Software Factories

3 Modellgetriebene Softwareentwicklung

WS 2019/20

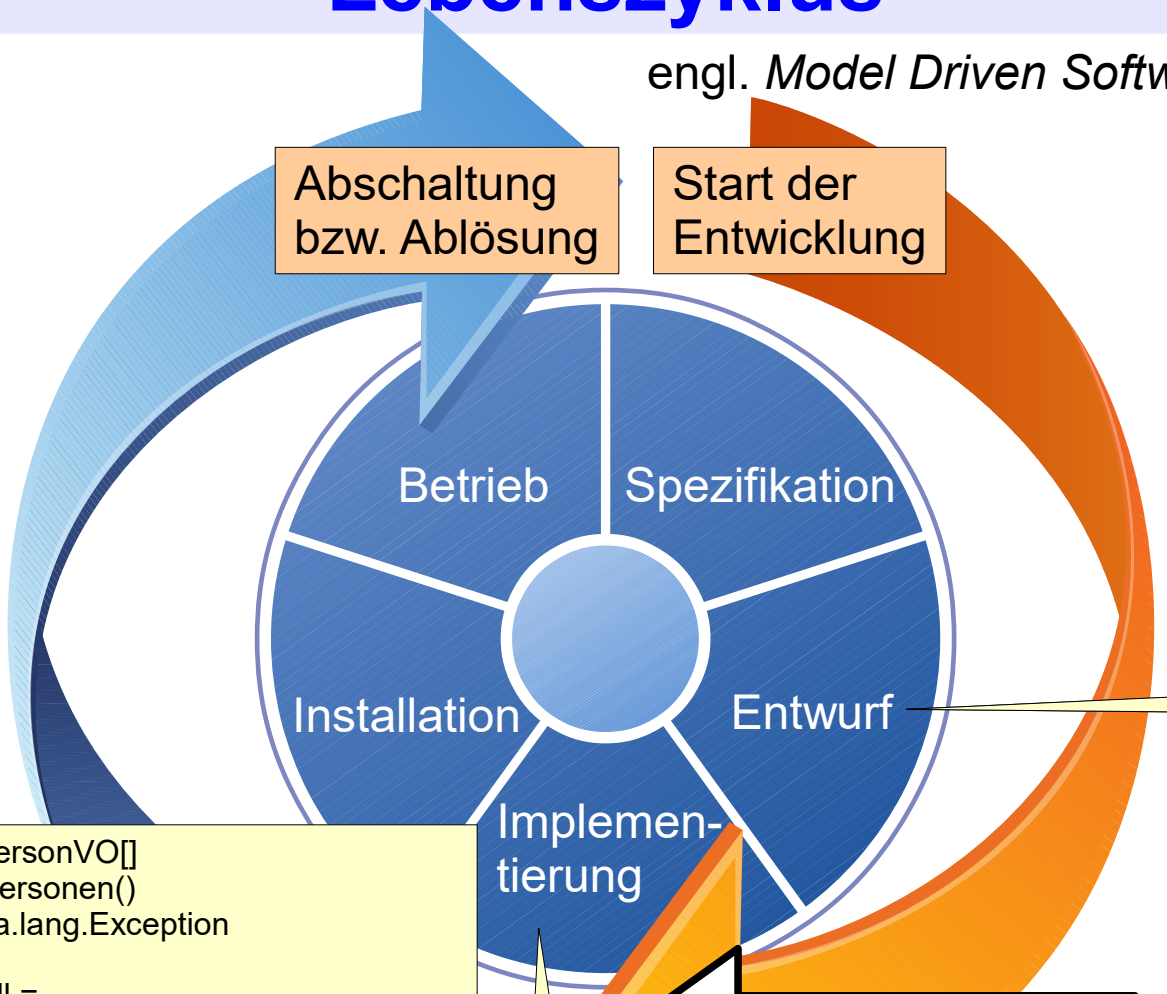
Prof. Dr. Dirk Müller

Übersicht

- Einordnung im Lebenszyklus
- Ziele
- Hebung des Abstraktionsniveaus
- *Model Driven Architecture (MDA)*
- Domänenspezifische Sprachen (DSLs)
- Metamodellierung
  - *Meta Object Facility (MOF)*
  - *Unified Modeling Language (UML)*
- Grundidee
- Entwicklung einer Infrastruktur
- Zusammenfassung

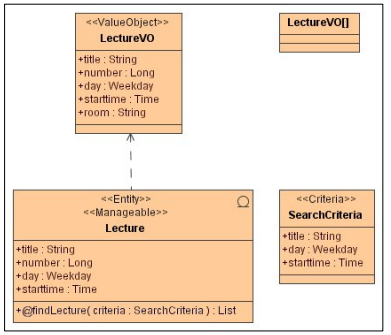
Modellgetriebene SW-Entwicklung (MDSD) im Lebenszyklus

engl. *Model Driven Software Development*




```
 protected vo.PersonVO[]
handleGibAllePersonen()
        throws java.lang.Exception
 {
    Collection coll =
getPersonDao().loadAll(PersonDao.TRANSFO
RM_PERSONVO);
      PersonVO[] personen = new
PersonVO[coll.size()];
      return (PersonVO[])coll.toArray(personen);

  }
```



Dirk Müller: Software Factories

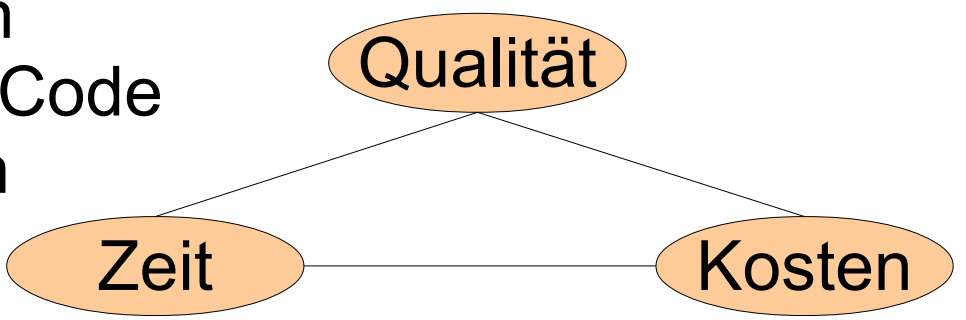
Definition

- „Modellgetriebene Softwareentwicklung (*Model Driven Software Development*, MDSD) ist ein Oberbegriff für Techniken, die aus formalen Modellen automatisiert lauffähige Software erzeugen.“ [StaV07], S. 11
- formale Modelle
  - vollständige Beschreibung eines Aspekts einer Software
  - nicht nur in UML, auch: eigene visuelle Sprachen und textuell
- Erzeugung lauffähiger Software
  - Übersetzung durch Generator und Compiler (zur *Build*-Zeit) oder Interpreter (zur Laufzeit)
  - Nicht-Ausführbarkeit als Indikator für Beschreibungslücken
- automatisch
  - *Build*-Prozess beginnt bei Modellen
  - Modelle nehmen Rolle der Quelltexte ein

Quelle: [StaV07], S. 11 f.

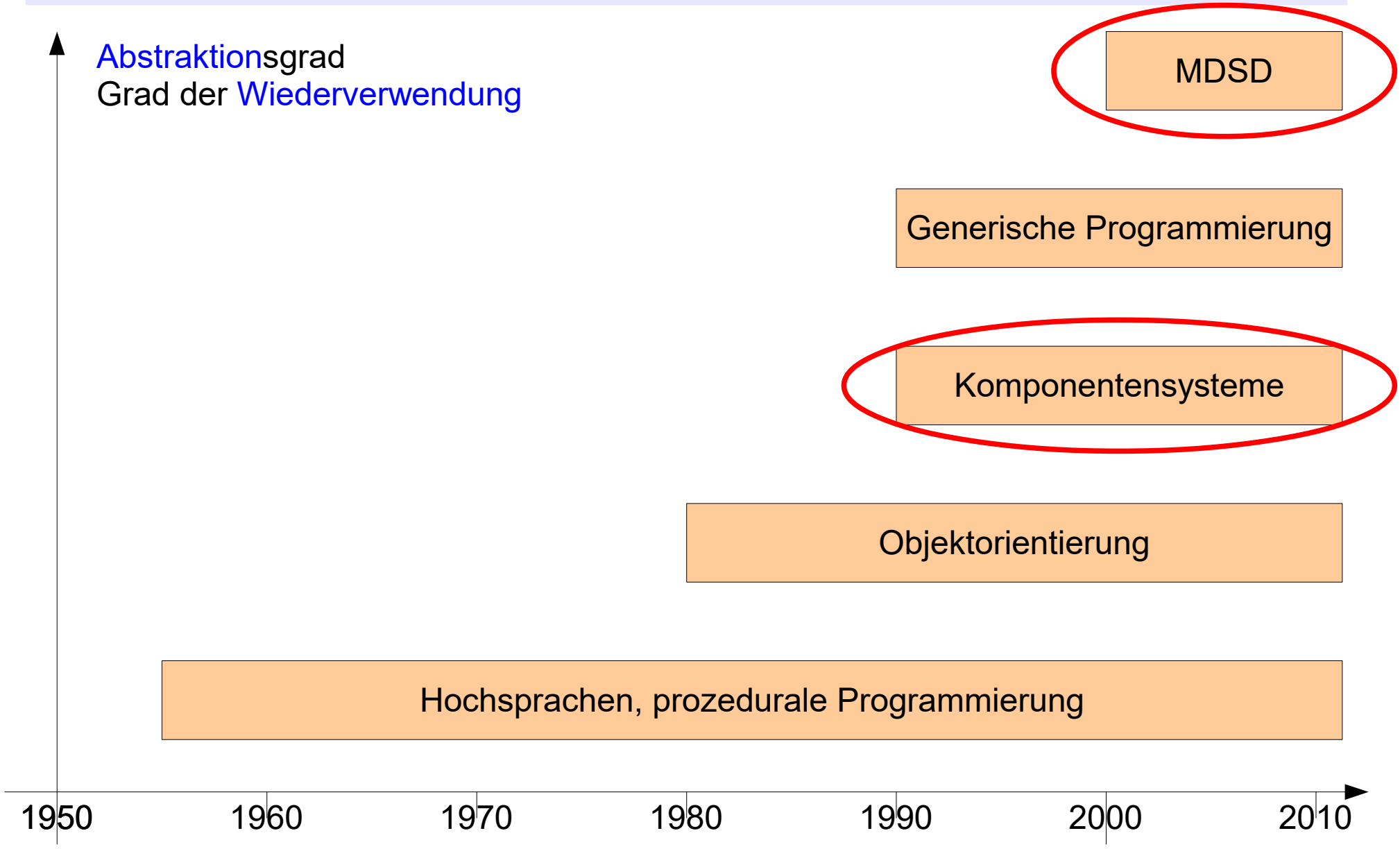
Ziele modellgetriebener Softwareentwicklung

- schnellere Entwicklung durch Automatisierung: lauffähiger Code aus formalen Modellen durch Transformationen



- bessere Qualität der Software: SW-Architektur konsistent in ganzer Implementierung
- Trennung von Verantwortlichkeiten (*Separation of Concerns*): weniger Redundanz, bessere Änderbarkeit
- höherer Grad von Wiederverwendung, Expertenwissen in der Breite verfügbar => Software Factories
- bessere Handhabbarkeit von Komplexität durch Abstraktion
- laut *Model Driven Architecture* (MDA): Interoperabilität (herstellerunabhängig) + Portabilität (plattformunabhängig)

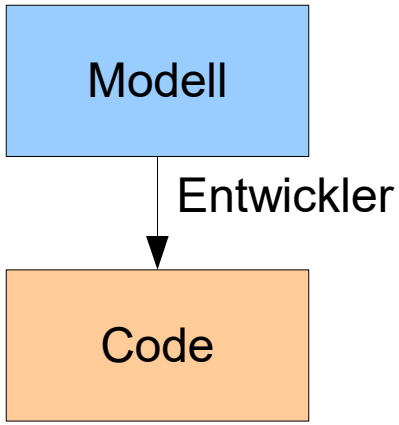
Evolution in der Informatik



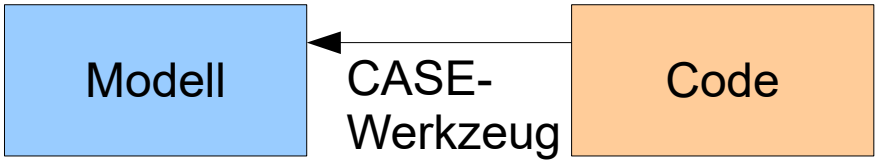
Abgrenzung der MDSD

Modelle sind nichts komplett Neues, allerdings werden sie auf eine neue Art und Weise eingesetzt

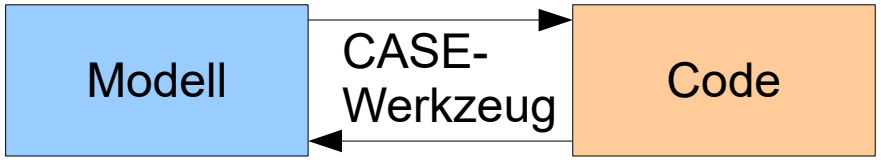
Forward Engineering



Reverse Engineering



Roundtrip Engineering



Model-Driven Software Development

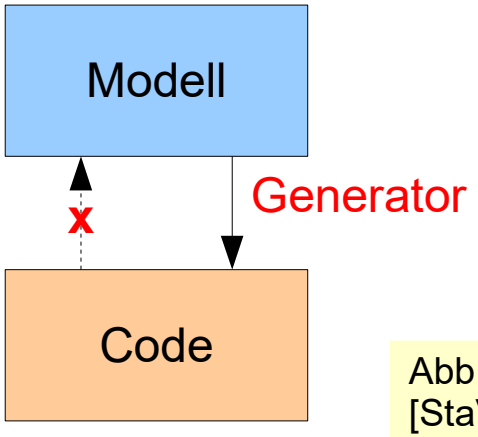
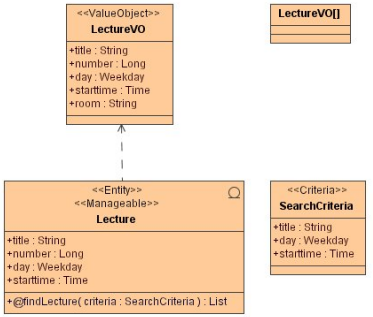


Abb. basiert auf [StaV07]

Hebung des Abstraktionsgrades durch MDSD




```
 protected vo.PersonVO[]
handleGibAllePersonen()
        throws java.lang.Exception
  {
    Collection coll =
getPersonDao().loadAll(PersonDao.TRANSFO
RM_PERSONVO);
      PersonVO[] personen = new
PersonVO[coll.size()];
      return (PersonVO[])coll.toArray(personen);

  }
```

```
LDA 23
SHL 1
MOV $3
```

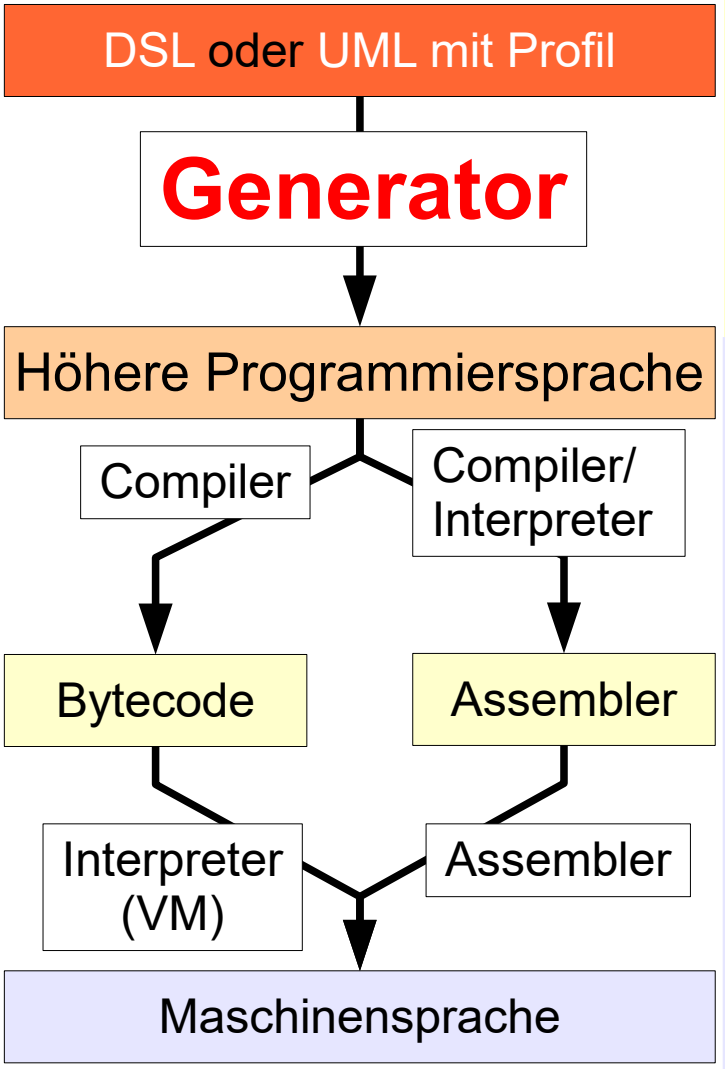
```
11101011001011001010
01000010101110100010
```

2000er

1960er

1950er

1940er

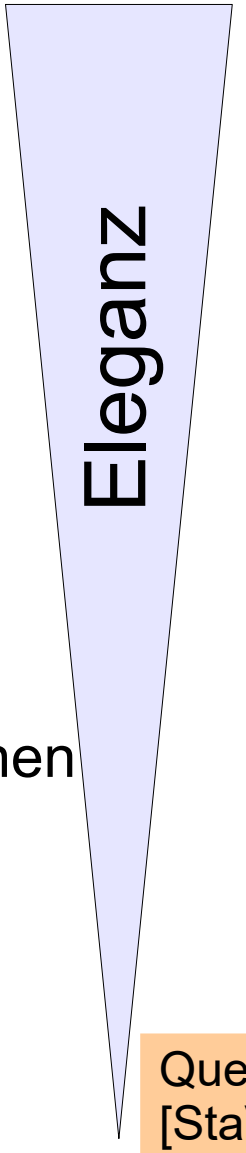


domänen-spezifisch

universell

Generierter und manuell erstellter Code ohne *Roundtrip Engineering*

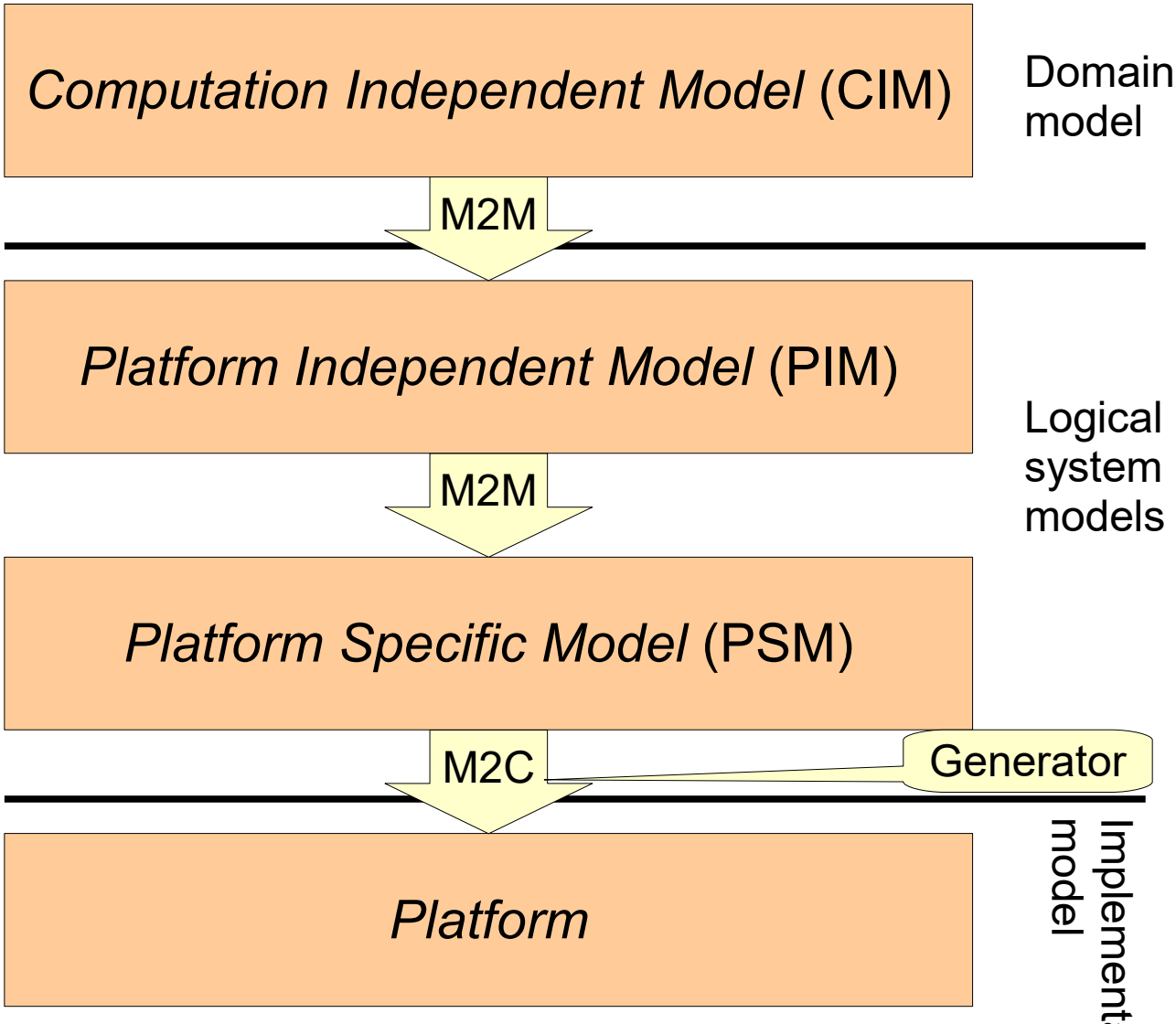
1. Abstraktion
   - entspricht 100%-iger, vollautomatischer Codegenerierung; häufig schwer zu finden
2. *Tagging* des Modells
   - Code-Entscheidungen ins Modell hochgezogen, ohne das Abstraktionsniveau zu erhöhen
   - Gefahr der Verunreinigung des Modells mit Implementierungskonzepten, die für Modellierer/Fachexperten kompliziert/fremd sind
3. Code-Klassentrennung
   - Nutzung von Entwurfsmustern, um handgeschriebenen Code in eigene Klassen auslagern zu können (Proxy, Adapter, Vererbung, *Composite*, *Decorator*)
4. *Tagging* des Codes
   - Kopplung mit Trennmarkierungen in einer Datei
   - Versionsverwaltung wird erschwert



Quelle: [StaV07]

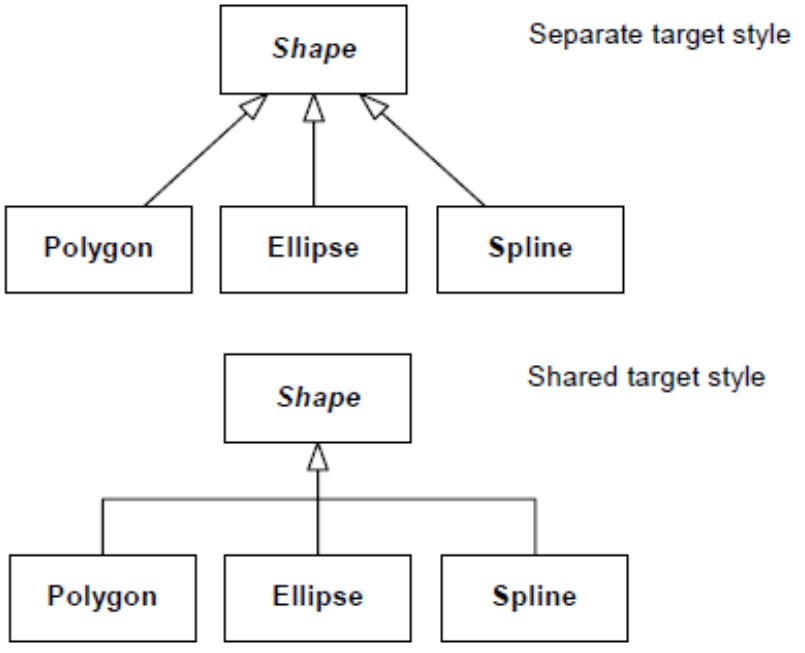
Model Driven Architecture (MDA)

- OMG-Entwicklung
- Motto: „*Model once, run anywhere.*“
- Zielplattformen
  - *CORBA/C++* bzw. *.NET/C#*
  - *J2EE/Java*
  - *XML*
- Transformationen
  - Modell-zu-Modell
  - Modell-zu-Code



Syntax formaler Sprachen

- Syntax
  - abstrakte Syntax: Modellelemente + Beziehungen zwischen ihnen
  - konkrete Syntax: Darstellung von Modellen und Texten, kann grafisch oder textuell sein
- abstrakte Syntax mit mehreren konkreten Syntaxen (verschiedenen Darstellungen) möglich
- UML: „*presentation option*“ als Spielraum bei konkreter Syntax
  - z. B. Generalisierungsbeziehung zwischen Klassen



Semantik formaler Sprachen

- statische Semantik
  - Wohlgeformtheitskriterien
  - legt Nebenbedingungen (*Constraints*) für Modell/Text fest
  - Nebenbedingungen eines Modells in der UML in der deklarativen, seiteneffektfreien *Object Constraint Language* (OCL) spezifiziert

    Company::recruit(p:Person)

    ```
    pre: -- none
    post: (employees.size=employees@pre.size+1) &&
          (employees.includes(p))
    ```
  - z. B. Variablendeklaration in vielen Hochsprachen nötig
  - MDSD: Erkennung von Modellierungsfehlern bereits zur Modellierungszeit möglich
- dynamische Semantik
  - Bedeutung der Modell-/Sprachelemente
  - gut möglich durch Referenzimplementierung und -modell oder durch Transformationsvorschriften

Domänenspezifische Sprachen (DSLs)

- engl. *domain-specific language*
- Def. DSL: Modellierungs-/Programmiersprache, die für ein begrenztes Wissensgebiet, eine Domäne, eingesetzt wird (Fachsprache)
- Abgrenzung gegen universelle Sprachen (UML, Hochsprachen)
- z. B. *LaTeX*, *SQL, EJB-Profil der UML, Petri-Netze*
- Bestandteile
  - konkrete Syntax: Symbole (textuell/grafisch)
  - abstrakte Syntax
  - statische Semantik

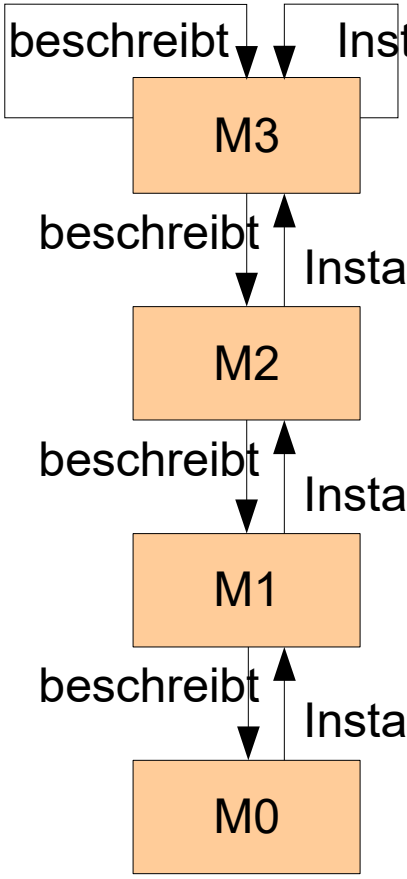
  } Metamodell
  - dynamische Semantik

Ziele der Metamodellierung

- zentraler Ansatz, um MDSD zu ermöglichen
- Konstruktion domänenspezifischer Modellierungssprachen
  - *Domain-specific language* (DSL)
  - Metamodell beschreibt abstrakte Syntax (Struktur der Sprache) und statische Semantik (Wohlgeformtheitskriterien, *Constraints*)
- Modellvalidierung
  - gegen *Constraints* des Metamodells
- Beschreibung von M2M-Transformationen
  - Abbildungsvorschrift zwischen zwei Metamodellen
- Codegenerierung
  - Generierungs-Templates mit Bezug auf das Metamodell der DSL
- Werkzeugintegration
  - Anpassung von Modellierungswerkzeugen an die Domäne

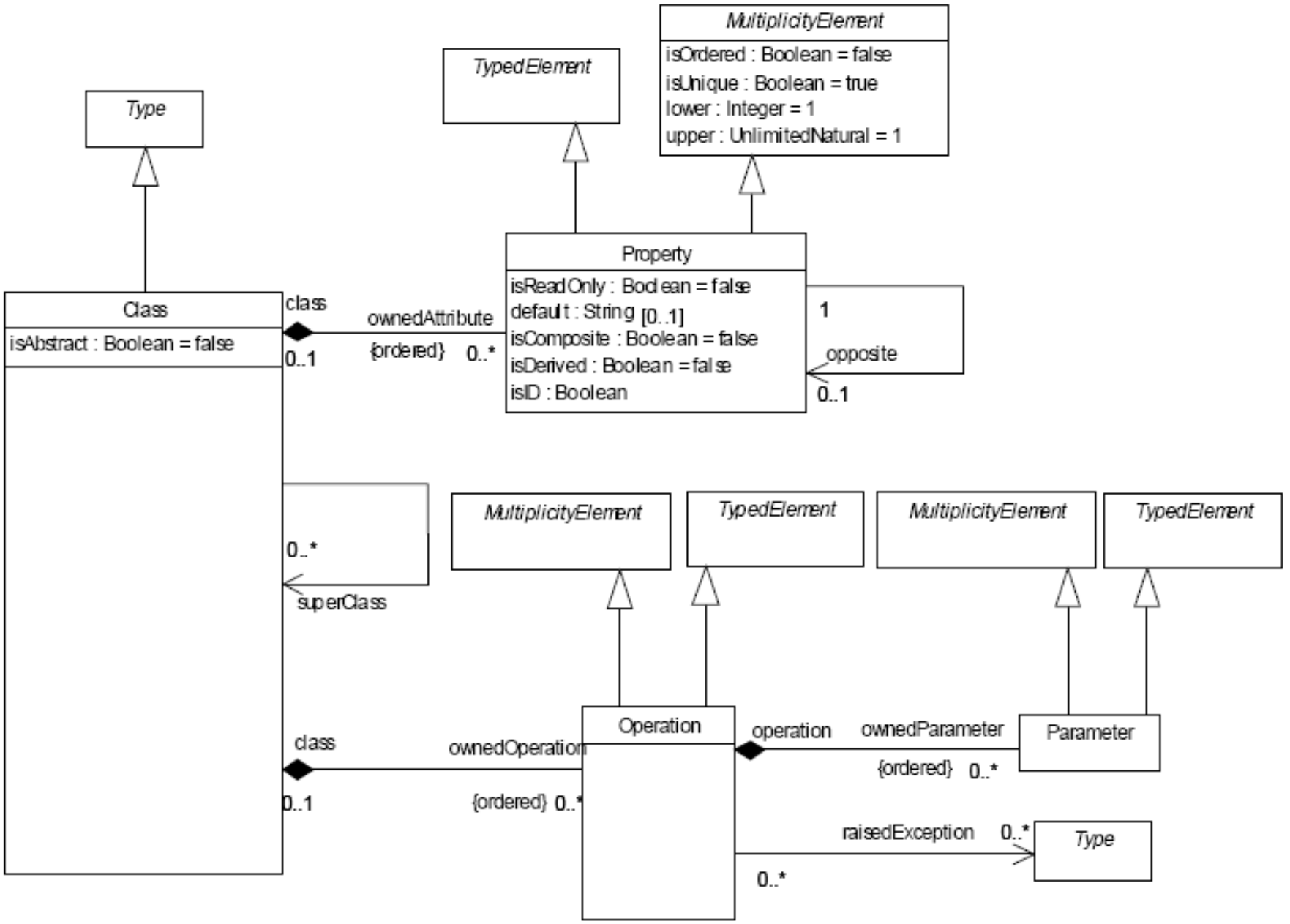
Quelle: [StaV07]

Meta Object Facility (MOF)

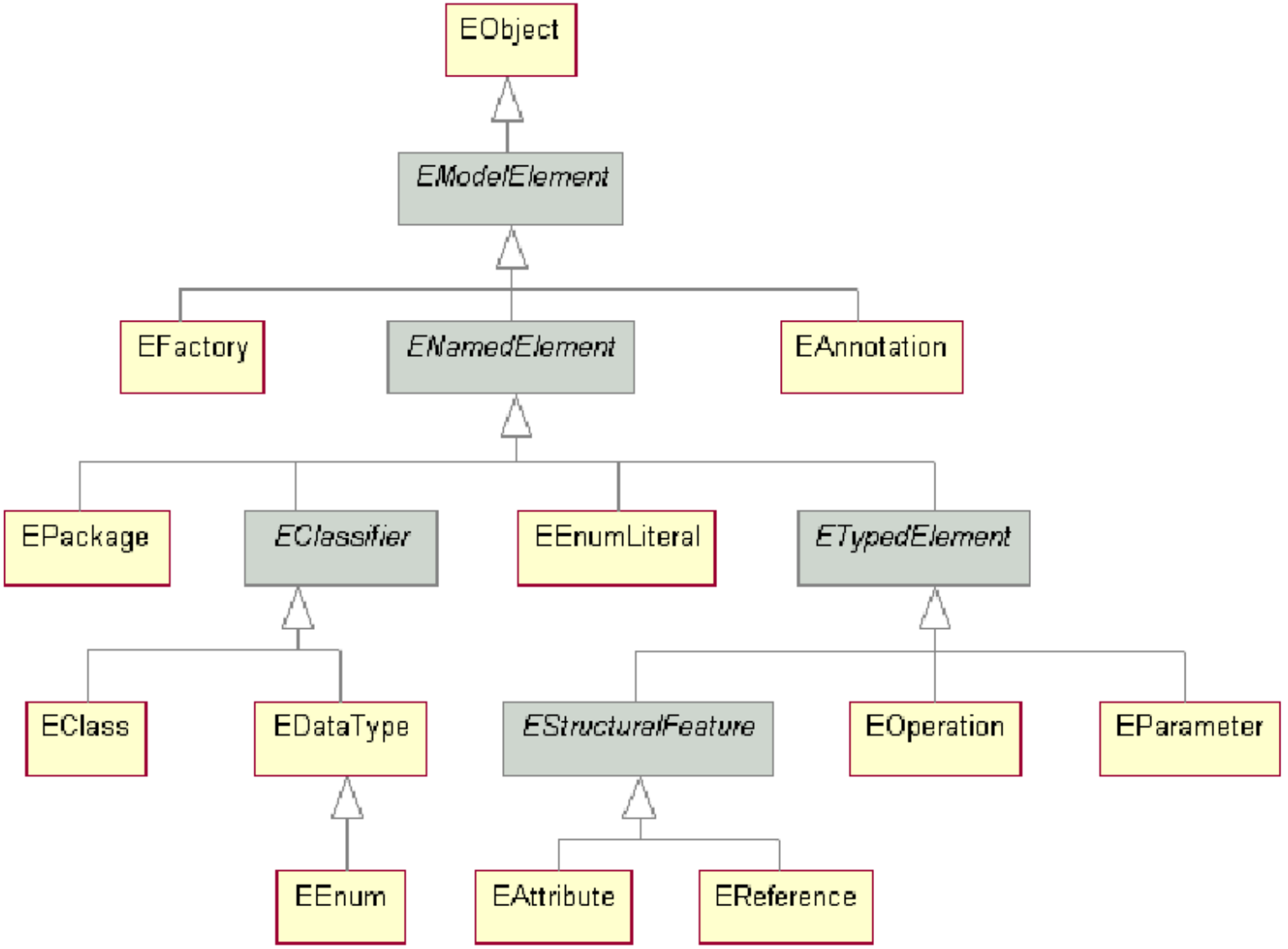


Schicht	Beschreibung	Beispiel
M3	„Vokabular“, Metametamodell	*Meta Object Facility* (MOF)
M2	„Grammatik”, Metamodell	UML-Metamodell
M1	„Sprache”, (Nutzer-)Modell	Nutzer-Modelle in UML
M0	„Realität”, Datenschicht, Instanzen zur Laufzeit	Objekte

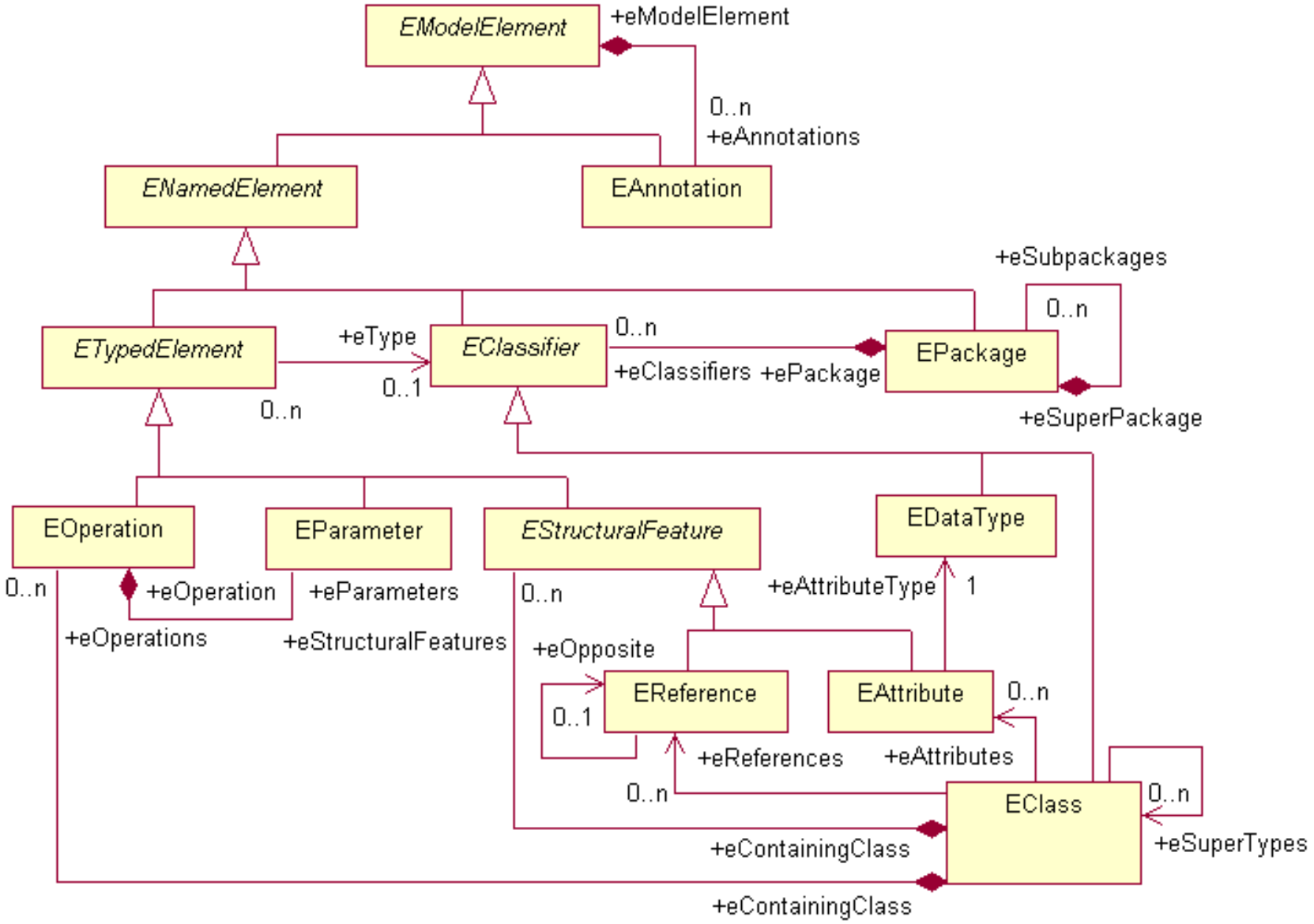
Auszug aus *Essential MOF*: Klassen



Ecore-Klassenhierarchie



EMF-Metametamodell (*Ecore*)

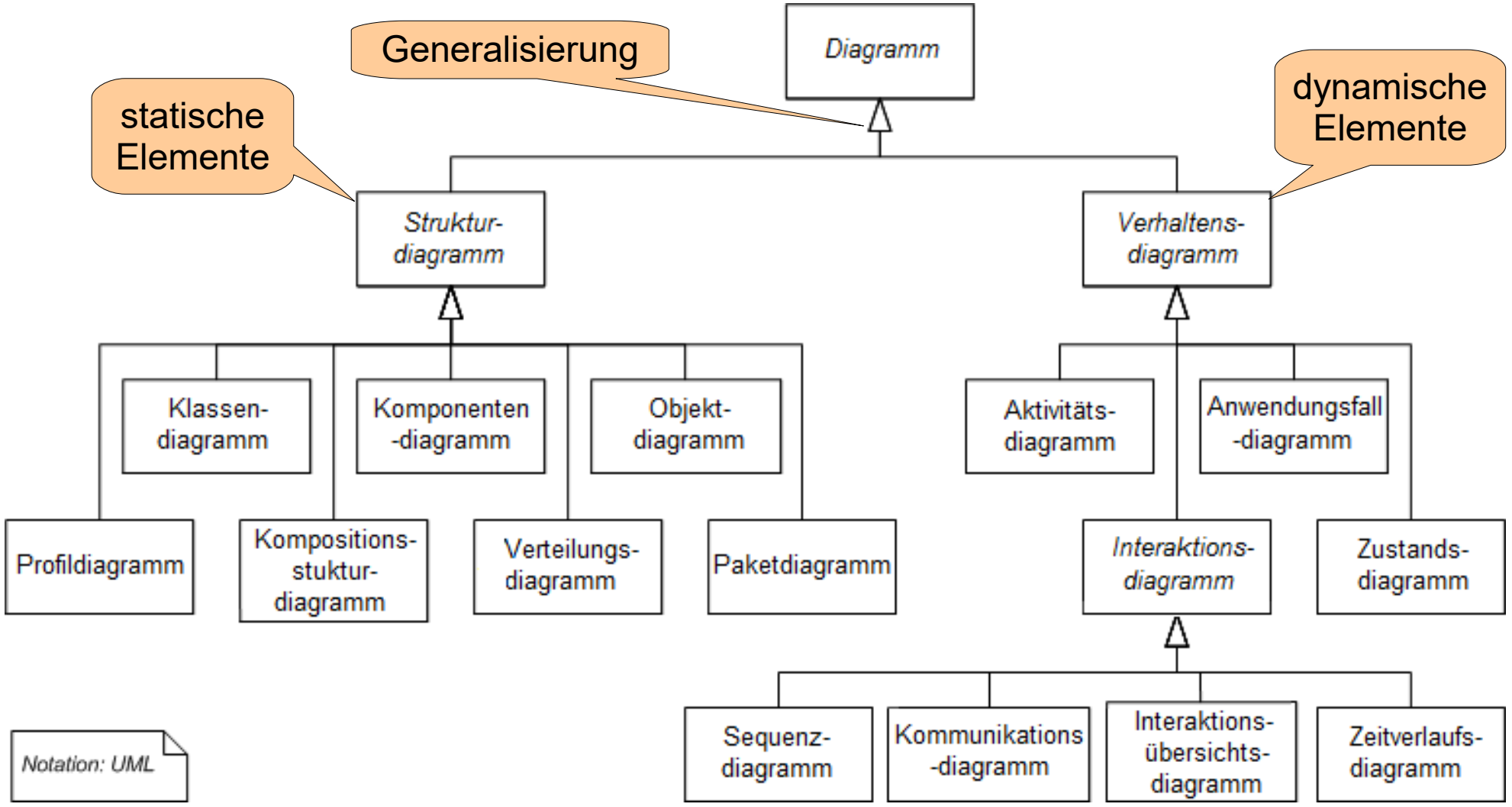


Unified Modeling Language (UML)

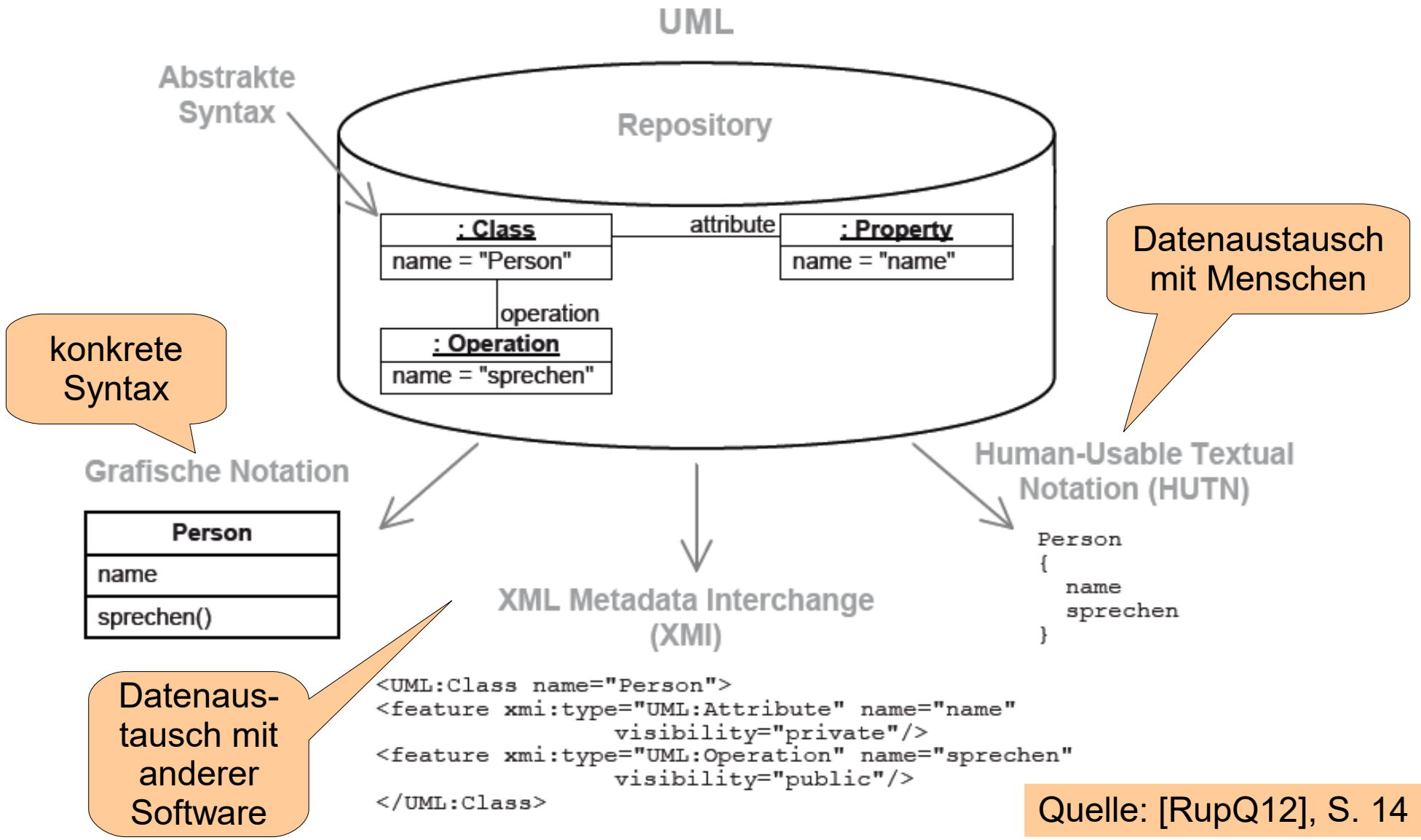
- *Grady Booch*, *Ivar Jacobson*, *James Rumbaugh*, 1990er
- vereinheitlichte Modellierungssprache
- aktuelle Version: 2.5 (Juni 2015)
- UML-2-Spezifikation mit klassisch 4 Teilen
  - *Infrastructure Specification*
  - *Superstructure Specification*
  - *Object Constraint Language (OCL)*
  - *Diagram Interchange*
- abstrakte Syntax durch Metamodell
- konkrete Syntax durch visuelle Modellierungselemente
  - z. B. Rechteck für eine Klasse
- statische Semantik durch OCL-*Constraints*

nicht mehr extra seit 2.5

UML-Diagrammarten

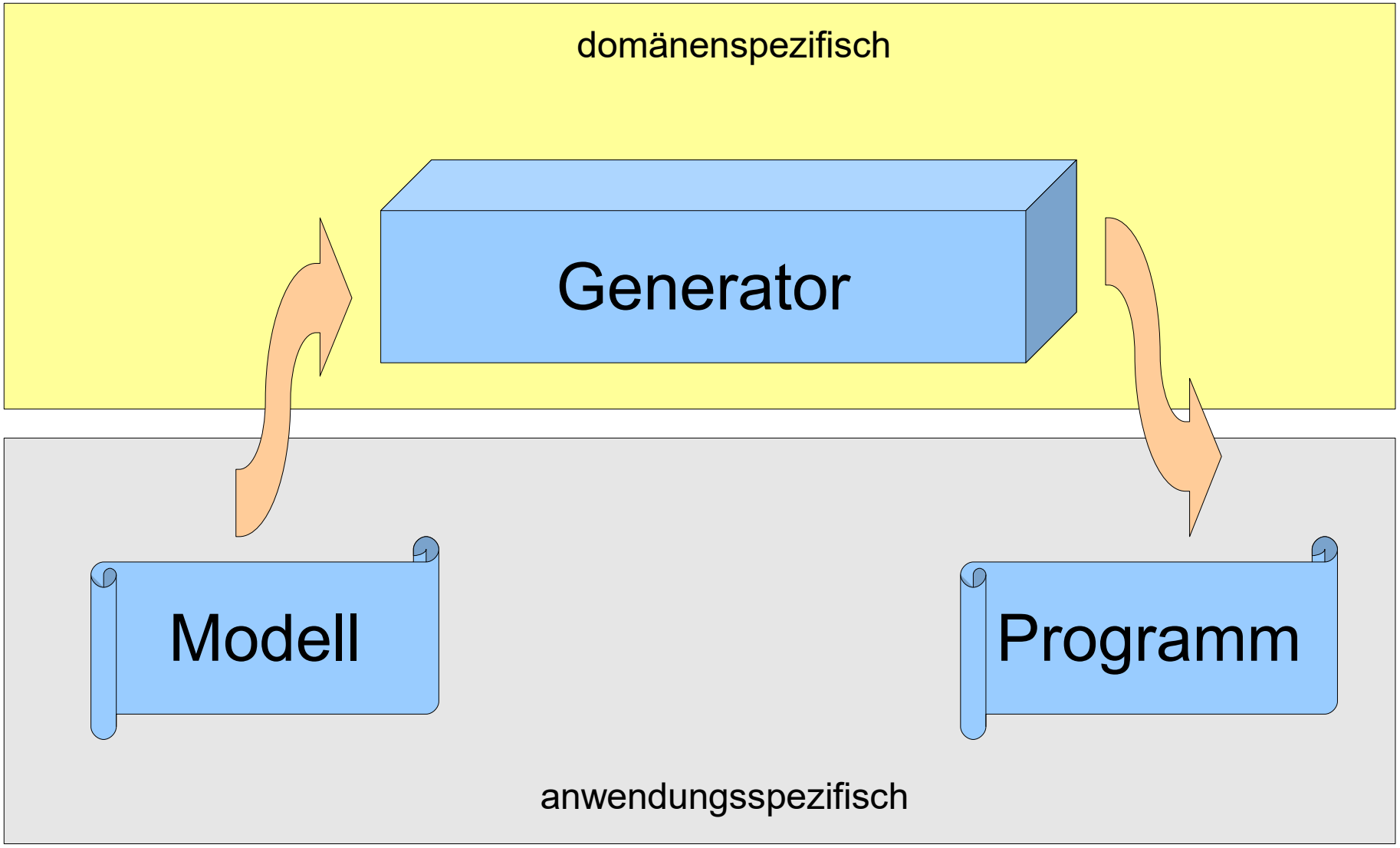


Repräsentation von UML-Elementen



Quelle: [RupQ12], S. 14

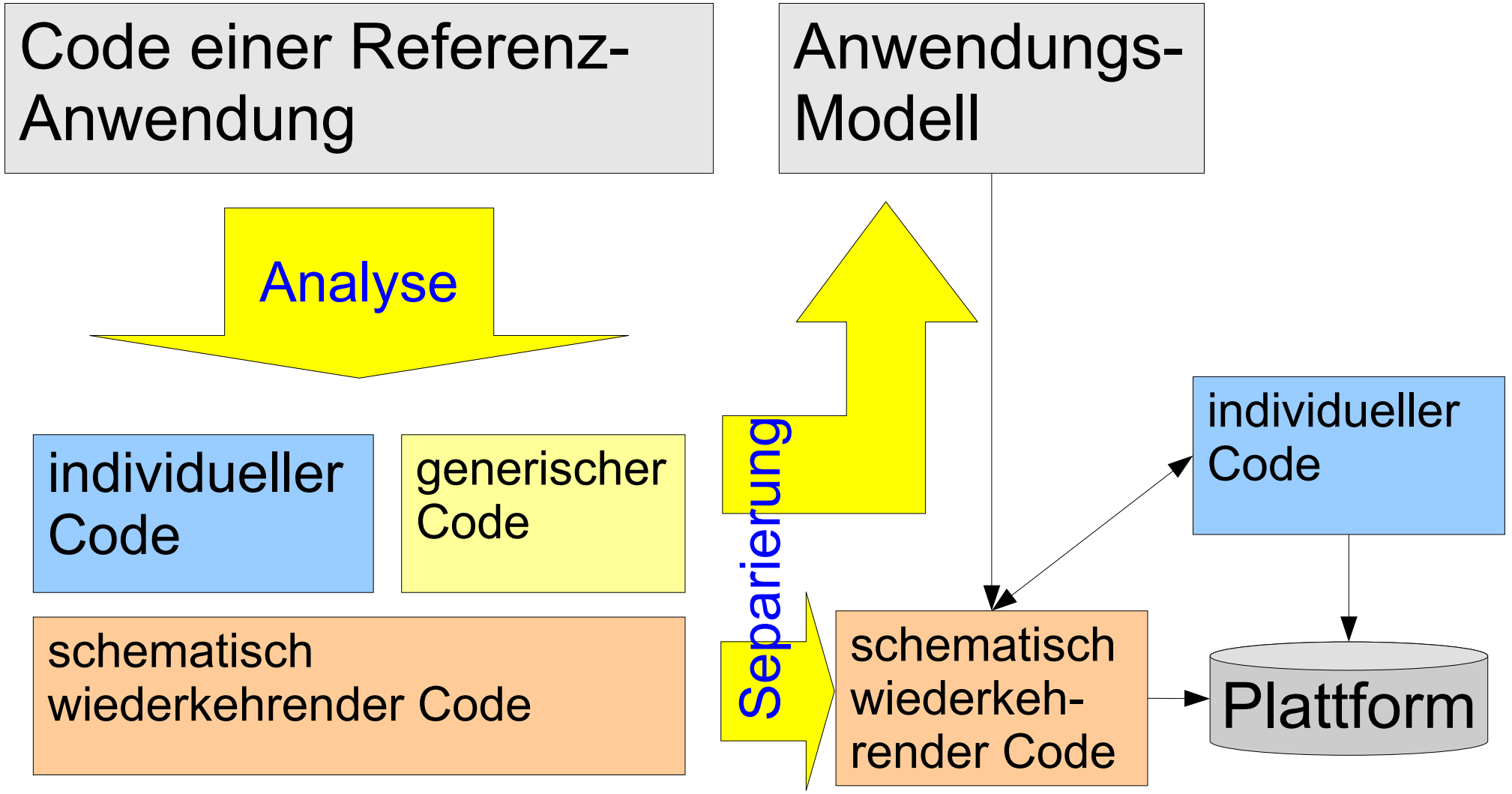
Prinzip der MDSD



Entwicklung einer MDSD-Infrastruktur

- Ursprung: Entwicklung von Softwareproduktlinien
- zweigleisige Softwareentwicklung
  - eine konkrete Applikation
  - MDSD-Infrastruktur für die Entwicklung von Softwarefamilien
- MDSD-Infrastruktur
  - Modellierungssprache
    - Metamodell für abstrakte Syntax und statische Semantik
    - konkrete Syntax
  - Generator
  - Zielplattformen
- „Henne-oder-Ei“-Problem
  - Infrastruktur und Applikation werden verschränkt entwickelt

Grundlegende Idee hinter MDSD



Agile Techniken und MDSD

- Paarprogrammierung
  - Paar kann jeweils aus einem technischen und einem Domänenexperten gebildet werden
- Testgetriebene Entwicklung
  - Testcode kann ebenso aus dem Modell generiert werden
- *Refactoring*
  - Codierung der Architektur in den Transformationen als konsequente Fortführung des Ansatzes
  - auch auf Modellen und Transformationen möglich
- *Timeboxing*
  - iterativ-inkrementelle Entwicklung mit Prototypen
  - durch Generatoren und Compiler schneller *Build* unterstützt
- Fazit: gute Verträglichkeit

Quelle: [StaV07]

Zusammenfassung

- höhere Abstraktionsebene
  - grafischer/textueller Entwurf (in DSL) wird in textuelle Implementierung (Code in Hochsprache) automatisch überführt
- MDA fokussiert auf
  - Interoperabilität: Herstellerunabhängigkeit durch Standards
  - Portabilität: Generierung von Code für verschiedene Plattformen
- Metamodellierung zentral
  - Beschreibung domänenspezifischer Sprachen (DSLs) und der Transformationen zur Codegenerierung
  - MOF als Metadaten-Architektur mit 4 Ebenen
- Generator erzeugt schematisch wiederkehrenden Code aus Modell
  - weniger Redundanz, bessere Änderbarkeit
  - schneller (bei Softwarefamilie) und bessere Qualität

Literatur

[1] Object Management Group: “Model Driven Architecture (MDA), MDA Guide rev. 2.0“, 1.6.2016, Download am 13.3.2016, http://www.omg.org/cgi-bin/doc?ormsc/14-06-01.pdf

[2] Object Management Group: „OMG Unified Modeling Language ™ (OMG UML), Version 2.5“, 2015, Download am 14.3.2016, http://www.omg.org/spec/UML/2.5/PDF