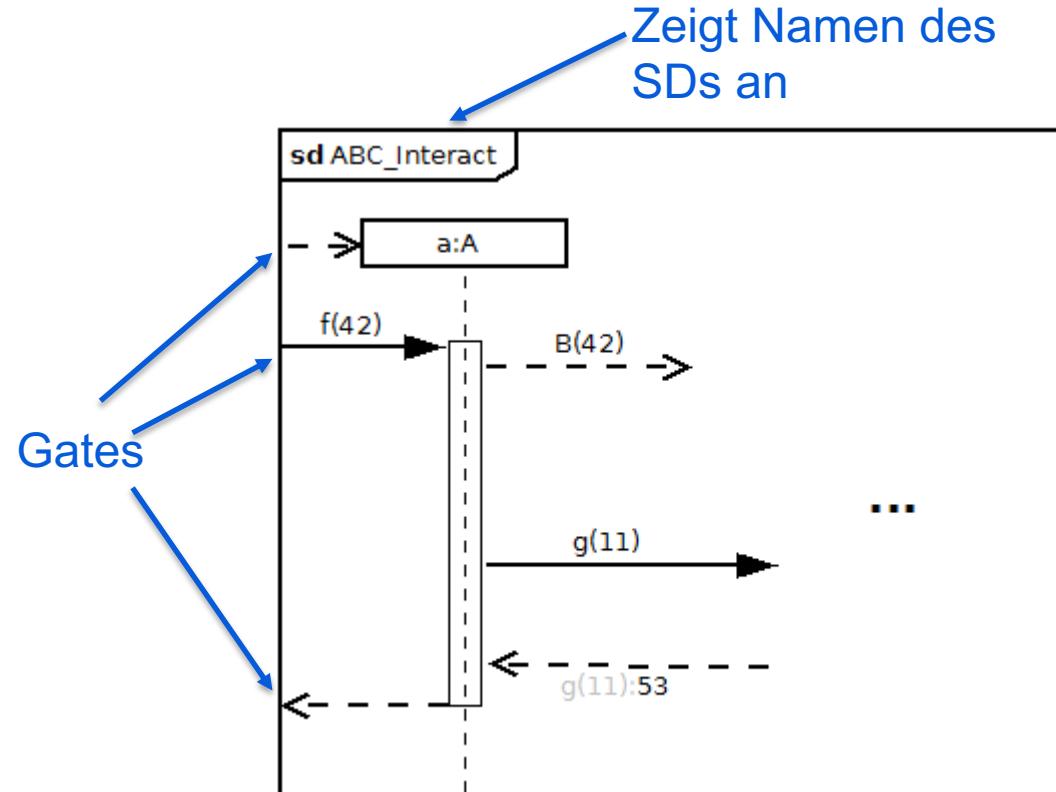
# BEM LOST-FOUND MESSAGES UND SOG. GATES



→ Gates sind Anknüpfungspunkte, die bei Referenzen aufgenommen werden:



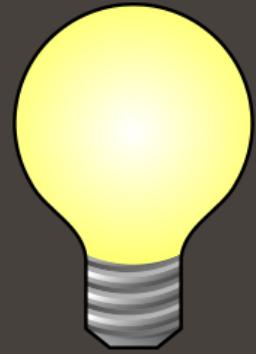
„ref“ zeigt an,  
dass anderes  
Diagramm  
referenziert wird





# 03

## Sequenzdiagramme – 2 verschiedene Semantiken



Ziel:  
Elemente im Überblick erfassen

# ZWEI VERSCHIEDENE SEMANTIKEN



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Die wichtigsten Elemente des Sequenzdiagramms kennengelernt
  - Syntax
- Es gibt aber auch zwei Arten der Bedeutung (Semantik):
  - "Aufrufsemantik"
    - Orientiert sich daran wie in Programmiersprachen Methoden aufgerufen werden
    - Sehr nah an Programmiersprachen
  - "Signalsemantik"
    - Sagt nur allgemein aus, dass “Signale ausgetauscht” werden
    - Viel allgemeinere/unbestimmte Aussage

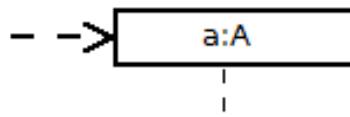
# DIE "AUFRUF-SEMANTIK":



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## Nachrichtentypen wie Aufruf/Rücksprung von Methoden:

1. Erzeugungsnachricht (Create Message):



2. Synchrone Nachricht (blockiert):



3. Asynchrone Nachricht (blockiert nicht):



**BEM:** Exceptions werden auch “nur” als normale Nachrichten behandelt.

→ Alle Elemente möglich und Bedeutung  
wie vorher erklärt

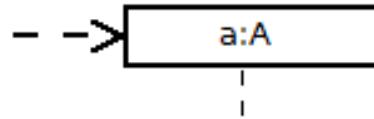
# DIE "SIGNAL-SEMANTIK"



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## Übermittlung von Signalen:

- Es wird nur abstrakt von Signalen gesprochen
- Mögliche Nachrichten:
  1. Erzeugungsnachricht (Create Message):



- 2. Signale:



→ Signale sind als asynchron definiert

- Deshalb wieder der Asynchronpfeil  
(Finde ich persönlich etwas ungeschickt, weil für ein Signal vielleicht gar nicht klar ist, ob es asynch oder synch ist)

# ZWEI VERSCHIEDENE SEMANTIKEN



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

**!** Diese Unterscheid. der Semantiken gibt es nicht offiziell in UML

→ Ableitung von mir aus [UMLglasklar; S. 430]:

“Im UML-Kommunikationsmodell bilden Nachrichten zwei Formen des Informationsaustausch ab:”

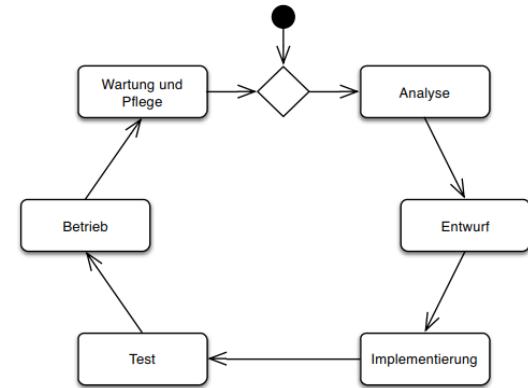
1. “Aufruf / Rücksprung einer Operation” (≜ Aufrufsemantik)
2. “Übermittlung von Signalen” (≜ Signalsemantik)

“Der Aufruf einer Operation ... Sie können sowohl synchrone als auch asynchrone Aufrufe modellieren. Hingegen ist die Übermittlung eines Signals immer asynchron ...”

- Warum spreche ich jetzt von Aufruf- und Signalsemantik?
  - Für Anfänger griffiger, wenn man es unterscheiden kann
  - Wichtig, weil Sequenzdiagramme in verschiedenen Phasen etwas verschieden verwendet werden.
  - Siehe folgende Folie

# WARUM BRAUCHEN WIR ZWEI SEMANTIKEN?

- Die “Aufrufsemantik” ist sehr detailliert
  - Sehr nah an Programmiercode & Programmabläufen
    - Viele Details wie z.B.: wer ruf wen wie genau auf, Rücksprünge müssen definiert sein, ...
  - Schon sehr genaue Festlegungen → nur in Design & Implem.!
- Bei der “Signalsemantik” sind die Signale eher unbestimmt
  - Eignet sich besonders, wenn man noch nicht, weiß ob es wirklich ein Aufruf oder etwas anderes ist
  - Eignet sich also gut in den frühen Phasen (Analyse oder früher Entwurf), wenn man noch nicht festlegen möchte, ob es ein synch. oder asynch. Aufruf (oder was auch immer...) ist
  - siehe dazu auch Dokument mit Bem. zur Robustness Analysis  
(→ siehe später auf Homepage zu PiG III – Grobentwurf)



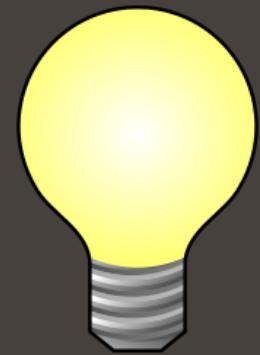


04

## Kommunikationsdiagramme – Modellelemente im Überblick

Ziel:

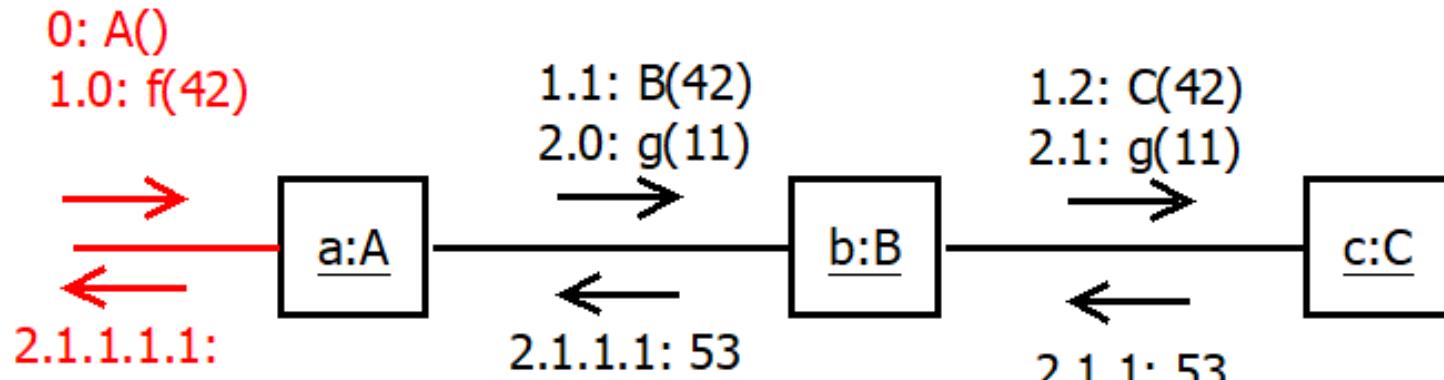
Die Elemente im Überblick erfassen



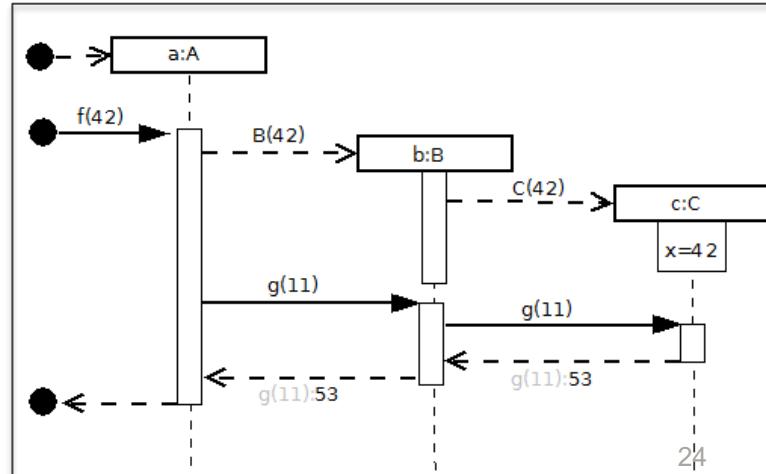
# VORHERIGES BEISPIEL ALS KOMMUNIKATIONSDIAGRAMM:



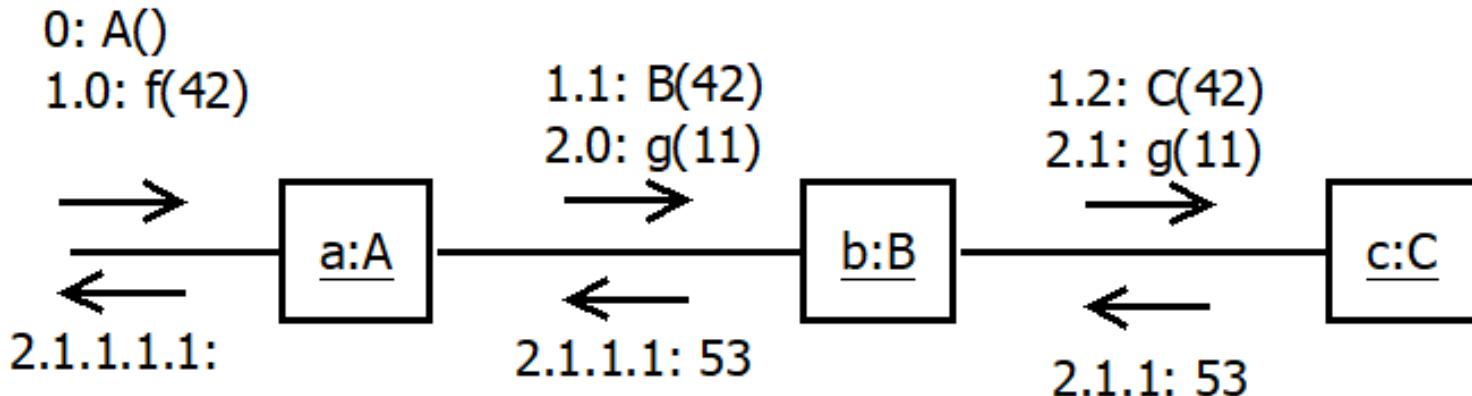
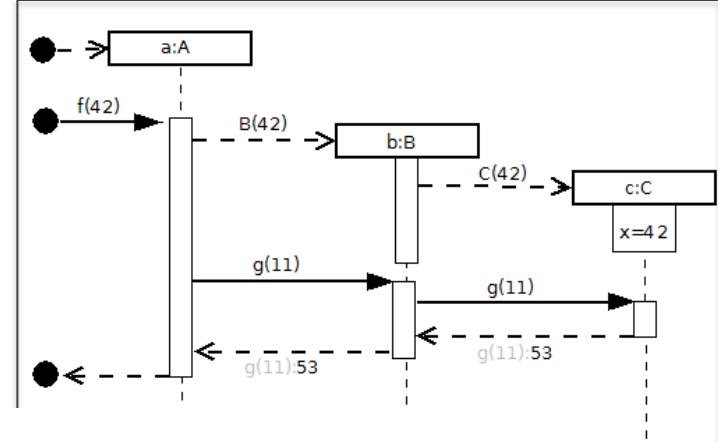
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim



BEM zu rot markiertem Bereich:  
Vorschlag von Bodo Igler für  
lost/found Messages  
(gibt es nicht in UML2.x bzw. ist dort  
nicht geklärt)



# KOMMUNIKATIONSDIAGRAMM WEITERE BEMERKUNGEN



- BEM: Reihenfolge ist nur anhand der Nummerierung ersichtlich → keine Vorgaben wie nummeriert wird!
  - BEM zu Pfeilen:
    - UML 1.x: Gleiche Pfeile wie in Sequenz-Diagramm
    - UML 2.x: Nur  $\longrightarrow$  (== Semantikverlust! ☹)
- Für mich: Sie können beides verwenden!

# EIGENSCHAFTEN KOMMUNIKATIONSDIAGRAMME



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Im Prinzip: Kommunikationsdiagramm hat den gleichen Informationsgehalt wie Sequenzdiagramm
- **ABER:** Nicht alles kann dargestellt werden:
  - Fokus auf (Nachrichten-)Verbindungen zwischen den Kommunikationspartnern
  - Information über Erzeugung und Zerstörung geht verloren
  - Zusammenhänge zw. Nachrichten sind schwerer lesbar
- **Stärke des Kommunikationsdiagramms:**
  - Zwitter aus Struktur- und Interaktionsdiagramm
    - Objekte werden oft in selber Anordnung wie in Klassendiagramm angeordnet (== Struktursicht)
    - Nachrichten zeigen dann die Interaktionen
      - Z.B. gut bei Robustness Analysis (→ PiG – Grobentwurf)!

# ENTSCHEIDUNGSHILFE



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

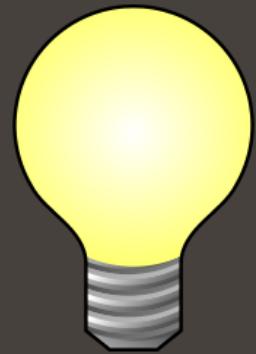
- Wann am besten welches Diagramm einsetzen?
- Zeitliche Abfolge wichtig → eher Sequenzdiagramm
  - Auch Aktivitäten innerhalb der Objekte teilweise darstellbar
- Eher Struktur wichtig → eher Kommunikationsdiagramm
- BEM: Sequenzdiagramm wird wesentlich häufiger verwendet
  - Finde ich auch meistens besser geeignet



# 05

## Weitere Interaktionsdiagramme

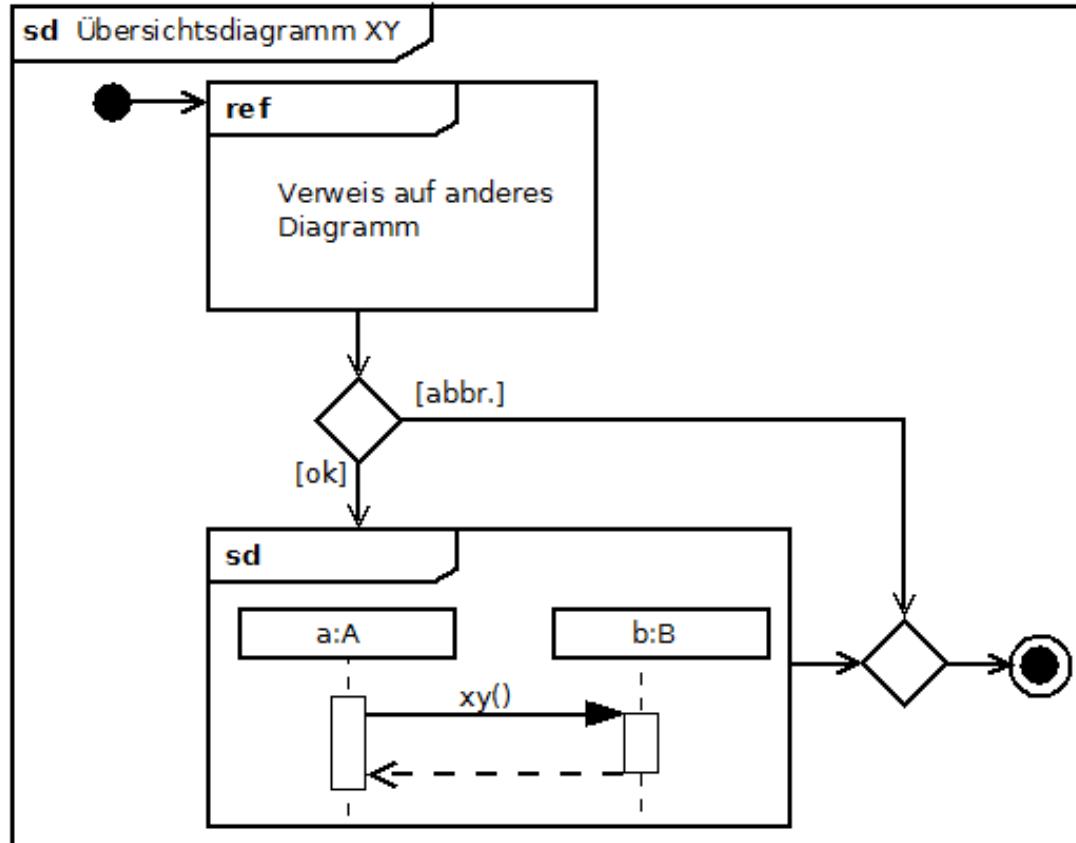
Ziel:  
Kurze Vorstellung der anderen Interaktionsdiagramme



# INTERAKTIONSÜBERSICHT-DIAGRAMM



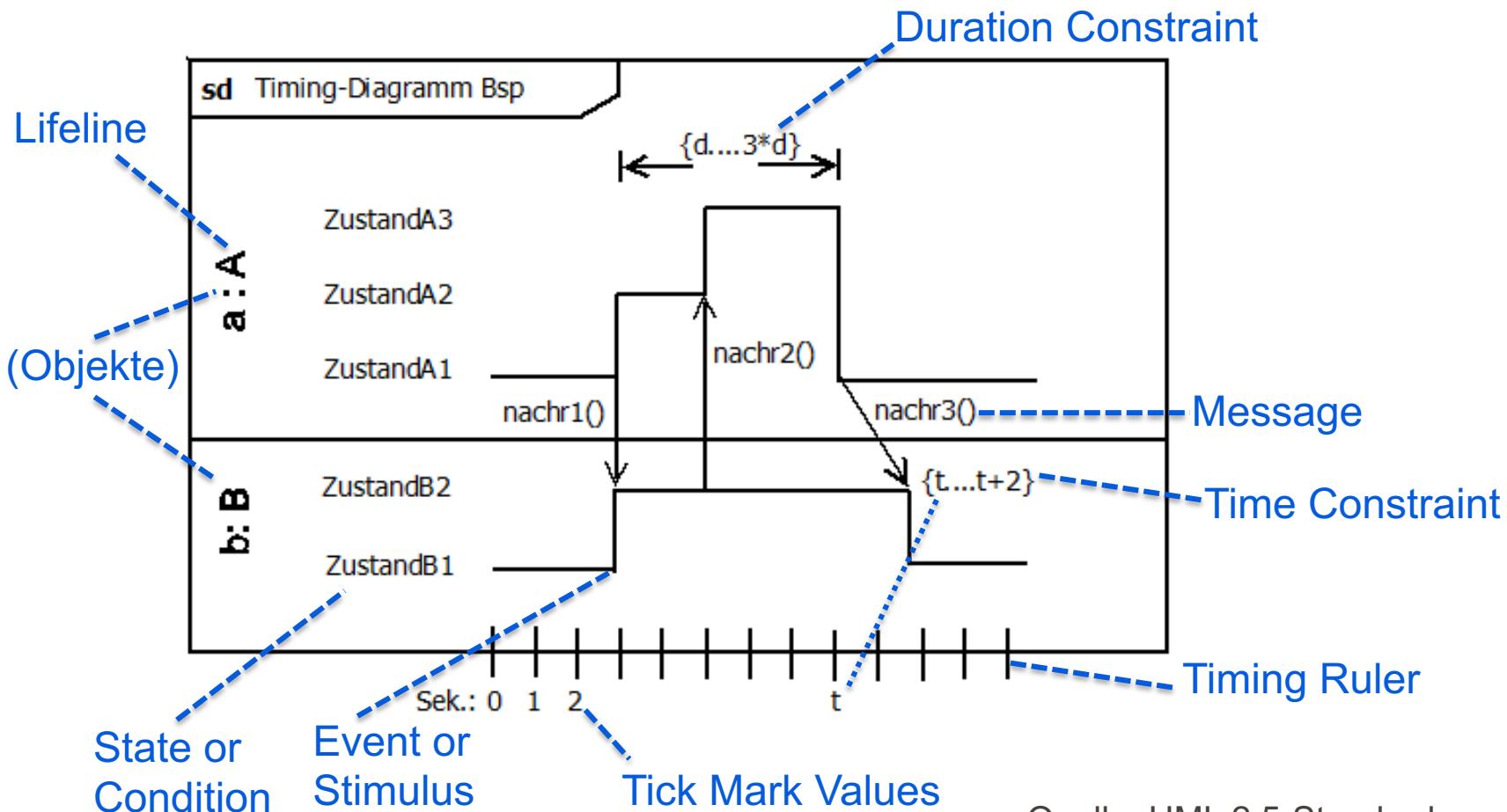
Metadiagramm, das die Zusammenhänge zw. verschied. Interaktionsdiagrammen zeigt:



# TIMING-DIAGRAMM (ZEITVERLAUFSDIAGRAMM)



Zeigt Nachrichtenaustausch und Zustandswechsel verschiedener Objekte zu bestimmten Zeitpunkten an:



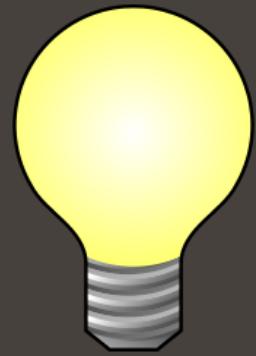


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# 06

# Fazit

Ziel:  
Was haben wir damit gewonnen?



# WAS HABEN WIR GELERNT?



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Interaktionsdiagramme
  - Sequenzdiagramm (das wichtigste)
  - Kommunikationsdiagramm
  - (Timing-Diagramm)
  - (Interaktionsübersichtsdiagramm)
- **Sequenzdiagramm**
  - Zeigt Objekte und deren Nachrichtenaustausch im Zeitverlauf
- Kommunikationsdiagramm
  - Zeigt Objekte und deren Nachrichtenaustausch in einer Struktursicht  
(vgl. Objektdiagramm)



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

AUF GEHT'S!!

SELBER MACHEN UND LERNEN!!



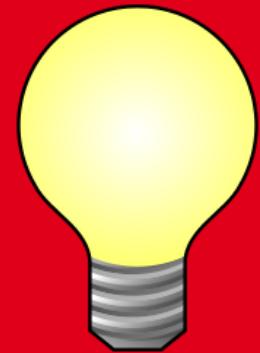


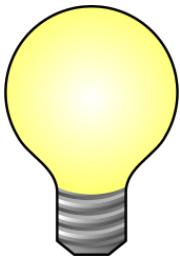
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

07.01.2021

# Programmieren im Großen I

Einführung in das Programmieren im Großen





Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# AGENDA

Einführung ins Thema

Der Softwareentwicklungsprozess

Das OO-Vorgehensmodell

Fazit

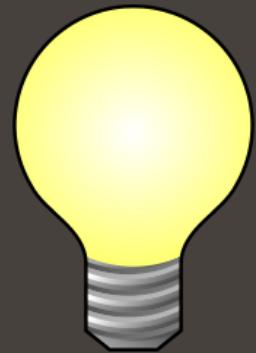


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# 01

## EINFÜHRUNG INS THEMA

Ziel:  
Die Eckpunkte des Themas kennenlernen



# WORUM GEHT'S?



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Programmieren im Kleinen:
    - kleines Programm ( $\approx$  1 - 20 Klassen)
    - 1 oder 2 Entwickler
    - Schlankes Vorgehen möglich:
      - (Entwurf, Spezifikation,) Implementierung und Test
    - ordentlich (fehlerfrei, wartbar, . . . )
- Kann man auch ohne Prozess oder mit unstrukturiertem Prozess noch einigermaßen gut hinbekommen



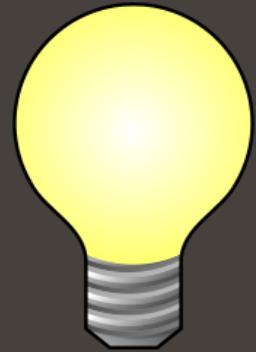
# WORUM GEHT'S?

- Programmieren im Großen:
  - Software-System, z.B.:
    - großes Programm (100 Klassen oder mehr)
    - mehrere Programme als Gesamtsystem
  - viele Entwickler
  - echter Kunde(n)
    - Viele Beteiligte (Stakeholder)
  - vollständiger Softwareentwicklungsprozess
  - ordentlich (fehlerfrei, wartbar, . . . )
- Man braucht einen gescheiten Softwareentwicklungsprozess
  - Muss Menschen und Artefakte koordinieren
  - Man braucht Projektmanagement, Anforderungen, Design, Implementierung, Testen, ...



02

## Der Softwareentwicklungsprozess (Wiederholung)



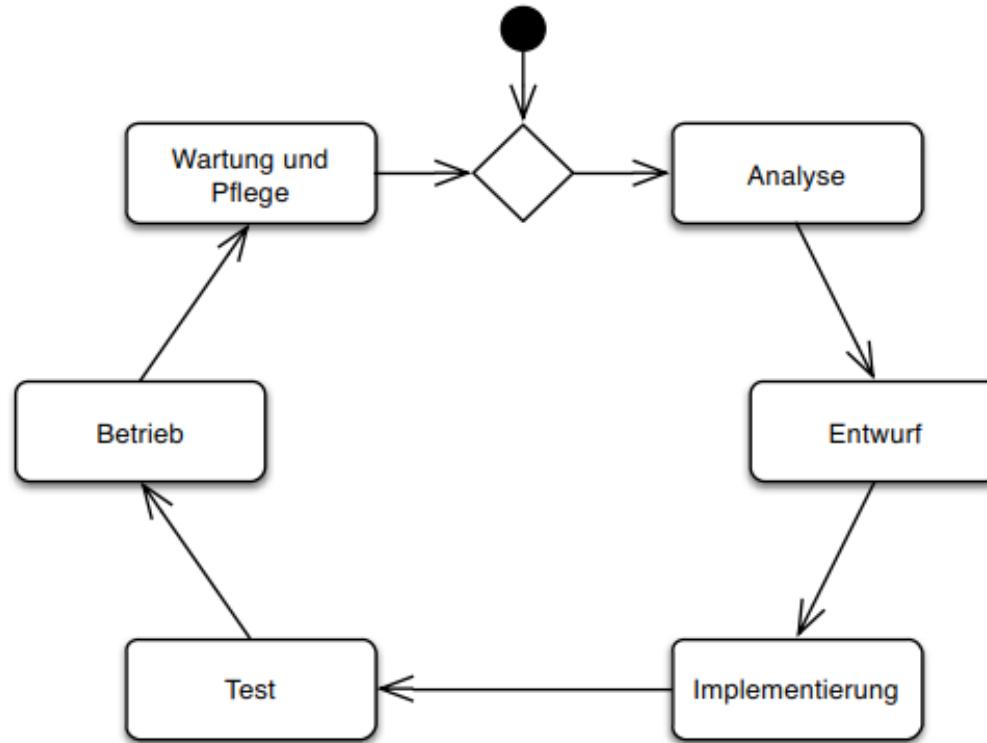
Ziel:

Nochmals den Softwareentwicklungsprozess genauer  
kennenlernen

# LEBENSZYKLUS VON SOFTWARE



Die typischen Tätigkeiten bei der SW-Entwicklung:



Vorsicht: Ist eine Idealisierung!

→ In der Praxis kann auch mal von Implementierung wieder zur Analyse zurückgesprungen werden, ...

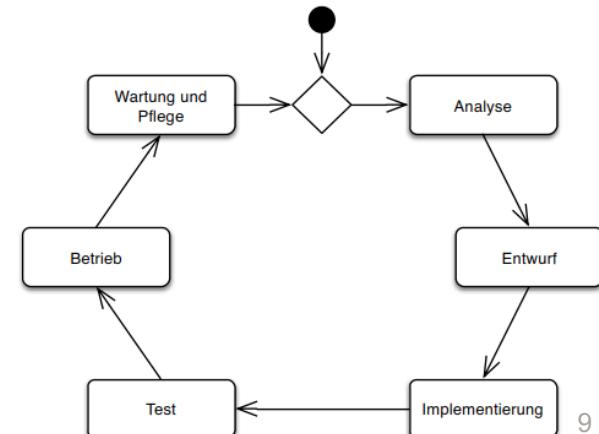
# LEBENSZYKLUS VON SOFTWARE (SOFTWARE-LIFE-CYCLE)



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

Typische Tätigkeiten bei der Software-Entwicklung:

1. Analyse: Was will der Kunde? (= Anforderungen)
2. Entwurf: Wie soll das zu bauende System sein?
  1. grob: Grobentwurf (Architektur/Architecture)
  2. detailliert: Feinentwurf (Detailed Design)
3. Implementierung: Entwurf → Programm
4. Test: Erfüllt das Programm die Anforderungen und den Entwurf?
5. Betrieb: Verwendung des Programms
6. Wartung und Pflege
  - Änderungswünsche/Fehler
    - Was will der Kunde?
    - ...



# MEHR VORGABEN SIND NÖTIG



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Typische Fragen bei der Software-Entwicklung:
  - Wie fangen wir an?
  - Was sollen wir tun?
  - Wie verteilen wir die Aufgaben?
  - Wie machen wir's richtig?
  - . . .

→ Hier sind mehr Vorgaben nötig



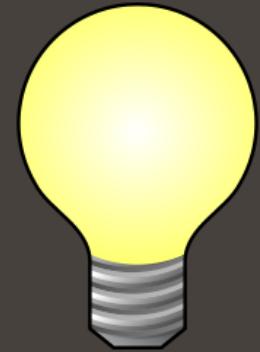
03

## Das OO-Vorgehensmodell

Ziel:

Vorgehensmodelle kennenlernen (Wiederholung)

Unser OO-Vorgehensmodell kennenlernen



# MEHR VORGABEN SIND NÖTIG



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## → Vorgehensmodelle

- = Bestimmte Vorgaben für die Durchführung von Softwareentwicklungsprojekten
- Typische Vorgaben:
  - Abfolge von Phasen/Tätigkeiten
  - Artefakte = Resultate von Phasen/Tätigkeiten, z.B.
    - Beschreibung der Anforderungen in bestimmter Form
    - Testfallbeschreibungen in bestimmter Form
    - Quellcode-Dateien gemäß Codier-Richtlinien
  - Zusammenhänge zwischen den Phasen/Tätigkeiten
  - Andere organisatorische Aspekte

# VORGEHENSMODELLE – BEISPIELE



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Frühe Modelle
  - Wasserfall
  - V-Modell
    - (Deutsche Erfindung – oft benutzt, z.B. Behörden, Automotive)
- Modelle der 2. Generation
  - Spiralmodell (von Barry Boehm)
  - V-Modell mit mehreren Zyklen
- Objektorientierte Modelle (3. Generation)
  - Rational Unified Process (RUP)
- Agile Methoden (4. Generation)
  - eXtreme Programming
  - SCRUM

(Siehe auch Vorlesung 01 und Vorlesung 12 später)

# UNSER OO-VORGEHENSMODELL



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Wir benötigen für das Praktikum, spätere Projekte, . . .
  - einen Rahmen für OOAD (= Objektorient. Analyse & Design)
  - solide
  - erprobt
  - abgespeckte Variante des (R)UP (= Rationale Unified Process)
- RUP insgesamt → für unsere Zwecke viel zu aufwändig
  - Betrachten auszugsweise die für uns wichtigsten RUP-Bestandteile
    - gute Orientierung für die OO-Softwareentwicklung
- Für manche Erklärung verwende ich jedoch auch V-Modell
  - Kann man manches besser erklären
  - Was ich erkläre, ist kompatibel zu RUP (anderes nicht!)

# UNSER OO-VORGEHENSMODELL – AUSGEWÄHLTE TÄTIGKEITEN



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Wir betrachten folgende Tätigkeiten („disciplines“):
  - Anforderungsanalyse („Requirements“)
    - Was will der Kunde?
  - Analyse und Entwurf („Analysis and Design“)
    - Wie soll das zu bauende System sein?
  - Implementierung („Implementation“)
    - Das System bauen
  - Test:
    - Wie stelle ich sicher, dass das System das tut, was es tun soll?
- Wir gehen nicht ein auf:
  - Geschäftsprozessmodellierung („Business Modeling“)
  - Inbetriebnahme („Deployment“)

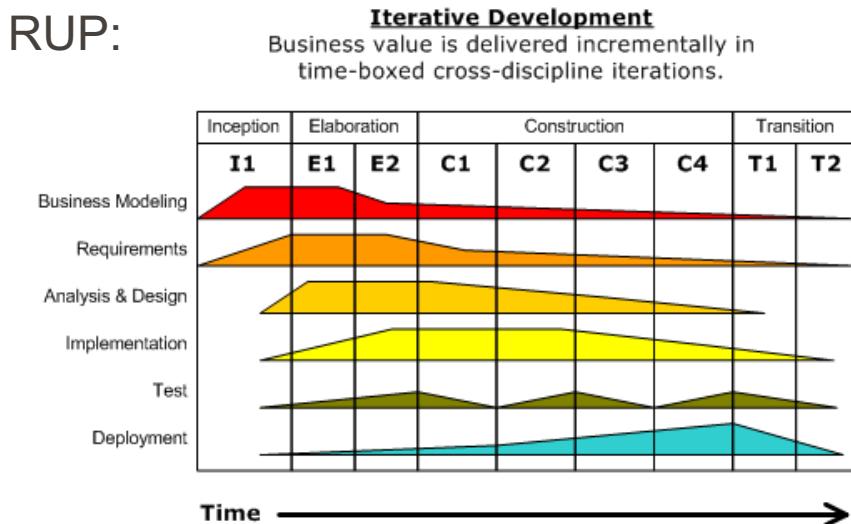
# UNSER OO-VORGEHENSMODELL – AUSGEWÄHLTE TÄTIGKEITEN



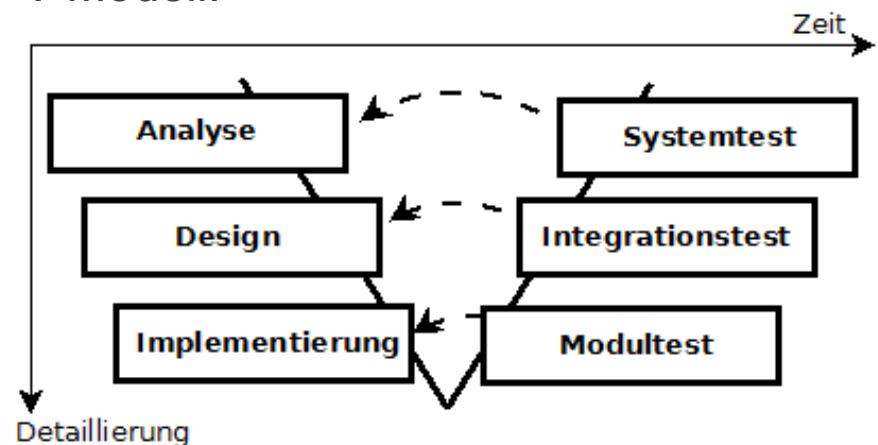
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Wir betrachten folgende Tätigkeiten („disciplines“):
  - Anforderungs-Analyse („Requirements“)
  - Analyse und Entwurf („Analysis and Design“)
  - Implementierung („Implementation“)
  - Test

RUP:



V-Modell:



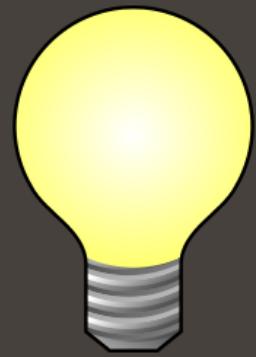
→ Ich verwende die V-Modellzeichnung (eingängiger für Sie), aber ich erkläre die Tätigkeiten nach RUP-Stil (etwas besser an OO angelehnt)



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## 04 Fazit

Ziel:  
Was haben wir damit gewonnen?





# WAS HABEN WIR GELERNT?

- Wofür steht Programmieren im Großen?
  - Entwicklung großer SW-Systeme
    - Mehrere Programme, viele Klassen, ...
  - Viele Beteiligte
- Zur Koordin. braucht man einen SW-Entwicklungsprozess
  - Es gibt hier viele verschiedene
    - Sog. Vorgehensmodelle
- Unser OO-Vorgehensmodell:
  - Abgespecktes RUP-Vorgehensmodell
- Zur leichteren Illustration verwende ich aber V-Modell



# WEITERFÜHRENDE LITERATUR

- Kleuker: Grundkurs Software-Engineering mit UML [<http://dx.doi.org/10.1007/978-3-8348-9843-2>].
- Zuser et al: Software-Engineering mit UML und dem Unified Process [BF 500 92].
- Larman, C.: Applying UML and Patterns [30 BF 500 78].



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

AUF GEHT'S!!

SELBER MACHEN UND LERNEN!!



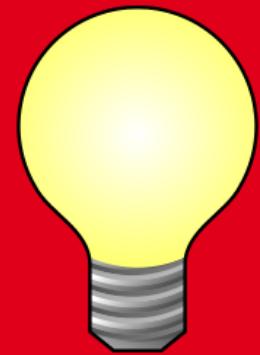


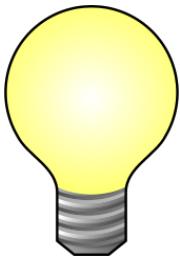
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

07.01.2021

# Programmieren im Großen II

Anforderungsanalyse





## AGENDA

Einführung ins Thema

Das Fachmodell

Anwendungsfälle (Use Cases)

Nichtfunktionale Anforderungen

GUI

Qualitatssicherung

Das war's noch lange nicht

Fazit

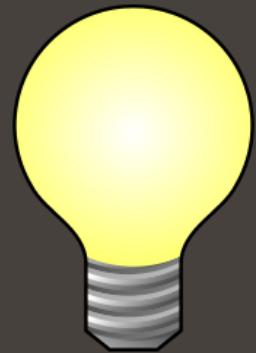


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# 01

## EINFÜHRUNG INS THEMA

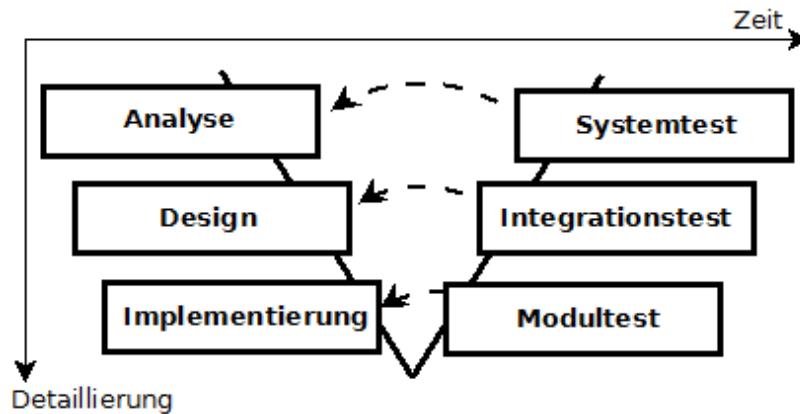
Ziel:  
Die Eckpunkte des Themas kennenlernen





# WORUM GEHT'S?

- Letztes Mal:
  - Wir brauchen ein Vorgehensmodell
  - Verschiedene Phasen:

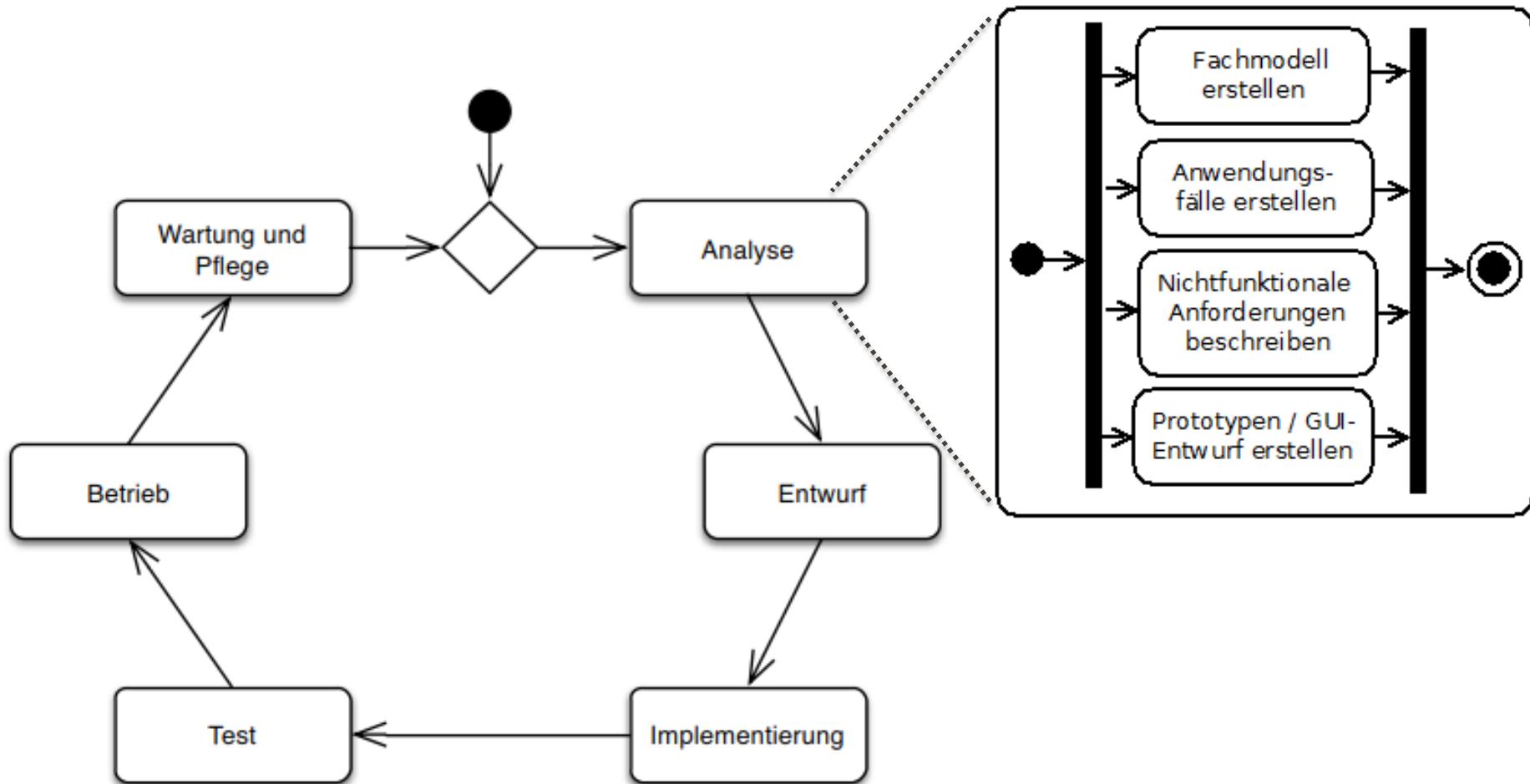


- Heute:
  - Anforderungsanalyse

# ANFORDERUNGSANALYSE – UNTERAKTIVITÄTEN:



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim



# ANFORDERUNGSANALYSE – EINFÜHRUNG



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Ziel:
  - Wir wissen genau, was der Kunde will
- Input: Kundengespräche, vorhandene Dokumente, . . .
- Output:
  - statische Aspekte: „Analyse-Klassen-Diagramm“ = „Fachmodell“  
= im wesentlichen: UML-Klassen-Diagramm(e)  
  („Datenmodell“, „Domänenmodell“ („domain model“))
  - dynamische Aspekte: Anwendungsfälle („Use Cases“)  
= im wesentlichen textuelle Beschreibungen
  - Nichtfunktionale Anforderungen (z.B. Performance, Sicherheit, . . .)
  - ggfs. Prototypen, GUI-Entwurf, . . .

# DURCHGÄNGIGES BEISPIEL – MP3-PLAYER



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Für die Erklärung der Phasen nutze ich möglichst ein durchgängiges Beispiel
  - Angenommen wir sollen einen MP3-Player für den PC (wie z.B. WinAmp) entwickeln
- BEM: Ist vielleicht etwas veraltet  
(nicht so sexy, wie eine Handy-App)
  - ABER:
    - Sehr einfaches Beispiel
    - Sehr eingängig (jeder von Ihnen kennt das)
  - MP3-Player auf dem Handy funktionieren ähnlich

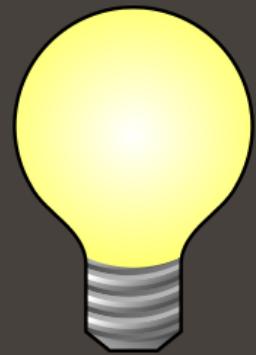


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## 02

# Das Fachmodell

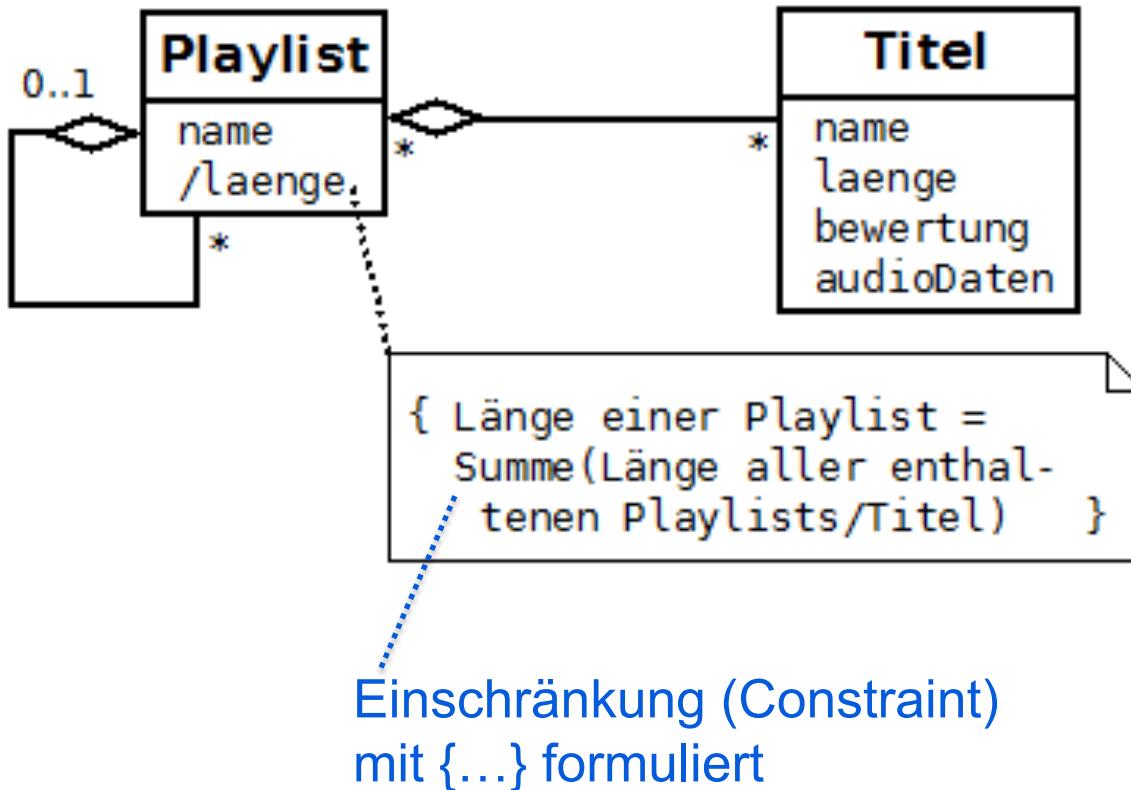
Ziel:  
Das Fachmodell kennenlernen



# FACHMODELL – BEISPIEL



- Beispiel: (unvollständiges) Fachmodell für die Playlistenverwaltung eines Software-MP3-Players:



# FACHMODELL – DEFINITION & FORM



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Definition Fachmodell:
  - Beschreibung der statischen Aspekte (Daten) des Anwendungsgebiets in der Sprache des Anwendungsgebiets  
(= Sprache des Kunden)
- Form:
  - UML-Klassendiagramm(e)
    - + Kommentare
    - + Einschränkungen (Constraints)  
(in natürlicher oder formaler Sprache (z.B. OCL = Object Constraint Language))
    - dynamische Aspekte (d.h. keine Methoden und Interfaces)
    - Auch Attribut-Typen und andere Details dürfen weggelassen werden
  - BEM: Auch ER-Modelle eignen sich natürlich vorzüglich

# FACHMODELL – WAS IST ES NICHT?



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim



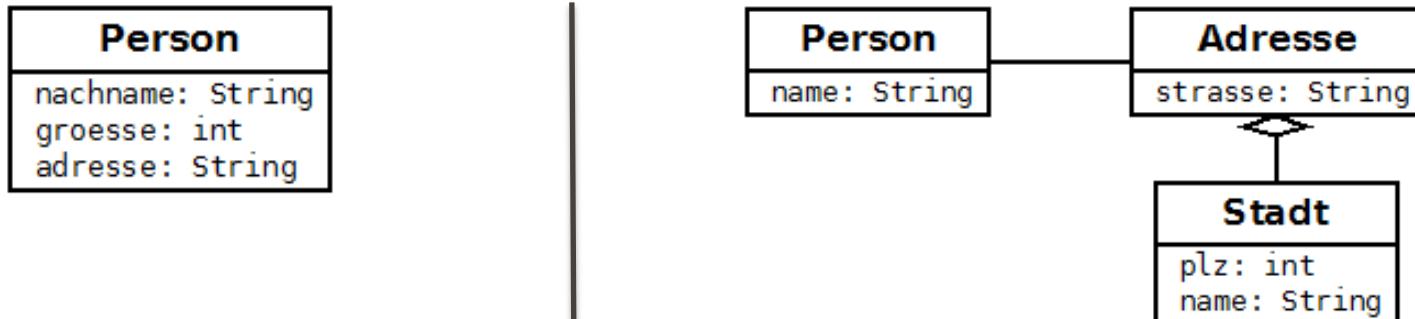
## Vorsicht

- i.d.R.: Klassen-Diagramm für Fachmodell  $\neq$  Klassen-Diagramm für Java-Klassen
- gleiche Notation für völlig verschiedene Anwendungsbereiche!
- Fachmodell bildet nur die fachlichen Zusammenhänge ab
- Im Design kommen noch viele andere Aspekte hinzu
  - Fachmodell bildet aber meist die Ausgangsbasis

Fachmodell:



“Fachmodell” im Design:



→ In Zukunft Domänenmodell  
genannt!

# FACHMODELL – WIE KOMMT MAN DAZU?



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Bewährte Vorgehensweisen
  - Bewährtes aus der Vorlesung „Datenbanken“
    - z.B.: Normalformen
  - Analysemuster (→ siehe Praktikum)
- Wie finde ich die Klassen, Attribute, Assoziationen, . . . für das Fachmodell?
  - Hilfsmittel I: Typische Situationen mit Hilfe von Objektdiagrammen veranschaulichen → OD zeichnen
  - Hilfsmittel II: Textanalyse nach R. J. Abbott: „Program Design by Informal English Descriptions“. Comm. of the ACM, 26, #11, 1983.
  - Hilfsmittel III: Methode von Bjarne Stroustrup
  - ... es gibt noch andere Hilfsmittel
    - TIP: Am besten miteinander kombinieren

# TEXTANALYSE NACH ABOTT\* - VORGEHENSWEISE:



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

1. Gut zuhören & relevante Daten sammeln
2. Zuordnung:
  - Hauptwörter (Substantive) → Liste von Objekten, Attributen
  - Zeitwörter (Verben) → Liste von Beziehungen (& Methoden)
3. Liste bereinigen:
  - Redundanzen streichen (z.B. Synonyme)
  - Irrelevante Begriffe streichen
  - Vage Begriffe streichen oder präzisieren
4. Klassen mit Attributen und Beziehungen (Assoziationen)  
ableiten → Klassendiagramm erstellen

\* R. J. Abbott: „Program Design by Informal English Descriptions“. Comm. of the ACM, 26, #11, 1983.

# B. STROUSTRUP\*\* – BEZIEH. ZW. KLASSEN FINDEN\*



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

<b>Ist</b> a ein b?	Ist b ein Spezialfall von a? Ein Auto <b>ist</b> ein Fortbewegungsmittel.
<b>Hat</b> a ein b?	<b>Besitzt</b> a ein Objekt vom Typ b? Ein Auto <b>hat</b> einen Lenker. Ein Auto <b>ist aber kein</b> Lenker.
<b>Benutzt</b> a ein b?	<b>Benutzt</b> a ein Objekt vom Typ b? Ein Auto <b>benutzt</b> Straßen. Ein Auto <b>ist keine</b> Straße. Ein Auto <b>hat keine</b> Straße.

\* Siehe auch OOSE: 14\_OO5\_Wie\_zum\_Objektmodell

\*\* B. Stroustrup: The C++ Programming Language, 1998.

# B. STROUSTRUP\*\* – BEZIEH. ZW. KLASSEN FINDEN\*



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

Deutet auf Vererbung hin



*Ist* a ein b?

Ist b ein Spezialfall von a?  
Ein Auto *ist* ein Fortbewegungsmittel.

Deutet auf Attribut hin



*Hat* a ein b?

*Besitzt* a ein Objekt vom Typ b?  
Ein Auto *hat* einen Lenker.  
Ein Auto *ist aber kein* Lenker.

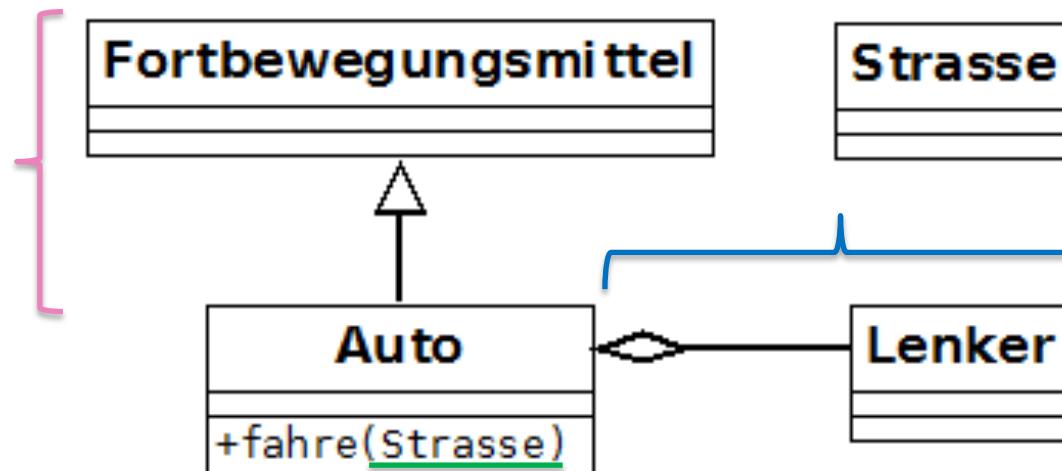
Deutet auf anderes hin

z.B.: Parameter in Methode,  
Attribut, das benutzt wird,  
Assoziation, ...



*Benutzt* a ein b?

*Benutzt* a ein Objekt vom Typ b?  
Ein Auto *benutzt* Straßen.  
Ein Auto *ist keine* Straße.  
Ein Auto *hat keine* Straße.



\* Siehe auch OOSE: 14\_OO5\_Wie\_zum\_Objektmodell

\*\* B. Stroustrup: The C++ Programming Language, 1998.

# TIP: ANSÄTZE AM BESTEN KOMBINIEREN



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

Mögliche Kombinierung der Ansätze:

1. Textanalyse nach Abott
    - Mögliche Klassenkandidaten, Attribute, Aktionen, ... finden
  2. Beziehungen nach B. Stroustrup analysieren
    - Beziehungen zwischen Klassenkandidaten, ... aufdecken
  3. Im Zweifelsfalls / wenn es nicht klar ist?  
→ Objektdiagramme verschiedener Situationen zeichnen
- + Evtl. andere Analysemuster  
(siehe spätere Vorlesung zu Mustern & Praktikum)



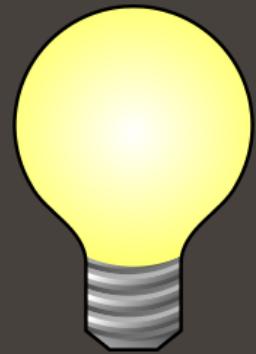
03

## Anwendungsfälle (Use Cases)

Ziel:

Anwendungsfälle spezifizieren lernen

→ Spezifikation der Funktionalen Anforderungen



# ANWENDUNGSFÄLLE (USE CASES)



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Definition Anwendungsfälle (Use Cases):
  - Beschreibung der dynamischen Aspekte (**Datenverarbeitung**) des Anwendungsgebiets in der Sprache des Anwendungsgebiets (= Sprache des Kunden)
- Definition Anwendungsfall (Use Case):
  - **textuelle** Beschreibung **eines** Vorgangs eines Systems

# ANWENDUNGSFÄLLE – ANWENDUNGSFALLSCHABLONE



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Wie geht man jetzt genau vor?
  - Unterstützung durch Schablone (Template)  $\triangleq$  Formular
  - Die einzelnen Punkte unterstützen dabei, um nichts wesentliches zu vergessen
- Wo gibt es Anwendungsfallschablonen?
  - z.B. auf <http://alistair.cockburn.us>
  - z.B. Seite der Sophist Group: <https://www.sophist.de>



## Vorsicht:

- Eine Schablone ist nur eine Schablone.
- Verstehen & an die eigenen Bedürfnisse anpassen!

# ANWENDUNGSFÄLLE – ANWENDUNGSFALLSCHABLONE



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

Vorschlag was in Schablone vorkommen sollte:

- Name + Eindeutig ID (macht Querverweise leichter)
  - Ziel
  - Beteiligte Akteure
  - Verwendete (andere) Anwendungsfälle
  - Auslöser, Vorbedingung, Nachbedingung
    - (für Erfolgs- und Misserfolgs-Fälle)
  - Standardablauf (Szenario, in dem alles gut und normal verläuft)
  - Alternative Ablaufschritte
    - Auflistung **aller** Ausnahmen/Fehlerfälle
    - Verhalten in Ausnahmesituation
  - Zeitverhalten, Verfügbarkeit
  - Fragen & Kommentare
- Querverweise
- Nichtfunktionale Anforderungen

# ANWENDUNGSFÄLLE – ANWENDUNGSFALLSCHABLONE



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

Am besten man verwendet eine Tabelle:

<b>ID: Name</b>	
<b>Ziel</b>	
<b>Akteure</b>	
<b>Status</b>	
<b>Verwendete UseCases</b>	
<b>Auslöser</b>	
<b>Vorbedingungen</b>	
<b>Nachbedingungen (Ergebnis)</b>	<b>Erfolg:</b>  <b>Misserfolg:</b>
<b>Standardablauf</b>	
<b>Alternative Ablaufschritte</b>	
<b>Zeitverhalten</b>	
<b>Verfügbarkeit</b>	
<b>Fragen</b>	
<b>Kommentare</b>	

# ANWENDUNGSFALLBESCHREIB. – BEISPIEL (MP3-PLAYER):



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

<b>ID: Name</b>	UC001: Titel hinzufügen
<b>Ziel</b>	Der Titel ist in der gewünschten Playliste enthalten.
<b>Akteure</b>	Anwender
<b>Status</b>	Entwurf
<b>Verwendete UseCases</b>	-
<b>Auslöser</b>	Kauf/Erhalt eines neuen Titels.
<b>Vorbedingungen</b>	Titel muss in Form einer Audio-Datei im Datei-System vorliegen. Das System ist geöffnet und zeigt das Hauptfenster
<b>Nachbedingungen (Ergebnis)</b>	<b>Erfolg:</b> Keine Veränderung an der Original-Audio-Datei, System enthält die Informationen über den Titel und die zugehörigen Audio-Daten.  <b>Misserfolg:</b> Keine Veränderung am System.

# ANWENDUNGSFALLBESCHREIB. – BEISPIEL (MP3-PLAYER):



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

<b>Standardablauf</b>	<ol style="list-style-type: none"><li>1: Anwender klickt auf die Funktion „Titel hinzufügen“.</li><li>2. Das System zeigt den Dialog „Titel hinzufügen“ (DIALOG_XY)</li><li>3: Anwender wählt eine Playlist aus.</li><li>4: ...</li><li>5: ...</li><li>6: ...</li><li>7: System importiert den Titel.</li></ol> <p>Querverweis über ID auf einen GUI-Entwurf (siehe später)</p>
<b>Alternative Ablaufschritte</b>	<ol style="list-style-type: none"><li>3a: Noch keine Playlist vorhanden</li><li>... 4a: Titel bereits vorhanden</li><li>4b: ... andere Ausnahme ... ... 7a: Import schlägt fehl ...</li></ol>
<b>Zeitverhalten</b>	Keine Einschränkung
<b>Verfügbarkeit</b>	Muss immer möglich sein
<b>Fragen</b>	
<b>Kommentare</b>	

→ 1. Iteration

# ANWENDUNGSFALLBESCHREIB. – BEISPIEL (MP3-PLAYER):



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

<b>Standardablauf</b>	1: Anwender klickt auf die Funktion „Titel hinzufügen“. 2. Das System zeigt den Dialog „Titel hinzufügen“ (DIALOG_XY) 3: Anwender wählt eine Playlist aus. 4: ... 5: ... 6: ... 7: System importiert den Titel.
<b>Alternative Ablaufschritte</b>	3a: Noch keine Playlist vorhanden ... 4a: Titel bereits vorhanden
<b>Alternative Ablaufschritte</b>	3a: Noch keine Playlist vorhanden → System erstellt automatisch eine Default-Playlist ... 4a: Titel bereits vorhanden → Korrektur durch Anwender oder Abbruch 4b: ... andere Ausnahme ... → ... ... 7a: Import schlägt fehl → ??? ...
<b>Zeitv</b>	
<b>Verf</b>	
<b>Frag</b>	
<b>Kom</b>	

→ 2. Iteration

# ANWENDUNGSFALLBESCHREIB. – BEISPIEL (MP3-PLAYER):



Standardablauf	1: Anwender klickt auf die Funktion „Titel hinzufügen“. 2. Das S 3: Anwe 4: ... 5: ... 6: ... 7: System importiert den Titel.
	<b>Offene Punkte:</b> <ul style="list-style-type: none"><li>- Name der Default-Playlist bei alternativem Ablaufschritt 3a?</li><li>- Systemverhalten bei alternativem Ablaufschritt 7a?</li></ul>
Alternative Ablaufschritte	3a: Noch keine Playlist vorhanden → System erstellt automatisch eine Default-Playlist ... 4a: Titel bereits vorhanden → Korrektur durch Anwender oder Abbruch 4b: ... andere Ausnahme ... → ... ... 7a: Import schlägt fehl → ??? ... → <b>Sofort notieren</b>
Ze Fr Ve	Fragen Name der Default-Playlist bei alternativem Ablaufschritt 3a? Systemverhalten bei alternativem Ablaufschritt 7a?
Fragen	
Kommentare	

→ Klären & nächste Iteration, ...

→ So viele Iterationen bis **möglichst** alles beschrieben

# ANWENDUNGSFALLBESCHR. – WAS NOCH BEACHTEN?

- Vor- und Nachbedingungen:
  - Beschreibung als Zustände
    - Vorbed.: Benutzer ist eingeloggt,  
System zeigt Hauptbildschirm
    - Nachbed.: System hat Auftrag geprüft und abgearbeitet
- Schritte:
  - Erster Schritt beginnt da wo die Vorbedingung aufgehört hat  
→ 1. Benutzer wählt ... im Hauptbildschirm
  - Oft immer ein Wechsel zw. Benutzer\* und System  
→ 1. Benutzer wählt ... im Hauptbildschirm
    2. System zeigt den Dialog ...
    3. Benutzer wählt ...
    4. System ...
- Immer nur EIN Anwendungsfall pro Schablone!

ID: Name	UC01: Titel hinzufügen
Ziel	Der Titel ist in der gewünschten Playliste enthalten.
Akteure	Anwender
Status	Entwurf
Verwendete UseCases	-
Auslöser	Kauf/Erhalt eines neuen Titels.
Vorbedingungen	Titel muss in Form einer Audio-Datei im Datei-System vorliegen. Das System ist geöffnet und zeigt das Hauptfenster
Nachbedingungen (Ergebnis)	Erfolg: Keine Veränderung an der Original-Audio-Datei, System enthält die Informationen über den Titel und die zugehörigen Audio-Daten.
Standardablauf	Misserfolg: Keine Veränderung am System. 1: Anwender klickt auf die Funktion „Titel hinzufügen“. 2: Das System zeigt den Dialog „Titel hinzufügen“ (DIALOG_XY) 3: Anwender wählt eine Playlist aus. 4: ... 5: ... 6: ... 7: System importiert den Titel.
Alternative Ablaufschritte	3a: Noch keine Playlist vorhanden → System erstellt automatisch eine Default-Playlist ... 4a: Titel bereits vorhanden → Korrektur durch Anwender oder Abbruch 4b: ... andere Ausnahme ... → ... ... 7a: Import schlägt fehl → ??? ...
Zeitverhalten	Keine Einschränkung
Verfügbarkeit	Muss immer möglich sein
Fragen	
Kommentare	

\*(oder anderem Akteur!)

# ANWENDUNGSFÄLLE – ANWENDUNGSFALLSCHABLONE



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## Mögliche Statusmodelle:

ID: Name	
Ziel	
Akteure	
Status	
Verwendete UseCases	
Auslöser	
Vorbedingungen	
Nachbedingungen (Ergebnis)	<b>Erfolg:</b>  <b>Misserfolg:</b>
Standardablauf	
Alternative Ablaufschritte	
Zeitverhalten	
Verfügbarkeit	
Fragen	
Kommentare	

```
graph LR; Start(( )) --> Entwurf[Entwurf]; Entwurf -- "Review mit Kunde" --> Akzeptiert[Akzeptiert von Kunde]; Akzeptiert -- "(*) Fehler entdeckt, Änderung gewünscht" --> Implementiert[Implementiert & Getestet]; Akzeptiert -- "Systemtest bestanden" --> Ausgeliefert[Ausgeliefert]; Implementiert -- "Abnahmetest bestanden" --> Ausgeliefert; Ausgeliefert --> Implementiert;
```

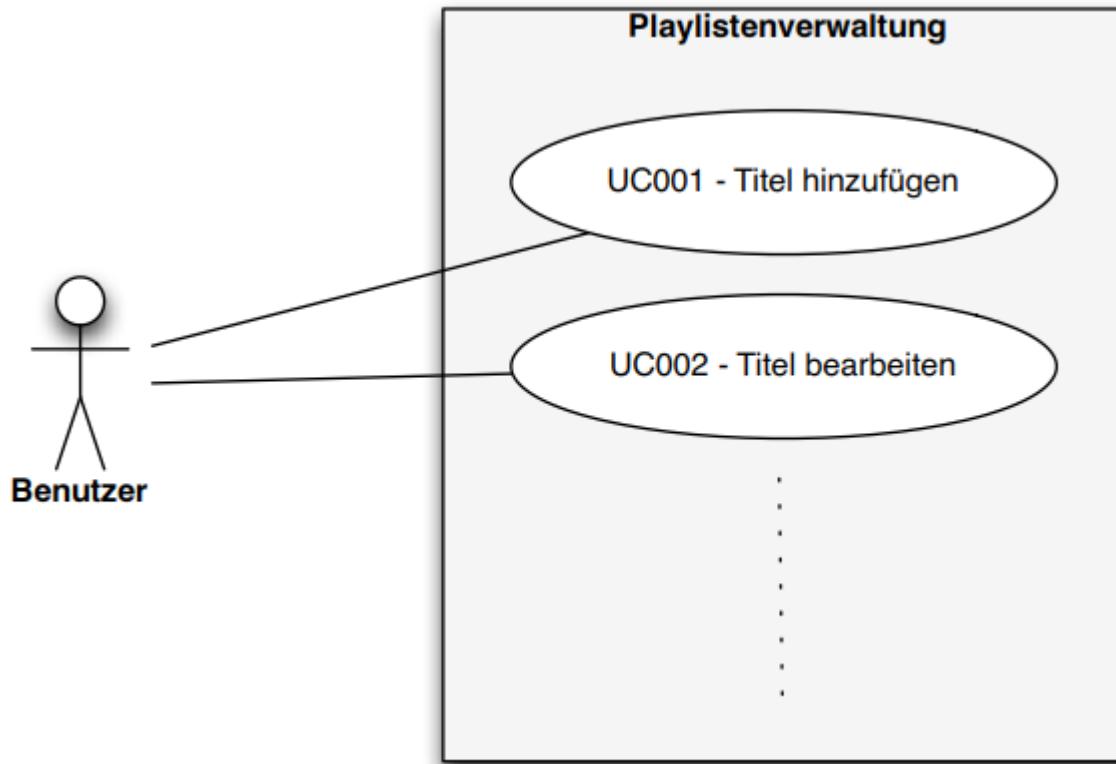
Mögliche Statusmodelle

# ANWENDUNGSFALLDIAGRAMM – BEISPIEL (MP3-PLAYER)



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

BSP: UseCase-Diagramm



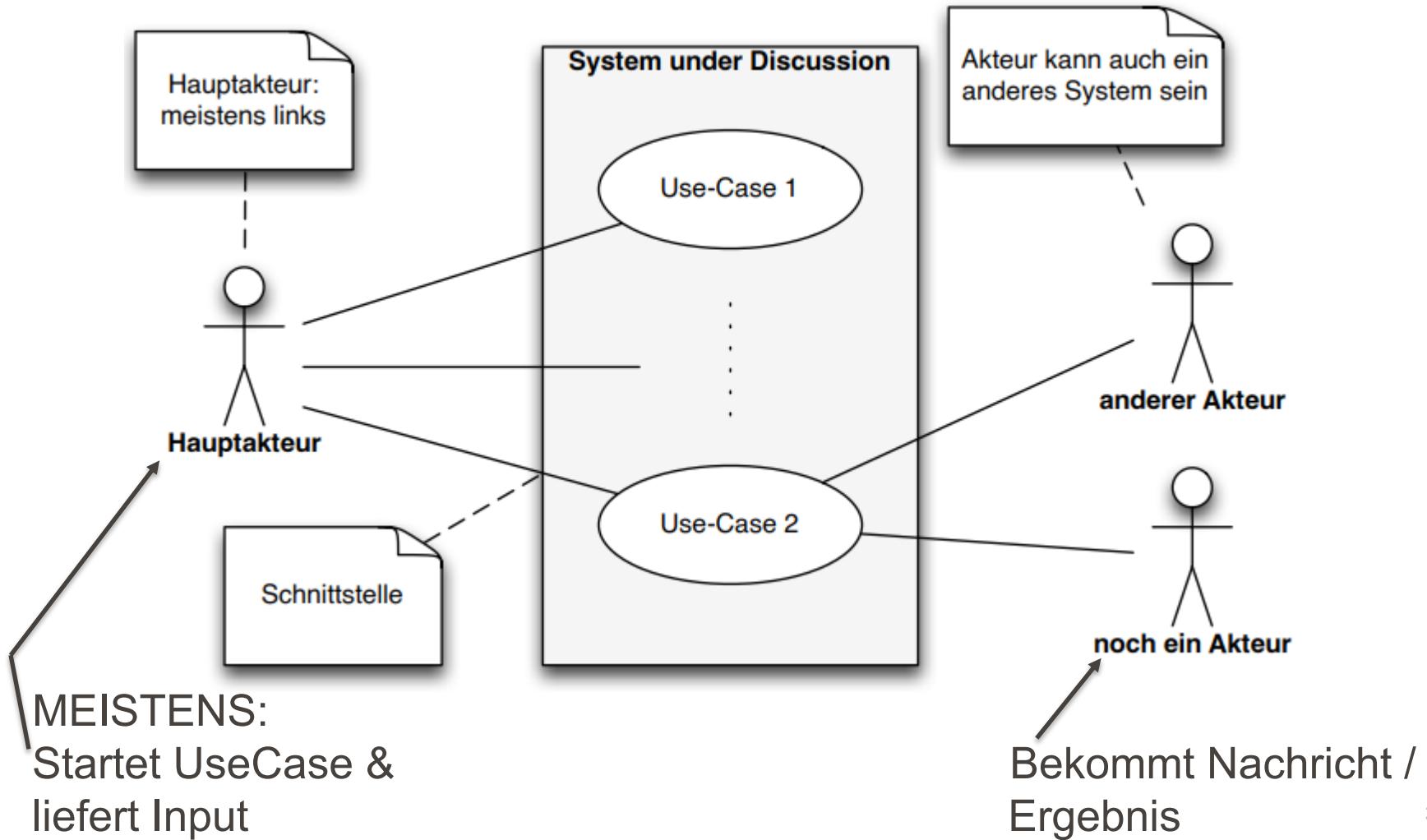
→ Auch Teil der UML zur Übersicht, über Anwendungsfälle

# UML – ANWENDUNGSFALLDIAGRAMM



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

Allgemeine Form des UseCase-Diagramms:



# UML – ANWENDUNGSFALLDIAGRAMM



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Das Anwendungsfalldiagramm (UseCase-Diagram) enthält:
  - Auflistung der Namen der Anwendungsfälle
  - Auflistung der Akteure
  - Beteiligung von Akteuren an Anwendungsfällen
  - <<extends>>, <<includes>> (sparsam verwenden)

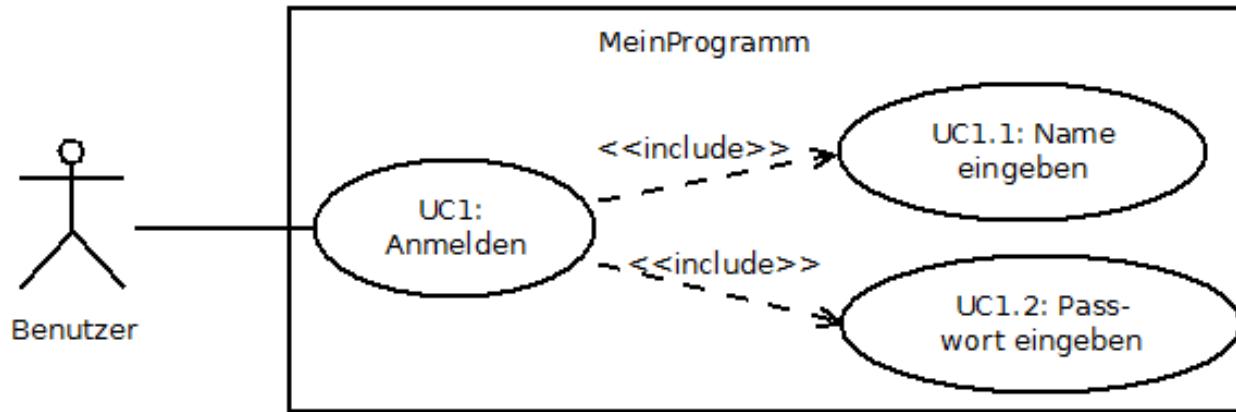
 Ein Anwendungsfalldiagramm ist keine:

- Darstellung von Ablaufschritten
- Darstellung zeitlicher Zusammenhänge
- Keine Reihenfolge!
- Ein Anwendungsfalldiagramm ist kein Aktivitätsdiagramm!

# → BESCHREIBUNG EINES ANWENDUNGSFALLS:



- Was halten Sie von diesem Use Case Diagramm?

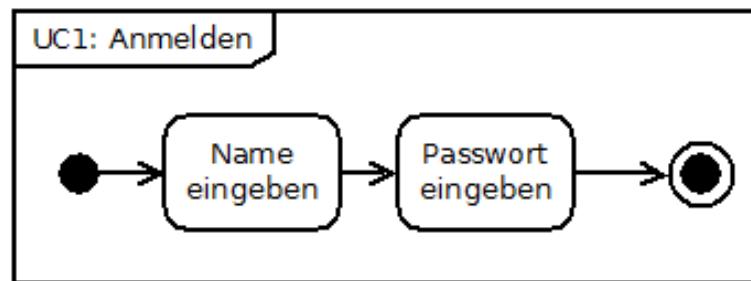
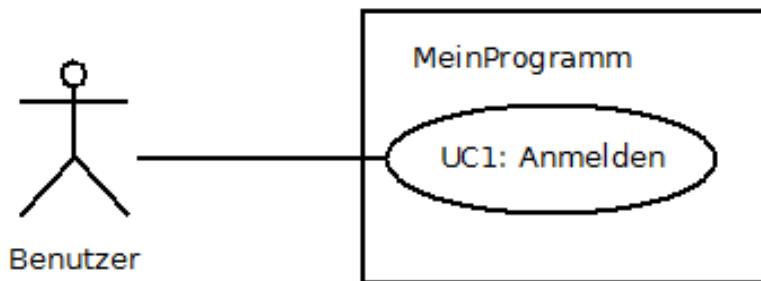


Definiert nicht, dass UC1.1 vor UC1.2



Viel zu feingranular → Anwendungsfalldiagramm ist kein Aktivitätsdiagramm!

→ Dann besser so:



# → BESCHREIBUNG EINES ANWENDUNGSFALLS:



Hochschule RheinMain	
ID: Name	UC01: Titel hinzufügen
Version	Der Titel ist in der gewünschten Playlist enthalten.
Autor	Anwender
Entwurf	-
System	Kauf/Erhalt einer neuen Track.
Wieso	Um einen in Form einer Audio Datei im Daten-System vorliegen.
Verbindungen	Das System ist geöffnet und zeigt das Hauptfenster
Nachbedingungen (Ergebnis)	Die Verbindung in den Original-Audio-Datei, System enthält die Informationen über den Titel und die zugehörigen Audio-Daten.
Möglich:	Keine Veränderung am System.
Standardablauf	1: Anwender klickt auf die Funktion „Titel hinzufügen“ 2: Das System zeigt den Dialog „Titel hinzufügen“ (DIALOG_XY) 3: Anwender wählt eine Playlist aus. 4: ... 5: ... 6: ... 7: System importiert den Titel. 8: Noch keine Playlist vorhanden → System erstellt automatisch eine Default-Playlist
Alternative Ablaufschritte	... 4a: Titel bereits vorhanden → Korrektur durch Anwender oder Abbruch 4b: ... andere Ausnahme ... - ... ... 7a: Import schlägt fehl → ???
Zulassen	Keine Einschränkung
Verfügbarkeit	Muss immer möglich sein
Fragen	
Kommentare	

## • Wichtigste Form: Textuelle Beschreibung →

### • Mögliche (zusätzliche) Formen:

- Anwendungsfalldiagramm
- 1 Anwendungsfall → 1 Aktivitätsdiagramm
- 1 Anwendungsfall → 1 Zustandsdiagramm
- mehrere Anwendungsfälle → 1 Aktivitätsdiagramm
- mehrere Anwendungsfälle → 1 Zustandsdiagramm
- Auch möglich: Sequenzdiagramm
- BSP: Zustandsdiagramme für Sitzheizung, ...



### Vorsicht:

- Ein Anwendungsfalldiagramm ist kein Aktivitätsdiagramm!
- Ein Anwendungsfalldiagramm ist kein Aktivitätsdiagramm!
- Ein Anwendungsfalldiagramm ist kein Aktivitätsdiagramm!

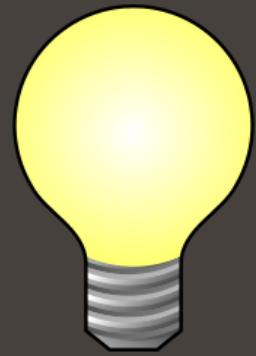


04

## Nichtfunktionale Anforderungen

Ziel:

Nichtfunktionale Anforderungen erfassen



# NICHTFUNKTIONALE ANFORDERUNGEN



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Übliche Kategorien:
  - Performance & Zeitverhalten
  - Veränderbarkeit, Wartbarkeit
  - Bedienbarkeit
  - Sicherheit (Security & Safety)
  - Testbarkeit
  - Korrektheit
  - Verfügbarkeit, Zuverlässigkeit
  - ...
- Übliche Beschreibungsform: Textuelle Beschreibung
- Aber auch in Modellen
  - Z.B.: Zeitverhalten als Constraints in UML-Diagrammen

# NICHTFUNKTIONALE ANFORDERUNGEN



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim



Vorsicht:

- Werden sehr gerne vernachlässigt!
  - Leider auch hier etwas aus Zeitgründen!
- Viele Projekte scheitern, weil wichtige NFkt. Anforderungen vernachlässigt wurden
  - Z.B.: SW zu langsam (Performance), SW schlecht bedienbar, SW nicht mehr wartbar, SW nicht mehr änderbar
    - Oft Kombination aus mehreren
- Problem: Oft schwer greifbar und werden deshalb vergessen

→ Mehr dazu:

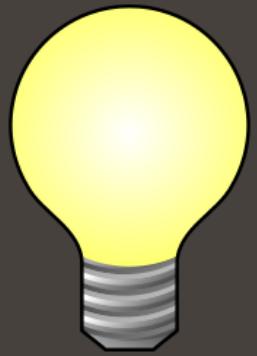
- Wahlpflichtveranstaltung Anforderungsmanagement (im WS)



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# 05 GUI

Ziel:  
Kundenanforderungen für die GUI erfassen



# GUI IN DER ANFORDERUNGSANALYSE



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Was sollte man zur GUI in der Anforderungsanalyse aufschreiben?
  - Zumindest die Informationen, die benötigt werden, um
    - Anwendungsfälle
    - Fachmodellzu verstehen.
  - Oft: Vorgriff auf Entwurf
    - Grober Entwurf der GUI
      - Fenster mit UI-Elementen (Werkzeug: z.B. UI-Editor)
      - (grobe) Navigation (z.B. mit Zustandsdiagramm)
      - Klick-Prototyp (z.B. mit HTML)

# GUI IN DER ANFORDERUNGSANALYSE



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

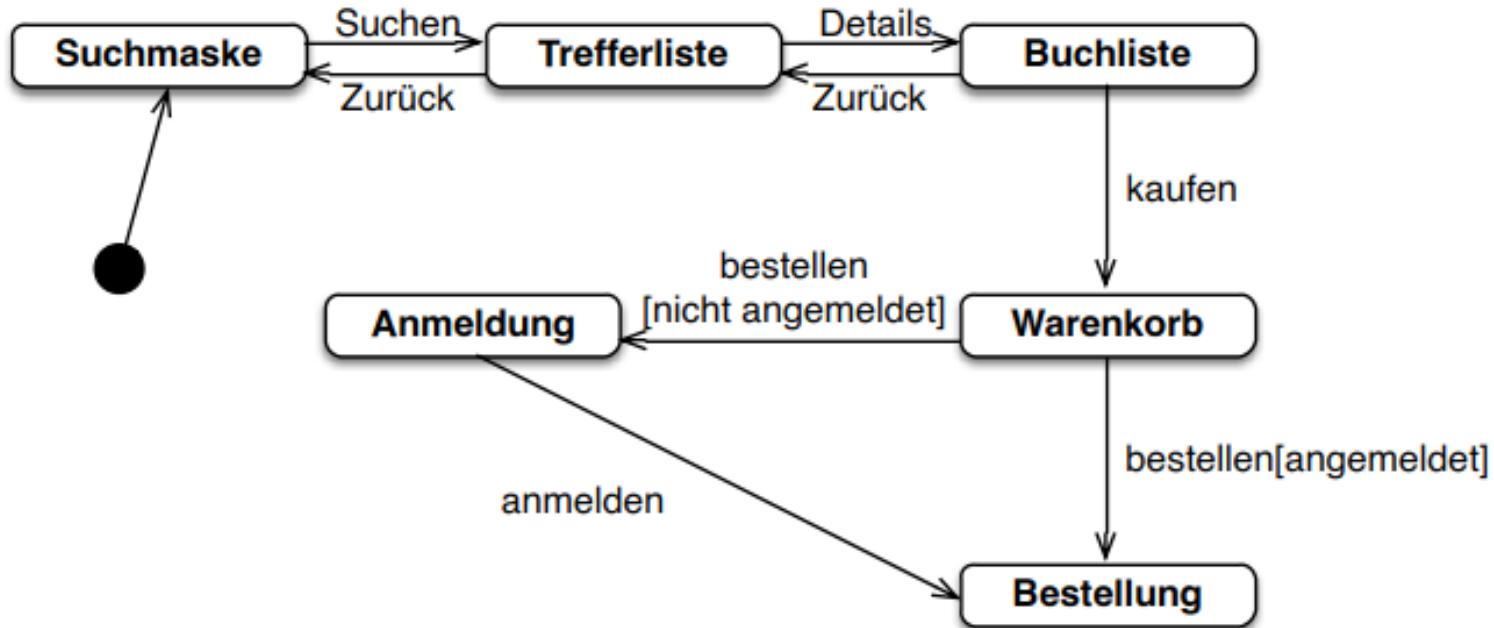
- Warum sollte man bereits bei der Anforderungsanalyse auf die GUI eingehen?
  - Risiken minimieren
- Schlechte Bedienbarkeit führt oft zum Scheitern von Projekten
  - BSP: IBM Lotus Notes
    - Super Technologiekonzept → Perfekt für jede IT-Abteilung
    - ABER: Bedienbarkeit fürchterlich
      - Z.B.: F3 → Beenden des Programms ohne Speichern
  - Gute Bedienbarkeit kann über vieles hinwegtrösten → Hype
    - BSP: Apple
      - Super Bedienbarkeit
      - Völlig überteuerte Hardware, Kaum Reparaturmöglichkeiten, Inkompatibilitäten / Vendor-Lockin, Teurere Onlineshoppingpreise,

...

# GUI IN DER ANFORDERUNGSANALYSE



- Beispiel: Navigationsregeln für Online-Buchladen



→ Zustandsdiagramm:

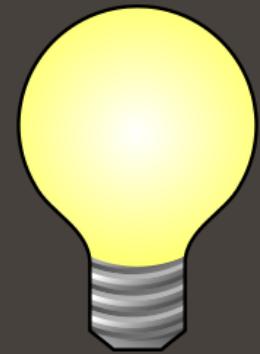
- Zustände  $\triangleq$  Bildschirmmasken / Bedienabschnitten in Bildschirmmasken
- Übergänge  $\triangleq$  Aktionen (z.B. Klicks auf Buttons, ...)



# 06

## Qualitätssicherung

Ziel:  
Vorgriff auf spätere Vorlesung  
→ Jedoch fängt hier schon alles an



# QUALITÄTSSICHERUNG



- Wie können wir sicherstellen, dass . . .
  - die Anforderungsanalyse richtig durchgeführt wurde?
  - die (noch zu bauende) Anwendung die Anforderungen erfüllt?
- Antworten:
  - Überprüfung der erstellten Dokumente → Reviews
  - Vorbereitung des funktionalen Systemtests
    - Input: Use Cases, Fachmodell, GUI-Entwurf
    - Output: Testfälle + Testskripte + ggfs. Testwerkzeuge
  - Vorbereitung der nichtfunktionalen Systemtests
    - Input: Nichtfunktionale Anforderungen
    - Output: Testfälle + Testskripte + ggfs. Testwerkzeuge

→ Details dazu später bei der Vorlesung über Testen

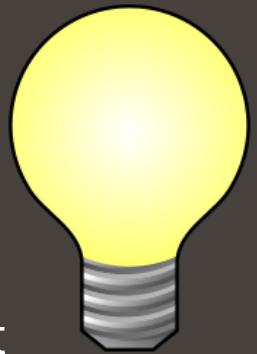


07

## Das war's noch lange nicht

Ziel:

Ausblick auf Wahlpflichtvorlesung Anforderungsmanagement



# ANFORDERUNGSANALYSE – DAS WAR SEHR, SEHR KURZ!



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

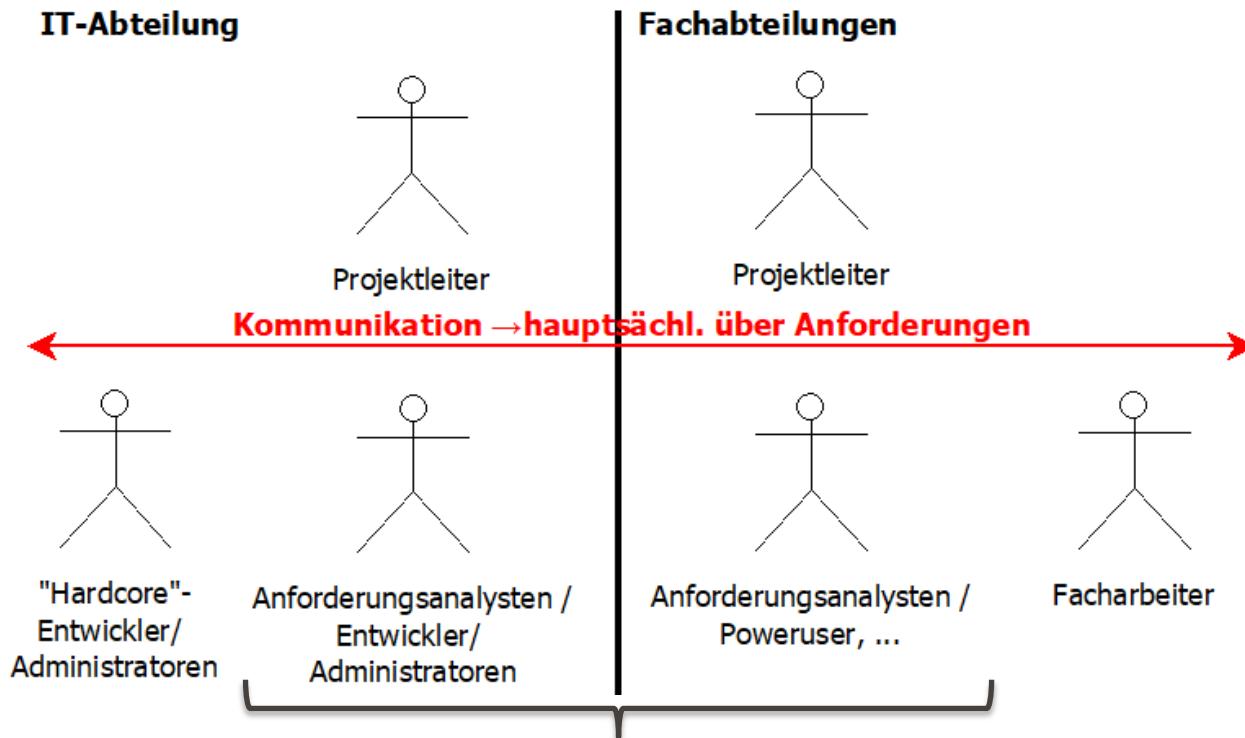
- Das war alles sehr, sehr kurz
  - ABER: Thema ist sehr wichtig, weil Anforderungen die Basis aller weiteren Schritte bilden
  - Gerade auch Kernaufgabengebiet der WI
    - Siehe folgende Folie
- In der Wahlpflichtvorlesung Anforderungsmanagement können Sie wesentlich mehr dazu erfahren
- Wie finde ich Anforderungen überhaupt?
    - Muss man aufdecken
    - Sehr viel Psychologie & Kommunikation
  - Wie schreibe ich Anforderungen möglichst eindeutig auf?
  - Nichtfunktionale Anforderungen
  - Wie manage ich Anforderungen und Anforderungsänderungen?
  - ...

# ANFORDERUNGSMANAGEMENT – EIN KERNGEBIET DER WI



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Kernaufgabengebiet der WI
  - IDEE der WI: Übersetzer Fachabteilung ↔ IT-Abteilung:



Typisches Berufsbild WI  
→ Sie können natürlich auch etwas anderes machen  
(Gerade unser technischer Fokus ist sehr gefragt)

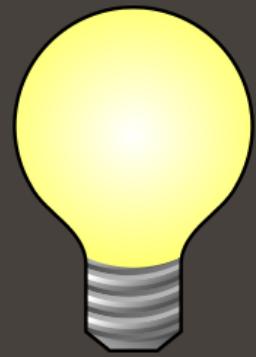


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# 08

# Fazit

Ziel:  
Was haben wir damit gewonnen?





# WAS HABEN WIR GELERNT?

- Anforderungen sind sehr wichtig
  - Müssen ja wissen was wir eigentlich entwickeln wollen
- Verschiedene Aspekte
  - Fachmodell
  - Anwendungsfälle
  - Nichtfunktionale Anforderungen
  - GUI
  - Qualitätssicherung
- Leider haben wir viel zu wenig Zeit
  - Das Thema ist viel wichtiger
  - Wird auch so gerne unterschätzt



# WEITERFÜHRENDE LITERATUR

- Kleuker: Grundkurs Software-Engineering mit UML [<http://dx.doi.org/10.1007/978-3-8348-9843-2>].
- Zuser et al: Software-Engineering mit UML und dem Unified Process [BF 500 92].
- Larman, C.: Applying UML and Patterns [30 BF 500 78].



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

AUF GEHT'S!!

SELBER MACHEN UND LERNEN!!



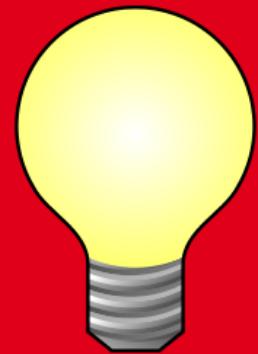


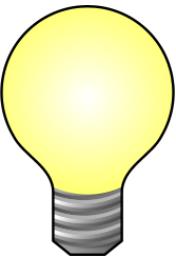
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

14.01.2021

# Programmieren im Großen III

Grobentwurf (Architektur)





## AGENDA

Einführung ins Thema

Begriffsabgrenzungen

Erste Schritte

Fundamental Modeling Concepts

Die 3-Schichten-Architektur

Den Grobentwurf überprüfen (Robustness Analysis)

Wie geht's dann weiter?

Fazit

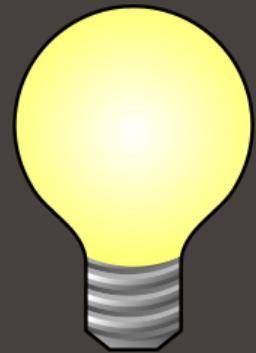


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# 01

## EINFÜHRUNG INS THEMA

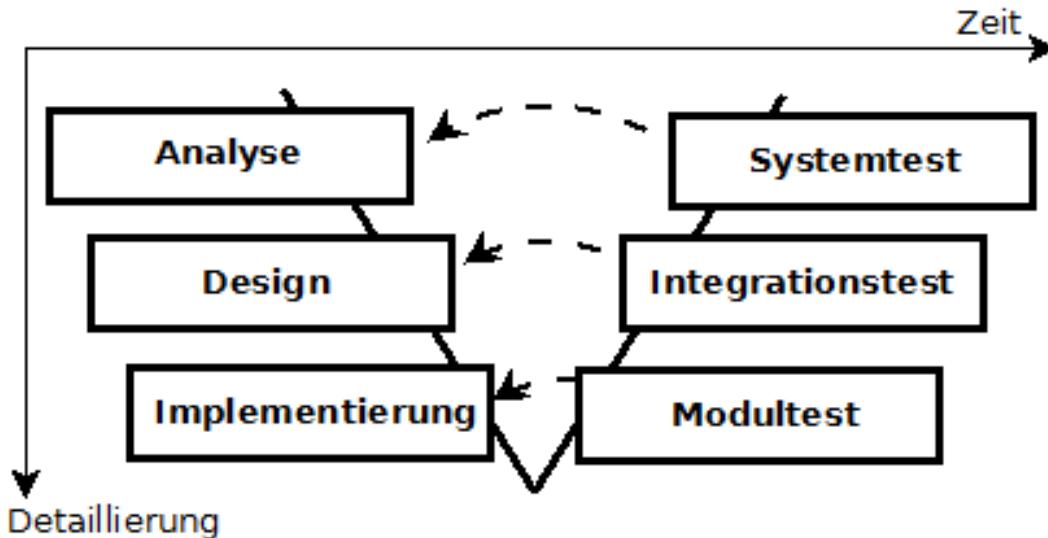
Ziel:  
Die Eckpunkte des Themas kennenlernen





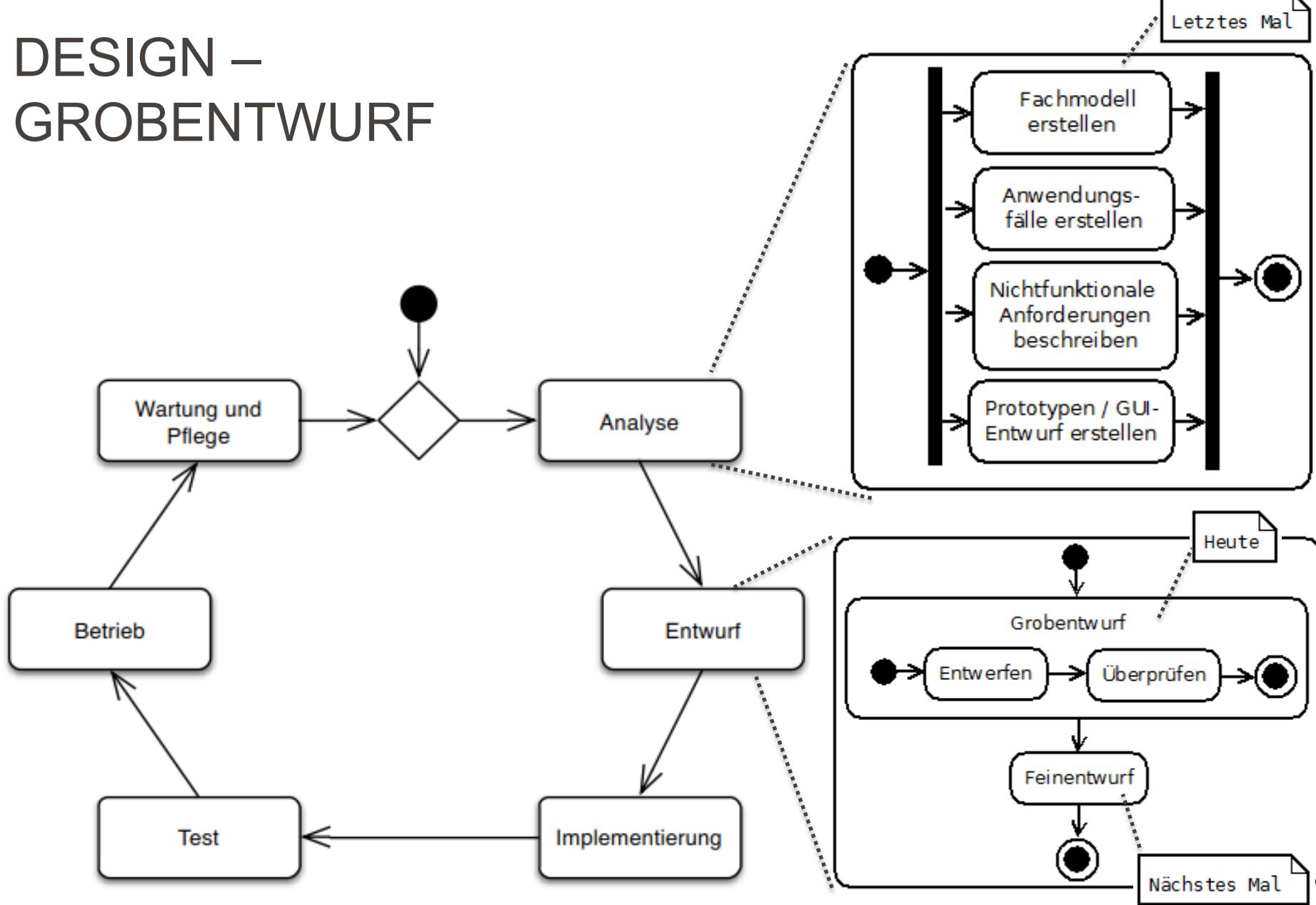
# WORUM GEHT'S?

- Wir brauchen ein Vorgehensmodell
  - Verschiedene Phasen:



- Heute besprechen wir:
  - Design: Grobentwurf (Architektur)

# DESIGN – GROBENTWURF



# ENTWURF (DESIGN) – WORUM GEHT'S?



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

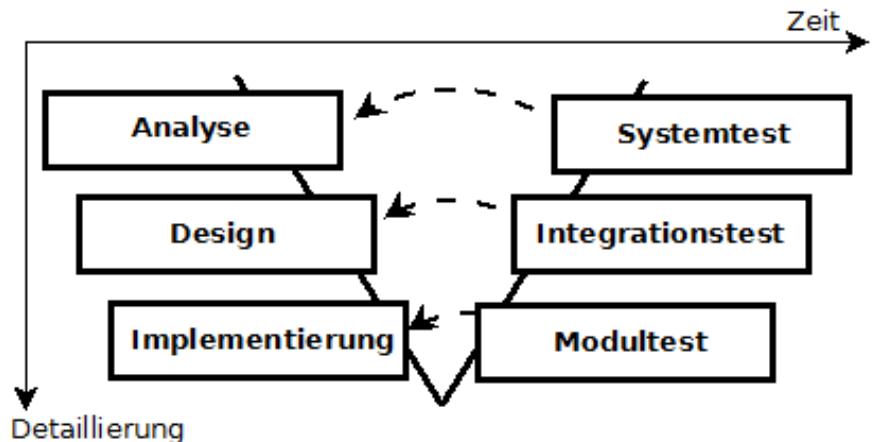
- Grundlegende Frage
  - Wie soll das zu bauende System sein?
- Entwurf (Design)
  - Tätigkeiten
    - Grobentwurf (Architektur) → heute
    - Feinentwurf (Detailed Design) → nächstes Mal
  - Sprechweise im (R)UP:
    - „Analysis and Design“
  - Sprechweise im V-Modell:
    - „Design“

# HINWEIS: ANALYSE IN V-MODELL & RUP

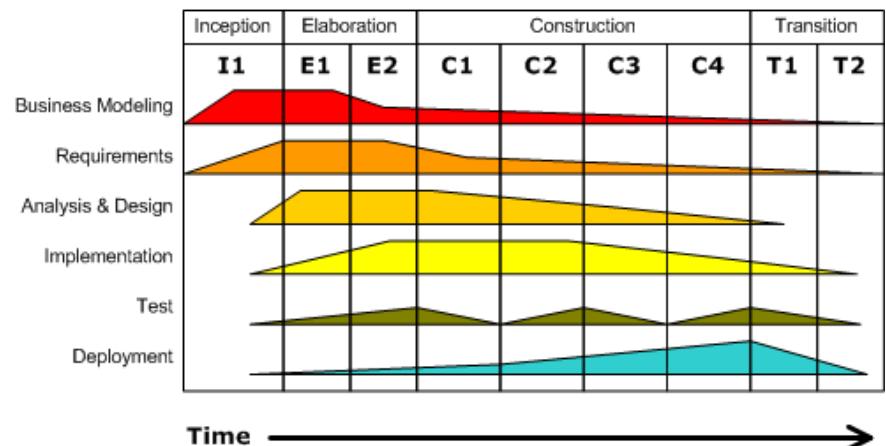


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Analyse im V-Modell:
  - Anforderungserhebung & Analyse (letzte Einheit)
- Analyse im (R)UP:
  - Grobentwurf erstellen (Architektur)
  - + Grobentwurf im Hinblick auf Anforderungen überprüfen (Robustness Analysis)



**Iterative Development**  
Business value is delivered incrementally in time-boxed cross-discipline iterations.



# GROBENTWURF – WORUM GEHT'S?



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

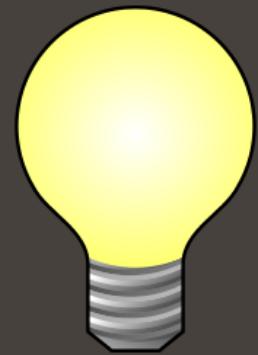
- Ziele:
  - Grobentwurf (Architektur) für das zu bauende System
  - Anforderungen sind vollständig und widerspruchsfrei
  - Grobentwurf erfüllt Anforderungen
  - System kann gebaut werden (build-able)



## 02

# Begriffsabgrenzungen

Ziel:  
Missverständnisse Vorbeugen  
→ Begriffe sauber abgrenzen





# VORSICHT: EIN GERÜCHT GEHT UM



Hochschule RheinMain  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Manche Autoren behaupten . . .
  - Fachmodell (aus der Analyse)
  - + Typen
  - + Methoden
  - ⇒ OO-Programm



Das funktioniert nur in einfachen Fällen!

- Meistens:
- Mehrstufiger Entwurfsprozess
  - Weitere/andere Klassen
  - Umcodierung der Daten (andere Datenstruktur)
  - . . .

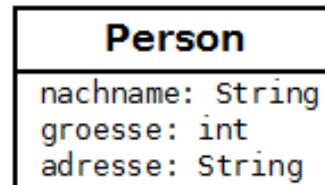


# VORSICHT: EIN GERÜCHT GEHT UM



→ **Fachliche Sicht betrachtet alles ohne technische Details**

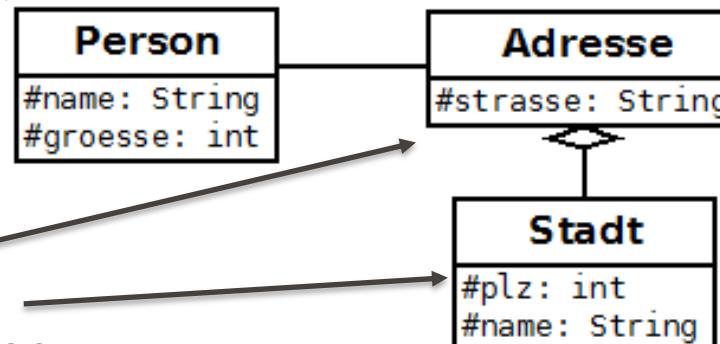
- Wird in der Analyse verwendet, um **NUR** die fachlichen Anforderungen zu ermitteln → Fachmodell



Hier funktioniert es schon mal nicht!!

→ **Technische Sichten**

- In Architektur, Detailed Design, Implementierungsdoku verw.
- Es werden auch die für die Lösung **relevanten** technischen Details dargestellt:



Technische Lösung  
z.B. für spezifische Suche  
nach Städten oder Adressen

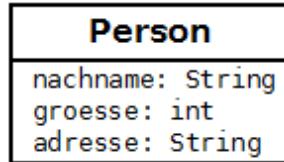
# BEGRIFFE ZUR KLAREREN ABGRENZUNG:



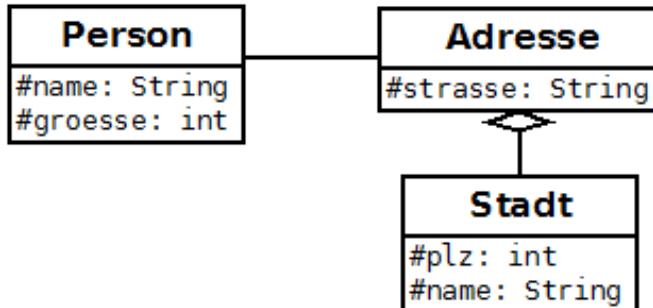
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

Zur klareren Abgrenzung verwenden wir folgende Begriffe:

- In (Anforderungs-)Analyse:
  - **Fachmodell**: Klassenmodell zur fachl. Beschreibung der Anf:



- In Architektur (Grobdesign) und Detailed Design (Feindesign):
  - **Domänenmodell**: techn. Klassenmodell der Fachklassen



Vorsicht: Wir nennen das so, leider werden in der Literatur oft die Begriffe für Fachmodell, Domänenmodell, fachl. Datenmodell, ... durcheinander gewürfelt

# BEGRIFFE ZUR KLAREREN ABGRENZUNG:



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

Generell gilt damit:

- In (Anforderungs-)Analyse:
  - **Fachmodell\***: Klassenmodell zur fachl. Beschreibung der Anf.
    - + Dynamische Aspekte (z.B. Use Cases)
- In Architektur (Grobdesign) und Detailed Design (Feindesign):
  - **Domänenmodell\***: techn. Klassenmodell der Fachklassen
    - + Dynamische Aspekte (Geschäftsregeln)
      - Meist als Methoden in Klassen des Domänenmodells
      - Oder eigener Controller (siehe später MVC-Muster)
    - + Sehr viele weitere Klassen für das „DrumHerum“ (GUI, Persistenz, ...)

Geschäftslogik

\*Unsere Terminologie, damit Sie es besser unterscheiden können



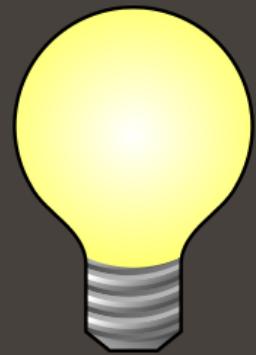
Vorsicht: Auch Geschäftsregeln (Business Rules), Geschäftslogik, ... werden oft im Zusammenhang mit Anforderungsanalyse benutzt und es gibt auch hier keine klare Abgrenzung!



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## 03 Erste Schritte

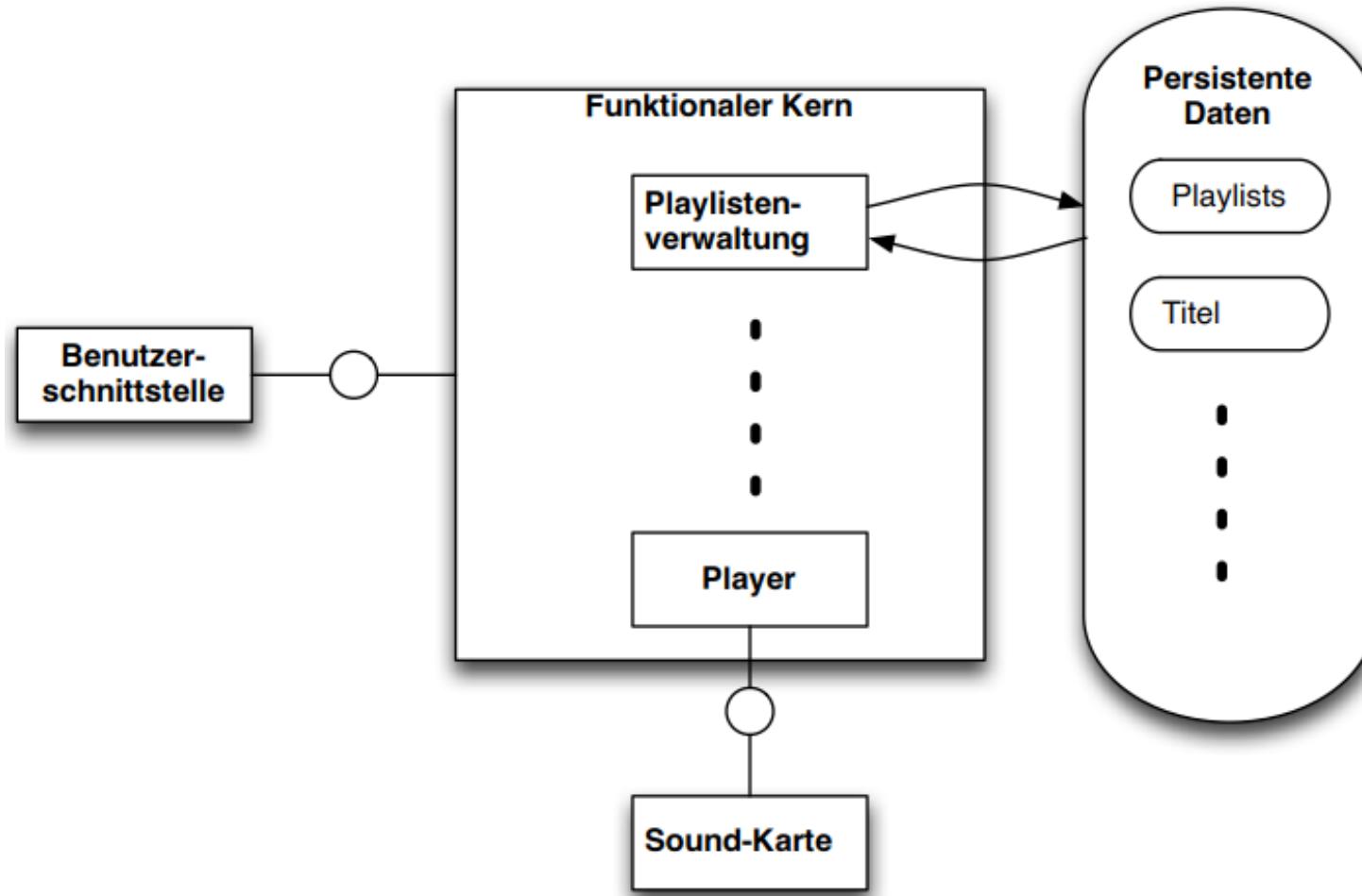
Ziel:  
Erste Schritte anhand eines Beispiels





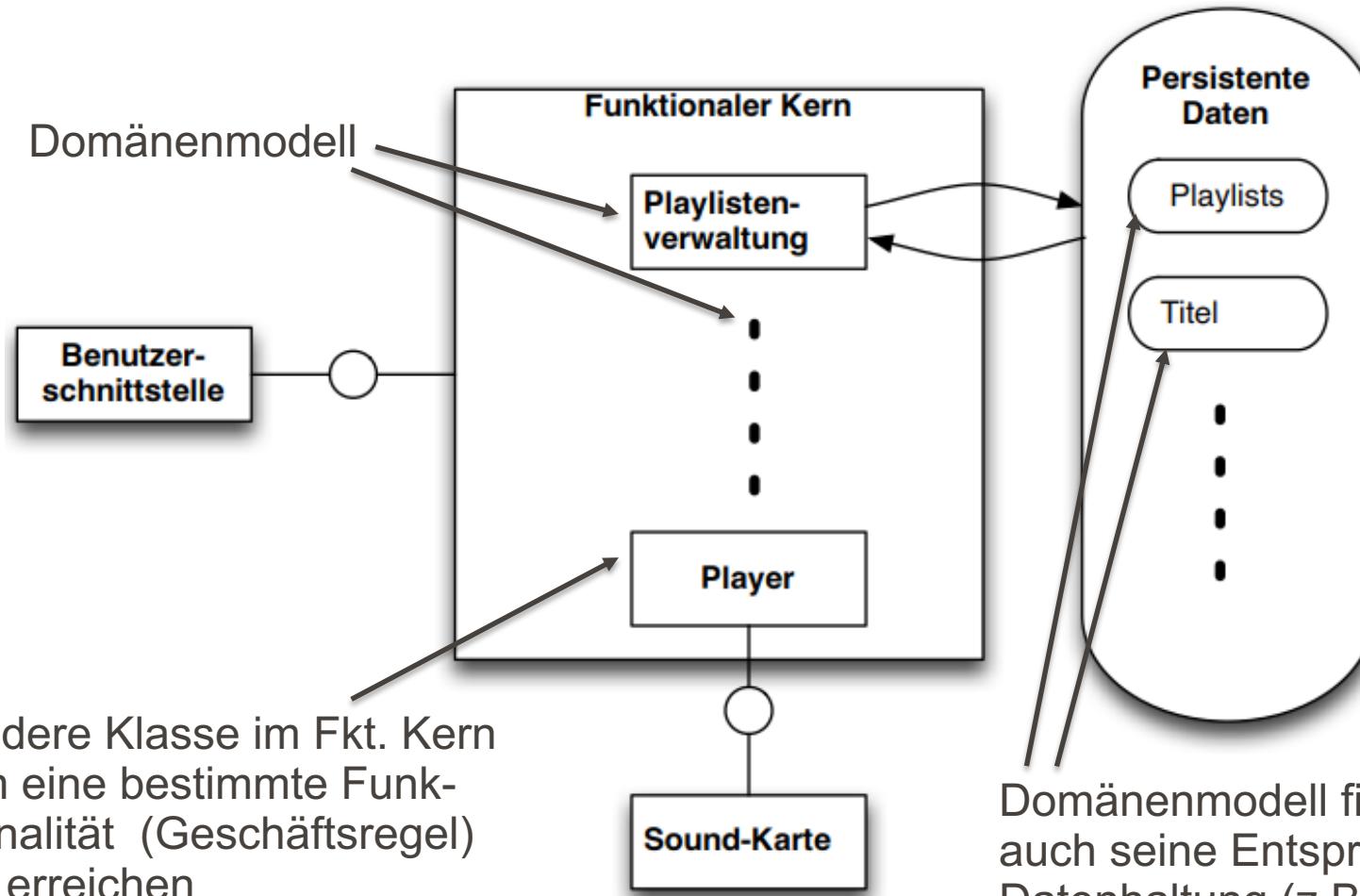
# BEISPIEL GROBENTWURF

- BSP: Grobentwurf MP3-Player (unvollständig):



# BEISPIEL GROBENTWURF

- BSP: Grobentwurf MP3-Player (unvollständig):



# ARCHITEKTUR (GROBENTWURF)



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Ziel: Grundlegenden Aufbau der Anwendung festlegen
- Input:
  - Anforderungen
  - Andere Entscheidungen (dokumentieren!)
- Muss für Architektur:
  - erfüllt Anforderungen
  - „solide“ (Architektur verträgt z.B. problemlos Änderungen an den funktionalen Anforderungen)
    - Leider schwer die Änderungen vorherzusehen
    - Leider auch so schwer zu überprüfen
    - Erfahrung



## Vorsicht:

- Architektur-Entscheidungen haben weitreichende Konsequenzen!
- Genau analysieren und die wichtigsten Dokumentieren!



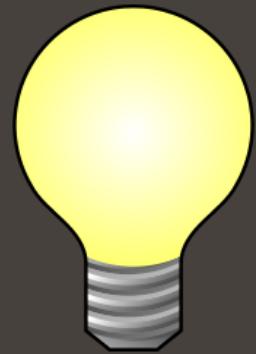
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

04

## Fundamental Modeling Concepts (FMC)

Ziel:

Die Architekturentwurfssprache FMC kennenlernen



# WIE KANN MAN DIE ARCHITEKTUR MODELLIEREN?



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Mit UML
  - Klar, ist das die Intention der UML
  - Allerdings gibt es ein paar Defizite:
    - Konzentriert sich sehr auf die Datenmodellsicht (Z.B. Klassen)
    - Datenspeicherebene wird vernachlässigt
      - Zugriff auf Datenspeicher wird nicht thematisiert
    - Rel. frühe Phase → Manchmal noch gar nicht klar, ob ein Konzept wirklich eine Klasse wird
      - (oder eher Attribut, Methode, oder ...)
    - Kompositionssstruktur-Diagramme wären prinzipiell geeignet
      - ABER für den Zweck recht umständlich
    - Verteilungsdiagramme scheinen sich anzubieten
      - ABER für Software-Architektur nur bedingt geeignet
        - (Fokus: Verteilung der Software auf Hardware)
- UML ist nicht so gut geeignet

# WIE KANN MAN DIE ARCHITEKTUR MODELLIEREN?



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Hilfreiche **und** einfache Notation für Architektur nötig
  - FMC-Block-Diagramme
- BEM: Auch über Profilmechanismus könnte man UML auch „umbiegen“ eine ähnliche Syntax & Semantik wie FMC-Blockdiagramm zu haben
  - Technical Architecture Management (TAM) → FMC + UML (<http://www.fmc-modeling.org/>)

# FMC – ALLGEMEINE INFOS



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- FMC = Fundamental Modeling Concepts
  - Siehe: <http://www.fmc-modeling.org/>
  - Im Umfeld von SAP (Hasso-Plattner-Institut) entwickeltes Modellierungskonzept für Architekturen
    - Ursprünge schon aus den 1970ern in Uni Kaiserslautern entwickelt
- FMC besteht eigentlich aus 3 Diagrammarten:
  1. FMC-Block-Diagramme → Struktur der Software
  2. Petri-Netze → Dynam. Abläufe / „Geschäftsprozesse“
  3. Wertebereichsdiagramme (WBD) → Erweiterte Entity-Relationship-Diagr.
- Vorschlag hier:
  - 2 & 3 kann man auch sehr gut durch UML abdecken (vgl. TAM)
    - 2. Petri-Netze → Aktivitätsdiagramme (oder BPMN, ...)
    - 3. WBD → UML-Klassendiagramm (→ siehe Domänenmodell)
  - Block-Diagramm bringt aber Vorteile für die Architekturübersicht

# FMC-BLOCKDIAGRAMM – KOMPONENTEN:



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Agent (= aktive/interagierende Komponente):



- Kommunikationskanal (= passive Komponente ohne Gedächtnis):



→ Verbindet Agenten

- Speicher (= passive Komponente mit Gedächtnis):

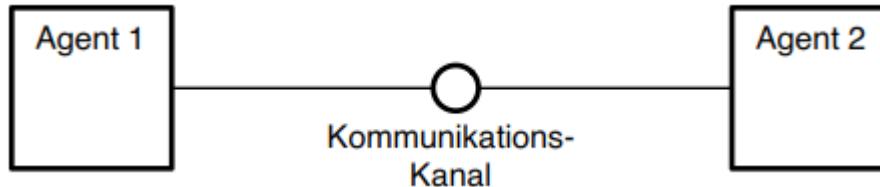


# NOTATION FMC-BLOCKDIAGR. – KOMMUNIKATION

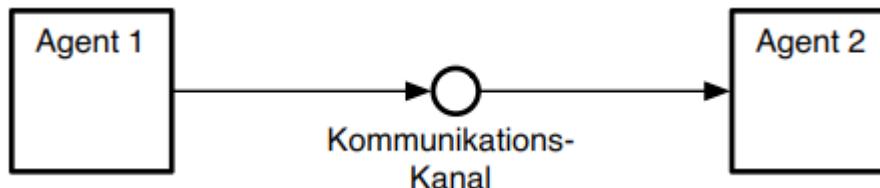


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

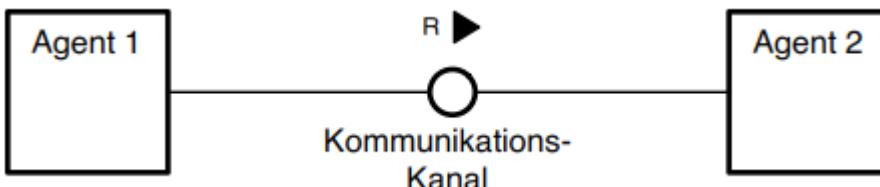
- Kommunikation zwischen Agenten:
  - Richtung unspezifisch (unbekannt/unwichtig):



- Unidirektional (nur eine Richtung):



- Anfrage/Antwort:

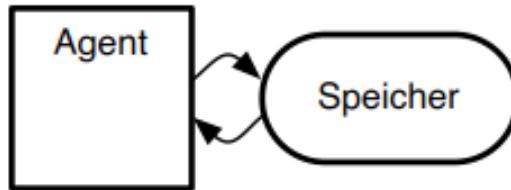


# NOTATION FMC-BLOCKDIAGR. – KOMMUNIKATION

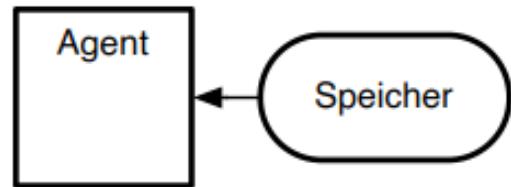


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

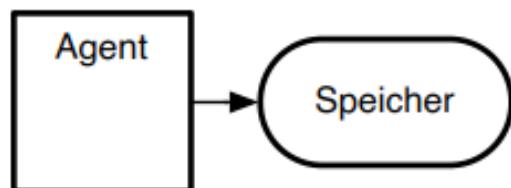
- Speicherzugriff :
  - lesen + schreiben (Normalfall):



- nur lesen (häufiger Fall):



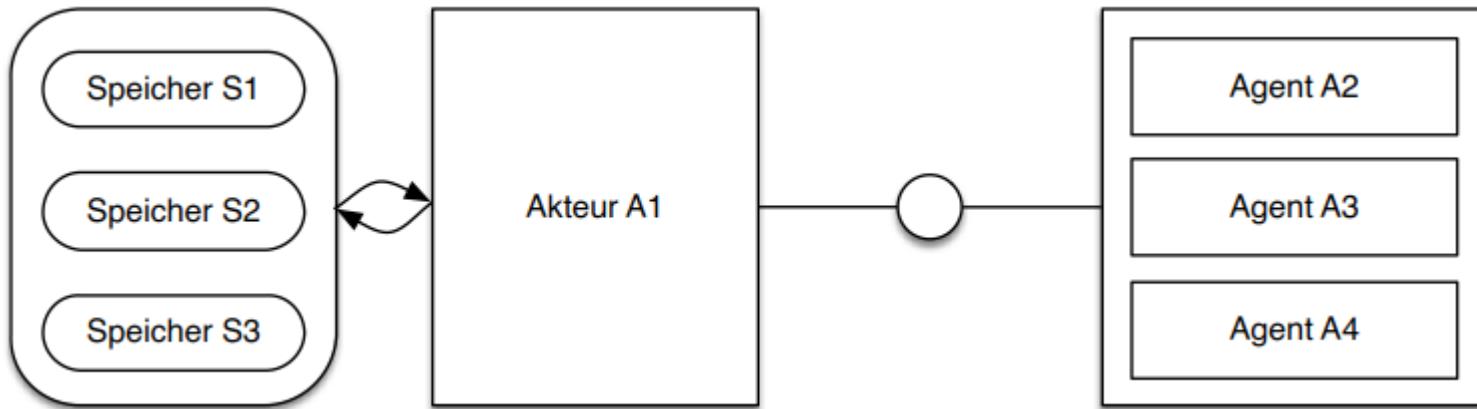
- bei jedem Zugriff erst löschen, dann schreiben (seltener Fall):



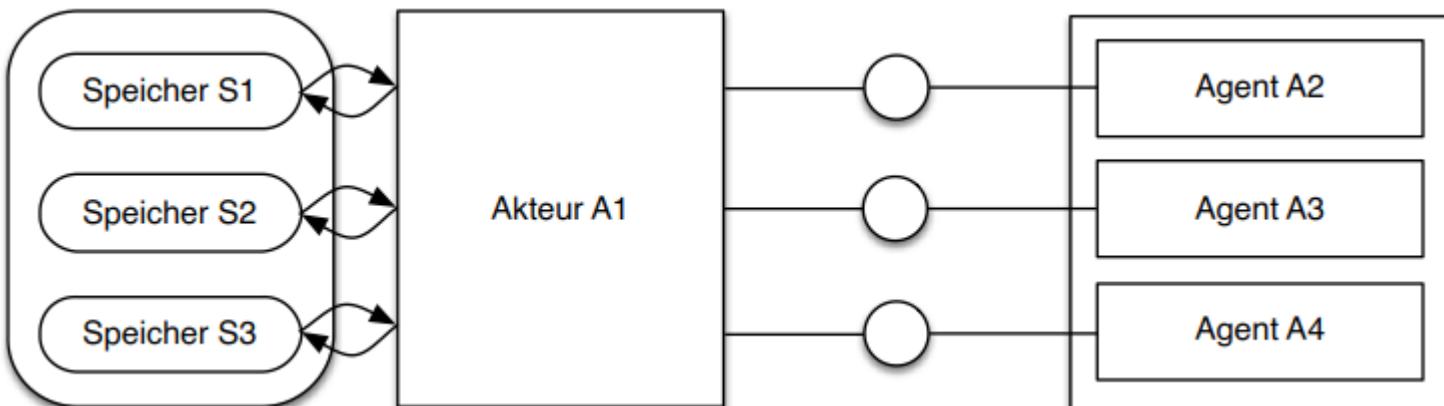
# NOTATION FMC-BLOCKDIAGR. – KURZSCHREIBWEISE



- Zwei gleichwertige FMC-Block-Diagramme:
  - „Kurzschreibweise“:



- „Ausgeschrieben“:





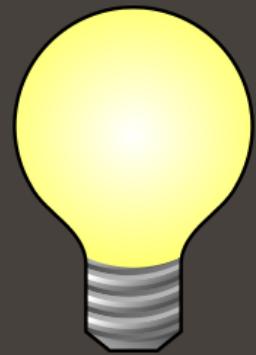
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

05

## Die 3-Schichten Architektur

Ziel:

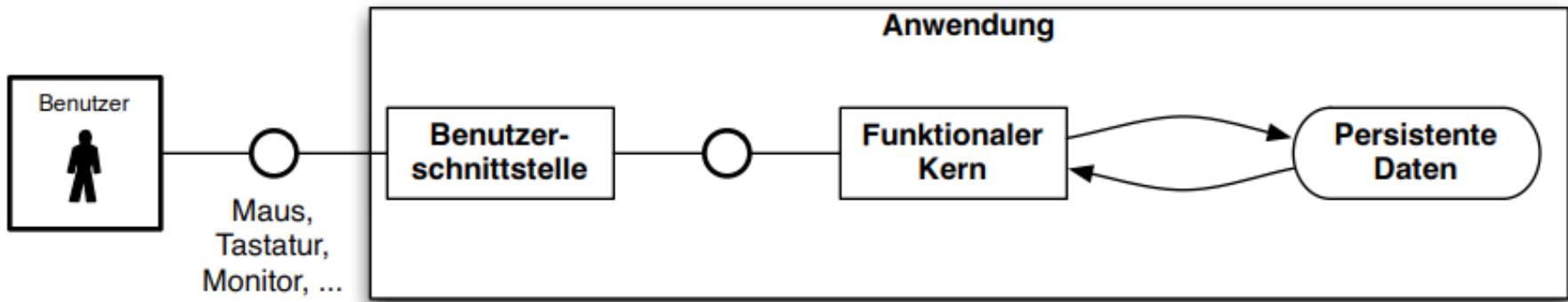
Das Architekturmuster 3-Schichten-Architektur  
kennenlernen



# DIE 3-SCHICHTEN-ARCHITEKTUR



- 3-Schichten-Architektur (Engl.: 3-Tier Architecture):



→ Beliebter Architektur-Ansatz (Muster)

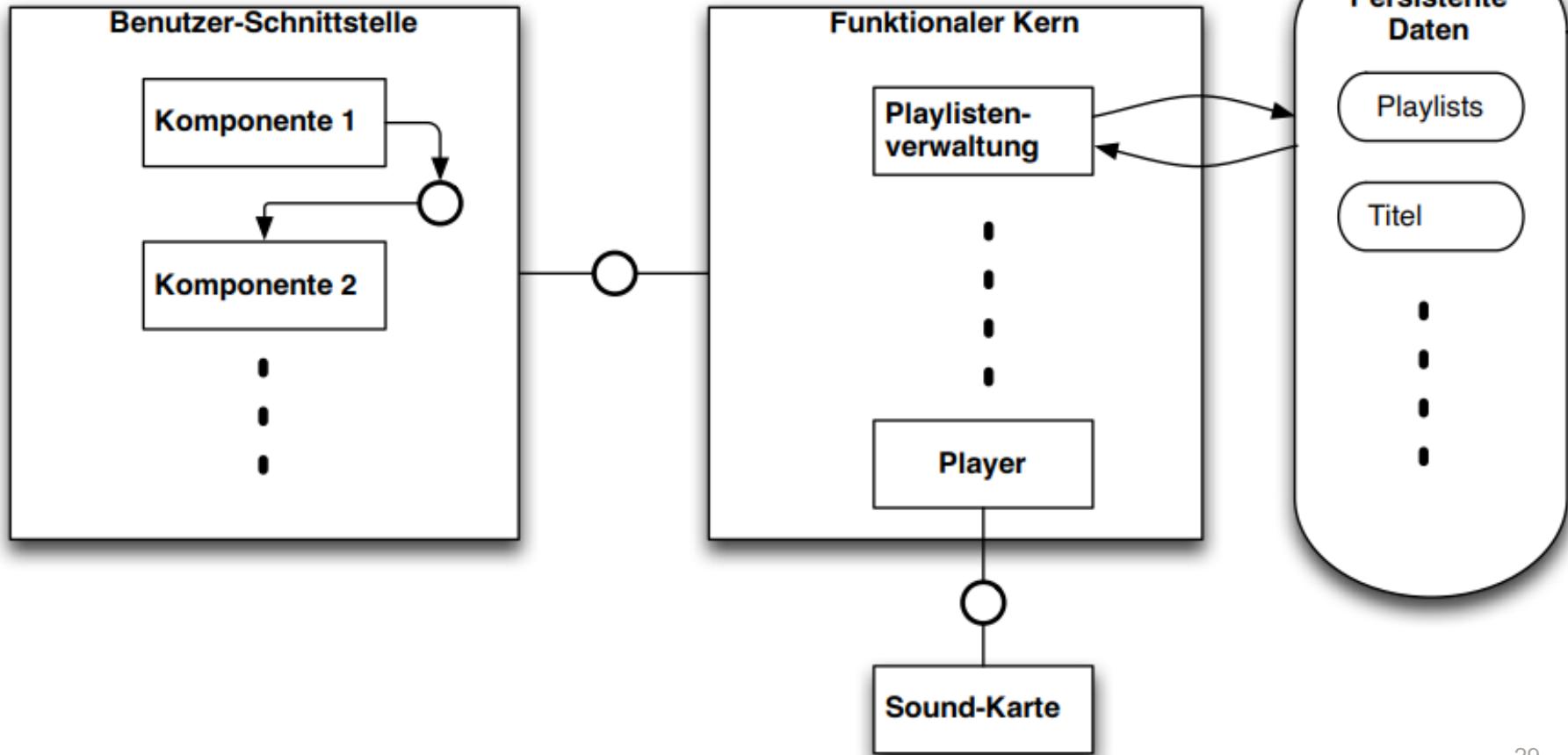
- Aufgabenteilung in 3 Schichten:
  - Benutzerschnittstelle:
    - Darstellung + Benutzereingaben
  - Funktionaler Kern:
    - Domänenmodell + Anwendungs-/Geschäfts-Logik
  - Datenhaltung: siehe Vorlesung „Datenbanksysteme“

# BEISPIEL 3-SCHICHTEN-ARCHITEKTUR



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Beispiel: MP3-Player in 3-Schichten-Architektur:  
→ Grobentwurf (verfeinert, immer noch unvollständig)



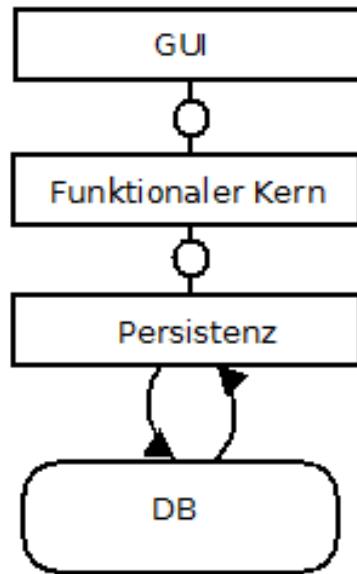
# HINWEIS ZUR 3-SCHICHTEN-ARCHITEKTUR



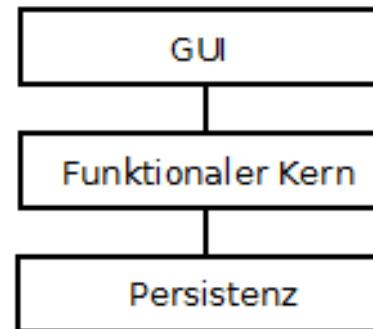
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Wird oft eher umgekehrt von oben nach unten dargestellt  
→ Schichten

So (z.B. FMC):



Oder auch so (z.B. UML):



- Oft wird DB weggelassen und als Teil der Persistenzschicht betrachtet
- Kann aber auch ohne DB sein (z.B. dann XML-Dateien, ...)
  - Deshalb: Besser auch die DB, ... darstellen

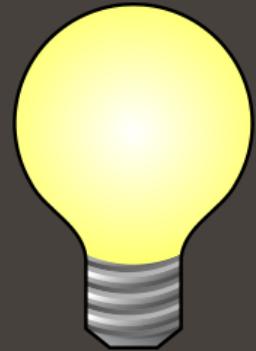


06

## Den Grobentwurf überprüfen

Ziel:

Den Grobentwurf mittels Reviews überprüfen  
→ Am Beispiel der Robustness Analysis (aus RUP)



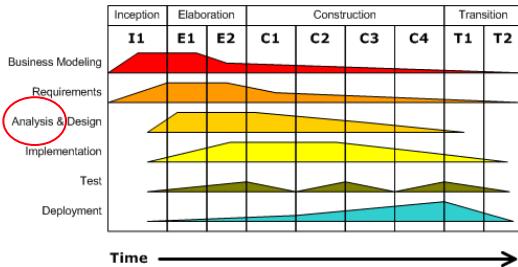
# GROBENTWURF ÜBERPRÜFEN – GRUNDIDEE



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

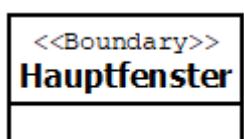
- Typische Fragen / Probleme beim Grobentwurf
  - Sind alle Anforderungen abgedeckt?
  - Haben wir etwas vergessen?
  - Kann das funktionieren?
  - . . .
- Wie herausfinden? → Robustness Analysis
  - Grobentwurf „zum Leben erwecken“:
    - Anwendung als Maschine betrachten
    - Mit der Maschine konkrete Szenarien durchspielen
      - Sequenz-/Kommunikations-Diagramme zeichnen
    - Welche Szenarien?
      - Siehe: Anwendungsfälle (Use Cases)

# EXKURS – DAS RUP ANALYSIS PROFILE:



- Es gibt auch noch ein Profil für die Analyse im RUP  
→ Erweiterung der UML über Stereotypen, ...
- Elemente einer Anwendung können in folgende Elemente untergliedert werden:
  - «Actor» – Akteure (siehe UseCase-Diagramm)
  - «Boundary» – Schnittstellen-Komponenten (z.B.: UI)
  - «Control» – Programmlogik / Kontrollfluß  
(z.B.: Geschäftslogik / Veränd. von Entity-Objekten)
  - «Entity» – Datenelemente  
(z.B. Klassen im Fach-/Domänenmodell)

- Mögliche Darstellung in UML:



# ROBUSTNESS ANALYSIS DURCHFÜHREN



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Vorgehensweise:

1. Zerlegung der Anwendung in

- Actor-Objekte (Benutzer, Externe Systeme)
- Boundary-Objekte (Schnittstellen-Komponenten)
- Control-Objekte (Programmfluss)
- Entity-Objekte (Fachliche Daten)

}  $\hat{=}$  Architektur

2. Für jeden Anwendungsfall durchspielen:

- Standardablauf (= 1 Szenario)
- Alternative Szenarien (Alternative Schritte)

3. Bei Fehlern/Unstimmigkeiten:

- Anforderungen korrigieren und/oder
- Grobentwurf korrigieren/verfeinern
- Zurück zu 2.

# ROBUSTNESS ANALYSIS DURCHFÜHREN



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Durchspielen
  - BSP Gemeinsam an der Tafel / Dokumentenkamera
    - FMC-Entwurf
    - Szenarien
    - Sequenz-/Kommunikations-Diagramme
  - Siehe dazu PDF auf der Homepage

# ROBUSTNESS ANALYSIS – WOZU?



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Vorteile
  - + frühzeitig Fehler erkennen
  - + besseres Verständnis der Architektur
  - + Risiko verringern
    - (Architektur wird auch gern für weitere Planung der Arbeitspakete verwendet, ...)
- Nachteil
  - (zunächst) höherer Aufwand
    - Beispiel für eine Art von Architekturenreview
    - Immer empfohlen!



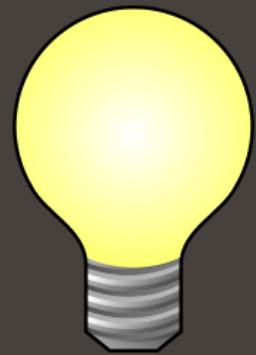
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

07

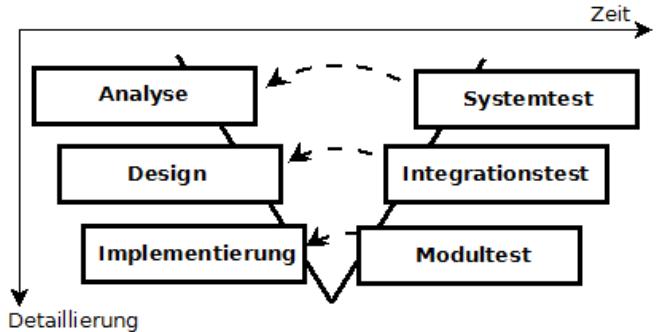
## Wie geht's dann weiter?

Ziel:

Was passiert im Anschluß?

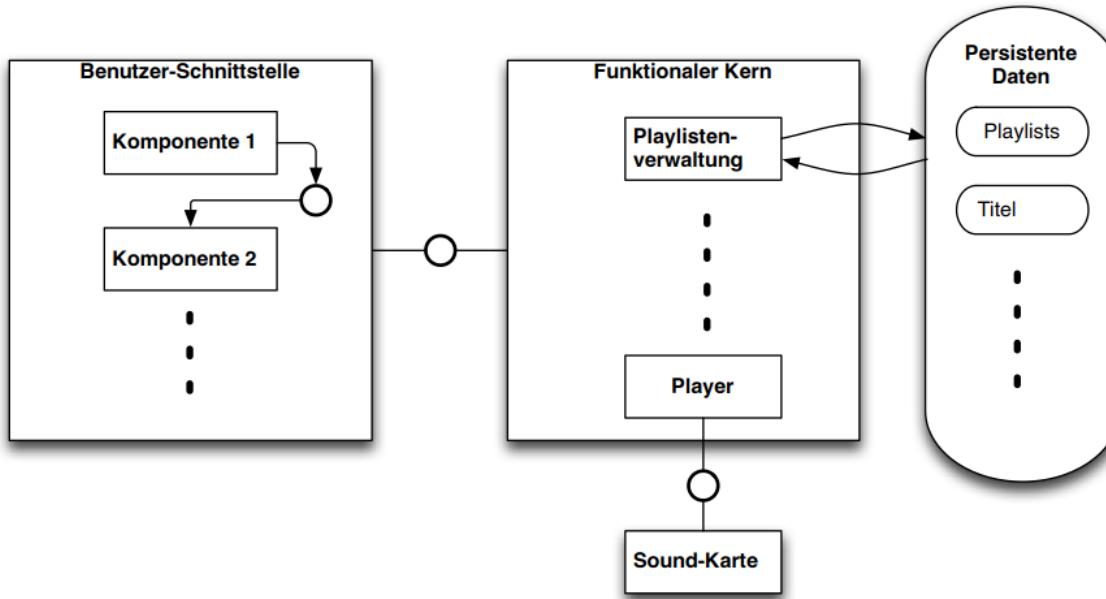


# TESTVORBEREITUNG



- Weiterer Bestandteil der Qualitätssicherung:  
→ Testvorbereitung
- Vorbereitung des Integrationstests (Details später)
  - Input: Grobentwurf
    - Welche Komponenten gibt es?
    - Wie kommunizieren diese miteinander?
  - Output:
    - Grundlegender Ansatz (bottom-up, top-down, . . . )
    - Top-Level-Testfälle  
(Zusammenstecken der letzten Verbindungen.)

# VORSCHAU: FEINENTWURF



## → Feinentwurf/Implementierung

- Wie werden FMC-Komponenten dann im Detail umgesetzt?
- Mehrere Möglichkeiten:
  - FMC-Komponente → Klasse(n), Attribut(e), Methode(n), . . .
- UND:
  - Spätestens jetzt entsteht das Domänenmodell

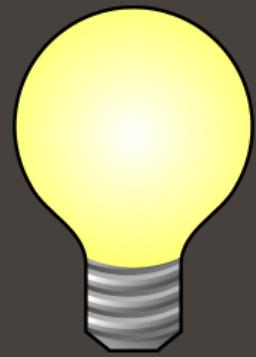


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# 08

# Fazit

Ziel:  
Was haben wir damit gewonnen?



# WAS HABEN WIR GELERNT?



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Entwurfsphase teilt sich in zwei Unterphasen:
  - Grobentwurf
  - Feinentwurf
- Grobentwurf
  - Festlegen der Architektur
  - FMC-Notation für Blockdiagramme
  - 3-Schichten-Architektur
    - Architekturmuster → Siehe spätere Vorlesung über Muster
    - Wahrscheinlich populärste Architektur für PC-Softwaresysteme
- Grobentwurf überprüfen
  - Z.B.: Robustness Analysis
- Ab hier kann man auch den Integrationstest vorbereiten

# WEITERFÜHRENDE LITERATUR



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Object-Oriented Analysis and Design:
  - Kleuker: Grundkurs Software-Engineering mit UML [<http://dx.doi.org/10.1007/978-3-8348-9843-2>]
  - Zuser et al: Software-Engineering mit UML und dem Unified Process [BF 500 92]
  - C. Larman: Applying UML and Patterns [30 BF 500 78]
- FMC: Compositional structures and block diagrams
  - P. Tabeling: Home of Fundamental Modeling Concepts [<http://www.fmc-modeling.org/>]
  - Knoepfel et al: Fundamental modeling concepts [30 BF 000 158]



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

AUF GEHT'S!!

SELBER MACHEN UND LERNEN!!



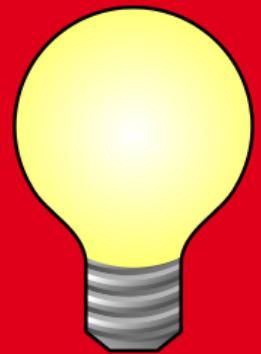


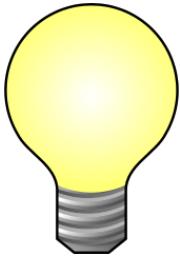
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

21.01.2021

# Programmieren im Grossen IV

Feinentwurf





Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# AGENDA

Einführung ins Thema

Vorgehensweise

Beispiel “Playlistenverwaltung”

Entwurfsmuster

Fazit

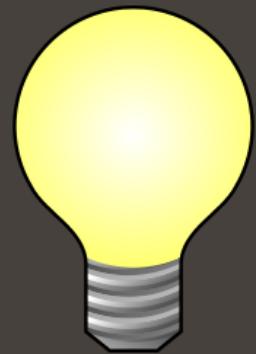


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

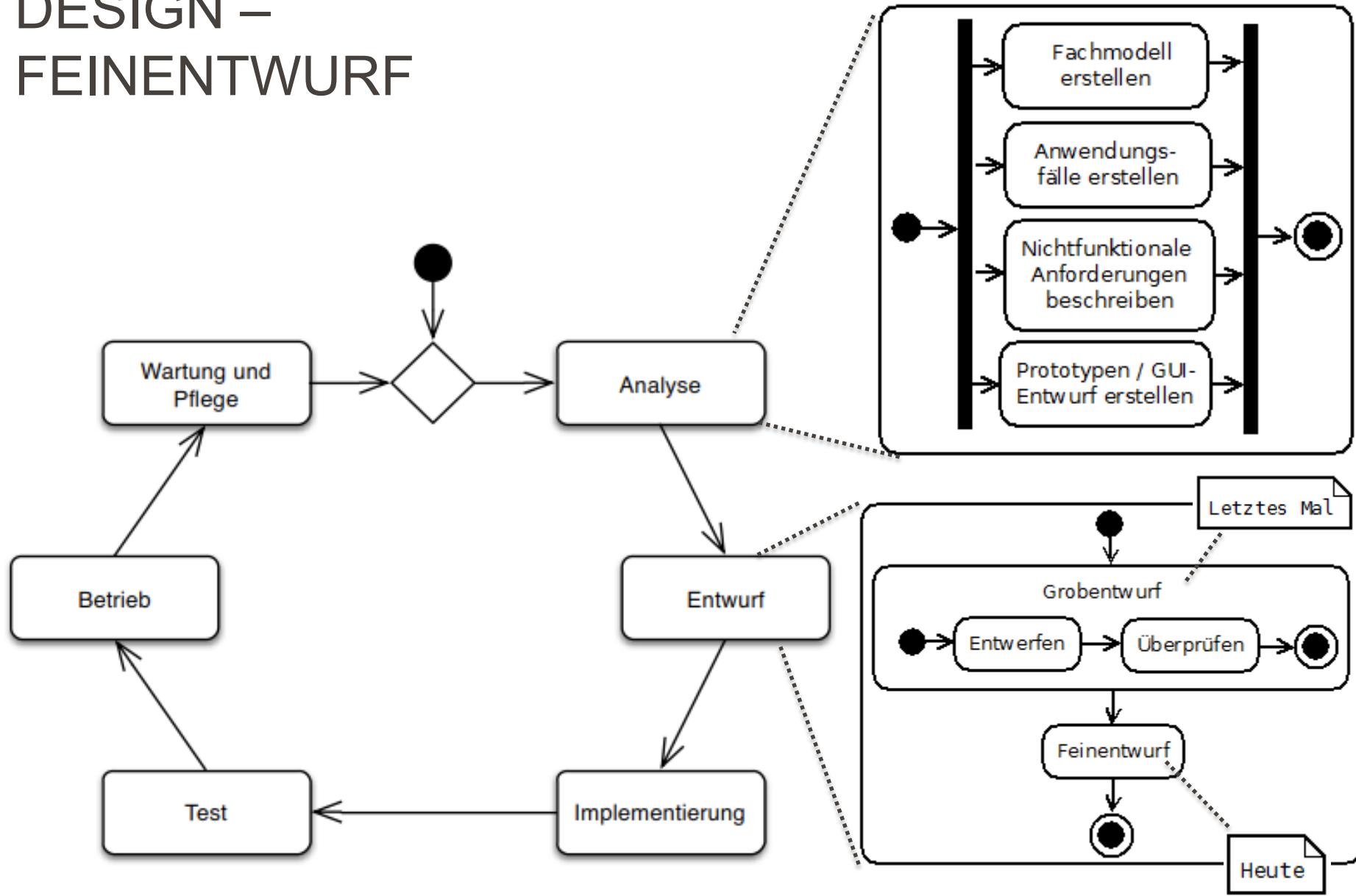
# 01

## EINFÜHRUNG INS THEMA

Ziel:  
Die Eckpunkte des Themas kennenlernen



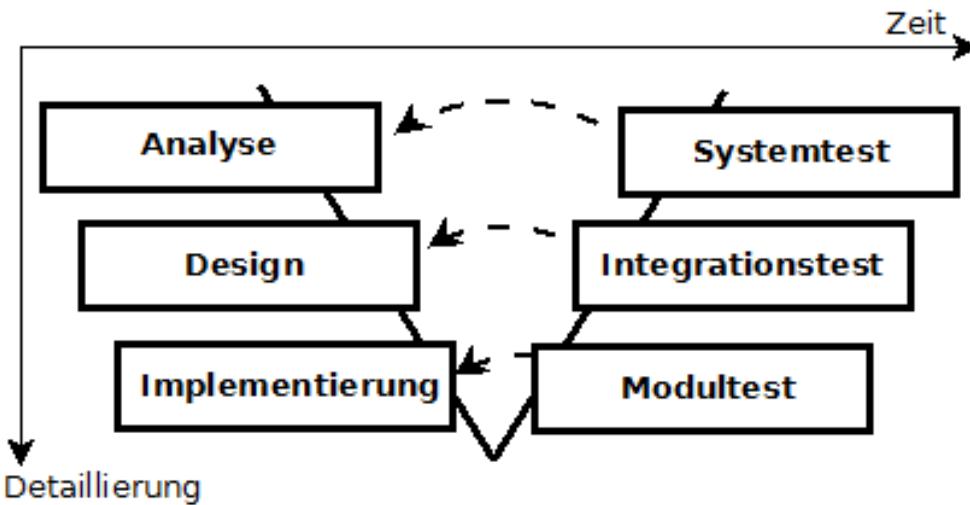
# DESIGN – FEINENTWURF





# WORUM GEHT'S?

- Wir brauchen ein Vorgehensmodell
  - Verschiedene Phasen:



- Heute besprechen wir:
  - Design: Feinentwurf

# ENTWURF (DESIGN) – WORUM GEHT'S?



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Grundlegende Frage
  - Wie soll das zu bauende System sein?
- Entwurf (Design)
  - Tätigkeiten
    - Grobentwurf (Architektur) → letztes Mal
    - **Feinentwurf** → **heute**
  - Sprechweise im (R)UP:
    - „Analysis and Design“
  - Sprechweise im V-Modell:
    - „Design“

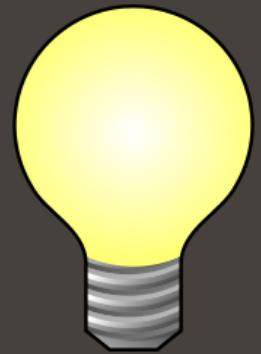


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## 02

# Vorgehensweise

Ziel:  
Prinzipielle Vorgehensweise kennenlernen



# FEINENTWURF – VORGEHENSWEISE



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Input:
  - Grobentwurf (inkl. GUI-Entwurf)
  - Resultate der Anforderungsanalyse
  - Erfahrung (typische, erprobte Lösungen)
- Tätigkeit: Grobentwurf (schrittweise) verfeinern
- Output:
  - Programmentwurf in Form von Klassendiagrammen
  - + Spezifikation (zumindest grob/teilweise)
  - + ggfs. Pseudocode
  - + ggfs. Sequenz-, Aktivitäts-, . . . -Diagramme
  - + ggfs. Datenbank-Tabellen/-Constraints
  - + . . .

⇒ Ziel: Programmierer können direkt loslegen

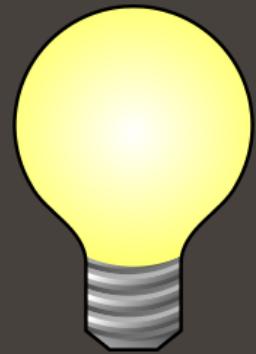


03

## Beispiel „Playlistenverwaltung“

Ziel:

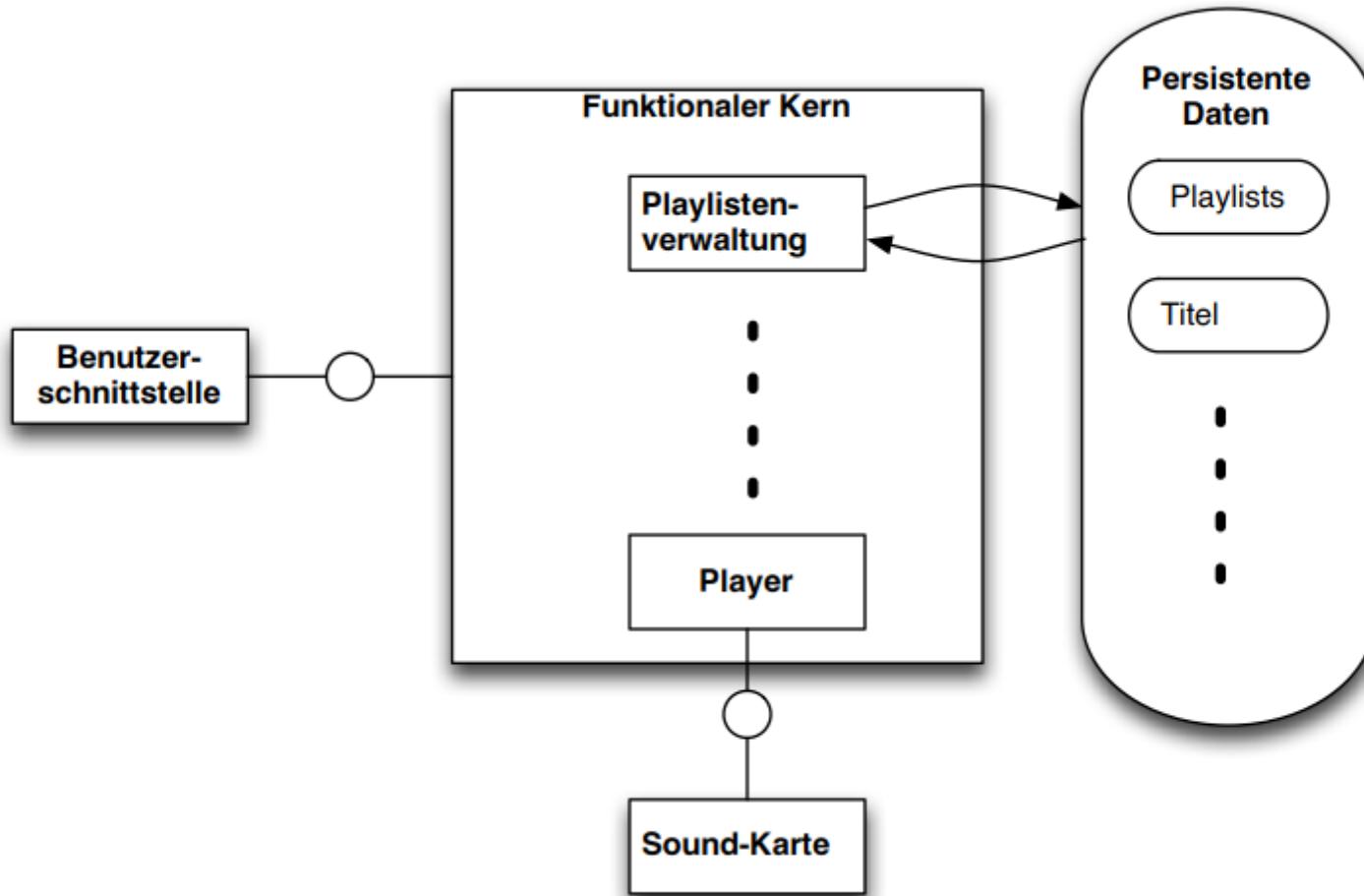
Ein konkretes Beispiel





# BEISPIEL GROBENTWURF

- BSP: Grobentwurf MP3-Player mit FMC:



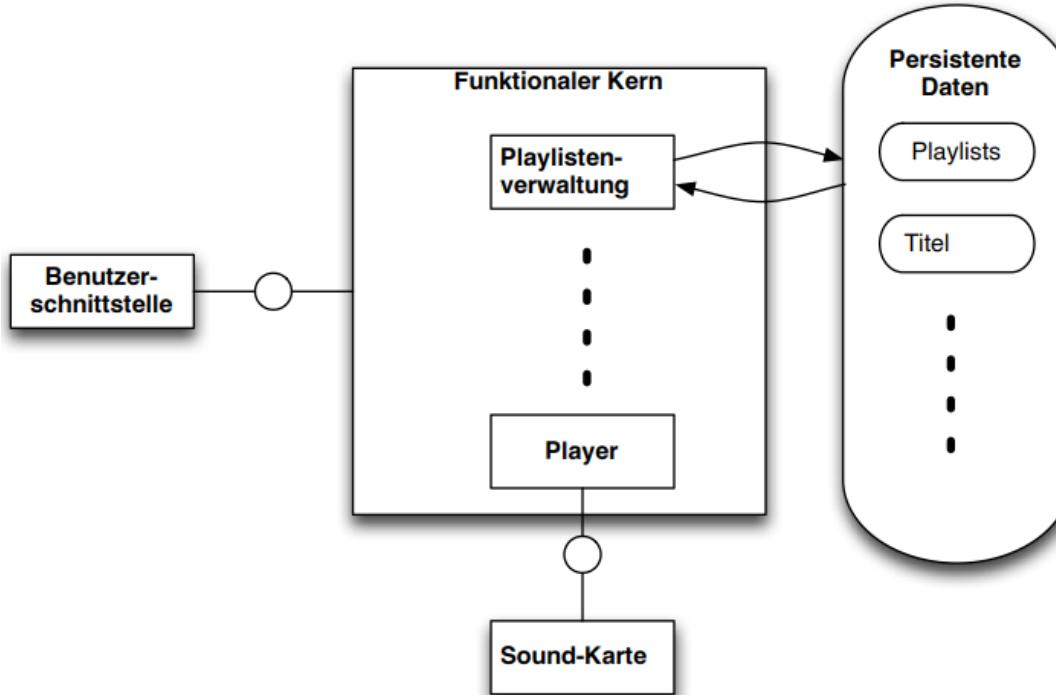
→ Mit Robustness Analysis überprüft

# HINWEIS: WIE DEN GROBENTWURF UMSETZEN?



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Wie Grobentwurf mit FMC jetzt in Feinentwurf umsetzen?



- Mehrere Möglichkeiten für jede FMC-Komponente:
- FMC-Komponente → Klasse(n), Objekt(e)
  - FMC-Komponente → Attribut (Variable) einer Klasse
  - FMC-Komponente → Methode(n) (Algorithmus), . . .

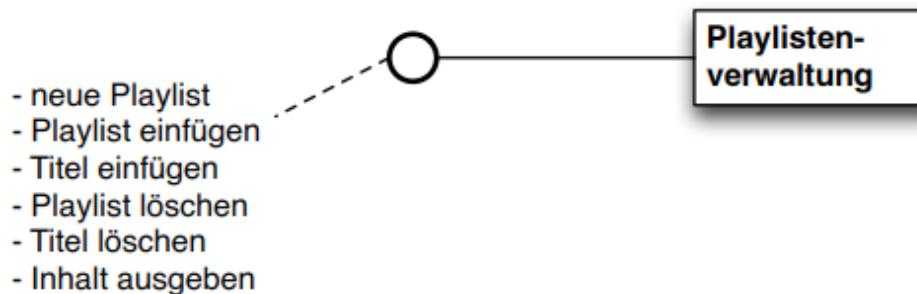
# BEISPIEL FEINENTWURF „PLAYLISTENVERWALTUNG“



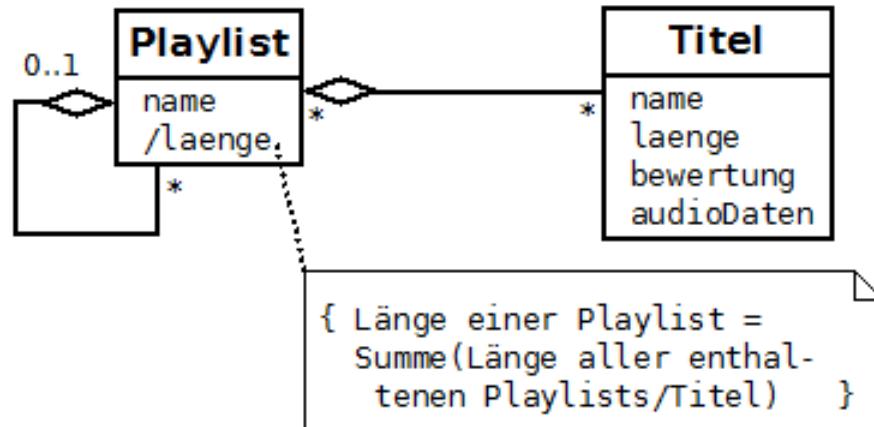
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## INPUT:

- Passender Auszug aus dem Grobentwurf  
→ Grobe Schnittstellenbeschreibung für „Playlistenverw.“:



- Passender Auszug aus dem Fachmodell zur „Playlistenverw.“



- ggfs. weitere passende Auszüge aus den Anforderungen

# BEISPIEL FEINENTWURF „PLAYLISTENVERWALTUNG“



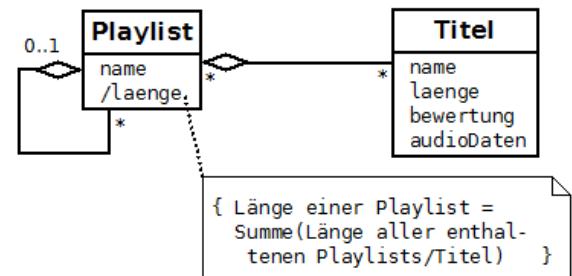
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## OUTPUT: Detailliertere Schnittstelle zur „Playlistenverwaltung“

Was benötigen wir?

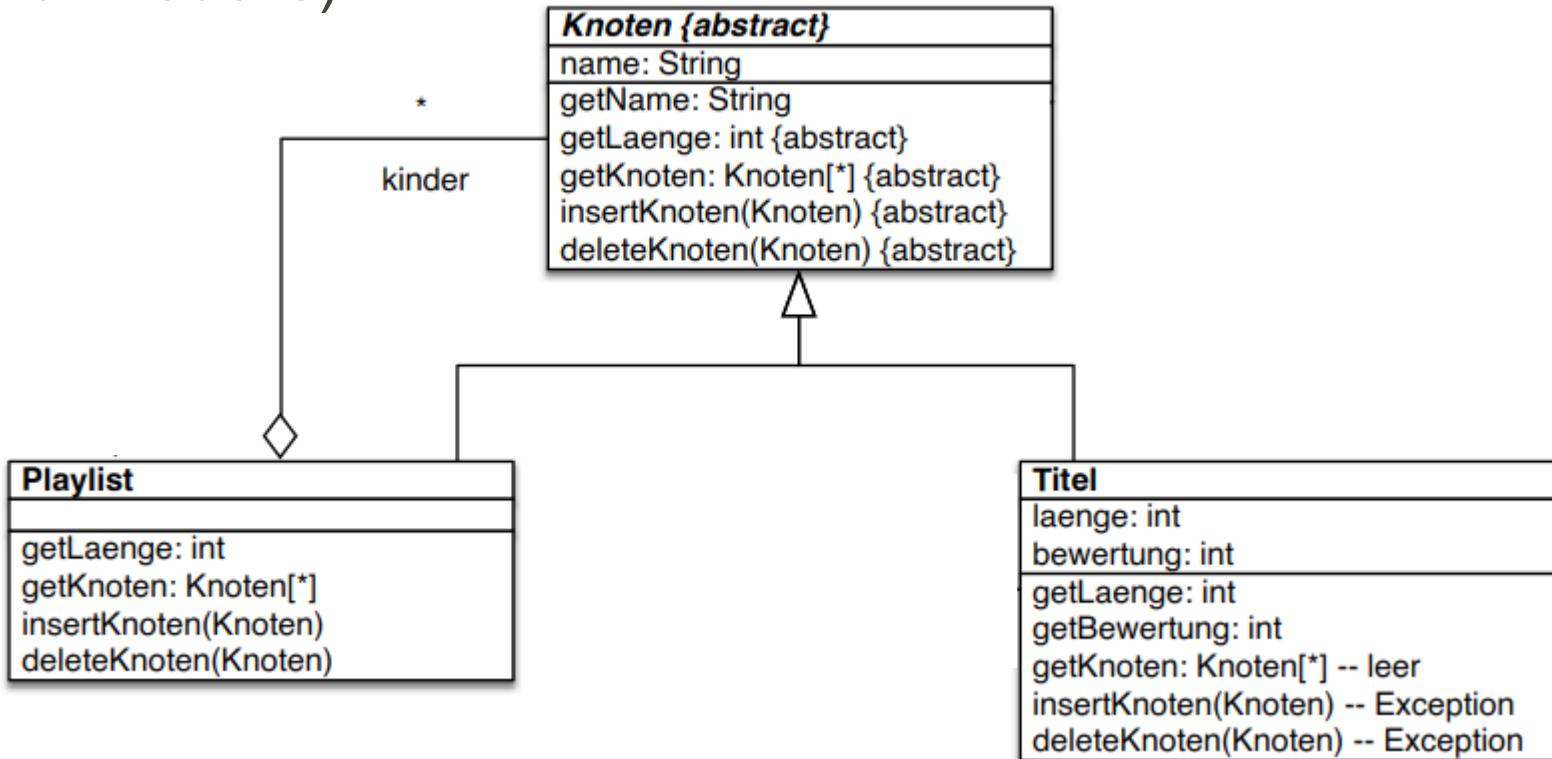
- Baum-Struktur mit verschiedenartigen Knoten
  - Playlist
  - Titel
- Navigation, zumindest: Von oben nach unten
- Verwaltung, zumindest:
  - Knoten einfügen
  - Knoten löschen
- Delegieren von Aufgaben nach unten
  - Obere Knoten (Elternknoten) erledigen Aufgaben mit Hilfe der untergeordneten Knoten
  - Letzte Knoten (Blätter) müssen Aufgaben alleine erledigen

# BEISPIEL FEINENTWURF „PLAYLISTENVERWALTUNG“



**OUTPUT:** Detailliertere Schnittstelle zur „Playlistenverwaltung“

→ (Unvollständiger) Feinentwurf des Domänenmodells (Datenmodells):

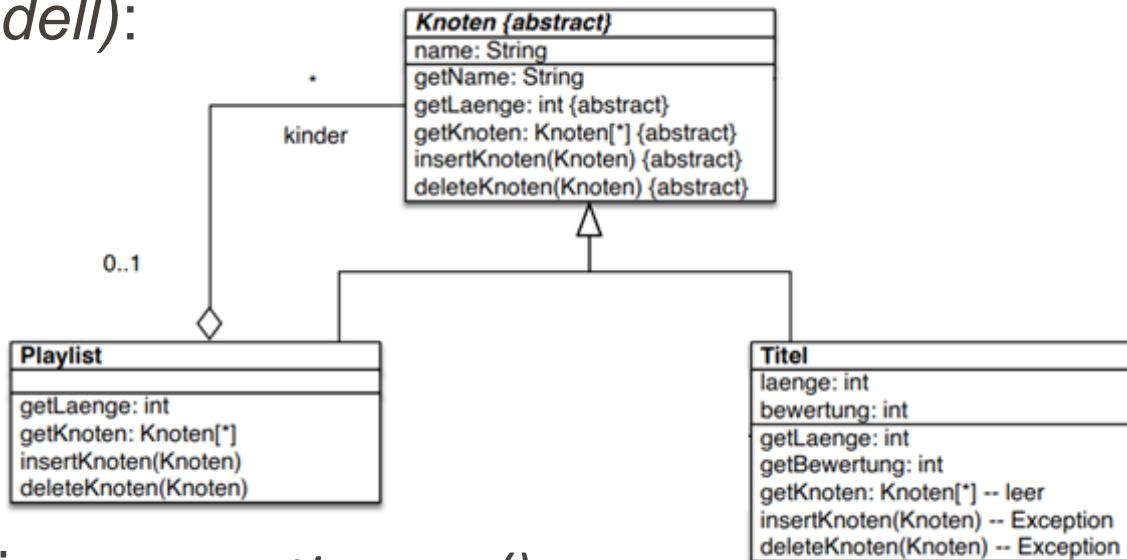


# BEISPIEL FEINENTWURF „PLAYLISTENVERWALTUNG“



**OUTPUT:** Detailliertere Schnittstelle zur „Playlistenverwaltung“

- (Unvollständiger) Feinentwurf des *Domänenmodells (Datenmodell)*:



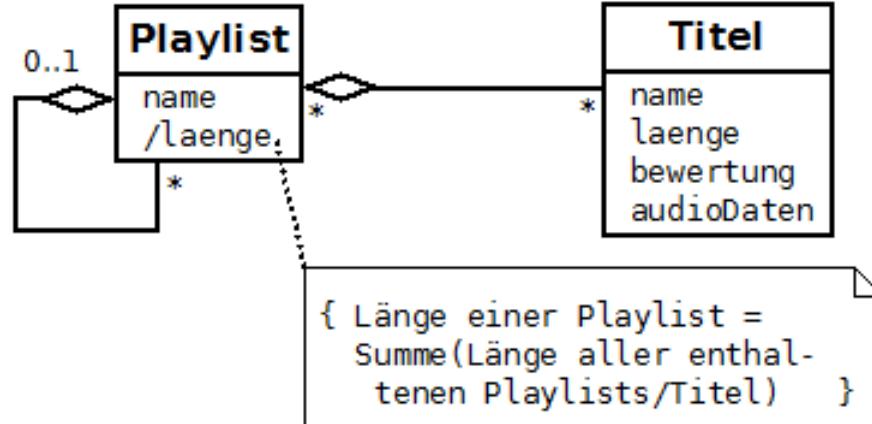
- Spezifikation von `getLaenge()`:
  - `Titel.getLaenge()`: gibt Länge des Titels zurück
  - `Playlist.getLaenge()`: gibt Summe von `getLaenge()` aller Unterknoten zurück (leere Summe=0)
- . . .

(Siehe delegieren von  
Aufgaben zwei Folien vorher)

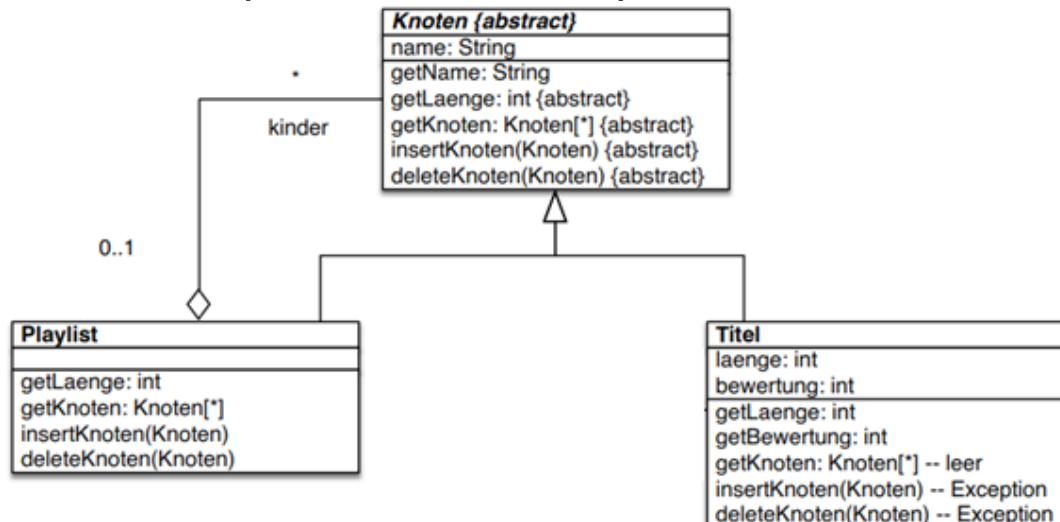
# NOCHMALS ZUM VERGLEICH



- *Fachmodell aus der Analyse (rein aus Anforderungen):*



- *Domänenmodell (Datenmodell) im Feinentwurf:*



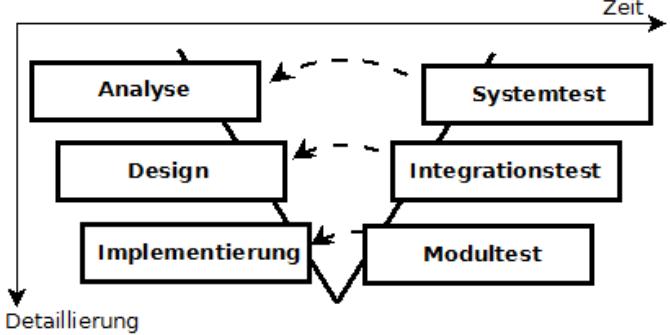
# WEITERE BEMERKUNGEN ZUM FEINENTWURF



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Wichtige Frage: Wann hört man auf?
    - D.h.: Was modelliert man? ↔ Was programmiert man?
  - Dazu gibt es leider keine einfache Antwort
    - Auch etwas eine Idealvorstellung der durchgängigen Modellierung
    - Hängt davon ab wie durchgängig die Werkzeugkette ist
      - Wenn enge Verzahnung vorhanden, kann man sehr detailliert durchmodellieren
      - Ansonsten Gefahr, dass Modell und Code schnell auseinanderdriften
    - In vielen Projekten: Eher rudimentäre Modellierung der Eckdaten → Rest wird programmiert
- Jedes Projekt muss hier selbst die richtige Dosis finden

# TESTVORBEREITUNG IM FEINENTWURF



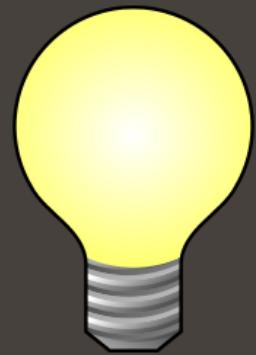
- Vorbereitung des Integrationstests:
  - Input: Feinentwurf
    - Welche Klassen gibt es?
    - Wie kommunizieren diese miteinander?
  - Output:
    - Ansatz (bottom-up, top-down, . . . )
    - Mindestens ein Testfall für jede Verbindung
- Vorbereitung des Modultests:
  - Black-Box-Testfälle für die Module (Klassen)



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## 04 Entwurfsmuster

Ziel:  
Lösungen für wiederkehrende Probleme kennen



# WIEDERKEHRENDE AUFGABEN- STELLUNGEN IM FEINENTWURF



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## Aus unserem vorherigen Beispiel:

Wir benötigen:

- Baum-Struktur mit verschiedenartigen Knoten
- Navigation, zumindest: von oben nach unten
- Verwaltung, zumindest: Knoten einfügen, löschen
- Delegieren von Aufgaben nach unten

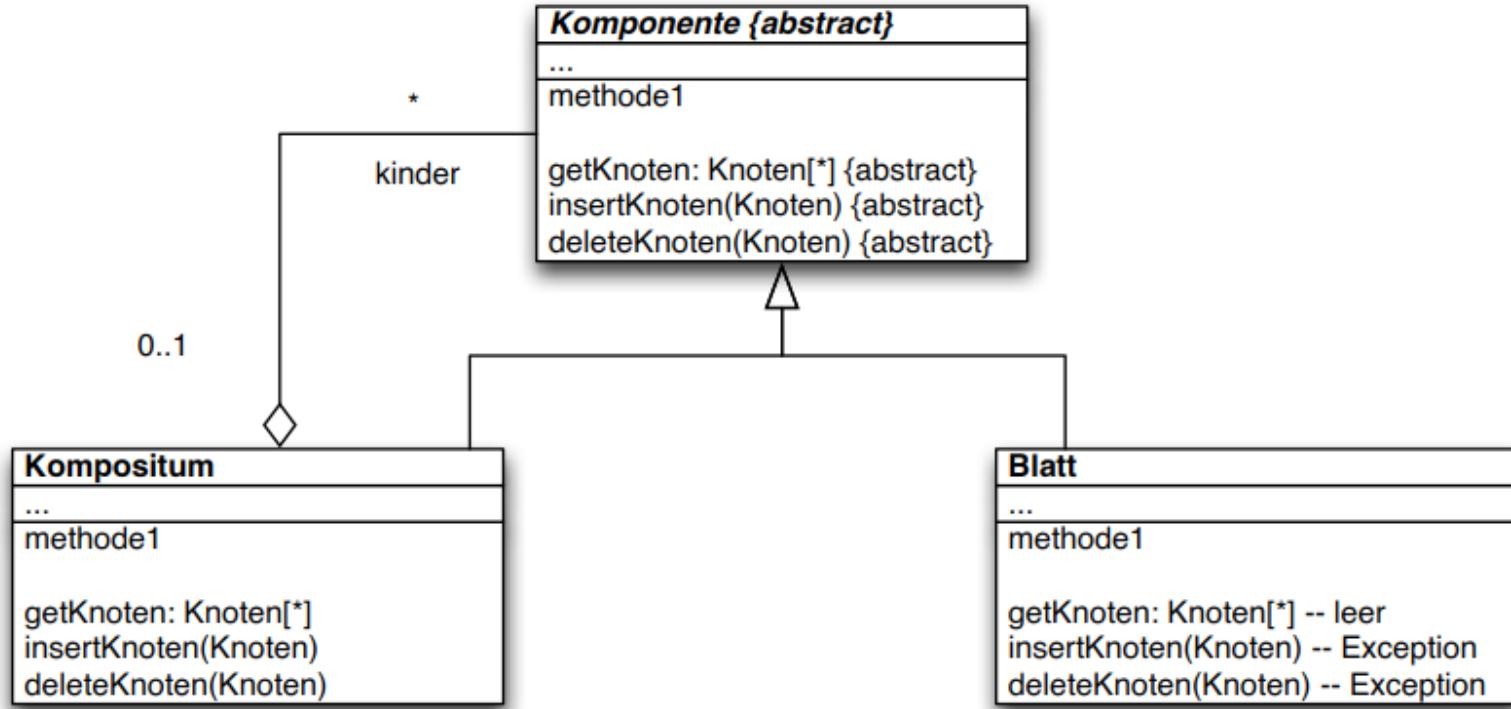
→ Solche oder sehr ähnliche Aufgaben werden in vielen Situationen gebraucht – Z.B.:

- Dateisystem,
- Evtl.: Unsere Programmieraufgabe im 2. Semester

# ZUGEHÖRIGE, TYPISCHE, ERPROBTE LÖSUNG:



- Wir setzen das im allgemeinen Fall so um:



*Kompositum.methode1*: delegiert Aufgabe an Unterknoten

→ Diese immer wiederkehrende Aufgabenstellung + diese typische, erprobte Lösung (+ zusätzliche Beschreibungen)

wird *Entwurfsmuster „Kompositum“* genannt

# DEFINITION & ARTEN VON MUSTERN



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Muster (in der Software-Entwicklung)  
= typische, erprobte Lösung für ein immer wiederkehrendes Problem in einem bestimmten Kontext
- Analysemuster (analysis pattern)  
= typische, erprobte Lösung für ein immer wiederkehrendes Problem beim Erstellen von Fachmodellen, etc.
- Architekturmuster (architectural pattern)  
= typische, erprobte Lösung für ein immer wiederkehrendes Problem beim Grobentwurf
- **Entwurfsmuster (design pattern)**  
= typische, erprobte Lösung für ein immer wiederkehrendes Problem beim Feinentwurf
- ...

# MUSTER-BESCHREIBUNGEN



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Übliche Form für die Beschreibung eines Entwurfsmusters:
  1. Name + andere Namen
  2. Beispiel
  3. Kontext
  4. Problem
  5. Lösung
  6. Vor- und Nachteile
  7. Verwandte Muster
- Musterkataloge
  - Sammlung von Mustern + deren Zusammenwirken  
(Unterstützend, Widersprechend, Neutral)
  - BSP: Entwurfsmuster von Gamma et al. (→ Literaturverzeichnis)  
„Gang of Four (GoF)-Muster“

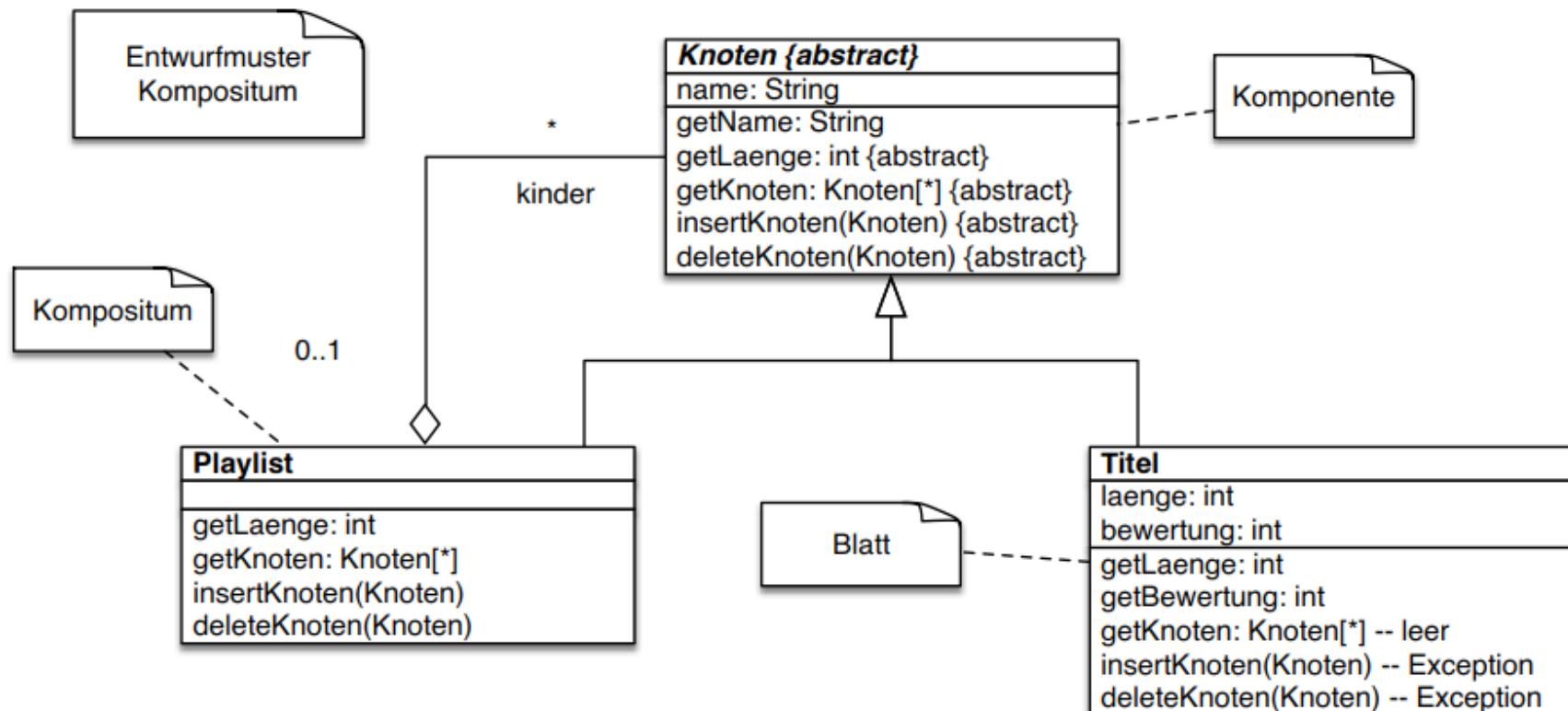
# KENNZEICHNUNG EINES VERWENDETEN ENTWURFSMUSTERS



Hochschule RheinMain  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## BSP: „Kompositum“-Muster in unserem Domänenmodell

- Kennzeichnung durch Kommentare:



- BEM: Entwurfsmuster und die Rollen der beteiligten Klassen kann man in UML auch mit dem Modell-Element „Kollaboration“ kennzeichnen.<sup>25</sup>

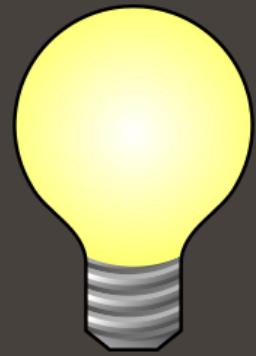


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# 06

# Fazit

Ziel:  
Was haben wir damit gewonnen?





# WAS HABEN WIR GELERNT?

- Feinentwurf
  - Detaillierung des Grobentwurfs
  - Besonders wichtig: Das Domänenmodell
- Entwurfsmuster
  - Bieten bewährte Lösungen für wiederkehrende Probleme
  - Heute: Kompositum

→ Nächste Woche: Die Musteridee allgemein

# WEITERFÜHRENDE LITERATUR



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Object-Oriented Analysis and Design:
  - Kleuker: Grundkurs Software-Engineering mit UML [<http://dx.doi.org/10.1007/978-3-8348-9843-2>]
  - Zuser et al: Software-Engineering mit UML und dem Unified Process [BF 500 92]
  - C. Larman: Applying UML and Patterns [30 BF 500 78]
- Entwurfsmuster:
  - Gamma et al.: Entwurfsmuster [30 BF 500 40].
    - Der Klassiker aus den 90-igern. Ganz gut zum Nachschlagen.
    - Trocken zu lesen.
  - Freeman et al: Entwurfsmuster von Kopf bis Fuß [30 BF 500 98]
    - Schöne, umfassende Einführung. Witzig geschrieben.



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

AUF GEHT'S!!

SELBER MACHEN UND LERNEN!!



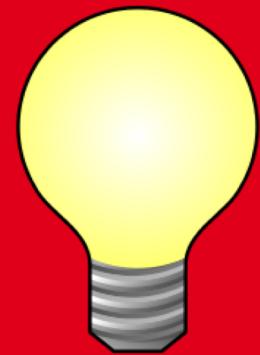


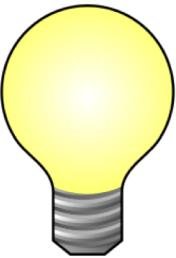
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

28.01.2021

# Programmieren im Großen IV

Muster





# AGENDA

Einführung ins Thema

Was sind Muster?

Entwurfsmuster

(Besucher, Beobachter, Liste gängiger Muster)

Architekturmuster

(3-Schichten, MVC, Liste gängige Muster)

Allgemeine Prinzipien

Fazit

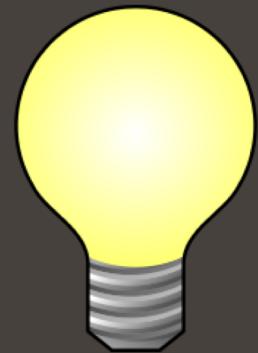


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# 01

## EINFÜHRUNG INS THEMA

Ziel:  
Die Eckpunkte des Themas kennenlernen



# WIEDERKEHRENDE PROBLEME



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

Oft gibt es immer wiederkehrende Probleme

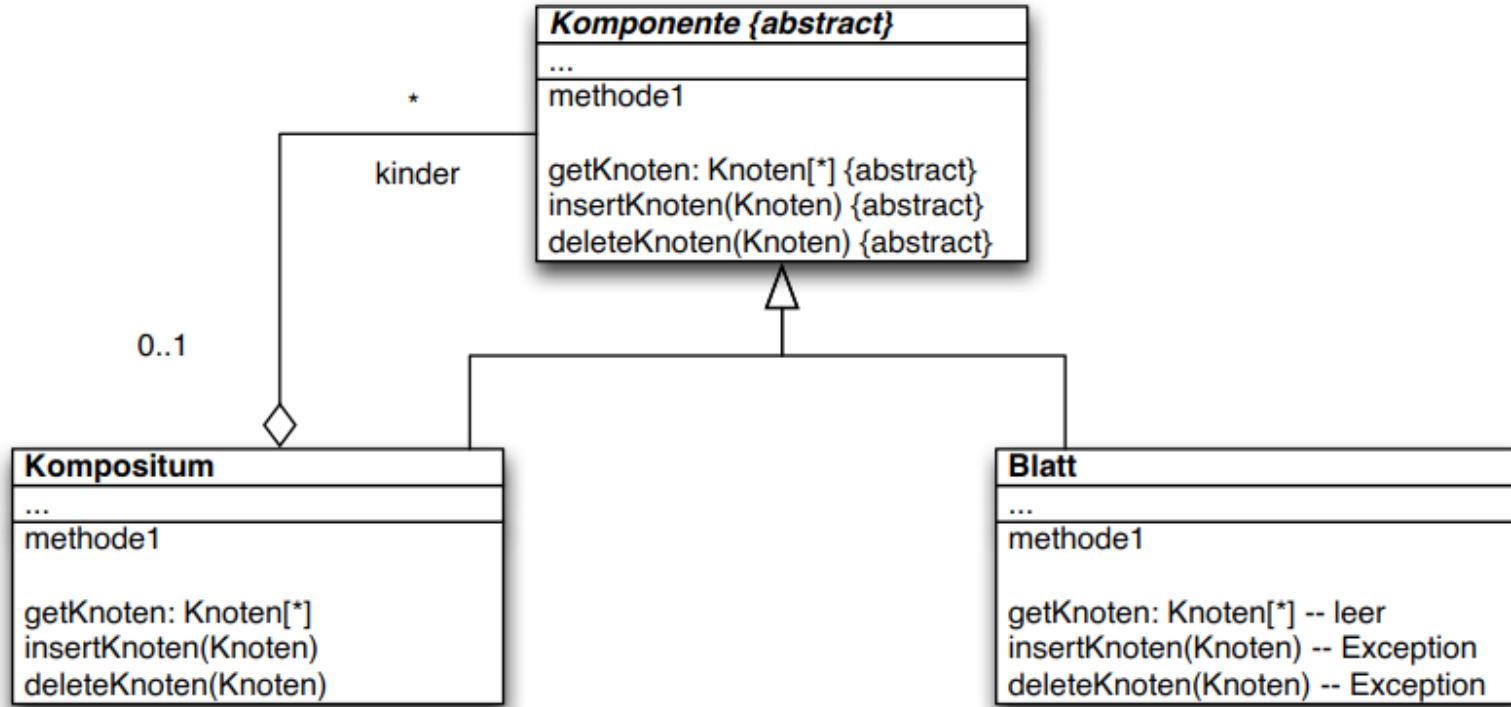
Aus dem Beispiel der letzten Vorlesung:

- Baum-Struktur mit verschiedenartigen Knoten
  - Navigation, zumindest: von oben nach unten
  - Verwaltung, zumindest: Knoten einfügen, löschen
  - Delegieren von Aufgaben nach unten
- Solche oder sehr ähnliche Aufgaben werden in vielen Situationen gebraucht - Z.B.:
- Dateisystem,
  - Evtl.: Unsere Programmieraufgabe im 2. Semester
- Hier helfen Muster!

# ZUGEHÖRIGE, TYPISCHE, ERPROBTE LÖSUNG:



- Wir setzen das im allgemeinen Fall so um:



*Kompositum.methode1*: delegiert Aufgabe an Unterknoten

→ Diese immer wiederkehrende Aufgabenstellung + diese typische, erprobte Lösung (+ zusätzliche Beschreibungen)

wird *Entwurfsmuster „Kompositum“* genannt



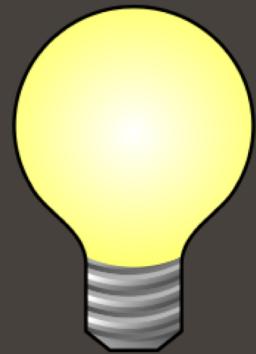
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

02

## Was sind Muster?

Ziel:

Nochmals den Mustergedanken genauer beleuchten



# DEFINITION MUSTER (PATTERNS)



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Muster (in der Software-Entwicklung)  
= typische, erprobte Lösung für ein immer wiederkehrendes Problem in einem bestimmten Kontext
- Konzept wurde von Christopher Alexander (Bauarchitektur) entwickelt
  - Bsp: Window Place: Beschreibt wo gute Orte für Fenster sind
    - Varianten: Window seat, Bay window, Big windows
    - Wirkung: Erzeugen Plätze für unser Wohlbefinden
  - Erlaubt Kommunikation von Expertenwissen
    - Bietet bewährte Lösung gerade auch für Einsteiger
    - Vereinfacht die Kommunikation zwischen Experten

# DEFINITION MUSTER (PATTERNS)



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## Grundideen:

- Problem bestehend aus Konflikten oder Kräften
  - Musterinvarianten Teil → Kern des Musters
  - Mustervarianter Teil → Veränderlicher Teil zur Einbettung in Umgebung des Musters
  - Konsequenzen → Neue Kräfte
  - Muster kommunizieren miteinander über Konsequenzen/Kräfte:  
Spektrum: Unterstützend ↔ Konflikt verursachend
- Diese Grundideen bilden auch Grundstruktur zur Musterbeschreib.
- Oft in Musterschablone gepackt

# MUSTER-BESCHREIBUNGEN



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Übliche Form für eine Musterbeschreibung:
  1. Name + andere Namen
  2. Beispiel (konkretes Bsp.)
  3. Kontext (genauerer Zusammenhang)
  4. Problem (abstrahiert)
  5. Lösung (abstrahiert)
  6. Vor- und Nachteile
  7. Verwandte Muster / Kommunikation zw. Mustern  
(Unterstützung ↔ Konflikte)
- Musterkataloge:
  - Sammlung von Mustern + deren Zusammenwirken  
(Unterstützend, Widersprechend, Neutral)
  - z.B. Entwurfsmuster von Gamma et al (s. Literaturverzeichnis)

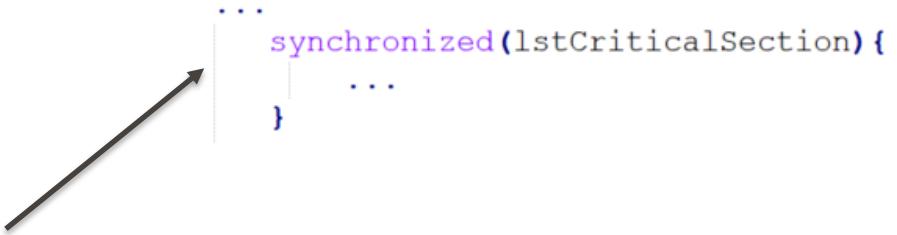
# ARTEN VON MUSTERN



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Anforderungsmuster (requirement patterns)  
= adressieren Probleme bei der Spezifikation von Anforderungen
- Analysemuster (analysis pattern)  
= adressieren Probleme beim Erstellen von Fachmodellen, etc.
- Architekturmuster (architectural patten)  
= adressieren Probleme bei der Architektur (Grobentwurf)
- Entwurfsmuster (design pattern)  
= adressieren Probleme beim Feinentwurf (Detailed Design)

# ARTEN VON MUSTERN



- **Idiome**
  - = Beschreiben gewisse typische „Redewendungen“ in Programmiersprachen, die zu Lösung spezifischer Probleme verwendet werden
- **Prozessmuster (Best Practices)**
  - = Adressieren Probleme mit Softwareentwicklungsprozessen
    - Auch Best Practices genannt → Agile Methoden bauen darauf
    - BEM: Ward Cunningham (XP) & Alistair Cockburn (Crystal) haben Mustergedanken in das SE getragen (schon vor GoF)
- **Allgemeine Prinzipien / Heuristiken**
  - Sog. Daumenregeln (Z.B.: DRY-Prinzip, Defensive Progr., ...)
  - Z.B. GRASP (siehe später)
- **Antipatterns**
  - Beschreiben schlechte/ungeeignete Lösungen

# LEIDER KÖNNEN WIR NICHT ALLE MUSTERARTEN BEHANDELN



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Deshalb besprechen wir:
  - Entwurfsmuster (Design Patterns)
    - Besucher
    - Beobachter
    - (Method-Object-Pattern)
    - Liste gängiger Muster
  - Architekturmuster (Architectural Patterns)
    - 3-Schichten-Architektur
    - MVC-Muster
    - Liste gängiger Architekturmuster
  - Allgemeine Prinzipien
    - GRASP-Patterns

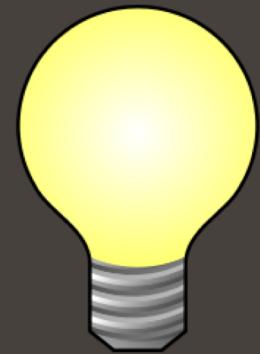


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## 03a

# Entwurfsmuster – Besucher (Visitor)

Ziel:  
Entwurfsmuster Besucher genauer kennenlernen

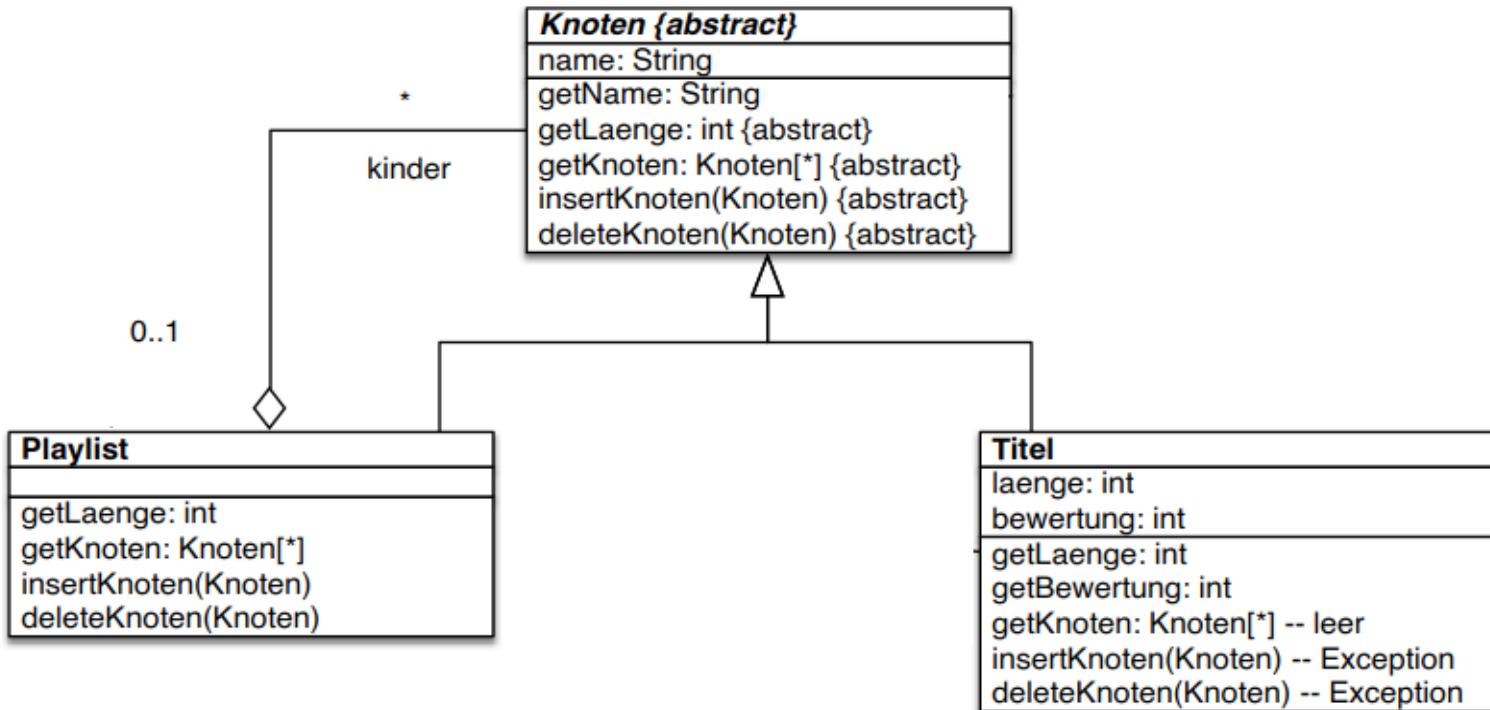


# BEISPIEL FEINENTWURF „PLAYLISTENVERWALTUNG“



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

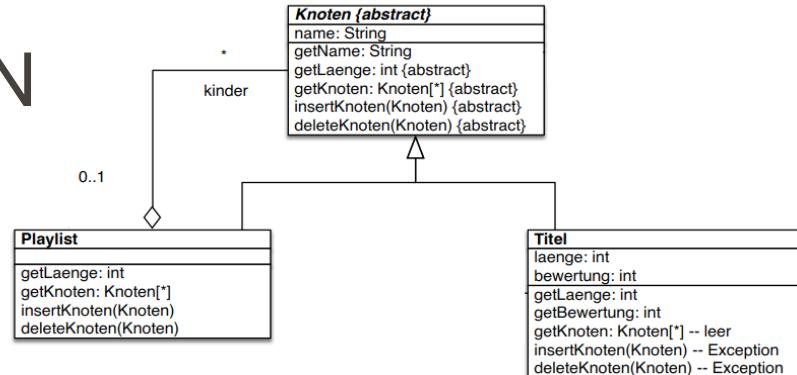
## Unser Domänenmodell:



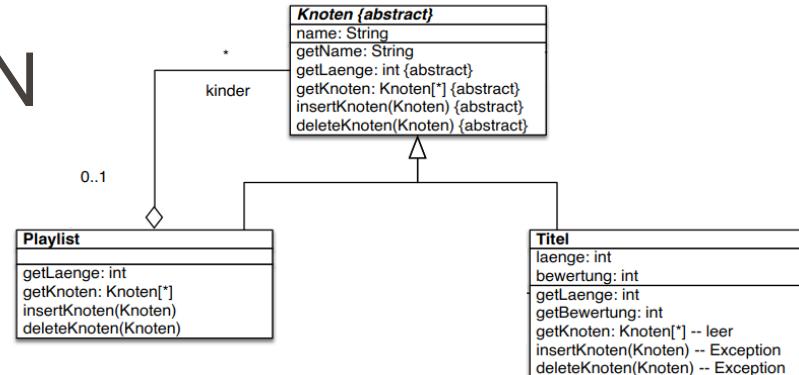
Jetzt: Erweiterung um die Berechnung der Bewertung für Playlist-Objekte

# AUSGANGSÜBERLEGUNGEN FÜR BESUCHER-MUSTER

- Mögliche Umsetzung:
  - neue Methode *Playlist.getBewertung*: gibt Durchschnitt von *getBewertung* der untergeordneten Knoten zurück
  - neue abstrakte Methode *Knoten.getBewertung*
- Mögliche Nachteile:
  - Wenn mehrere solcher Methoden hinzugefügt werden sollen und wenn es viele Unterklassen von Knoten gibt, dann:
    - Schnittstelle von Knoten wird aufgebläht
    - (fast) alle Klassen in der Vererbungshierarchie müssen jedes Mal angefasst werden

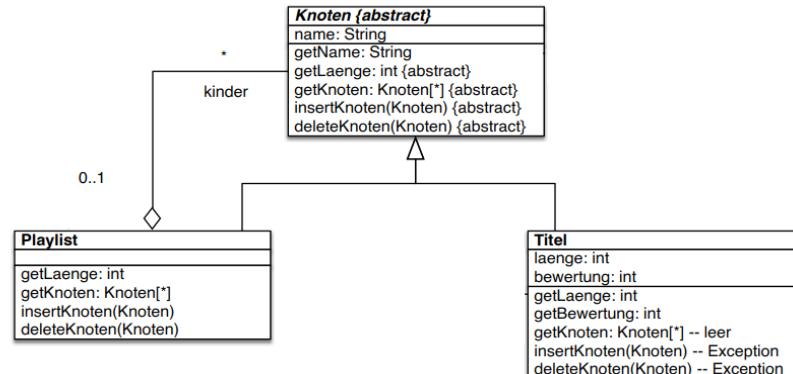


# AUSGANGSÜBERLEGUNGEN FÜR BESUCHER-MUSTER



- Bessere Idee:
  - Kapseln der neuen Funktionalität „Berechnung der Bewertung“ in eigener Klasse
- Konsequenzen der besseren Idee:
  - Wenn mehrere solcher Methoden hinzugefügt werden sollen und wenn es viele Unterklassen von *Knoten* gibt, dann:
    - Jede neue Funktionalität → eine neue Klasse
    - Schnittstelle von *Knoten* verändert sich nicht
    - Keine bestehende Klasse in der Vererbungshierarchie muss angefasst werden

# UMSETZUNG: 1. VERSUCH



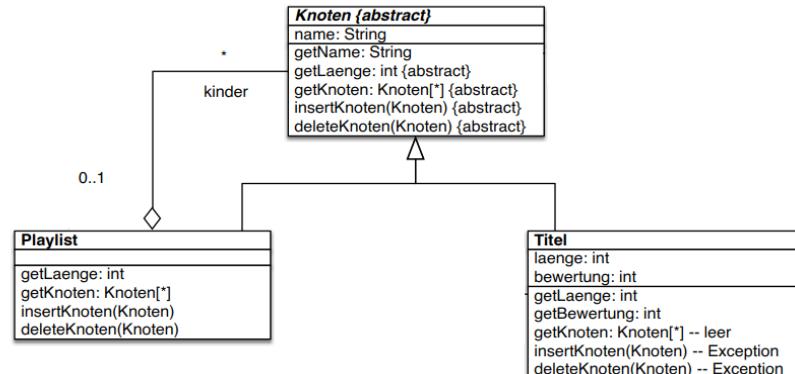
- Eigene Klasse zur Berechnung der Bewertung (fehlerhaft):

```
public class Bewertung {  
    public static double getBewertung(Titel t) {  
        return t.getBewertung();  
    }  
  
    public static double getBewertung(Playlist p) {  
        ...  
        for (Knoten k: p.getKnoten()) {  
            summe += getBewertung(k);  
        }  
        ...  
    }  
}
```

## Problem:

Lässt sich nicht kompilieren, da es keine passende Methode `getBewertung(Knoten k)` gibt!

# UMSETZUNG: 2. VERSUCH



- Eigene Klasse zur Berechnung der Bewertung (fehlerhaft):

```
public class Bewertung {
    public static double getBewertung(Titel t) {...}
    public static double getBewertung(Playlist p) {
        // wie getBewertung(Knoten k)
    }
    public static double getBewertung(Knoten k) {
        ...
        for (Knoten kind: k.getKnoten()) {
            summe += getBewertung(kind);
        ...
    }
```

## Problem:

*getBewertung(Titel)* und *getBewertung(Playlist)* werden niemals aufgerufen, sondern stets *getBewertung(Knoten)*.  
→ Polymorphieproblem

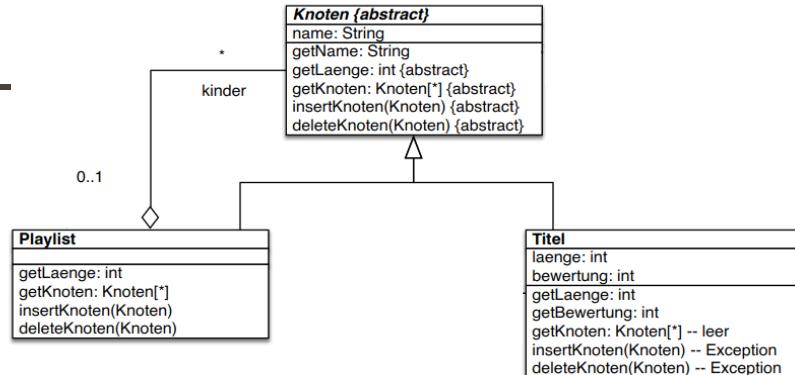
# UMSETZUNG: 2. VERSUCH – POLYMORPHIEPROBLEM

- Eigentliches Problem:

```
public class A {...}  
public class B extends A {...}
```

```
public class AndereKlasse{  
    public ... f(A a) {...}  
    public ... f(B b) {...}
```

```
...  
A a1 = new A(...);  
... = f(a1);  
A a2 = new B(...);  
... = f(a2);
```



**Warum greift der Polymorphismus hier nicht?**

Bereits zur Kompilierzeit wird anhand der Parameter entschieden, ob `f(A)` oder `f(B)` in den Bytecode eingesetzt wird.

# BEMERKUNG ZU POLYMORPHIE IN JAVA UND C++



- In Java und C++ funktioniert der Polymorphismus nur für Methoden in derselben Klasse / Klassenhierarchie

```
public class A { ... f(...) {...} }  
public class B extends A { ... f(...) {...} }  
...  
A a1 = new A(...);  
a1.f(...);  
A a2 = new B(...);  
a2.f(...);
```

The diagram illustrates polymorphism in Java or C++. It shows two classes, A and B, both defining a method f(). An object a1 of class A and an object a2 of class B are created. Arrows point from each object's call to its respective class's method definition. A dashed box encloses both definitions of f(), indicating they are visible to both objects.

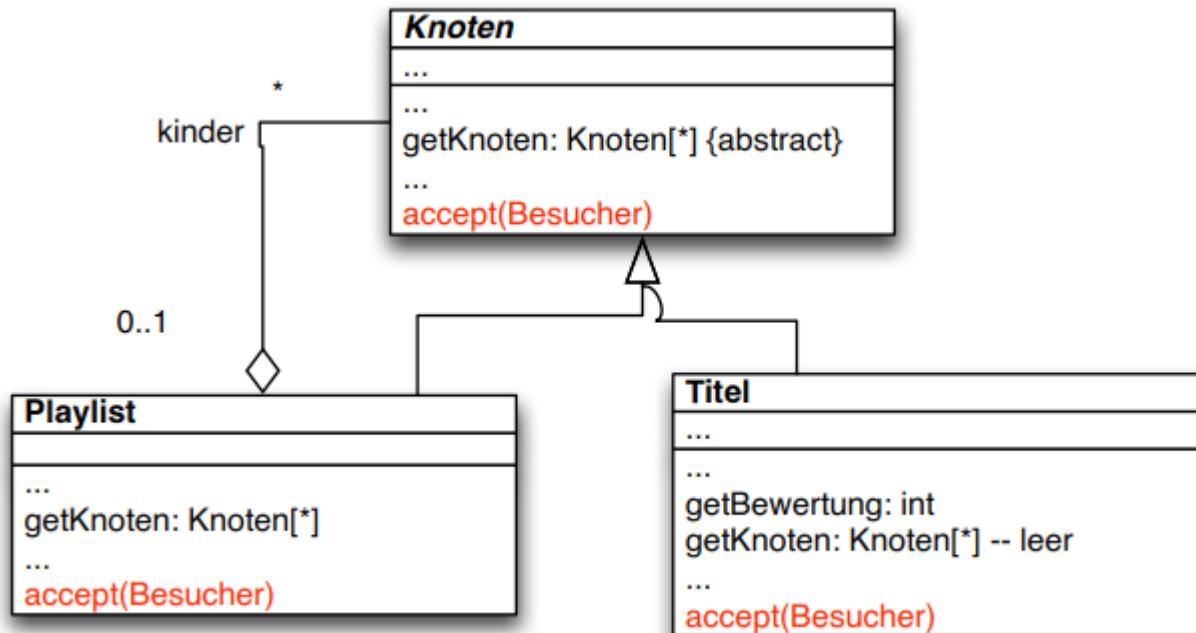
A.f(...)

B.f(...), weil f in A und B definiert

- Funktioniert jedoch nicht wenn die Methode in einer anderen Klasse/ Klassenhierarchie definiert ist (siehe Bsp. vorher)
- **BEM:** Es gibt Programmiersprachen, in denen der Polymorphismus auch für Parameter funktioniert („Multimethoden“)
  - Z.B.: Scala, Groovy (beide Java-basiert und sehr interessant!), Haskell, ...

# UMSETZUNG: 3. VERSUCH – DIESES MAL Klappt's!

- Entwurfsmuster „Besucher“ – Teil 1:
  - accept*-Methode für alle Klassen der Baum-Struktur:



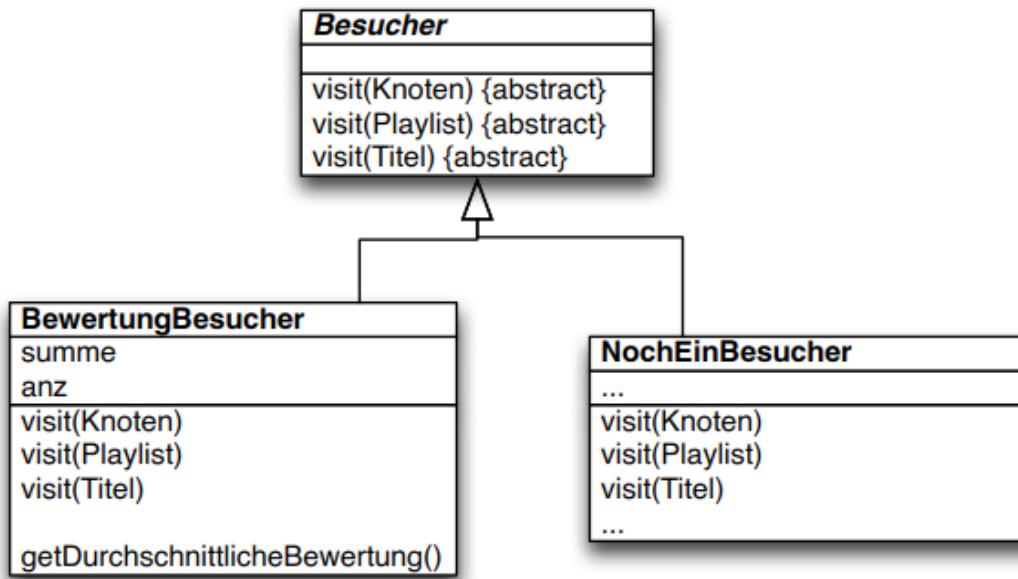
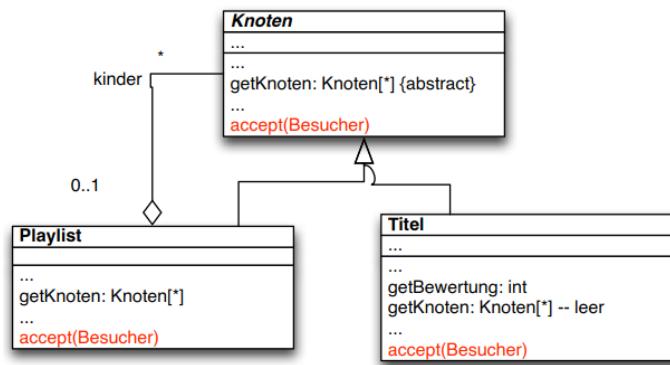
- Gleiche Implementierung von `accept(Besucher b)` für alle Klassen: `b.visit(this)`



Vorsicht: Diese eine Code-Zeile muss in der `accept`-Methode **jeder** UnterkLASSE stehen (wegen Polym.-Problem)!

# UMSETZUNG: 3. VERSUCH – DIESES MAL Klappt's!

- Entwurfsmuster „Besucher“ – Teil 2:
  - Gemeinsame Schnittstelle/Oberklasse *Besucher* aller Klassen, die Zusatzberechnungen auf Baum-Struktur durchführen:

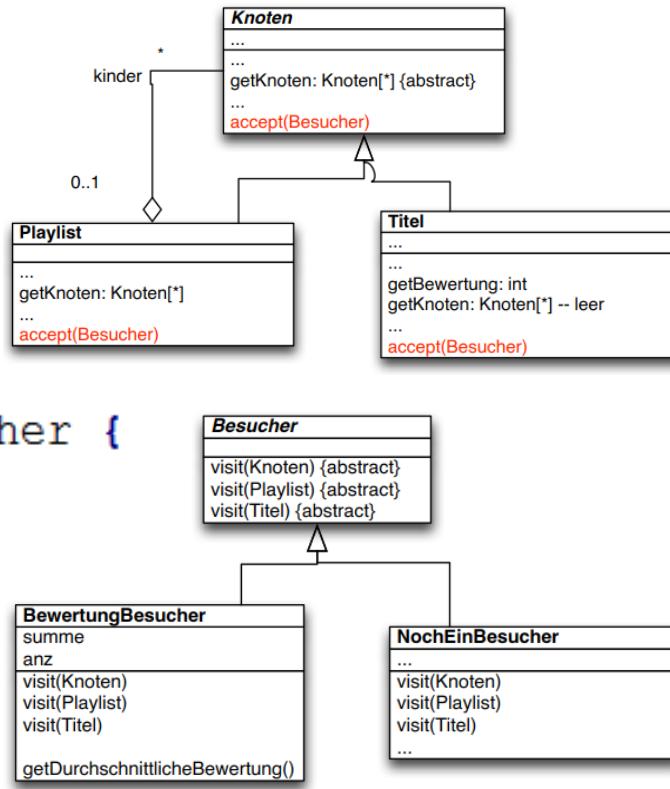


- und Aufruf von `visit` für Unterknoten **nur** indirekt über Aufruf von `accept` in der jeweiligen Datenstruktur!

# UMSETZUNG: 3. VERSUCH – DIESES MAL Klappt's!

- Implementierung einer Besucher-Klasse:

```
public class BewertungBesucher extends Besucher {  
    int summe; int anz;  
    public void visit(Titel t) {  
        summe += t.getBewertung(); anz++;  
    }  
    public void visit(Playlist p) {  
        for (Knoten k: p.getKnoten()) {  
            k.accept(this);  
        }  
    }  
    public void visit(Knoten k) {  
        //... wird das jemals aufgerufen? ...  
        EH.Assert(false,...,"Unbehandelter Knotentyp {0}",k);  
    }  
    public double getBewertung() {  
        if (anz>0) {return ((double)summe)/anz;}  
        else {return 0.0;}  
    }  
}
```



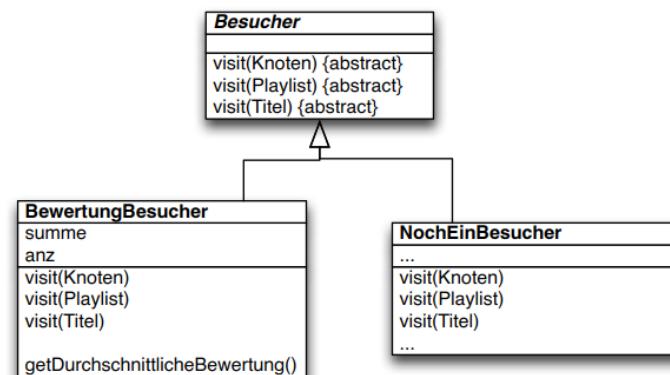
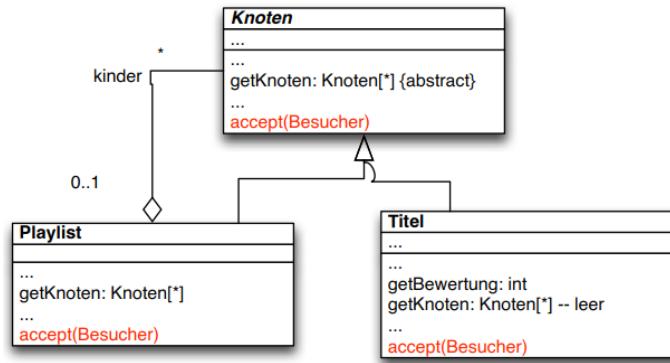
# UMSETZUNG: 3. VERSUCH – DIESES MAL Klappt's!

- Verwendung einer Besucherklasse:

```
// hinter diesem Knoten haengt
// ggfs. ein ganzer Baum
Knoten k = ...;
BewertungBesucher b = new BewertungBesucher();

// Ausloeser fuer rekursiven Abstieg
// bis hin zu allen Blaetttern
k.accept(b);

System.out.println(b.getBewertung());
```

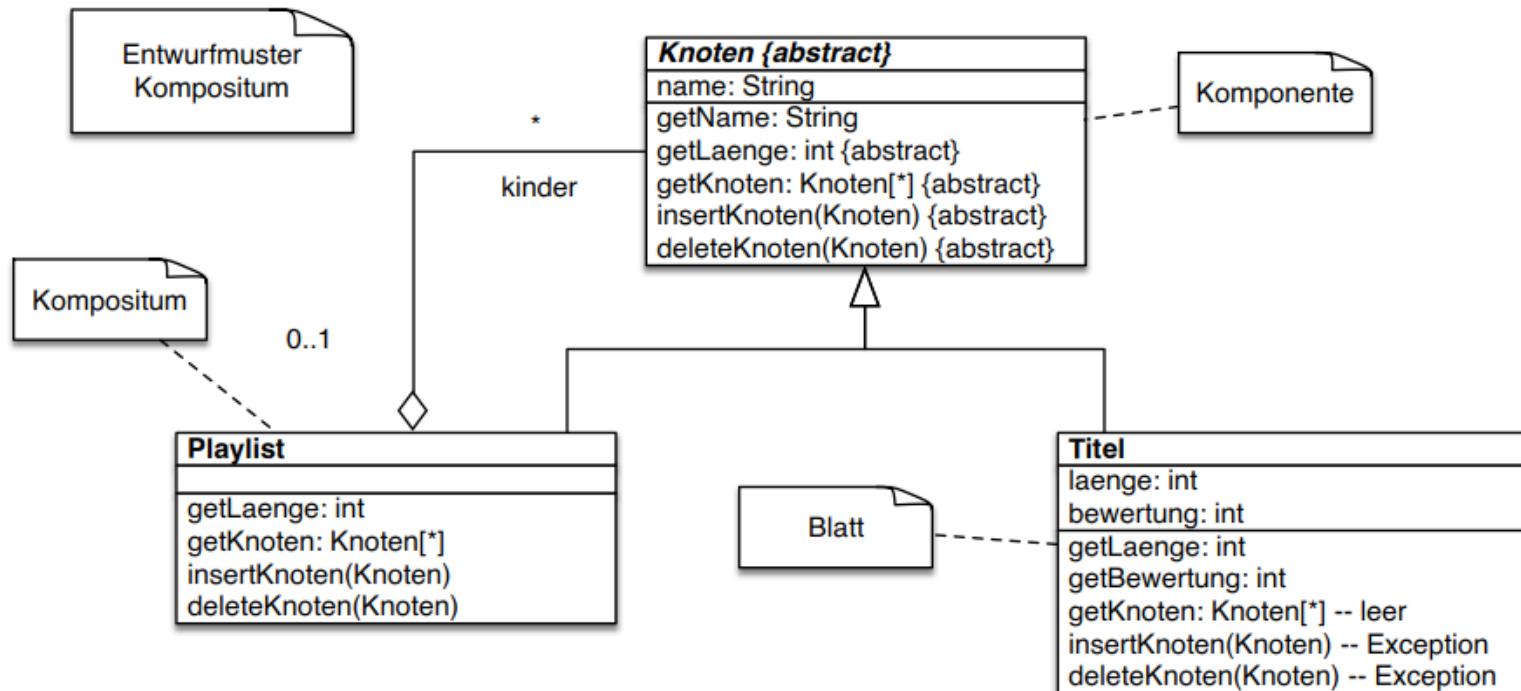


# BEISPIEL FEINENTWURF „PLAYLISTENVERWALTUNG“



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

Wir erinnern uns:



→ Kompositum-Muster

- Jetzt: Besucher-Muster
- Beide harmonieren miteinander
- Gutes Beispiel für sich ergänzende / verstärkende Muster

# BESUCHER-MUSTER IM ÜBERBLICK



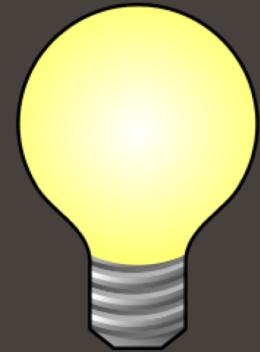
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Kern des Entwurfsmusters:
    - Wechselseitiger Aufruf von *accept* und *visit*  
(2 Mal Polymorphismus):
      - *knoten.accept(besucher)* → offenbart tatsächlichen Typ von *knoten*
      - *besucher.visit(knoten)* → offenbart tatsächlichen Typ von *besucher*
      - ⇒ *visit* kennt tatsächlichen Typ von *knoten* und *besucher*
  - Verwendung:
    - Oft zusammen mit „Kompositum“
    - ABER: Kompositum ist keine zwingende Voraussetzung
      - Können auch andere komplexere Datenstrukturen sein, die traversiert werden müssen  
(z.B. Bäume, Graphen, ...)
- Gut, wenn komplexe Strukturen für verschiedenen Funktionalitäten durchtraversiert werden müssen



03b

## Entwurfsmuster – Beobachter (Observer)



Ziel:

Entwurfsmuster Besucher genauer kennenlernen

# INTERAKTIVE PROGRAMME



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Heutzutage haben wir Programme, die mit den Benutzern interagieren
  - Interaktive Programme
- Interaktive Programme:
  - Programm wartet auf Aktion des Benutzers
  - Aktion löst ein Ereignis aus (Maus-Klick, Button-Klick, ...)
- Frage: Wie löst man diese Ereignis-Verarbeitung am geschicktesten?
  - Nicht starr, sondern möglichst flexibel

# INTERAKTIVE PROGRAMME-BEOBACHTER-MUSTER



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Wie löst man diese Ereignis-Verarbeitung am geschicktesten?

## → Beobachter Muster:

- Zwei grundlegende Phasen im Ablauf:
  - Phase 1 (vor dem Start der Ereignis-Verarbeitung)
    - Ereignis-Empfänger registrieren
  - Phase 2 (während der Ereignis-Verarbeitung)
    - Ereignis tritt ein → alle registrierten Ereignis-Empfänger benachrichtigen
  - (Evtl. Phase 3)
    - Wenn nicht mehr benötigt, Ereignis-Empfänger abmelden
- Andere Namen:
  - Call-Back, Listener, Observer, Publisher-Subscriber, . . .

# INTERAKTIVE PROGRAMME-BEOBACHTER MUSTER



- Code-Beispiel: Ereignis „Klick auf Menüentrag“

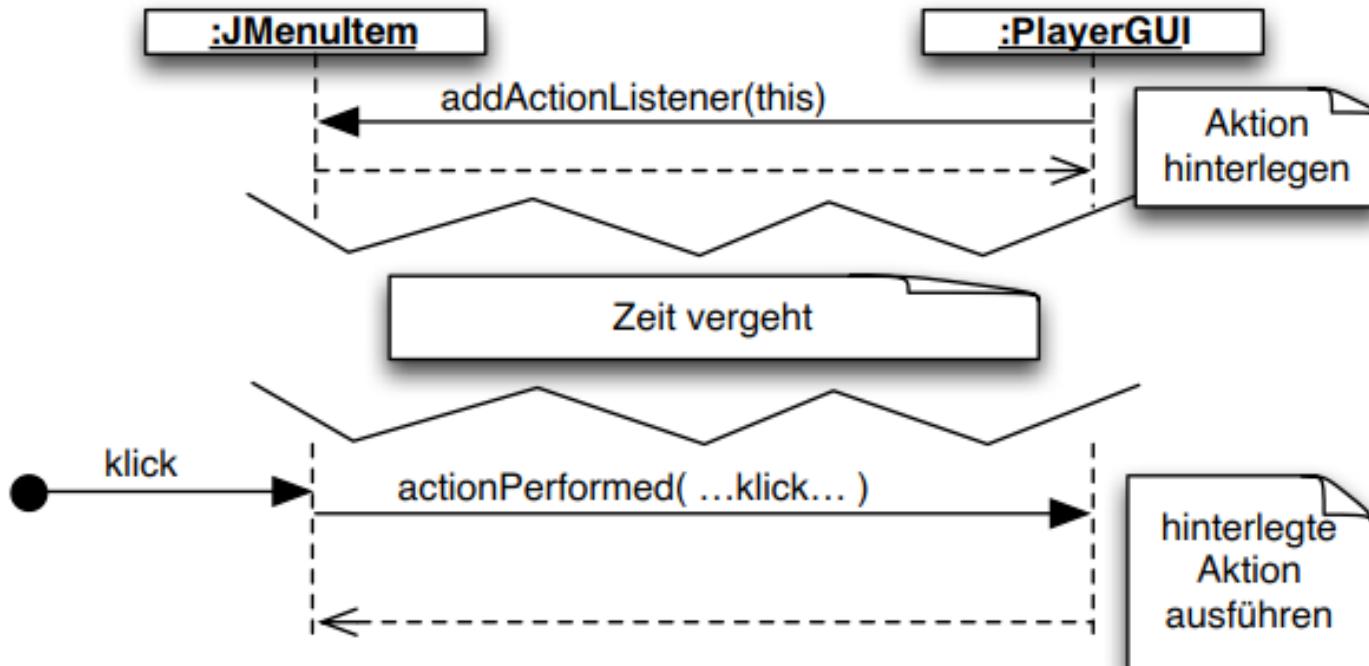
```
01 public class PlayerGUI extends JPanel implements ActionListener
02 {
03     JMenuItem newPlaylist;
04     public MeineGUI() {
05         ...
06         newPlaylist=new JMenuItem("New Playlist");
07         // hinterlegte Aktion(en) registrieren
08         newPlaylist.addActionListener(this);
09     }
10
11     public void actionPerformed(ActionEvent arg0) {
12         // hinterlegte Aktion(en) ausfuehren
13         ...
14     }
15     ...
16 }
```

# INTERAKTIVE PROGRAMME-BEOBACHTER MUSTER



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Code-Beispiel: Ereignis „Klick auf Menüentrag“



# DAS BEOBACHTER-MUSTER\*



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Kurzbeschreibung:
  - „Definiere eine 1-zu-n-Abhangigkeit zwischen Objekten, so dass die anderung des Zustands eines Objekts dazu fuhrt, dass alle abhangigen Objekte benachrichtigt werden.“
- Andere Namen:
  - Observer, Publisher-Subscriber, Call-Back, Listener



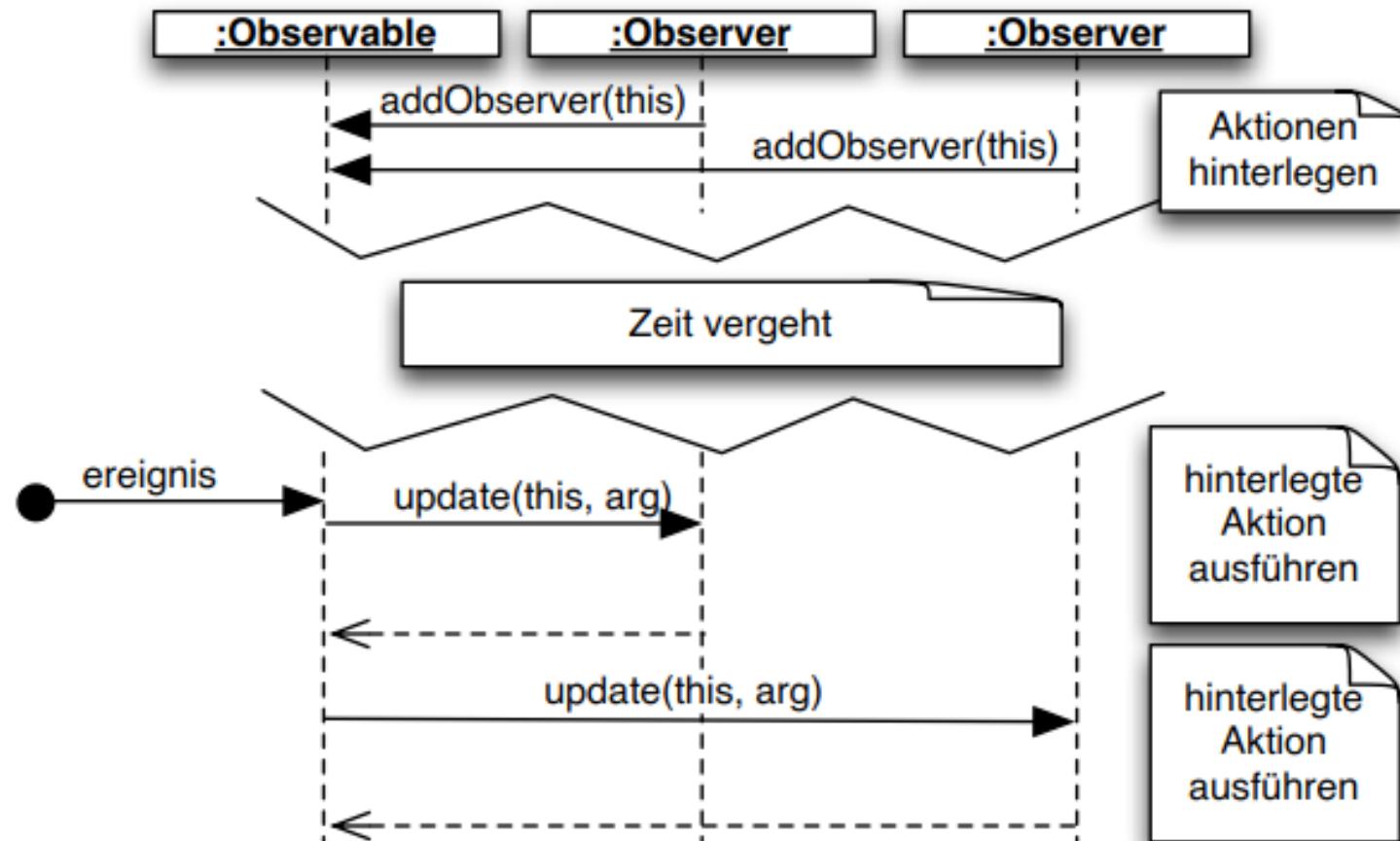
# DAS BEOBACHTER-MUSTER\*

- Problem:
  - Mehrere miteinander interagierende Klassen
    - Aufrechterhalten der Konsistenz zwischen den Objekten
- Lösung: Aufteilung (grob)
  - Subjekt (Observable)
    - kennt seine Beobachter
    - hat Methoden zum An-/Abmelden von Beobachtern
    - hat Methode zum Benachrichtigen der Beobachter
  - Beobachter (Observer)
    - hat Methode, die vom Subjekt bei Veränderungen aufgerufen wird → Benachrichtigung



# DAS BEOBACHTER-MUSTER\*

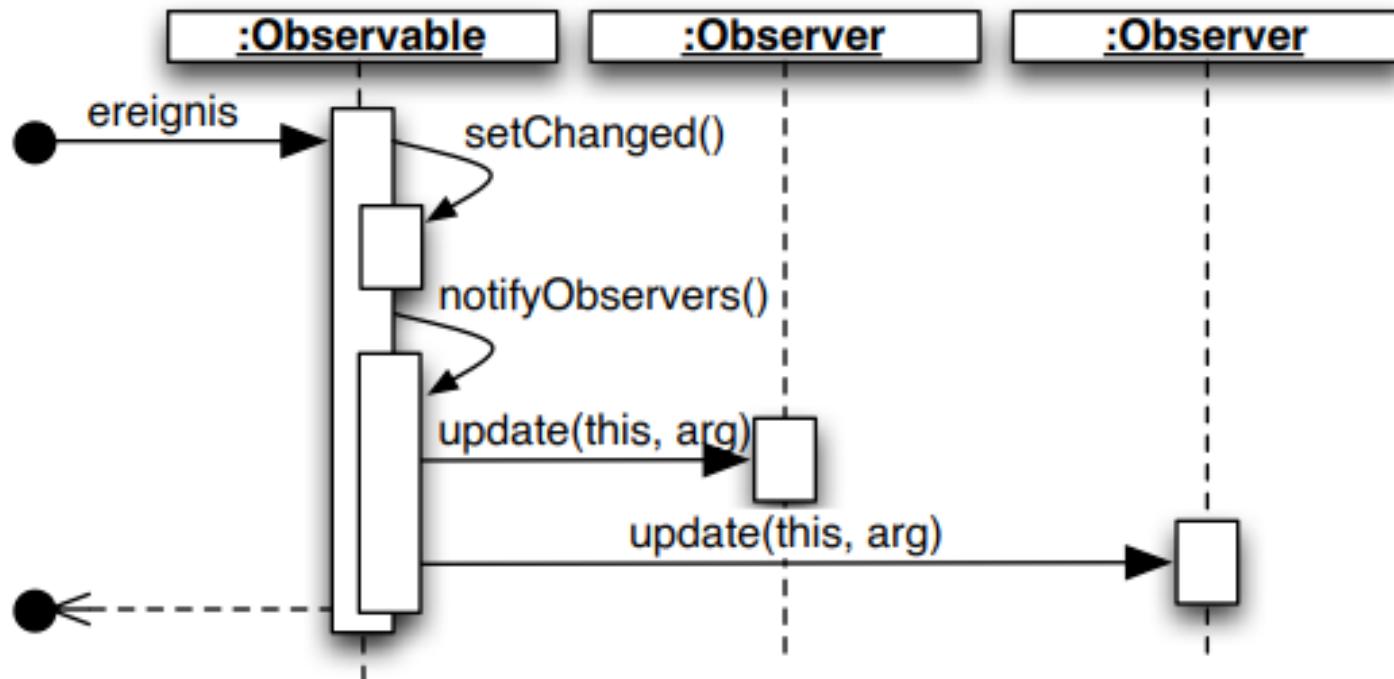
- Grober Ablauf mit zwei Beobachtern:





# DAS BEOBACHTER-MUSTER\*

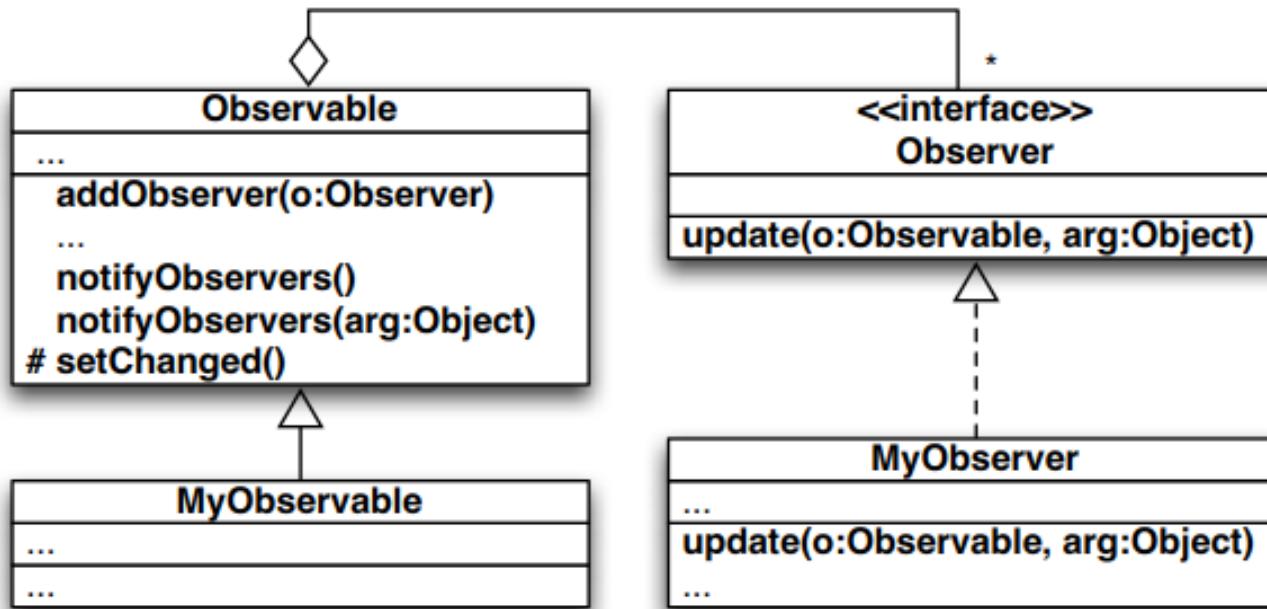
- Details des Ablaufs:





# DAS BEOBACHTER-MUSTER\*

- Klassendiagramm:

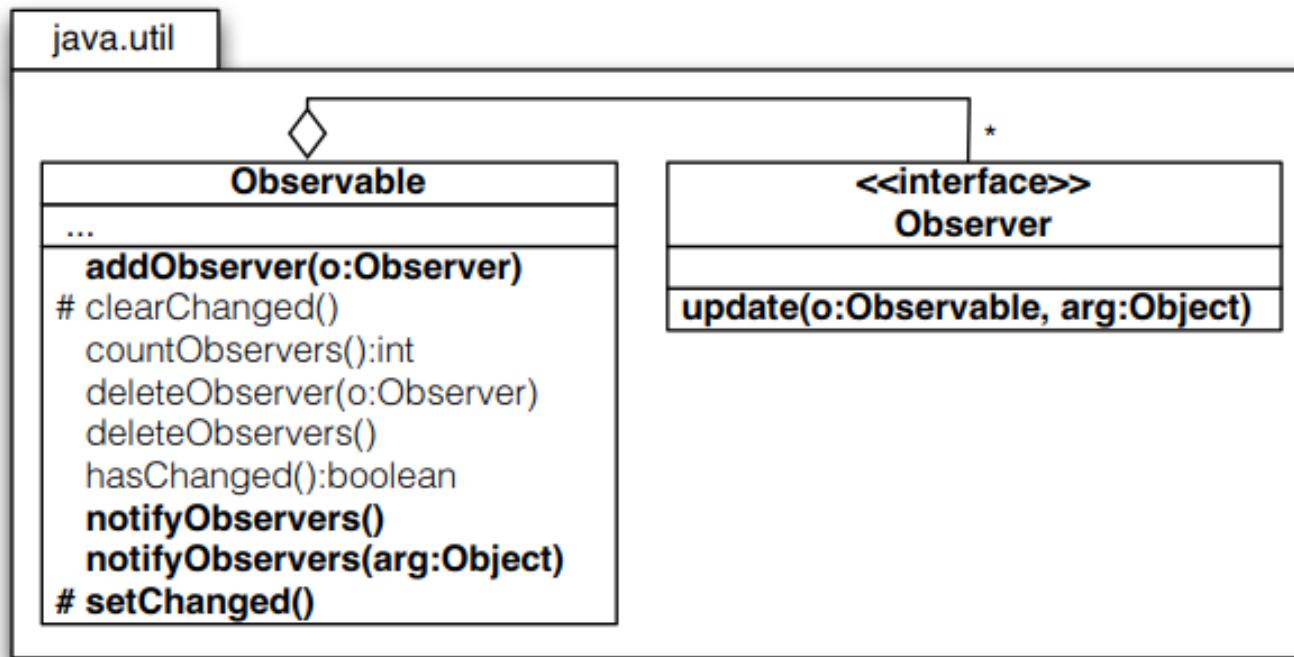


# DAS BEOBACHTER-MUSTER IN JAVA



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

BEM: Der allgemeine Teil des Musters ist sogar in der Java API vorhanden\*:



→ Eigenes Subjekt von Observable erben und für die Observer das Observer Interface implementieren

\* Leider ab Java 9 deprecated ☹ → stattdessen PropertyChangeListener aus "java.beans" verwenden

# DAS BEOBACHTER-MUSTER IN JAVA



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Aber auch alle typischen Listener-Interfaces und die GUI-Komponenten auf die man sich für die Events registrieren kann folgen dem Observermuster
  - Es heisst nur Listener statt Observer
  - Die Eventmethoden haben andere -spezifischere- Namen und Parameter



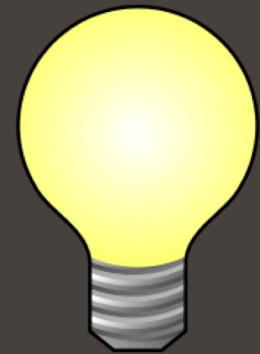
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

03c

## Entwurfsmuster – Gängige Muster\*

Ziel:

Liste gängiger Muster kennenlernen



\* Nach Gamma et al. (GoF)

# ERZEUGUNGSMUSTER\*



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Abstrakte Fabrik (Abstract Factory)
- **Erbauer (Builder)**
- **Fabrikmethode (Factory Method)**
- Prototyp (Prototype, ≈ Objekterzeugung in JavaScript)
- Singleton
  - Ist für manche eher ein Idiom (auf Codeebene)
  - Mittlerweile kritisch gesehen, da es aufgrund seiner Starrheit Probleme beim Unittesting (v.a. bei Parallelisierung) geben kann

# STRUKTURMUSTER\*



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- **Adapter**
- **Brücke (Bridge)**
- Dekorierer (Decorator)
- Fassade (Facade)
- Fliegengewicht (Flyweight)
- **Kompositum (Composite)**

# VERHALTENSMUSTER\*



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- **Befehl (Command)**
- **Beobachter (Observer, Listener)**
- **Besucher (Visitor)**
- Interpreter
- **Iterator**
  - (in Java und C++ inzwischen fest eingebaut)
- Memento
- **Schablonenmethode (Template Method)**  
(vgl. Method-Object-Pattern auf der nächsten Folie)
- **Strategie (Strategy)**
- Vermittler (Mediator)
- Zustand (State)
  - (wird von manchen inzwischen als Anti-Muster angesehen)
- **Zuständigkeitskette (Chain of Responsibility)**

# SONSTIGE VERHALTENSMUSTER



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Method-Object-Pattern
  - Hatten wir in Programmiermethoden, 1. Vorlesung
  - Kein GoF-Pattern
    - Sondern aus Kent Beck: Smalltalk Best Practice Patterns
  - IDEE: Eine Klasse steht für einen Algorithmus  
(Algorithmus wird in einer Klasse gekapselt)
  - Sehr ähnlich zu Template-Method-Pattern (aber ohne Vererbung)
  - Interagiert hervorragend mit:
    - Strategy (Auswahl verschiedener Algorithmen durch User)
    - Template-Method  
(Abstrakt formulierter Grundalgorithmus wovon geerbt und offene Methoden überschrieben werden können)

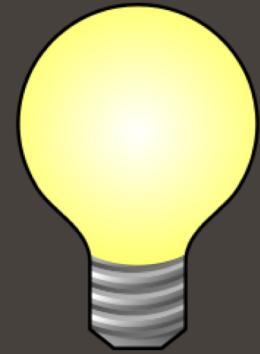


04a

## Architekturmuster – 3-Schichten-Arch.

Ziel:

Die 3-Schichten-Architektur kennenlernen  
(eigentlich Wiederholung)

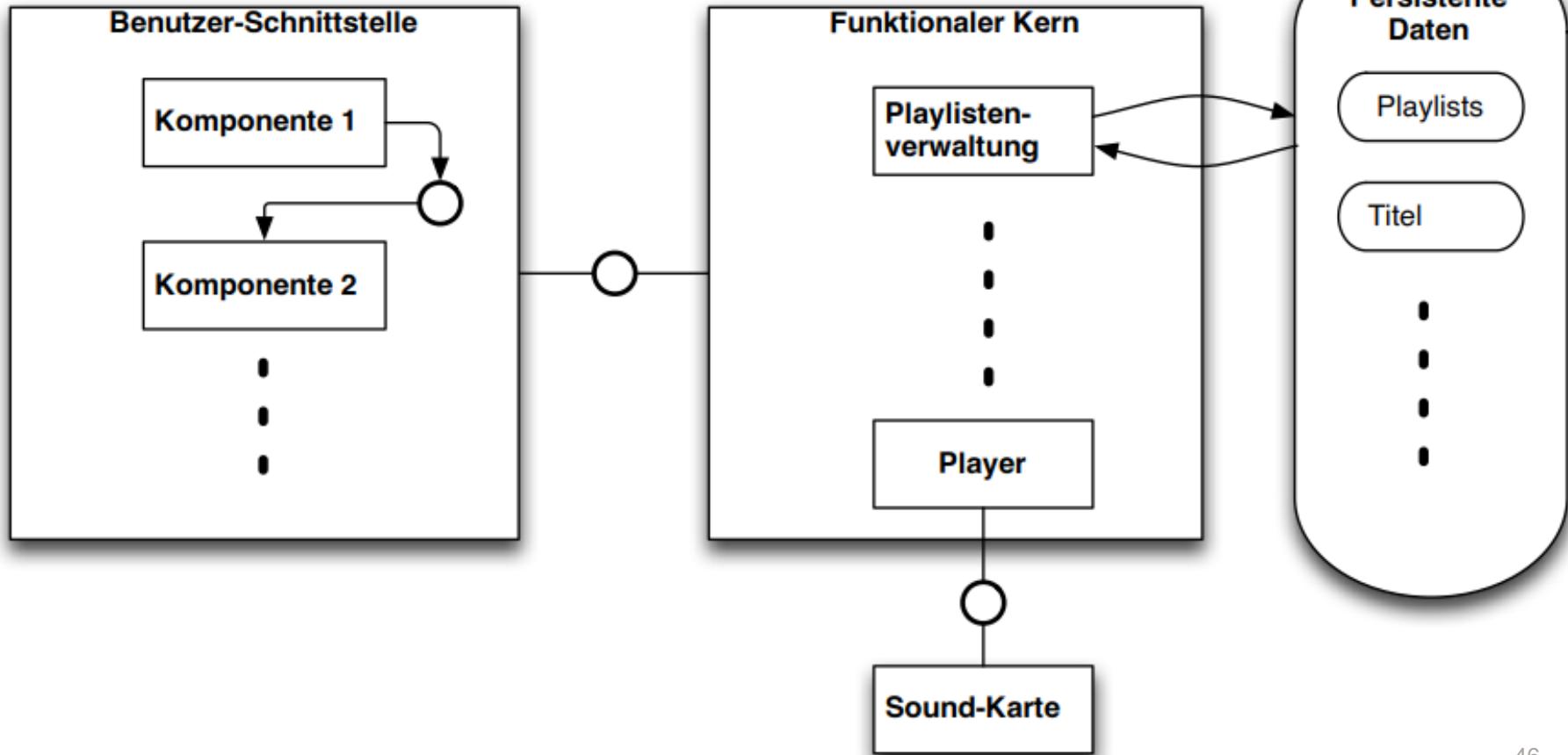


# BEISPIEL 3-SCHICHTEN-ARCHITEKTUR



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Beispiel: MP3-Player in 3-Schichten-Architektur:  
→ Grobentwurf (verfeinert, immer noch unvollständig)



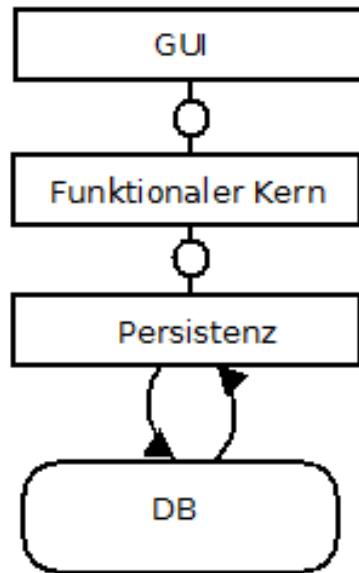
# HINWEIS ZUR 3-SCHICHTEN-ARCHITEKTUR



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Wird oft eher umgekehrt von oben nach unten dargestellt  
→ Schichten

So (z.B. FMC):



Oder auch so (z.B. UML):



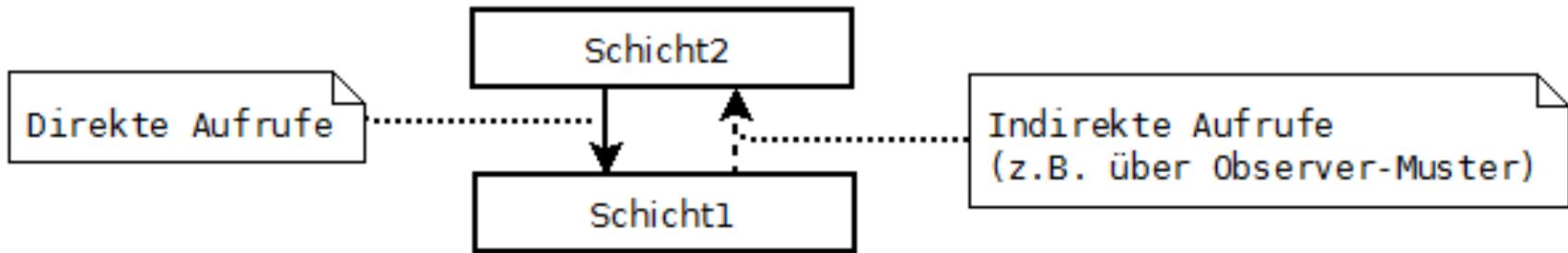
- Oft wird DB weggelassen und als Teil der Persistenzschicht betrachtet
  - Kann aber auch ohne DB sein (z.B. dann XML-Dateien, ...)
  - Deshalb: Besser auch die DB, ... darstellen

# WEITERER HINWEIS ZU SCHICHTEN-ARCHITEKTUREN



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Meist kennt nur eine Schicht die andere Schicht direkt
  - Meist obere Schicht kennt die darunter liegende Schicht direkt
  - Kommunikation von darunter liegender Schicht erfolgt indirekt
    - Z.B. über Observer-Muster



→ So wird eine Entkoppelung erreicht  
(Siehe auch GRASP-Pattern Loose Coupling)

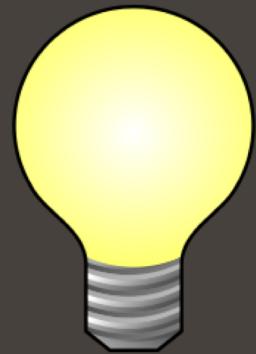


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## 04b

# Architekturmuster – MVC

Ziel:  
Das ModelViewController-Muster als  
Architekturmuster kennenlernen

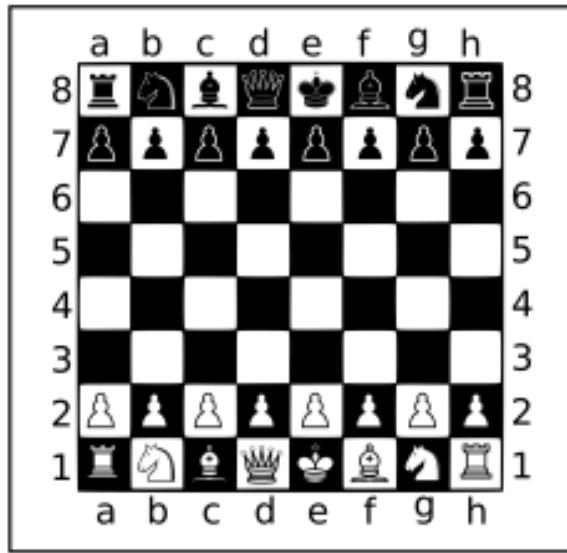


# BEISPIEL: EIN SCHACHPROGRAMM



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Grafische Benutzeroberfläche (ohne Menü):



Bildquelle (und Lizenz):

[http://commons.wikimedia.org/wiki/File:AAA\\_SVG\\_Chessboard\\_and\\_chess\\_pieces\\_02.svg](http://commons.wikimedia.org/wiki/File:AAA_SVG_Chessboard_and_chess_pieces_02.svg)

# BEISPIEL: EIN SCHACHPROGRAMM



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

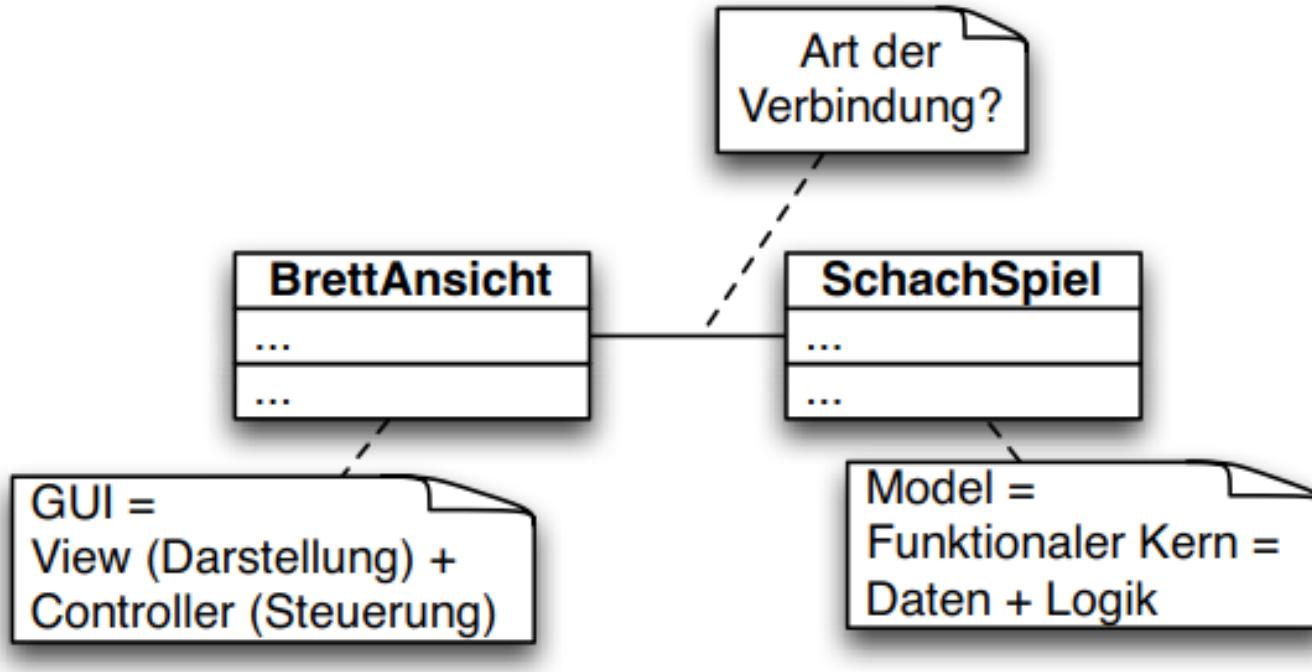
- Nichtfunktionale Anforderungen:
  - Ziel: Flexible GUI
    1. GUI soll leicht veränderbar sein, z.B.:
      - anderes Aussehen des Spielbretts
      - Swing → anderes GUI-Framework (z.B. Java FX)
    2. GUI soll leicht erweiterbar sein, z.B.:
      - neues Fenster: Historie der Spielzüge
      - neues Fenster: Direkteingabe
  - Auswirkung auf Entwurf:
    1. Saubere Trennung zwischen GUI und Spielelogik
      - GUI: Darstellung + Steuerung (Maus/Tastatur)
      - Funktionaler Kern: Spielzustand (Daten) + Spielelogik
    2. Spezieller Benachrichtigungs-Mechanismus

# BEISPIEL: EIN SCHACHPROGRAMM



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Entwurf:



→ Prinzipielle Idee: Saubere Trennung zwischen GUI und Spielelogik

# BEISPIEL: EIN SCHACHPROGRAMM



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Erweiterte GUI mit neuem Fenster “Zugansicht”:

Bisherige Züge:

a2-a3  
b7-b5

...  
...

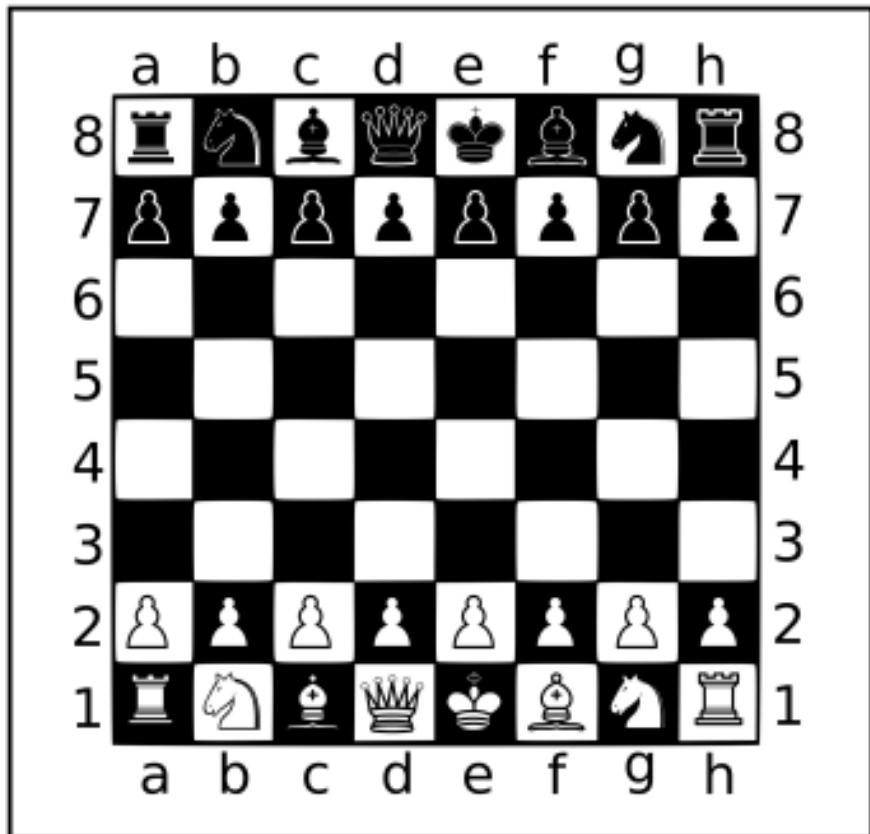
Bildquelle (und Lizenz):

[http://commons.wikimedia.org/wiki/File:AAA\\_SVG\\_Chessboard\\_and\\_chess\\_pieces\\_02.svg](http://commons.wikimedia.org/wiki/File:AAA_SVG_Chessboard_and_chess_pieces_02.svg)

# BEISPIEL: EIN SCHACHPROGRAMM



- Erweiterte GUI mit neuem Fenster “Direkteingabe”:



The chessboard diagram shows a standard starting position of a chess game. The board is an 8x8 grid with columns labeled a-h and rows labeled 1-8. Black pieces are positioned at a8, b8, c8, d8, e8, f8, g8, h8, a7, b7, c7, d7, e7, f7, g7, h7, a2, b2, c2, d2, e2, f2, g2, h2. White pieces are at a1, b1, c1, d1, e1, f1, g1, h1.

Bisherige Züge:

a2-a3  
b7-b5  
...  
...

Spieler 1 am Zug:

von:  nach:

**Zurück** **OK**

Bildquelle (und Lizenz):

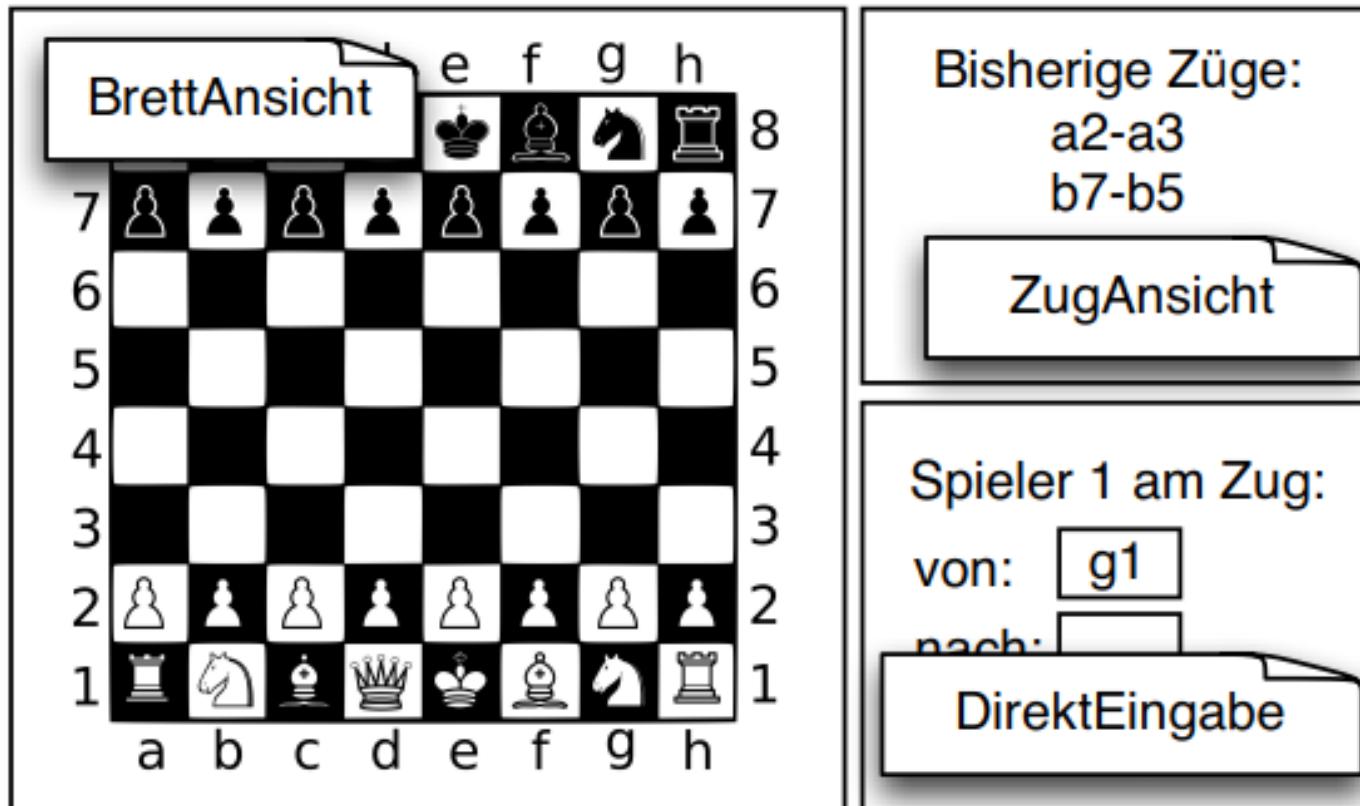
[http://commons.wikimedia.org/wiki/File:AAA\\_SVG\\_Chessboard\\_and\\_chess\\_pieces\\_02.svg](http://commons.wikimedia.org/wiki/File:AAA_SVG_Chessboard_and_chess_pieces_02.svg)

# BEISPIEL: EIN SCHACHPROGRAMM



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Erweiterte GUI – Klassennamen der Fenster:



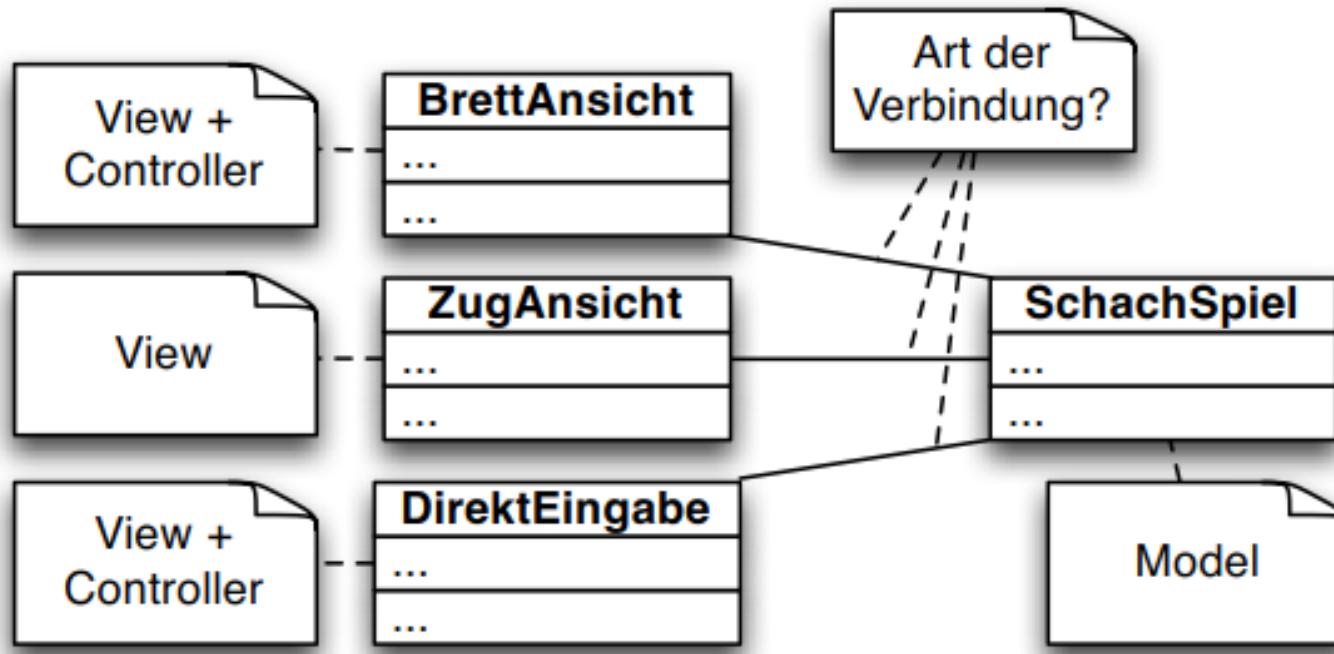
Bildquelle (und Lizenz):

[http://commons.wikimedia.org/wiki/File:AAA\\_SVG\\_Chessboard\\_and\\_chess\\_pieces\\_02.svg](http://commons.wikimedia.org/wiki/File:AAA_SVG_Chessboard_and_chess_pieces_02.svg)

# BEISPIEL: EIN SCHACHPROGRAMM



- Erweiterter Entwurf:



→ Prinzipielle Idee: Saubere Trennung zwischen GUI und Spielelogik

# BEISPIEL: EIN SCHACHPROGRAMM



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## Geeigneter Benachrichtigungsmechanismus?

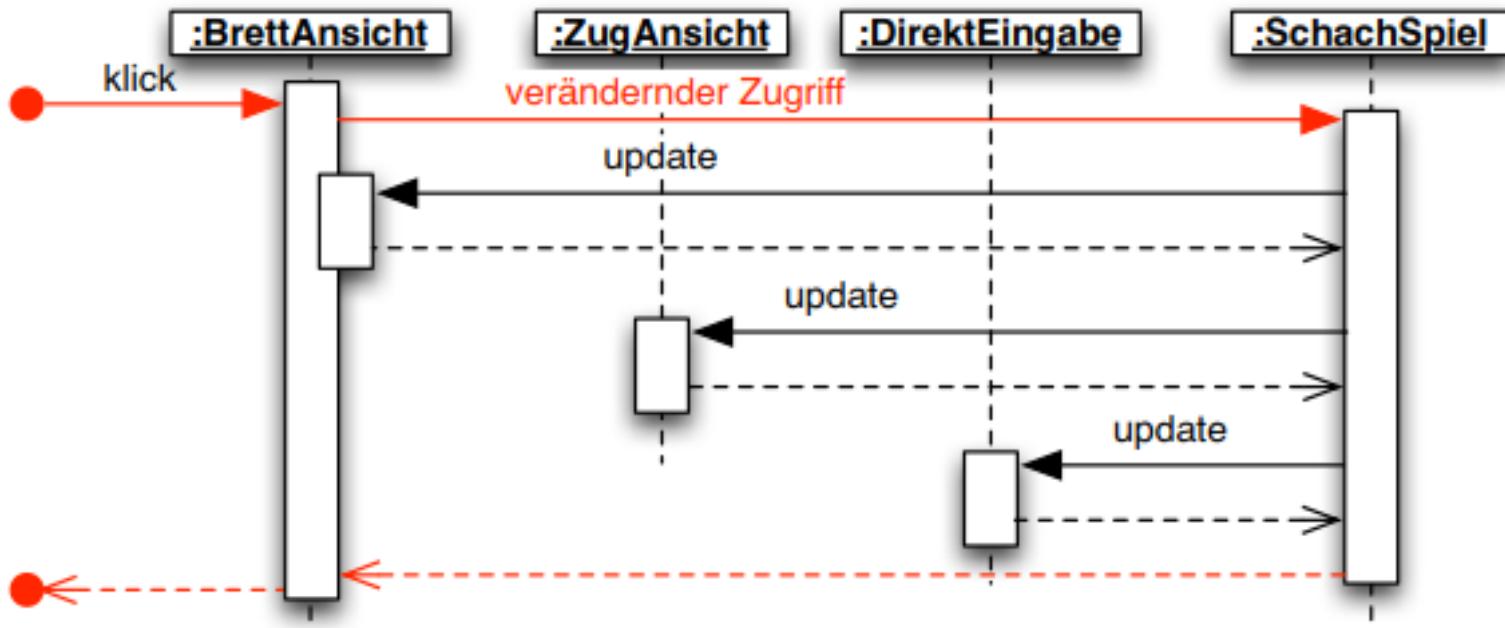
- Mögliche Ereignisse:
    - BrettAnsicht: Bewegen einer Spielfigur
    - DirektEingabe: Eingabe und Bestätigung eines Zugs
  - Problem:
    - Alle drei Ansichten (BrettAnsicht, ZugAnsicht, DirektEingabe)
    - Müssen stets den gleichen aktuellen Zustand anzeigen.
- Lösung:
- Bei Veränderung des Spielzustands:
    - Nachricht an alle Fenster: „Es hat sich was verändert.“
  - Verwendung des Entwurfsmusters „Beobachter“

# BEISPIEL: EIN SCHACHPROGRAMM



## Geeigneter Benachrichtigungsmechanismus?

- Grober Ablauf bei drei Fenstern
  - Auslöser: Klick in BrettAnsicht (z.B. Drag-And-Drop)

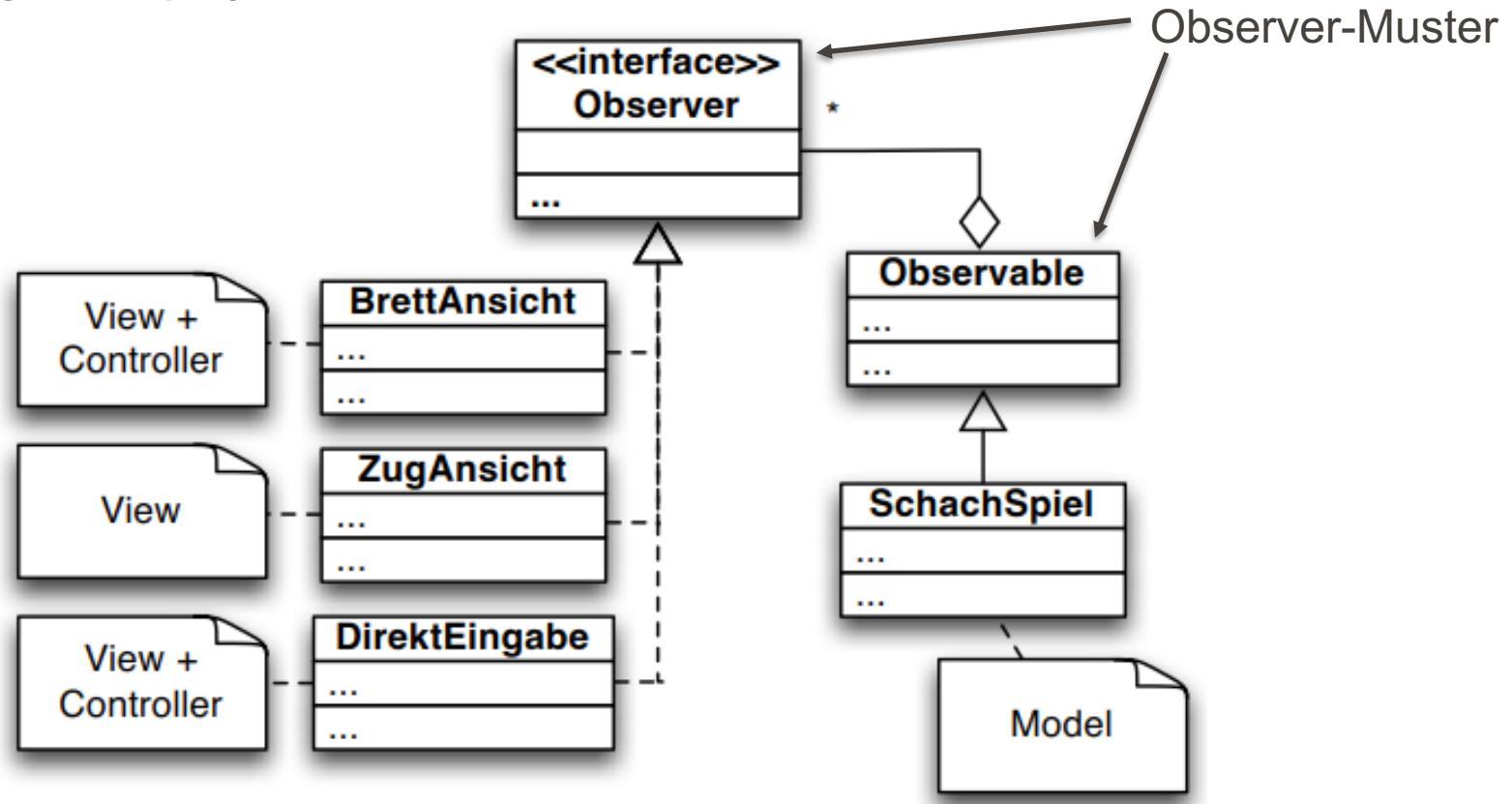


# BEISPIEL: EIN SCHACHPROGRAMM



## Geeigneter Benachrichtigungsmechanismus?

- Klassen-Entwurf:

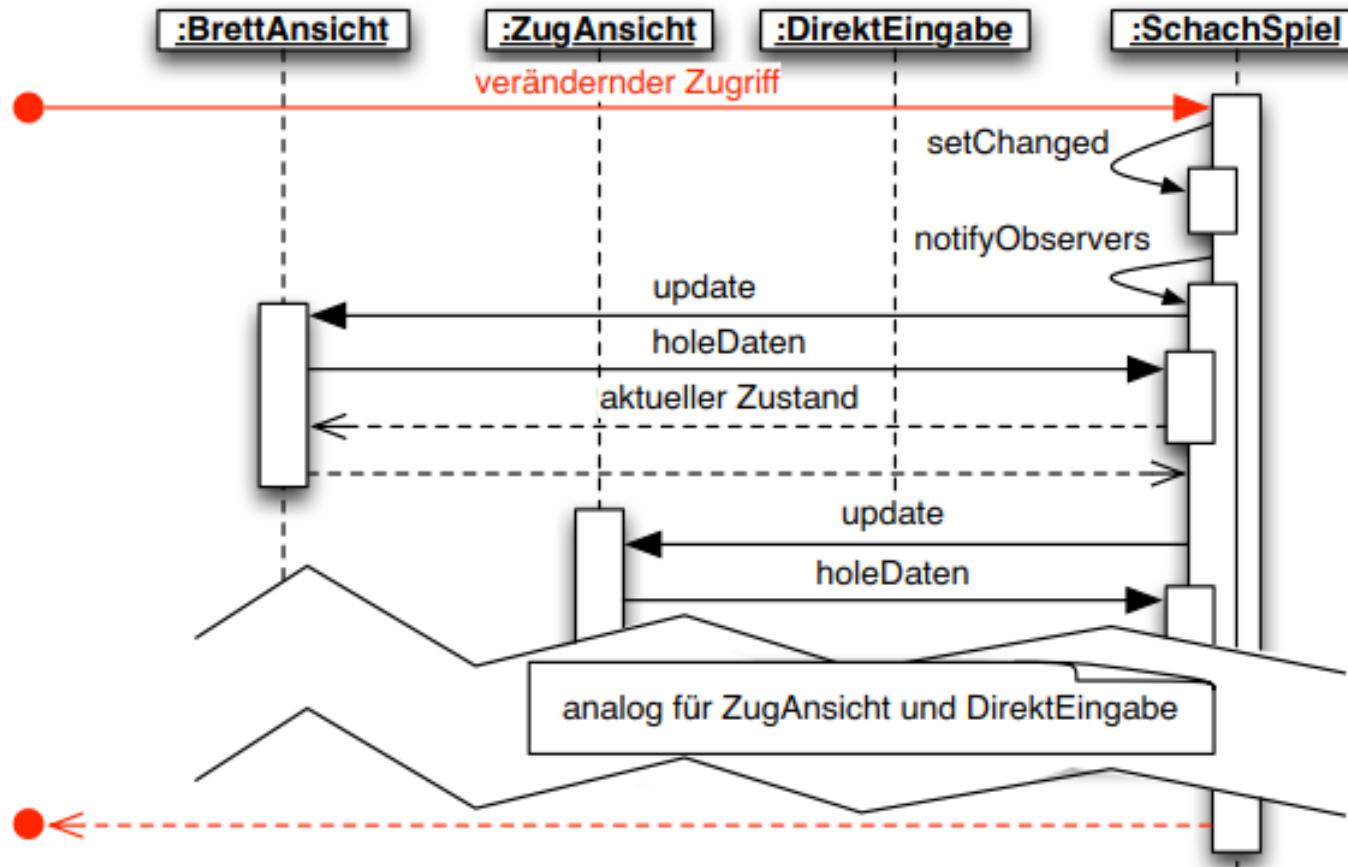


# BEISPIEL: EIN SCHACHPROGRAMM



## Geeigneter Benachrichtigungsmechanismus?

- Ablauf im Detail:



# DAS ARCHITEKTURMUSTER MODEL-VIEW-CONTROLLER\*



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Kurzbeschreibung:
  - „Das Model-View-Controller-Muster (MVC) unterteilt eine interaktive Anwendung in drei Komponenten. Das Modell enthält die Kernfunktionalität und die Daten. Ansichten (engl. views) präsentieren dem Anwender Informationen. Steuerungskomponenten sind für die Bedieneingaben verantwortlich. Ansichten und Steuerungskomponenten zusammen umfassen die Benutzerschnittstelle. Ein Mechanismus zur Benachrichtigung über Änderungen [...] sichert die Konsistenz zwischen der Benutzerschnittstelle und dem Modell.“
- Kontext:
  - Interaktive Anwendungen mit einer flexiblen Mensch-Maschine - Schnittstelle

\* Nach Buschmann et al.

# DAS ARCHITEKTURMUSTER MODEL-VIEW-CONTROLLER\*



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

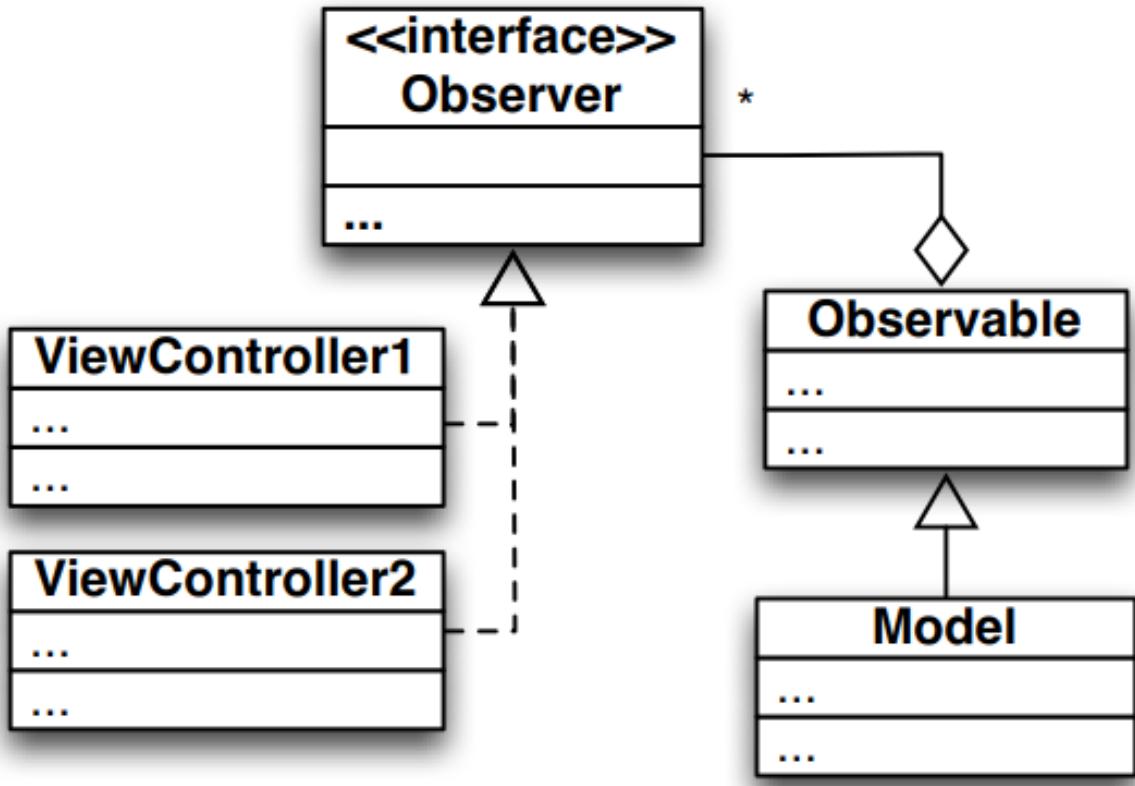
- Problem:
  - enge Kopplung zwischen GUI und Funktionalität macht GUI inflexibel
- Lösung:
  - Unterteilung einer Anwendung in drei Bereiche
    - Model = nur Kernfunktionalität (= Funktionaler Kern)
    - View = nur Darstellung der Daten aus dem Model
    - Controller = nur Benutzereingabe entgegennehmen und entsprechende Methode im Model aufrufen
  - Model informiert
    - interessierte Views
    - interessierte Controller
  - über Veränderungen

# DAS ARCHITEKTURMUSTER MODEL-VIEW-CONTROLLER\*



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Lösung (Klassen-Entwurf):

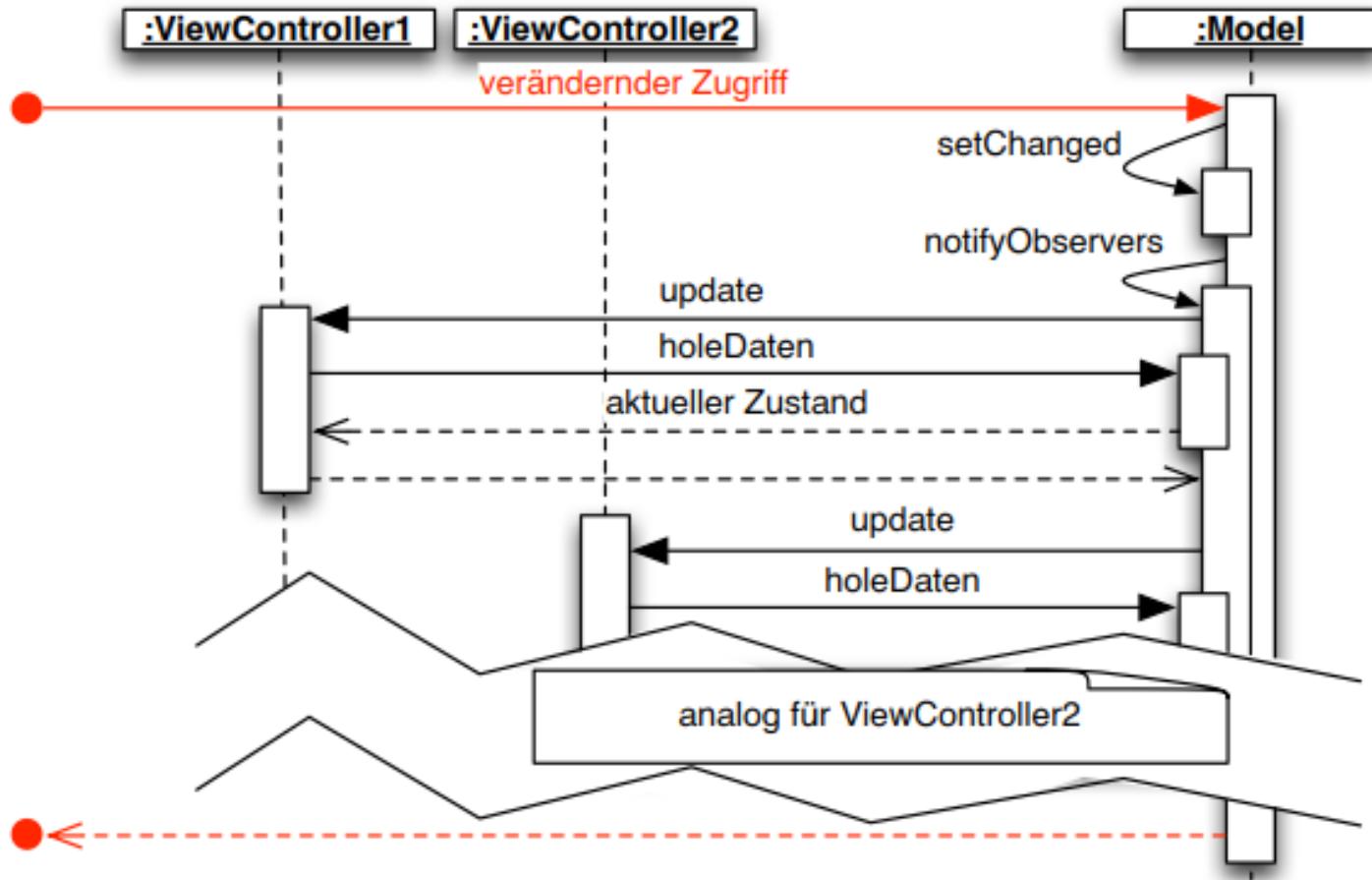


# DAS ARCHITEKTURMUSTER MODEL-VIEW-CONTROLLER\*



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Lösung (Ablauf):



# MODEL-VIEW-CONTROLLER – VARIANTEN:



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Aufteilung von View und Controller:
  - Variante 1: View und Controller in einem Objekt
    - Z.B. Java Swing
  - Variante 2: View und Controller je in einem separatem Objekt
    - Das „Original“ aus den Achtzigern (Smalltalk)
    - War aber damals sehr langsam → Heute nicht mehr so das Problem
- Datenzugriff:
  - Variante 1: View holt aktuellen Zustand vom Model (Methoden in Model für „hole Daten“)
  - Variante 2: Aktueller Zustand als Parameter in update

# MODEL-VIEW-CONTROLLER – WEITERE BEMERKUNGEN:



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Im 2. Semester Programmiermethoden:
    - JTree, TreeModel, ... → Funktioniert nach MVC-Muster
    - ABER: Nur eine einzelne GUI-Komponente
    - Eher ein Entwurfsmuster (für GUI-Komponenten)
      - Tatsächlich der Ursprung des MVC-Musters
      - MVC ist Grundlage für Java AWT/Swing  
(und andere GUI-Frameworks)
  - Hier:
    - Eher auf Ebene des gesamten Programms
    - Das gesamte Programm funktioniert nach diesem Prinzip
    - Architekturmuster
- D.h. es kann ein Muster wie MVC auch auf verschiedenen Ebenen des Designs geben

# MODEL-VIEW-CONTROLLER – WEITERE BEMERKUNGEN:



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim



## Vorsicht: Namensverwirrung

- „MVC“ für Web-Anwendungen  $\neq$  MVC das hier besprochen wird
- „MVC“ für Web-Anwendungen:
  - (Ursprünglicher Name „Model 2“ bei JSP)
  - 3-Schichten-Architektur für den Aufbau einer Web-Anwendung:
    - „View“: Darstellung + ggfs. Steuerung Browser
    - „Controller“: Geschäftslogik + ggfs. Steuerung Server
    - „Model“: Daten
  - zwingende Trennung von „View“ und „Controller“
  - **keine** Verbindung zwischen „View“ und „Model“ ( $\rightarrow$  „Controller“!)
  - **keine** Verwendung des Observer-Musters
    - Kein Push durch Server vorhanden  $\rightarrow$  mittlerweile schon!



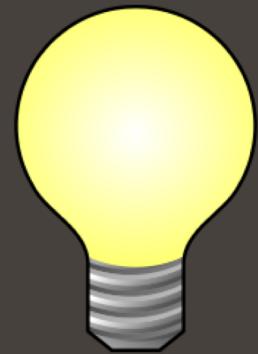
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

04c

## Architekturmuster – Gängige Muster

Ziel:

Liste gängiger Muster kennenlernen



# KATALOG AN ARCHITEKTURMUSTERN\*



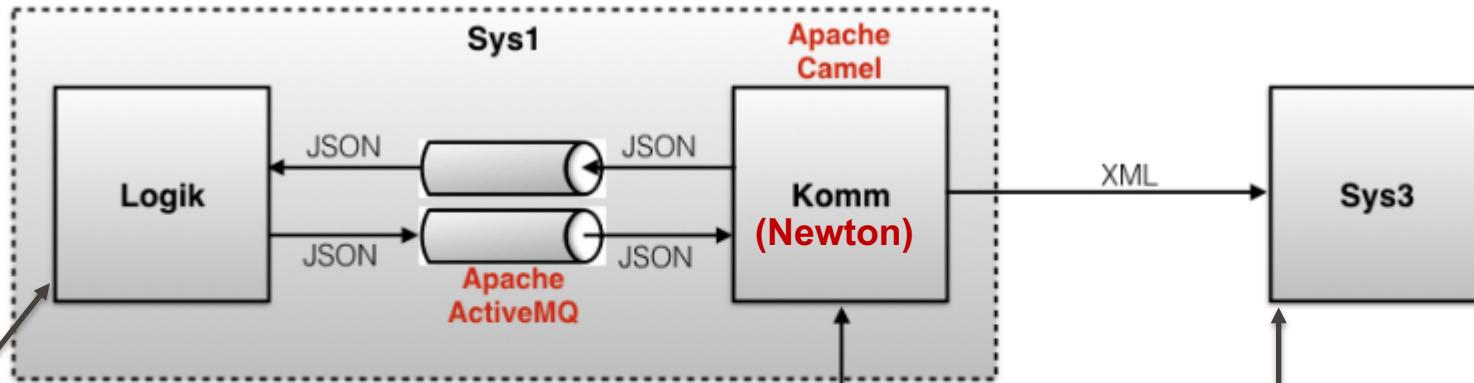
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- **Layers (Schichtenarchitektur)**
  - Allgemein Schichtenarchitektur
  - Verbreiteste: 3-Schichtenarchitektur
  - Aber auch: OSI-Schichtenmodell oder TCP/IP  
(vgl. Netzwerktechnik), ...
- Pipes-and-Filters ( $\approx$  UNIX-Pipes, Funktionale Progr., ...)
- Blackboard
- Broker
- **Model-View-Controller (MVC)**
- Presentation-Abstraction-Control
  - Leichte Erweiterbarkeit für Agenten-basierte Systeme
- Microkernel
- **Reflection** (siehe Programmiermethoden (auch Prinzip von Simple))

# BSP ZU PIPES-AND-FILTERS-ARCHITEKTURMUSTER



- Anwendungsarchitektur in einem Logistikunternehmen:



Sys1.Logik:  
Frachtenfragen  
werden mit  
Transportmitteln  
gematcht und  
ein Angebot  
erstellt

Sys1.Komm:  
Eigener Newton-  
Server zum Routen  
von Nachrichten  
über Apache Camel  
→ Integriert eigentl.  
3 Anwendungen  
miteinander

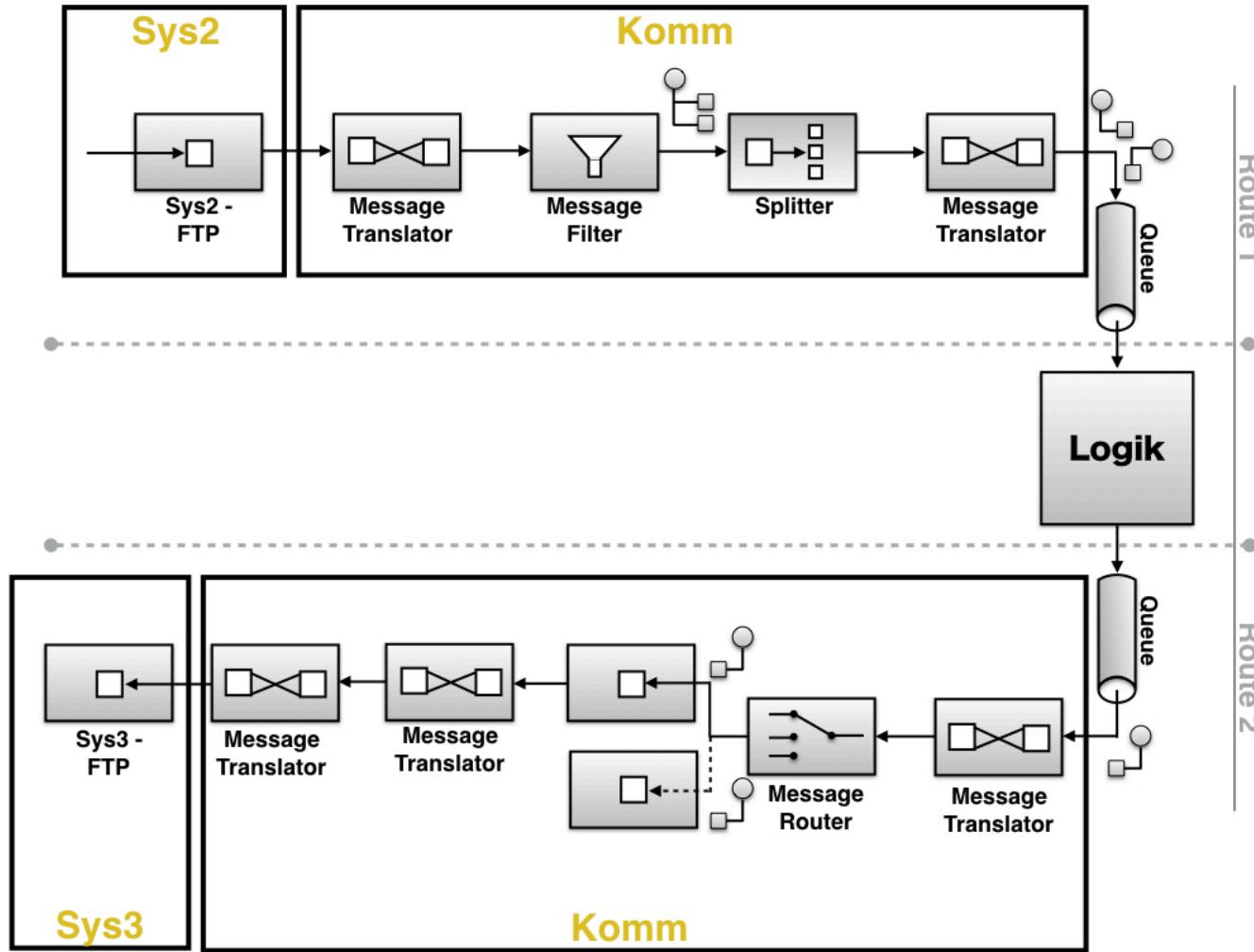
(Vergleiche auch Unternehmensarchitektur!)

Transportaufträge (Fracht,  
Transportmittel und Route)  
gehen über XML an Sys3  
zur Frachtverfolgung

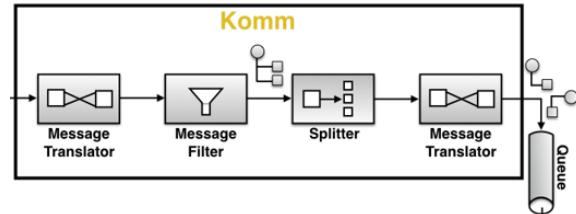
Sys2 (internationales System)  
stellt per FTP Transportmittel  
und Transportrouten zur  
Verfügung

# BSP ZU PIPES-AND-FILTERS-ARCHITEKTURMUSTER

- Routendefinition mittels Enterprise Integration Patterns\* (EIP):



# BEISPIEL ZU PIPES-AND-FILTERS



- Codebsp. Route 1 mittels Apache Camel:

```
public void configRoute1() throws Exception{
    Sys2MessageDataFormat dataFormat= new Sys2MessageDataFormat();
    from("ftp://sys2@xyftp.de:21/infodat?user=usr1&password=pwd123")
        .routeId("Route1")
        .autostartup(autostart)
        .bean(dataFormat, "unmarshal(*)")
        .bean(MessageFilter.class)
        .split().simple("${body}")
        .bean(Sys2ToSys1MessageConverter.class)
        .bean(toJsonFormat.class)
        .to("jms:queue:EingehendeQueue");
}
```

Hier wird auch  
Reflection benutzt  
→ Diese Klassen  
müssen auch  
entwickelt  
werden

- Apache Camel ist eine Domänenspezifische Sprache zur Umsetzung von Kommunik. zwischen Systemen mittels EIPs
- Funktioniert wie Funktionale Programmierung
- Pipes-And-Filters-Prinzip (Muster) auf Architekturebene

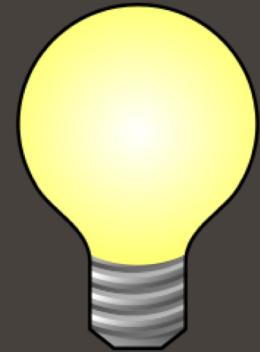


# 05

## Allgemeine Prinzipien

Ziel:

Die sog. GRASP-Muster als Beispiel  
für allgemeine Prinzipien kennenlernen  
(siehe auch: LARMAN, C.: Applying UML with Patterns; Kap. 16)





# WAS IST GRASP\*?

- GRASP = General Responsibility Assignment Software Patterns
  - engl. “to grasp something” = “etwas verstehen/begreifen”
- Sammlung von allgemeingültigen Entwurfsprinzipien
  - Verteilung von Zuständigkeiten auf Klassen
  - Wesentlich allgemeiner als Entwurfsmuster
  - Keine rezeptartige Lösung (wie z.B. Muster)
- Zusammenhang mit Entwurfsmustern:
  - Viele Entwurfsmuster unterstützen eines oder mehrere GRASP-Prinzipien
    - Viele Entwurfsmuster entstanden als eine mögl. Lösung für die Umsetzung eines oder mehrerer GRASP-Prinzipien

# CREATOR\*

## (= ERZEUGER-PRINZIP)



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Eine Klasse B ist besser geeignet Objekte einer Klasse A zu erzeugen, wenn eine der folgenden Bedingungen gilt:
  - B ist eine Aggregation von A (B besteht aus A-Objekten)
  - B enthält A-Objekte (contains)
    - Z.B. B hat Liste, die A-Objekte enthält
  - B erfasst A-Objekte (records)
    - Z.B. Fehlerlog, das einzelne Fehlereinträge erfasst
  - B verwendet A-Objekte mit starker Kopplung
    - Starke Abhängigkeit der Klasse B von A
  - B die Initialisierungsdaten für A hat
    - D.h. B ist Experte bezüglich Erzeugung von A
    - B erhält sie z.B. über Konstruktor

# LOW COUPLING\*

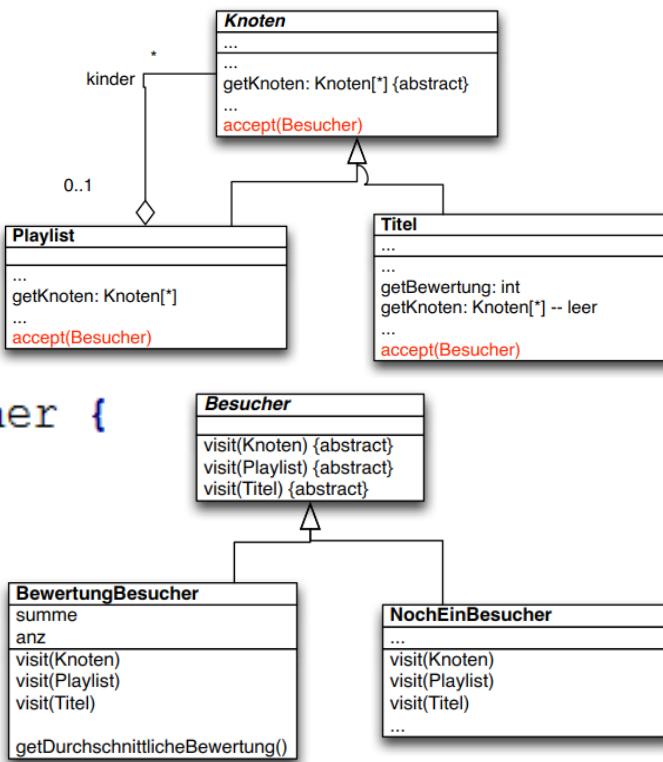
## (LOSE KOPPLUNG)



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Kurzbeschreibung:
  - Vermeide unnötige Verbindungen
  - Vermeide insbesondere Verbindungen zu instabilen Klassen/Schnittstellen
- Beispiel: MVC
  - Trennung zwischen GUI und funktionalem Kern
  - Funktionaler Kern ist **nicht abhängig** von (instabiler) GUI

# HINWEIS: BESUCHER & LOSE KOPPLUNG:



```

public class BewertungBesucher extends Besucher {
    int summe; int anz;
    public void visit(Titel t) {
        summe += t.getBewertung(); anz++;
    }
    public void visit(Playlist p) {
        for (Knoten k: p.getKnoten()) {
            k.accept(this);
        }
    }
    public void visit(Knoten k) {
        //... wird das jemals aufgerufen? ...
        EH.Assert(false,...,"Unbehandelter Kr")
    }
    public double getBewertung() {
        if (anz>0) {return ((double)summe)/ar}
        else {return 0.0;}
    }
}
  
```

Besucher entscheidet wie zum nächsten Datenelement gesprungen wird

- Jeder Besucher muss die besuchte Datenstruktur genau kennen
- **enge Kopplung**
- Was ist, wenn Datenstruktur komplexer ist / wird?  
(z.B. auch zyklische Abh.)

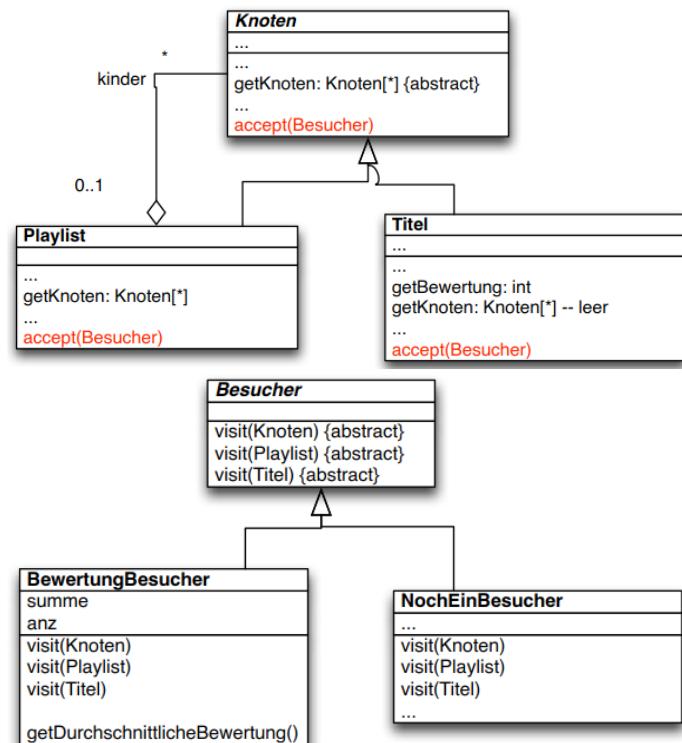
# HINWEIS: BESUCHER & LOSE KOPPLUNG

- Andere Variante:  
→ Datenstruktur lenkt Besucher

```
public class Playlist ... {
    public void starteBesuch(Besucher bes) {
        bes.visit(this);
    }
    public void accept(Besucher bes) {
        for (Knoten k: p.getKnoten()) {
            if(k instanceof Titel){bes.visit((Titel)k);}
            else {bes.visit((Playlist)k);}
        }
    }
}
```

```
public class BewertungBesucher extends Besucher {
    ...
    public void visit(Titel t) {...}
    public void visit(Playlist p) {p.accept(this); }
    public void visit(Knoten k) { ... }
}
```

Zum Einstieg  
beim „root“



Die besuchte Struktur  
managt den Abstieg in  
die Unterstrukturen

# WEITERE GRASP-MUSTER\*



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

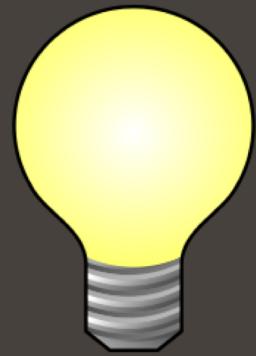
- Controller
  - Vgl. Controller bei MVC-Architekturmuster
- High Cohesion (Hohe Kohäsion)
- Polymorphismus (kennen wir)
  - Vgl. auch Strategy-Pattern der GoF
- Pure Fabrication
  - Reine Erfindung eines Elements
  - Z.B. Method-Objekt-Pattern
- Indirection
  - Vgl. Controller bei MVC-Architekturmuster
- Protected Variations
  - Z.B. Interfaces als Entkopplungsmechanismus für Variationspunkte



# 06

# Fazit

Ziel:  
Was haben wir damit gewonnen?



# WAS HABEN WIR GELERNT?



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Der Mustergedanke ist sehr wichtig im SE
  - Kondensiertes Expertenwissen
  - Leicht für Anfänger anwendbar
  - Gut zur Kommunikation zw. Experten geeignet
- Es gibt verschiedene Muster
  - Anforderungs-, Analyse-, Architektur-, Entwurfs-, ...
- Entwurfsmuster
  - Kompositum, Besucher, Beobachter, ...
- Architekturmuster
  - 3-Schichten, MVC, ...
- Allgemeine Prinzipien
  - GRASP



# WEITERFÜHRENDE LITERATUR

- Freeman et al: Entwurfsmuster von Kopf bis Fuß.
  - Schöne, umfassende Einführung zu Entwurfsmustern
  - Witzig geschrieben
- Starke: Effektive Software-Architekturen
  - Überblick zu Architektur- und Entwurfsmustern, Entwurfsprinzipien.
- Gamma et al: Entwurfsmuster.
  - Klassiker zu Entwurfsmustern (auch „Gang of Four“ genannt)
- Buschmann et al: Pattern-orientierte Software-Architektur. Bd. 1.
  - Klassiker zu Entwurfs- und v.a. Architekturmustern
- Larman, C.: Applying UML and Patterns [30 BF 500 78].
  - GRASP (Entwurfsprinzipien) ausführlich



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

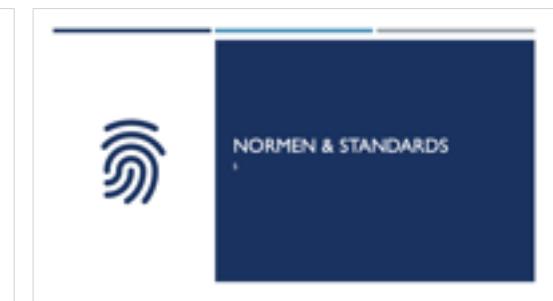
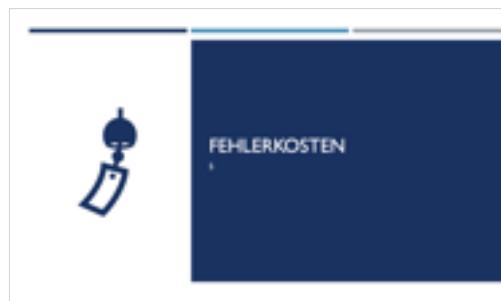
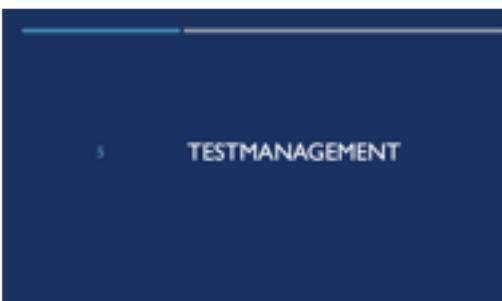
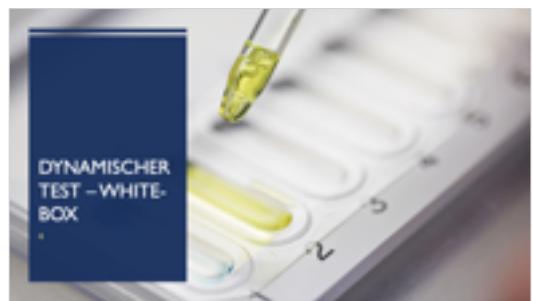
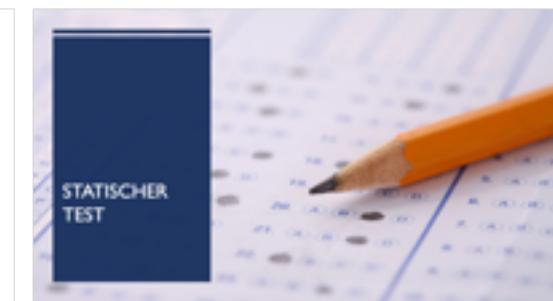
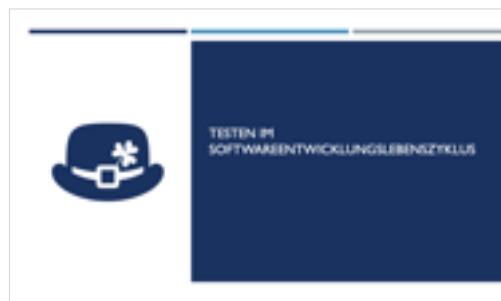
AUF GEHT'S!!

SELBER MACHEN UND LERNEN!!



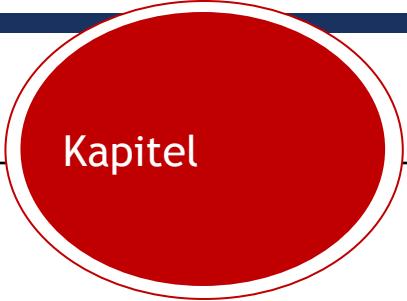


TESTEN





# EINLEITUNG



Kapitel

## EINLEITUNG



Certified Tester

Einstieg → Auswirkungen von Software-Fehlern

Anhang → Buchempfehlungen, Zeitschriften,  
Organisationen, Tagungen

# ZIELE DER LEHRVERANSTALTUNG

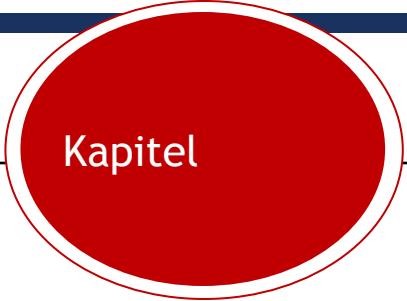


- Grundlagenvermittlung im Bereich Prüfen und Testen von Software
- Erklärung der Begriffe, Aufgaben und Tätigkeiten, Methoden und Testentwurfsverfahren
- Inhalt der Lehrveranstaltung deckt einen international festgelegten Lehrstoff für Weiterbildungseinrichtungen ab
- Vorbereitung für die Prüfung  
*Certified Tester - Foundation Level*
  - International anerkanntes Zertifikat
  - Anerkannte professionelle Spezialisierung
  - Branchenübergreifend  
(kommerzielle Software, Automotive, Web, Entertainment, ...)



# ISTQB® - FOUNDATION LEVEL 2018

<b>Grundlagen des Testens</b>	<b>Testen im Software Lebenszyklus</b>	<b>Statischer Test</b>	<b>Testverfahren</b>	<b>Testmanagement</b>	<b>Werkzeugunterstützung für das Testen</b>
Was ist Testen?	Software Entwicklungs-Lebenszyklusmodelle	Grundlagen Statischer Test	Kategorien von Testverfahren	Testorganisation	Überlegungen zu Testwerkzeugen
Warum ist Testen notwendig?	Teststufen	Reviewprozess	Black-box Testverfahren	Testplanung und -schätzung	Effektive Nutzung von Werkzeugen
7 Grundsätze des Testens	Testtypen		White-box Testverfahren	Testüberwachung und -steuerung	
Testprozess	Wartungstest		Erfahrungsbasierte Testverfahren	Konfigurationsmanagement	
Psychologie des Testens				Risiken und Testen	
				Fehlermanagement	



## EINLEITUNG



Certified Tester

Einstieg → Auswirkungen von Software-Fehlern

Anhang → Buchempfehlungen, Zeitschriften,  
Organisationen, Tagungen



# EINSTIEG

WAS IST EIN SOFTWARE-FEHLER?



9.9.1945

15:45

# DER ERSTE BUG!

Admiral "Amazing Grace" Hopper<https://www.youtube.com/watch?v=S6sh8CxwWx8>

Motte im Rechner Mark II verursacht Fehler  
in Relay Nr. 70, Panel F.

9th September 1947:  
Moth 'bug' discovered inside a Harvard computer

Mr. Grace Murray  
Hopper beseitigt  
und dokumentiert  
Fehler:  
»First actual case  
of bug being found.«

»offen«-sichtlicher  
Fehler!  
Beseitigung ist  
einfach!

9/9

0800 Auton started  
1000 .. stopped - auton ✓ { 1.2700 9.037 847 025  
1300 (032) MP - MC 1.98264000 9.037 846 995 correct  
033 PRO 2 2.130476415  
correct 2.130676415  
Relays 6-2 in 033 failed special speed test  
in relay 10.000 test.  
Relays changed  
1100 Started Cosine Tape (Sine check)  
1525 Started Multi Adder Test.

1545  Relay #70 Panel F  
(Moth) in relay.

16100 Auton startet.  
1700 closed down.

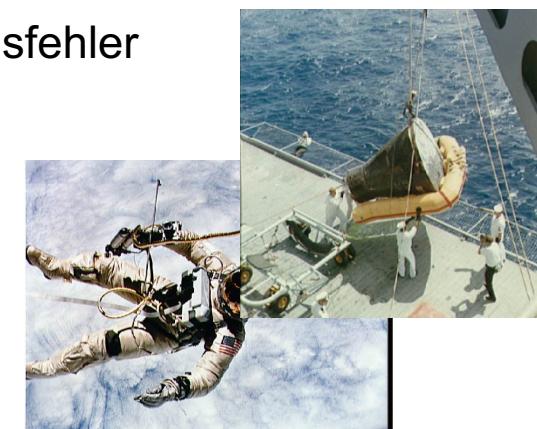
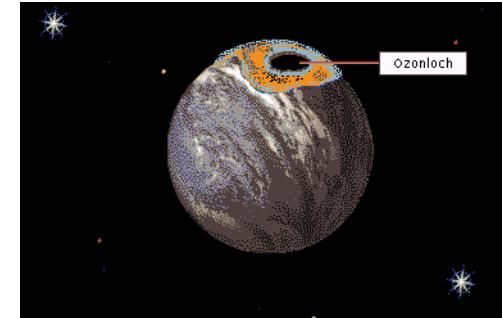
First actual case of bug being found.

<https://de.wikipedia.org/wiki/Programmfehler#/media/File:H96566k.jpg>



# AUSWIRKUNGEN VON SOFTWARE-FEHLERN

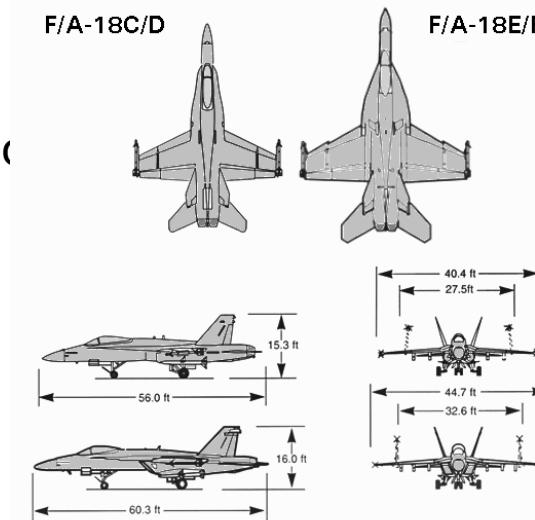
- NASA - Erdbeobachtungssatelliten 1979-1985
  - Ozonloch 7 Jahre (!) lang nicht erkannt
  - Ursache: Softwarefehler – Veränderung der Ozonschicht als Sensordrift durch automatische Nullpunkt Korrektur »herausgemittelt«
- ESA, Kourou, Franz. Guyana, 4. Juni 1996
  - Selbstzerstörung der Ariane 5 beim Jungfernflug 39 Sekunden nach dem Start
  - Ursache: Softwarefehler – Lageregelungssoftware aus Ariane 4 ohne Test gegen Start-Trajektorie der Ariane 5 wiederverwendet, dadurch Konvertierungsfehler
- Bemannte NASA-Raumkapsel Gemini V
  - Verfehlte ihren Landeplatz um ca. 160 Kilometer
  - Ursache: Softwarefehler – Rotation der Erde um die Sonne nicht berücksichtigt!





# F-18 AM ÄQUATOR: DOKUMENTATIONSFEHLER

- US Air Force, Programm zur Raketensteuerung
  - Effizienz bei Flug einer Rakete über Äquator:
  - Koordinaten nicht neu berechnet, sondern nur Vorzeichen geändert
  - Effekt: Rakete drehte sich bei Äquator-Überflug um eigene Achse
  - störte niemanden!
  - Software in Autopilot des Jägers F-18 übernahm
  - Äquator-Überflug: Maschine dreht sich auf den Kopf!
  - Glück gehabt: bereits im Simulator bemerkt





## T-MOBILE 2009: VIELE BETROFFENE ...



- 21. April 2009: Software-Fehler legt T-Mobile-Netz lahm
  - ab 16 Uhr Handys von Millionen T-Mobile-Kunden nicht erreichbar
  - Automatische Ansage:

**"Dieser Anschluss ist aus technischen Gründen vorübergehend nicht erreichbar. Bitte rufen sie später wieder an."**

- 19 Uhr: Techniker starten System neu
- 22 Uhr: Netz funktionierte wieder flächendeckend
- Ursache: **Softwarefehler** im Home Location Register (HLR). Dort sind Telefonnummern den SIM-Karten zugeordnet.



<http://www.spiegel.de/video/video-61903.html>



# SOFTWAREFEHLER IN SYSTEMEN (FAHRZEUGEN)

## Alles über Audi A4

Übersicht Tests News Erlkönige Gebrauchtwagen Usermeinungen Videos Neuwagen



Audi ruft 850.000 Autos zurück  
**Gefährlicher Airbag-Fehler im Audi A4**

In diesem Artikel

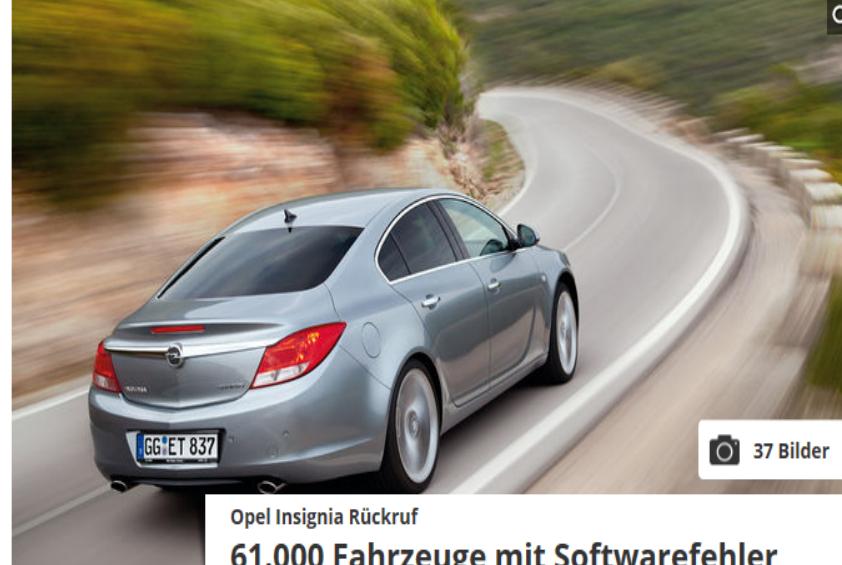
**1. Artikel**

2. Bildergalerie

<http://www.auto-motor-und-sport.de/news/audi-ruft-850000-autos-zurueck-gefaehrlicher-airbag-fehler-im-audi-a4-743967.html>, abgerufen am 16.04.2015

## Alles über Opel Insignia

Übersicht Tests News Erlkönige Gebrauchtwagen Usermeinungen Videos Neuwagen



Opel Insignia Rückruf  
**61.000 Fahrzeuge mit Softwarefehler**

In diesem Artikel

**1. Artikel**

2. Bildergalerie

<http://www.auto-motor-und-sport.de/news/opel-insignia-rueckruf-61-000-fahrzeuge-mit-softwarefehler-7483237.html>, abgerufen am 16.04.2015



# SOFTWAREFEHLER BEHINDERN UNS WEITERHIN ...

Fehler in der Software

## Zehntausenden Studenten droht Bafög-Verspätung

Seit dem ersten August gibt es höhere Bafög-Sätze und Freibeträge. Doch eine gängige Bearbeitungs-Software kennt diese Neuerungen nicht. Deshalb könnten viele Studenten zum Semesterstart erst einmal ganz ohne Geld dastehen.

24.08.2016

f Teilen    Twitter    x Teilen    E-mailen



<http://www.faz.net/aktuell/beruf-chance/campus/fehler-in-der-software-zehntausenden-studenten-droht-bafoga-verspaetung-14403775.html>, abgerufen am 12.03.2017

haft > Softwarefehler: Deutsche-Bank-Kunden kommen nur eingeschränkt an Geld

Softwarefehler

UPDATE 03.06.2016 18:20

## Deutsche-Bank-Kunden kommen nur eingeschränkt an Geld

Nachdem bei der Deutschen Bank online fehlerhafte Doppelbuchungen aufgetreten sind, können Kunden vielfach keine Terminals nutzen. VON SARAH KRAMER



<http://www.tagesspiegel.de/wirtschaft/softwarefehler-deutsche-bank-kunden-kommen-nur-eingeschraenkt-an-geld/13685606.html>, abgerufen am 12.03.2017



# ... UND GEFÄHRDEN UNS

MEDIZINPRODUKTE

0

von VERÖFFENTLICH 12. JANUAR 2017 · BEARBEITET 12. JANUAR 2017

## Roche informiert: Softwarefehler in der Diabetes-Management-App Accu-Chek Connect

[Accu-Chek](#) [Diabetes](#) [Roche Diabetes Care Deutschland GmbH](#)

Die Roche Diabetes Care Deutschland GmbH informiert über einen Softwarefehler in der Diabetes-Management-App Accu-Chek Connect.

Wie das Unternehmen mitteilt, wurde ein Softwarefehler in der Diabetes-Management-App Accu-Chek Connect der Versionen 1.2.0 und 1.2.2 (iOS und Android) entdeckt.



Symbolabbildung

<https://www.produktwarnung.eu/2017/01/12/roche-informiert-softwarefehler-in-der-diabetes-management-app-accu-chek-connect/4749>, abgerufen am 12.03.2017



Modelle der Marken Chevrolet, Buick und Cadillac sind von dem Rückruf betroffen.

Freitag, 09. September 2016

### Tödlicher Softwarefehler bei Airbags GM ruft 4,3 Millionen Autos zurück

Wiedereinmal machen die Airbags einem großen Autobauer Ärger. Mehr als vier Millionen Fahrzeuge muss GM in den USA zurückrufen. Der Fehler soll bereits einem Menschen das Leben gekostet haben.



Wegen Software-Problemen beordert die Opel-Mutter General Motors (GM) weltweit knapp 4,3 Millionen Fahrzeuge in die Werkstätten zurück. Bei Unfällen seien wegen des Defekts in seltenen Fällen die Airbags auf dem Fahrer- und Beifahrersitz nicht ausgelöst worden, teilte der in Detroit ansässige Konzern mit.

<http://www.n-tv.de/wirtschaft/GM-ruft-4-3-Millionen-Autos-zurueck-article18607516.html>,  
abgerufen am 12.03.2017



# WARUM TESTEN VON SOFTWARE IN AUTOMOTIVE?

## Self-Driving Uber Car kills Pedestrian in Ariona



<https://www.youtube.com/watch?v=iWGhXof45zl>

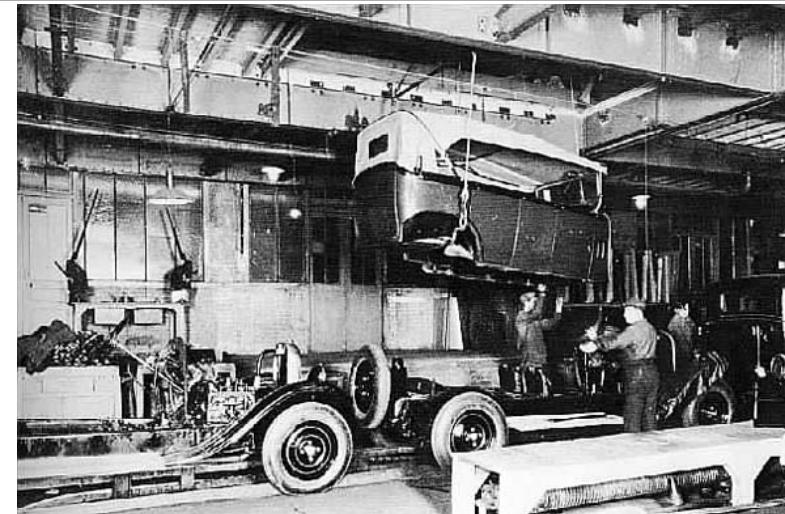


Quelle: <https://www.nytimes.com/2018/03/19/technology/uber-driverless-fatalty.html>, abgerufen 18.05.2018



# WIE TESTEN ANDERE?

- Automobil-Industrie
  - Eingangstest der Komponenten (Komponententest)
  - Laufende Zwischenkontrollen am Fließband (Integrationstest)
  - realitätsnaher Einsatztest (Systemtest)
  - Probefahrt des Kunden (Abnahmetest)
  - Renneinsatz (Performanztest, Lasttest)  
(Stabilität, Zuverlässigkeit, Robustheit)
  - Crashtest (Stresstest)





# AUSTESTEN?

- einfaches Programm soll getestet werden
  - drei ganzzahlige Eingabewerte
- Datentyp der Eingabewerte: 16 Bit Integerzahlen
  - $2^{16}$  unterschiedliche Werte möglich
- drei unabhängige Eingabewerten:
  - $2^{16} * 2^{16} * 2^{16} = 2^{48}$  Kombinationen.
- Jede dieser Kombinationen ist zu testen.
- Wie lange dauert es bei 100.000 Testfälle pro Sekunde?  
•



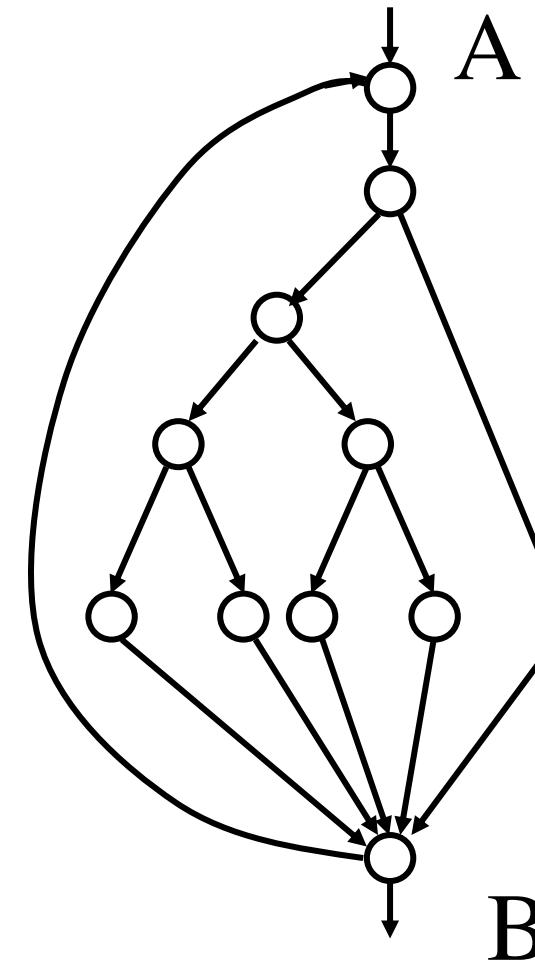
# AUSTESTEN?

- einfaches Programm soll getestet werden
  - drei ganzzahlige Eingabewerte
- Datentyp der Eingabewerte: 16 Bit Integerzahlen
  - $2^{16}$  unterschiedliche Werte möglich
- drei unabhängige Eingabewerten:
  - $2^{16} * 2^{16} * 2^{16} = 2^{48}$  Kombinationen.
- Jede dieser Kombinationen ist zu testen.
- Wie lange dauert es bei 100.000 Testfälle pro Sekunde?
- Es sind 281.474.976.710.656 Testfälle  
Dauer: ca. 90 Jahre



# AUSTESTEN?

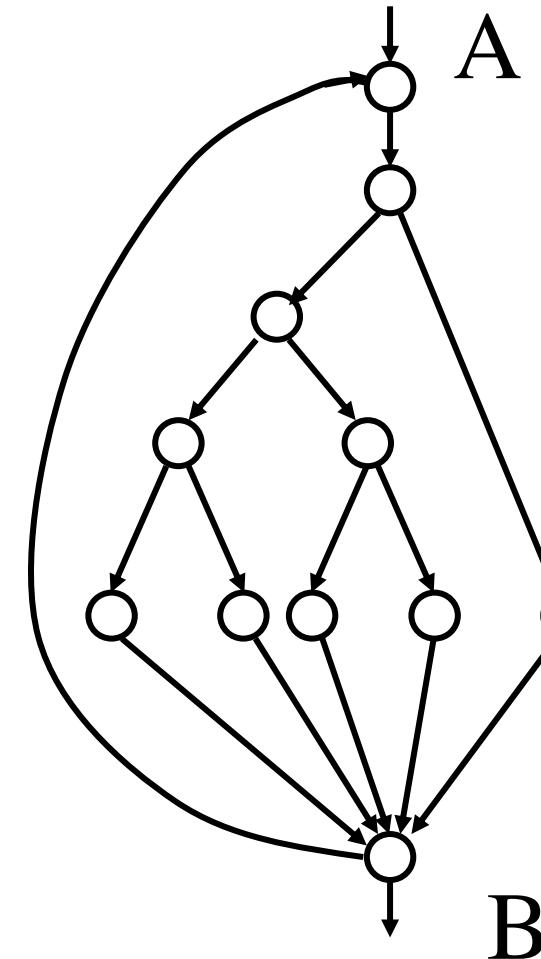
- Ein einfaches Programm soll getestet werden, das aus vier Verzweigungen (IF-Anweisungen) und einer umfassenden Schleife besteht und somit fünf mögliche Wege im Schleifenrumpf enthält.
- Unter der Annahme, dass die Verzweigungen voneinander unabhängig sind und bei einer Beschränkung der Schleifendurchläufe auf maximal 20, ergibt sich folgende Rechnung:  
$$5^1 + 5^2 + \dots + 5^{18} + 5^{19} + 5^{20}$$
- Es sollen alle Pfade mit maximal 20 Schleifendurchläufen getestet werden.
- Wie lange dauert das Austesten bei 100.000 Testfällen pro Sekunde?  
•





# AUSTESTEN?

- Ein einfaches Programm soll getestet werden, das aus vier Verzweigungen (IF-Anweisungen) und einer umfassenden Schleife besteht und somit fünf mögliche Wege im Schleifenrumpf enthält.
- Unter der Annahme, dass die Verzweigungen voneinander unabhängig sind und bei einer Beschränkung der Schleifendurchläufe auf maximal 20, ergibt sich folgende Rechnung:  
$$5^1 + 5^2 + \dots + 5^{18} + 5^{19} + 5^{20}$$
- Es sollen alle Pfade mit maximal 20 Schleifendurchläufen getestet werden.
- Wie lange dauert das Austesten bei 100.000 Testfällen pro Sekunde?
- Es sind 119.209.289.550.780 Testfälle  
Dauer: ca. 38 Jahre

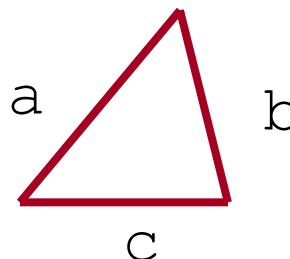




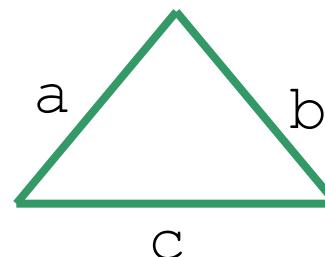
## ERSTE ÜBUNG (ZUM WARM WERDEN!)

Ein Programm ist zu testen, das 3 ganzzahlige positive Werte einliest und als Längen eines Dreiecks interpretiert.

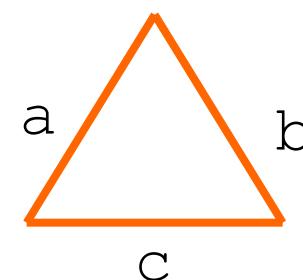
Das Programm gibt eine Meldung aus, ob es sich um ein **ungleichseitiges**, **gleichschenkliges** oder **gleichseitiges** Dreieck handelt.



$a \neq b \neq c$



$a = b \neq c$   
 $a \neq b = c$   
 $a = c \neq b$



$a = b = c$



## IHRE VORSCHLÄGE BITTE

- Eine schnelle Antwort auf die Fragen:
  - Wie viele Tests werden benötigt?
  - Mit welchen Testdaten würden Sie das Programmstück testen?



## MÖGLICHE TESTFÄLLE (1 VON 3)

Testfälle bestehen aus Testdaten und dem Soll-Ergebnis

1. 2,3,4 - zulässiges ungleichseitiges Dreieck
2. 2,2,2 - zulässiges gleichseitiges Dreieck
3. 2,2,1 - zulässiges gleichschenkliges Dreieck
- 4./5. 1,2,2 / 2,1,2 - Permutationen für gleichschenklige Dreiecke
6. 1,0,3 - kein Dreieck, eine Seitenangabe = 0
- 7./8. 0,1,3 / 1,3,0 - Permutationen
9. 5,-5,9 - kein Dreieck, eine Seitenangabe < 0
- 10./11. -5,5,9 / 5,9,-5 - Permutationen



## MÖGLICHE TESTFÄLLE (2 VON 3)

12.     1,2,3 - kein Dreieck  
Summe der beiden kürzeren Seiten = 3. Seitenlänge
- 13./14.  2,3,1 / 3,1,2 - Permutationen
15.     1,2,4 - kein Dreieck  
Summe der beiden kürzeren Seiten < 3. Seitenlänge
- 16./17.  2,4,1 / 4,1,2 - Permutationen
- 18./19.  0,0,0 - kein Dreieck oder Fehlermeldung  
alle Seiten = 0, zusätzlich 2 Seiten = 0 - Permutationen?
- 20.-22. Max\_int, Max\_int, Max\_int - zulässiges gleichseitiges Dreieck  
korrekte Dreiecksbestimmung beim Test mit maximalen Werten,  
zusätzliche Tests mit 2 oder 1 maximalem Wert



## MÖGLICHE TESTFÄLLE (3 VON 3)

- 23.-25. 1,1,4.4567 - Fehlermeldung »nicht ganzzahlige Werte«  
Permutationen? - zusätzlich mit 2 oder 3 Werten
- 26.-28. 1,1,& - Fehlermeldung »Eingabe von Buchstaben oder Sonderzeichen«  
Permutationen? - zusätzlich mit 2 oder 3 Werten
- 29./30. 1,2,3,4 / 2,3 - Fehlermeldung »falsche Anzahl von Werten«  
(wenn Eingabe möglich)
31. Max\_int/2 + 1, Max\_int/2 + 1, Max\_int/2 + 10 - zulässiges gleichschenkliges Dreieck  
(Überlauf oder richtige Berechnung?  
Bei  $a \leq b \leq c$ ; Prüfung der Dreiecksbedingung mit  $a+b > c$ , führt  $a+b$  zum Überlauf, s.a. Testfall 20)

Wie viele Tests hatten Sie überlegt?

in Anlehnung an  
Glenford J. Myers:  
Methodisches Testen von Programmen.  
7. Auflage 2001

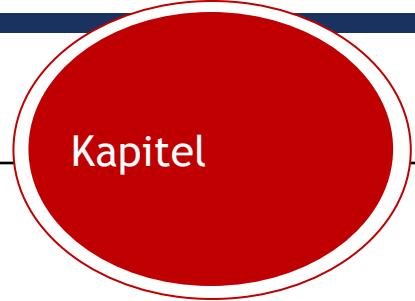


# SPITZ-, STUMPF- UND RECHTWINKLIGE DREIECKE

- Werden spitz-, stumpf- und rechtwinklige Dreiecke zusätzlich berücksichtigt, ergeben sich insgesamt 47 Testfälle.

**Resümee: Einfaches Problem, aber aufwendiger Test !**

E. H. Riedemann:  
Testmethoden für sequentielle und  
nebenläufige Software-Systeme.  
Teubner Verlag, Stuttgart 1997



## EINLEITUNG



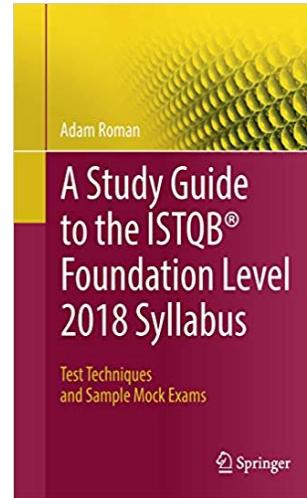
Certified Tester

Einstieg → Auswirkungen von Software-Fehlern

Anhang → Buchempfehlungen, Zeitschriften,  
Organisationen, Tagungen

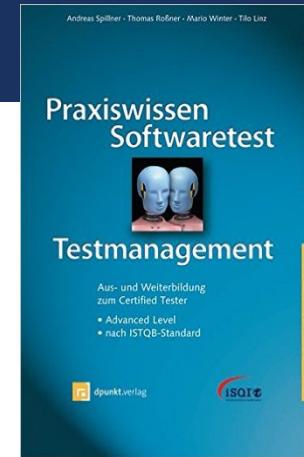
# BÜCHER ZUR ERGÄNZUNG (LEHRPLAN 2018)

- Oberbörsch, Klaus: Softwaretesten nach ISTQB Standard; Kompakt+ (Lehrplan 2018); 3. Februar 2019; Testen nach ISTQB Standard | Kompakt+: Auf Grundlage des aktuellen Lehrplans (Syllabus) von 2018 | ISBN: 9781727117424
- Roman, Adam: **A Study Guide to the ISTQB® Foundation Level 2018 Syllabus: Test Techniques and Sample Mock Exams**; Springer, Auflage: 1st ed. 2018 (13. November 2018); **ISBN-10:** 3319987399



# BÜCHER ZUR VERTIEFUNG

- Spillner, Andreas; Roßner, Thomas; Winter, Mario; Linz, Tilo:  
**Praxiswissen Softwaretest – Testmanagement**  
Aus- und Weiterbildung zum Certified Tester – Advanced Level  
nach ISTQB-Standard.  
dpunkt verlag, Heidelberg, Auflage: 4., überarb. u. erw. Aufl. (29. Mai 2014)



Ausführliche Beschreibung des Testmanagements – Vorbereitung auf die Certified Tester – Advanced Level – Testmanager Prüfung

- Liggesmeyer, Peter: **Software-Qualität**  
Spektrum Akademischer Verlag; Auflage: 2 (18. Juni 2009)



Aktualisierung des Buches „Modultest und Modulverifikation: State of the Art“ von 1990, ergänzt um Kapitel zum Test von eingebetteten und objektorientierten Systemen.

# BÜCHER ZUR VERTIEFUNG

- Westphal, Frank

Testgetriebene Entwicklung mit JUnit & FIT  
Wie Software änderbar bleibt  
dpunkt verlag, Auflage: 1 (10. Dezember 2012)

Inhalt: Unit Tests mit JUnit,  
Testgetriebene Programmierung,  
Refactoring, Häufige Integration,  
Testfälle schreiben von A bis Z,  
Isoliertes Testen durch Stub- und Mock-Objekte,  
Akzeptanztests mit FIT (Framework for Integrated Test)



- Pol, Martin; Koomen, Tim; Spillner, Andreas:

Management und Optimierung  
des Testprozesses.

dpunkt verlag, 2002 (2. Auflage)

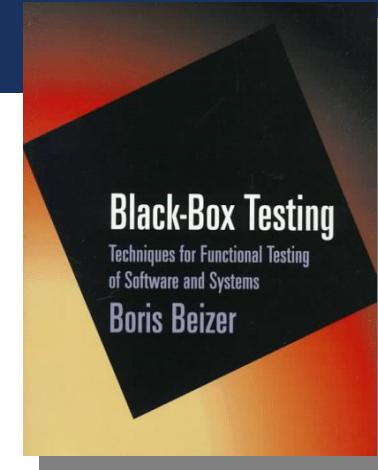
Ausführliche Beschreibung von TMap (Test Management Approach) und TPI (Test Process Improvement).  
Gut geeignet zur Analyse des bestehenden Testprozesses und zur schrittweisen Verbesserung.



# BÜCHER ZUR VERTIEFUNG

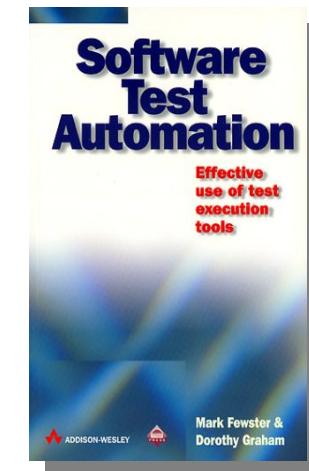
- Beizer, Boris:  
Black-Box Testing –  
Techniques for Functional Testing  
of Software and Systems.  
John Wiley & Sons, 1995

Ausführliche Beschreibung der Black-Box Verfahren.



- Fewster, Mark; Graham, Dorothy:  
Software Test Automation: effective  
use of test execution tools.  
Addison Wesley, 1999

Ausführliche Beschreibung zur Struktur und  
zum Aufbau der Testautomatisierung, mit guten  
und schlechten Beispielen.

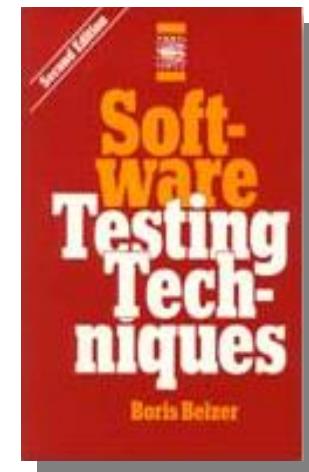


# BÜCHER ZUR VERTIEFUNG

- Myers, Glenford J.:  
Methodisches Testen von Programmen  
Oldenbourg, 2005 (8.Auflage)  
  
Klassischer Überblick. Für den Einstieg geeignet,  
besonders zur Methodik der Testfallerstellung.  
(Erstauflage „The Art of Software Testing“, 1979)



- Beizer, Boris  
Software Testing Techniques  
1990 (2.Auflage)  
  
Ausführliche Behandlung vor allem  
der White-Box-Testtechniken. Besonders  
geeignet für frühe Testphase, weniger  
für Systemtests.



# BÜCHER ZUR VERTIEFUNG

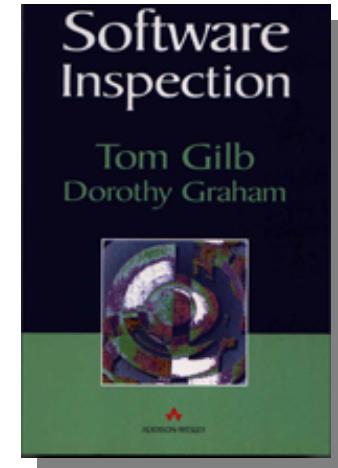
- Frühauf, Karol; Ludewig, Jochen;  
Sandmayr, Helmut  
Software-Prüfung  
Vdf Hochschulverlag, 2006

Beschreibung von Reviews und  
anderen Test- und Prüfverfahren.



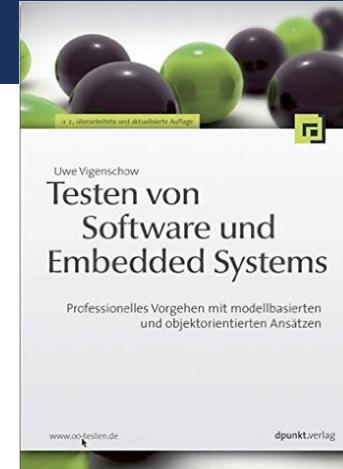
- Gilb, Tom; Graham, Dorothy  
Software Inspection  
Addison-Wesley, 1993

Ausführliche Beschreibung von Inspektionen.



# BÜCHER ZUR VERTIEFUNG

- Vigenschow, Uwe  
Testen von Software und Embedded Systems;  
dpunkt Verlag, Auflage: 2., überarb. u. akt. Aufl. 2010  
  
Beschreibung zum systematischen Vorgehen mit modellbasierten  
und objektorientierten Ansätzen.



- Link, Johannes  
Softwaretests mit JUnit.  
Techniken der testgetriebenen Entwicklung  
dpunkt Verlag; Auflage: 2 (10. Dezember 2012)  
  
Ausführliche Beschreibung des Test-First-Ansatz,  
allerdings ohne systematische Herleitung der Testfälle



# BÜCHER ZUR VERTIEFUNG

- Sneed, Harry M.; Winter, Mario  
Testen objektorientierter Software  
Das Praxishandbuch für den  
Test objektorientierter Client/Server Systeme  
Carl Hanser Verlag, München, 2001  
  
Guter Einstieg in den Test von OO-Software  
Leider z.Zt. vergriffen

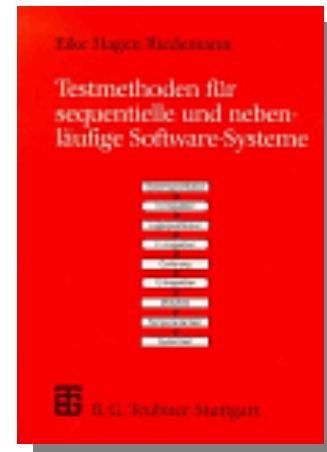


- Riedemann, Eike Hagen  
Testmethoden für sequentielle und  
nebenläufige Software-Systeme.  
Teubner, Stuttgart, 512 S., 1997  
  
Leider vergriffen  
Gute Grundlagenbeschreibung mit Schwerpunkten  
auf dem Test von nebenläufiger Software

vollständig als PDF-Dateien herunterladbar:  
<http://ls10-www.cs.uni-dortmund.de/~riedemann/Homepage/PRUEF.html>

siehe auch:  
<http://www-dssz.informatik.tu-cottbus.de/information/testen/zusammenfassung-testmethoden.pdf>

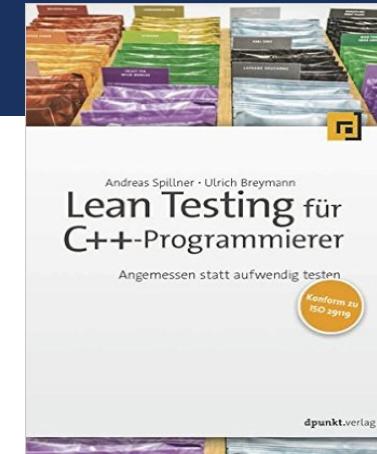
<http://www.abebooks.co.uk/9783519022749/Testmethoden-sequentielle-neben%C3%A4ufige-Software-Systeme-XLeit%C3%A4den-3519022745/plp>



# BÜCHER ZUR ERGÄNZUNG

- Spillner, Andreas; Breymann, Ulrich:  
**Lean Testing für C++-Programmierer**; Angemessen statt aufwendig testen; dpunkt.verlag GmbH; Auflage: I (2. Juni 2016);  
**ISBN-10:** 3864903084.

Ausführliche Beschreibung des Testens auf Komponentenebene.



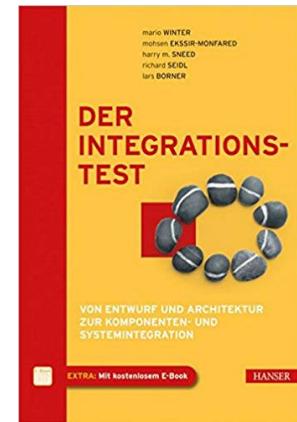
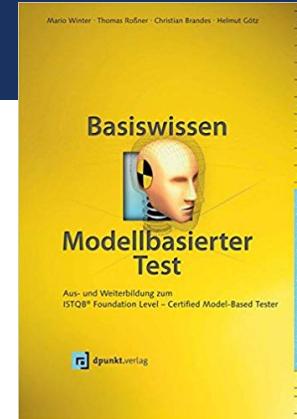
- Daigl, Matthias; Glunz, Rolf: **ISO 29119: Die Softwaretest-Normen verstehen und anwenden**; dpunkt.verlag GmbH; Auflage: I., Aufl. (28. Januar 2016).

Guter Zugang zur Softwaretestnorm ISO 29119



# BÜCHER ZUR ERGÄNZUNG

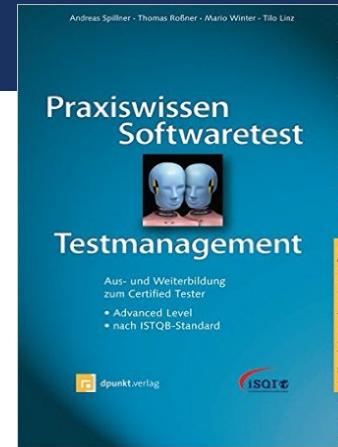
- Winter, M.; Roßner, T.; Brandes, C.; Götz, H.:  
**Basiswissen Modellbasierter Test**; 2., akt. und erw. Aufl.,  
dpunkt Verlag, Heidelberg, 2016.
- Winter, M.; Ekssir-Monfared,M.; Sneed, H.M.; Seidl, R.; Borner, L.:  
**Der Integrationstest – Von Entwurf und Architektur zur Komponenten- und Systemintegration**; Carl Hanser Verlag,  
München 2013.



# BÜCHER ZUR ERGÄNZUNG

- Spillner, Andreas; Roßner, T.; Winter, M.; Linz, T.: **Praxiswissen Softwaretest – Testmanagement;** 4. überarb. u. erw. Aufl. (29. Mai 2014); dpunkt verlag.

Aus- und Weiterbildung zum Certified Tester – Advanced Level nach ISTQB Standard.



- Bath, Graham; McKay, Judy:: **Praxiswissen Softwaretest - Test Analyst und Technical Test Analyst:** Aus- und Weiterbildung zum Certified Tester - Advanced Level nach ISTQB-Standard; 3., überarb. Auflage Februar 2015, dpunkt verlag.

Aus- und Weiterbildung zum Certified Tester - Advanced Level nach ISTQB Standard



# ZEITSCHRIFTEN ZUM THEMENGEBIET

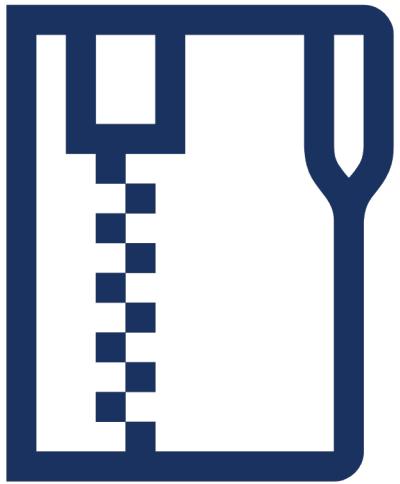
- Springer, Software Quality Journal:  
<http://www.springer.com/computer/swe/journal/11219>
- ASQF, SQ-Magazin: <https://www.asqf.de/sq-magazin/>
- The Journal of Software Testing, Verification and Reliability,  
John Wiley & Sons Ltd.  
[http://onlinelibrary.wiley.com/journal/10.1002/\(ISSN\)1099-1689](http://onlinelibrary.wiley.com/journal/10.1002/(ISSN)1099-1689)
- Better Software  
SQE Publications  
<http://www.stickyminds.com/BetterSoftware/magazine.asp>
- Professional Tester, Test Publishing Ltd.  
<http://www.professionaltester.com/>
- Softwaretechnik-Trends  
(Mitteilungen der GI Fachgruppe TAV u.a.)  
<http://pi.informatik.uni-siegen.de/stt/>



# ORGANISATIONEN

- Gesellschaft für Informatik e.V. -  
Fachgruppe TAV -  
Test, Analyse und Verifikation von Software  
(regelmäßige Treffen der Fachgruppe)  
<http://fg-tav.gi.de/>
- ASQF - Arbeitskreis Software Qualität und Fortbildung e.V.  
(Fachgruppentreffen und weitere Veranstaltungen)  
<https://www.asqf.de/fachgruppen/software-test/>
- GTB - German Testing Board e.V.  
(ISTQB Certified Tester Programm )  
<http://www.german-testing-board.info>





# GRUNDLAGEN DES SOFTWARETESTENS

## Grundlagen des Softwaretestens



Begriffe und Motivation

Grundsätze des Softwaretestens

Testprozess

Testfälle, Erwartete Ergebnisse und Testorakel

Psychologie des Testens

# WAS GILT ALS FEHLER ODER MANGEL?

- Situation fehlerhaft? → vorab die erwartete Situation festlegen
- **Fehler:**
  - Nichterfüllung einer festgelegten Anforderung
  - Abweichung zwischen Istergebnis und erwartetem Ergebnis
- **Mangel:**
  - Anforderung / berechtigte Erwartung nicht angemessen erfüllt
  - Nichtkonformität bzgl. beabsichtigtem oder festgelegtem Gebrauch



# URSACHENKETTE FÜR FEHLER

- **Fehler/Mangel**
  - seit Fertigstellung der Software vorhanden,
  - kommt erst bei Ausführung der Software zum Tragen.
- **Fehlerwirkung (failure)**: Das sichtbare Auftreten des Fehlers
- **Fehlerzustand (defect)**: Ursache einer Fehlerwirkung
- **Fehlhandlung (error)** einer Person: Ursache eines Fehlerzustands
- **Fehlermaskierung**
  - ein Fehlerzustand verhindert die Aufdeckung eines anderen

Was soll sichergestellt werden?

Fehlerwirkung (Failure)



Nach außen sichtbare Fehlverhalten

Fehlerzustand (Bug)



Zustand in der Anwendung der zu einer Fehlerwirkung führen kann.

Fehlhandlung (Error, Mistake)

Irrtum bei der Software-Entwicklung.

Fehlermaskierung

Wirkung eines Fehlers (Fehlerzustand) ist nach außen nicht sichtbar, weil er durch einen weiteren Fehler verborgen (überlagert / maskiert) wird.

# FEHLERWIRKUNG - DEFINITION



## ■ Fehlerwirkung (*failure*)

- Ein Ereignis in welchem eine Komponente oder ein System eine geforderte Funktion nicht im spezifizierten Rahmen ausführt. [ISO 24765, Glossar 3.2]
  - Wirkung eines Fehlerzustands, die nach »außen« sichtbar auftritt.
  - Abweichung zwischen erwartetem Ergebnis und Istergebnis.
  - Abweichung eines Systems von der erwarteten Lieferung/Leistung.
- 
- Anmerkungen zu Fehlerwirkung:
    - (selten) auch durch Höhenstrahlung, elektromagnetische Felder oder Hardwarefehler hervorgerufen.
    - Kann falscher Ausgabe, zu langsamer Ausführung oder Abbruch führen.

# FEHLERZUSTAND - DEFINITION



## Fehlerzustand (*defect*)

1. Eine Unzulänglichkeit oder ein Mangel in einem Arbeitsergebnis, sodass es seine Anforderungen oder Spezifikationen nicht erfüllt [Glossar V3.2]
2. Inkorrektes Teilprogramm
  - inkorrekte Anweisung
  - Inkorrekte Datendefinition
3. Zustand eines Programms, der ggf. (z.B. hohe Last) eine geforderte Funktion beeinträchtigt oder zu Fehlerwirkung führt

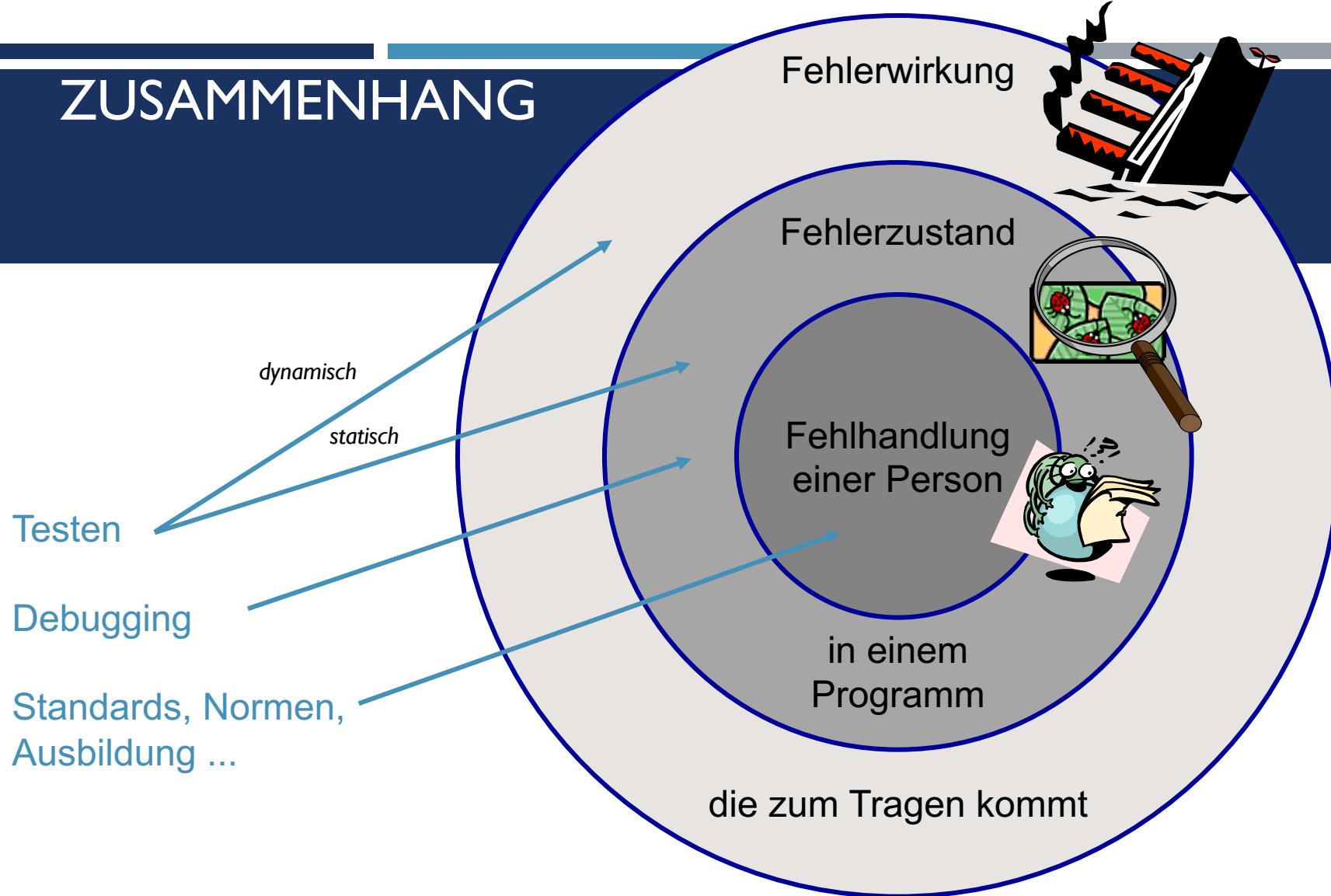
# FEHLHANDLUNG - DEFINITION



## Fehlhandlung (*error*)

1. **Die menschliche Handlung, die zu einem falschen Ergebnis führt. [IEEE 610, Glossar 3.2]**
2. Die menschliche Handlung
  - Entwickler: führt zu Fehlerzustand in der Software
  - Anwenders: Fehlbedienung mit unerwünschtem Ergebnis
3. Unwissentlich, versehentlich oder absichtlich ausgeführte Handlung oder Unterlassung, die ggf. zu einer Beeinträchtigung der Softwarefunktion führt

# ZUSAMMENHANG





## TESTEN – DEFINITION (NACH GLOSSAR 3.2)

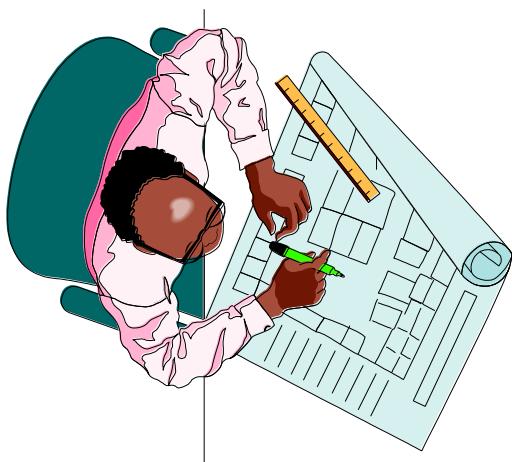
Der Prozess, der aus allen Aktivitäten des Lebenszyklus besteht (sowohl statisch als auch dynamisch), die sich mit

- der Planung,
- Vorbereitung und
- Bewertung

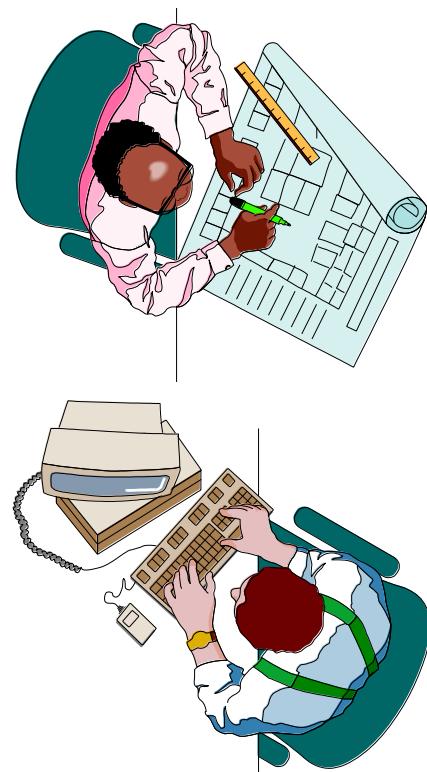
eines Softwareprodukts und dazugehöriger Arbeitsergebnisse befassen.

Ziel des Prozesses: Sicherstellen, dass

- diese allen festgelegten Anforderungen genügen,
- ihren Zweck erfüllen und
- etwaige Fehlerzustände finden.



# ZIELE DES TESTENS



## ■ Mögliche Ziele des Testens:

- Arbeitsergebnisse bewerten: Anforderungen, User-Stories, Architekturdesign und Code
- Vollständigkeit des Testobjekts prüfen
- Fehlerzustände vermeiden
- Die Risikostufe mangelhafter Softwarequalität reduzieren (z.B. für unentdeckte Fehlerwirkungen)
- Validieren des Testobjekts gegen Erwartungen der Stakeholder
- Den Stakeholdern Informationen für fundierte Entscheidungen geben, insbesondere bezüglich des Qualitätsniveaus des Testobjekts
- Vertrauen in das Qualitätsniveau der Software schaffen

# VALIDIERUNG UND VERIFIZIERUNG - DEFINITIONEN



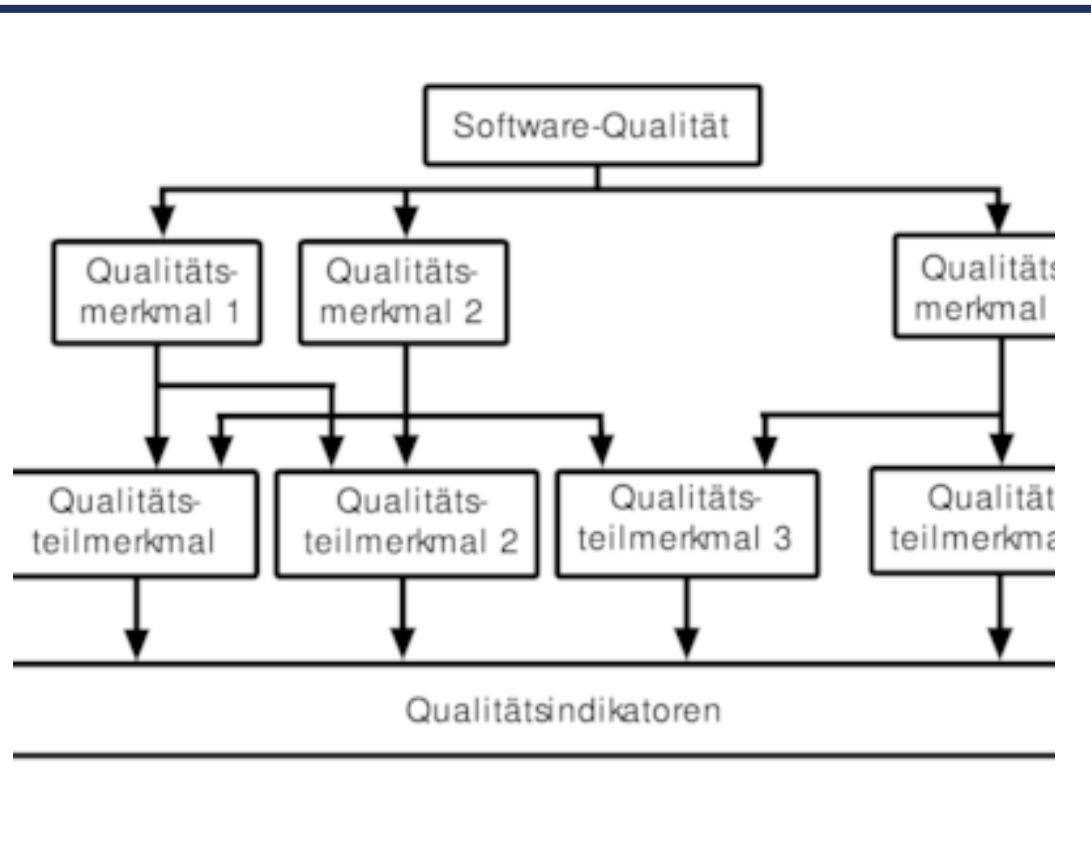
## Validierung

- Bestätigung durch Bereitstellung eines objektiven Nachweises, dass die Anforderungen für einen spezifischen beabsichtigten Gebrauch oder eine spezifische beabsichtigte Anwendung erfüllt worden sind. [ISO 9000, Glossar 3.2]
- Haben wir das **richtige System** realisiert?

## Verifizierung

- Bestätigung durch Bereitstellung eines objektiven Nachweises, dass festgelegte Anforderungen erfüllt worden sind. [ISO 9000, Glossar 3.2]
- Haben wir das **System richtig** realisiert?

# WAS IST SOFTWAREQUALITÄT?



## Qualität:

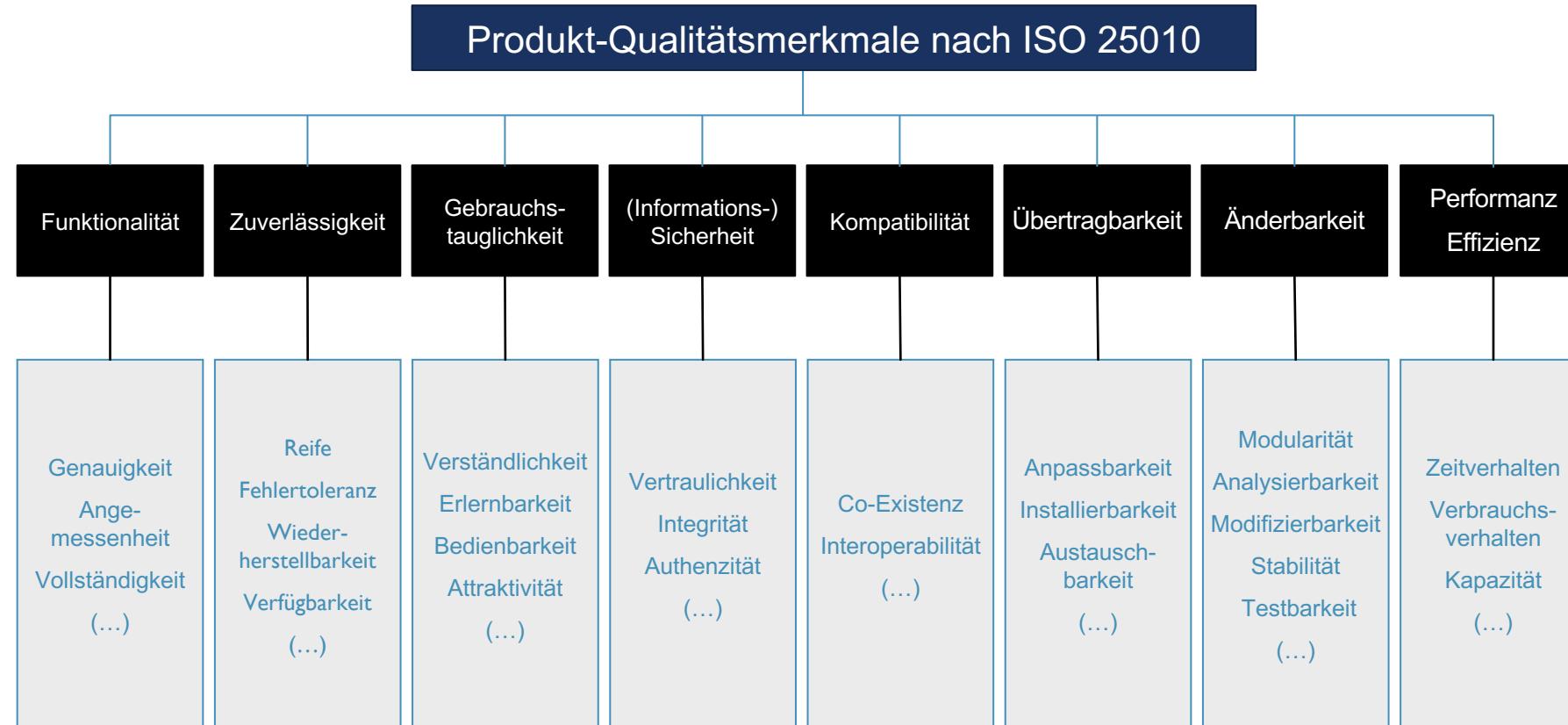
- Der Grad, in dem ein System, eine Komponente oder ein Prozess die Kundenerwartungen und -bedürfnisse erfüllt. [IEEE 610, Glossar 3.2]
- Der Grad, in dem ein Satz inhärenter Merkmale Anforderungen erfüllt. [ISO 9000:2000]

## Softwarequalität:

- Gesamtheit der Funktionalitäten und Merkmale eines Softwareprodukts, die sich auf dessen Eignung beziehen, festgelegte oder vorausgesetzte Erfordernisse zu erfüllen. [Glossar 3.2]
- Qualitätsmerkmale beziehen sich auf Anforderungen
  - Funktionale Anforderungen (Fachlichkeit, Funktionen, Schnittstellen, ...)
  - Nicht-Funktionale Anforderungen (Qualitäts- und Realisierungsanforderungen, Projektspezifische Anforderungen, ...)



# QUALITÄT NACH ISO 25010



**ISO 25010: Software-Engineering - Qualitätskriterien und Bewertung von Softwareprodukten (SQuaRE) - Qualitätsmodell und Leitlinien**

## GRUNDLAGEN DES SOFTWARETEST ENS



Begriffe und Motivation

Grundsätze des Softwaretestens

Testprozess

Testfälle, erwartete Ergebnisse und Testorakel

Psychologie des Testens

# GRUNDSÄTZE ZUM TESTEN (IM ÜBERBLICK)



In den letzten 50 Jahren haben sich folgende Grundsätze zum Testen herauskristallisiert und können somit als Leitlinien dienen:

**Grundsatz 1: »Testen zeigt die Anwesenheit von Fehlerzuständen, nicht deren Abwesenheit«**

**Grundsatz 2: »Vollständiges Testen ist nicht möglich«**

**Grundsatz 3: »Frühes Testen spart Zeit und Geld«**

**Grundsatz 4: »Häufung von Fehlerzuständen«**

**Grundsatz 5: »Vorsicht vor dem Pestizid-Paradoxon«**

**Grundsatz 6: »Testen ist kontextabhängig«**

**Grundsatz 7: »Trugschluss: „Keine Fehler“ bedeutet ein brauchbares System«**

# TESTAUFWAND?

- »Testen ist ökonomisch sinnvoll, solange die Kosten für das Finden und Beseitigen eines Fehlers im Test niedriger sind als die Kosten, die mit dem Auftreten eines Fehlers bei der Nutzung verbunden sind.«

M. Pol, T. Koomen, A. Spillner:  
Management und Optimierung des  
Testprozesses. dpunkt-Verlag, 2. Auflage, 2002

- »Ein guter Test ist wie eine Haftpflichtversicherung: Er kostet richtig Geld, lässt aber den Projektleiter und den Kunden ruhig schlafen. Zum guten Schlaf gehört auch eine gute Versicherung, die alle möglichen Risiken abdeckt. Zum Vertrauen in die Software gehört ein guter Test, der die ganze Produktionswirklichkeit abdeckt.«

Siedersleben, J. (Hrsg): Softwaretechnik,  
Praxiswissen für Softwareingenieure. 2. Auflage,  
Hanser, 2003

# WEITERES ZUM TESTEN (I)

- Testaufwand in der Praxis:  
25% bis 50% des Entwicklungsaufwands
- Testintensität und -umfang in Abhängigkeit vom Risiko und der Kritikalität festlegen
- Fehler verursachen wirklich hohe Kosten
  - Geschätzte Verluste durch Softwarefehler in Mittelstands- und Großunternehmen in Deutschland ca. 84,4 Mrd. Euro p.a.
  - Produktivitätsverluste durch Computerausfälle aufgrund fehlerhafter Software ca. 2,6% des Umsatzes - 70 Mrd. Euro p.a.

Studie der LOT Consulting Karlsruhe  
IT-Services 3/2001, S. 31

## WEITERES ZUM TESTEN (2)

Testen unterliegt immer beschränkten Ressourcen (besonders Zeit, wenn Testen erst am Ende der Entwicklung in Angriff genommen wird)

Besonders wichtig:

- Adäquate (zum Testobjekt und den Qualitätszielen passende) Testverfahren auswählen
- Unnötige Tests (die keine neuen Erkenntnisse liefern) vermeiden
- Sowohl Positiv- als auch Negativtests berücksichtigen
- Auch Tests auf Funktionalität, die nicht gefordert (evtl. sogar absichtlich schädlich) sind, müssen einbezogen werden

## GRUNDLAGEN DES SOFTWARETEST ENS



Begriffe und Motivation

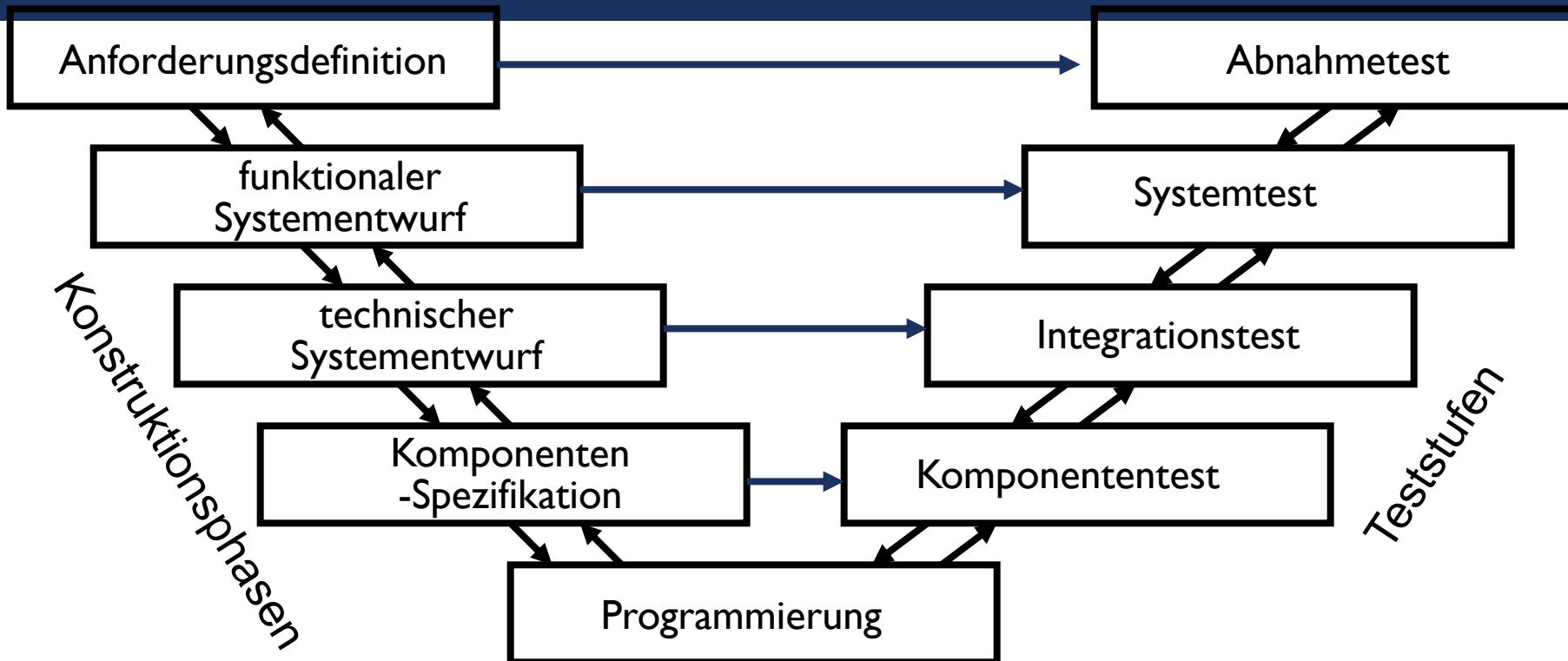
Grundsätze des Softwaretestens

Testprozess

Testfälle, Sollwerte und Testorakel

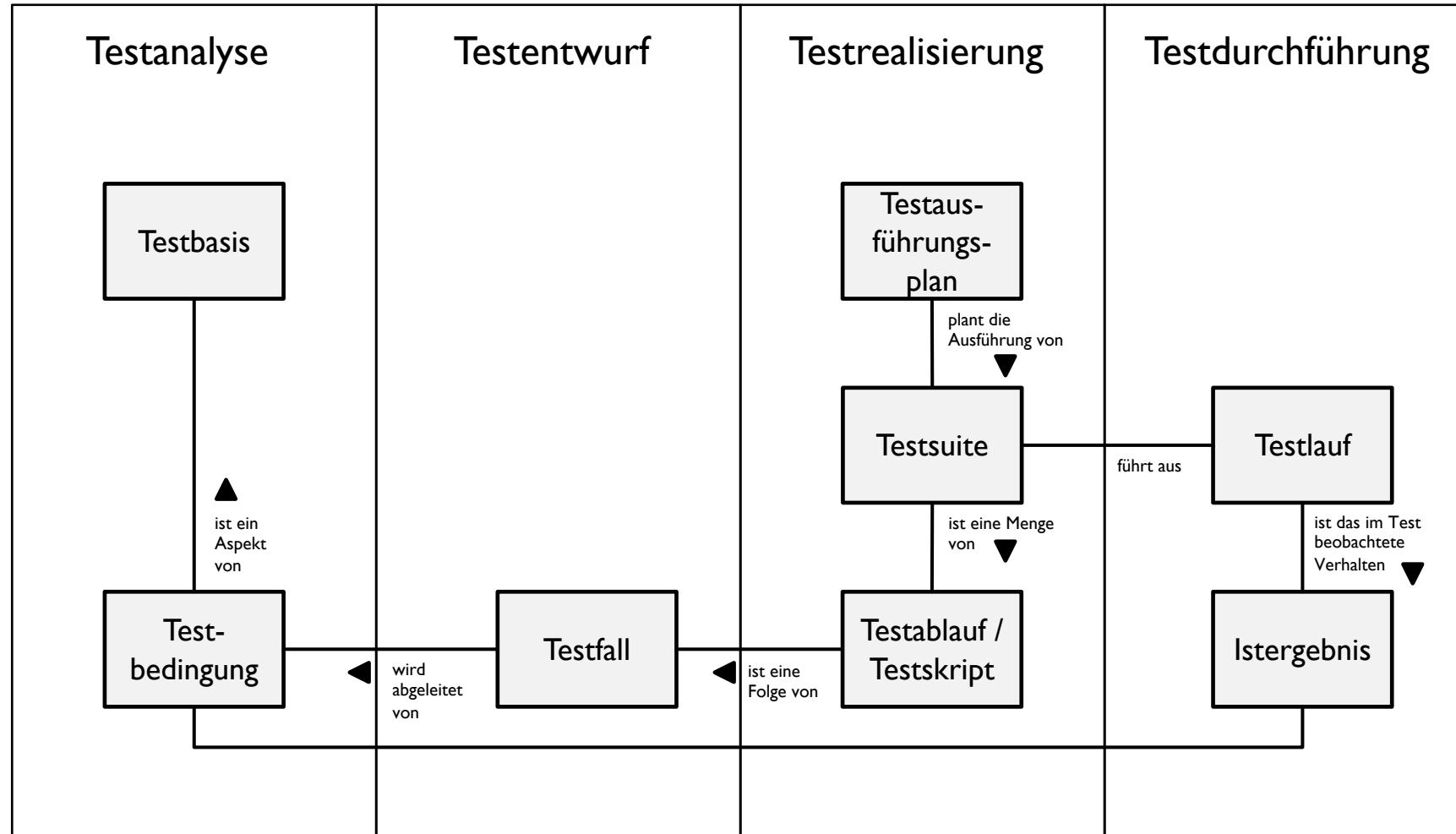
Psychologie des Testens

# ALLGEMEINES V-MODELL



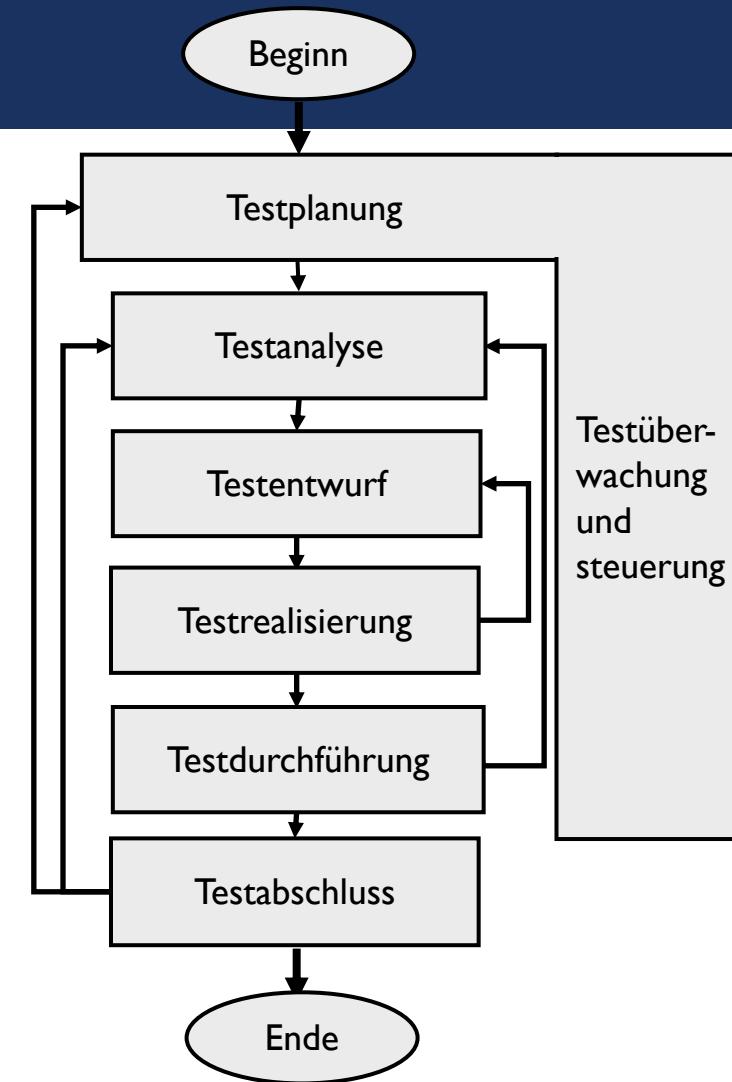
**Legende**  
→ Testfälle basieren auf den entsprechenden Dokumenten

# RÜCKVERFOLGBARKEIT

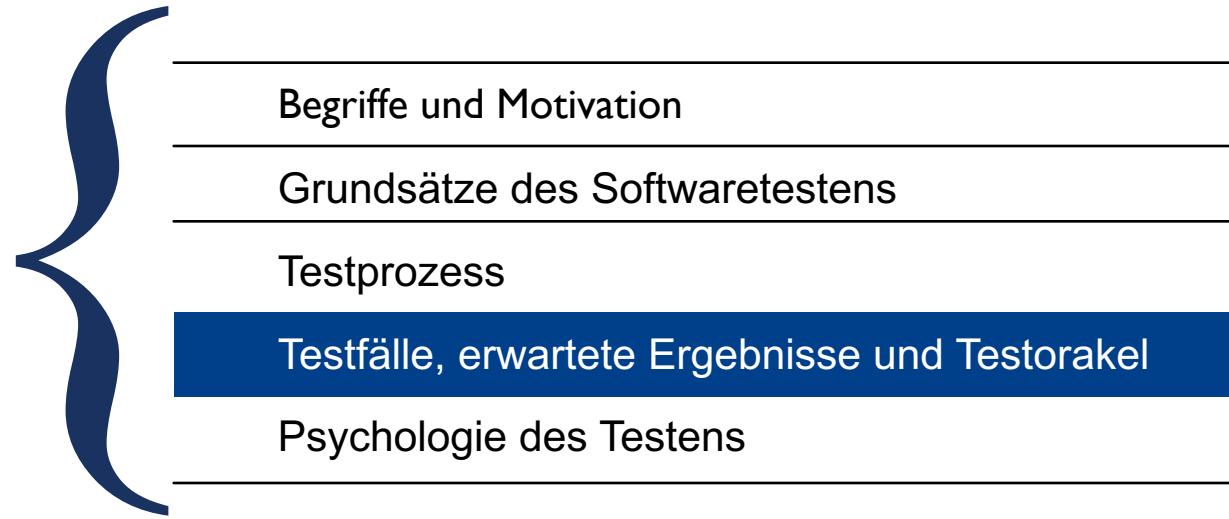


# AKTIVITÄTEN DES TESTPROZESSES

- Testplanung
- Testüberwachung und -steuerung
- Testanalyse
- Testentwurf
- Testrealisierung
- Testdurchführung
- Testabschluss
- Diese Aktivitäten werden z.T. zeitlich überlappend oder parallel ausgeführt.
- Der Testprozess ist für jede Teststufe geeignet zu gestalten.



## GRUNDLAGEN DES SOFTWARETEST ENS



# KRITERIEN FÜR TESTFÄLLE

- Testfälle zur Prüfung des erwarteten, spezifizierten Verhaltens
  - „Positivtest“
  - erwartete Eingaben bzw. erwartete Bedienung
- Testfälle zur Prüfung der spezifizierten Ausnahme- und Fehlersituationen
  - „Negativtest“
  - Fehleingaben bzw. Fehlbedienung (nicht immer einfach, z.B. Überlast)
  - Einstellung des Testers: Test soll zeigen, dass eine Komponente oder ein System nicht funktioniert
- Testfälle zur Prüfung von unspezifizierten Ausnahmen
  - „Negativtest“ bzw. „Robustheitstest“
  - unerwartete Fehleingaben bzw. unerwartete Fehlbedienung



# TESTSPEZIFIKATION – ABSTRAKTE UND KONKRETE TESTFÄLLE (1)

## Beispiel

Eine Firma hat ein Programm in Auftrag gegeben, das die Weihnachtsgratifikation der Mitarbeiter in Abhängigkeit von der Firmenzugehörigkeit berechnen soll.

In der Beschreibung der Anforderungen findet sich folgende Textpassage:

»Mitarbeiter erhalten ab einer Zugehörigkeit zur Firma von mehr als drei Jahren 50 % des Monatsgehalts als Weihnachtsgratifikation. Mitarbeiter, die länger als fünf Jahre in der Firma tätig sind, erhalten 75 %. Bei einer Firmenzugehörigkeit von mehr als acht Jahren wird eine Gratifikation von 100 % gewährt.«

Wie sehen adäquate Testfälle aus?



## TESTSPEZIFIKATION – ABSTRAKTE UND KONKRETE TESTFÄLLE (2)

- Aus dem Text lassen sich folgende Zusammenhänge für die Gewährung der Gratifikation in Abhängigkeit der Firmenzugehörigkeit aufstellen:

Firmenzugehörigkeit  $\leq 3$  ergibt Gratifikation = 0 %

$3 < \text{Firmenzugehörigkeit} \leq 5$  ergibt Gratifikation = 50 %

$5 < \text{Firmenzugehörigkeit} \leq 8$  ergibt Gratifikation = 75 %

Firmenzugehörigkeit  $> 8$  ergibt Gratifikation = 100 %



# TESTSPEZIFIKATION – ABSTRAKTE UND KONKRETE TESTFÄLLE (3)

Abstrakter Testfall	1	2	3	4
Eingabewert x (Firmenzugehörigkeit)	$x \leq 3$	$3 < x \leq 5$	$5 < x \leq 8$	$x > 8$
Erwartetes Ergebnis (Gratifikation in %)	0	50	75	100

Konkreter Testfall	1	2	3	4
Eingabewert x (Firmenzugehörigkeit)	2	4	7	12
Erwartetes Ergebnis (Gratifikation in %)	0	50	75	100

Anmerkung: Es wurde keine Rand-, Vor- und Nachbedingungen vermerkt,  
die Testfälle wurden nicht systematisch hergeleitet,  
nur positive Tests mit erwarteten Ergebnissen

# ERWARTETE ERGEBNISSE UND TESTORAKEL

- Nach jeder Testfalldurchführung: Fehlerwirkung ja / nein?.
  - Ist-Ergebnis mit erwartetem Ergebnis vergleichen
- Das erwartete Ergebnis des Testobjekts muss vorab spezifiziert werden.
  - Aufgabe des Testers: geeignete Quellen finden
- "Testorakel" wird befragt, um die erwarteten Ergebnisse vorherzusagen.
- Haben Sie gemerkt, dass der Fall „Betriebszugehörigkeit <= 3 Jahre“ nicht spezifiziert wurde?

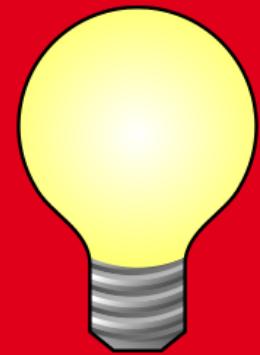


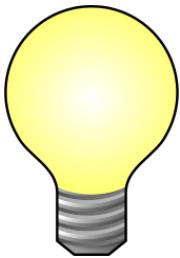
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

04.02.2021

# Programmieren im Grossen VI

Testen





## AGENDA

Einführung ins Thema

Erste Überlegungen

Black-Box, White-Box, Grey-Box

Äquivalenzklassen- und Randwertanalyse

Kombinierung von Testfällen (Testvektoren)

Zustandsbasierte Tests

Tests auf verschiedenen Abstraktionsebenen

Testvorbereitung und -durchführung

Fazit

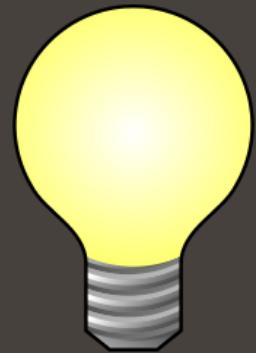


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# 01

## EINFÜHRUNG INS THEMA

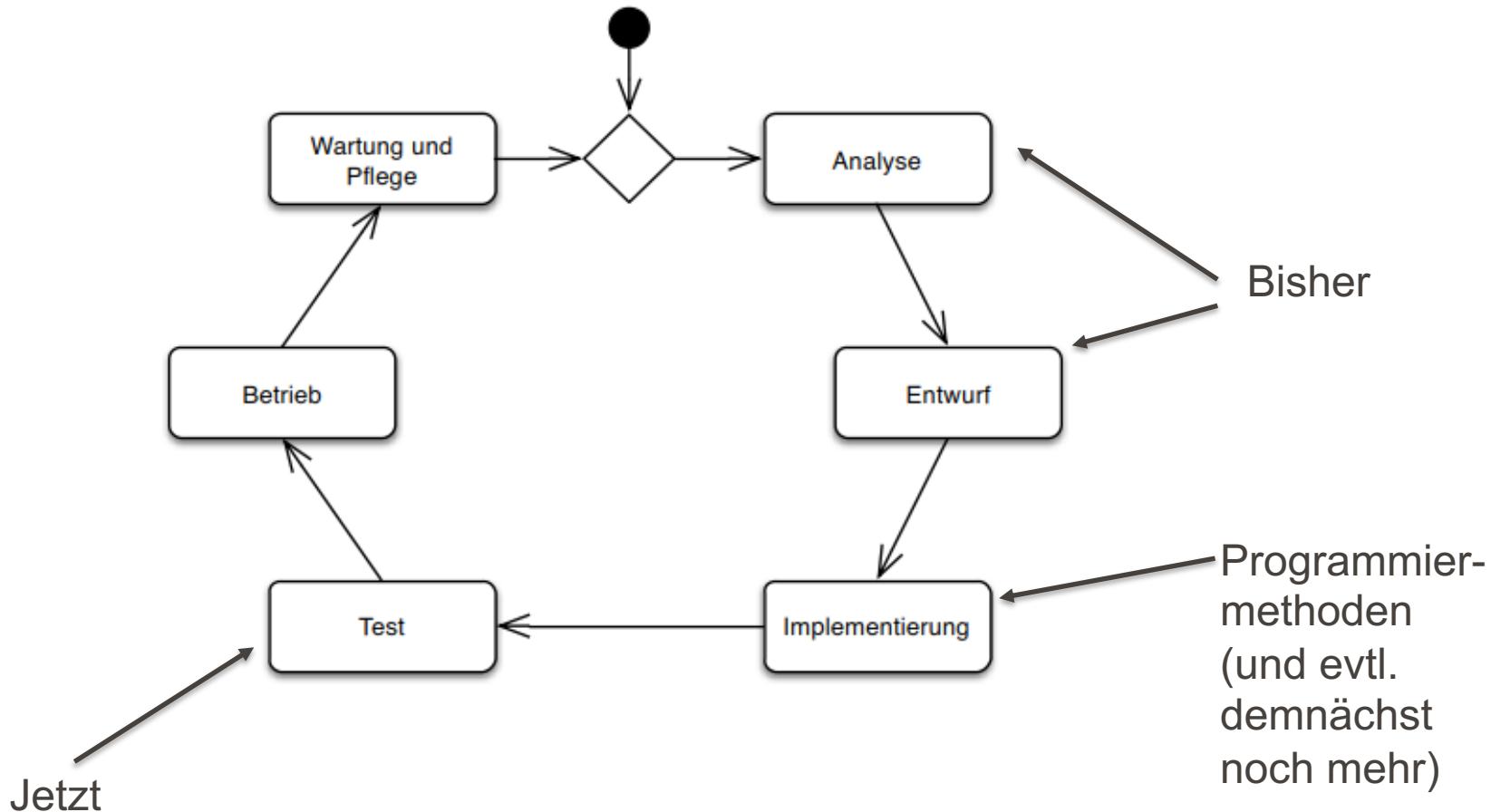
Ziel:  
Die Eckpunkte des Themas kennenlernen





# WORUM GEHT'S?

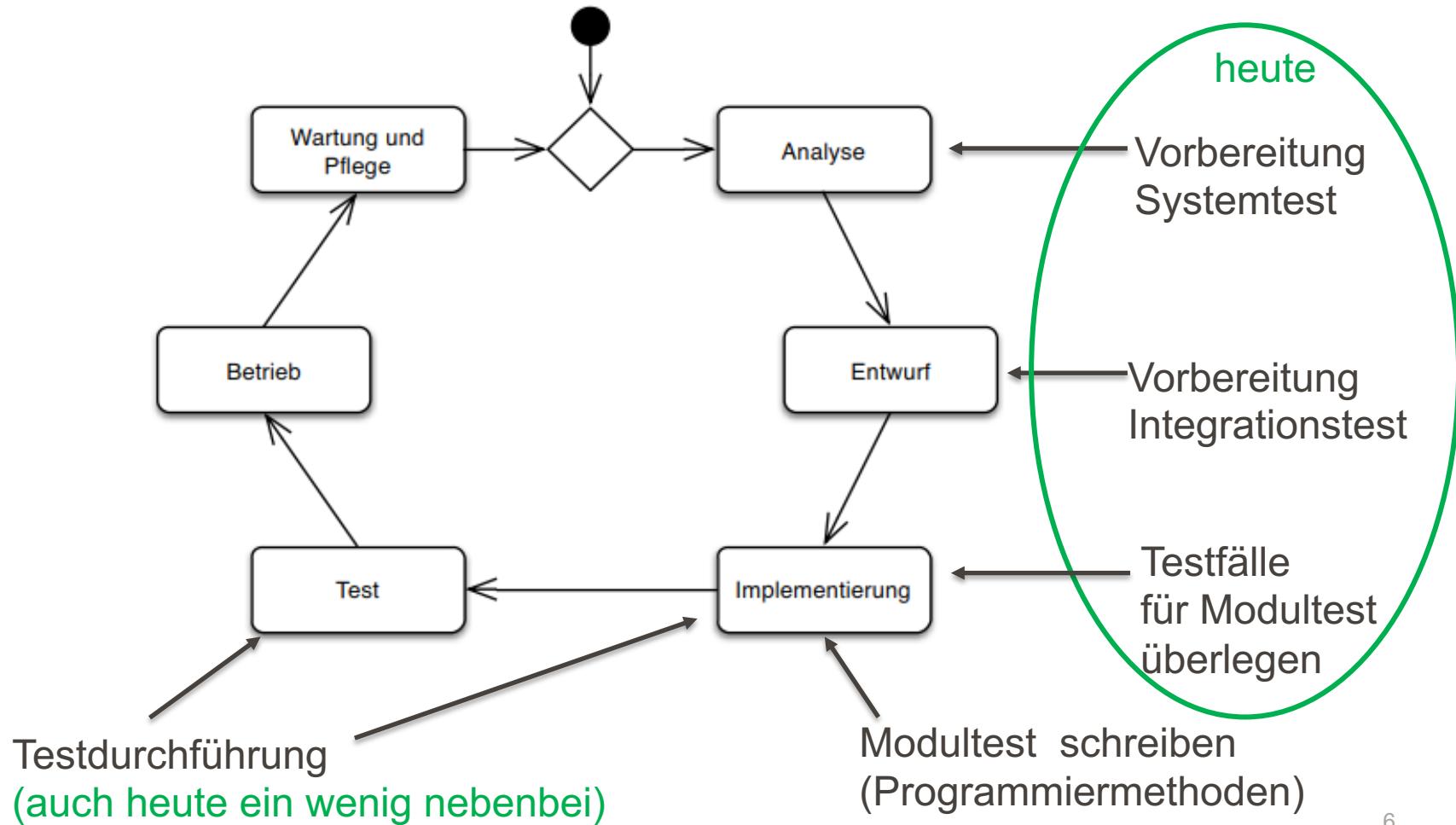
- Die typischen Tätigkeiten bei der SW-Entwicklung:



# TESTS VORBEREITEN UND DURCHFÜHREN



- Die typischen Tätigkeiten bei der SW-Entwicklung:



# UNTERSCHIEDUNG TESTEN ↔ DEBUGGEN



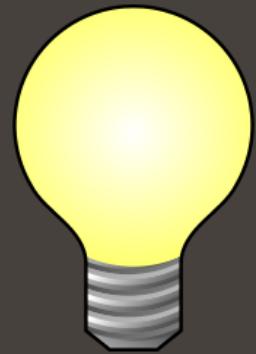
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Testen: Wie finde ich Fehler?
- Debuggen: Wie finde (und behebe) ich die Fehlerursache?



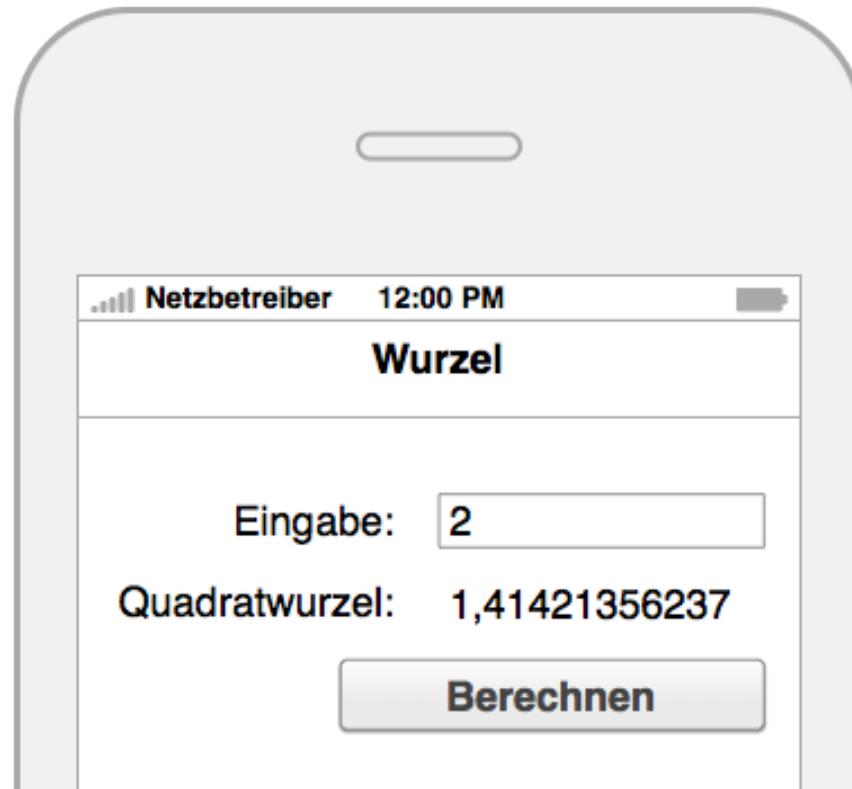
## 02 Erste Überlegungen

Ziel:  
Erste grundsätzliche Rahmenbedingungen explorieren



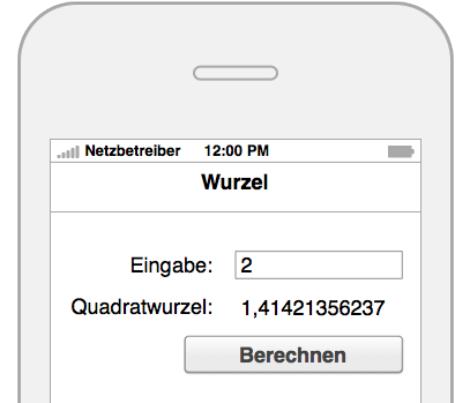


# BEISPIELANWENDUNG:



→ Wie findet man jetzt hier die Fehler?

# BEISPIELANWENDUNG – WIE DIE FEHLER FINDEN?



Wie soll man beim Testen dieser Anwendung vorgehen?

- Typische und sinnvolle Vorgehensweise:
  - Geeignete Werte eingeben
  - Vergleich mit erwarteten Ergebnissen
- ABER: Was sind die geeigneten Eingaben für den Test?

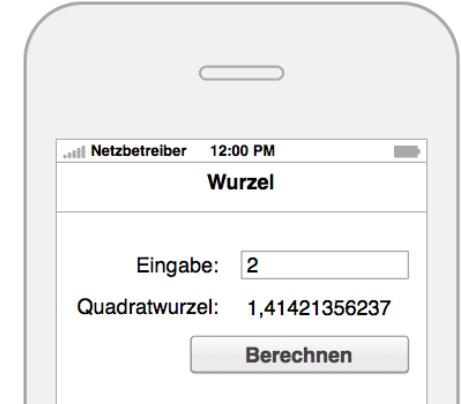
# BEISPIELANWENDUNG – MIT WELCHEN WERTEN TESTEN?

Mit welchen Eingabewerten testen?

- Schlechte Testansätze:
  - alle möglichen Werte
  - nicht testen
  - chaotisch/zufällig testen

## Alle Werte testen:

- Eingabe: alle 64-bit-Fließkommazahlen
- Anzahl der verschiedenen Eingaben:  
 $\approx 2^{64} \approx 10^{0,3 \cdot 64} \approx 10^{20}$
- Annahme: 1 Mrd. Testfälle / s  
 $\rightarrow 10^{20} \text{ s} / 10^9 = 10^{11} \text{ s} \approx 3.000 \text{ Jahre}$



# ANFORDERUNGEN AN GUTE TESTANSÄTZE



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## Anforderungen an gute Testansätze:

- strukturiert
- wiederholbar
- sinnvolle (ökonomische) Auswahl der Testdaten:
  - möglichst wenige Testdaten
  - „möglichst gemein“
- **BEM:** Bei sicherheitskritischen Anwendungen (z.B. Steuerung eines Atomkraftwerks) genügt ein Test mit ausgewählten Werten nicht
  - Test + Verifikation

# WIE FINDET MAN DIE FEHLERURSACHE?



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

The screenshot shows a user interface for calculating square roots. On the left, there is an input field labeled "Eingabe:" containing the number "2". Below it, the text "Quadratwurzel:" is followed by the result "-10,03", which is circled in red. To the right of the result is a button labeled "Berechnen". A red line connects the circled result to the text "≠ erwartetes Ergebnis (korrekt: 1,41...)" located on the far right.

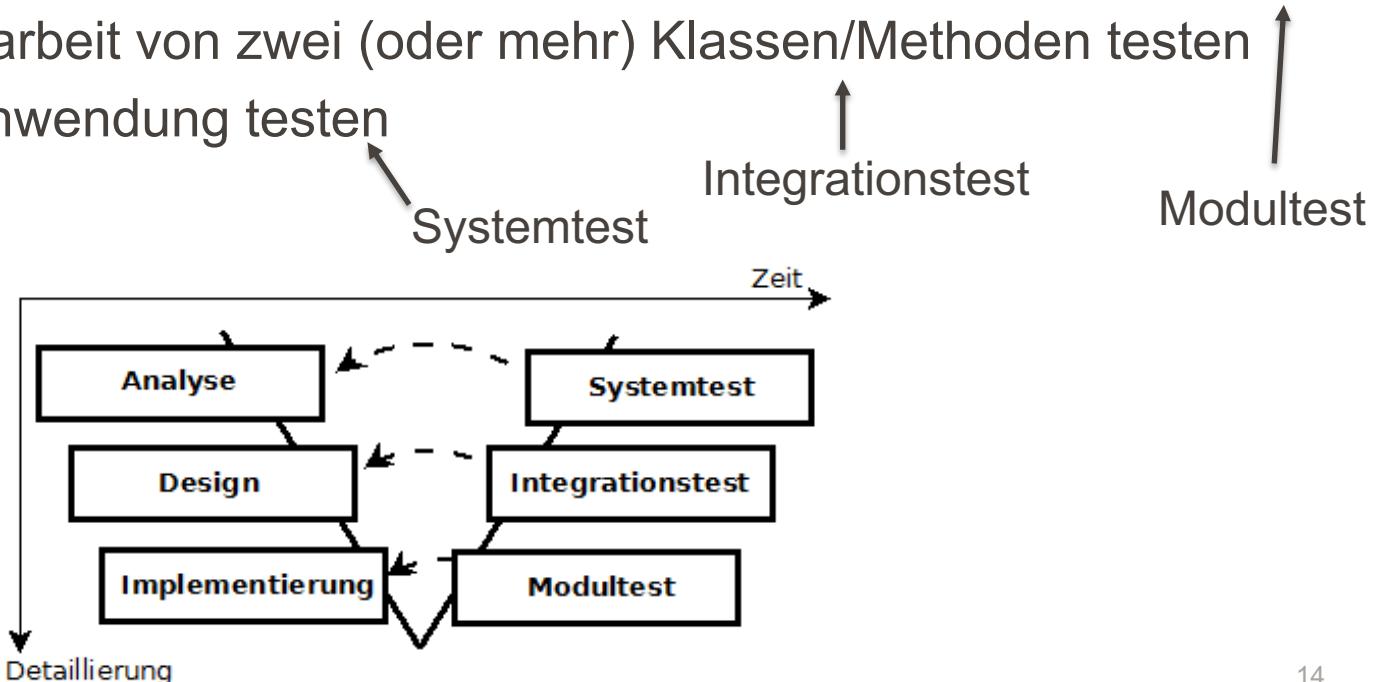
- Wo könnte die Fehlerursache liegen?
  - GUI-Code
  - Wurzelfunktion
  - Verbindung zwischen GUI und Wurzelfunktion
  - ...

→ Wie findet man die Fehlerursache möglichst schnell?

# WIE FINDET MAN DIE FEHLER-URSACHE MÖGLICHST SCHNELL?



- Am besten: Zur rechten Zeit an der richtigen Stelle suchen!
  - Klingt trivial, aber ist gar nicht so einfach
- Mehrstufige Vorgehensweise beim Testen:
  1. Jedes Modul (Klasse/Methode) unabhängig von den anderen testen
  2. Zusammenarbeit von zwei (oder mehr) Klassen/Methoden testen
  3. Gesamte Anwendung testen



# MEHRSTUFIGE VORGEHENSWEISE → TESTEBENEN

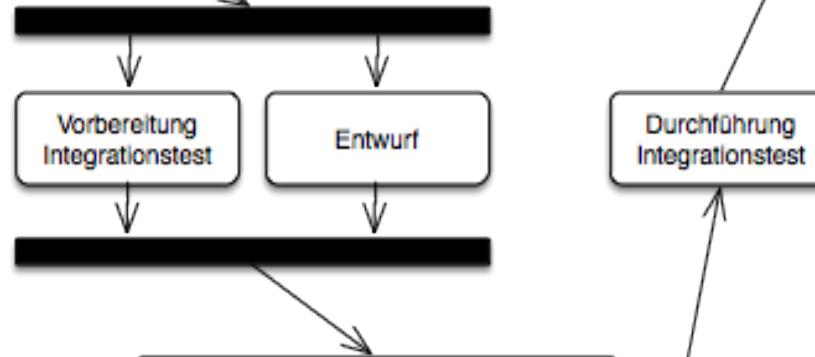


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

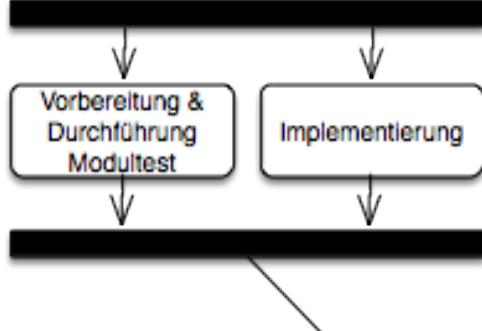
## Systemtest



## Integrationstest



## Modultest



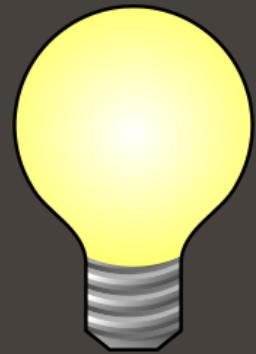


03

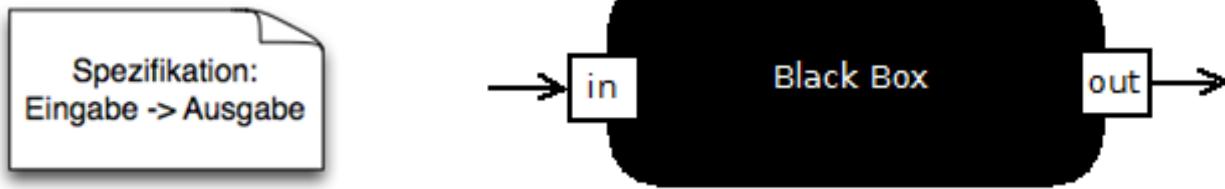
## Black-Box, White-Box, Grey-Box

Ziel:

Verschiedene Testansätze hinsichtlich  
der Sichtbarkeit kennenlernen

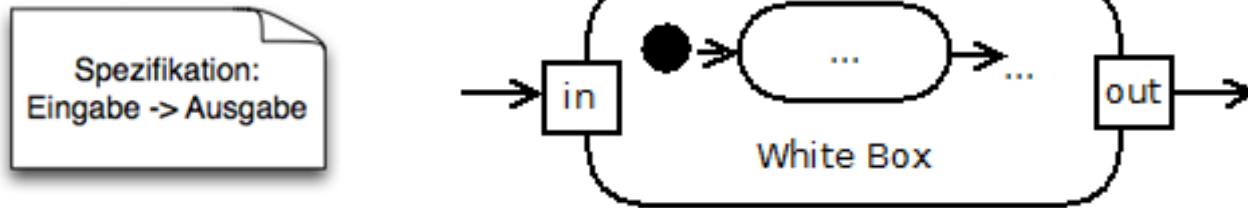


# BLACK-BOX



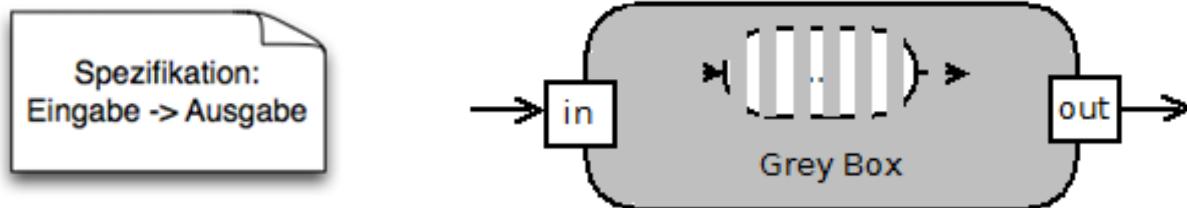
- Merkmale von Black-Box-Tests:
  - Test aus Sicht des Verwenders/Anwenders  
→ von außen
  - keine Sicht auf Implementierung/auf das Innere  
→ undurchsichtige Hülle („black box“)
  - alleinige Grundlage: Spezifikation
  - keine Grundlage: Implementierung
- Einsatz:
  - **Systemtest** → **immer Black Box!!**
  - Aber auch möglich: Integrations- und Modultest

# WHITE-BOX



- Merkmale von White-Box-Tests:
  - Test aus Sicht des Entwicklers  
→ nach innen
  - Sicht auf Implementierung/auf das Innere  
→ durchsichtige Hülle („white box“)
  - Grundlage: Spezifikation + Implementierung
- Einsatz:
  - Eher Modultest
    - Bei sicherheitskritischen Anw. → Pfad-/Anweisungsüberdeckung
  - Vielleicht auch Integrationstest

# GREY-BOX



- Merkmale von Grey-Box-Tests:
  - Sicht auf Implementierung/auf das Innere manchmal möglich/nötig  
→ leicht durchsichtige Hülle („grey box“)
  - Grundlage: Hauptsächlich Spezifikation
  - ABER: Manchmal benutzt man auch Implementierungswissen
- Einsatz:
  - Eher Modultest (wahrscheinlich meistens)
  - Wahrscheinlich auch meistens im Integrationstest

# FINDEN VON TESTFÄLLEN



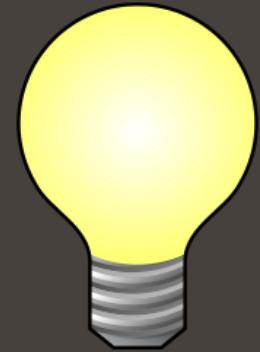
- Beliebte Ansätze zum Finden von Testfällen:
  - Äquivalenzklassenbildung
  - Randwertanalyse
  - Zustandsbasierter Test
  - Ursache-Wirkungs-Analyse
- Geeignet für Black-Box-Test, weil man dazu nur die Schnittstellen kennen muss
- Aber auch bei Grey-Box oder White-Box
  - Man verwendet dazu aber dann auch Implementierungswissen
- Wichtig: Das sind Ansätze
  - (keine einfachen Lösungsformeln zum Einsetzen)



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# 04

## Äquivalenzklassenbildung und Randwertanalyse



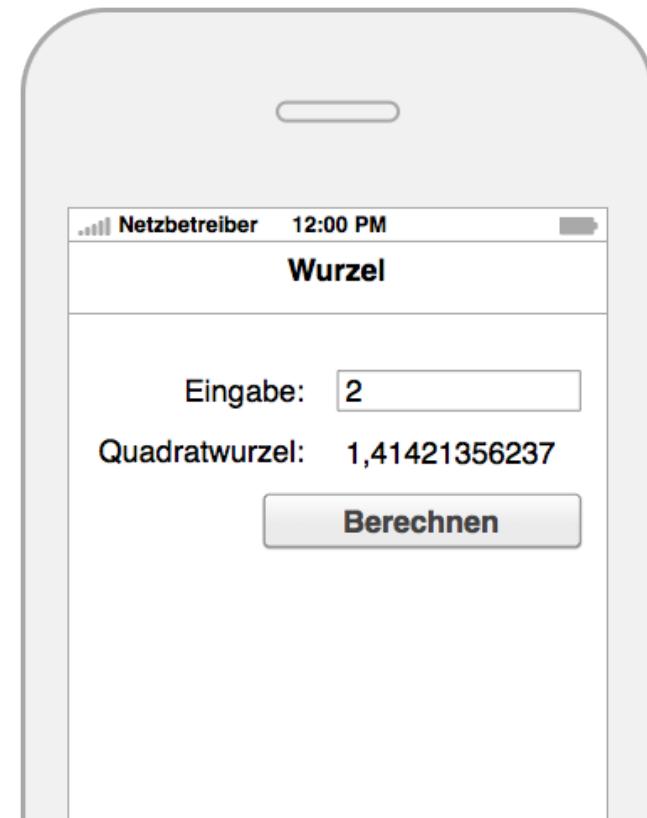
Ziel:  
Verschiedene Testansätze kennenlernen

# ÄQUIVALENZKLASSENBILDUNG (MIT DEM BSP VON VORHER)



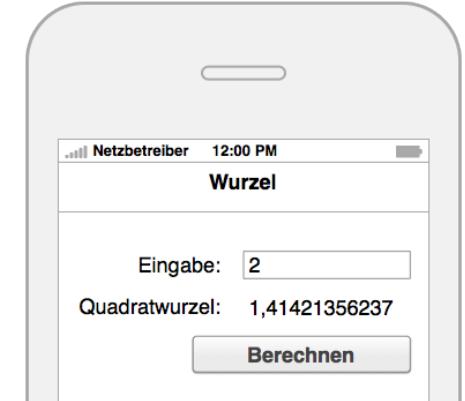
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Was meint Äquivalenzklassenbildung?
  - Zerlegung der Menge aller möglichen (denkbaren) Testdaten in Mengen gleichartiger Testdaten (= Äquivalenzklassen).
- Beispiel von vorher:
  - Mögl. Zerlegung in Äquivalenzklassen:
    - Gültige Eingabe
    - Ungültige Eingabe
      - Zahl zu groß
      - Zahl zu klein
      - Keine Zahl



# ÄQUIVALENZKLASSENILDUNG (MIT DEM BSP VON VORHER)

→ Hier hilft eine Tabelle:



ID	Beschreib. der Äquivalenzklasse	Bedingung („logischer Testfall“)	Beispielhafter Wert („konkreter Testfall“)
AK1	Zahl zu groß	Zahl > MAX	...
AK2	gültige Eingabe	$0 \leq \text{Zahl} \leq \text{MAX}$	2
AK3	Zahl zu klein	Zahl < 0	-1,234
AK4	keine Zahl	...	abc

# RANDWERTANALYSE (MIT DEM BSP VON VORHER)

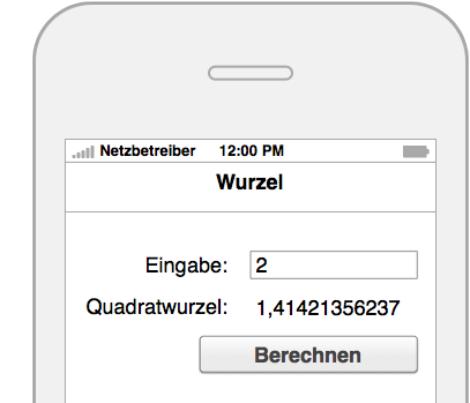
→ Ränder der Äquivalenzklassen analysieren  
& sinnvolle Randwerte feststellen:

ID	Beschreib. der Äquivalenzklasse	Bedingung („logischer Testfall“)	Beispielhafter Wert („konkreter Testfall“)
AK1	Zahl zu groß	Zahl > MAX	...
AK2	gültige Eingabe	$0 \leq \text{Zahl} \leq \text{MAX}$	2
AK3	Zahl zu klein	Zahl < 0	-1,234
AK4	keine Zahl	...	abc

Diagramm zur Analyse der Ränder:

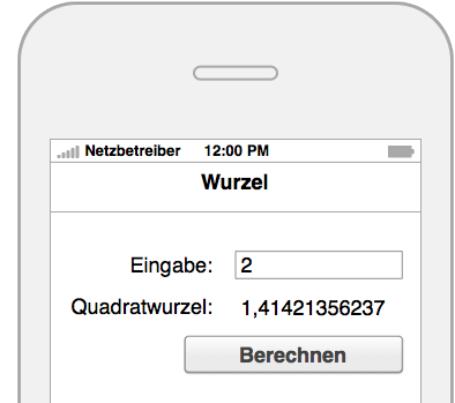
- AK1: Sinnvolle Ränder (???)
- AK2: Sinnvolle Ränder (Sinnvolle Ränder)
- AK3: Sinnvolle Ränder (???)
- AK4: Sinnvolle Ränder (???)

BEM: ???  $\triangleq$  (eher) kein sinnvoller Rand (Randwert) ableitbar



# RANDWERTANALYSE (MIT DEM BSP VON VORHER)

→ Sinnvolle Ränder in die Tabelle:

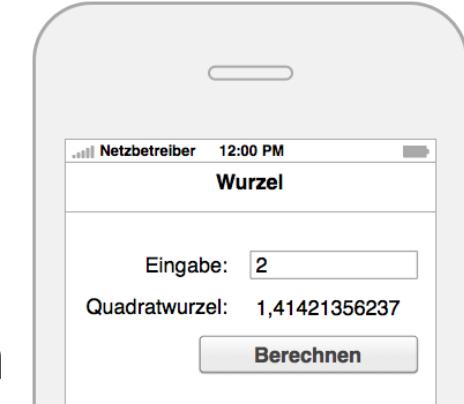


ID	Art (AK / RW)	Beschreib. der Äquivalenz- klasse	Bedingung „logischer Testfall“)	Beispielhafter Wert „konkreter Testfall“)
AK1	AK	Zahl zu groß	Zahl > MAX	...
RW1	Rand	gültige Eingabe: oberer Rand	Zahl = MAX	MAX
AK2	AK	gültige Eingabe	$0 \leq \text{Zahl} \leq \text{MAX}$	2
RW2	Rand	gültige Eingabe: unterer Rand	Zahl = 0	0
AK3	AK	Zahl zu klein	Zahl < 0	-1,234
AK4	AK	keine Zahl	...	abc

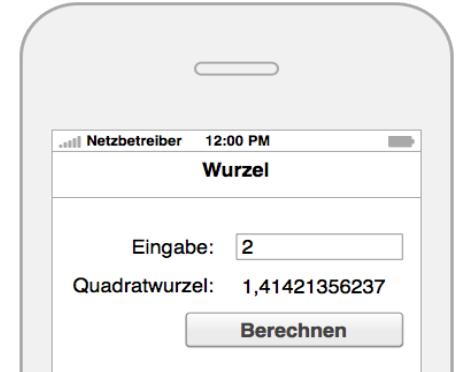
# RANDWERTANALYSE (MIT DEM BSP VON VORHER)

→ Bei Rändern immer den Übergangsbereich testen:

ID	Art (AK / RW)	Beschreib. der Äquivalenz- klasse	Bedingung „logischer Testfall“)	Beispielhafte Werte „konkreter Testfall“)
AK1	AK	Zahl zu groß	Zahl > MAX	...
RW1	Rand	gültige Eingabe: oberer Rand	Zahl = MAX	MAX+1, MAX, MAX-1
AK2	AK	gültige Eingabe	$0 \leq \text{Zahl} \leq \text{MAX}$	2
RW2	Rand	gültige Eingabe: unterer Rand	Zahl = 0	-MIN, 0, + MIN
AK3	AK	Zahl zu klein	Zahl < 0	-1,234
AK4	AK	keine Zahl	...	abc



# IST DAS SCHON ALLES?

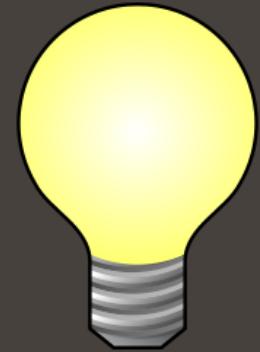


- Diese Anwendung hat nur einen Parameter
- Oftmals gibt es aber eine Vielzahl von Parametern
  - Kombinationen von Parameter führen zu gewünschten Ergebnissen
  - Es müssen auch Parameterkombinationen getestet werden, ob sie jeweils zu den erwünschten Ergebnissen führen
- Wie macht man das?



# 05

## Kombinierung von Testfällen (Testvektoren)



Ziel:

Tests haben mehrere Parameter

Verschiedene Testbedingungen geschickt zu Testvektoren kombinieren

# BSP: DIE SUBSTRING-METHODE IN JAVA



(Siehe auch 2. Semester PMT  
Aufgabenblatt zu Unittesting)

```
public String substring(int beginIndex,  
                      int endIndex)
```

Returns a string that is a substring of this string. The substring begins at the specified beginIndex and extends to the character at index endIndex - 1. Thus the length of the substring is endIndex - beginIndex.

Examples:

```
"hamburger".substring(4, 8) returns "urge"  
"smiles".substring(1, 5) returns "mile"
```

#### Parameters:

beginIndex - the beginning index, inclusive.

endIndex - the ending index, exclusive.

#### Returns:

the specified substring.

#### Throws:

IndexOutOfBoundsException - if the beginIndex is negative, or endIndex is larger than the length of this String object, or beginIndex is larger than endIndex.

# BSP: DIE SUBSTRING-METHODE IN JAVA



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Ziel: Wir müssen die substring-Methode mittels Unitest testen  
(Vgl. Programmiermethoden Aufgabenblatt 02)
- Wie gehen wir **systematisch** vor?
  - Problem: Mehrere Parameter & Kombinationen
  - Lösung:
    - Wir analysieren zunächst jeden Parameter nach Äquivalenzklassen und Randwerten
    - Danach suchen wir daraus günstige Kombinationen



Vorsicht:

- Substring hat **3** Parameter:
  1. Den Stringinhalt
  2. beginIndex
  3. endIndex

# AK- UND RW-ANALYSE VON STRINGINHALT



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

ID	Art (AK / RW)	Beschreib. der Äquivalenz- klasse	Bedingung „logischer Testfall“)	Beispielhafte Werte „konkrete Testfälle“)
AK1_1	AK	Gültiger String mit Inhalt	String mit Länge $\geq 0$	"hamburger"
RW1_1	Rand	Leerstring	String mit Länge = 0	"h", "" <del>null</del>



Hier macht null keinen Sinn,  
Weil null.substring(...,...)  
→ NullPointerException

# AK- UND RW-ANALYSE VON BEGININDEX (BI)



ID	Art (AK / RW)	Beschreib. der Äquivalenz- klasse	Bedingung („logischer Testfall“)	Beispielhafte Werte („konkrete Testfälle“)
AK2_1	AK	bl kleiner 0	$bl < 0$	$bl = -10$
RW2_1	RW	bl gleich 0	$bl = 0$	-1, 0, 1
AK2_2	AK	bl im gültigen Bereich	$0 \leq bl \leq el$	$bl=4, el=8$
RW2_2	RW	bl gleich el	$bl = el$	$bl=el-1, bl=el,$ $bl=el+1$
AK2_3	AK	bl grösser el	$bl > el$	$bl = el + x$ $(bl > str.length())$

Legende:

bl – beginIndex, el – endIndex,

str – Stringinhalt

Was ist mit  $bl > str.length()$ ?  
 → Anscheinend durch  $el > str.length()$   
 abgedeckt!  
 → Zur Not mit aufnehmen!

# AK- UND RW-ANALYSE VON ENDINDEX (EI)



ID	Art (AK / RW)	Beschreib. der Äquivalenz- klasse	Bedingung „logischer Testfall“)	Beispielhafte Werte („konkrete Testfälle“)
AK3_1	AK	el kleiner bl	el < bl	el = bl - X
RW3_1	RW	el gleich bl	el = bl	el=bl -1, ei = bl, el = bl+1
AK3_2	AK	el ist im gültigen Bereich	bl ≤ el ≤ str.length()	bl=4, el=8
RW3_2	RW	el gleich str.length()	el = str.length()	el=str.length()-1, el = str.length, ei= str.length +1
AK3_3	AK	el > str.length	el> str.length()	el=str.length() + X

Legende:

bl – beginIndex, el – endIndex, str – Stringinhalt



# KOMBINIERUNG:

StringInhalt:

ID	...
AK1_1	...
RW1_1	...

beginIndex:

ID	...
AK2_1	..
RW2_1	...
AK2_2	...
RW2_2	...
AK2_3	...

endIndex:

ID	...
AK3_1	..
RW3_1	...
AK3_2	...
RW3_2	...
AK3_3	...

Menge aller möglichen Kombinationen:

$$\{(AK1\_1, AK2\_1, AK3\_1), (AK1\_1, AK2\_1, RW3\_1), \dots, \dots\}$$

→ Kombinatorische Explosion → ungeschickt

→ Wir müssen die sinnvollen Kombinationen reduzieren!

# WIE DIE ANZAHL DER SINNV. KOMBINATIONEN REDUZIEREN?



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Beziehungen zwischen den Testfällen analysieren und daraufhin die wichtigen Kombinationen aussieben!
  - Aber wie?
- Durch Ursache-Wirkungs-Analyse (in vereinfachter Form):
  - Aufteilung der Testfälle in zwei Kategorien:
    - Normal („gut“) → Gutfälle
    - Ausnahme/Fehler → Schlechtfälle
  - (Angenommene) Beziehungen:
    - Fehler in guten Testfällen heben sich nicht gegenseitig auf
      - gute Testfälle können gemeinsam getestet werden
    - Fehler in Ausnahme-/Fehler-Testfällen können sich überdecken.
      - Ausnahme-/Fehler-Fälle dürfen nicht gemischt werden

# KOMBINATION – EINTEILUNG IN GUT- UND SCHLECHTFÄLLE:



StringInhalt:

ID	Log. Testfall	...
AK1_1	String mit Länge > 0	...
RW1_1	String mit Länge = 0	...

beginIndex:

ID	Log. Testf.	...
AK2_1	bl < 0	...
RW2_1	bl = 0	...
AK2_2	0 ≤ bl ≤ el	...
RW2_2	bl = el	...
AK2_3	bl > el	...

endIndex:

ID	Log. Testfall	...
AK3_1	el < bl	...
RW3_1	el = bl	...
AK3_2	bl ≤ el ≤ str.length	...
RW3_2	el = str.length	...
AK3_3	el > str.length	...

Hirn einschalten:

→ Folgende Kriterien sind zueinander gleich (redundant):

$$RW2\_2 \leftrightarrow RW3\_1, AK2\_3 \leftrightarrow AK3\_1$$

→ Können wir streichen

**BEM:** Passiert, wenn Parameter nicht völlig unabhängig voneinander

normal
Ausnahme/ Fehler

# KOMBINATION – EINTEILUNG IN GUT- UND SCHLECHTFÄLLE



StringInhalt:

ID	Log. Testfall	...
AK1_1	String mit Länge > 0	...
RW1_1	String mit Länge = 0	...

beginIndex:

ID	Log. Testf.	...
AK2_1	bl < 0	...
RW2_1	bl = 0	...
AK2_2	0 ≤ bl ≤ el	...
RW2_2	bl = el	...
AK2_3	bl > el	...

endIndex:

ID	Log. Testfall	...
AK3_2	bl ≤ el ≤ str.length	...
RW3_2	el = str.length	...
AK3_3	el > str.length	...

→ Jetzt kombinieren

# KOMBINATION GUT- UND SCHLECHTFÄLLE:

StringInhalt:

ID	Log. Testfall	...
AK1_1	String mit Länge > 0	...
RW1_1	String mit Länge = 0	...

beginIndex:

ID	Log. Testf.	...
AK2_1	bl < 0	...
RW2_1	bl = 0	...
AK2_2	0 ≤ bl ≤ el	...
RW2_2	bl = el	...
AK2_3	bl > el	...

endIndex:

ID	Log. Testfall	...
AK3_2	bl ≤ el ≤ str.length	...
RW3_2	el = str.length	...
AK3_3	el > str.length	...

Stringinhalt	beginIndex	endIndex
AK1_1	AK2_1	AK3_2
AK1_1	AK2_1	RW3_2
AK1_1	AK2_1	AK3_3
AK1_1	RW2_1	AK3_2
AK1_1	RW2_1	RW3_2
AK1_1	RW2_1	AK3_3
...	...	...

Ist redundant zu einer Zeile weiter oben

Schlecht, weil sich 2 Schlechtfälle überlagern

Ist hier nicht AK1\_1 und RW2\_1 redundant?

Hui, das werden ganz schön viele  
 → Kombinatorische Explosion  
 → Besser: Aussieben!!



# KOMBINATION GUT- UND SCHLECHTFÄLLE:

StringInhalt:			beginIndex:			endIndex:		
ID	Log. Testfall	...	ID	Log. Testf.	...	ID	Log. Testfall	...
AK1_1	String mit Länge > 0	...	AK2_1	bl < 0	...	AK3_2	bl ≤ el ≤ str.length	...
RW1_1	String mit Länge = 0	...	RW2_1	bl = 0	...	RW3_2	el = str.length	...
			AK2_2	0 ≤ bl ≤ el	...	AK3_3	el > str.length	...
			RW2_2	bl = el	...			
			AK2_3	bl>el	...			

→ (vereinfachte) Ursache-Wirkungs-Analyse nutzen:

- Allgemeine Kombinationsregeln (Kochrezept):

## 1. Gutfälle:

- 1.1: Jeder Gutfall muss mindestens ein Mal in einer Kombination, in der nur Gutfälle vorkommen, getestet werden

## 2. Schlechtfälle:

- 2.1: Jeder Schlechtfall darf nur mit Gutfällen kombiniert werden
- 2.2: Jeder Schlechtfall muss mind. einmal vorkommen

# KOMBINATION GUT- UND SCHLECHTFÄLLE:

StringInhalt:

ID	Log. Testfall	...
AK1_1	String mit Länge > 0	...
RW1_1	String mit Länge = 0	...

beginIndex:

ID	Log. Testf.	...
AK2_1	bl < 0	...
RW2_1	bl = 0	...
AK2_2	0 ≤ bl ≤ el	...
RW2_2	bl = el	...
AK2_3	bl > el	...

endIndex:

ID	Log. Testfall	...
AK3_2	bl ≤ el ≤ str.length	...
RW3_2	el = str.length	...
AK3_3	el > str.length	...

String inhalt	beginIndex	endIndex
AK1_1	RW2_1	AK3_2
AK1_1	AK2_2	RW3_2
RW1_1	RW2_2	RW3_2
R2.1 & R2.2	AK1_1	AK2_1
	AK1_1	AK2_3
AK1_1	RW2_1	AK3_3

Alle Gutfälle schon abgedeckt



Alle Schlechtfälle schon abgedeckt



→ Schon fertig!



→ Lineares Wachstum statt Kombinatorischer Explosion!

# TESTVEKTOREN & WEITERE HINWEISE

String inhalt	beginIndex	endIndex
AK1_1	RW2_1	AK3_2
AK1_1	AK2_2	RW3_2
RW1_1	RW2_2	RW3_2
AK1_1	AK2_1	AK3_2
AK1_1	AK2_3	RW3_2
AK1_1	RW2_1	AK3_3

Testvektor

StringInhalt:		
ID	Log. Testfall	...
AK1_1	String mit Länge > 0	...
RW1_1	String mit Länge = 0	...

beginIndex:		
ID	Log. Testf.	...
AK2_1	bl < 0	...
RW2_1	bl = 0	...
AK2_2	0 ≤ bl ≤ el	...
RW2_2	bl = el	...
AK2_3	bl > el	...

endIndex:		
ID	Log. Testfall	...
AK3_2	bl ≤ el ≤ str.length	...
RW3_2	el = str.length	...
AK3_3	el > str.length	...

Minimale Menge der Testvektoren für sauberen Test

## Weiterer Hinweise:

- Diese Parameterkombinationen beim Testen werden oft als Testvektoren bezeichnet
- Wie man sieht funktioniert die hier vorgestellte Vorgehensweise sowohl für Systemtest (z.B. GUI-Eingabe) als auch Modultest, ...

# VON LOG. TESTF. ZU KONKREten TESTF.

StringInhalt:			beginIndex:			endIndex:		
ID	Log. Testfall	...	ID	Log. Testf.	...	ID	Log. Testfall	...
AK1_1	String mit Länge > 0	...	AK2_1	bl < 0	...	AK3_2	bl ≤ el ≤ str.length	...
RW1_1	String mit Länge = 0	...	RW2_1	bl = 0	...	RW3_2	el = str.length	...
			AK2_2	0 ≤ bl ≤ el	...	AK3_3	el > str.length	...
			RW2_2	bl = el	...			
			AK2_3	bl>el	...			

- Bisher haben wir nur die log. Testfälle betrachtet.  
→ Bei konkreten Testfällen für Randwerte gibt es aber mehrere Werte:

ID	Art (AK / RW)	Beschreib. der Äquivalenz-klasse	Bedingung („logischer Testfall“)	BSP-Werte („konkrete Testfälle“)
AK2_1	AK	bl kleiner 0	bl < 0	bl=-10
RW2_1	RW	bl gleich 0	bl = 0	-1, 0, 1
AK2_2	AK	bl im gültigen Bereich	0 ≤ bl ≤ el	bl=4, el=8
		...		

Manche davon sind auch noch Gut- und manche Schlechtfälle

→ Wie geht man damit um?



# VON LOG. TESTF. ZU KONKREten TESTF.

StringInhalt:			beginIndex:			endIndex:		
ID	Log. Testfall	...	ID	Log. Testf.	...	ID	Log. Testfall	...
AK1 _1	String mit Länge > 0	...	AK2_1	bl < 0	...	AK3_2	bl ≤ el ≤ str.length	...
RW 1_1	String mit Länge = 0	...	RW2_1	bl = 0	...	RW3_2	el = str.length	...
			AK2_2	0 ≤ bl ≤ el	...	AK3_3	el > str.length	...
			RW2_2	bl = el	...			

ID	Art (AK / RW)	Beschreib. der Äquivalenz-klasse	Bedingung („logischer Testfall“)	BSP-Werte („konkrete Testfälle“)
AK2_1	AK	bl kleiner 0	bl < 0	bl=-10
RW2_1	RW	bl gleich 0	bl = 0	-1, 0, 1
AK2_2	AK	bl im gültigen Bereich	0 ≤ bl ≤ el	bl=4, el=8
		...		

→ Zwei Möglichkeiten:

1. Man verteilt die Werte auf die angrenzenden AKs:

AK2\_1: bl= -1

RW2\_1: bl= 0

AK2\_2: bl= 1

- Möglicher Nachteil: AKs werden nur an den Rändern getestet  
 → Vielleicht geht aber etwas in der Mitte schief?

# VON LOG. TESTF. ZU KONKREten TESTF.

StringInhalt:			beginIndex:			endIndex:		
ID	Log. Testfall	...	ID	Log. Testf.	...	ID	Log. Testfall	...
AK1_1	String mit Länge > 0	...	AK2_1	bl < 0	...	AK3_2	bl ≤ el ≤ str.length	...
RW1_1	String mit Länge = 0	...	RW2_1	bl = 0	...	RW3_2	el = str.length	...
			AK2_2	0 ≤ bl ≤ el	...			...

ID	Art (AK / RW)	Beschreib. der Äquivalenz-klasse	Bedingung („logischer Testfall“)	BSP-Werte („konkrete Testfälle“)
AK2_1	AK	bl kleiner 0	bl < 0	bl=-10
RW2_1	RW	bl gleich 0	bl = 0	-1, 0, 1
AK2_2	AK	bl im gültigen Bereich	0 ≤ bl ≤ el	bl=4, el=8
		...		

→ Zwei Möglichkeiten (empfohlen):

2. Man erweitert den Test dann um die spez. Randwerttests:

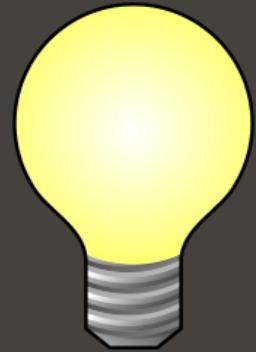
String inhalt	beginIndex	endIndex
AK1_1	RW2_1	AK3_2
...	...	...

String inhalt	beginIndex	endIndex
AK1_1	RW2_1 (-1)	AK3_2
AK1_1	RW2_1 (0)	AK3_2
AK1_1	RW2_1 (1)	AK3_2
	...	



# 06

## Zustandsbasierte Tests



Ziel:

Wie testet man Software, die zustandsabhängig ist?

Zustandsabhängig  $\triangleq$  die SW verhält sich je nach Zustand evtl. anders

# ZUSTANDSBASIERTES TESTEN – UNSER MP3-PLAYER BEISPIEL



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

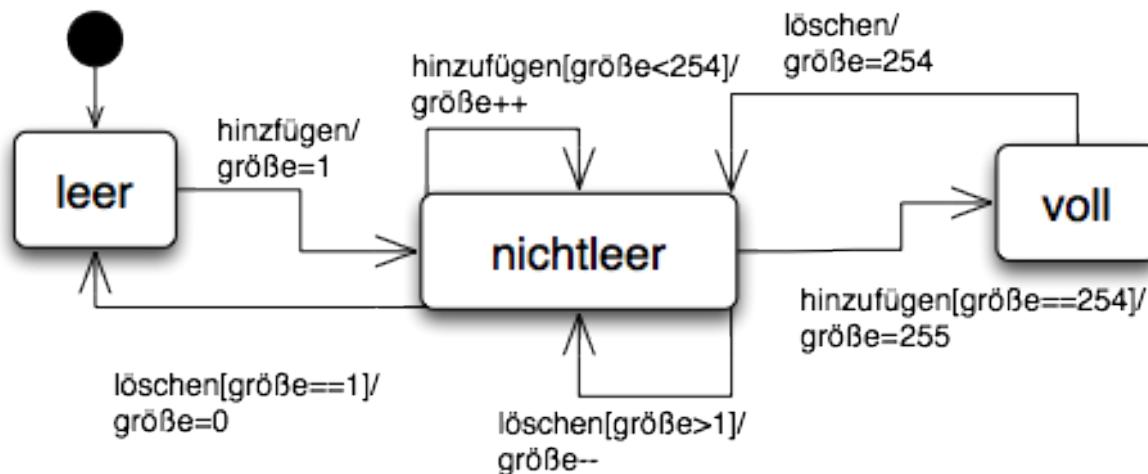


- In den einzelnen Anwendungsfällen werden implizit Zustände und Zustandswechsel der Playlist beschrieben
- Beim Testen berücksichtigen! → Zustandsbasiertes Testen

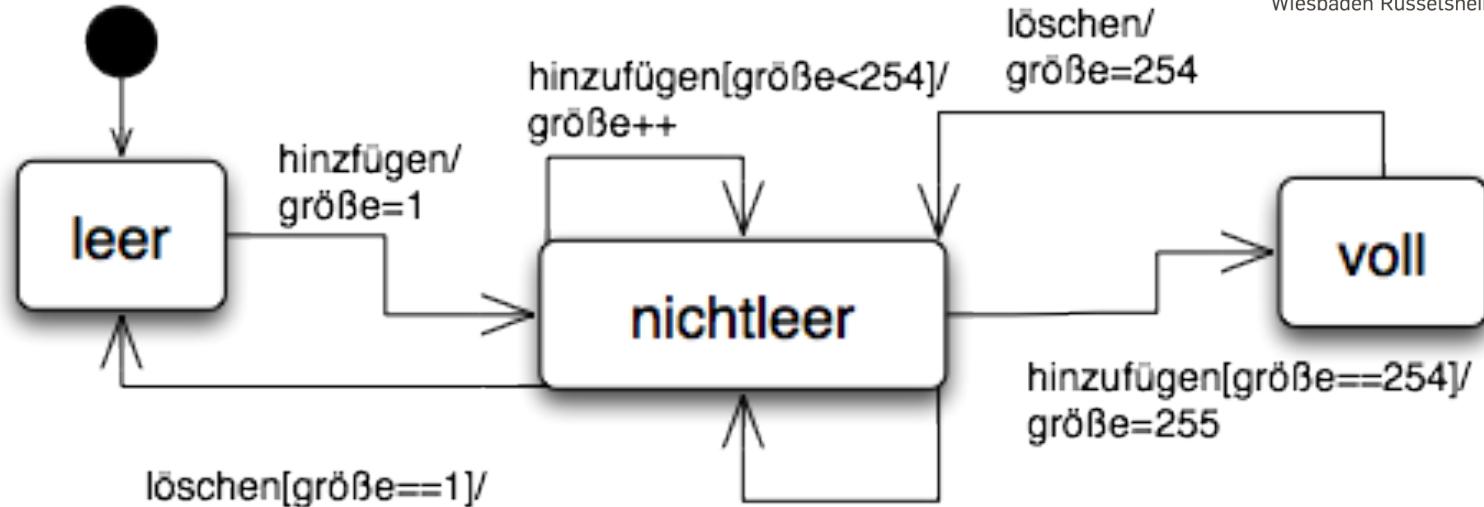
# ZUSTANDSBASIERTES TESTEN



- Voraussetzungen: Spezifikation der Zustände und Zustandsübergänge → z.B. Zustandsdiagramm
- Vorgehen:
  - Zustandsdiagramm → Testfälle ableiten
  - jeder Übergang → 1 Testfall
  - jeder „Nicht-Übergang“ → 1 Testfall  
(z.B. „löschen“ im Zustand im „leer“)
- BSP: Zustandsdiagramm für Playlist:



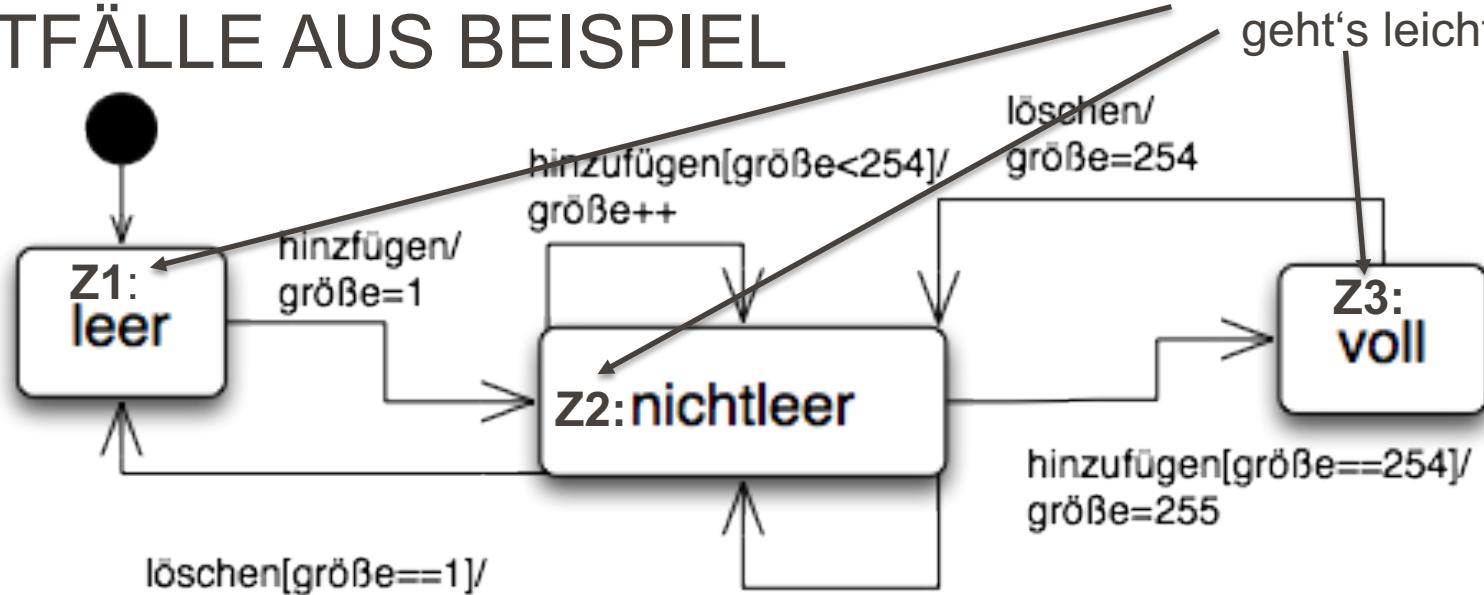
# ZUSTANDSBASIERTES TESTEN – TESTFÄLLE AUS BEISPIEL



Id	Testfall		Zustand nachher/ erwartetes Ergebnis
	Zustand vorher	Ereignis	
ZT01	–	Playlist anlegen	leere Playlist
ZT02	leer	Titel hinzufügen	Playlist mit 1 Titel
ZT03	leer	Titel löschen	leer + Fehlermeldung?
ZT04	nichtleer, 1 Titel	Titel hinzufügen	Playlist mit 2 Titeln
...	...	...	...

# ZUSTANDSBASIERTES TESTEN – TESTFÄLLE AUS BEISPIEL

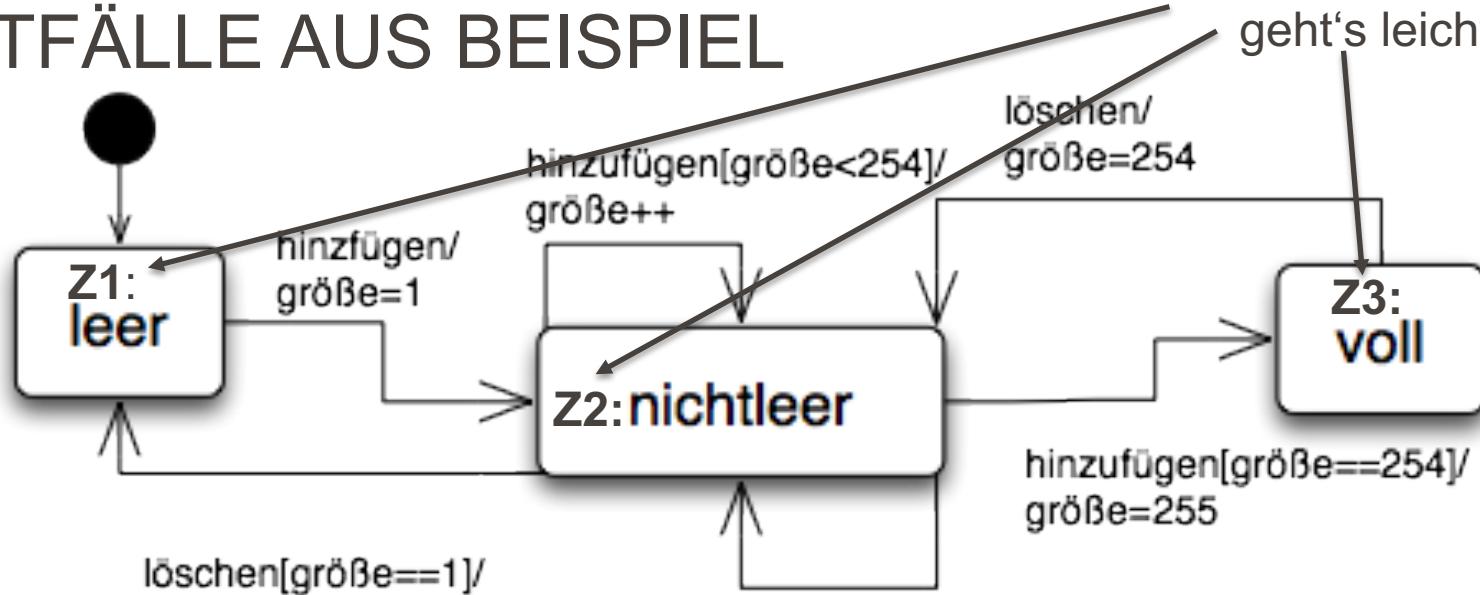
TIP: Mit Zustand-Ids geht's leichter



Id	Testfall		Zustand nachher/ erwartetes Ergebnis
	Zustand vorher	Ereignis	
ZT01	–	Playlist anlegen	Z1 (leere Playlist)
ZT02	Z1	Titel hinzufügen	Z2 (Playlist mit 1 Titel)
ZT03	Z1	Titel löschen	Z1+ Fehlermeldung?
ZT04	Z2 (1 Titel)	Titel hinzufügen	Z2 (Playlist mit 2 Titeln)
...	...	...	...

# ZUSTANDSBASIERTES TESTEN – TESTFÄLLE AUS BEISPIEL

TIP: Mit Zustand-Ids geht's leichter



Id	Testfall		Zustand nachher/ erwartetes Ergebnis
	Zustand vorher	Ereignis	
ZT01	–	Playlist anlegen	Z1 (leere Playlist)
ZT02	Z1	Titel hinzufügen	Z2 (Playlist mit 1 Titel)
ZT03	Z1	Titel löschen	Z1+ Fehlermeldung?
ZT04	Z2 (1 Titel)	Titel hinzufügen	Z2 (Playlist mit 2 Titeln)
...	...	...	...



# WEITERE ALLGEMEINE HINWEISE ZU ZUSTANDSBASIERTEN TESTS



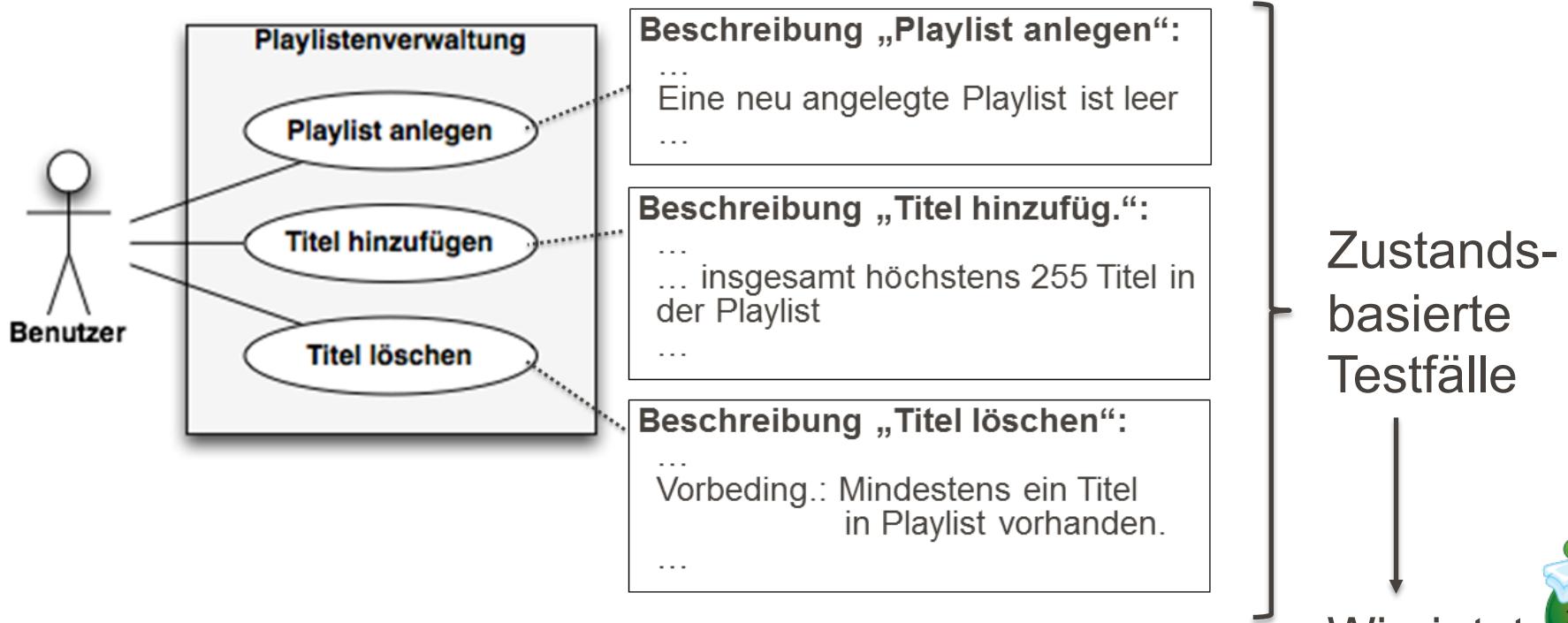
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Basis für Testfälle:
    - Üblich: alle Übergänge
    - Besser: alle Ereignisse, d.h. auch:
      - Fehlerfälle
      - zu ignorierende Ereignisse
  - Herstellen der Testsituation:
    - Testsituation kann i.d.R. nicht in 1 Schritt hergestellt werden
    - i.d.R. mehrere Schritte zur Vorbereitung
      - Ausnahme: Startzustand
- Zusammenfassen von Testfällen bei der Testdurchführung:
- Auf dem Weg zu einer best. Testsituat. → kann man andere Testfälle „nebenbei“ machen
  - Zustandsautom. kann durch geschicktes Komb. und Aufbauen der Testfälle durchlaufen werden → effizienter Test
-  ABER: Scheitert ein Testfall, sind andere auch betroffen!

# ZUSTANDSBASIERTE TESTS MIT ANDEREN TESTS KOMBINIEREN



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim



Titel in Playlist:

Titel
name: Zeichenkette
...

{name hat mind. 1 Zeichen,  
max. 31 Zeichen und keine  
Sonderzeichen}

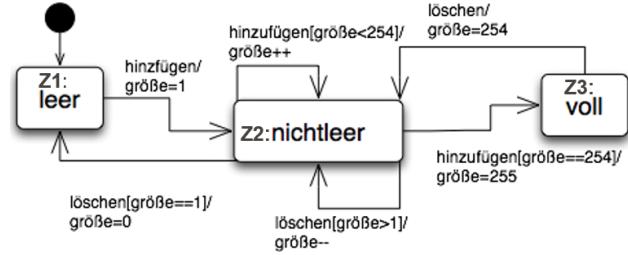
Äquivalenzklassen &  
Randwertanalyse

Wie jetzt  
kombinieren?



# WIE ZUSTANDSBASIERTE UND ANDERE TESTS KOMBINIEREN?

→ Wie vorher



## 1. Gut- und Schlechtfälle bestimmen:

Zustände Playlist:

Id	Testfall		...
	Zustand vorher	Ereignis	
ZT01	–	Playlist anlegen	...
ZT02	Z1	Titel hinzufügen	...
ZT03	Z1	Titel löschen	...
ZT04	Z2, (1 Titel)	Titel hinzufügen	...
...	...	...	...
ZT08	Z3	Titel hinzufügen	...
...	...	...	...

Titelname für Titel in Playlist:

Id	Beschreibung	...
AK1	ungültige Zeichen in Name	...
AK2	Name zu lang	...
RW1	maximale Länge	...
AK3	Name hat gültige Länge	...
RW2	minimale Länge	...
AK4	Name zu kurz	...

# KOMBINIEREN

Id	Testfall		...	Id	Beschreibung	...
	Zustand vorher	Ereignis				
ZT01	-	Playlist anlegen	...	AK1	ungültige Zeichen in Name	...
ZT02	Z1	Titel hinzufügen	...	AK2	Name zu lang	...
ZT03	Z1	Titel löschen	...	RW1	maximale Länge	...
ZT04	Z2, (1 Titel)	Titel hinzufügen	...	AK3	Name hat gültige Länge	...
...	...	...	...	RW2	minimale Länge	...
ZT08	Z3	Titel hinzufügen	...	AK4	Name zu kurz	...
...	...	...	...			

## 2. Sinnvoll kombinieren:

Zustandsb. Testfälle	Äquiv. & Randwerte
ZT01	-
ZT02	AK1
ZT02	AK2
ZT02	RW1
ZT03	-
ZT04	AK4
ZT04	RW2
ZT04	AK3
ZT08	AK3

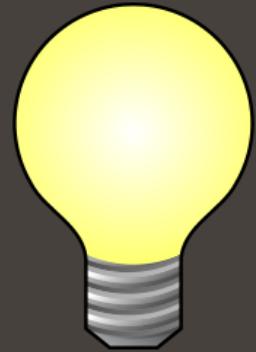
Nur sinnvoll im Zustand Z1 & Z2 mit Gutfällen → Hirn einschalten!

(Hier wird nur aufgefüllt)



07

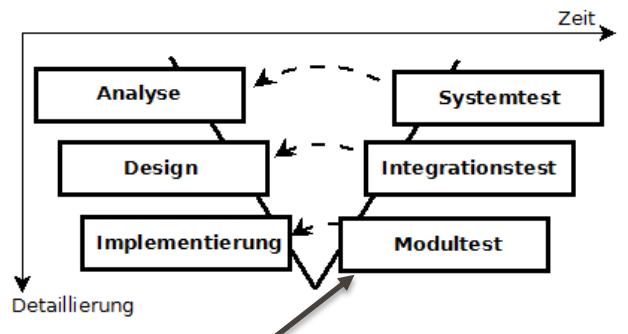
## Tests auf verschiedenen Abstraktionsebenen



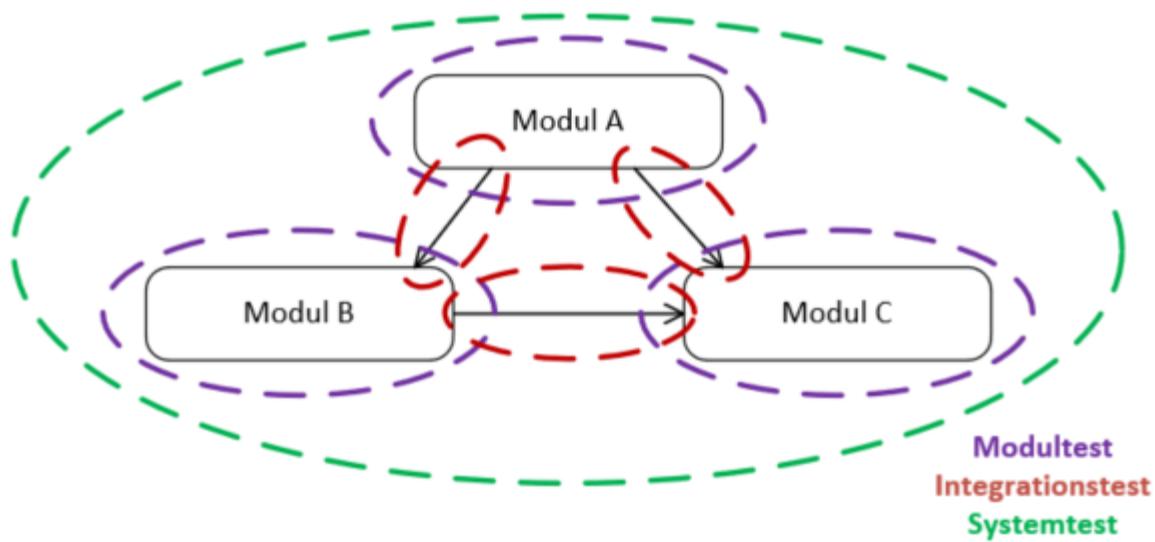
Ziel:

Verschiedene Ebenen des Tests kennenlernen

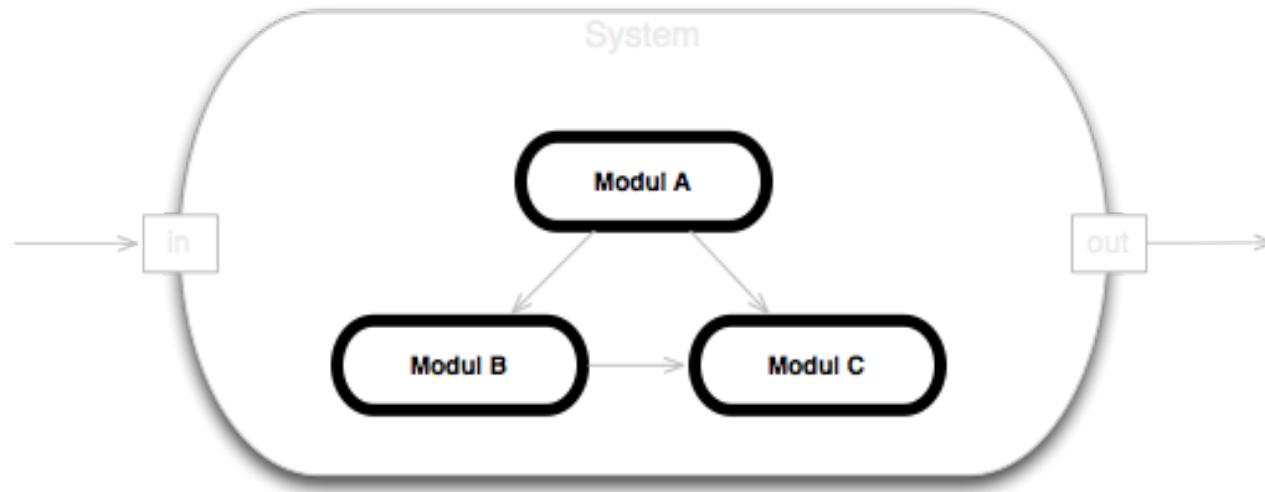
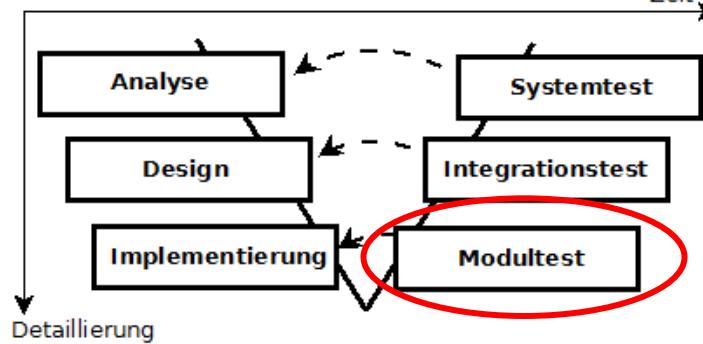
# TESTS AUF VERSCHIEDENEN ABSTRAKTIONSEBENEN



- Es gibt Tests auf verschiedenen Abstraktionsebenen:
  - Modultest: Jedes Modul für sich
  - Integrationstest: Verbindungen zwischen den Modulen
  - Systemtest: Das System als Ganzes
  - Abnahmetest: Abnahmetest des Kunden (oft Systemtest)



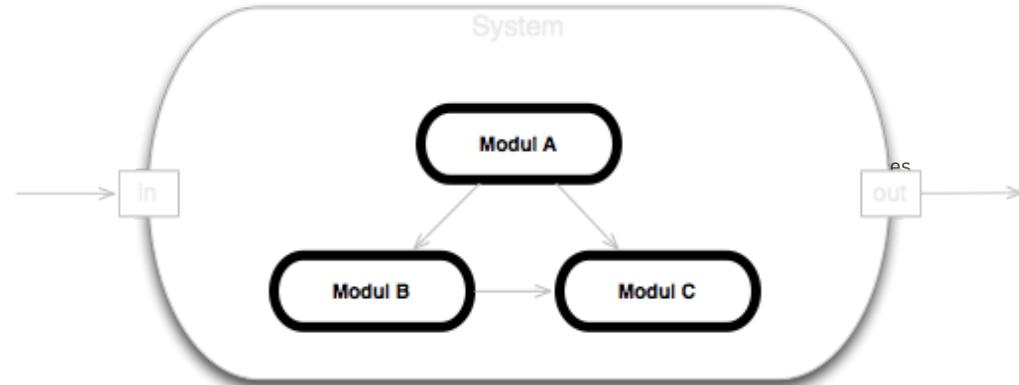
# I. MODULTEST



Merkmale:

- Jedes Modul für sich.
- White-Box-Test &&|| Black-Box-Test
- Test durch Programmierer
- Empfehlenswert: Test First (erst Testfall entwickeln)

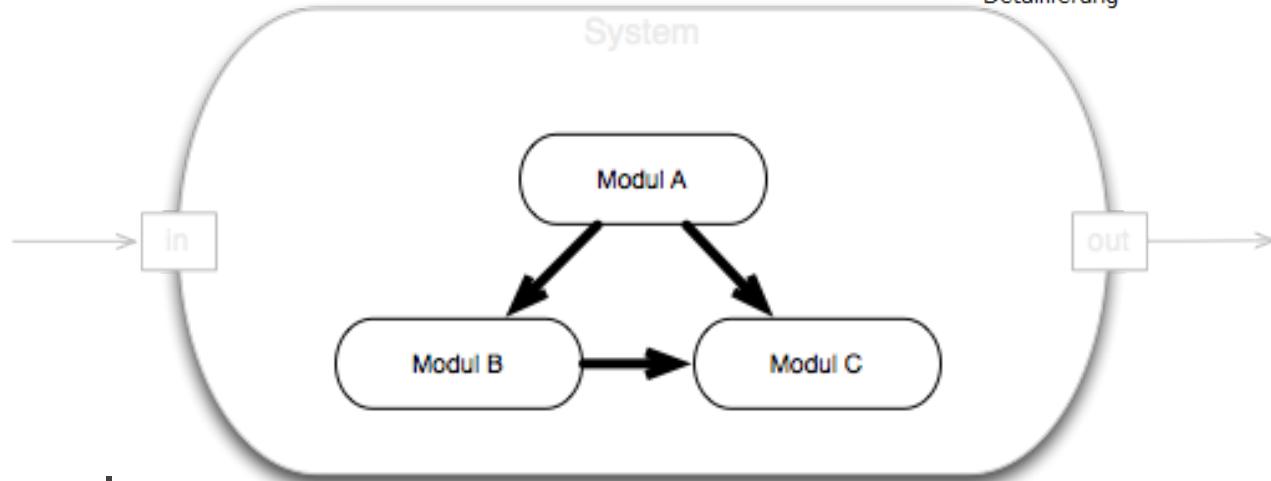
# I. MODULTEST



## Werkzeuge:

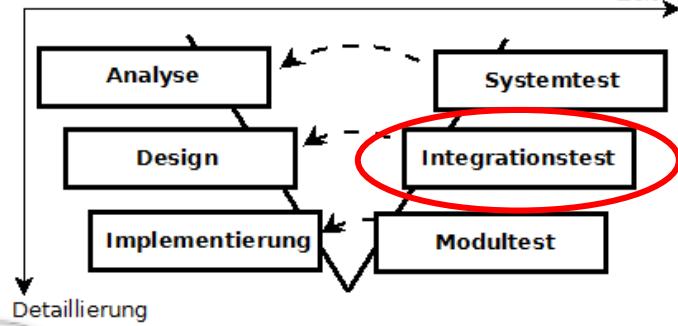
- Empfohlen: JUnit (bzw. \*Unit)
- Zusätzliche Module, die nur für den Test benötigt werden:
  - test stub
    - z.B. im Modultest von B: Testklasse als Ersatz für C  
→ Mocking (Siehe PMT-Folien zu Unittesting)
  - test driver
    - z.B. im Modultest von B: Testklasse als Ersatz für A
      - häufig: test driver = \*Unit-Klasse

## II. INTEGRATIONSTEST

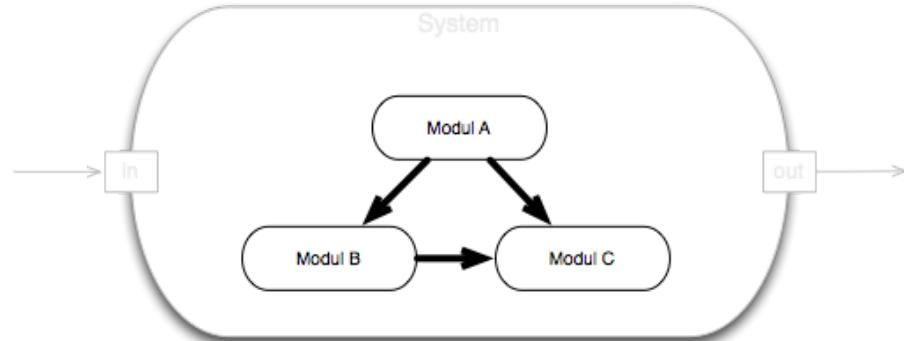


Merkmale:

- Nach dem Modultest
- KEINE Wiederholung des Modultest!
- Fokus: Schnittstellen
- Meist: geeignete Mischung aus White-Box- und Black-Box-Tests
- beliebtes Testwerkzeug: JUnit (\*Unit)



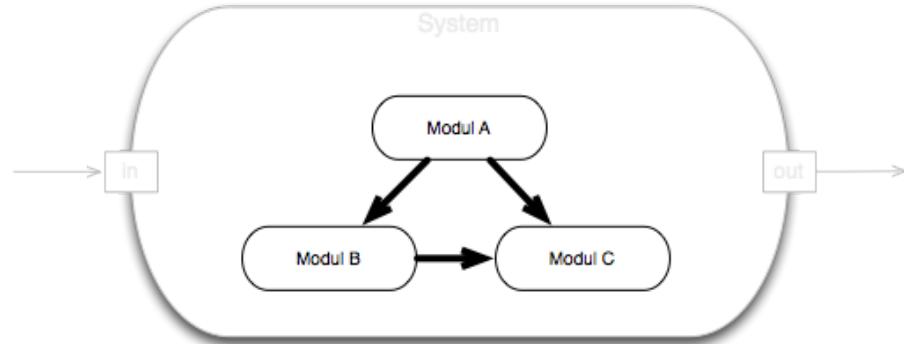
## II. INTEGRATIONSTEST



### Sinnvolle Integrationsansätze:

- bottom-up
  - hier: erst B → C, dann A → (B+C)
- top-down
  - hier: erst A → (B+C), dann B → C
  - i.d.R. test stubs notwendig
- cluster
  - große Modulanzahl → Gruppieren zu Clustern
  - bottom-up oder top-down in jedem Cluster
  - Zusammenfügen der Cluster: bottom-up oder top-down

## II. INTEGRATIONSTEST



Testansätze:

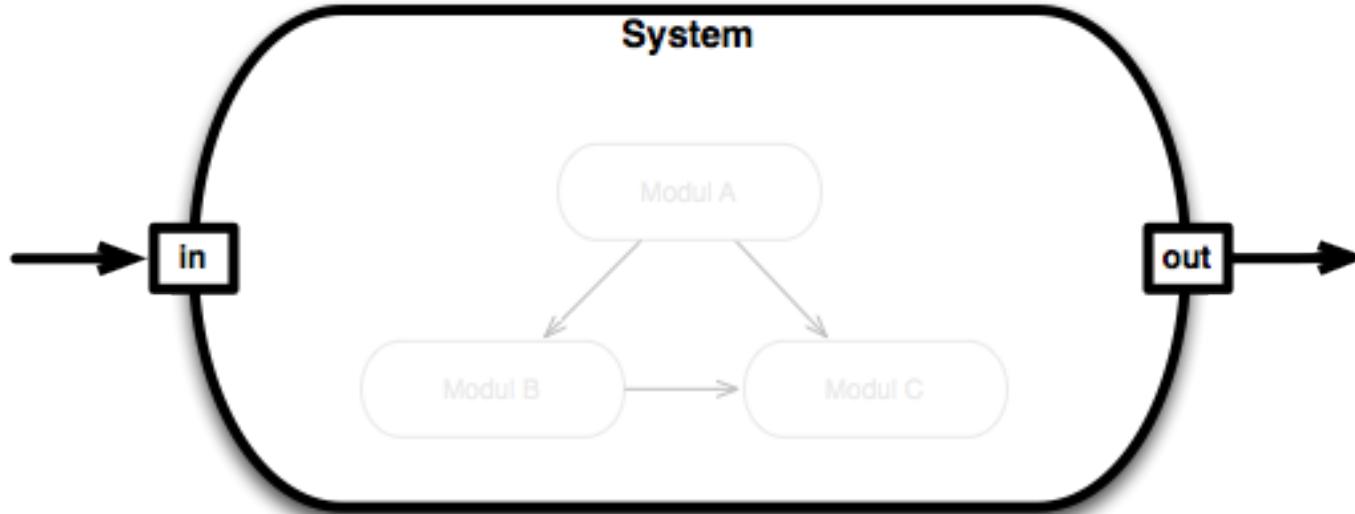
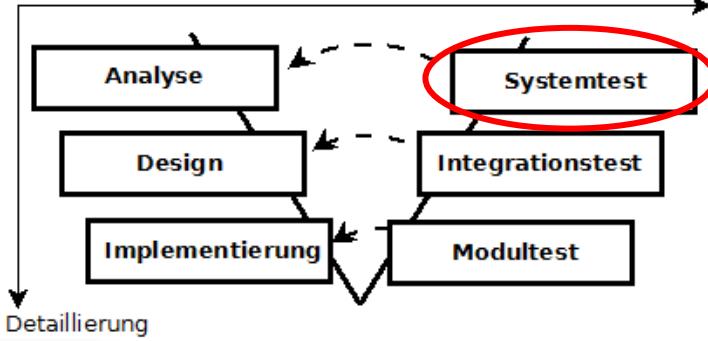
1. Lassen sich die zu integrierenden Module überhaupt „zusammenstecken“ (z.B. gemeinsam kompilieren)?
2. Quellen für Testfälle:
  - für jede betroffene Schnittstelle (SST):
    - 1 Testfall für jede Art des Aufrufs
    - Bsp: SST mit mehreren Methoden → für jede Methode:
      - » 1 Testfall für den Normalfall
      - » 1 Testfall für jeden Ausnahme-/Fehlerfall
  - Sicherstellen, dass die komplette Aufrufhierarchie abgedeckt wird  
→ Jeder noch so tief verschachtelte Methodenaufruf muss mind. 1 Mal getestet werden.



BEM: keine Wiederholung der Modultests

- ABER: ggfs. Wiederverwendung von Bestandteilen der Modultests

# III. SYSTEMTEST

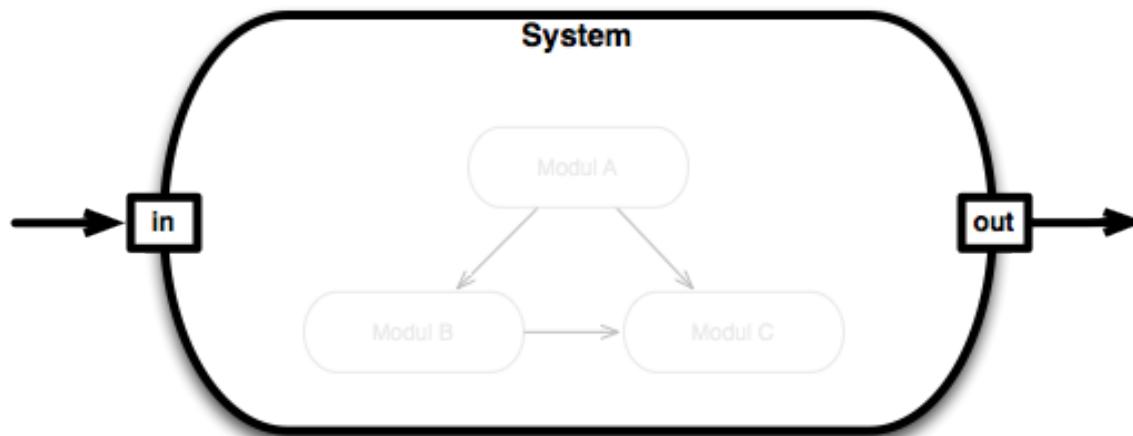


Merkmale:

- Test aus Benutzersicht  
→ Grundlage: Anwendungsfälle, Fachmodell, GUI-Entwurf  
nicht-funktionale Anforderungen, ...
- i.d.R. ausschließlich Black-Box-Tests



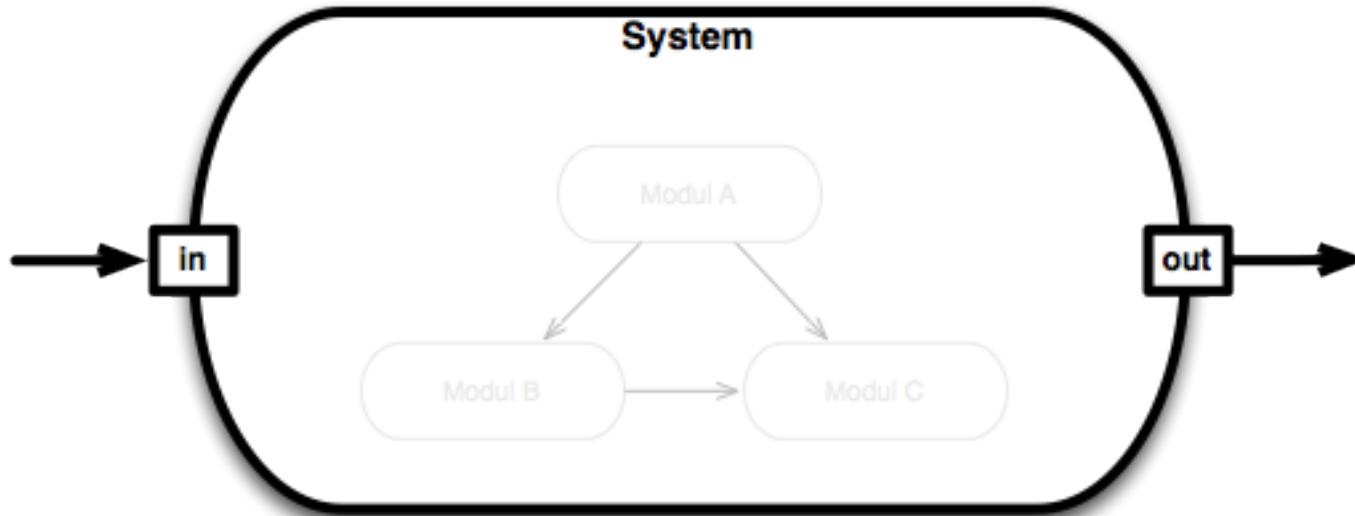
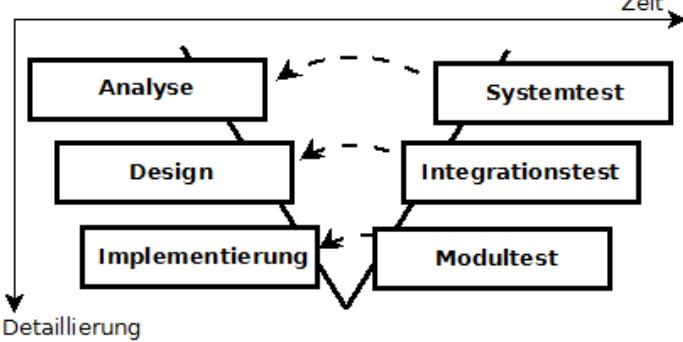
# III. SYSTEMTEST



Typische nichtfunktionale Tests auf der System-Ebene:

- Leistungstest (Volllastverhalten)
- Stressstest (Überlastverhalten)

# IV. ABNAHMETEST (SONDERFALL)



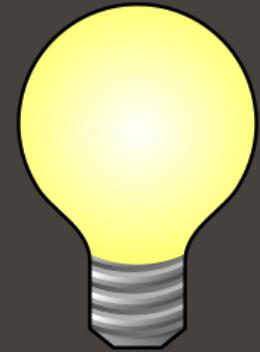
Merkmale:

- Meist: Abnahmetest ≈ Systemtest, aber:
  - Durchführung von/beim Kunden
  - Bei mehreren Installationen:
    - Alpha-Test: Kunde kommt zum SW-Entwickler
    - Beta-Test: SW wird bei Pilotkunden getestet



08

## Testvorbereitung & -durchführung



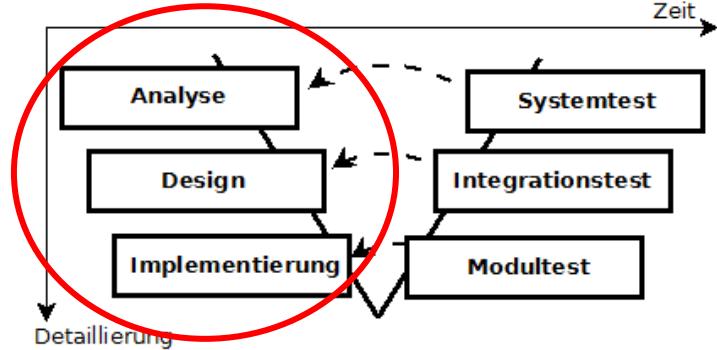
Ziel:

Was muss zur Vorbereitung von Tests passieren?

Was ist bei der Testdurchführung wichtig?

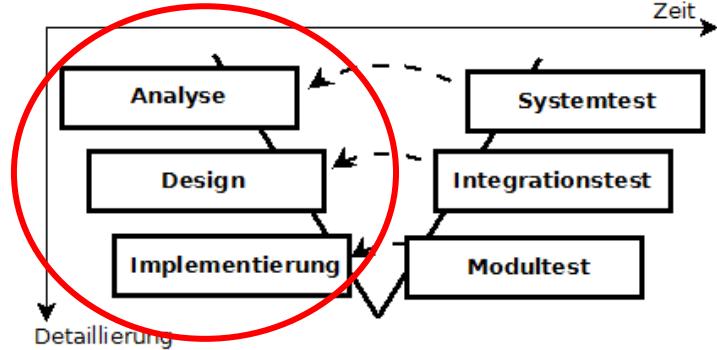
# TESTVORBEREITUNG

- Ziele:
  - einfach durchführbare Tests
  - einfach reproduzierbare Tests
  - vollständige Tests
- Tipp: So früh wie möglich die Tests vorbereiten
  - Tests sind wirklich vorhanden, wenn sie benötigt werden
  - „geschenkte“ frühzeitige Überprüfung der Spezifikation
    - Ungereimtheiten in Spez. werden aufgedeckt



# TESTVORBEREITUNG

- Grober Ablauf:
  - Testfälle erstellen: logisch + konkret  
→ Müssen auch spezifiziert werden
  - Testmethode festlegen: automatisiert oder manuell
  - Testskripte erstellen
  - Testdaten erstellen
  - Wenn automatisiert:
    - Testskripte automatisieren
    - ggfs. Testwerkzeuge erstellen  
test driver, test stub, ...



# TESTVORBEREITUNG – TESTFÄLLE SPEZIFIZIEREN



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Testfälle müssen spezifiziert werden
- Mögliche Beschreibung eines Testfalls:
1. Id
  2. Testsituation
  3. Vorbedingung
  4. Eingabe  
(Wert, Parameter, Benutzeraktion,...)
  5. Randbedingung
  6. Erwartetes Ergebnis  
(Ausgabe, Systemreaktion, ...)

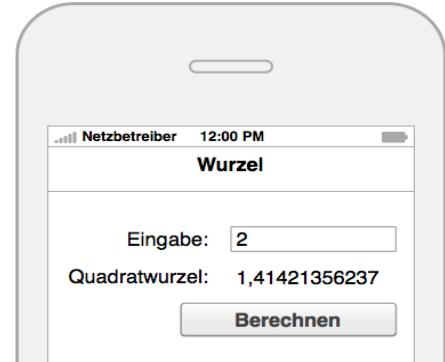
# TESTVORBEREITUNG – TESTFÄLLE SPEZIFIZIEREN



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Testfälle müssen spezifiziert werden
  - Mögliche Beschreibung eines Testfalls:
    1. Id
    2. Testsituation
    3. Vorbedingung
    4. Eingabe  
(Wert, Parameter, Benutzeraktion,...)
    5. Randbedingung
    6. Erwartetes Ergebnis  
(Ausgabe, Systemreaktion, ...)
- Auch wichtig: Wo kommt der Testfall her?
  - Verbindung zu Spezifikation, Programm, ... muss klar erkennbar sein → Traceability

# TESTVORBEREITUNG – TESTFÄLLE BEISPIEL



- Vorher erarbeitete Liste mit log. & konkreten Testfällen:

ID	Art (AK / RW)	Beschreib. der Äquivalenz- klasse	Bedingung „logischer Testfall“)	BSP-Werte „konkreter Testfall“)
AK1	AK	Zahl zu groß	Zahl > MAX	...
RW1	Rand	gültige Eingabe: oberer Rand	Zahl = MAX	MAX+1, MAX, MAX-1
AK2	AK	gültige Eingabe	$0 \leq \text{Zahl} \leq \text{MAX}$	2
		...		

# TESTVORBEREITUNG – TESTFÄLLE BEISPIEL



ID	Art (AK / RW)	Beschreib. der Äquivalenz-klasse	Bedingung („logischer Testfall“)	BSP-Werte („konkreter Testfall“)
AK1	AK	Zahl zu groß	Zahl > MAX	...
RW1	Rand	gültige Eingabe: oberer Rand	Zahl = MAX	MAX+1, MAX, MAX-1
AK2	AK	gültige Eingabe	$0 \leq \text{Zahl} \leq \text{MAX}$	2
		...		

→ Daraus jetzt konkrete Testfälle ableiten:

ID	Vorbed.	Parameter (Testdaten)	Erwartetes Ergebnis
AK1	-	1.000.000.000	Anzeige der Fehlermeldung „Zahl zu groß“ für 2 Sekunden.
RW1	-	999.999.999	31.622,77659
...	...	...	...

# TESTVORBEREITUNG – TESTSKRIPT ERSTELLEN



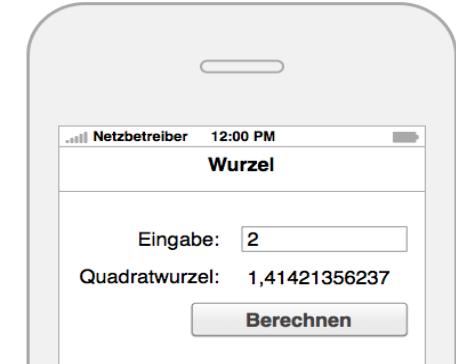
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## Grober Ablauf beim Erstellen eines Testskripts:

1. Testfälle kategorisieren
    - grundlegend (SmokeTest)
    - normal
    - Fehler/Ausnahme
  2. Nötige Testdaten erstellen
  3. Testskript erstellen
    - Testskript = Abfolge konkreter Testschritte für jeden konkreten Testfall
    - Mögliche Aufteilung/Gliederung:
      - Ein Testskript für grundlegende Funktionalität (SmokeTest)
      - Testskripte für normale Funktionalität
      - Testskripte für Fehler-/Ausnahme-Fälle
- Vortest der grundsätzl.  
Basisfunktionen.  
Bei Fehlschlag, werden  
weitere Tests abgebrochen

# TESTVORBEREITUNG – TESTSKRIPT BEISPIEL

- Beispiel Testskript für Wurzelanwendung:



#	TF-ID	Eingabe	Erwartetes Ergebnis	Tatsächl. Ergebnis
0	XY	App „Wurzel“ starten	Anzeige des Splash-Screen für max. 3s. Inhalt Splashscreen: siehe Abbildung 1.	
1		Klick auf Eingabefeld „Eingabe“	Cursor in Eingabefeld „Eingabe“	
2	AK2	Eingabe der Zeichenkette „2“	...	
3	AK2	Klick auf Button „Berechnen“	Resultat im Ausgabefeld „Quadratwurzel“: 1,414213562	
...	...	...	...	

# TESTVORBEREITUNG – TESTSKRIPT ERSTELLEN

#	TF-ID	Eingabe	Erwartetes Ergebnis	Tatsächl. Ergebnis
0	XY	App „Wurzel“ starten	Anzeige des Splash-Screen für max. 3s. Inhalt Splashscreen: siehe Abbildung 1.	
1		Klick auf Eingabefeld „Eingabe“	Cursor in Eingabefeld „Eingabe“	
2	AK2	Eingabe der Zeichenkette „2“	...	
3	AK2	Klick auf Button „Berechnen“	Resultat im Ausgabefeld „Quadratwurzel“: 1,414213562	
...	...	...	...	



- Weitere Hinweise zur Testskripterstellung:
  - Ein Testskript ist konkret und muss (im Prinzip) einfach abgearbeitet werden können.
    - Testspezifikation
  - Sicherstellen, dass kein Testfall vergessen wird:
    - Querverweis Testfall ↔ Testskript (siehe TF-ID)
    - und/oder berücksichtigte Testfälle abhaken.

# TESTVORBEREITUNG – TESTSKRIPTE & TESTEBENEN:



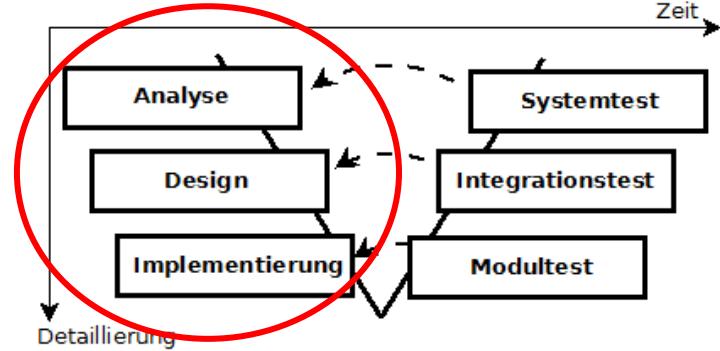
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Wie sehen Testskripte auf verschiedenen Testebenen aus?
  - Modultest:
    - meist: (voll)automatisiert
    - Testskript für Modultest = Code einer \*Unit-Klasse
    - Tipp: Testfälle als Kommentare in \*Unit-Klasse
  - Integrationstest:
    - Meist: Analog zu Modultest
  - Systemtest:
    - Meist: (leider) manuell
      - Automatisierung möglich, aber aufwändig
      - z.B. GUI-Testing (immer mehr im kommen!)
    - Testskript für Systemtest = detaillierte Checkliste mit vorgegebener Reihenfolge
    - Zielsetzung: muss von normalem Benutzer ausführbar sein

# TESTVORBEREITUNG

Möglichkeit I:

Rechtzeitige und  
gute Testvorbereitung



# TESTVORBEREITUNG

Möglichkeit II:

Keine oder  
mangelhafte  
Testvorbereitung



(Leider passiert das sehr oft)

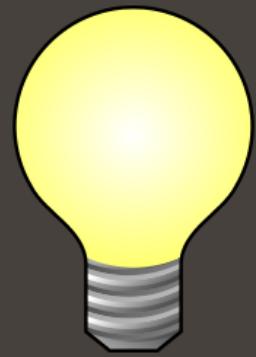


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# 09

# Fazit

Ziel:  
Was haben wir damit gewonnen?



# WAS HABEN WIR GELERNT?



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Testebenen
  - Modultest
  - Integrationstest
  - System- und Abnahmetest
- Testansätze hinsichtlich Sichtbarkeit:
  - Black-Box, White-Box, Grey-Box
- Ansätze zum Finden von Testfällen und Testvektoren:
  - Äquivalenzklassenbildung und Randwertanalyse
  - Kombinieren von Testfällen → Testvektoren bilden
  - Zustandsbasierter Test
- Tests auf verschiedenen Abstraktionsebenen
  - Modultest, Integrationstest, Systemtest, Abnahmetest
- Testdurchführung (Schlüssel zum Erfolg: Gute Vorbereitung!)



# LITERATUR

- Spillner, A.; et al.: Basiswissen Softwaretest. dpunkt Verlag, 2012.
- Kleuker, S.: Grundkurs Software-Engineering mit UML. Vieweg + Teubner. 2009.  
[als eBook über unsere Bibliothek verfügbar]
- Myers, G. J.: The Art of Software Testing. John Wiley and Sons. 1979.  
[fast alle hier vorgestellten grundlegenden Ideen zum Testen gehen auf dieses Buch zurück]
- Projektgruppe Softwaretechnik: Kurzanleitung Testen. THM. 2012  
[[homepages.thm.de/~hg11260/mat/testentwurf.pdf](http://homepages.thm.de/~hg11260/mat/testentwurf.pdf)]



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

AUF GEHT'S!!

SELBER MACHEN UND LERNEN!!



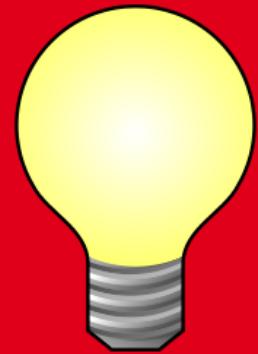


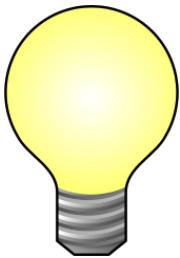
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

11.02.2021

# Programmieren im Grossen VII

Durchführen von Projekten





Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# AGENDA

Einführung ins Thema

Vorgehensmodelle im Detail

Technische Infrastruktur

BSP: Versionsverwaltung

Fazit

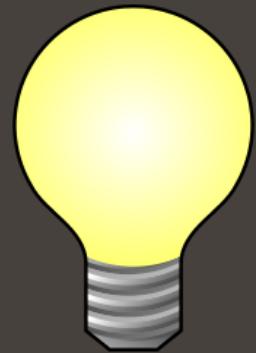


Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# 01

## EINFÜHRUNG INS THEMA

Ziel:  
Die Eckpunkte des Themas kennenlernen



# TYPISCHE FRAGE ZU ANFANG EINES PROJEKTS



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Wie gehe ich bei der Softwareentwicklung am besten vor?
    - Suche und nutze ein geeignetes Vorgehensmodell
  - Muss ich bei der Softwareentwicklung alles mühsam von Hand machen?
    - Nein, es gibt eine Menge nützlicher Werkzeuge, die einen unterstützen
- In dieser Vorlesungseinheit besprechen wir:
- Vorgehensmodelle nochmals im Detail
  - Geeignete Werkzeuge zur Projektunterstützung

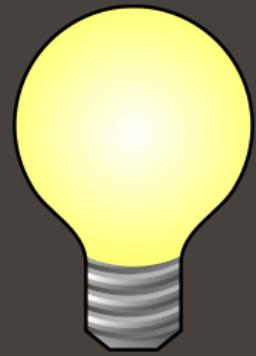


02

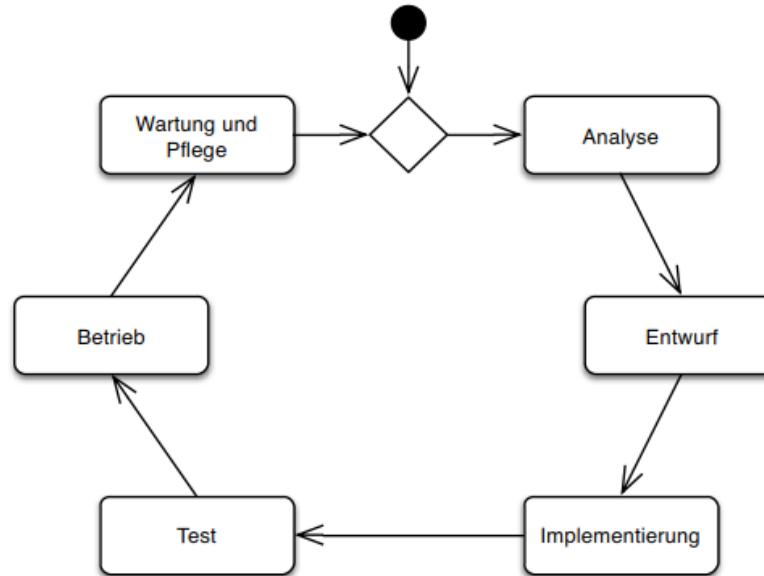
## Vorgehensmodelle im Detail

Ziel:

Nochmals Details zu den Vorgehensmodellen



# BSP FÜR VORGEHENSMODELLE – LEBENSZYKLUS VON SOFTWARE



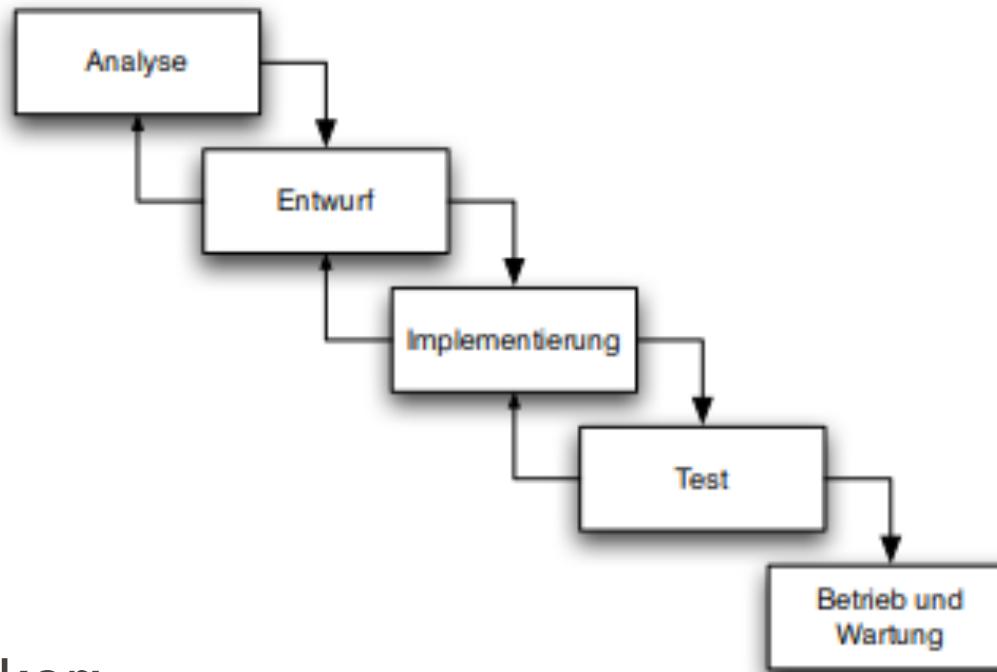
→ Beschreibt die typischen Tätigkeiten bei der SW-Entwicklung

- einfache, klare Struktur
- sehr schlichtes Vorgehensmodell
- wichtigste Aussagen:
  - Software-Entwicklung umfasst 6 grundlegende Tätigkeiten
  - zyklisch
    - („Nach dem Spiel ist vor dem Spiel.“ Sepp Herberger)

# BSP FÜR VORGEHENSMODELLE – WASSERFALLMODELL



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim



→ Der Klassiker:

- schwergewichtig & Big Bang Integration
- einfache, klare Struktur
- erlaubt: Rückschritt
- nicht erlaubt: Überschneidung der Phasen

# VORGEHENSMODELLE ALLGEMEIN



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Ein Vorgehensmodell gibt vor:
  - Reihenfolge des Arbeitsablaufs
  - Aktivitäten und Artefakte
  - Fertigstellungskriterien
  - Verantwortlichkeiten, nötige Kompetenzen und Qualifikationen
  - Standards, Richtlinien, Methoden und Werkzeuge
  - . . .
- Ziemliche Unterschiede jedoch in der Ausführlichkeit:
  - Die Bandbreite reicht von „detaillierten“ (umfassende Liste an Vorgaben) bis hin zu „sehr groben“ Vorgehensmodellen
    - Bsp: „Software Life Cycle“: sehr grob (zu allgemein für ein echtes Vorgehensmodell)

# GANZ GROBE EINORDNUNG DER VORGEHENSMODELLE



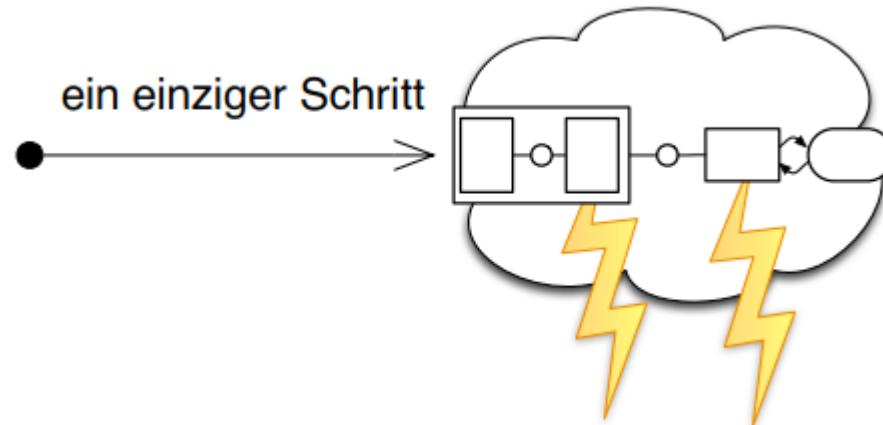
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Schwergewichtig
  - Planung sehr wichtig — spätere Veränderungen unerwünscht
  - Stark dokumenten-orientiert (→ Spezifikationen)
  - Stark ausgeprägtes Phasenkonzept mit Meilensteinen
- Agil / Leichtgewichtig
  - Schnelle Reaktion auf Veränderungen sehr wichtig
  - Konzentration auf das Wesentliche, um Mehrwert für den Kunden herzustellen
    - Eher Menschen-/Kunden-orientiert
    - Planung + Dokumentation eher nicht so wichtig
- Andere irgendwo dazwischen
  - ...

# HÄUFIG VERWENDETE SCHLAGWÖRTER



- Big Bang-Integration: Alles auf einen Schlag (Knall)



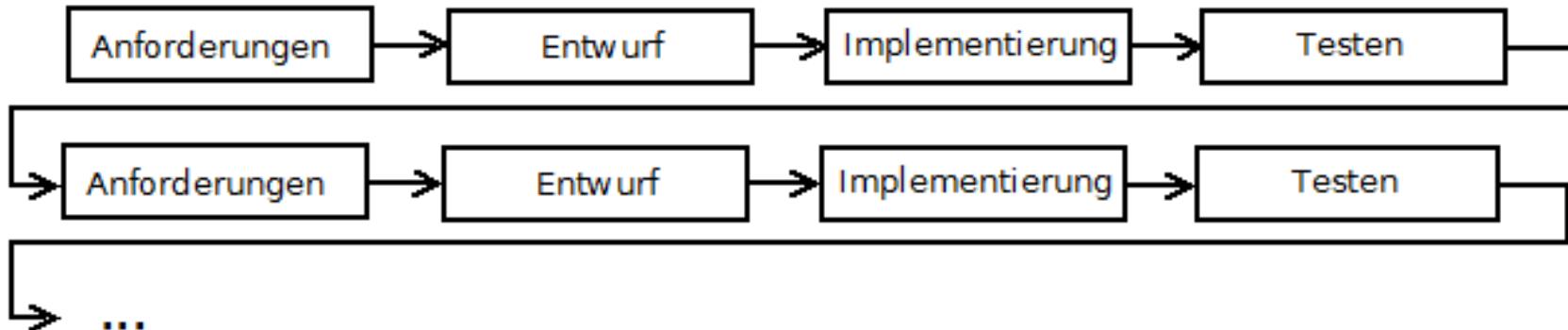
- Inkrementell: Software wächst in klein(er)en Schritten

- 
- 
- 
- ...

# HÄUFIG VERWENDETE SCHLAGWÖRTER



- Iterativ: wiederholte Anwendung einer Vorgehensweise

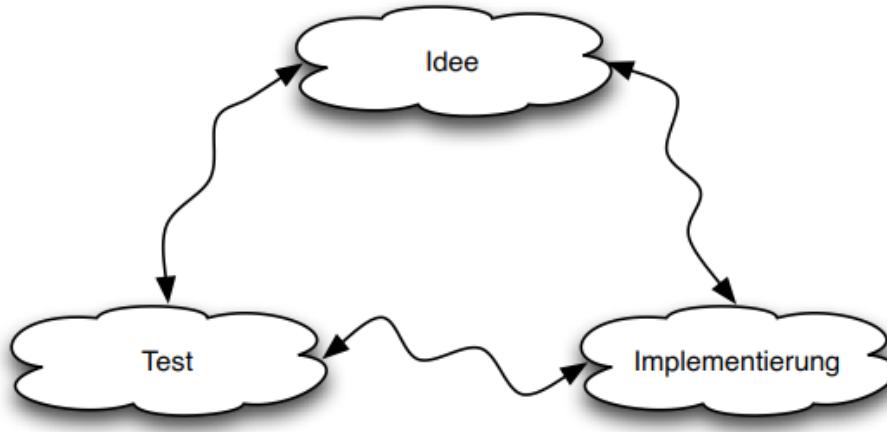


- Wichtige Bemerkungen:
  - Iterativ ermöglicht inkrementell
  - agil → iterativ und inkrementell + leichtgewichtig
  - Es gibt aber auch schwergewichtige Vorgehensmodelle, die iterativ und inkrementell sind

# BSP FÜR VORGEHENSMODELLE – BUILD-AND-FIX-CYCLE



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim



→ Sehr beliebt aber problematisch:

- Streng genommen kein Vorgehensmodell (zu allgemein)
- Verwandte Ansätze/Andere Namen:
  - Versuch und Irrtum (Trial and Error)
  - „Basteln“, „Hacken“, ...
- Sehr beliebt
- Manchmal sinnvoll: Neues ausprobieren, Prototypen, ...
- Sonst: problematisch
  - wächst einem schnell über den Kopf

# BSP FÜR VORGEHENSMODELLE – SPIRALMODELL



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Entwickelt von Barry Boehm
  - Iterativ & inkrementell

## 1. Festlegen der Ziele

## 2. Beurteilen von Alternativen, Risikoanalyse

Zustimmung  
durch  
Überprüfung

Kosten

Fortschritte

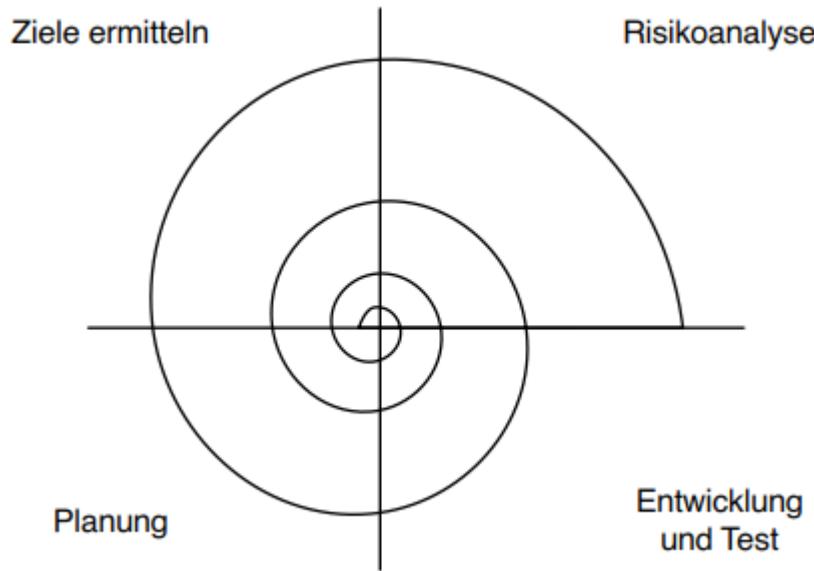
## 4. Planung des nächsten Zyklus

## 3. Entwicklung und Test

# BSP FÜR VORGEHENSMODELLE – SPIRALMODELL



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

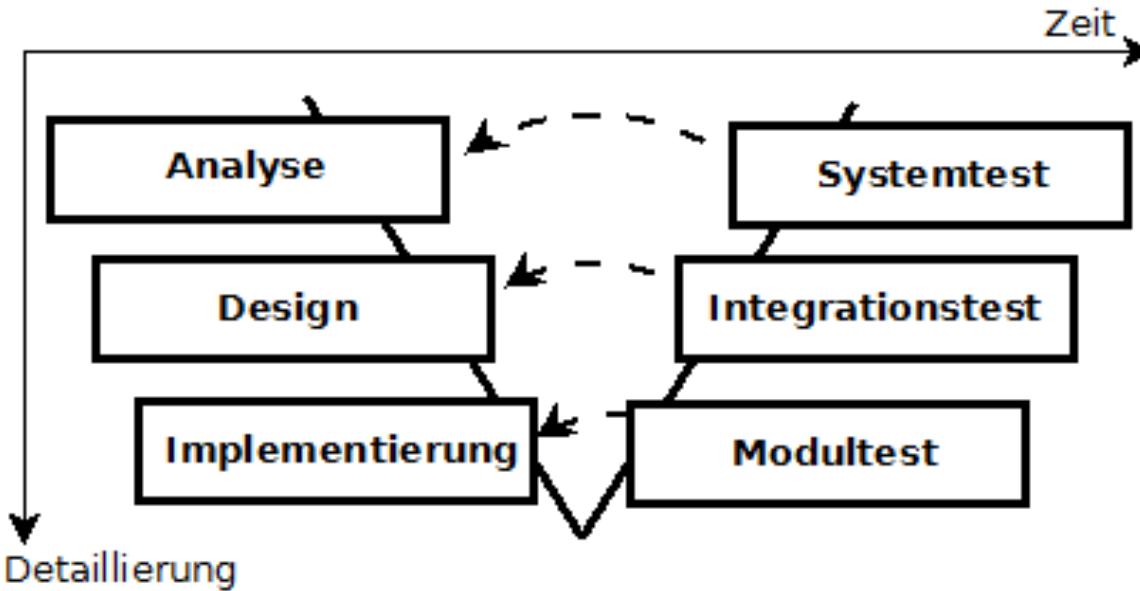


→ Beschreibung des „Prozesses der SW-Entwicklung“:

- Streng genommen kein Vorgehensmodell
  - (eher: Prozessmodell)
- Fokus: Risiken minimieren
- Iterativ & inkrementell
- in jeder Iteration:
  - Einbettung eines (anderen) Vorgehensmodells

# BSP FÜR VORGEHENSMODELLE – V-MODELL

- Abwandlung des Wasserfall-Modells
  - Deutsche Erfindung (TU München → siehe Manfred Broy)
  - oft in Deutschland benutzt, z.B. Behörden, Automotive, ...



- Leider sehr starr
- Zunächst nicht iterativ geplant → Nur ein Zyklus

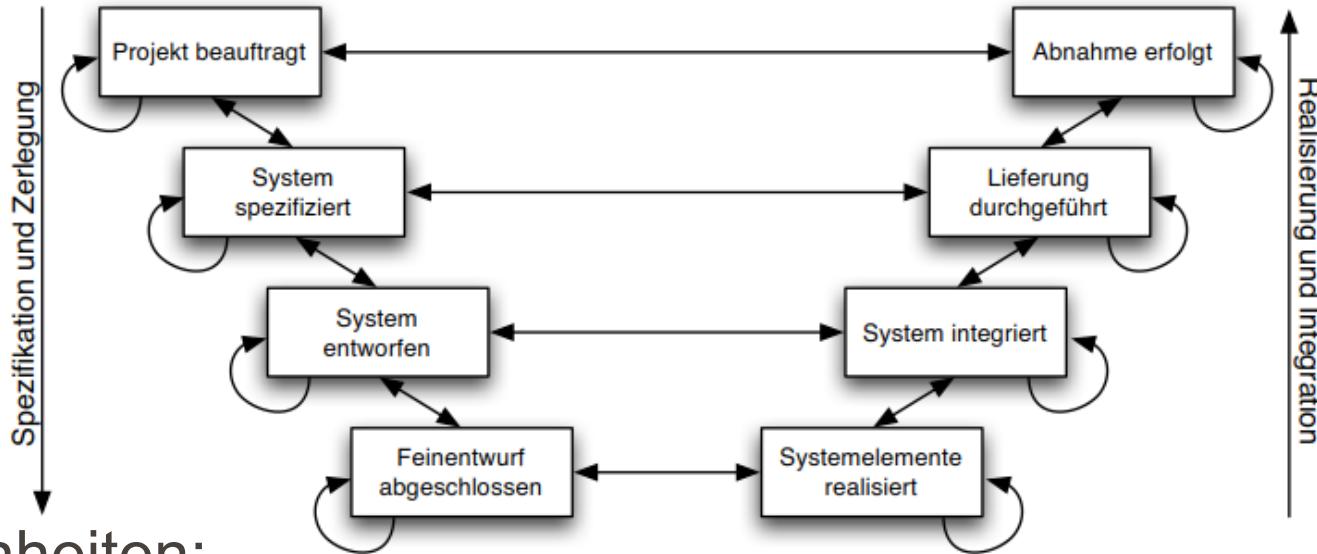
# BSP FÜR VORGEHENSMODELLE –

## V-MODELL



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Auch meilensteinorientierte Betrachtung möglich:



→ Eigenheiten:

- Schwerpunktig & Big Bang-Integration
- Abgewandeltes Wasserfallmodell
- Fokus: Qualitätssicherung
- Sehr komplex/kompliziert
- Teilweise zwingend vorgeschrieben (z.B. Behörden, Automot.)

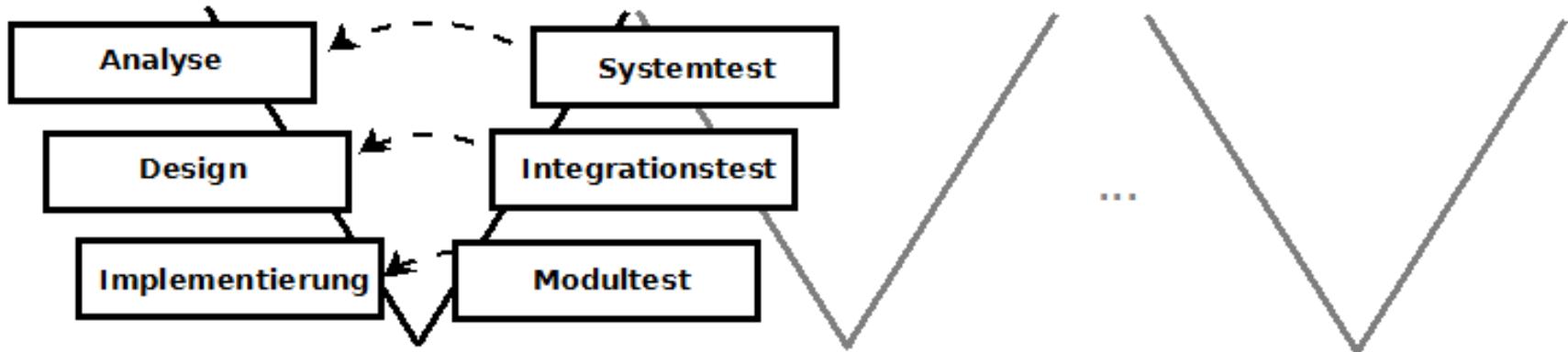
# BSP FÜR VORGEHENSMODELLE –

## V-MODELL



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Weiterentwicklung für iterative Entwicklung:



- Redesign (seit 2005): V-Modell XT
  - Iterativ, inkrementell, wesentlich flexibler und besser skalierbar (Reaktion auf RUP, Spiralmodell & Agile Methoden)

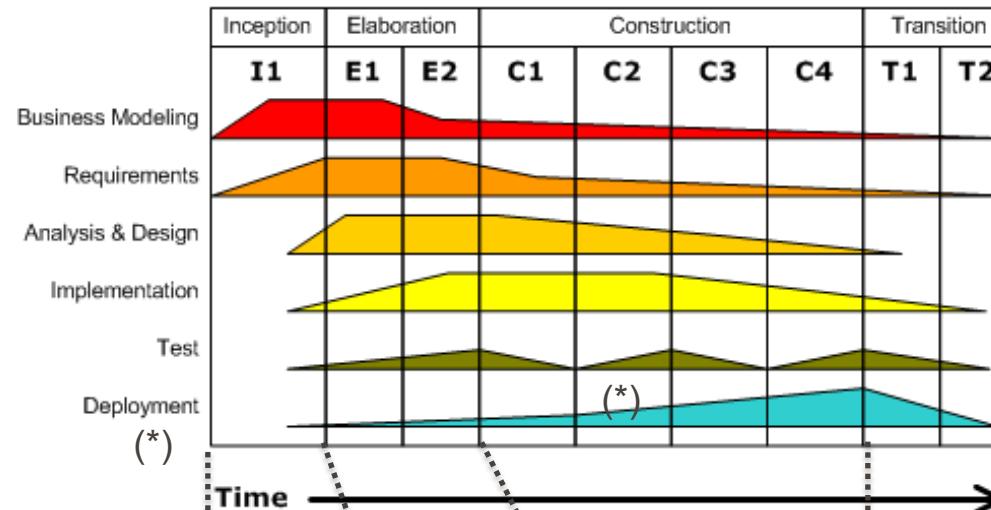
# BSP FÜR VORGEHENSMODELLE – (RATIONAL) UNIFIED PROCESS



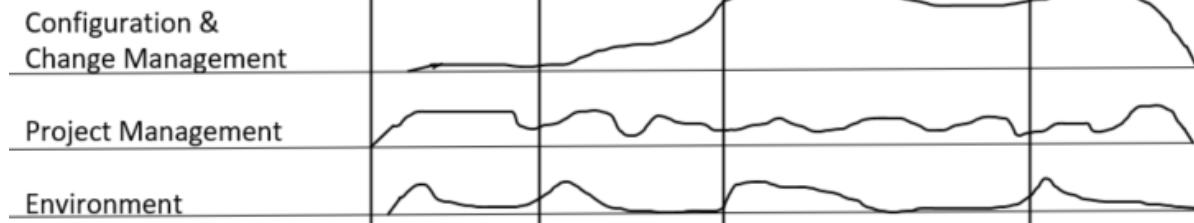
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## Iterative Development

Business value is delivered incrementally in time-boxed cross-discipline iterations.



(\*) Frühere Bilder hatten noch folgende Prozesse:



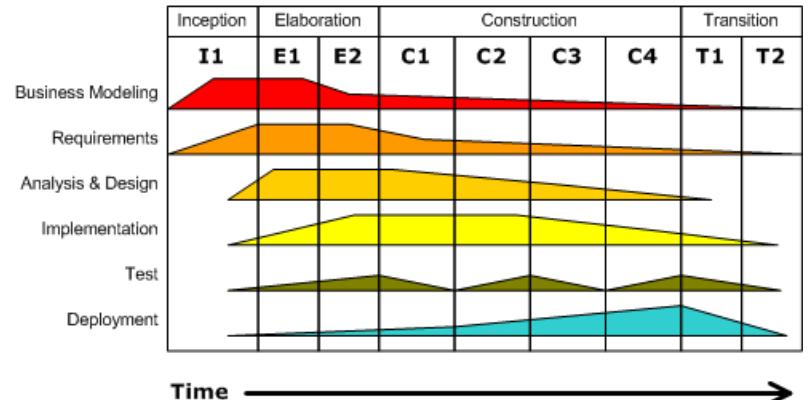
# BSP FÜR VORGEHENSMODELLE – (RATIONAL) UNIFIED PROCESS



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

## Iterative Development

Business value is delivered incrementally in time-boxed cross-discipline iterations.



## → Charakteristika:

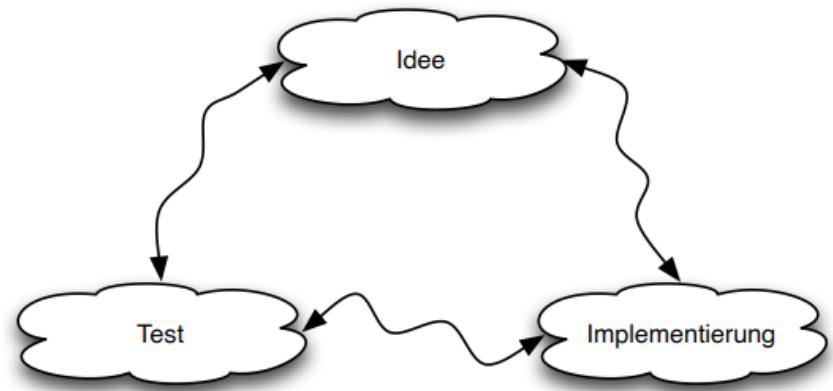
- Schwerpunktig
- Iterativ und inkrementell
  - In jeder Iteration werden hinzugefügt: Weitere Use Cases, weitere Details (z.B. alternative Ablaufschritte), weitere Anwendungsbestandteile, . . .
- Objektorientiert (UML)
- Anwendungsfall-orientiert
- Architektur-zentriert



# AGILE METHODEN

- „Relativ“ neu
  - Abkehr von starren Prozessen
  - Keine richtigen Vorgehensmodelle in klassischen Sinn
  - Eher Sammlung von Prinzipien (Best Practices)
    - Muster-Idee → Prozessmuster (siehe Vorlesung zu Mustern)
  - Siehe auch: **Agiles Manifest**
- Typische Beispiele:
  - eXtreme Programming (XP) → Für Entwicklung
    - Nach Win Vista-Katastrophe hat Microsoft auf XP gesetzt → Win 7
  - SCRUM
    - Eigentlich eher eine Projektmanagementmethode
    - Für Entwicklung kann z.B. XP genutzt werden oder anderes
    - Derzeit richtig „IN“

# BUILD-AND-FIX-CYCLE ↔ AGILE METHODEN



→ Agilen Methoden werden gern mit verwechselt

- „Agil“ wird gerne als Schlagwort verwendet, aber es wird dabei „Hacking“ / Build-And-Fix-Cycle gemeint
- Zugegeben: Agile Methoden funktionieren sehr ähnlich
- ABER bei Agilen Methoden gilt:

- Best Practices greifen wie Zahnräder ineinander
  - Best Practices → Prozessmuster!
  - Z.B.: Kontinuierliches autom. Testen ↔ Refactoring ↔ ...

Wirkungen  
verstärken sich  
gegenseitig

→ Agile Methoden verlangen **viel Eigendisziplin** des Entwicklers alle Bausteine einzuhalten → sonst zerfällt das „Getriebe“

→ Hier z.B. müssen dann für alle „Ideen“ auch autom. Tests erstellt werden, es müssen Refactorings stattfinden, ...

→ Hängt sehr stark von „guten“ Leuten ab

# BSP FÜR AGILE METHODEN – EXTREME PROGRAMMING (XP)



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

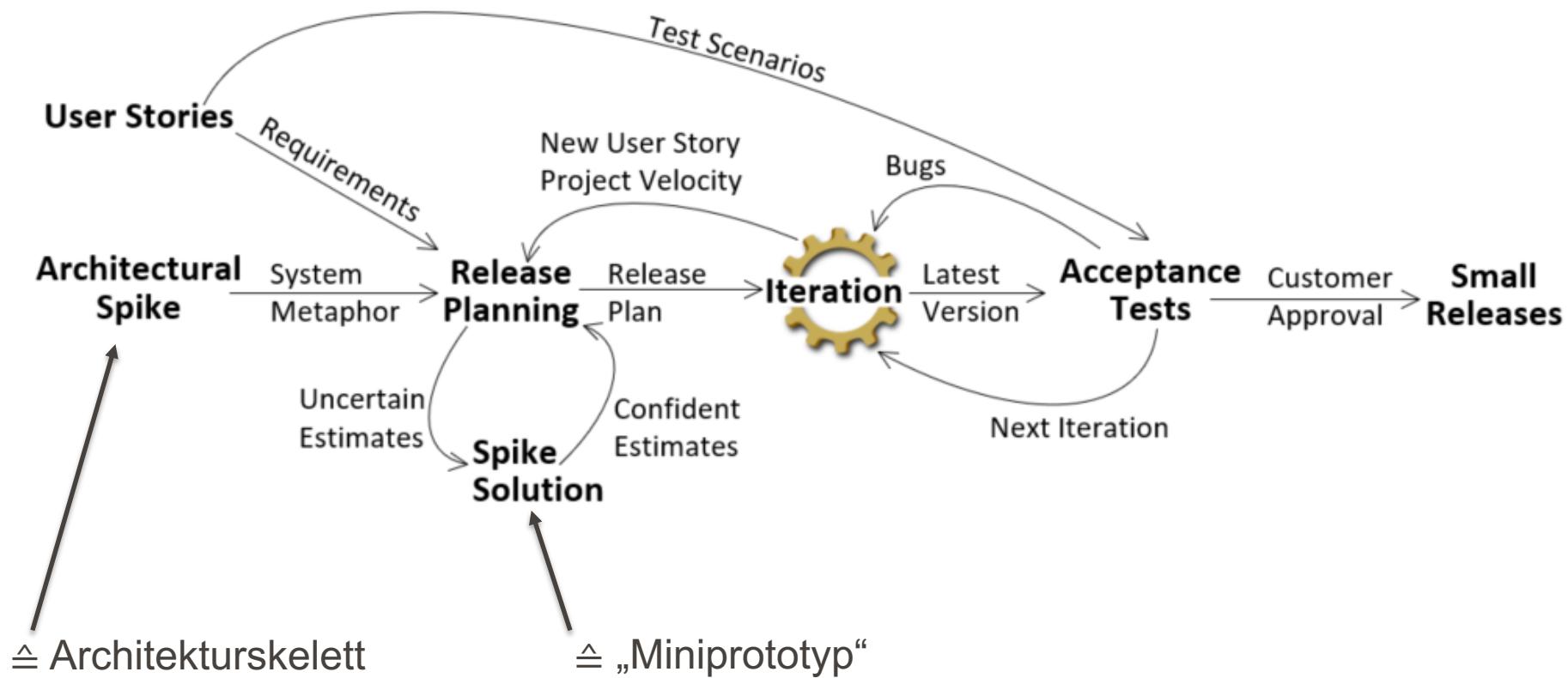
- Häufiges Missverständnis:
  - „extreme“  $\hat{=}$  extrem schlampige Programmierung / „Hacking“
- Eigentliche Bedeutung von „eXtreme“
  - „Take best practices to the extreme!“  
→ Best Practice-Gedanke leben
- 12 Prinzipien (Best Practices):
  - Testen ist gut → Test-Driven Development
    - (Testfirst & Testautom.)
  - Frühes Integrieren ist gut → Continuous Integration
  - Code Review ist gut → Pair Programming
  - Kommunikation mit Kunde wichtig → Kunde ist Teil des Teams
  - ...

# BSP FÜR AGILE METHODEN – EXTREME PROGRAMMING (XP)



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- „Artefakt- & Prozessmodell“:



# BSP FÜR AGILE METHODEN – EXTREME PROGRAMMING (XP)



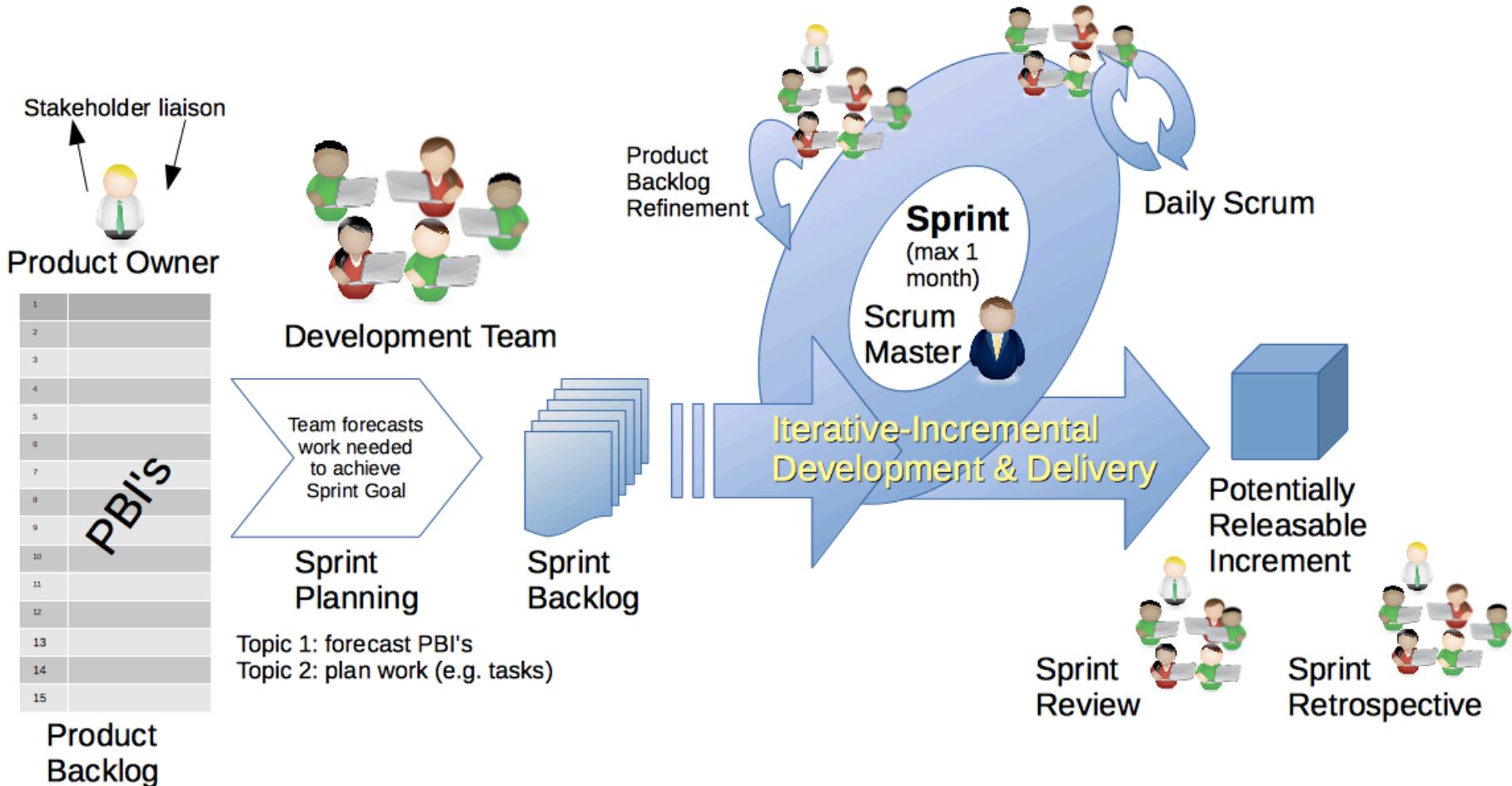
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Anforderungen an das Team:
  - **hohes Maß an Eigendisziplin**
  - geeignete Qualifikation → gute Leute essentiell!
- Weitere Merkmale von eXtreme Programming
  - Agil (erste richtig populäre agile Methode)
  - Anforderungen werden nur kurz als sog. User Stories skizziert
  - Veränderung von Anforderungen ist kein Problem (“Embrace Change”)
  - Iterativ und inkrementell → jede Iteration als „Time Box“
    - Evolutionärer Prototyp wird zum System
    - Immer lauffähiges System zur Hand
    - Möglichst alle Tests werden automat. und immer auf “grün”
  - Kurzer aber vollständiger Entwicklungszyklus → Greifbares Ergebnis
  - Aktivitäten: coding, testing, listening, designing
  - Definierte Rollen

# BSP FÜR AGILE METHODEN – SCRUM



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim



**PBI's = Product Backlog Items**  
(z.B.: User Stories, Epics, Change Requests, Bugs, ...)

# BSP FÜR AGILE METHODEN – SCRUM



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Weitere Agile Methode → derzeit sehr „IN“
  - Eigentl. kein Vorgehensmodell, sondern PM-methode
  - SCRUM kann mit anderen Meth. (z.B. XP) kombiniert werden
- Konzepte:
  - Kurzer Entwicklungszyklus (Sprint) mit Zwischenetappen
    - 4 Wochen: 1 Sprint (ein Inkrement/Entwicklungszyklus)
    - 1 Woche: Weekly Scrum
    - 1 Tag: Daily Scrum (Meeting zum Lösen der Probleme des Tags)
  - Anforderungen werden als User Stories nur kurz skizziert
  - Sammlung im Backlog → Priorisierung der User Stories
  - Zu Beginn eines Sprints werden die wichtigsten User Stories aus dem Backlog genommen und in den Sprint Backlog übernommen und geplant

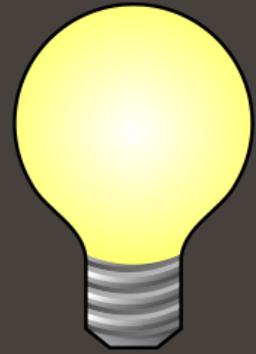


03

## Technische Infrastruktur

Ziel:

Geeignete Werkzeuge zur Unterstützung in Projekten  
kennen und nutzen



# TECHNISCHE INFRASTRUKTUR – WOFÜR BRAUCHEN WIR DAS?



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Was meint technische Infrastruktur?
  - Sammlung an Werkzeugen (Programmen), die bei der SW-entwicklung helfen
  - Arbeiten hoffentlich so zusammen, dass Sie eine wirkliche Infrastruktur bilden
    - Möglichst keine Brüche
- Wofür ist das wichtig?
  - Effizientes Arbeiten & Strukturierung
  - Kollaboratives Arbeiten im Team
  - Bewältigung der aus den verschiedenen Artefakten und der daran arbeitenden Menschen entstehenden Komplexität

# TECHNISCHE INFRASTRUKTUR



- Die wichtigsten Werkzeuge in einem Entwicklungsprojekt:
  - Integrierte Entwicklungsumgebung (Eclipse IDE, NetBeans, . . .)
  - Testautomatisierung (JUnit, NUnit, PHPUnit, . . . )
  - Versionsverwaltung (Subversion (svn), Git (derzeit sehr populär))  
→ Wichtig für Konfigurationsmanagement (siehe folgendes Kap.)
  - Build-Automatisierung (make, ant, gradle, . . .) → DevOps
  - Continuous Integration Server (Cruise Control, Trac)
  - Issue-Tracking (Bugzilla, Mantis, . . . ), Backlog-Verwaltung
  - Projektmanagement (Open Workbench, MS Project, . . . )
- Empfehlung:
  - Nicht zu viele **neue** Werkzeuge auf ein Mal
  - Lieber nach und nach einführen

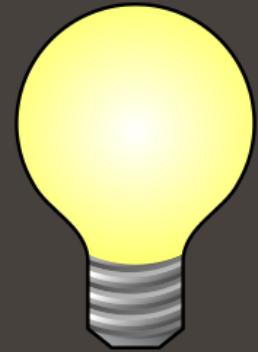


# 04

## BSP: Versionsverwaltung

Ziel:

Nutzen von Versionsverwaltungssystemen kennenlernen  
→ Konfigurationsmanagement kennenlernen



# VERSIONSVERWALTUNGS-WERKZEUGE



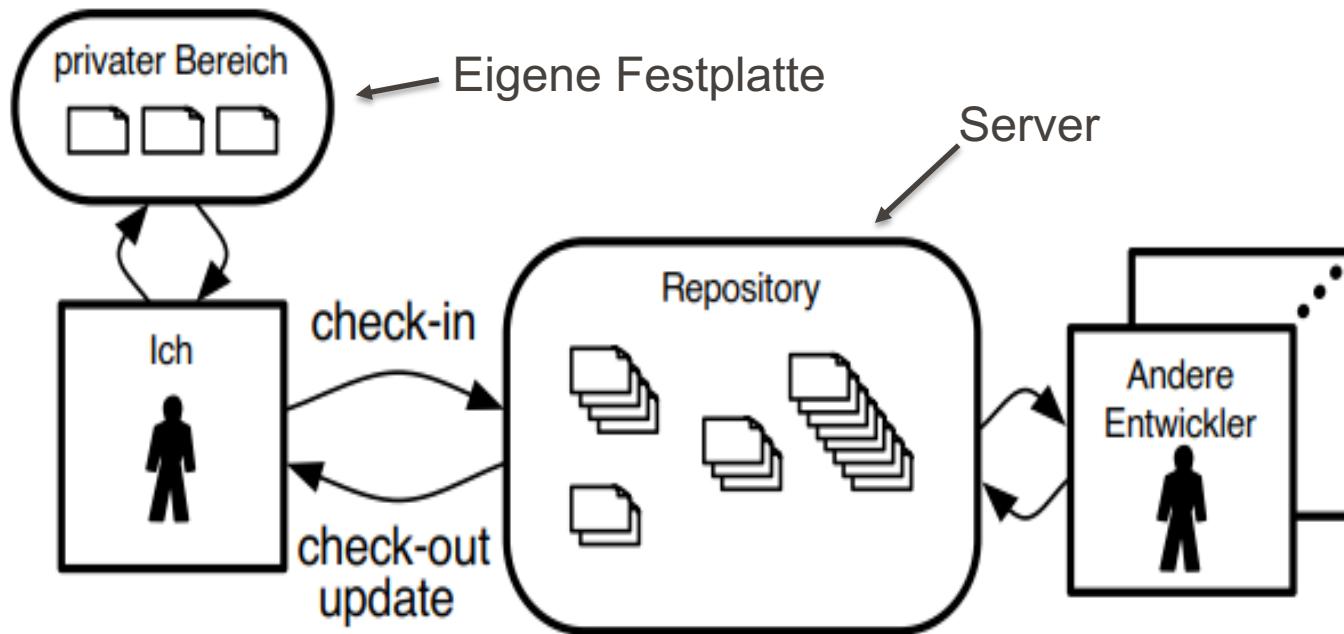
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Wofür brauchen wir eine Versionsverwaltung?
  - Verwaltung aller Artefakte mit den verschiedenen Versionen
    - Backup-Restore
      - Wiederherstellen eines Standes, falls etwas kaputt gegangen ist
    - Nachvollziehbarkeit von Veränderungen
      - Stand der Datei X vor dem letzten Release
    - Parallele Entwicklung ermöglichen
    - Rekonstruktion vorheriger Konfigurationen
      - z.B. Quellcode-Dateien für Binaries, die mit Release XY geliefert wurden.
- Versionsverwaltung ist **essentieller Bestandteil einer professionellen Softwareentwicklung**
  - Nur Amateure & Bastler arbeiten ohne!!

# VERSIONSVERWALTUNGS-WERKZEUGE



- Bekannte Versionsverwaltungen:
  - Subversion (SVN) – Open Source, immer noch oft verwendet
  - Git – Open Source + neuere –revolutionäre– Konzepte
- Grober Aufbau einer „klassischen“ Versionsverwaltung:



# VERSIONSVERWALTUNGS-WERKZEUGE



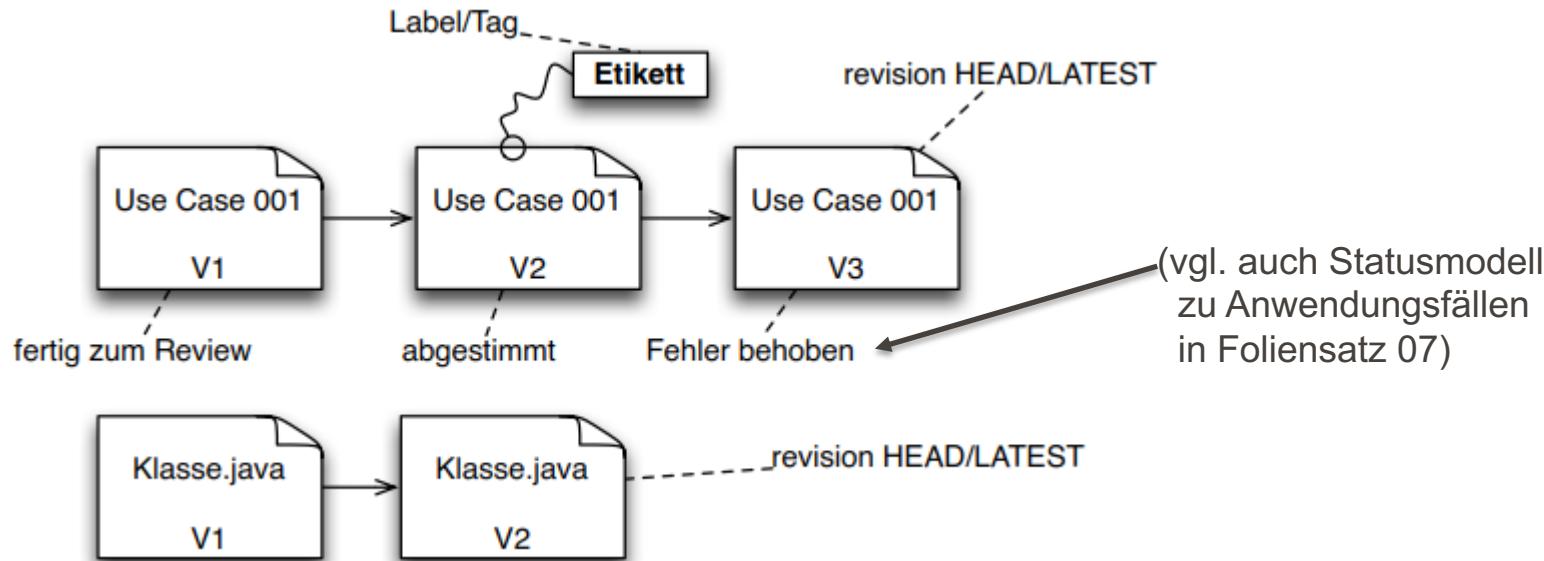
Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Grundlegende Anforderungen:
  - Transparenz der Änderungen
    - Veränderung: Wer? Wann? Was?
  - Simultaner Zugriff + Konfliktlösung
  - Rekonstruktion
    - jederzeit beliebige Revision eines beliebigen Artefakts
    - jederzeit beliebige Konfiguration (alle Dateien zu best. Stand)
  - Baselining / Labeling
    - Einfrieren von Ständen verschiedener Dateien zueinander (z.B. alle Stände der Dateien, die bei einem Release betroffen sind)  
→ Bekommen ein sog. Label

# WIE FUNKTIONIERT DIE VERSIONIERUNG VON DATEIEN?



- Für jede Datei / Verz. wird ein Repräsentant (Head) angelegt
- Für jede Version der Datei / Verz. → sog. Revision erzeugt
  - Die Revision enthält die Datei in der jeweiligen Version
  - Daraus entsteht dann jeweils ein Historienbaum für jede Datei:

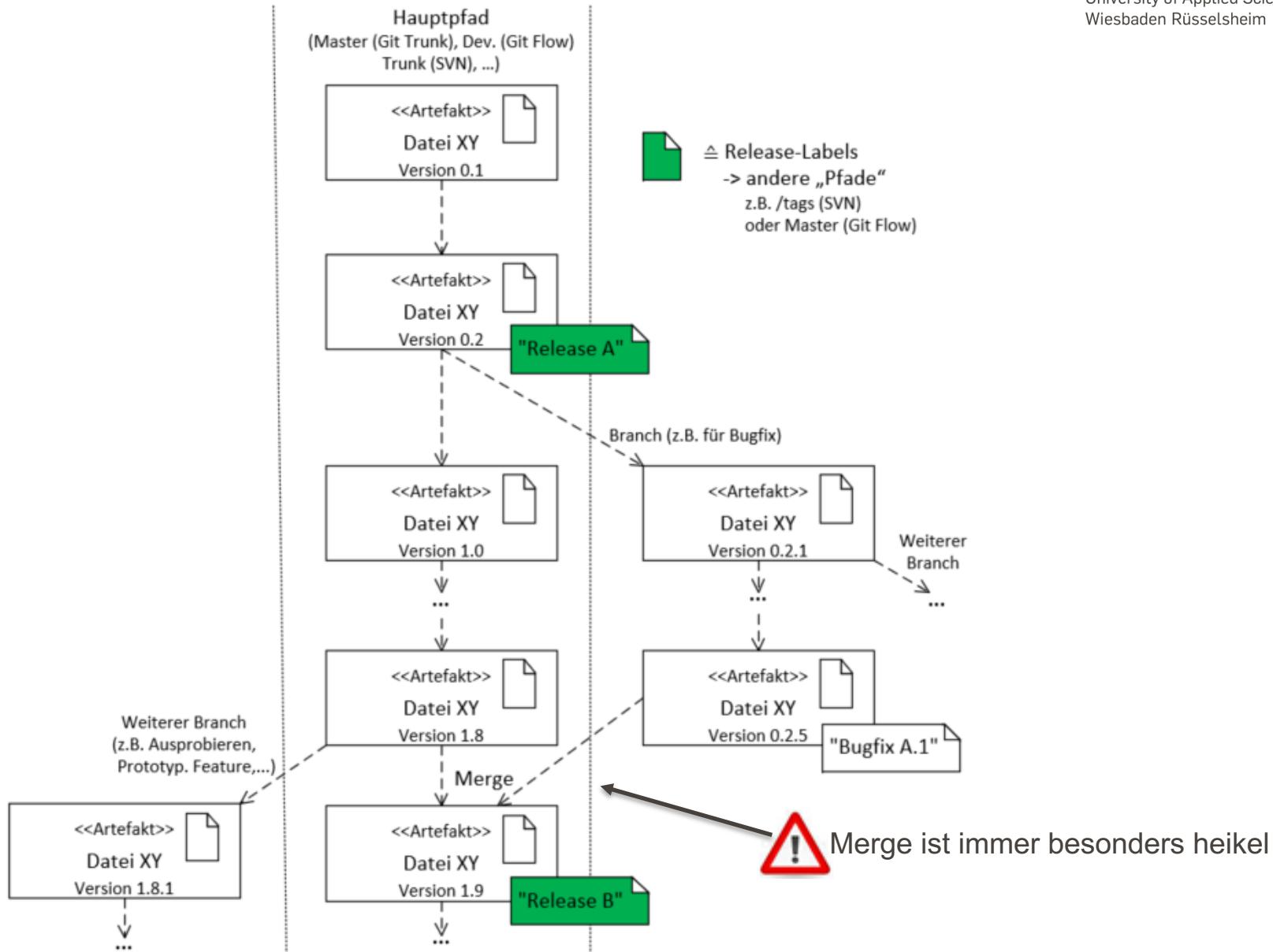


- Jede Revision hat eine Revisionsnummer und kann z.B. durch Labels/Tags weiter gekennzeichnet werden

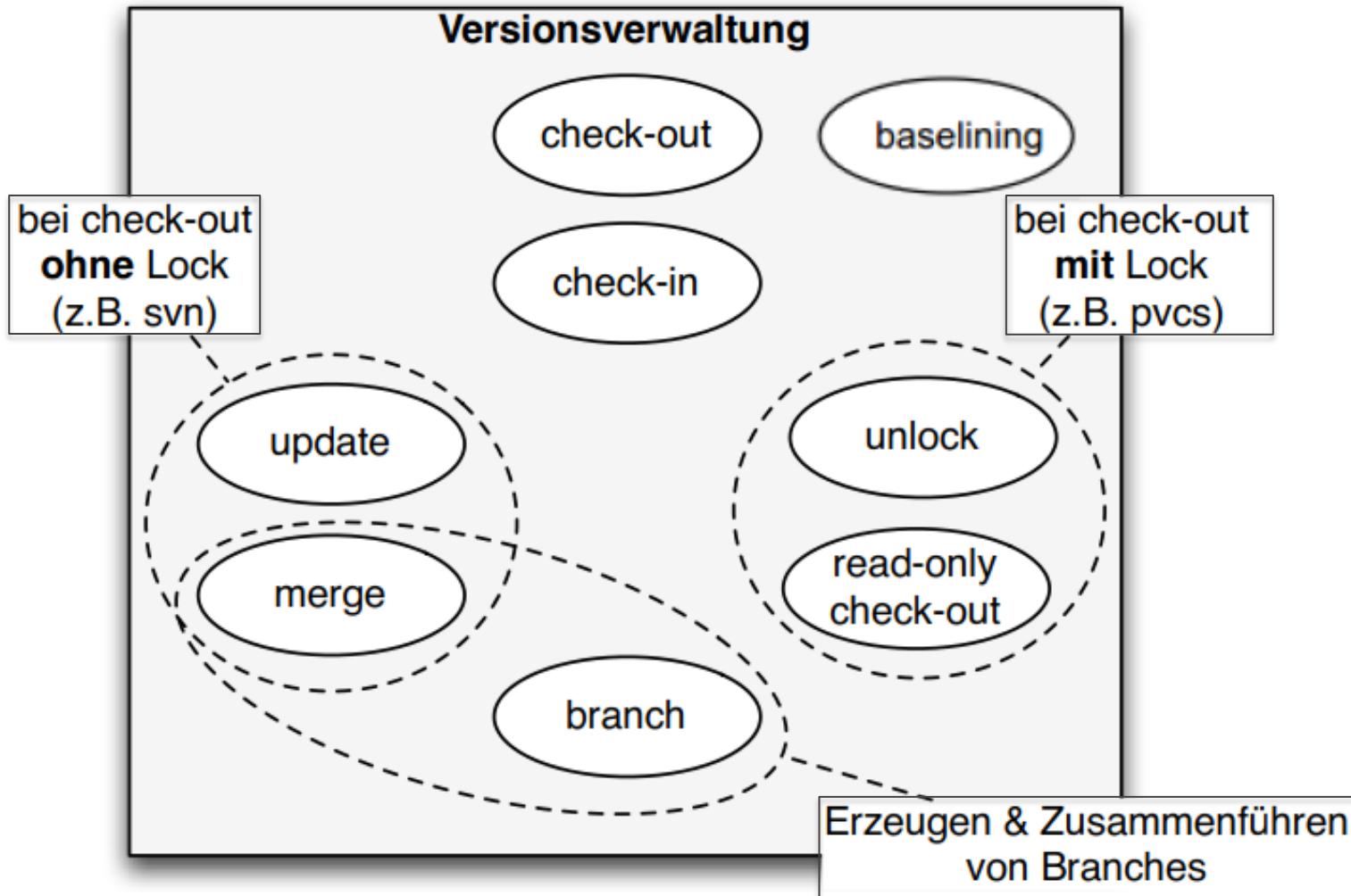
# VERSIONSVERWALTUNG – BSP. FÜR VERSIONSBAUM:



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim



# VERSIONSVERWALTUNG – GRUNDELGENDE USE CASES



# KONFIGURATIONSMANAGEMENT



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Was sollte versioniert werden?
  - Alle Dateien, die mit einem Projekt zu tun haben
    - Code, Tests, Anforderungen, Designdokumente, Sonstige Texte, Möglichst alle Entwicklungswerkzeuge (haben auch Versionen!)
- Zu bestimmten Zeiten müssen alle Dateistände zueinander eingefroren werden → Baselines
  - Zu definierten Meilensteinen (meist vor und nach dem Review, ...)
  - Zu bestimmten Meilensteinen müssen evtl. nur bestimmte Dateien zueinander versioniert werden (sog. Konfiguration)
- Zu freigegebenen Releases müssen evtl. noch Bugfixes, Patches, Service Packs nachgereicht werden
  - Branching erforderlich → Verzweigung der Entwicklung
  - Branches müssen später wieder in den Hauptzweig gemergt werden

# KONFIGURATIONSMANAGEMENT



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim



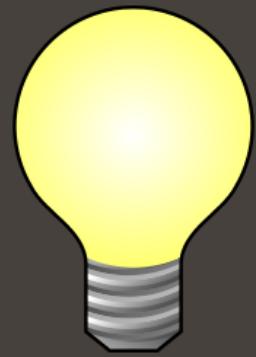
- Puh, das ist ja doch ganz schön kompliziert!
- Deshalb gibt es in größeren Projekten meist mindestens einen Verantwortlichen, der hier den Überblick behält
  - Prozess wird als Konfigurationsmanagement bezeichnet
- Typische Tätigkeiten des Konfigurationsmanagements:
  - Konfigurationsmanagementplan entwickeln
    - Welche Dateien müssen versioniert werden?
    - Wann müssen welche Baselines gezogen werden?
    - Wie verläuft ein Branching-Prozess?
    - ...
  - Baselining durchführen, Probleme in der Versionsverwaltung lösen
  - Bei Branches den Rückmerge sicherstellen
  - ...



# 05

# Fazit

Ziel:  
Was haben wir damit gewonnen?



# WAS HABEN WIR GELERNT?



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Vorgehensmodelle im Detail
  - Big Bang, iterativ, inkrementell
  - Schwergewichtig ↔ leichtgewichtig
  - Build-and-Fix-Cycle, Software Life Cycle, Spiralmodell
  - Wasserfall, V-Modell, RUP
  - Agile Methoden: eXtreme Programming, SCRUM
  - Lastenheft, Pflichtenheft
- Technische Infrastruktur zur Softwareentwicklung
  - Die wichtigsten Werkzeugarten
  - Im Fokus: Versionsverwaltung & Konfigurationsmanagement

# GESAMTÜBERBLICK ÜBER DAS IN SWT GELERNTE:



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

- Modellierungssprachen
  - 1. UML
    - Struktur- und Verhaltens-Diagramme
    - Einsatzzweck (Analyse — Entwurf — Implementierung)
  - 2. FMC-Blockdiagramme
- Programmieren im Großen
  - Analyse (Anwendungsfälle, Fachmodell, GUI-Entwurf)
  - Entwurf (Grobentwurf, Feinentwurf, Muster)
  - Test
- (Programmieren im Kleinen) (Programmiermethoden)  
←
  - Doku (Design by Contract, Refactoring, )
  - JUnit-Test)



# LITERATUR

- Vorgehensmodelle & Werkzeuge für die SW-Entwicklung
  - Kleuker: Grundkurs Software-Engineering mit UML.  
[<http://dx.doi.org/10.1007/978-3-8348-9843-2>]
  - Balzert: Lehrbuch der Softwaretechnik. 2008 [BF 000 103].
  - Hoffmann: Software-Qualität. 2008  
[<http://dx.doi.org/10.1007/978-3-540-76323-9>]
  - Van Vliet: Software Engineering, Wiley 2008.
- (R)UP
  - Kleuker: Grundkurs Software-Engineering mit UML  
[<http://dx.doi.org/10.1007/978-3-8348-9843-2>]
  - Zuser et al: Software-Engineering mit UML und dem Unified Process.  
[BF 500 92].
  - C. Larman: Applying UML and Patterns [30 BF 500 78].



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

AUF GEHT'S!!

SELBER MACHEN UND LERNEN!!



## Robustness Analysis

### Verwendeter Use Case:

Interessant ist, dass beim Durchspielen in diesem Jahr mir aufgefallen ist, dass der ganze 1. Schritt eigentlich sehr ungenau ist und, dass hier einiges passiert (siehe v.a. auch Sequenzdiagramm).

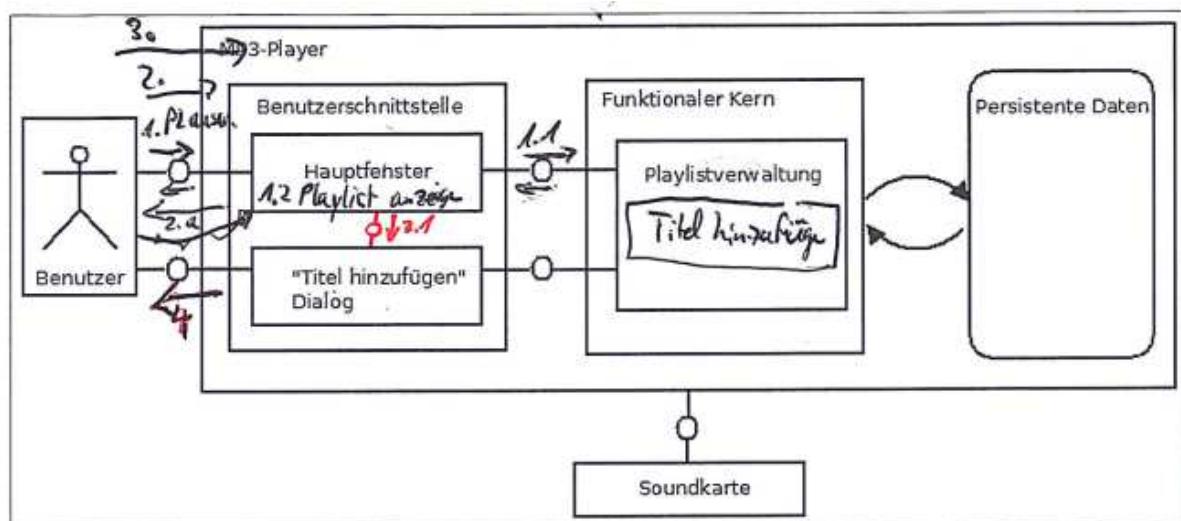
Man sieht also, dass die Robustness Analysis echt auch sehr gut ist, um Unstimmigkeiten bei Anforderungen aufzudecken:

Use Case "Titel hinzufügen" (Standardablauf)

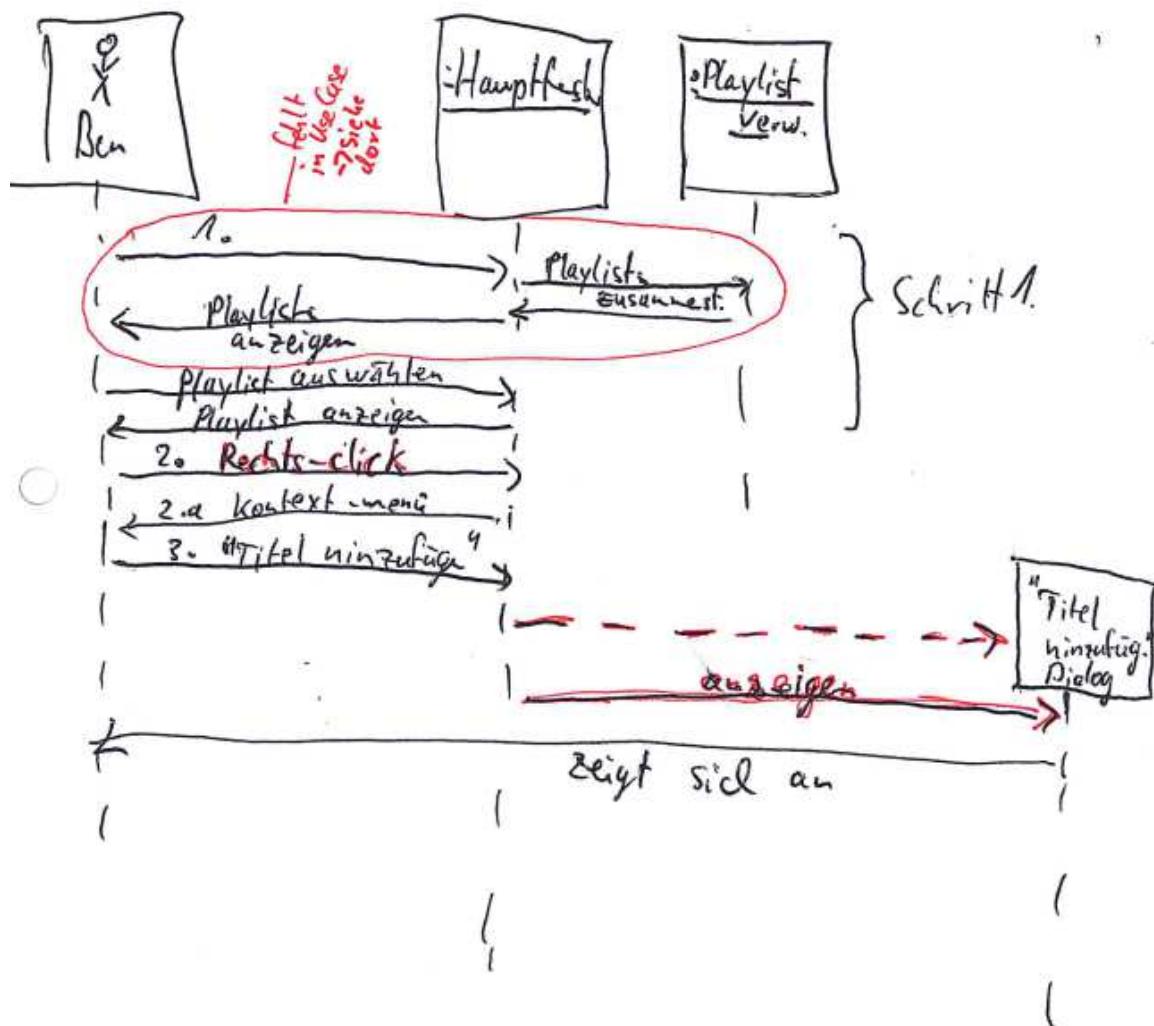
~~+ Fehlende Teil aus Sequenzdiagramm~~

1. Der Anwender wählt im Hauptfenster eine Playlist aus.
2. ~~User → rechts-click~~  
Das System zeigt das Kontext-Menü an.
3. Der Anwender klickt auf "Titel hinzufügen" im Kontext-Menü.
4. Das System zeigt den Dialog "Titel hinzufügen" an.
5. ...

Kommunikationsdiagramm basierend auf dem verwendeten Grobdesign (Architektur):



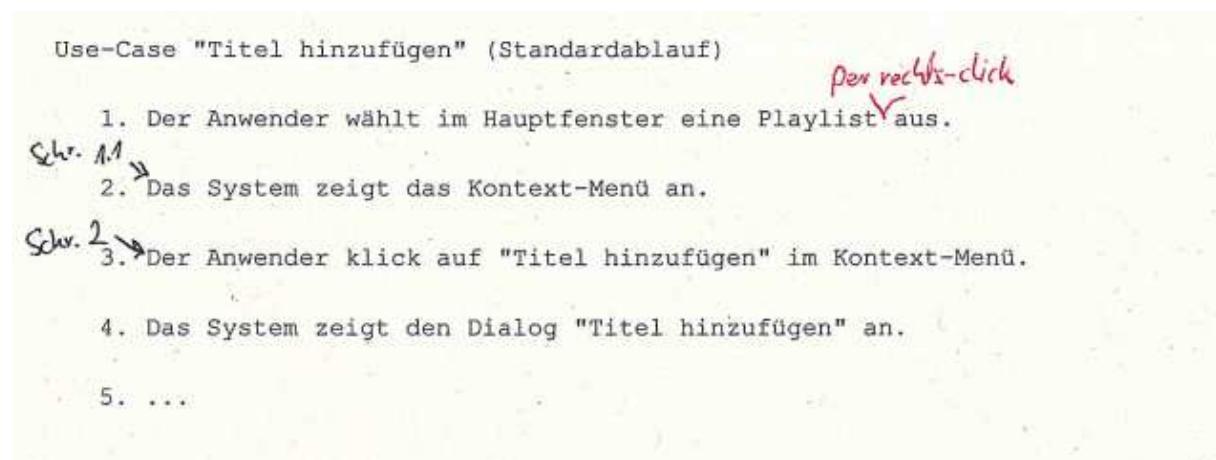
Durchgespieltes Szenario als Sequenzdiagramm - wie in der Vorlesung:



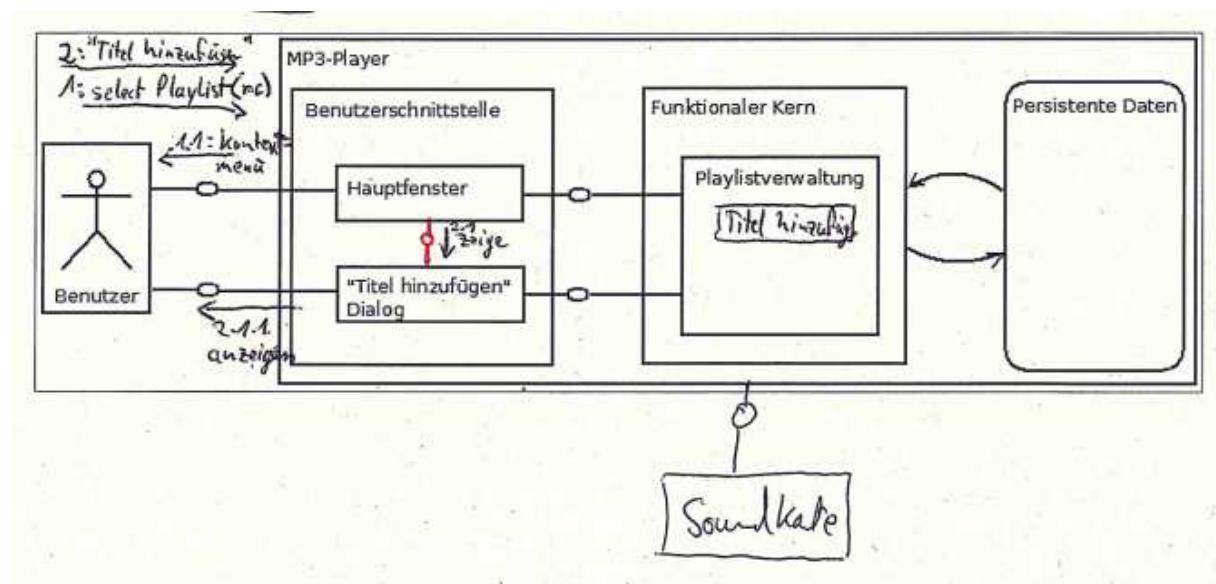
## Robustness Analysis WS 2015/2016

Die Robustness Analyse aus dem vorherigen Jahr. Diese ist noch ganz interessant wegen der Gedankengänge, die ich dann unten zu dem Auffinden der Unterschiede zw. Aufruf- und Signalsemantik geführt haben – lesen Sie das einmal aufmerksam durch.

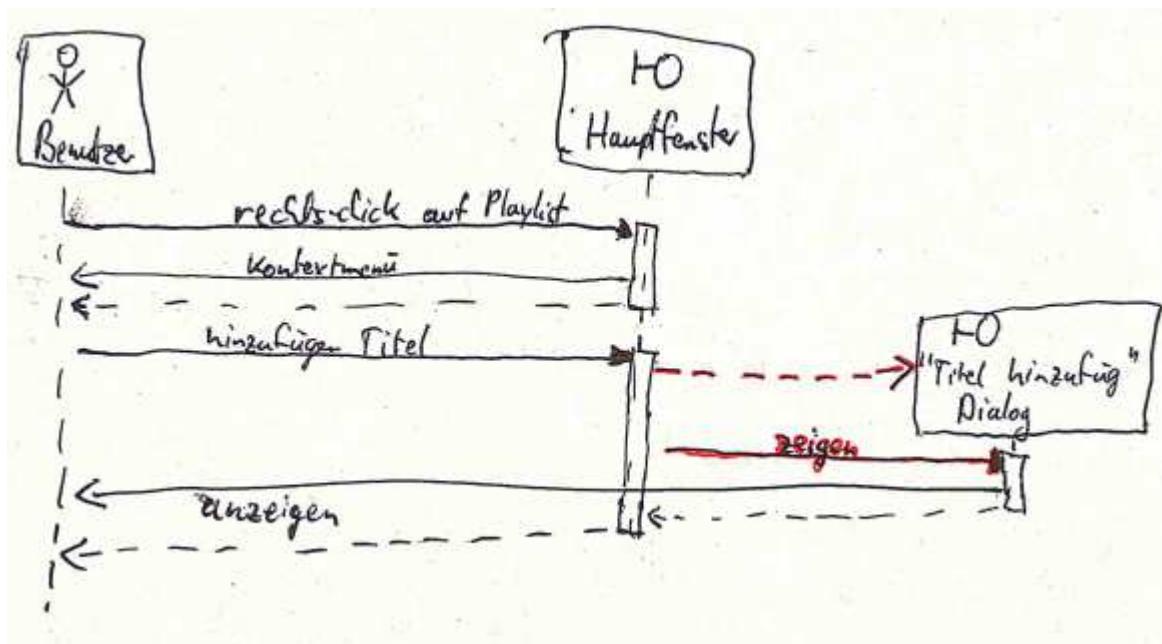
Verwendeter Use Case:



Kommunikationsdiagramm basierend auf dem verwendeten Grobdesign (Architektur):



Durchgespieltes Szenario als Sequenzdiagramm - wie in der Vorlesung:



**Sequenzdiagramm1:** Szenario in der Vorlesung unter der Verwendung der Aufruf-Semantik

### Hinweise zu dem Szenario:

Ich habe versucht das Szenario nach UML 2.0 Vorgaben abzubilden. Die geschlossenen durchgezogenen Pfeile stellen entsprechend synchrone Aufrufe dar und gehen vom Benutzer aus.

Da GUI-Elemente normalerweise am Bildschirm angezeigt werden, jedoch dann nicht klar ist wann der Benutzer wirklich reagiert, sollte hier, wenn man es schon korrekt macht, ein asynchroner Pfeil (offener durchgezogener Pfeil) kommen.

### Nachteile:

- Der UML 2.x Standard **lässt leider nicht mehr zu** wie in UML 1.x die Rückgabepfeile bei synchroner Kommunikation **wegzulassen**, weshalb jetzt viele gestrichelte Pfeile nötig sind.
- Was mir jedoch noch weniger gefällt, ist die Tatsache, dass man sich ja noch in einer relativ frühen Phase befindet. Hier stellen die Festlegungen auf synchrone und asynchrone Aufrufe und die Erzeugung „Titel hinzufügen“-Dialogs tatsächlich die Gefahr einer Vorfestlegung dar, die ja erst dann im Detailed Design vielleicht erfolgen sollte. Das ist nicht optimal und tatsächlich kenne ich so aus der Praxis auch eher, dass man in diesen Phasen nur versucht das echt gesicherte Wissen über die Kommunikation festzulegen.
- Umgekehrt, jetzt einfach so die Pfeile für asynchrone Kommunikation zu verwenden (wie das in der Praxis so passiert) nur weil das einfacher ist, ist auch irgendwie unbefriedigend, weil man dann leicht eine Vorfestlegung auf asynchrone Kommunikation macht, die man nicht möchte.

Viel besser wäre hier tatsächlich, wenn man einen Pfeil hätte, der den Typ der Nachricht unbestimmt ließe. Entsprechend habe ich auch nochmals nachrecherchiert und, wenn man [UMLglasklar; S. 430]

genau liest, so gibt es neben den synchronen und asynchronen Methodenaufrufen tatsächlich einen weiteren Nachrichtentyp.

Dieser Typ bedeutet einfach, dass ein Signal zwischen zwei Kommunikationspartnern ausgetauscht wird. Leider ist dieser Typ so definiert, dass einfach angegeben ist, dass Signale asynchron sind und damit auch mit dem asynchronen Pfeil definiert werden [UMLglasklar; S. 430].

D.h., wenn man asynchrone Pfeile in den frühen Phasen verwendet, kann man sich hier darauf berufen, dass man hier zunächst nur über Signale spricht, hier aber noch unbestimmt ist, ob die Kommunikation synchron oder asynchron erfolgt. Mir persönlich würde hier ein eigener Pfeil, der anzeigt, dass noch unbestimmt ist, ob die Kommunikation asynch oder synch ist besser gefallen, aber den gibt es leider nicht.

Entsprechend mit der Signal-Semantik würde mir eigentlich in dieser frühen Phase (Anforderungsphase und Architektur) doch Szenariodiagramm2 besser zusagen.



#### Nochmals zusammengefasst:

Beide Sequenzdiagramme sind richtig.

Jedoch wäre Sequenzdiagramm2 vielleicht doch in den frühen Phasen mit dieser Signal-Semantik zu bevorzugen – man sollte hier jedoch dann beachten, dass in diesem Fall noch nicht festgelegt ist wie genau dieses Signal dann im System umgesetzt ist und damit auch nicht festgelegt ist, ob dies asynchron oder synchron passiert!

Sequenzdiagramm1 wäre vielleicht doch eher dann ein Diagramm, das man dann in der Phase des Detaillierten Designs zeichnet, weil es dann auf die spezifischen technischen Details wie synchrone Aufrufe bei User-Klicks auf die GUI, jedoch asynchrone Anzeige der GUI beim Nutzer.