

IDE-Integration einer
MDSD Toolchain

ex|Xcellent
solutions

Achim Demelt
Andreas Lux

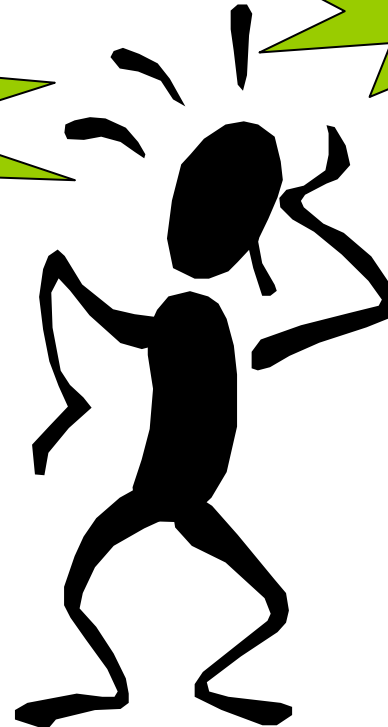




Die Lösung:

Möglichst viel **modellieren**

Möglichst viel **generieren**



Aufgaben, die bei der Entwicklung in einem typischen MDSD Projekt anfallen:

- x|** Formale Beschreibung fachlicher Anforderungen in einem Modellierungstool (Analyse)
- x|** Beschreibung technischer Abbildung in einem (anderen?) Modellierungstool (Design)
- x|** Generierung / Implementierung der Anforderungen
- x|** Test der Implementierung
- x|** Deployment der Lösung inklusive einer (generierten?) Dokumentation



Fachliche Analyse durch Analysten



Ableitung der fachlichen Beschreibungen in ein Designmodell durch den Entwickler.



Konkretisierung des Datenmodells / Prozessbeschreibungen durch den Entwickler



Generierung von Klassen- / Methodenrumpfen aus dem Modell



Ausprogrammieren generierter Rumpfe (Roundtrip!?)



Änderung der Anforderungen!!!



Welcher Eindruck bleibt beim Entwickler hängen?

- x| Mehr Arbeit in „ungeliebten“ Modellierungstools

(Modellierung wird oft als Ballast empfunden)

- x| Umständlichere Implementierung

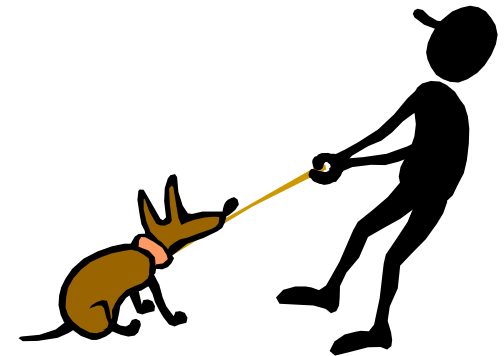
(Implementierungsphasen werden oft durch zusätzliche Generierungsläufe „unterbrochen“)

- x| => Gefühlte **Ineffizienz**



Gefühlte ineffektive Entwicklungsarbeit, da

- x| mehrere Steps pro Iteration
- x| mehrere Tools
- x| Häufige Neugenerierung notwendig
- x| Modell und Code in unterschiedlichen Systemen versioniert werden



Beim Entwickler ergeben sich daraus folgende Konsequenzen:

- x| Erhöhung des Frustrafaktors
- x| Schlechtere Arbeitsqualität
- x| Schlechte Zielerreichung

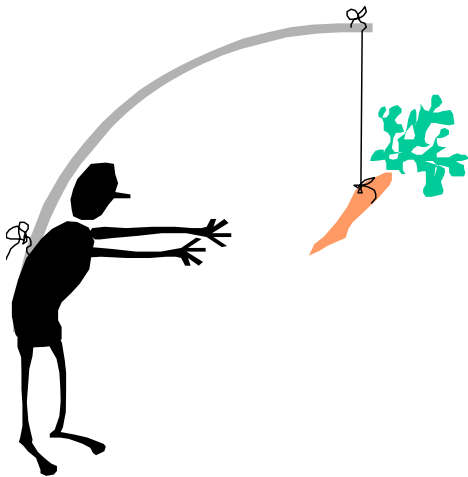
=> „Modellgetriebene SW-Entwicklung bringt keine Vorteile“



Muss das so sein?

Wie sieht die optimale Umgebung für MDSD aus?

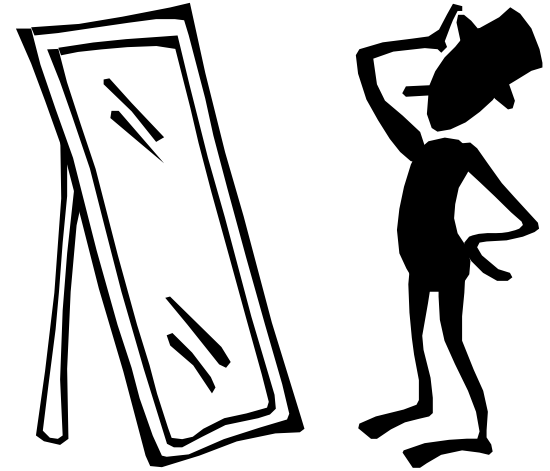
- x|** Modellierung und Implementierung im selben Tool
- x|** Schnelle Navigation **im** Modell und Code
- x|** Schnelle Navigation (Roundtrip) **zwischen** Modell und Code
- x|** Schnelle (Code-)Generierung aus IDE und im Batch
- x|** Generierung muss jederzeit möglich sein
- x|** Konsequente Trennung von technischen und fachlichen Details.



Modell hat Code-Qualität!

- x|** Gemeinsame Versionierung von Code und Modell
- x|** Verwendung einer dem Problem und den Frameworks angemessenen Modellierungssprache (=> DSL?!)
- x|** Codier-Komfort auch bei der Modellierung
- x|** Visualisierung von Modellierungsfehlern und Regelverletzungen (analog zur Syntaxvalidierung im Code)
- x|** Wiederverwendung von Modellinhalten

=> **Entwickler muss sich wiederfinden**



Anforderungen an die Plattform:

- x| Möglichkeit zur grafischen Modellierung von fachlichen und technischen Sachverhalten
- x| Unterstützung von eigenen Metamodellen
- x| Muss zwingend eine IDE sein bzw. enthalten (Inkl. großem Toolset für die Entwicklung)
- x| Einbindung eines (Code-)Generators
- x| Muss um eigene Features erweitert werden können

=> **Eclipse !?**



Ganz konkret: wie lässt sich das nun umsetzen?

- x| Modellierung 1: Verwendung von UML 2.1
- x| Modellierung 2: GMF-basierte grafische Editoren
- x| Code-Generierung: oAW
- x| Dokugenerierung: ???



Intention:

x| Unterstützt bereits die meisten UML 2 Diagramme

Was wir vermissen:

x| Descriptionfeld für Dokumentation

x| Verknüpfungen zwischen Modellelementen und Diagrammen

x| Keine UI-Entwürfe mit der UML

x| UML kann zwar erweitert, aber nur schwer eingeschränkt werden

x| Generator für Dokumentationen

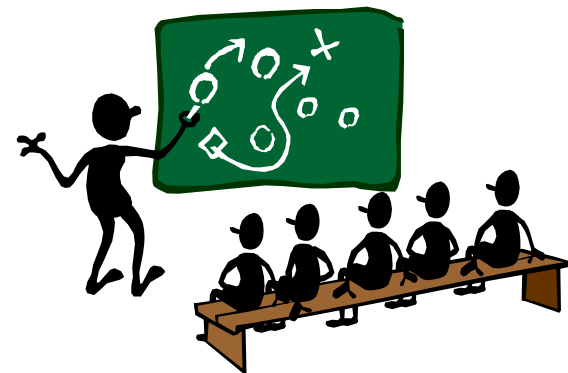
Intention

- x|** Reduktion des sprachlichen Umfang an die Möglichkeiten des verwendeten Frameworks
=> Reduktion der Komplexität

- x|** Anreicherung des Diagramms mit Komfortfunktionen
 - | „content assist“
 - | Suchfunktionen
 - | Navigation Modell <-> Code
 - | Komfortable OCL Unterstützung
 - | Annotations

Intention:

- x|** Einfache Möglichkeit zu UI Definitionen (und Generierung?)
- x|** Verknüpfung von UI- zu Klassen-Elementen
- x|** Nutzung von bestehenden Informationen zur UI-Entwurf und Dokumentation (-Generierung)
- x|** „Räumliche Nähe“ zu UseCases
- x|** Einbettung in generierte Dokument
- x|** Kein Toolwechsel



Intention:

- x| Nutzung eines „Standards“ zur (Code-) Generierung
- x| Verwendet EMF Modelle aus der Modellierung als Basis für die Codegenerierung
- x| Implizites (flüssiges) Auslösen des Generatorlaufs.
- x| Verstecken der oAW Konfiguration

Intention:

- x| Keine Modellierung zum Selbstzweck
- x| Nutzung „echter“ Inhalte für Dokumentationszwecke
- x| Keine Einschränkung auf UML-Modelle
- x| Einfache Handhabung
- x| Reproduzierbare Dokumente

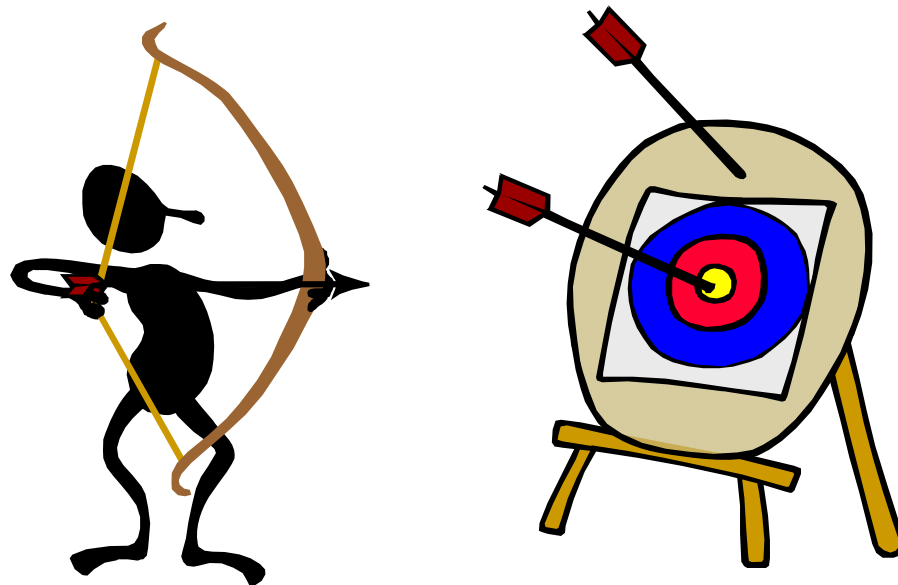
Was haben wir bisher aus dem Einsatz der Toolchain gelernt:

- x|** Bau einer eigenen Toolumgebung macht nicht für jeden Sinn
 - | Aufbau der Toolchain from Scratch ist relativ aufwändig.
- x|** Zielsetzung für eine MDSD Entwicklung muss frühzeitig klar sein
 - | Warum machen wir das Ganze?
- x|** Es wird viel Spezialwissen benötigt (z.B: GMF), das im Projekt selbst nicht verwendet werden kann
- x|** Das Team muss sich an eine andere Denkweise gewöhnen.
 - | Nicht für jeden Fall gibt es eine bekannte Lösung

Was haben wir bisher aus MDSD basierten Projekten gelernt?

- x|** Beteiligte müssen das Konzept mit tragen
- x|** Gefühlter Zeitverlust bei Generierung
- x|** Keine Einmal-Code-Generierung aus dem Modell.
- x|** Roundtrip durch fehlende Abstraktion wenig hilfreich
- x|** Probleme bei Standard Modellierungssprachen und Generatoren
- x|** Schneller Projektstart durch „New Project Wizard“





Vielen Dank für Ihre Aufmerksamkeit

