

# Was ist eigentlich aus MDA geworden?

Prof. Dr.-Ing. Rainer Neumann



# JUG Karlsruhe

[jug-ka.de](http://jug-ka.de) | [twitter.com/@jugka](https://twitter.com/@jugka)



# Zur Person

- ↗ 1990 - 1995 Studium der Informatik (TH Karlsruhe)
- ↗ 1995 - 2000 Wiss. Mitarbeiter (TH Karlsruhe)
  - ↗ Objekt-, Komponententechnologie und Softwareprozesse
  - ↗ Generative Programmierung und Übersetzerbau
- ↗ 2000 - 2001 IT-Consultant (Heyde AG)
  - ↗ Entwicklung von Frameworks für personalisierte Web-Sites
  - ↗ Personalisierung im Private Banking der Deutschen Bank
- ↗ 2001 - 2010 PTV AG, Karlsruhe
  - ↗ Entwicklung flexibler und skalierbarer Softwarelösungen
- ↗ Projektleitung verschiedener Logistikprojekte
- ↗ Entwicklungsleiter für den Bereich Kerntechnologien und Komponenten (Basisentwicklung)
- ↗ Vice President Software Development
- ↗ 2001 - 2006 Referent der DIA
  - ↗ Software aus Komponenten (mit Prof. Aßmann)
  - ↗ MDA in der Praxis
- ↗ Seit 09/2010 Professor an der HS Karlsruhe
  - ↗ Wirtschaftsinformatik
  - ↗ Schwerpunkt: IT-Systeme und ihre Modellierung

# Ein paar „Trendanalysen“ zum Einstieg

Im folgenden: Google-Trends-Charts (weltweit)

# MDA (Quelle: wikipedia)

Die Abkürzung **MDA** steht unter anderem für:

- Minimum Descent Altitude (Luftfahrtabkürzung), [Mindestsinkflughöhe](#) eines Flugzeuges während des Landeanfluges bei Instrumentennavigation
- [Missile Defense Agency](#), das Amt für Raketenverteidigung des US-Verteidigungsministeriums
- [3,4-Methylendioxyamphetamine](#) (ein Ecstasy-Wirkstoff)
- [Mail Delivery Agent](#), eine Software, die eingehende E-Mails annimmt und sie nach bestimmten Kriterien unter den Empfängern verteilt
- [Mobile Digital Assistant](#), Produktnname für einen PDA mit integriertem Mobiltelefon
- [Moldawien](#), ISO 3166 und olympisches Länderkürzel
- [Model Driven Architecture](#), in der Informatik ein Ansatz zur Verbesserung der Softwareentwicklung
- [Monochrome Display Adapter](#), Monochrom-Grafikkarte der ersten IBM-PCs
- [4,4'-Diaminodiphenylmethan](#) (= Methylendianilin), Grundstoff der Polymerchemie
- [Medizinischer Dokumentationsassistent](#), eine Berufsbezeichnung
- [Magen David Adom](#) (Roter Davidstern), der israelische Rettungsdienst
- Message Digest Algorithm, Verfahren zur Berechnung eines [Message Digest](#)
- die [Mobilitätsdrehscheibe Augsburg](#), ein Großprojekt zur Neugestaltung des Augsburg Hauptbahnhof und des Königsplatz sowie zum Ausbau des ÖPNV
- Malondialdehyd ( $OHC-CH_2-CHO$ ), ein Abbauendprodukt mehrfach ungesättigter peroxidierter [Fettsäuren](#)
- [Multidisplacement Amplification](#), eine Methode, um Gesamtgenome zu amplifizieren
- Manual Data Automatic, die Betriebsart "Handeingabe" bei einer [Siemens-Steuerung](#) in der CNC-Technik

Die Abkürzung **MdA** steht unter anderem für:

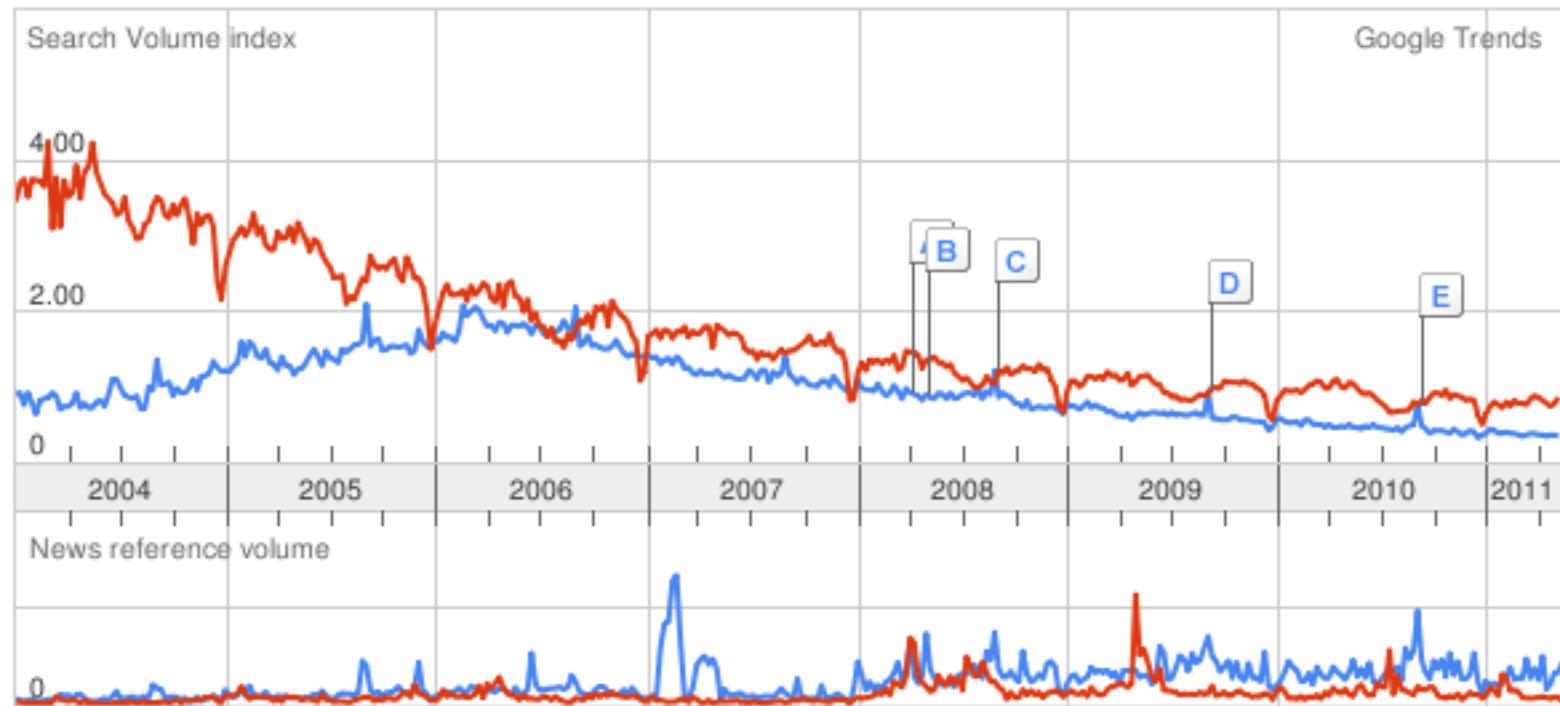
- [Museum der Arbeit](#) in Hamburg
- Mitglied des Abgeordnetenhauses, siehe [Mitglied des Landtages](#)

# MDA



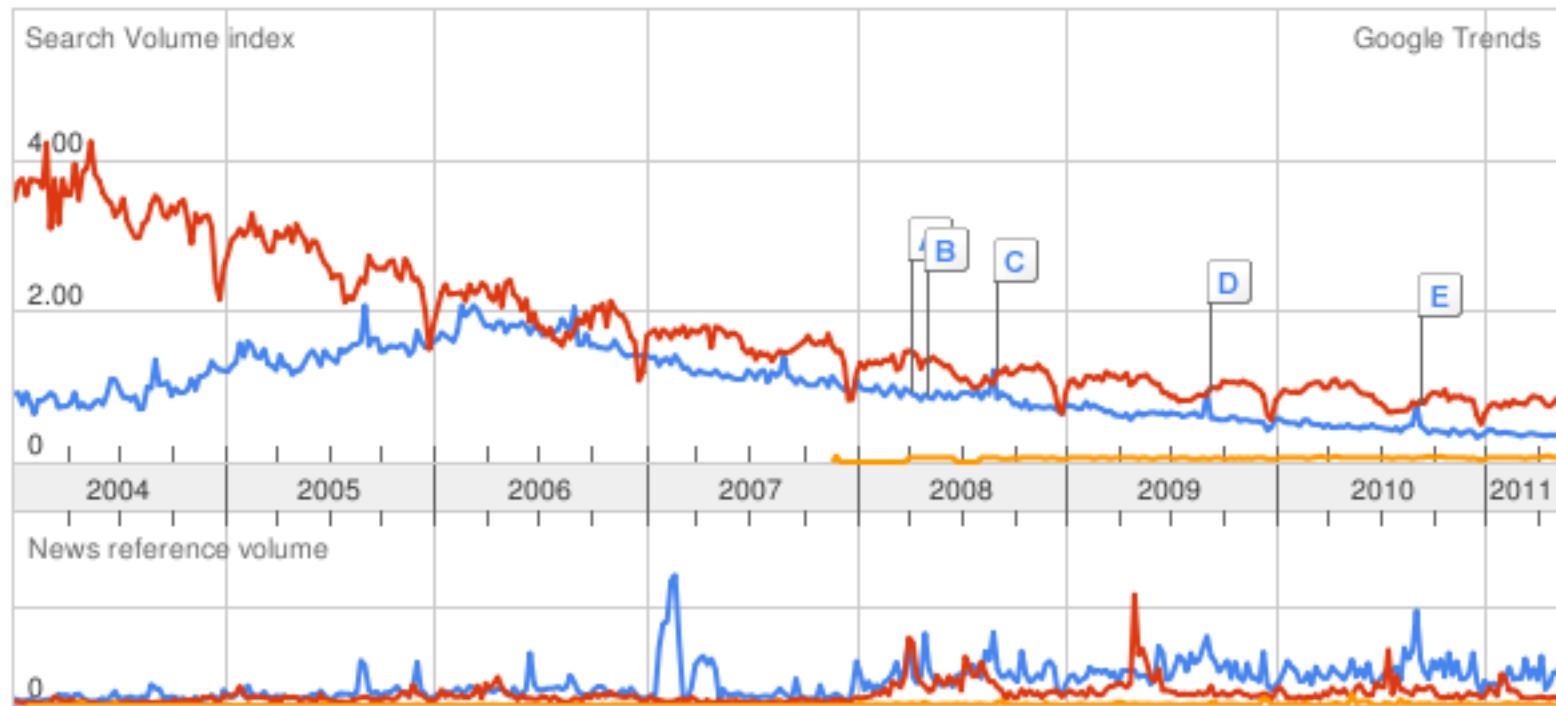
Quelle: Google Trends

# MDA, UML



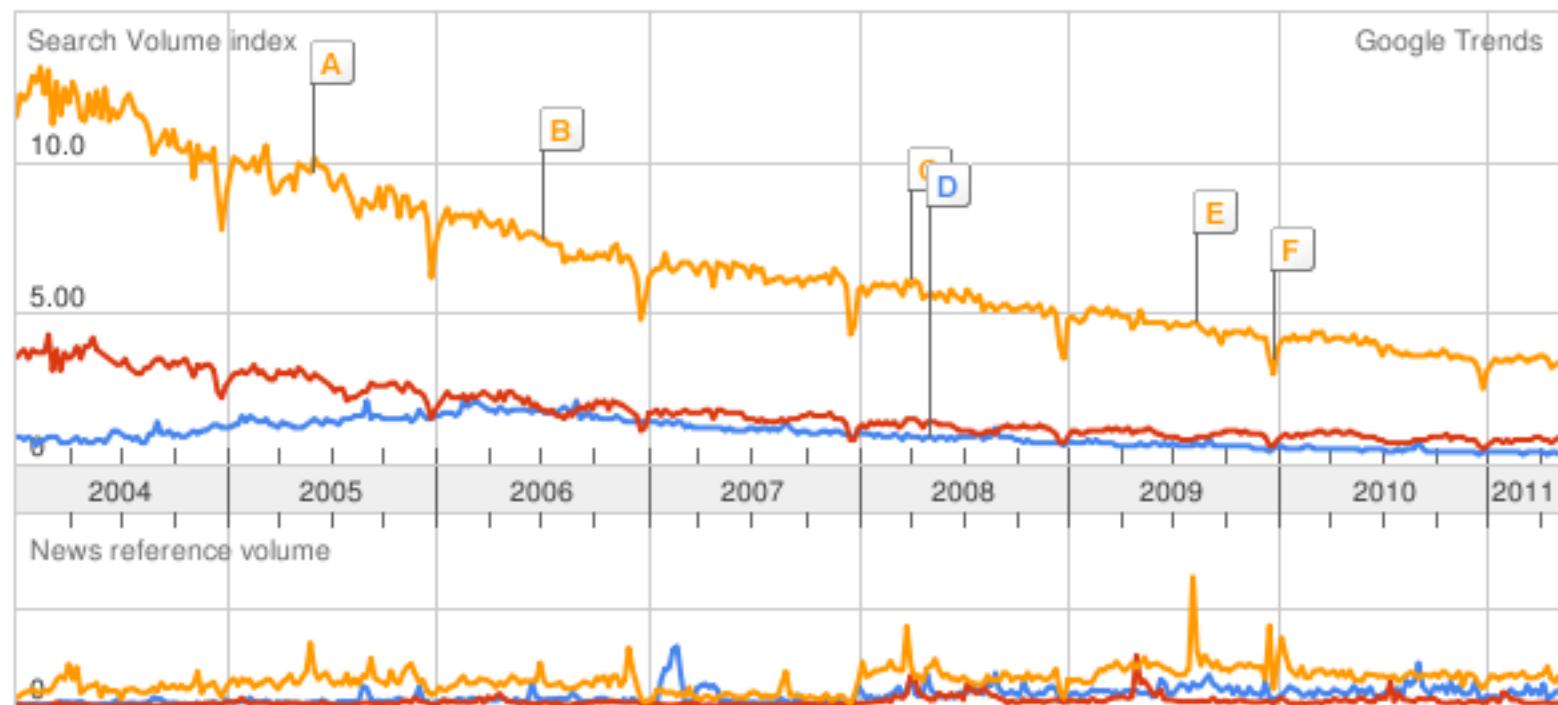
Quelle: Google Trends

# MDA, UML, BPMN



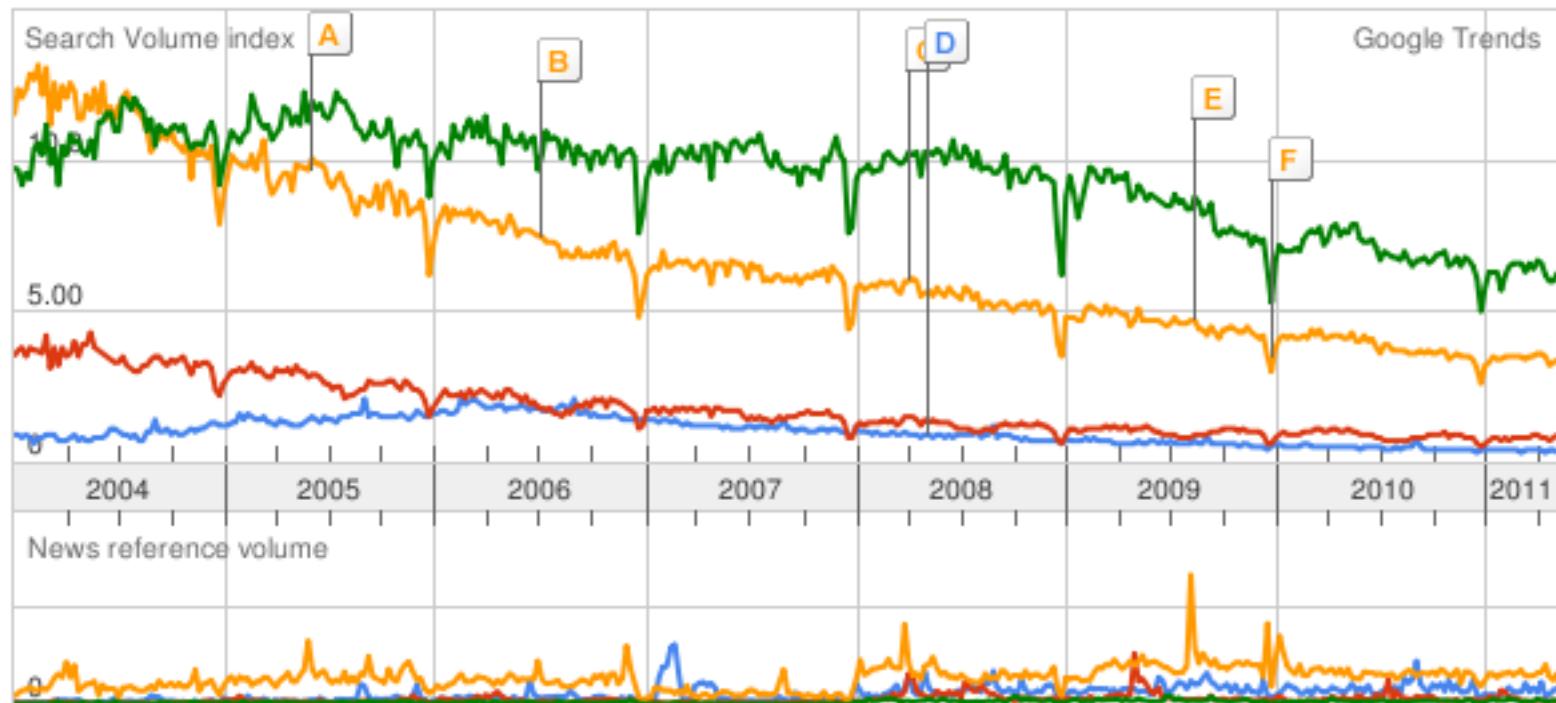
Quelle: Google Trends

# MDA, UML, XML



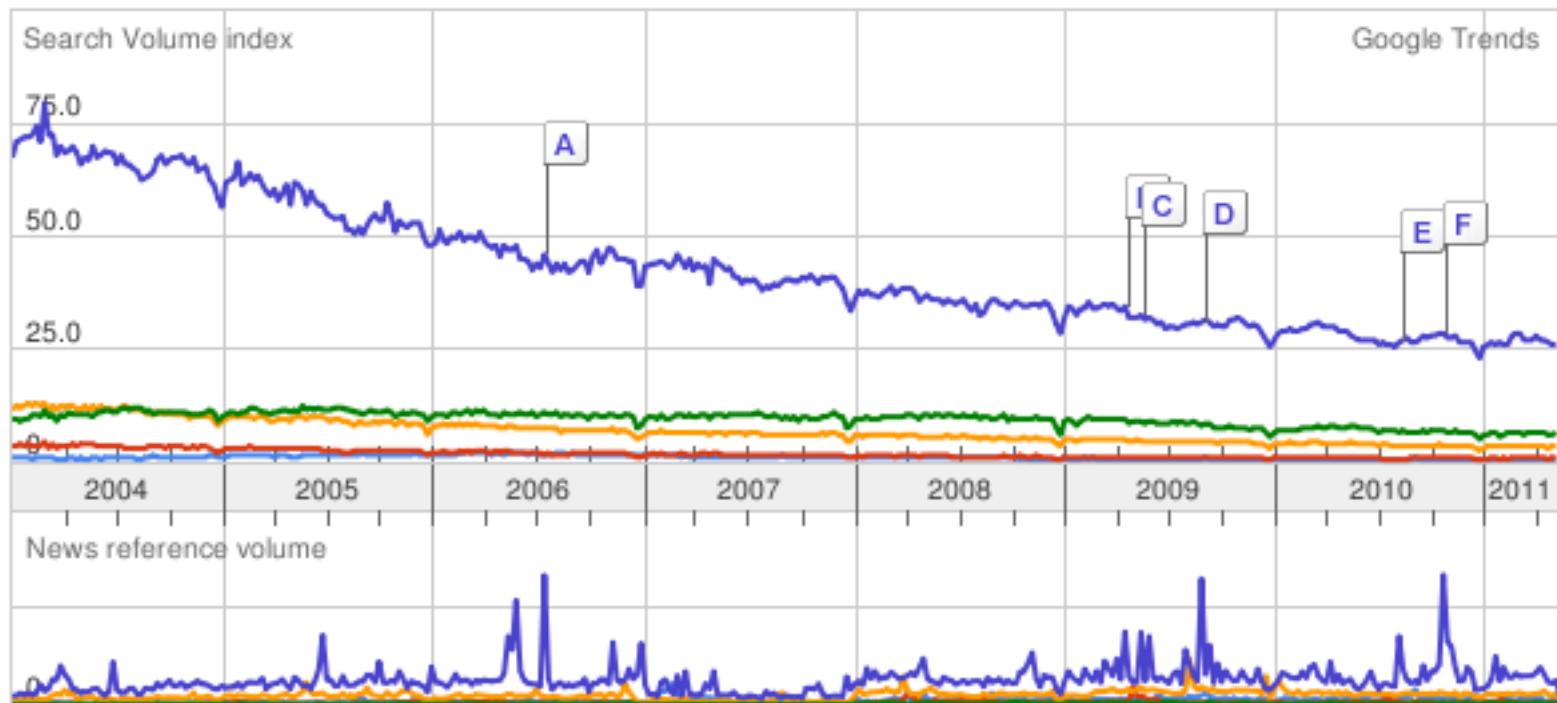
Quelle: Google Trends

# MDA, UML, XML, C#



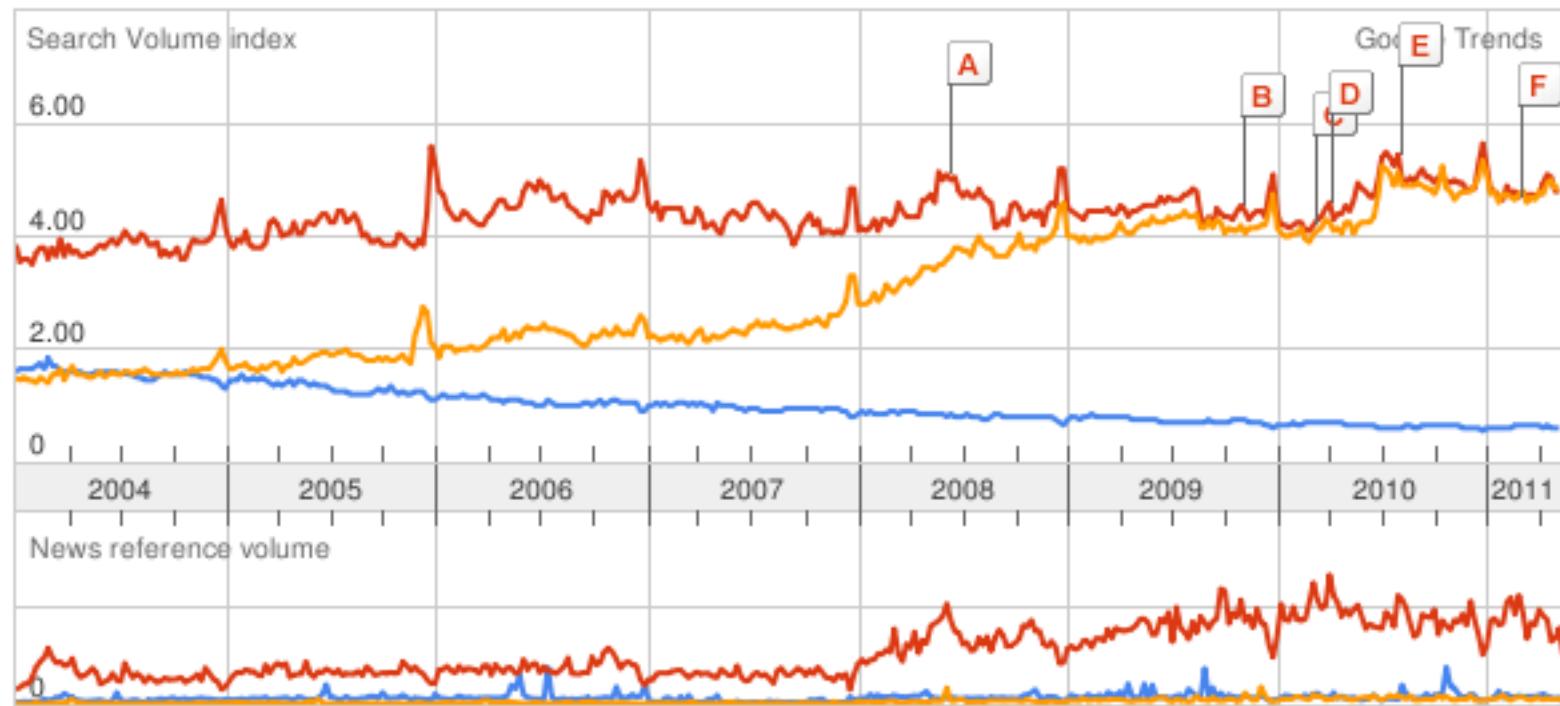
Quelle: Google Trends

# MDA, UML, XML, C#, JAVA



Quelle: Google Trends

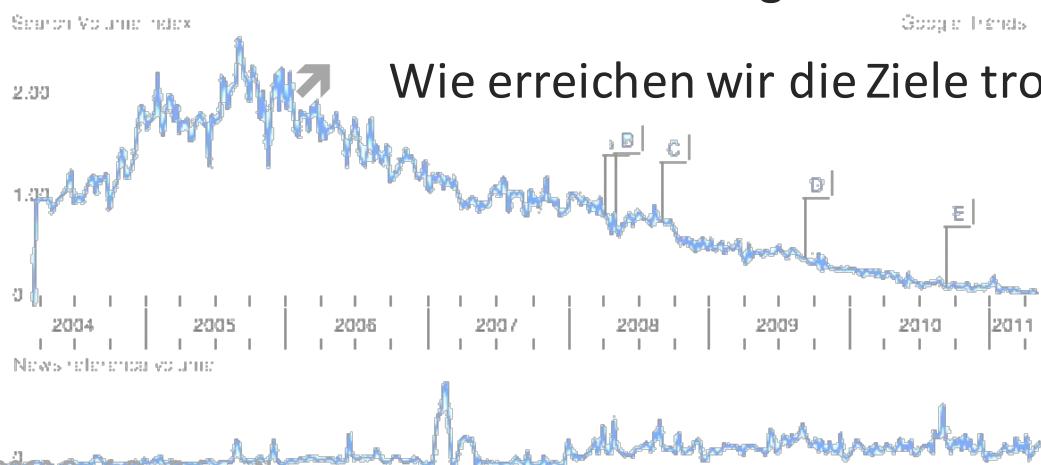
# JAVA (hier blau) im Vergleich zu...



Quelle: Google Trends

# Warum Sie trotzdem zuhören sollten...

- ↗ Welches Ziel hatte MDA?
- ↗ Techniken und Vorgehensweisen
- ↗ Alter Wein in neuen Schläuchen?
- ↗ MDA im praktischen Einsatz
- ↗ Warum MDA ausgestorben ist



Wie erreichen wir die Ziele trotzdem?

# Teil 1 – Welches Ziel hatte MDA?

# Probleme bei der Softwareentwicklung

Softwareprojekte enden selten erfolgreich

- ↗ Rechtzeitig
- ↗ Kostengerecht
- ↗ Mit gewünschter Funktionalität

# Ursachen

- ↗ Falsche Vorgehensweise
  - ↗ Falsche oder fehlende Pflichtenhefte
  - ↗ Über- und Unterspezifikation
  - ↗ Unklare Ziele
- ↗ Falsche Schätzungen
- ↗ Politische Preise

- ↗ Beherrschung zunehmender Komplexität
  - ↗ Zunehmende Komplexität der Technologie
  - ↗ Technologischer Wandel
  - ↗ Neue Paradigmen und Alter Wein in neuen Schläuchen
  - ↗ Heterogenität
- ↗ Zunehmende Komplexität der Funktionalität
  - ↗ Funktionsumfang
  - ↗ Integration von Diensten und Daten

# Softwareentwicklungsprozess

## Schwergewichtige Prozesse

- ↗ Fokus auf „geschlossene“ Durchführung
  - ↗ Wasserfall
  - ↗ V-Modell
  - ↗ (R)UP
- ↗ Probleme
  - ↗ Administrativer Overhead
  - ↗ Fokus auf Spezifikation und nicht auf Interaktion  
(Beidseitige Versicherung)
  - ↗ Löst nicht das Problem der falschen Funktionalität

## Fokus auf Interaktion mit Kunden

- ↗ Iterativ-inkrementelle Vorgehensweisen
- ↗ Rapid Prototyping
- ↗ eXtreme Programming
- ↗ Probleme
  - ↗ Beidseitige Unsicherheit  
(erfordert gute Vertrauensbasis)
  - ↗ Komplexität der Technologie verhindert kurze Zyklen
- ↗ **Ziel: Beherrschung der Komplexität!**

## Leichtgewichtige (agile) Prozesse

# Technische Komplexität (Techniken)

- ↗ Sprachen  
(Java, C++, C#, VB, Objective-C ...)
- ↗ Dokumenten- und Zeichenformate  
(CSV, XML, EDIFOR, Unicode, ...)
- ↗ Frameworks und Bibliotheken  
(J2EE, .NET, LEDA, Axis, JSF, ASP, ...)
- ↗ Datenbanken  
(MS-SQL-Server, Oracle, DB2, Cloudscape, SAP DB, mySQL...)
- ↗ Betriebssysteme  
(Windows, Linux, Mac-OS, iOS, Android...)
- ↗ Protokolle  
(TCP/IP, HTTP, WAP, SOAP, UDDI,...)
- ↗ Sonstiges  
(WfMS, Ontologies, Rule-Engines, ...)
- ↗ Heterogenität

# Technische Komplexität (Konzepte)

## ↗ Architekturen

- ↗ Mehrschichtige Architekturen
- ↗ Eingebettete Systeme
- ↗ Agentenorientierte Ansätze
- ↗ Verteilte Systeme
- ↗ Mobile Systeme
- ↗ Integrationsplattformen
- ↗ Optimierer und Übersetzer
- ↗ Compilers

## ↗ Paradigmen

- ↗ Objektorientierung
- ↗ Aspektorientierung
- ↗ Subjektorientierung
- ↗ Adaptive Programmierung
- ↗ Serviceorientierung
- ↗ Generative Programmierung

# Fachliche Komplexität

- ↗ Komplexe Anwendungen
- ↗ Gemischte und überlappende Domänen
- ↗ Übergreifende Lösungen
  - ↗ Betriebliche Abläufe
  - ↗ Integration mobiler und verteilter Dienste
  - ↗ Portale

# Konsequenzen

- ↗ Andauernde Ausbildungslücke
  - ↗ Fehlende Fähigkeiten
  - ↗ Fehlende Erfahrungen
  - ↗ Immense Ausbildungskosten
  - ↗ Technik verhindert fachliche Lösung
  - ↗ Langlaufende Entwicklung
- ↗ Qualitätsprobleme
  - ↗ Instabilität
  - ↗ Inperformance
  - ↗ Mangelhafte Qualitätssicherung
- ↗ Andauernde Frustration auf allen Seiten!

# Aktuelle Situation?

Gute Prozessmodelle helfen beim Durchführen und Managen von Projekten, in denen unzureichend ausgebildete, technologisch unerfahrene und entsprechend frustrierte Entwickler mit der wandelnden Technologie nur schwer mithalten können und sich deshalb zu wenig um die termin-, kosten- und funktionsgerechte Bereitstellung von Systemen kümmern können!

# Lösungsansatz Beherrschen der Technik

- ↗ Modularisierung
  - ↗ Divide et conquere
  - ↗ Beherrschung umfangreicher Systeme
- ↗ Wiederverwendung
  - ↗ Das Rad nicht neu erfinden
- ↗ Abstraktion
  - ↗ Von Techniken abstrahieren
  - ↗ Auf fachlicher Ebene entwickeln
- ↗ Programmieren sollte wieder so einfach sein wie in den 80ern!

# Wiederverwendung

- ↗ Informelles Wissen
  - ↗ Anleitungen
  - ↗ Entwurfs- und Architekturmuster
- ↗ Formalisierbares Wissen
  - ↗ Codefragmente
  - ↗ Vorlagen
- ↗ Code
  - ↗ Frameworks
  - ↗ Bibliotheken

# Kernziel

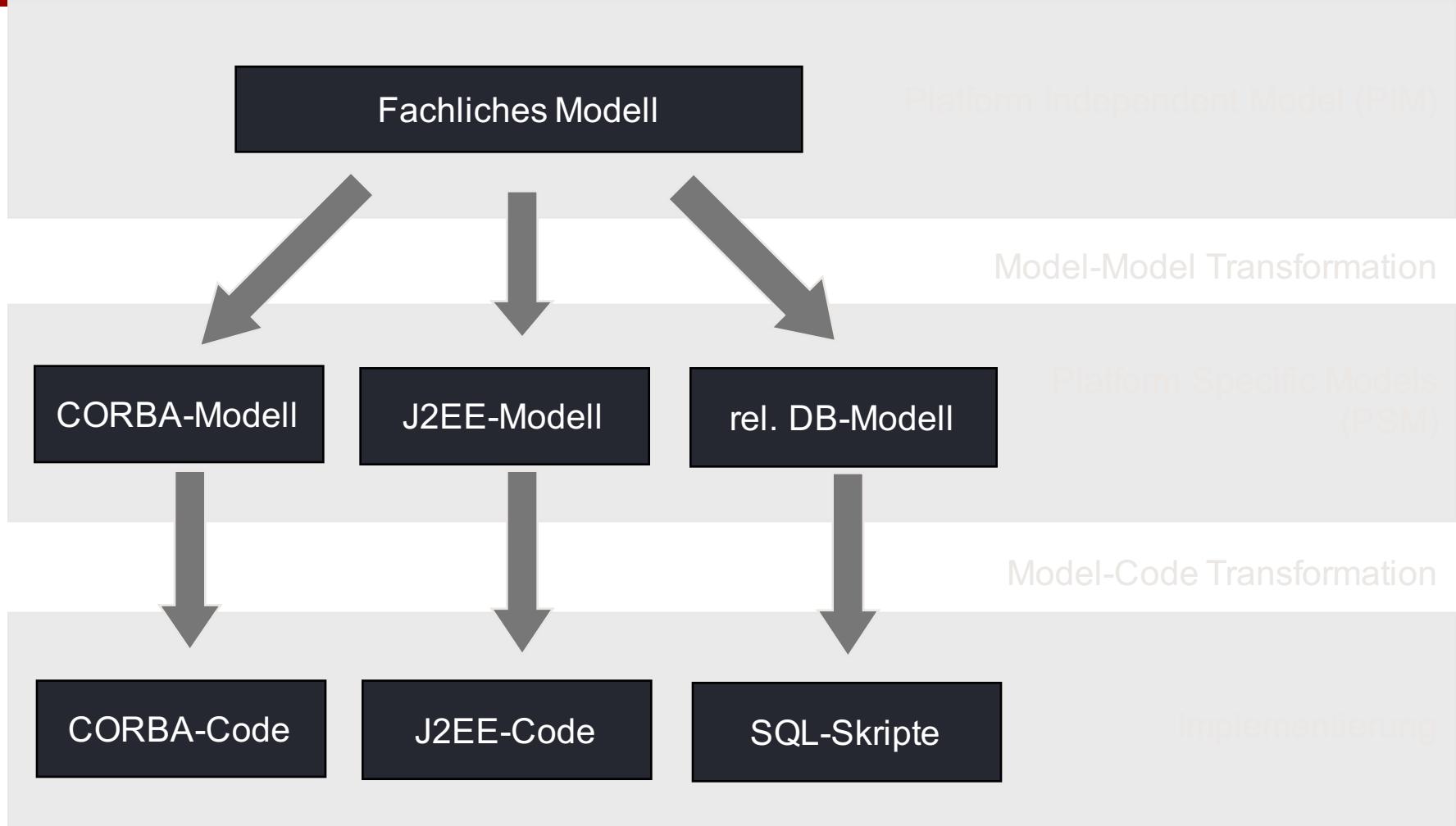
- ↗ Steigerung der Effizienz bei der Softwareentwicklung um eine Größenordnung durch Überwindung technologischer Komplexität

# Teil 2 – Techniken und Vorgehensweisen

# Der MDA-Ansatz

- ↗ Modellierung auf fachlicher Ebene
  - ↗ **Abstraktion** von Technik
- ↗ Transformation auf technische Ebene
  - ↗ **Wiederverwendung** von Architektur und Entwurfsmustern
  - ↗ Systematisierung des Codes
- ↗ Programmierung
  - ↗ Mit Fokus auf fachliche Probleme
  - ↗ Im vereinfachten technologischen Umfeld

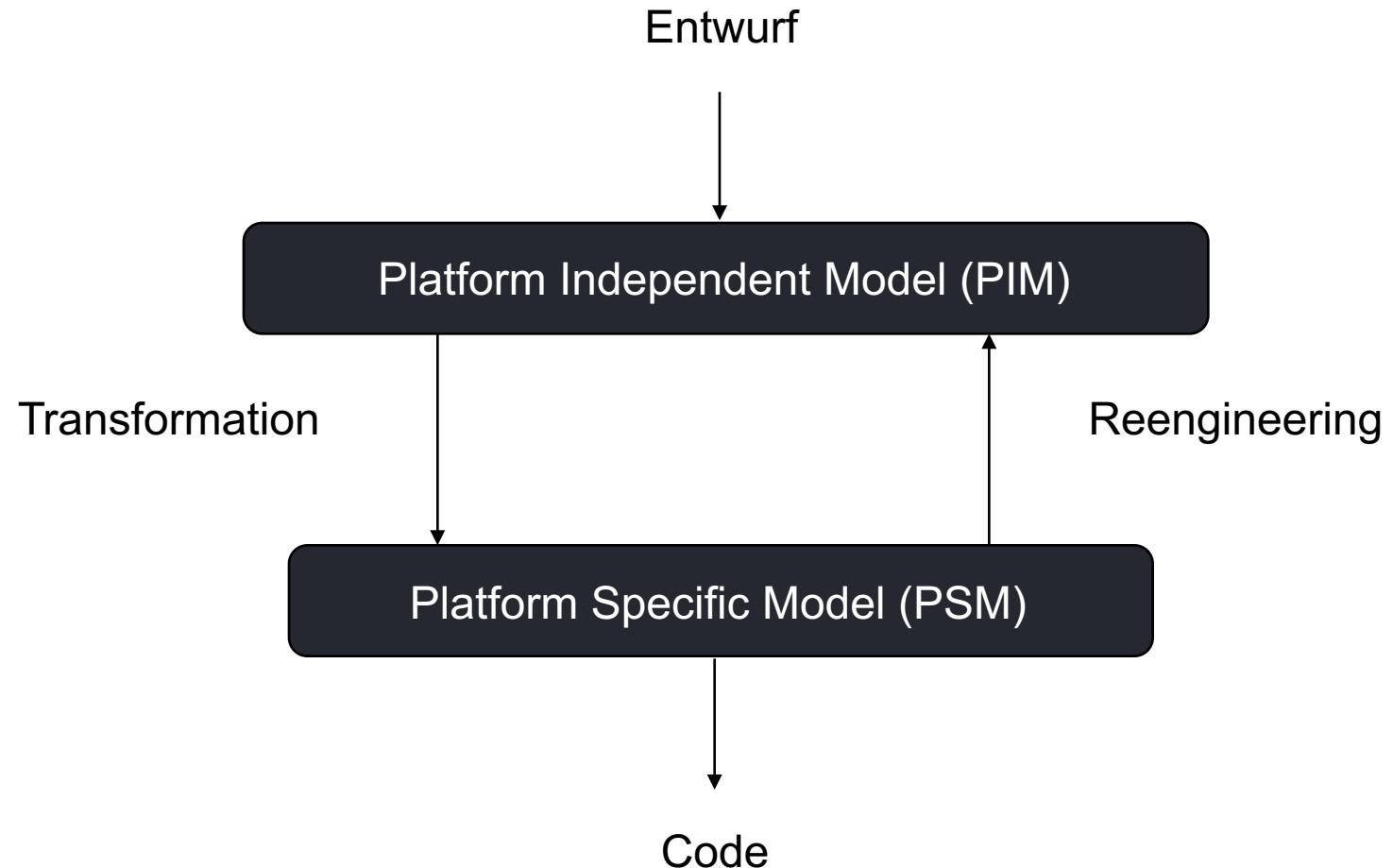
# MDA – Model Driven Architecture



# MDA – Grundbegriffe

- ↗ *Computation Independent Model (CIM)*
  - ↗ Analysemodell (informell)
  - ↗ Beispiel: Business Objekte und Business Prozesse
- ↗ *Platform Independent Model (PIM)*
  - ↗ Entwurfsmodell
  - ↗ Abstraktion von Technologie
  - ↗ Beispiel: 3-schichtige Architektur
- ↗ *Platform Specific Model (PSM)*
  - ↗ Implementierungsmodell
  - ↗ Beispiel: J2EE

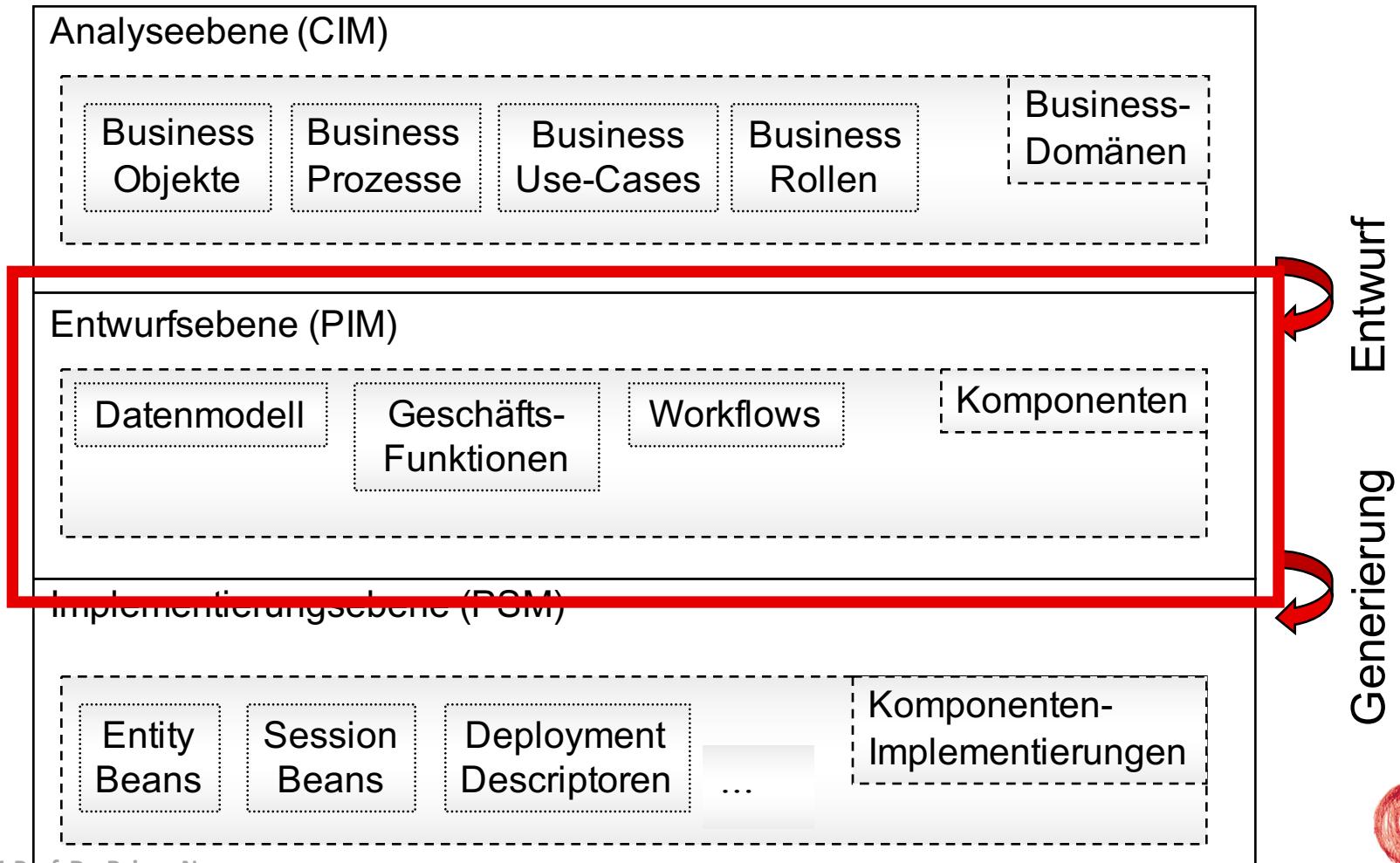
# MDA – Model Driven Architecture



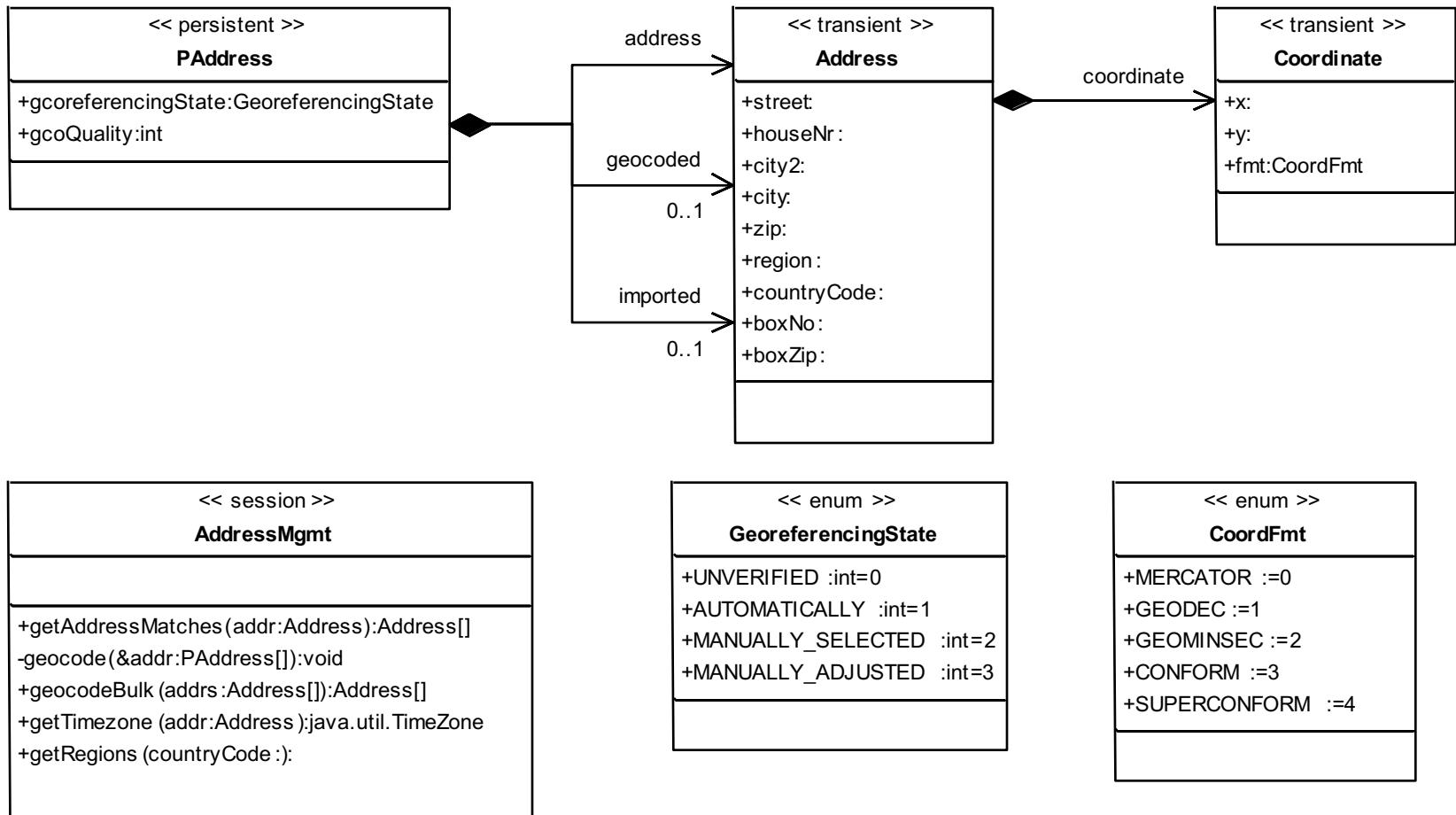
# MDA Grundbegriffe

- ↗ Bei MDA werden **Modelle**, die unterschiedliche **Plattformen** beschreiben, durch **Transformation** ineinander überführt.
- ↗ Die Modelle der Plattformen werden dabei durch Metamodelle (so genannte **Profile**) beschrieben.

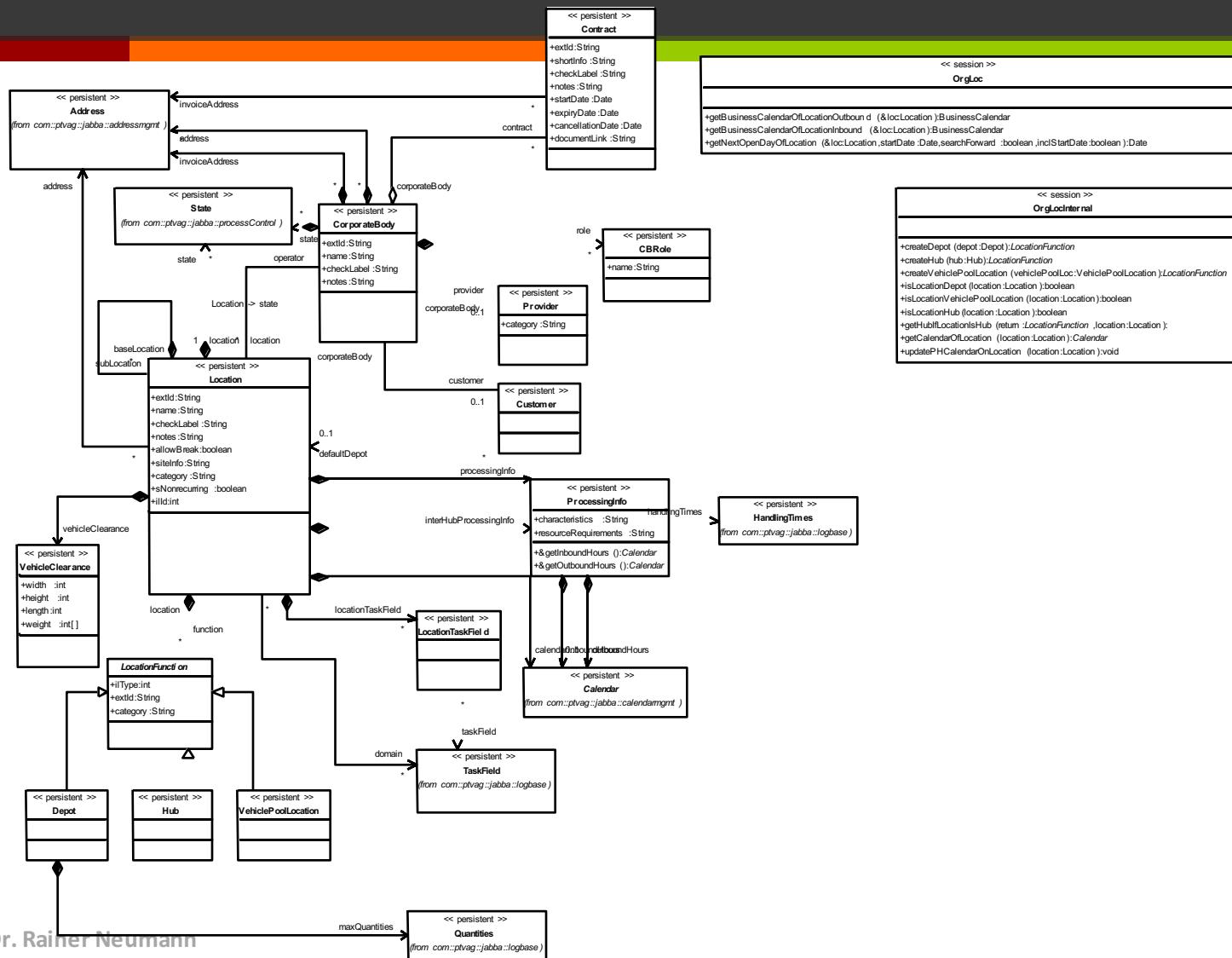
# Ein MDA Beispiel



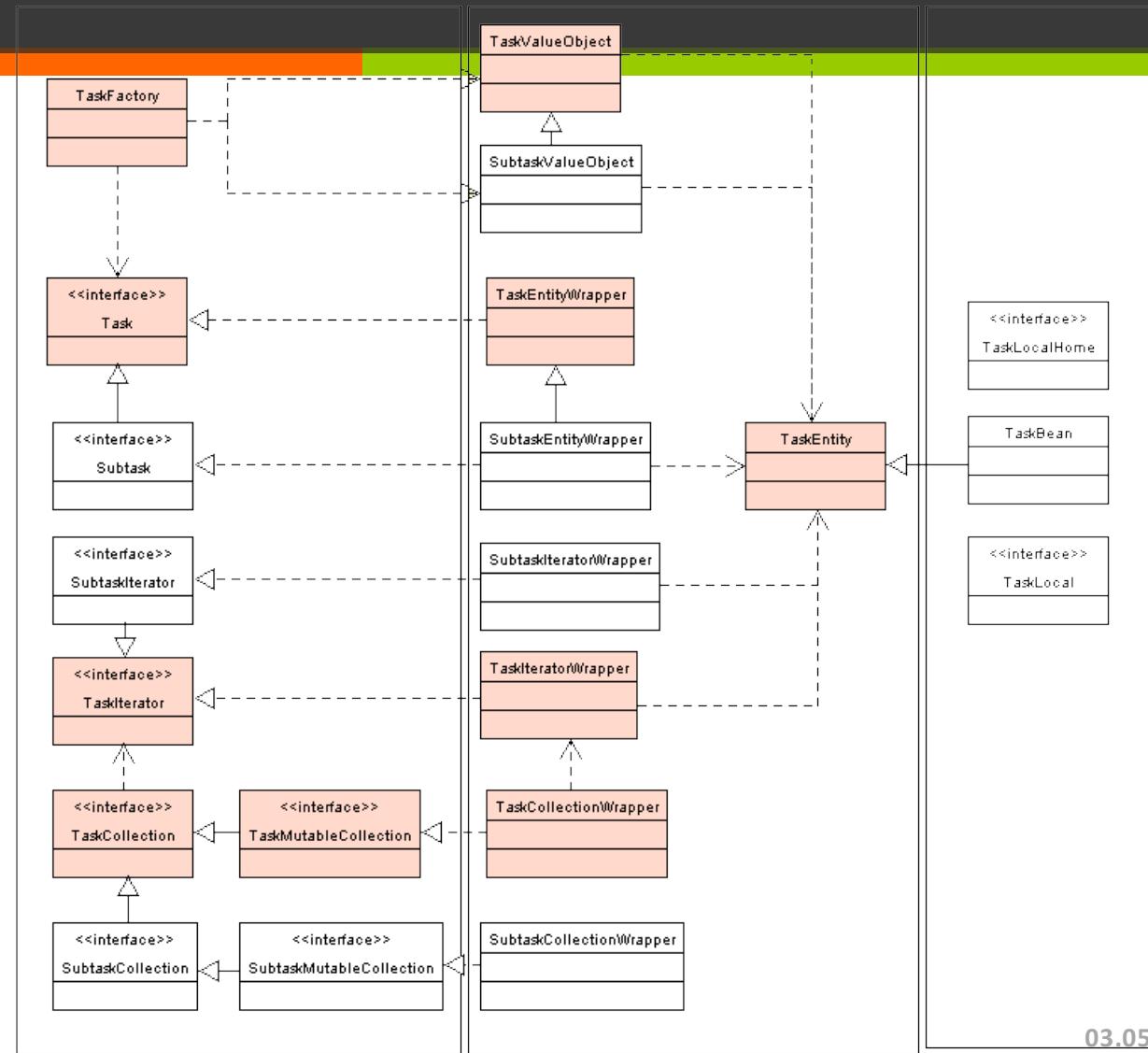
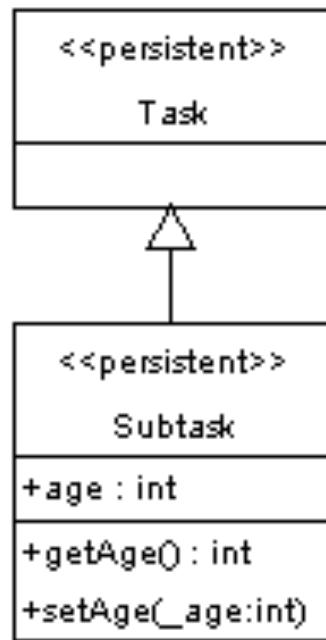
# Ein Beispielmodell im PSM



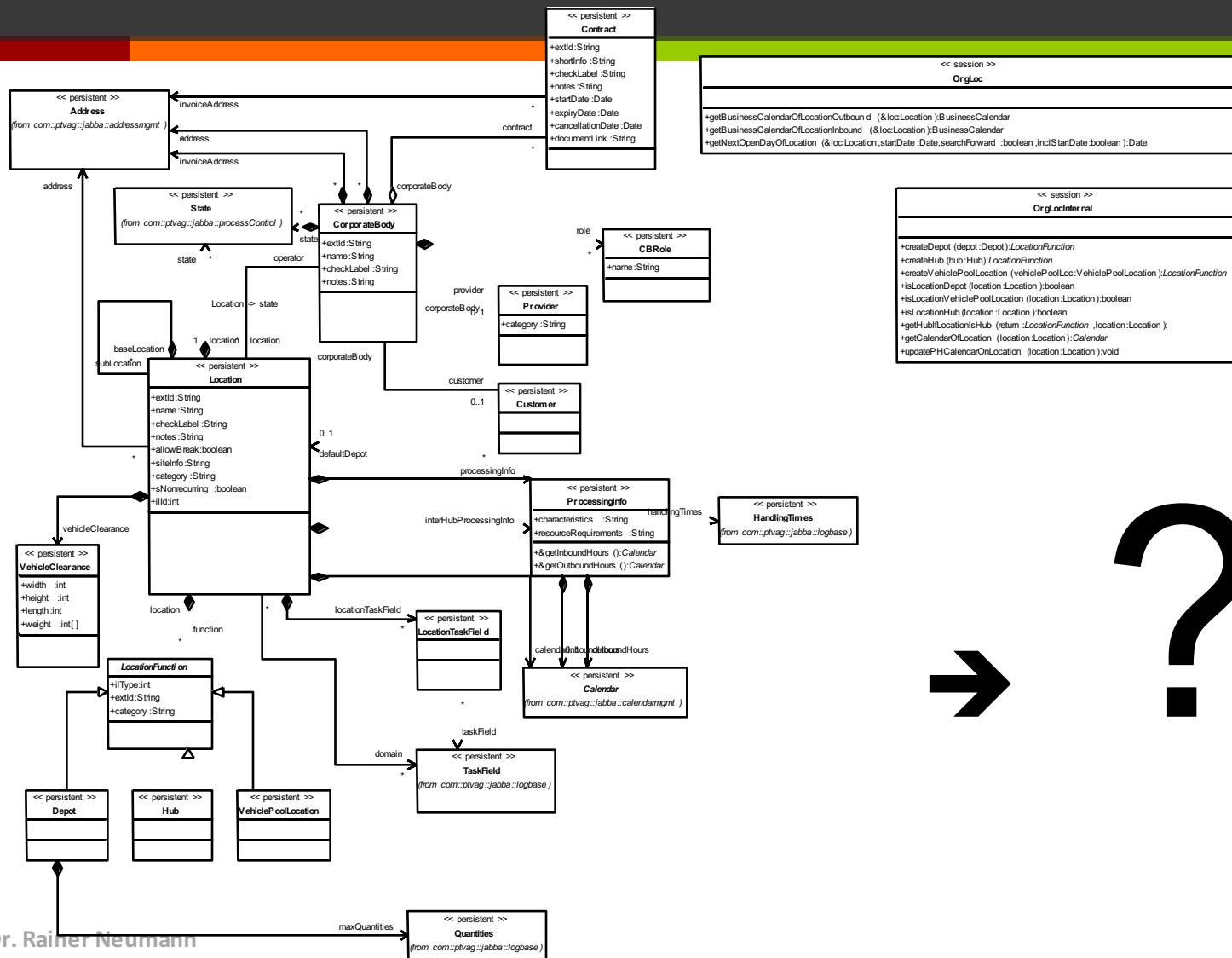
# Ein weiteres Beispielmodell im PSM



# Beispiel für ein Umsetzungsmuster



# Und am Beispiel?



# MDA – Kurz und Bündig

- Abstraktion von Technologie
- Fokussierung auf
  - Fachlichkeit (Modellierung im PIM)
  - Architektur (Transformation in das PSM)

# Teil 3 – Alter Wein in neuen Schläuchen?

# Literate Programming

## ↗ Grundidee

- ↗ Entwicklung von Algorithmen auf Basis zweier konsistenten Programmsichten
- ↗ Textuell
  - ↗ Abstrakte Beschreibung (näher am Problem)
  - ↗ Selbsterklärendes Programm
- ↗ Programmiersprachlich
  - ↗ Sematisch präzise
  - ↗ Übersetzbare und Ausführbar

## ↗ Beispiel: Initiale TeX-Implementierung

# Domänenspezifische Sprachen

## ↗ Beobachtung

- ↗ Allgemeine Programmiersprachen sind in spezifischen Domänen unnötig flexibel und dadurch wenig fokussiert

## ↗ Grundidee

- ↗ Beschreibung eines Problems = Lösung des Problems
- ↗ Weiterentwicklung des „Literate Programming“

## ↗ Beispiele

- ↗ Datenmodellierungssprachen
- ↗ Koordinationssprachen (z.B. Web Service Orchestration)
- ↗ Architektur- und Datenflusssprachen
- ↗ Seitenbeschreibungssprachen (ASP, JSP, ...)
- ↗ Übersetzerbausprachen
- ↗ Konfigurationssprachen (auch wenn sie scheinbar XML heißen)

# Aspektorientierte Programmierung (AOP)

- ↗ Grundidee
  - ↗ Unterschiedliche Aspekte eines Systems lassen sich unterschiedlich Beschreiben
  - ↗ Aspekte lassen sich geeignet miteinander verweben (Aspect weaving)
  - ↗ Kombination verschiedener Domänenpezifischer Sprachen
- ↗ Beispiele für Aspekte
  - ↗ Verteilung
  - ↗ Sicherheit
  - ↗ Logging und Tracing
  - ↗ ...

# Grafische Programmierung

- ↗ Mensch ist visuell orientiert
- ↗ Systementwicklung im Großen lässt sich durch visuelle Formalismen erleichtern
- ↗ Kommunikation anhand von Bildern ist einfacher
- ↗ Nur wer ein Bild malen kann, hat ein Problem verstanden!
- ↗ Beispiele
- ↗ Pläne und Zeichnungen in Architektur, Maschinenbau, Elektrotechnik, ...
- ↗ ER-Diagrammen
- ↗ Modellierung endlicher Automaten
- ↗ Datenfluss- und Kontrollflussgraphen
- ↗ Geschäftsprozessmodellierung mit ARIS/BPMN
- ↗ UML

# Grafische Programmierung

- ↗ Geeignet für
  - ↗ Einfache Abstraktion
  - ↗ Entwicklung von Zusammenhängen
- ↗ Problematisch
  - ↗ Programmieren im Kleinen
    - Kombination mit anderem Programmiermodell
  - ↗ Varianz in der Abstraktion
    - Erfordert Parametrisierung

# Generative Programmierung

- ↗ Allgemeine Bezeichnung für Erzeugung von Code aus Spezifikationen auf höherer Abstraktionsebene
- ↗ Implizite Generierung
  - ↗ Compiler umfasst Generator
  - ↗ Beispiele: Programmiersprachenkonstrukte
    - ↗ Templates / Generische Klassen
    - ↗ Konstrukte domänenspezifischer Sprachen
- ↗ Explizite Generierung
  - ↗ Transformation auf niedrigere Spezifikation
  - ↗ Generator von Programmiersprache(n) getrennt

# Was wir schon immer wussten...

- ↗ Abstrahierende/generative Ansätze...
  - ↗ ... sind nicht neu
  - ↗ ... sind praktikabel und im Einsatz
  - ↗ ... sind oft „besser“ als manuelles Programmieren
  - ↗ ... verbessern die Systemqualität
  - ↗ ... sind kostengünstig(?)
- ↗ aber sie erfordern...
  - ↗ ... Architekturelle Expertise
  - ↗ ... Domänenexpertise
  - ↗ ... Übersetzerbauwissen

# MDA – Zurück in die Zukunft

- ↗ MDA vereinigt die Vorteile verschiedener Ansätze
  - ↗ Visuelle Formalismen zum Programmieren im Großen
  - ↗ Parallelе Aspekte eines Systems
- ↗ Einheitliche „Programmiersprache“
  - ↗ ermöglicht generischen Übersetzerrahmen
  - ↗ Reduziert notwendiges Übersetzerbauwissen
- ↗ Entwicklung erfolgt auf mehreren Abstraktionsebenen
  - ↗ unterschiedliche Plattformen
- ↗ Flexibles Metamodell
  - ↗ Einfache Erstellung domänenspezifischer Sprachen

# Zusammenfassung

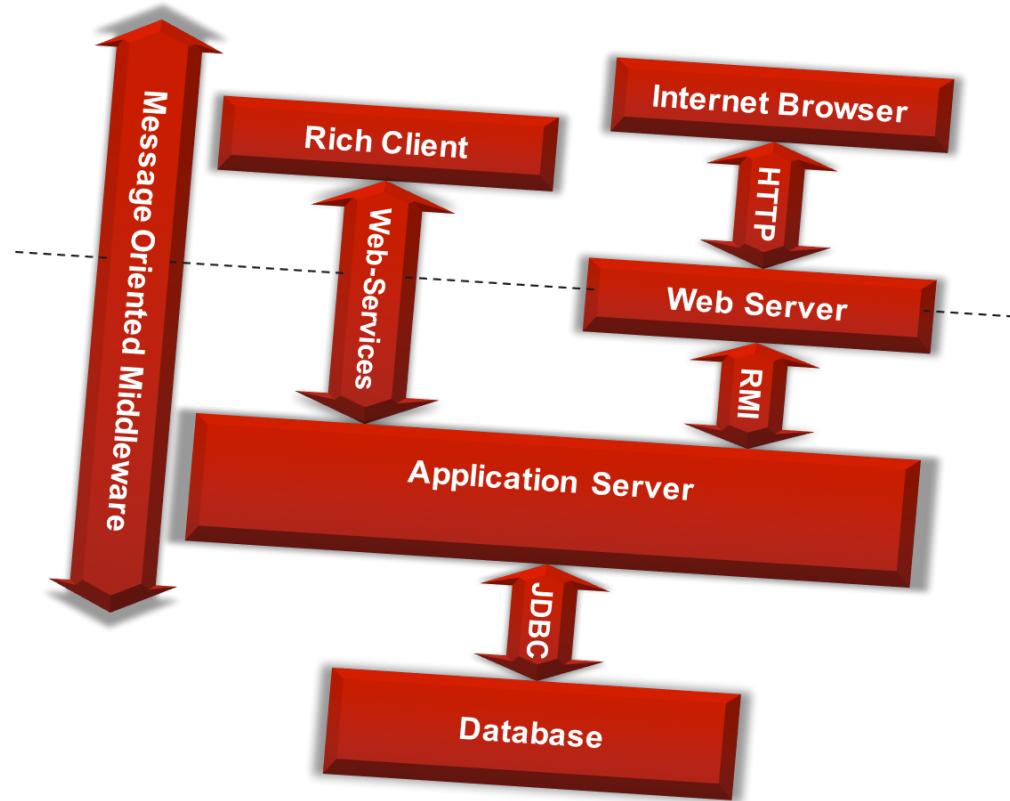
MDA ist...

- ↗ Generatives,
  - ↗ Aspektorientiertes,
  - ↗ Visuelles Programmieren in einer
  - ↗ Domänenspezifischen Sprache
- 
- ↗ ...und so einfach, dass es jeder nutzen kann!

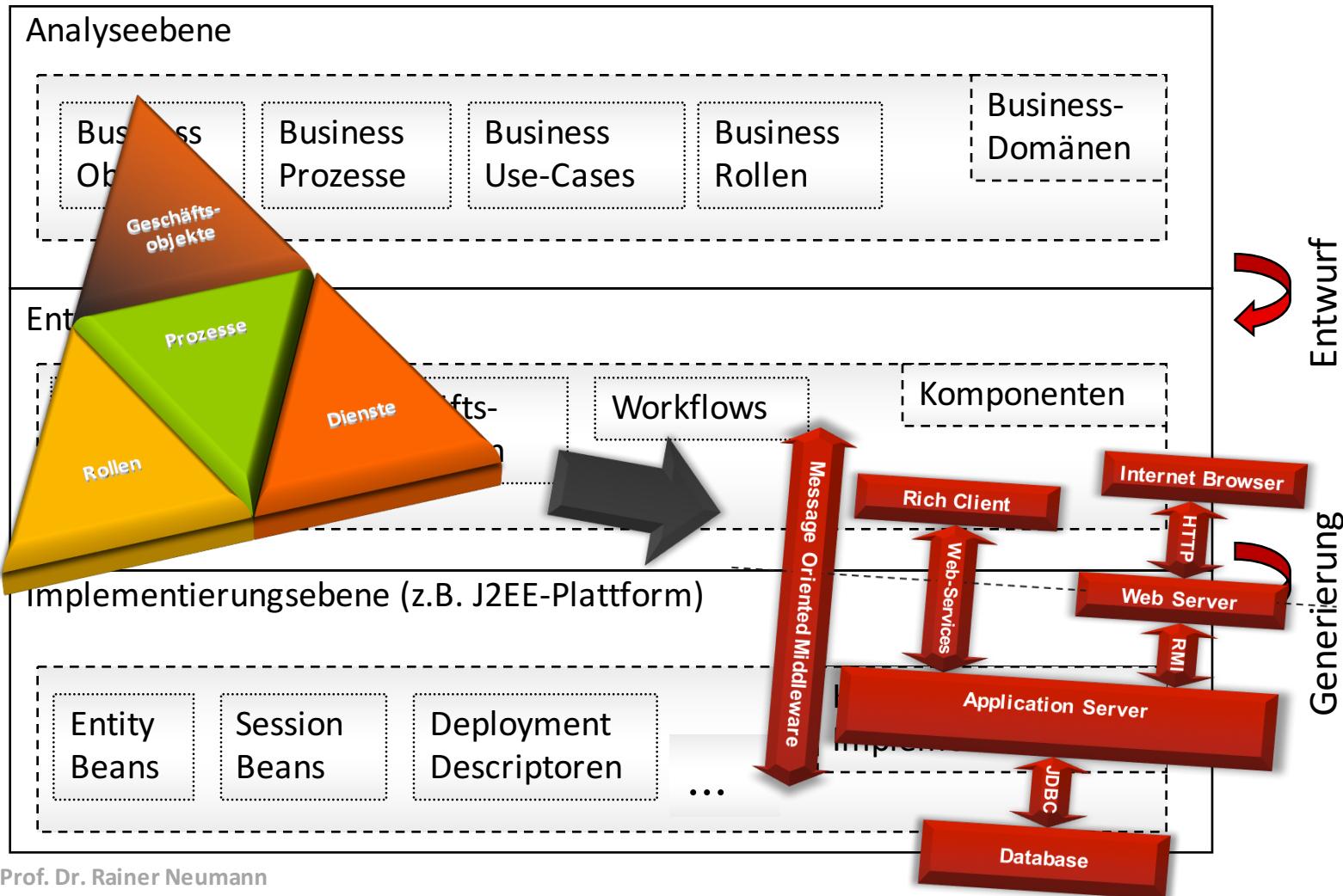
# Teil 4 – MDA im praktischen Einsatz

# Modellbasierte Entwicklung am Beispiel

- ↗ Anwendungsstudie: Planung von Sicherheitstransporten
  - ↗ Auftragsimport und -erfassung (täglich ca. 2000 Aufträge)
  - ↗ Transportbildung
  - ↗ Kommissionierung
  - ↗ Transportüberwachung
  - ↗ Call-Center-Lösung
  - ↗ Web-Portal für Kunden
- ↗ Technologien
  - ↗ J2EE App-Server
  - ↗ Oracle-DB
  - ↗ Web-GUI für Auftragsverwaltung
  - ↗ .NET-GUI für Planung
  - ↗ Kopplung zu mobilen Endgeräten
- ↗ Rahmenbedingungen
  - ↗ Laufzeit ca. 1 ½ Jahre
  - ↗ Team aus 5 Personen
  - ↗ Mittleres Leistungsniveau



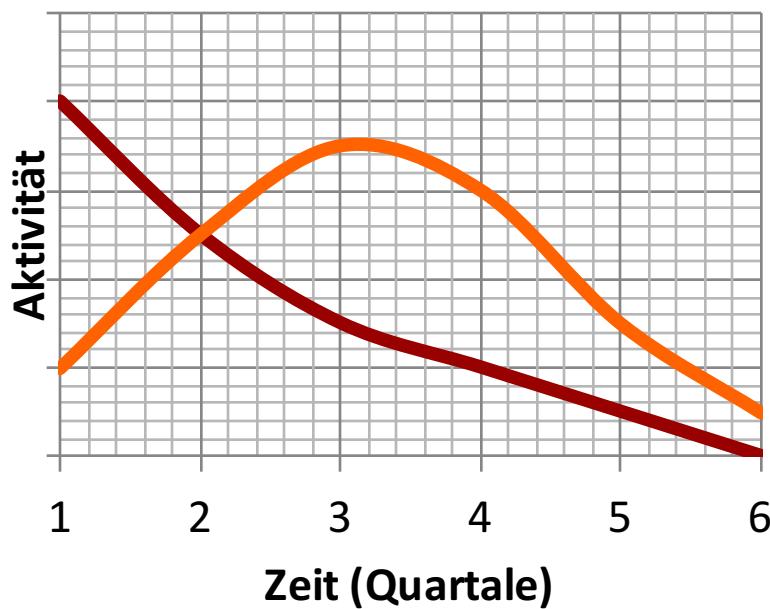
# Spezifikation und Generierung in der Fallstudie



# Prozessaspekte

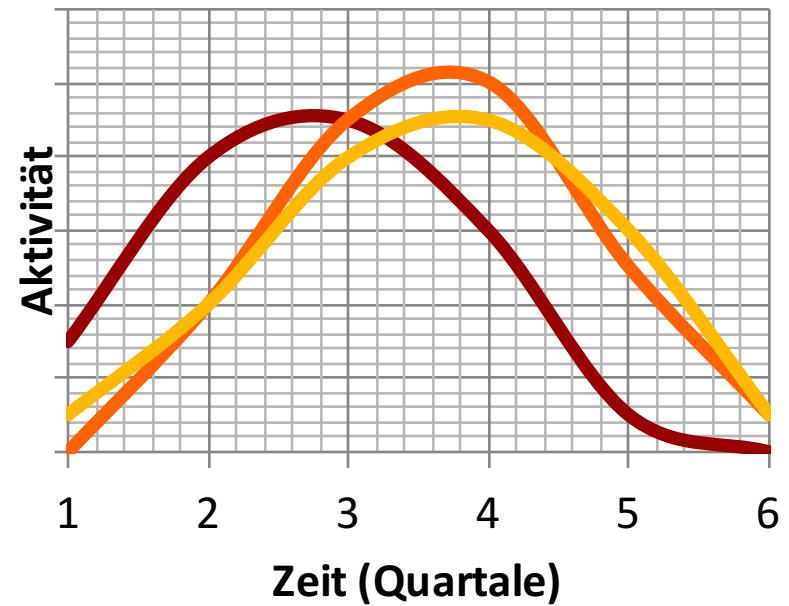
## Entwicklungsschwerpunkte

- Werkzeugentwicklung
- Anwendungsentwicklung

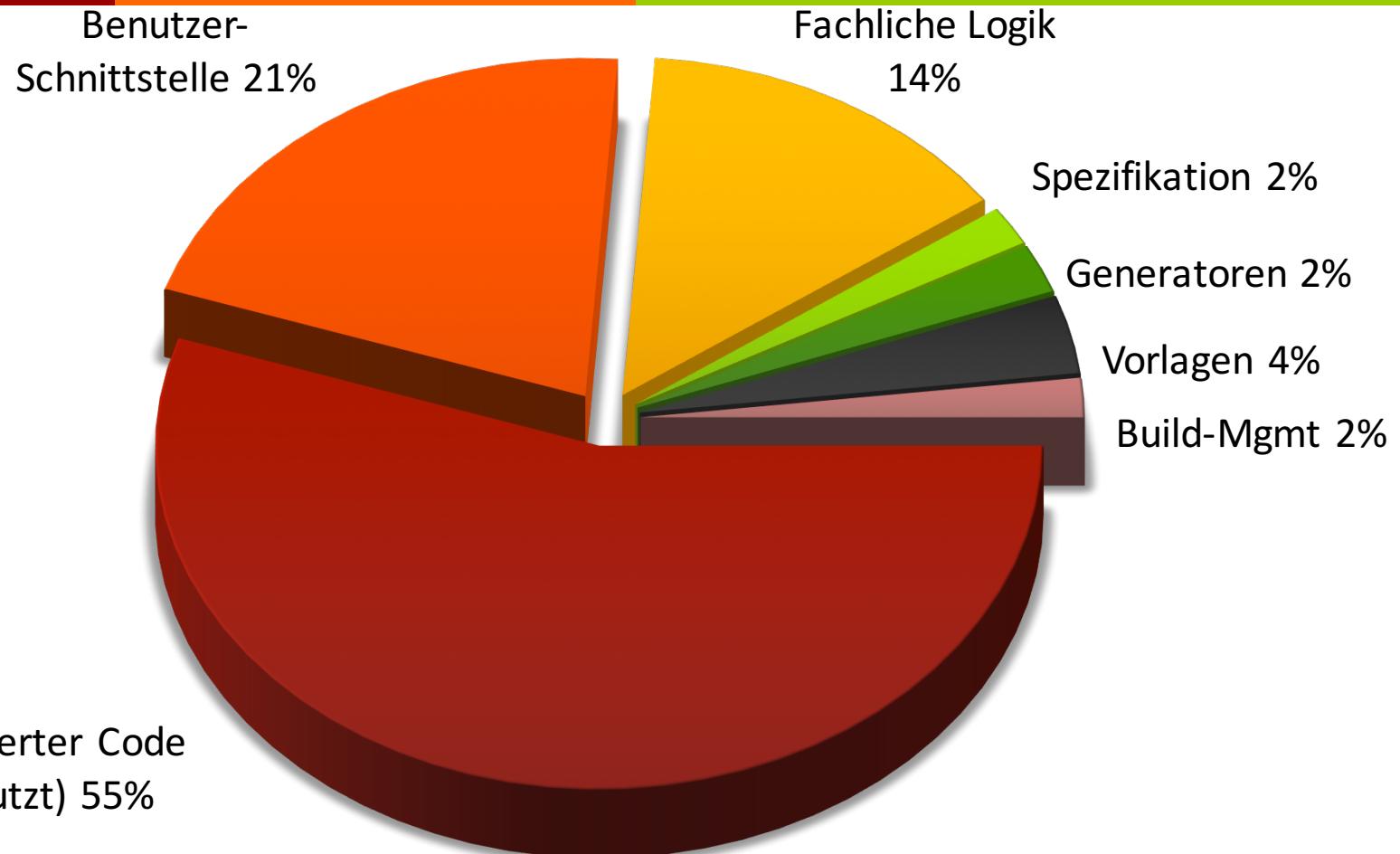


## Entwicklungsfookus

- Spezifikation
- Business-Logik
- GUI



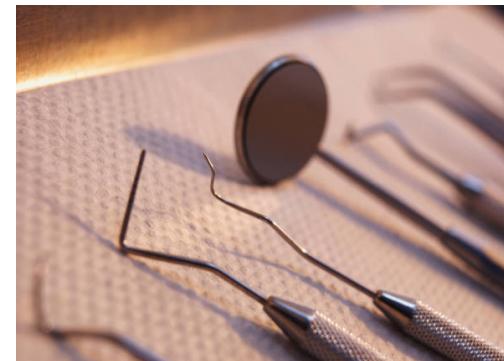
# Quantifizierung anhand messbarer Codebestandteile



# Teil 5 – Warum MDA ausgestorben ist

# Probleme des klassischen MDA-Ansatzes

- ↗ Fokussierung auf UML
- ↗ Detaillierung in der Abstraktion
- ↗ Problemfeld Versionierung
- ↗ Beherrschen der Buildprozesse



# Problemfeld: Fokussierung auf UML

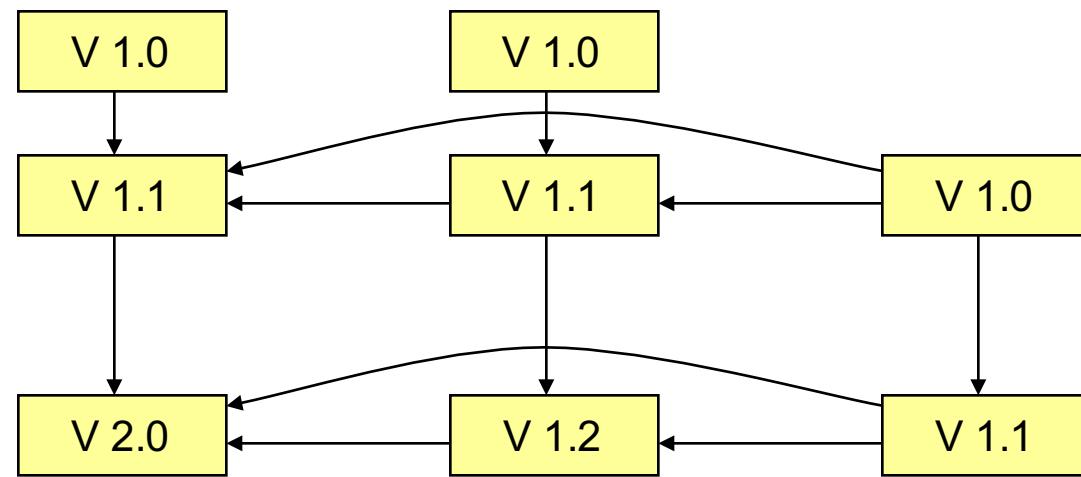
- ↗ Problem
  - ↗ Bilder sind gut für Abstraktionen, aber schlecht für Details
- ↗ Konsequenzen
  - ↗ „Da schreib‘ ich’s doch gleich lieber in Java...“
  - ↗ Zitat: „Können wir die alte XML-Spezifikation wieder haben?“
- ↗ Heutige „MDSD“-Werkzeuge nutzen viele Quellformate (UML/XMI, XMLs, Java, ...)

# Problemfeld: Detaillierung in der Abstraktion

- ↗ Probleme und Lösungen müssen auf einem jeweils passenden Niveau einfach und effizient beschreibbar sein
- ↗ Aber: Mehrere Quellen = Mehr Komplexität

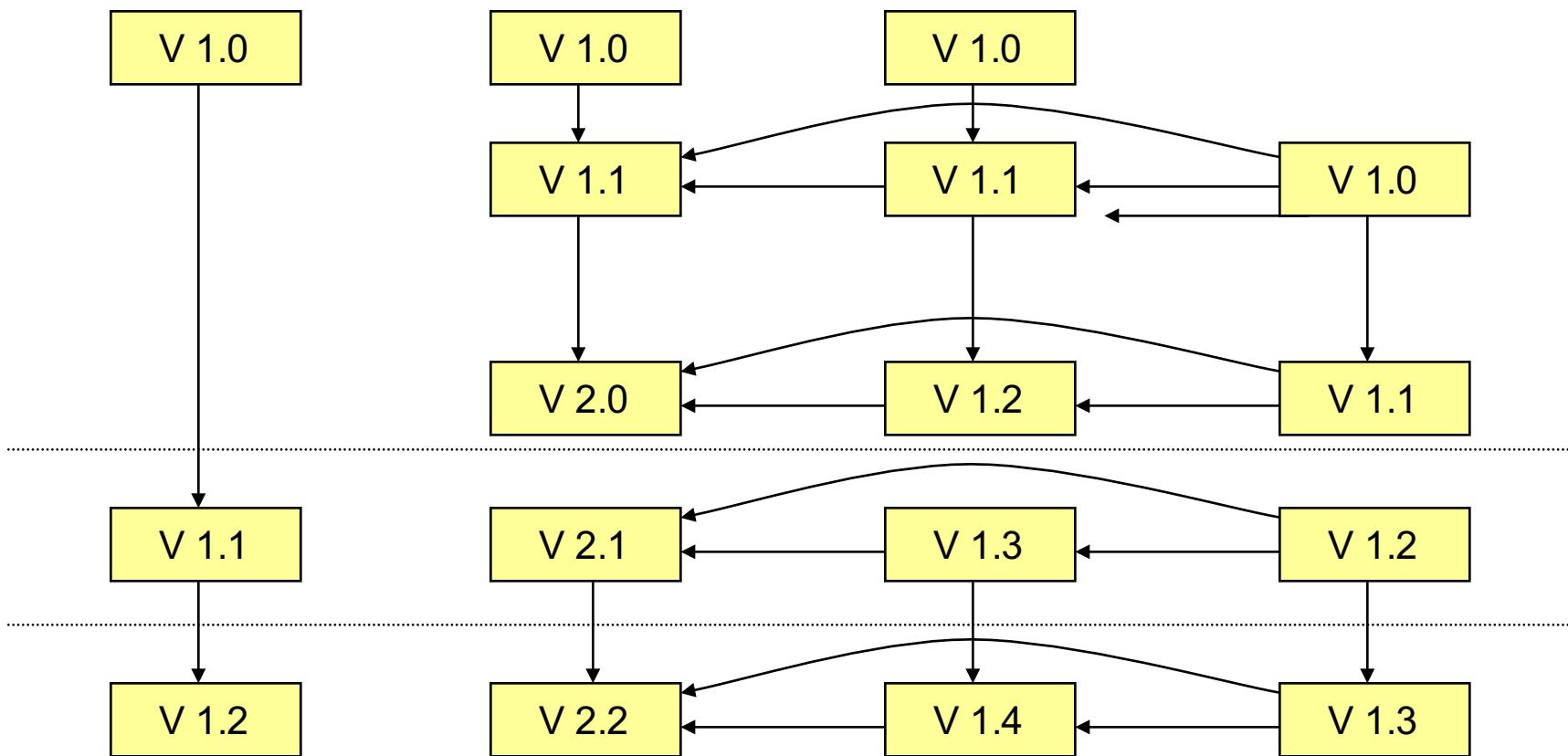
# Problemfeld Versionierung (ohne MDA)

Komponente 1      Komponente 2      Komponente 3



# Problemfeld Versionierung (mit MDA)

MDA-Werkzeug      Komponente 1      Komponente 2      Komponente 3



# Problemfeld Software-Build-Prozesse

## Unterschiedliche Prozesse

- ↗ Architekturentwicklungsprozess (Werkzeugbau)
  - ↗ Entwicklung von MDA-Werkzeugen
  - ↗ Ergebnis: Generator / Transformator
- ↗ Intra-Modul-Prozess (Komponentenbau)
  - ↗ Auflösen der Abhängigkeiten zu referenzierten Modulen
  - ↗ Vom Komponentenmodell zum Code
  - ↗ Ergebnis: Komponente in „binärer“ Form
- ↗ Inter-Modul-Prozess (Anwendungsbau)
  - ↗ Binden von Komponenten zu Produkten
  - ↗ Verwaltung der Versionen
  - ↗ Konsistenzprüfung
  - ↗ Ergebnis: Produkt in „binärer“ Form

# Anforderungen an die Prozesse



- ↗ Architekturentwicklungsprozess (Werkzeugbau)
  - ↗ Fokus auf Regressionstests für Generatoren
    - ↗ Testmodule
    - ↗ Testanwendungen
- ↗ Intra-Modul-Prozess (Komponentenbau)
  - ↗ Fokus auf Entwicklungseffizienz
    - ↗ Round-Trip-Zyklen
    - ↗ Werkzeugintegration
- ↗ Inter-Modul-Prozess (Anwendungsbau)
  - ↗ Fokus auf Konsistenz und Deployment
    - ↗ Konsistenz verwendeter Versionen
    - ↗ Setup-Bau



# Teil 6 – und jetzt...?

# Das ursprüngliche Ziel

- ↗ Steigerung der Effizienz bei der Softwareentwicklung um eine Größenordnung durch Überwindung technologischer Komplexität



# Ansätze aus heutiger Sicht

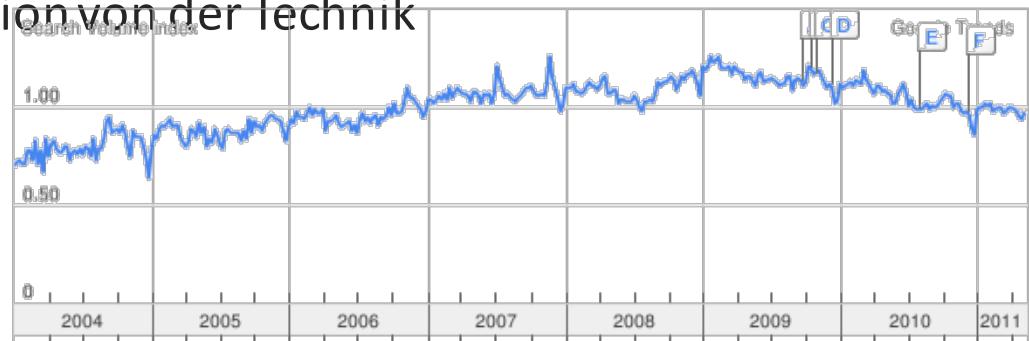
- ↗ Programmierseitig
- ↗ Modellierungsseitig



# Programmierung heute...

## ↗ Einsatz moderner Frameworks

### ↗ Abstraktion von der Technik



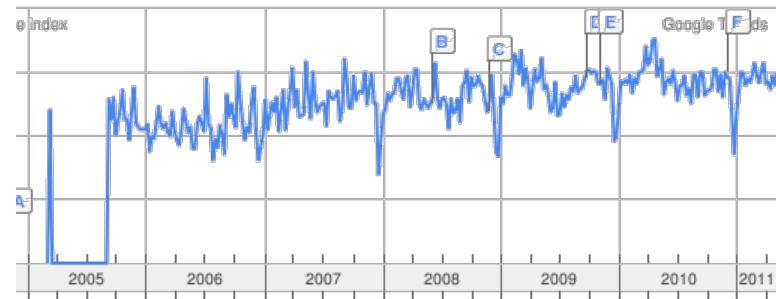
## ↗ Probleme

- ↗ Wird die Komplexität dadurch wirklich geringer?
  - ↗ Wieviele Frameworks nutzen Sie?
- ↗ Unzureichende Reife der Frameworks
  - ↗ Schnelle Ablöse
  - ↗ „Management by ct-lesen“

# Programmieren heute...

## ↗ Einsatz von Annotationen

- ↗ Deklaration im „Single-Source“
- ↗ Generierung aus dem Code heraus

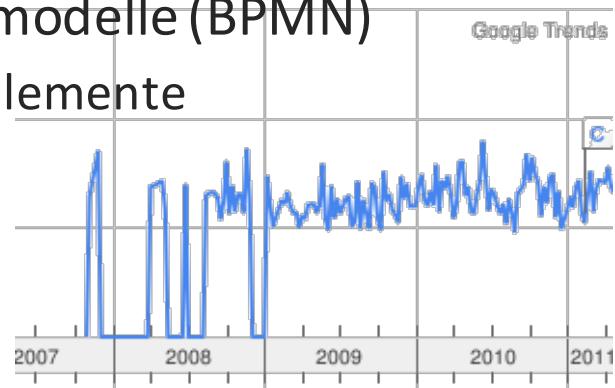


## ↗ Probleme

- ↗ Man muss die Zusammenhänge trotzdem verstehen...
- ↗ Komplexität wird nicht (immer) geringer
- ↗ kommen hier eigentlich die alten PRAGMAs wieder?

# Modellierungsseitig

- ↗ Trennung in
  - ↗ Textuelle Modelle
  - ↗ Grafische Modelle
- ↗ Verschiebung auf Prozessmodelle (BPMN)
  - ↗ Workflows als zentrale Elemente



- ↗ Programmierung schließt die Lücke zwischen Prozessmodell und Framework?

# Was ist eigentlich aus MDA geworden?

- ↗ MDA war eine Sammlung bekannter Techniken in einer interessanten Zusammenstellung
  - ↗ Kernideen waren lange bekannt
- ↗ Beherrschbare Teile sind in moderne Entwicklungsumgebungen eingeflossen
  - ↗ Annotationen zur Generierung
  - ↗ UML-Diagramme für Klassenstrukturen im Werkzeug
- ↗ Wann sehen wir das wohl wieder?
  - ↗ 1990 – Software through Pictures
  - ↗ 2005 – MDA
  - ↗ 2020 - ???

# Danke für's Zuhören

Email: Rainer.Neumann@hs-karlsruhe.de