ModSoft

Modellbasierte Software-Entwicklung mit UML 2
im WS 2014/15

Organisatorisches

Prof. Dr. Joachim Fischer
Dr. Markus Scheidgen
Dipl.-Inf. Andreas Blunk

fischer@informatik.hu-berlin.de

ModSoft - Wann und Wo?

RUD 25, 3.113

Zielgruppe

MSc
Dipl

ModSoft

Vorlesung

Mo: 9.15 – 10.45

Mi: 13.15 – 14.45

Praktikum

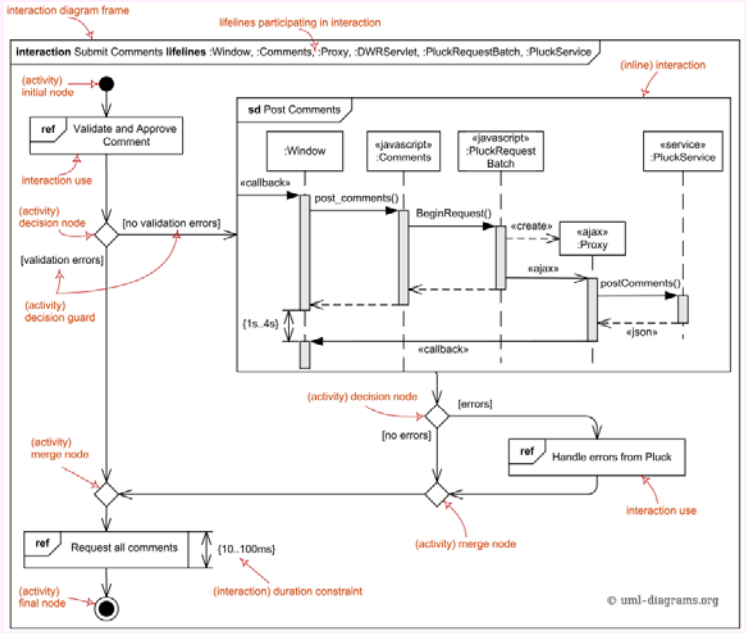
Mo: 11 - 13

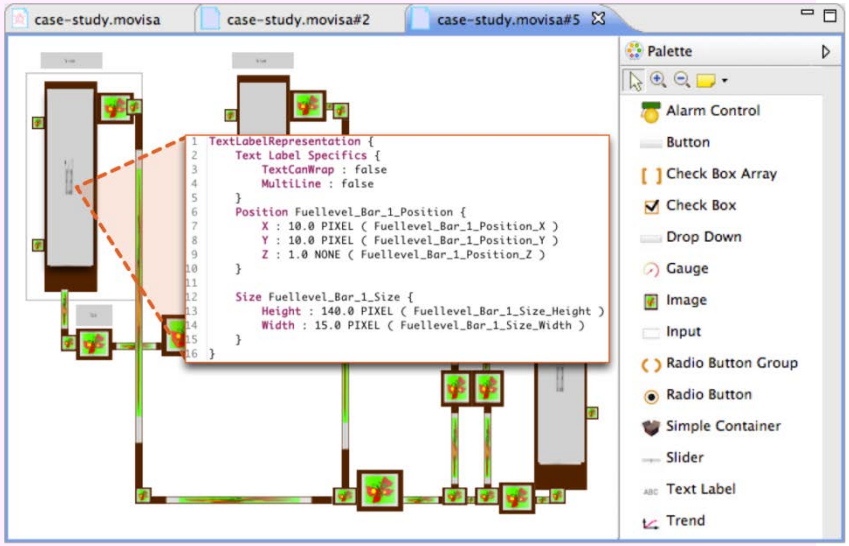
Start: 27.10.2014

- Homepage
  http://www.informatik.hu-berlin.de/sam/lehre

3. Modul
Modellbasierte Softwareentwicklung (ModSoft)

... basierend auf generisch verwendbaren Modellierungs-sprachen, wie der *Unified Modeling Language* (UML)





Zum Aufbau der Vorlesung

... basierend auf generisch verwendbaren Modellierungs-sprachen, wie der *Unified Modeling Language* (**UML**)

- Einführung in die Modellierung mit der UML und der Definition von UML basierend auf der *Meta-Object Facility* (MOF)
- Strukturmodellierung: Klassen-, Objekt-, Komponentendiagramme
- Verhaltensmodellierung: Zustandsautomaten, *Message Sequence Charts* (MSC), Kollaborations- u.Aktivitätsdiagramme
- Erweiterung der UML mit UML-Profilen

... basierend auf speziell für eine Problemdomäne entwickelten Modellierungsprachen: *Domain Specific Languages* (**DSL**)

- Einführung in Sprachen, Sprachaspekte u. Modellierung von Sprachen
- Eclipse-Entwicklung als Basis zur Sprachmodellierung u. Sprachwerkzeugentwicklung
- Sprachstruktur: Meta-Modellierung mit *EMF* u. *OCL*
- Textuelle u. graphische Notationen mit *xText* u. *GMF*
- Sprachsemantik: Codegenerierung mit *xTend* u. Modelltransformationen mit *ATL*

*Zum Aufbau der anteiligen **UML**-Vorlesung*

1. *Einführung*
2. *Strukturmodellierung: Klasse, Rolle, Use-Case*
3. *Object Constraint Language (OCL)*
4. *UML-2.5 Spracharchitektur*
5. *Strukturmodellierung*
6. *Verhaltensmodellierung*
7. *Spracherweiterung*

Praktischer Teil

UML-Werkzeug: Magic Draw

als kostenfrei erhältliches UML-Case-Werkzeug

- erlaubt die Erstellung von UML-Diagrammen
- JAVA-basiert, damit plattformunabhängig
- Modelle können
  - erstellt,
  - gespeichert,
  - gedruckt

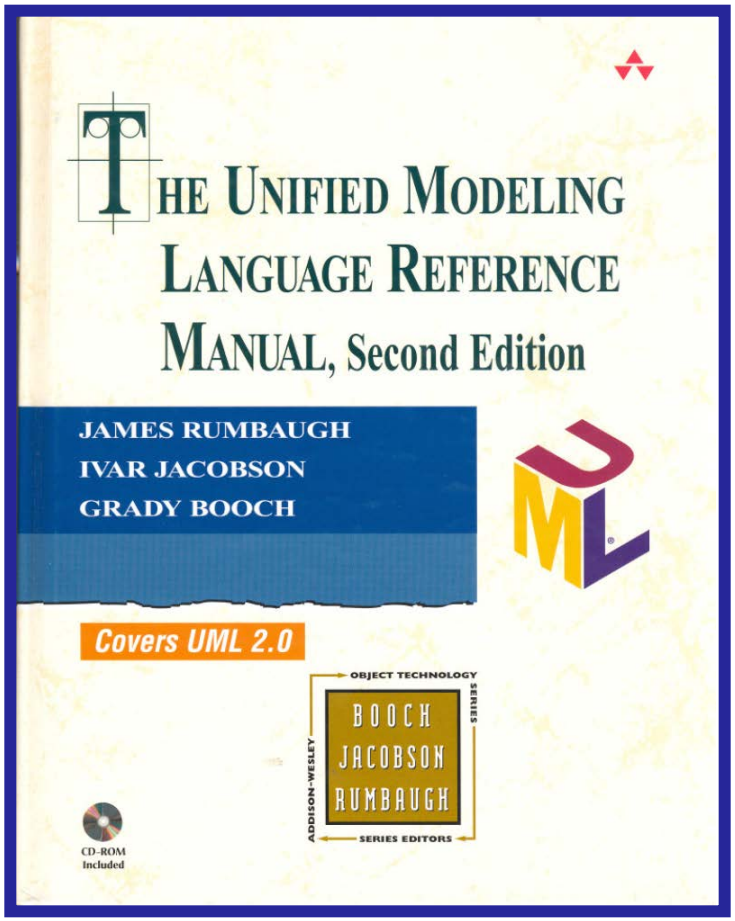
  werden

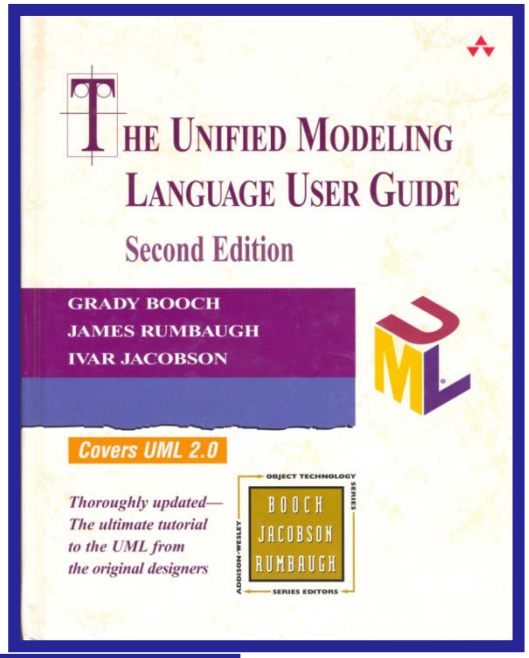
  ➔ damit einfacher und intuitiver Zugang zu UML möglich
- aus Klassendiagrammen kann man JAVA-Code erzeugen

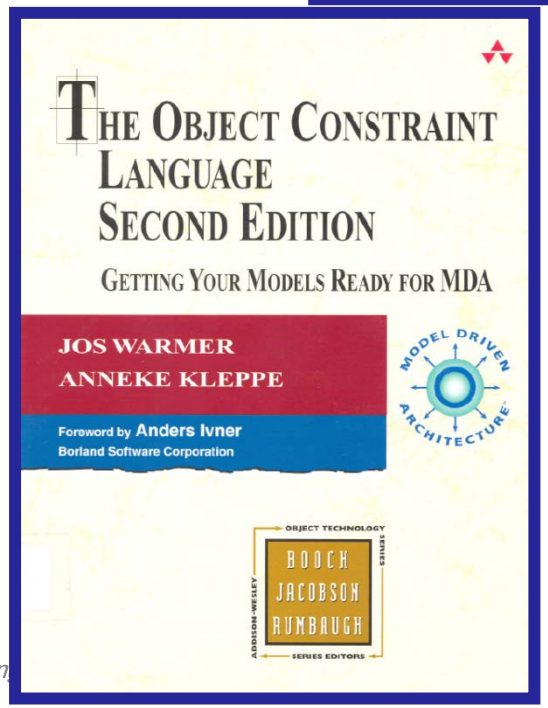
  offen: Code-Generierung aus Verhaltensbeschreibungen

Produktbeschreibung und Download:
(im Praktikum)

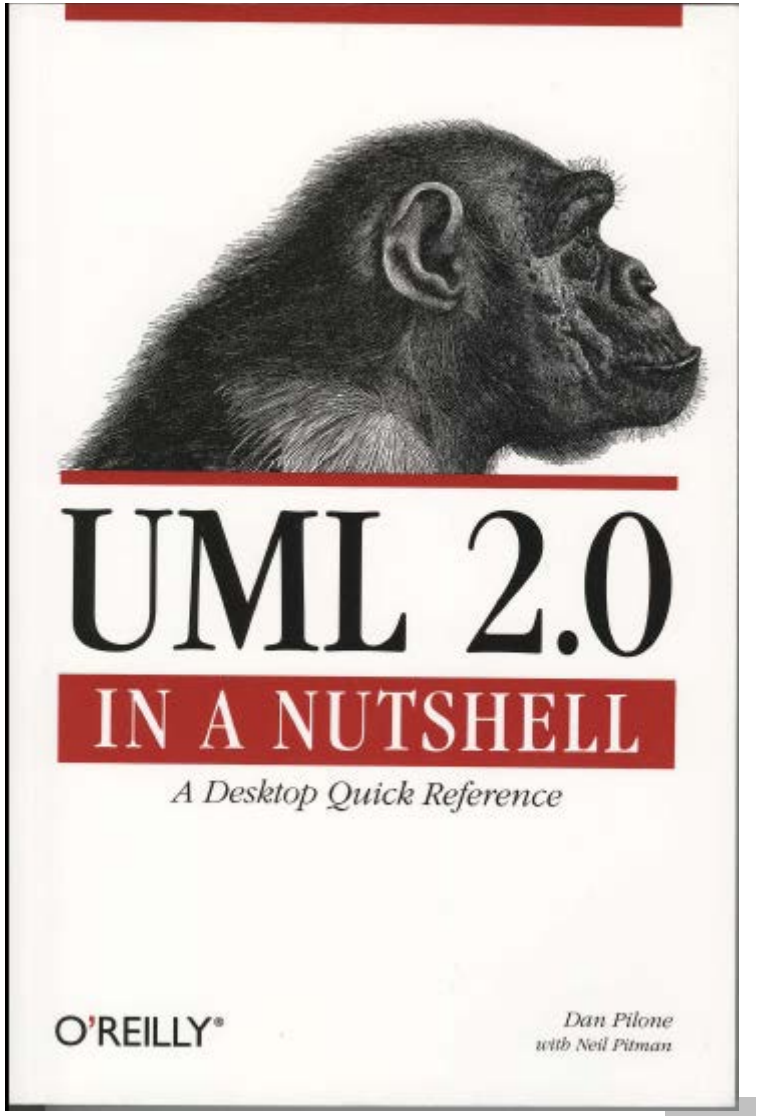
Unsere Quellen: Die Originale



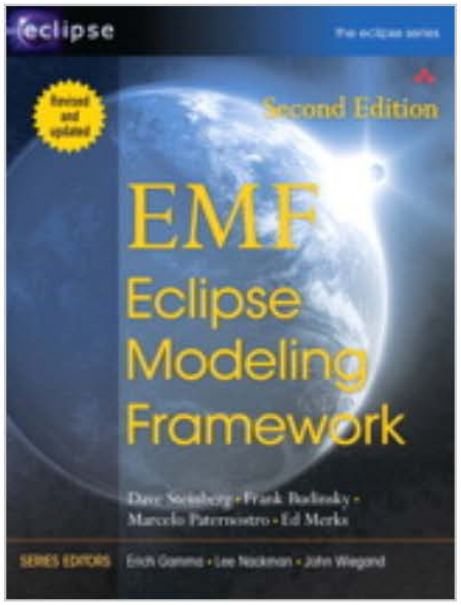




Unsere Quellen



Unsere Quellen



EMF: Eclipse Modeling Framework, 2nd Edition

By Dave Steinberg, Frank Budinsky, Marcelo Paternostro, Ed Merks

Published Dec 16, 2008 by Addison-Wesley Professional. Part of the Eclipse Series series.

Best Value Purchase

IKV++ Technologies AG

StyleCheck - Automated Guideline Compliance for Simulink®, Stateflow® and TargetLink® models

Model Based Design and auto-code generation is widely adopted in the automotive industry for embedded software development. Like coding standards, several modelling standards and modelling guidelines have been established by several leading industrial consortiums, like the MAAB and the MISRA. These guidelines are specifically aimed towards managing model complexity, help bring structure to model design, and ensuring robust, error-free and more compliant auto-code generation. Moreover, such standard rule compliance helps design teams comply with industry standards such as ISO 26262, IEC 61508 and DO 178B.

Zusätzlich über Homepage

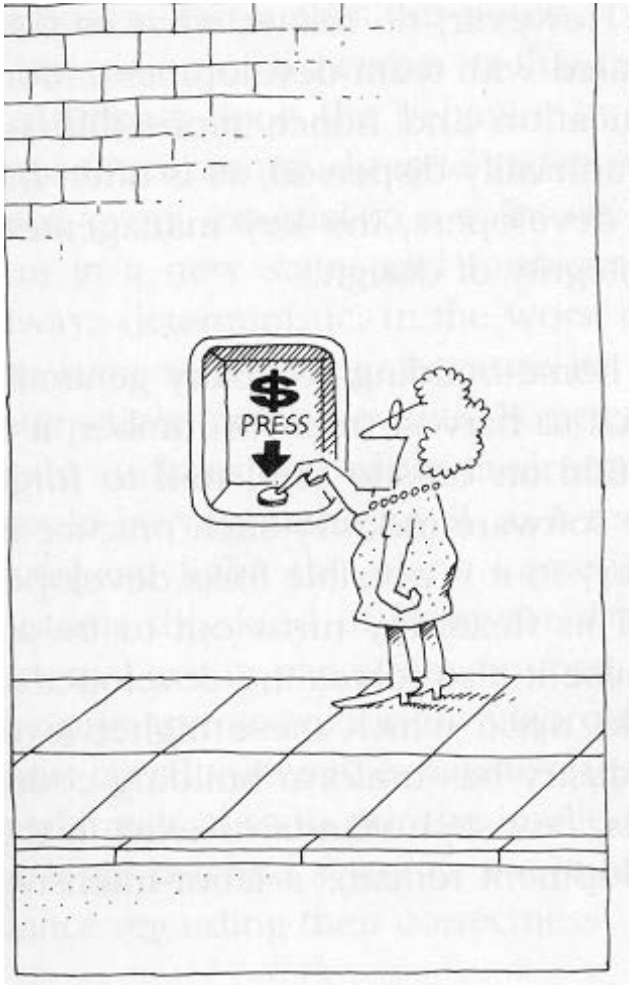
- Vorlesungsskripte

- OMG-Standards
  - MOF, Version 2.4 (70 Seiten)
  - UML, Version 2.5 (831 Seiten)
  - OCL, Version 2.4 (262 Seiten)

Teil I: Einführung

1. MDD als Trend in der Software-Entwicklung
2. UML, ein erster Blick
3. Grundlagen der Modellierung
4. Paradigmen der UML-Modellierung
5. Historie von UML
6. Modellierungselemente von UML im Überblick
7. Diagrammrepräsentationen in UML
8. Struktur des UML-Standards

Ewiges Ziel: Vereinfachung der Softwareentwicklung

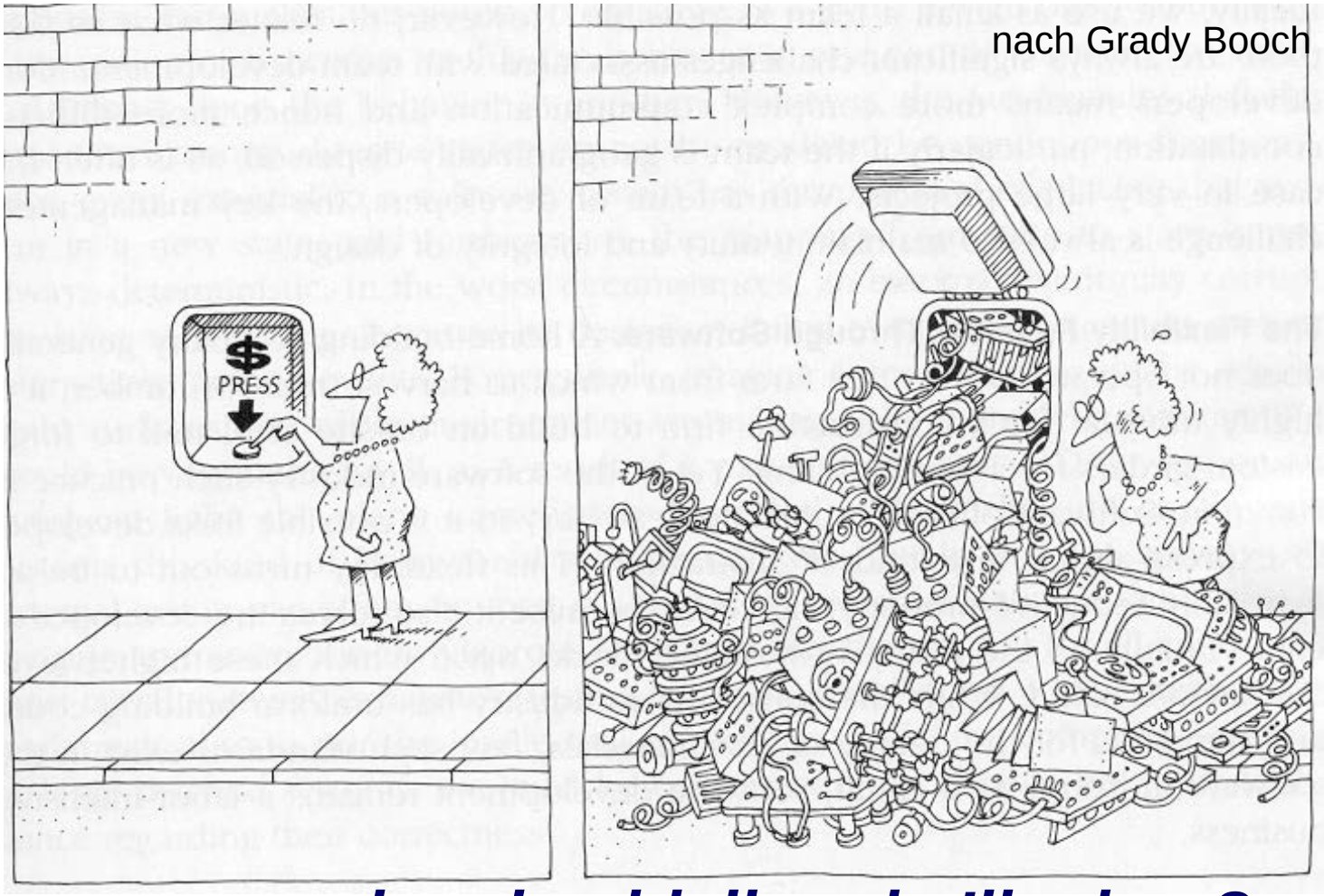


Herausbildung unterschiedlicher Paradigmen

- Strukturierte Programmierung
- …
- Objektkomposition
- Modelltransformationsparadigma
- Komponentenparadigma

Aktuell: Model-Driven Development

Vereinfachung der Softwareentwicklung



– oder eine bleibende Illusion ?

Typische Phasen der Systementwicklung

Was soll entwickelt werden?

Analyse

Studie Problemdomäne
beobachtbares Verhalten
funktionale/nichtfunktionale Aspekte

Wie soll das System funktionieren?

Design

Hinzufügen von Details
Algorithmen
Task- und Datenmanagement

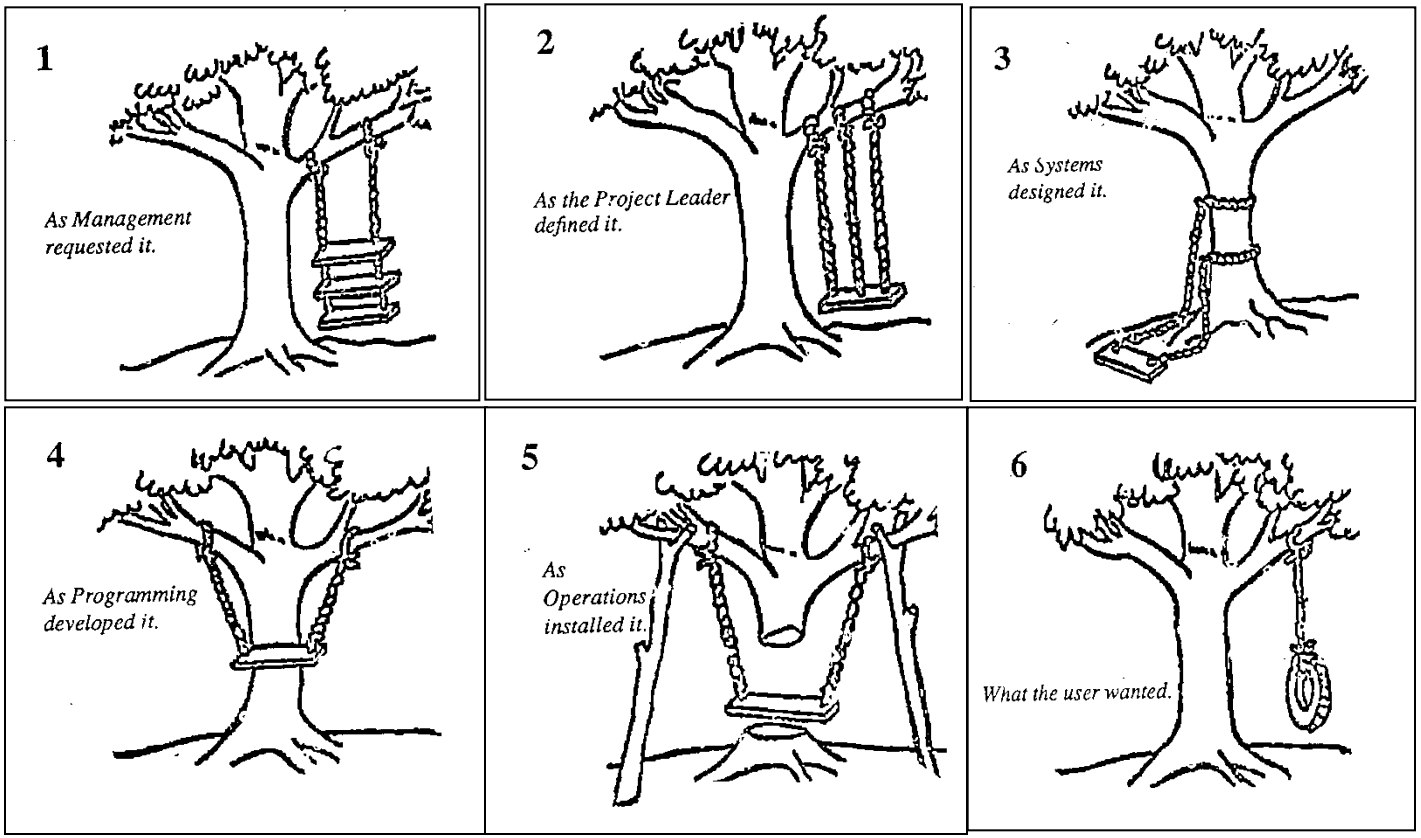
Womit soll dies technisch umgesetzt werden?

Implementierung

Programmierung
Anpassung an Betriebssystem etc.

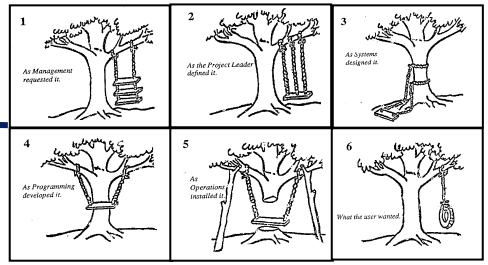
Probleme der traditionellen SW-Entwicklung

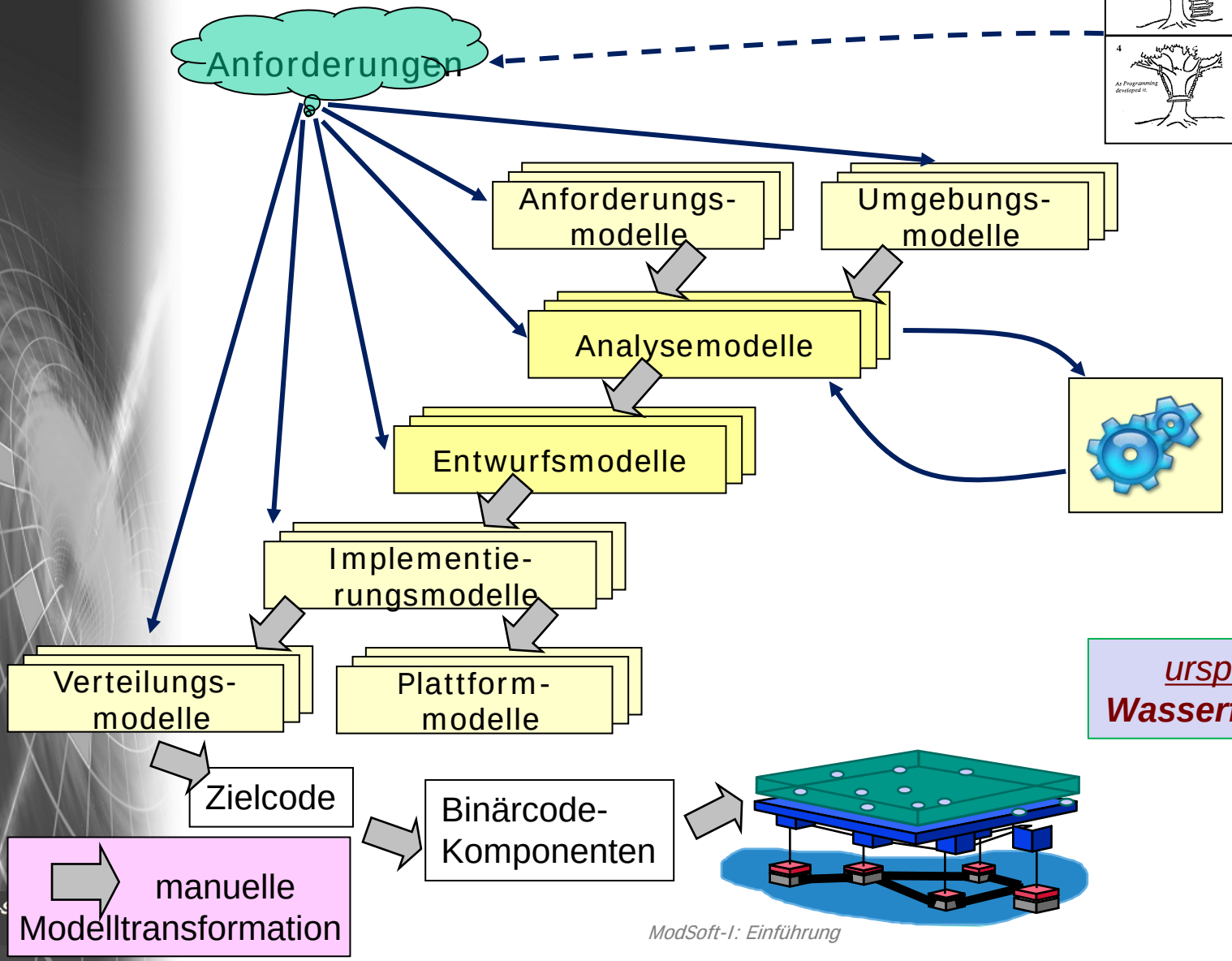
Wozu Modellierung ?



Alte Idee: Modellbasierte Software-Entwicklung

verteilter Systeme (vereinfacht)



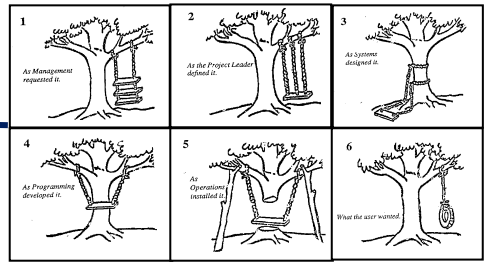


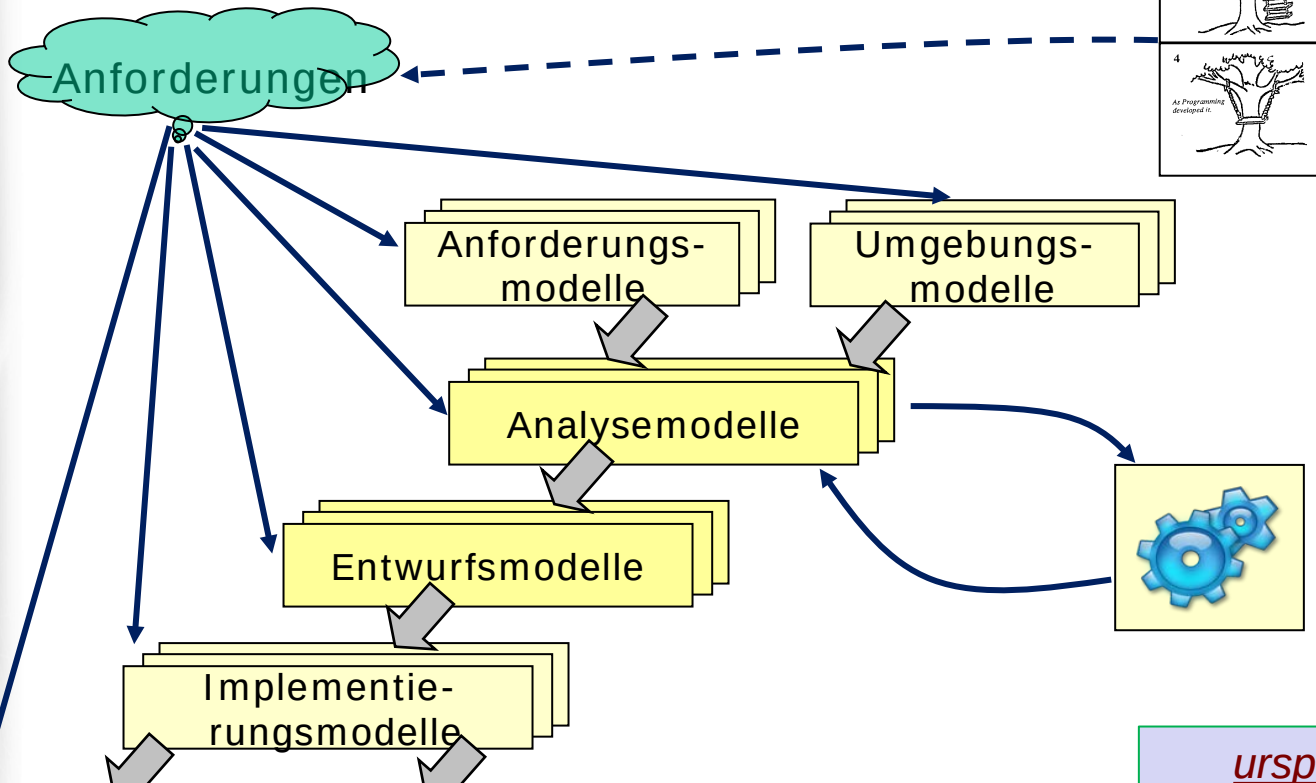
- *Informaler Dokumenten-review*
- *Test,*
- *ModelChecker*
- *Simulator*

ursprünglich:
Wasserfallmethode

Neuer Ansatz: Modellgetriebene Software-Entwicklung

verteilter Systeme (vereinfacht)

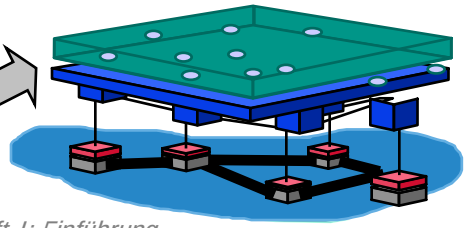




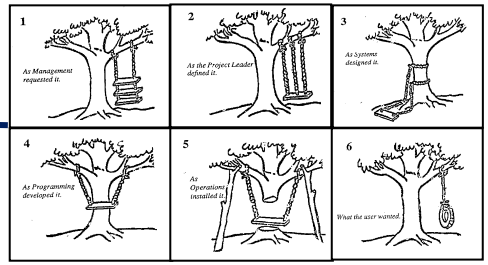
- *Informaler Dokumenten-review*
- *Test,*
- *ModelChecker*
- *Simulator*

ursprünglich:
Wasserfallmethode

später:
Spiralmodelle



Konzeptionelle Grundlage für automatische Modelltransformation



Anforderungen

unterschiedliche Modellierungssprachen mit Syntax und Semantik

Anforderungs-modelle

Umgebungs-modelle

Analysemodelle

Entwurfsmodelle

Implementie-rungsmodelle

Verteilungs-modelle

Plattform-modelle

Zielcode

Binärcode-Komponenten

UML-Metamodell in MOF definiert UML

MOF= EMOF←CMOF

MOF MetaObjectFacility als Infrastruktur

einheitliche, universelle Konzeptbasis für unterschiedliche Modellklassen mit freier konkreter Syntax

Meta-Modelle, formuliert mit MOF-Konzepten
zur Definition der abstrakten Syntax
von Sprache-1, … Sprache-n
dynamische Semantik ?

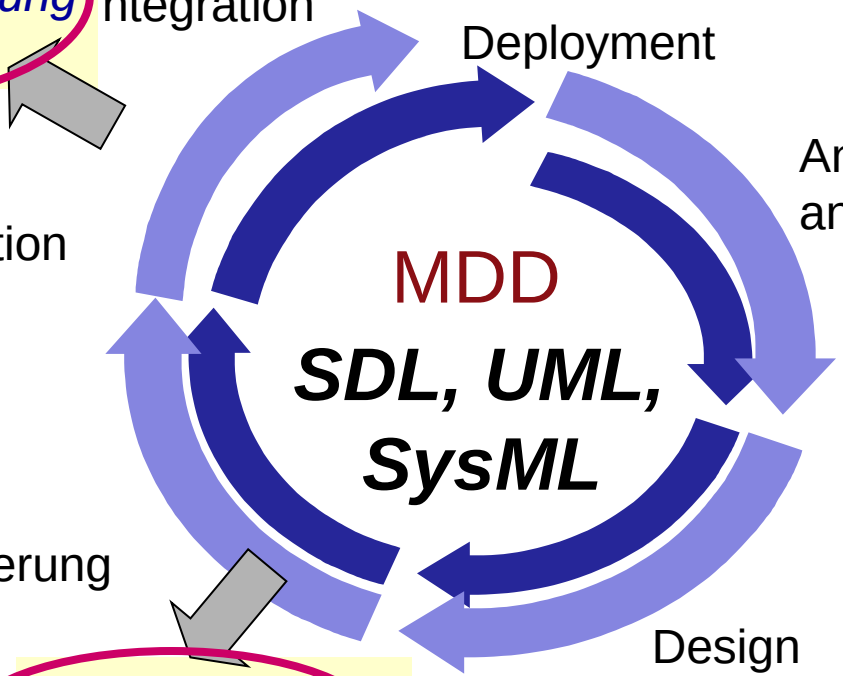
Modellgetriebene Software-Entwicklung

spiralförmig, inkrementell & iterativ

MDD:= Model Driven Development
- SW-Entwicklung ist modellzentriert (Modelle begleiten ges. SW-Lebenszyklus)
- automatische Transformationen für Modellübergänge
- spezifische Analysen (Checker, Simulatoren, …)
- partielle oder komplette Codegenerierung

Test funktionaler und nicht-funktionaler Rückkkopplungen

Wechselwirkung mit der Umgebung mit Simulation



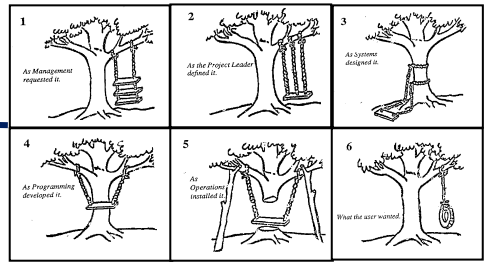
Echtzeit, Leistungsprognose Ausführung durch Simulation

Test nicht-funktionaler Eigenschaften

Ausführung durch Simulation

Test funktionaler Eigenschaften

Technische Grundlage für automatische Modelltransformation



Anforderungen

unterschiedliche Modellierungssprachen mit Syntax und Semantik

Anforderungs-modelle

Umgebungs-modelle

Analysemodelle

Entwurfsmodelle

Implementie-rungsmodelle

Verteilungs-modelle

Plattform-modelle

Zielcode

Binärcode-Komponenten

emf
ECLIPSE MODELING FRAMEWORK

EMF
Eclipse Modeling Framework

einheitliche, universelle Konzeptbasis für unterschiedliche Modellklassen mit freier konkreter Syntax

Ecore= EMOF

Meta-Modelle, formuliert mit EMF-Konzepten
zur Definition der abstrakten Syntax
von Sprache-1, … Sprache-n
dynamische Semantik ?

Eclipse Modeling Framework (EMF)

The EMF project is a modeling framework and code generation facility for building tools and other applications based on a structured data model. From a model specification described in XMI, EMF provides tools and runtime support to produce a set of Java classes for the model, along with a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor.

EMF (core) is a common standard for data models, many technologies and frameworks are based on. This includes **server solutions**, **persistence frameworks**, **UI frameworks** and **support for transformations**. Please have a look at the **modeling project for an overview of EMF technologies**.

EMF (Core)

EMF consists of three fundamental pieces:

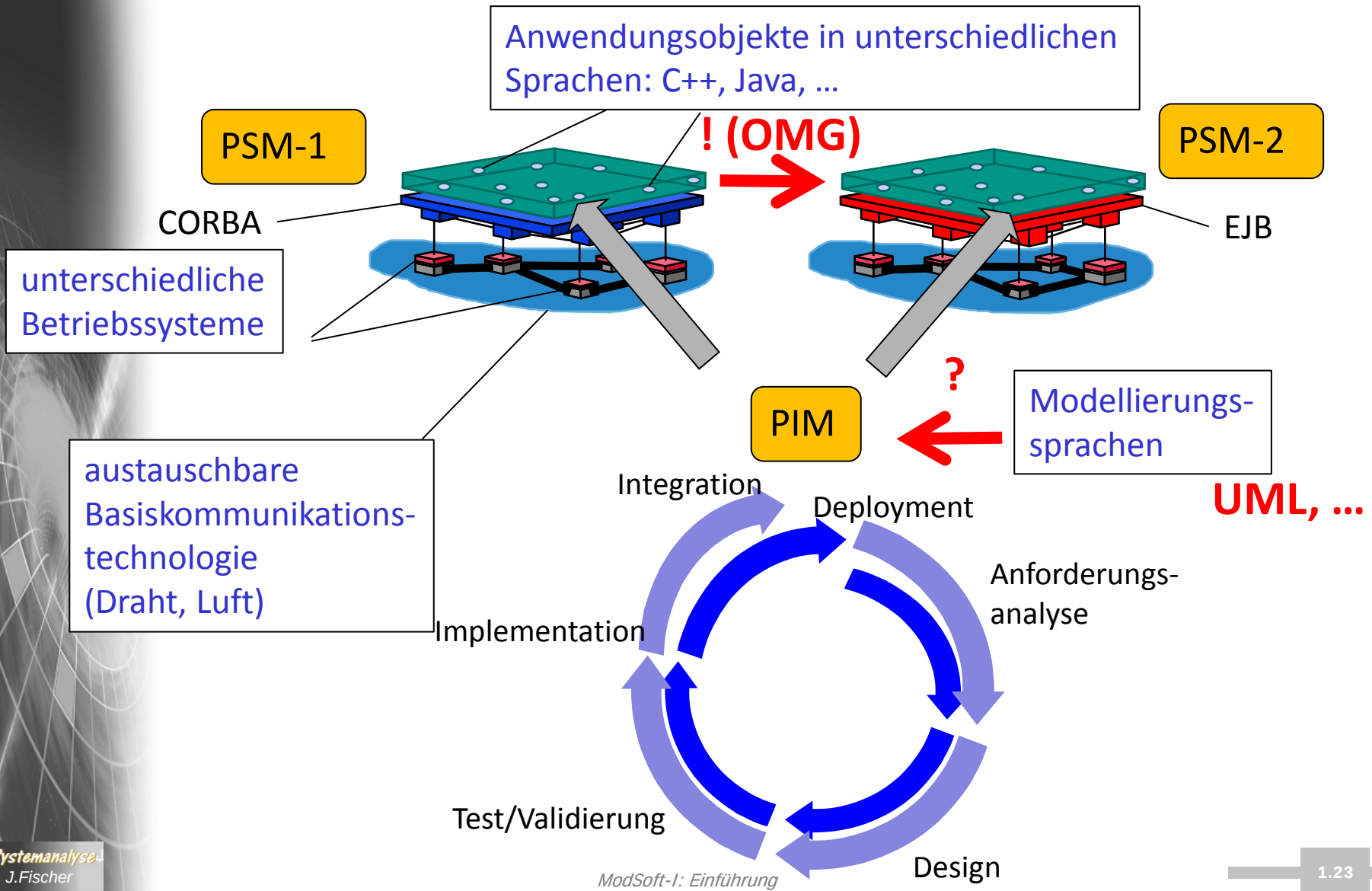
- **EMF** - The core EMF framework includes a **meta model (Ecore)** for describing models and runtime support for the models including change notification, persistence support with default XMI serialization, and a very efficient reflective API for manipulating EMF objects generically.
- **EMF.Edit -** The EMF.Edit framework includes generic reusable classes for building editors for EMF models. It provides
  - Content and label provider classes, property source support, and other convenience classes that allow EMF models to be displayed using standard desktop (JFace) viewers and property sheets.
  - A command framework, including a set of generic command implementation classes for building editors that support fully automatic undo and redo.
- **EMF.Codegen** - The EMF code generation facility is capable of generating everything needed to build a complete editor for an EMF model. It includes a GUI from which generation options can be specified, and generators can be invoked. The generation facility leverages the JDT (Java Development Tooling) component of Eclipse.

Ecore= EMOF

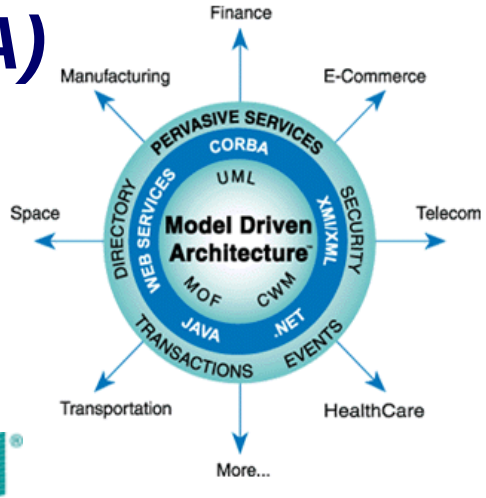
Quelle:
http://www.eclipse.org/modeling/emf/

Modellgetriebene Software-Entwicklung

spiralförmig, inkrementell & iterativ



Model-Driven Architecture (MDA)



- … fasst die gesammelten Erkenntnisse über
  - SW-Modelle, Modellierung und Transformation,
  - angereichert mit einer Reihe weiterer Standards

  zu einer offiziell anerkannten **Spezifikation** zur modellgetriebenen Softwareentwicklung zusammen

OMG
OBJECT MANAGEMENT GROUP

- **Ziel:**
  Abbildung des gesamter Softwareentwicklungsprozesses
  - von der Fachdomäne des späteren Anwenders,
  - über die Anforderungsanalyse
  - bis hin zur Implementierung des Zielsystems mit allen seinen Schichten)

  in Modellen,
  so dass das System selbst über Modelltransformation, erzeugt werden kann

- Sind alle Transformatoren geschrieben, so erreicht man auf diesem Weg eine hohe Wiederverwendbarkeit und Wartbarkeit

- Darüber hinaus gilt die MDA als ein möglicher Schlüssel zur anforderungsgetriebenen Softwareentwicklung, da die technischen Aspekte weitestgehend vollständig von den inhaltlichen (semantischen) Aspekten getrennt werden.

Spezielle Modelle der MDA

- Plattform Independent Models (PIM):
  die Modellierung der Fachdomäne (also der Zielwelt) ist **vollständig plattformunabhängig** zu gestalten,
  es sind also ausschließlich rein fachliche Aspekte zu betrachten und zu modellieren.

- **Plattform Description Models (PDMs)** sind Modelle,
  die die Zielplattform des Systems beschreiben.
  Über die Kombination von einem PIM, also einer formalen semantischen Beschreibung der Zusammenhänge und Abläufe mit einem PDM kann letztendlich über Modelltransformation das Zielsystem
  (welches im Sinne der MDA auch wieder nur ein Modell ist) generiert werden

- **Platform Specific Model (PSM)**
  ist das Ergebnis der Modelltransformation

Model-Driven Architecture (Leitsätze)

- **Formalisierung** ist ein wichtiger Baustein für ein erfolgreiches Qualitätsmanagement in Softwareprojekten.
  Speziell in den Bereichen der Anforderungs- und Systemanalyse besteht häufig noch ein hohes Optimierungspotential.

- Ein möglicher Weg, um den Formalisierungsgrad von Projektinformationen zu erhöhen, ist die Verwendung von **formal eindeutigen Modellen**.

  Für den erfolgreichen Einsatz von Modellen ist es jedoch unabdingbar, die **Syntax und die Semantik der Modelle (über Metamodelle)** exakt festzulegen.
  Ist dies einmal geschehen, ergibt sich meist eine deutliche Steigerung der Qualität wie auch der Effizienz in der Projektarbeit.

- Über den gezielten **Einsatz von Metamodellen** in der Softwareentwicklung können große Teile der Prozessaktivitäten automatisiert werden.

  Dennoch muss berücksichtigt werden, dass die Formalisierung eines Softwareentwicklungsprozesses **nicht in einem Schritt** erfolgen kann.
  Sie sollte vielmehr als ein **iterativer Prozess** verstanden werden, in dem die entstehenden Metamodelle von Projekt zu Projekt immer weiter verfeinert werden müssen.

Einsatz Modellierung in der SW-Entwicklung(1)

- Modellierung in allen Wissenschaftsdisziplinen das zentrale Paradigma zum Verständnis komplexer realer oder hypothetischer Systeme

- In der SW-Entwicklung lange Zeit nicht hoffähig :
  Alternative: von der Idee direkt zum gut dokumentierten Quellcode
  aber: Komplexität der Systeme bereiten erhebliche praktische Probleme

- **Achtung**: MDD verlangt nicht nur Konzepte, sondern integrierte Werkzeugunterstützung

  ➔MOF- , UML, EMF-Technologien sind stark im Kommen – weisen aber immer noch Lücken auf

Teil I: Einführung

1. MDD als Trend in der Software-Entwicklung
2. UML, ein erster Blick
3. Grundlagen der Modellierung (gerafft)
4. Paradigmen der UML-Modellierung
5. Historie von UML
6. Modellierungselemente von UML im Überblick
7. Diagrammrepräsentationen in UML
8. Struktur des UML-Standards

UML-Klassendiagramm

Person {reines UML}

boss: Person {0..1}
employer: Company {0..1}

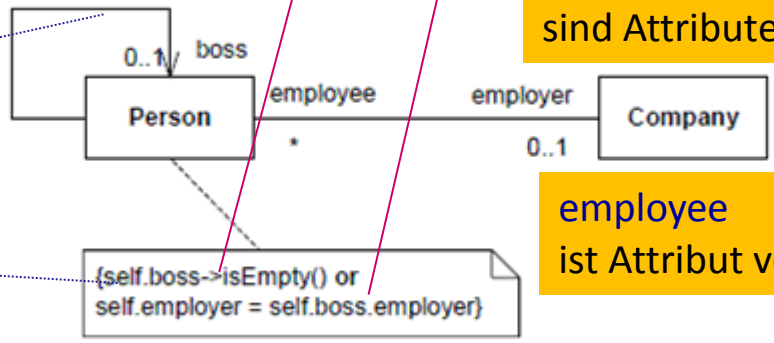
Person {UML+OCL-Interpretation}

boss: collection<Person> {0..1}
employer: Company {0..1}

- UML-Klassen sind konkretisierte Classifier
  UML-Assoziationen sind konkrete Classifier
  …
  UML-Kommentare sind konkretisierte Classifier

- Diagramm zeigt
  - UML-Anwendungsebene
  - zwei Klassen (von denen unterschiedliche Objekt-Mengen näher charakterisiert werden)
  - zwei namenlose Assoziationen/Beziehungen (an den Enden mit Objektmengen, die so in eine Zuordnung gebracht werden)
    eine Assoziation ist reflexiv
  - Objektmengen werden per Namen und Kardinalität näher beschrieben
  - ein Kommentar zur Klasse Person, genauer zu Objekten der Klasse Person
  - der Kommentar enthält {…}, der Inhalt wird als formale OCL-Regel interpretiert

Navigierbarkeit einer Assoziation



boss, employer
sind Attribute von Person

gilt als Invariante

employee
ist Attribut von Company

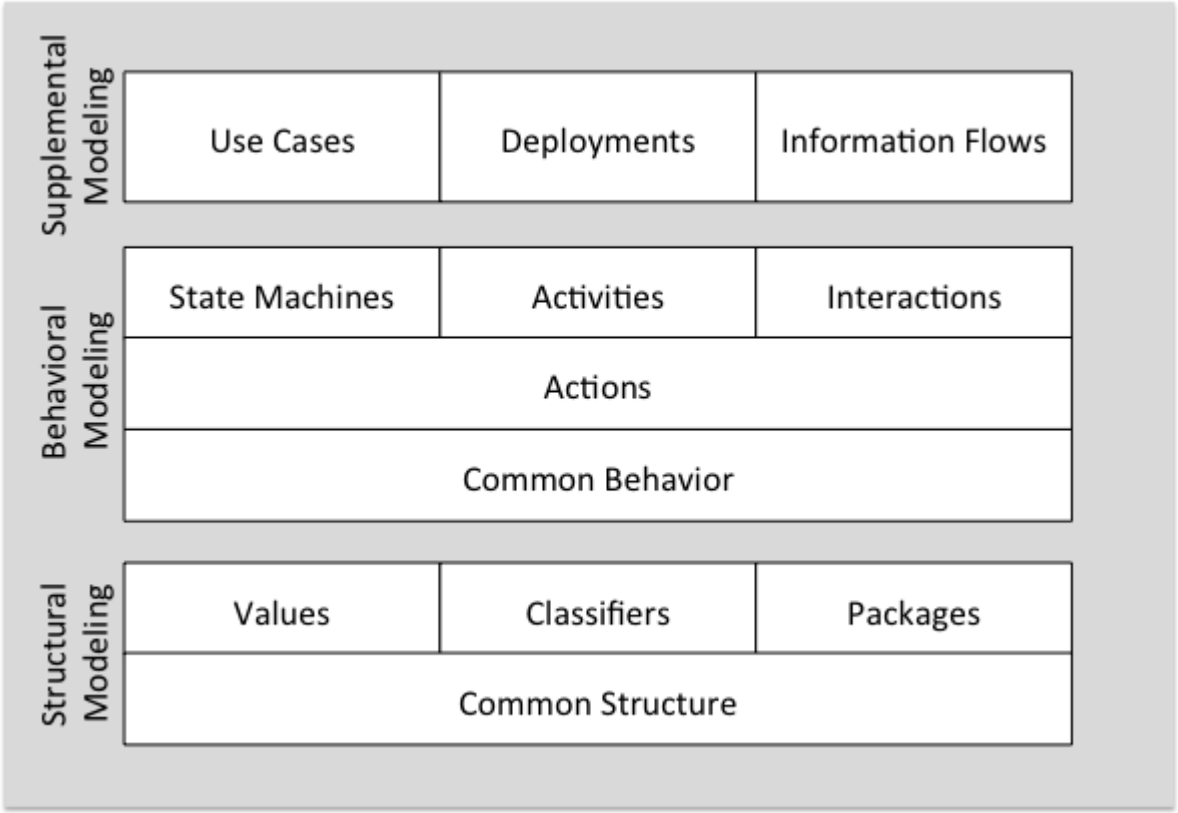
Definition von UML

- … erfolgt als Metamodell,
  welche Konzepte zur Verfügung stehen
  legt MOF fest.

- MOF hat keine konkrete Syntax, man benutzt UML
  (d.h. Kernkonzepte von UML)

- Analogien:
  - Compiler höherer Sprachen wurden oft in den Sprachen selbst implementiert (Bootstrapping)
  - der Duden zur Regelung der deutschen Sprache ist in Deutsch geschrieben

Konzept-Welten von UML



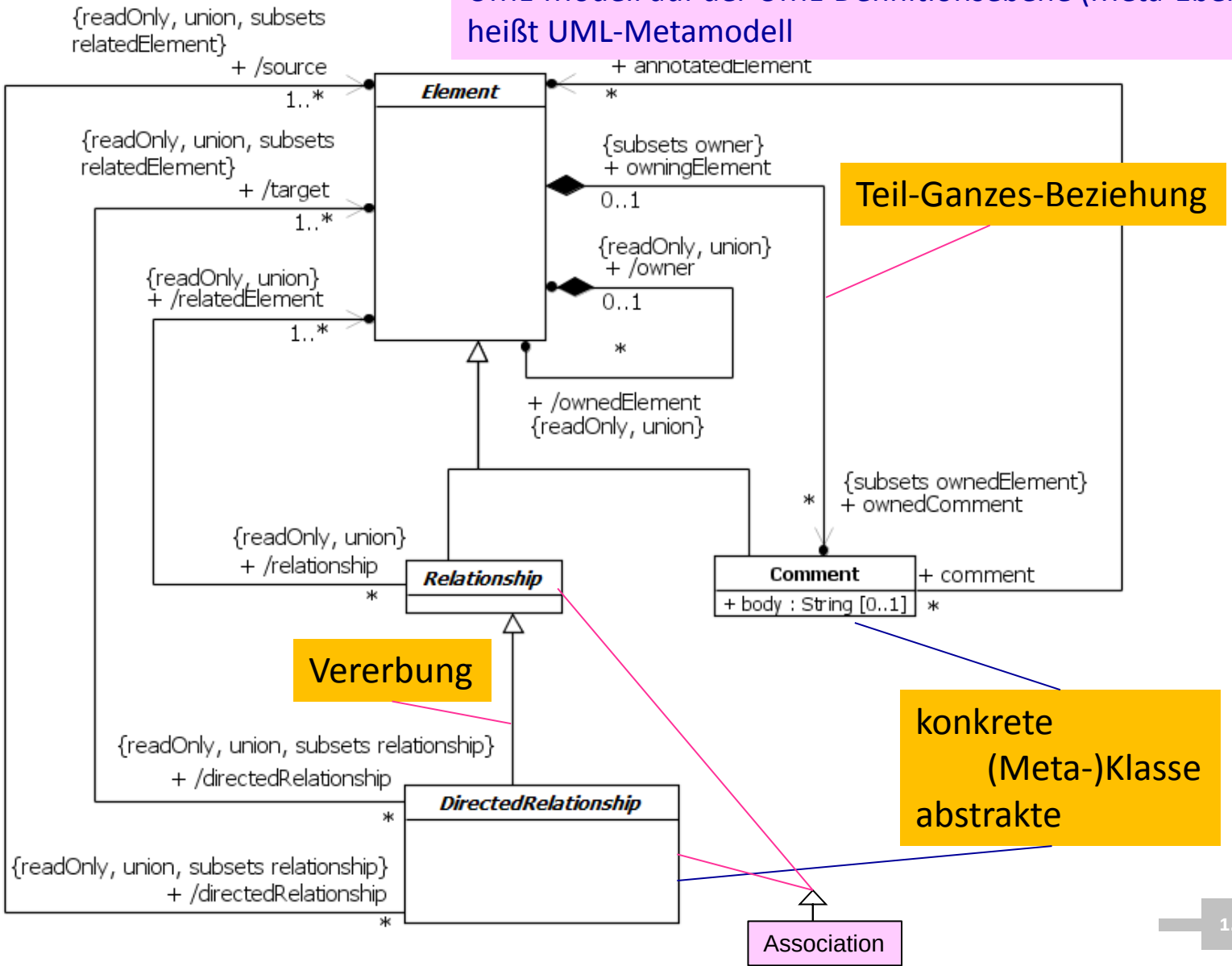
z.B.
- Klasse,
- Interface,
- Port,
- Komponente,

~ 37 verschiedene Konzepte

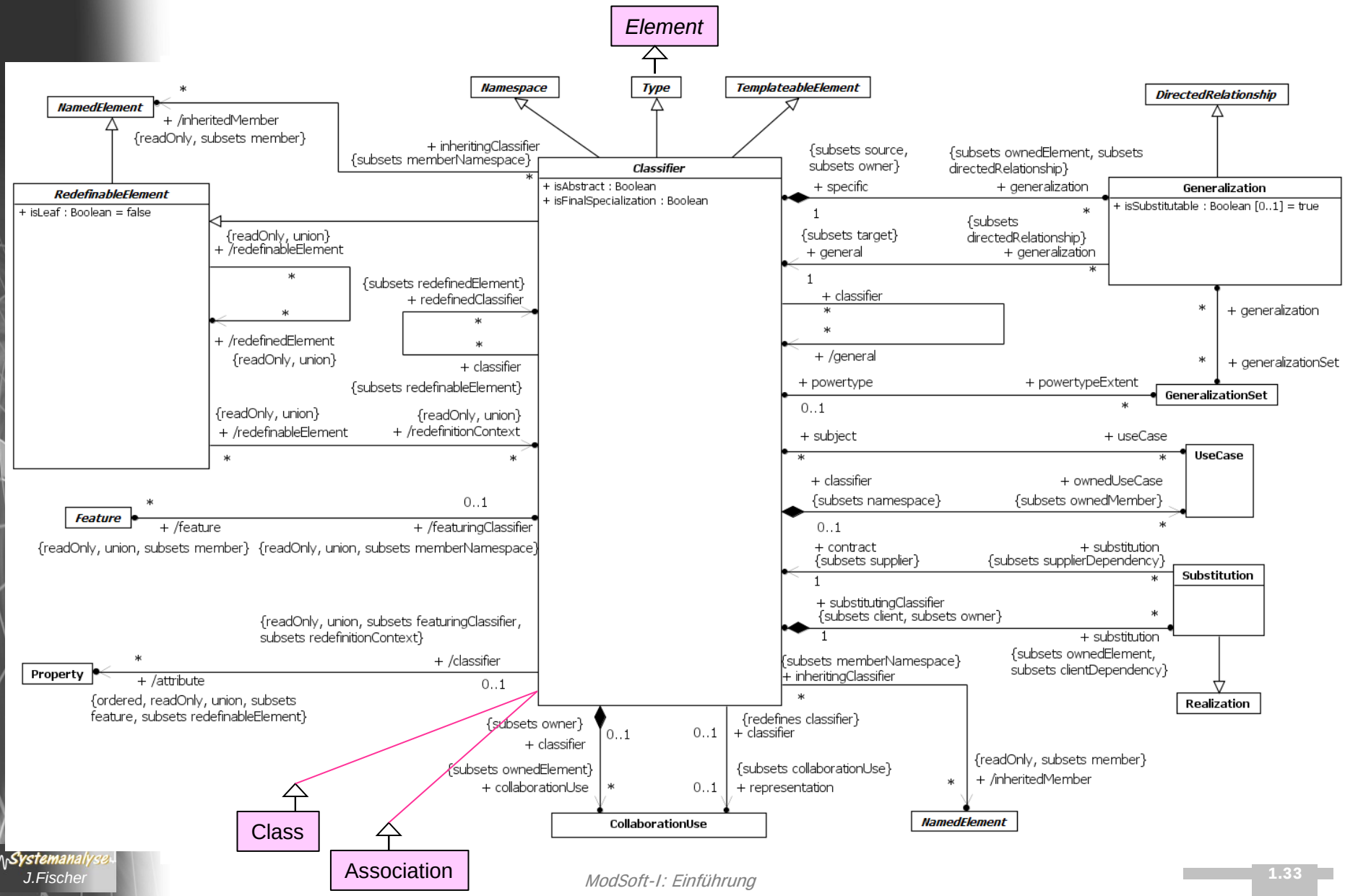
Quelle: OMG Unified Modeling Language TM (OMG UML), *Version 2.5*

UML-Metamodell (ein winziger Auszug)

UML-Modell auf der UML-Definitionsebene (Meta-Ebene) heißt UML-Metamodell



Meta-Ebene: UML-Klasse ist konkreter Classifier



Zentrale UML-Modellelement-Typen

aus diesen Konzepten wird die gesamte UML aufgebaut

1. ***Classifiers***.
   A classifier describes a set of objects. An object is an individual with a state and relationships to other objects. The state of an object identifies the values for that object of properties of the classifier of the object. (In some cases, a classifier itself may also be considered an individual; for example, see the discussion of static structural features in sub clause 9.4.3.)
2. ***Events***.
   An event describes a set of possible occurrences. An *occurrence* is something that happens that has some consequence with regard to the system.
3. ***Behaviors***.
   A behavior describes a set of possible executions. An *execution* is a performance of a set of actions (potentially over some period of time) that may generate and respond to occurrences of events, including accessing and changing the state of objects.

   (As described in sub clause 13.2, behaviors are themselves modeled in UML as kinds of classifiers, so that executions are essentially modeled as objects. However, for the purposes of the present discussion, it is clearer to consider behaviors and executions to be in a separate semantic category than classifiers and objects.)

Quelle: OMG Unified Modeling Language TM (OMG UML), *Version 2.5*