Persistenz und Validierung aus dem UML-Modell gewinnen?

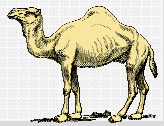
Vorstellung der Referenten

- Darko Palic (Fa. Xenovation)
  - Seit Anfang 90er in EDV-Betrieb, Automatisierung und Monitoring
  - Schwerpunkte seit 1999:
    - IT-Architekturen / technische PL in Enterprise-Projekten (Java & C#)
    - Software-Projektautomatisierung (Releasebuilds, Integrationstests, Codegenerierung)
    - Rational Team Concert (ALM-Beratung)
    - Performancetunings und proaktives Monitoring

- Michael Schulze (Fa. T-Systems)
  - Softwaretechnik-Studium an der Uni Stuttgart
  - Seit 2001 Java-Entwickler und Architekt bei T-Systems
  - Schwerpunkte: JEE-Architekturen, technische PL

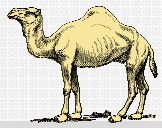
Kooperationsprojekt zwischen T-Systems und Xenovation

Ziel Generatoransatz zur Effizienzsteigerung der Softwareentwicklung



Die Agenda

- Motivation, Lösungsansatz und Ziele
- Toolchain
- Persistenzgenerierung
- Generierung von Validierungen
- Live- und Featuredemo
- Praxiserprobung
- Ausblick
- Überraschung



Motivation (1/2)

Herausforderungen

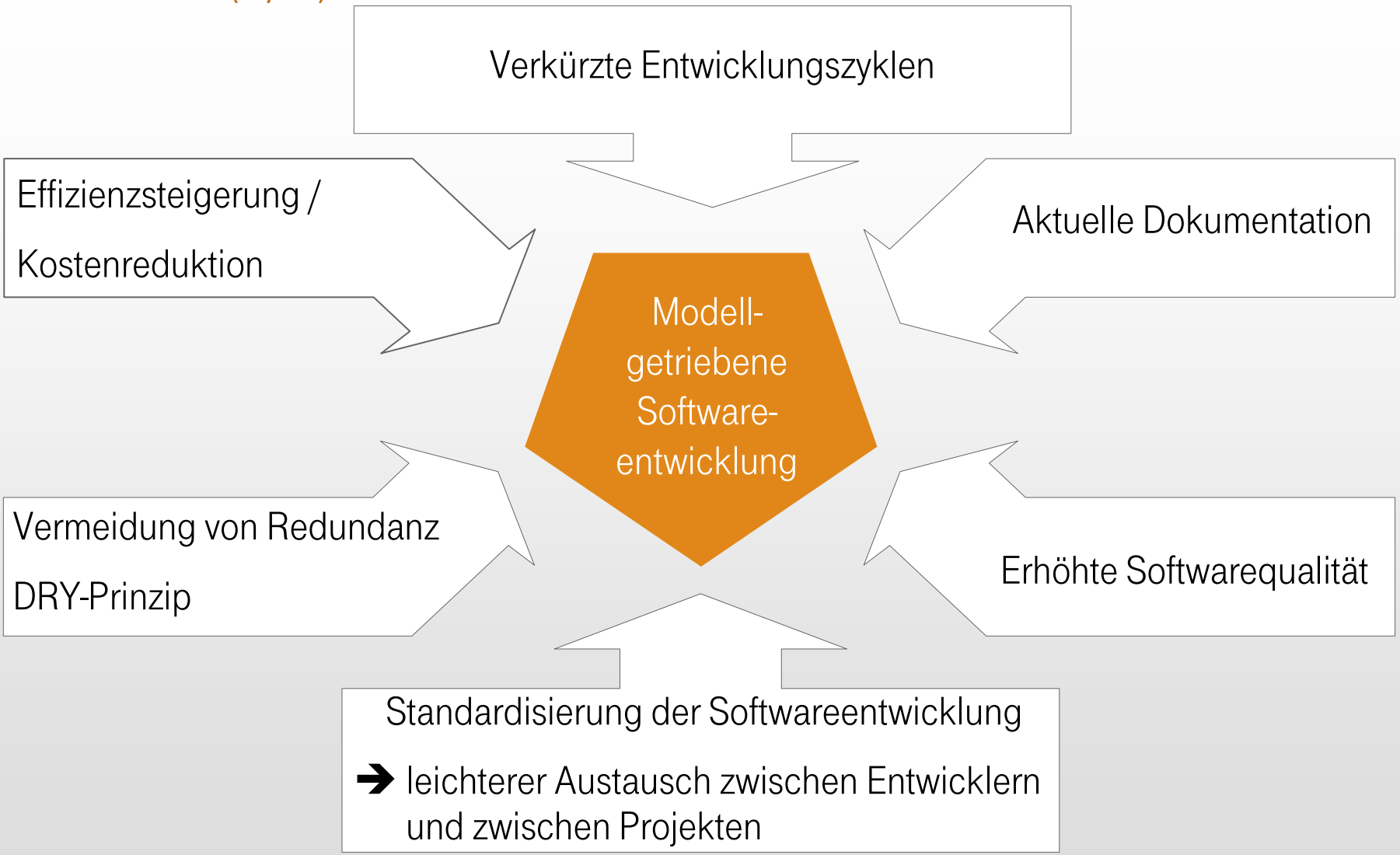
- Verkürzte Produkteinführungszeiten
- Steigender Kostendruck
- Steigende Qualitätsanforderungen
- Wiederholte Lösung von Standardproblemen
- Fehlende oder veraltete Dokumentation

Lösungsansätze

- Verlagerung der Entwicklung in Niedriglohnländer
  - Schwerpunktverlagerung Richtung Design und Architektur in Hochlohnländern
- Standardisierung
  - Etablierung von Blueprints
  - Komponenten-Vereinheitlichung,
- Prozessverbesserungen
  - XP
  - Scrum

Steigerung der Produktivität durch methodisches Vorgehen
Modellgetriebene Softwareentwicklung (MDSD)

Motivation (2/2)



Lösungsansatz und Ziele

Anforderungen

- Generatoren adressieren Standardprobleme im Softwareprojektalltag
- Sehr enge Anlehnung an entstehenden bzw. bestehenden Standards
- Einstiegshürden sehr gering halten (KISS)

Ausgrenzungen

- Keine reine MDA-Lösung
  - aus Modell entsteht keine vollständige Anwendung
  - UML soll IDE nicht ersetzen
- Keine komplette Eigenentwicklung (hohe Wiederverwendung von OpenSourceSoftware)

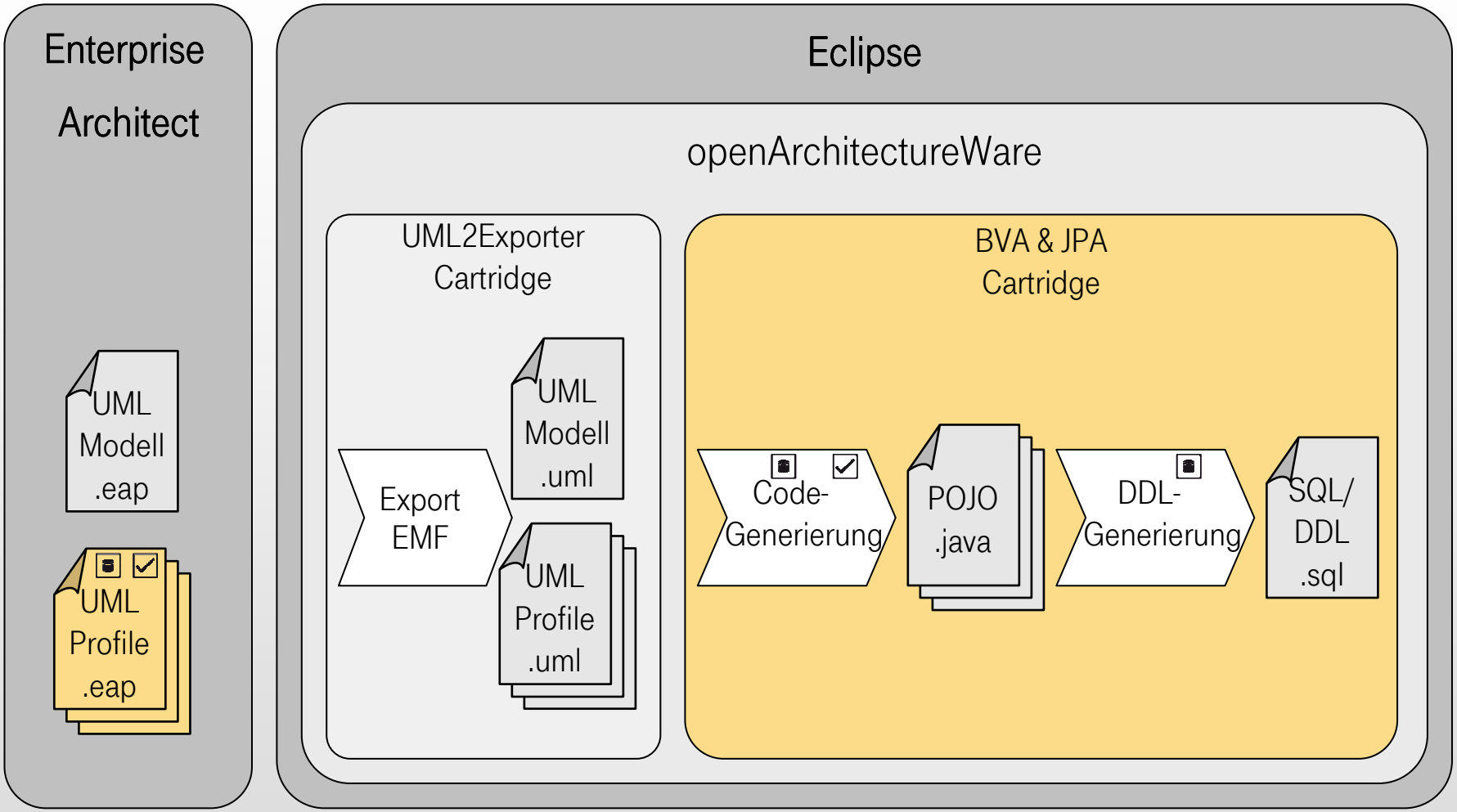
Lösungsansatz

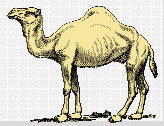
- Vorhergehende Marktanalyse verschiedener Generatorframeworks
- Aufbau der Toolchain
- Entwicklung der Generatoren für Persistenz und Datenvalidierung
- Erprobung des Ansatzes in Bestandsprojekten

➔ openArchitectureWare (Generator)
➔ Fornax (Generatorkomponenten)
➔ Projektvorgabe EnterpriseArchitect

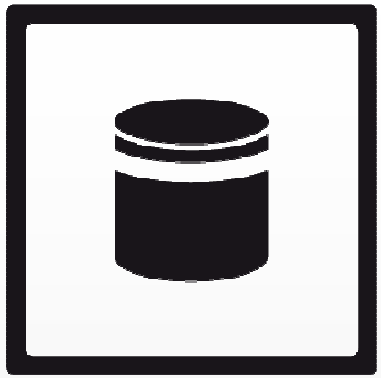
➔ UML 2.x (Modell)
➔ JavaPersistence API (Persistenz)
➔ Bean Validation API (Datenvalidierung)

Realisierung der Toolchain

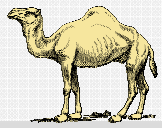




Persistenzgenerierung
JPA-Cartridge



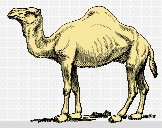
- Generierung von
  - Java-Klassen mit JPA-Annotationen aus UML-Klassen und UML-Stereotypen
  - SQL/DDL-Skripten
  - Standard-DAOs für CRUD
  - Integration mit Derby für lokale Tests
- Aus einem einfachen UML-Modell wird der Standardfall generiert. (Convention over Config)
  - hierfür ist kein großes DB/SQL-Wissen notwendig
  - DB-Admins / SQL-Profis können in den Generierungsprozess eingreifen, um Optimierungen einfließen zu lassen



Generierung von Validierungen
BVA-Cartridge

- Generierung von
  - Java-Klassen mit Validierungs-Annotationen nach Bean Validation API (BVA - JSR-303)
  - Hilfsmethoden für Aufruf der Validierungsregeln
- Validerungsregeln im UML-Modell hinterlegt
- einheitliche und konsistente Datenvalidierung für die GUI, Datenbank und Schnittstellen
- kann mit dem JPA-Cartridge kombiniert eingesetzt werden

Live- und Featuredemo



Features im Überblick

- Vorkonfigurierte Toolchain

JPA-Cartridge

- UML-Profil
- Entitäten mit JPA-Annotationen
- 1:n- und n:m-Beziehungen
- unterstützt Vererbung (one table per class hierarchy)
- **unterstützt fachliche (composite) und technische Schlüssel**
- Anpassung des DB-Mappings mit Tagged-Values
- DAO mit CRUD-Operationen
- Generierung der DDL

BVA-Cartridge

- UML-Profile
- Validierungs-Annotationen nach JSR-303
- **unterstützt anwendungsspezifische Validatoren**
- Gemeinsam mit JPA-Cartridge verwendbar

Herausforderung während der Umsetzung

Enterprise Architect

- exportiert selbst kein EMF-Modell, daher
  - Export über EA-Java-API mit UML2Exporter (SourceForge-Projekt) nach EMF
  - Erfahrungen und Bugfixes in UML2Exporter mit eingeflossen
- Kein Export gleich benannter Tagged-Values
- Keine Wert-Listen bei Tagged-Values

openArchitectureWare

- Hohe Einstiegshürde
- Debugging möglich aber schwierig
- Javabasic-Cartridge kommt mit mehrfach vererbten Stereotypen nicht zurecht
  - Modell-to-Modell Transformation in EMF notwendig

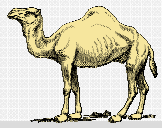
Herausforderungen aus der Praxiserprobung

JPA-Cartridge

- Erprobung in Bestandsprojekt zur Angebotskalkulation von Nutzfahrzeugen in Arbeit
  - Sanfte Migration bei Weiterentwicklungen
  - Notwendige Vorarbeiten
    - WebSphere 6.1 (JDK 1.5, JPA)
    - Parallelbetrieb JDBC/OJB/JPA

BVA-Cartridge

- Erprobung im Bestandsprojekt für Gebrauchtfahrzeugverkauf
  - Reverse-engineertes Modell führt EA und UML2Exporter an seine Grenzen
  - Integration in Buildprozess schwierig (Windows)

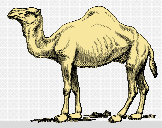


Fazit
Persistenz und Validierung aus dem UML-Modell gewinnen?

Neuprojekt: Ansatz problemlos umsetzbar

Bestandsprojekt: kann Umsetzung schwieriger werden

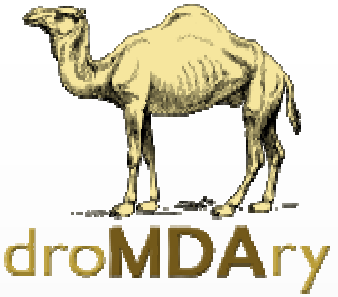
- Mögliche Probleme bei Einführung in Bestandsprojekten
  - Aufwand für das UML-Modell kann hoch werden
  - Evtl. Re-Engineering erforderlich, um Klassenmodell dem Generatormodell anzupassen
- Vorteile für Bestands- und Neuprojekte
  - UML ist für sehr großen Personenkreis inzwischen bekannte Notation
  - Zentrale Doku der Validierungsregeln und Persistenzinformationen im Modell
  - Effizienz gesteigert, da manuelle Arbeit durch Generierung ersetzt wird
  - DRY: Mapping, Java-Code und DDL werden konsistent generiert



Ausblick

- Offene Punkte
  - JPA
    - **Bidirektionale Beziehungen**
    - Weitere Vererbungsstrategien
    - Vollständiger Support für JPA 2.0
    - **Migrations-DDL**
  - BVA
    - Export von gleich benannter Tagged-Values
    - Unterstützung von Listen in Tagged-Values
    - **Unterstützung von Validatoren-Gruppen**
- Weitere Ideen
  - Portierung von oaw 4.3 nach oaw 5
  - **Support für weitere UML-Tools (z.B. MagicDraw)**
  - Generieren von JavaScript-Validatoren für Webapplikationen
  - Generatoren für weitere Sprachen (C#, C++, OpenC, JME)
  - Generator für Unit-Tests

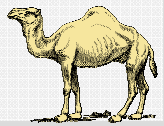
Veröffentlichung / Community



droMDAry

- Generatoren werden mit Open Source-Lizenz veröffentlicht
- Wird aktiv in Projekten eingesetzt, weitere Anwender erwünscht
- Weitere Informationen bei SourceForge
  - http://www.droMDAry.org
  - http://droMDAry.sourceforge.net
  - Forum, Mailingliste, Wiki

Lizenz: EPL



Vorstellung aller Beteiligten

Diplomanden

- Manuel Renz
  BVA
  (RZSystems)

- Matthias Ziegler
  JPA
  (RZSystems)

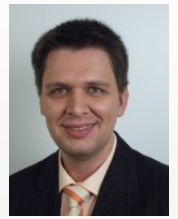
Anwendungsprojekte

- Michael Schulze
  JPA
  (T-Systems)

- Jochen Berger
  BVA
  (T-Systems)

Betreuer

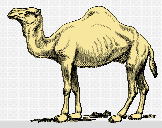
- Darko Palic
  Technische Betreuung
  (Xenovation)



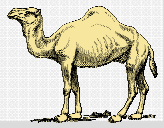
- Volker Rügner
  Unterstützung techn. Betreuung
  (Freelancer)

- Matthias Meusel
  Organisatorische Betreuung
  (T-Systems)

Fragen ?



Vielen Dank für Ihre Aufmerksamkeit.



Motivation (1/2)

Unterschiedliche Begriffsdefinition „Entwickler“

Selbstbild Entwickler

- nicht nur "Implementierer„, sondern auch
  - Designer & teilweise Architekt
- Fachkonzeptionist oder zumindest sehr akribischer Reviewer der Fachkonzeption
- Analyst
- oft auch noch Tester

➔Kosten meist zweitrangig, da ideale Lösung angestrebt wird

➔Doku nicht das Lieblingswerk

Entwickler aus Sicht des Managements

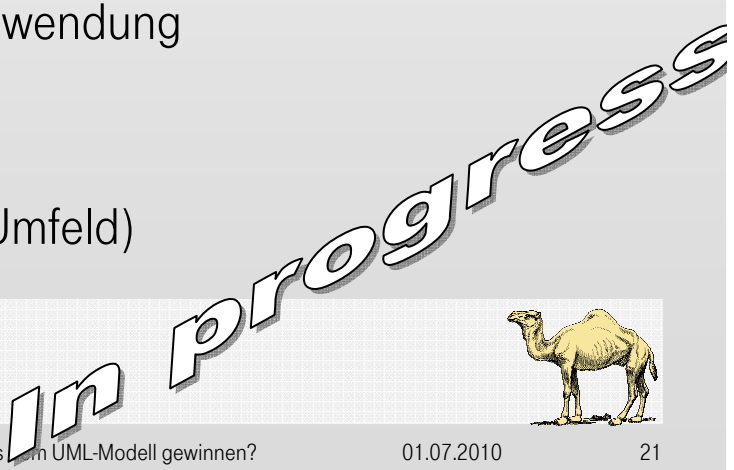
- Implementierer
- Skillverlagerung Richtung Design und Architektur vom Entwickler gewünscht
- Kostendruck durch Mitbewerber

➔Häufige Motivation für Verlagerung der Implementierung in günstigere Schwellenländer

➔Doku zwingend erforderlich für Verlagerung

Lösungsansatz und Ziele (1/2)

- Ansatz: Wenn Generatoransatz in Bestandsprojekt möglich ist, dann ist Integration in Neuprojekte möglich ➔ Konkrete Umsetzung eines MDSD-Ansatzes in Bestandsprojekten.
  - Lösungsansätze adressieren Standardprobleme in Projekten.
    - Persistenz
    - Datenvalidierung
  - Aufbau einer Toolchain mit vorgehender Evaluierung der auf dem Markt befindlichen Tools
  - Aufbau entsprechender Generatoren
- Ausgrenzungen
  - keine eierlegende Wollmilchsau
    - Generator erstellt aus UML keine vollständige Anwendung
    - UML-Editor soll IDE **nicht** ersetzen
  - das Rad nicht neu erfinden
    vorhandenes wiederverwenden (z.B. aus dem OSS-Umfeld)



Lösungsansatz und Ziele (2/2)

- Anforderungen
  - Eine einfache Toolchain mit minimal erforderlichen Eigenentwicklungen
  - Generatoren für Standardprobleme im Softwareprojektalltag
    => Werkzeugkasten für verschiedene Probleme
  - Einstiegshürden sehr weit nach unten ziehen (KISS)
    => Wunsch: keine Einstiegshürde
  - sehr enge Anlehnung an entstehenden oder bestehenden Standards
    => daher auch die Wahl UML als Basis
  - Berücksichtigung vorhandener Tools und Standards auf dem Markt
    - openArchitectureWare (Generator)
    - Fornax (Generatorkomponenten für POJOs)
    - Bean Validation API (BVA) mit Ref-Impl: hibernate-validation
    - Java Persistence API (JPA) mit Ref-Impl: hibernate
  - Projektvorgabe: Enterprise Architect