

JEAF:

anap|tecs

Modellgetriebene Entwicklungsprozesse in der
Praxis - eine Bestandsaufnahme

Tillmann Schall, anap|tecs GmbH

pure business solutions



JEAF:

▪ Agenda

- Grundlagen modellgetriebener Entwicklungsprozesse
- Schritte zur Einführung
- Erfahrungen aus der Praxis
- Fazit

JEAF:

- ⌘ Grundlagen modellgetriebener Prozesse



... die übernächste Version von Generatoren ;-)

: Motivation

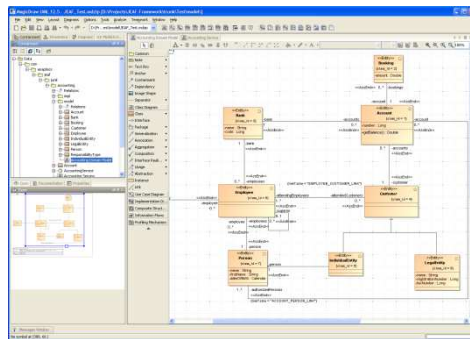
Auslöser für modellgetriebene Entwicklung

- Komplexität eines bestehenden Systems lässt sich nicht mehr beherrschen
- Schwächen im Design (z.B. Schnittstellen, Entkopplung)
- Große Einarbeitungszeiten und sehr große Unterschiede im Aufbau des Codes zwischen einzelnen Teams
- Hoffnung auf mehr Effizienz und weniger Fehler
- Vereinheitlichung von Code-Strukturen und Reduzierung von Freiheitsgraden

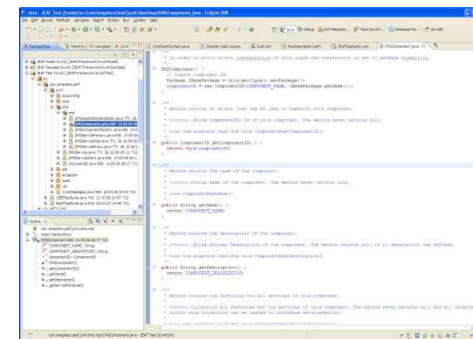
JEAF:

■ Idee

- Basis aller modell-getriebenen Entwicklungsprozesse ist die Idee, vor der eigentlichen Implementierung eines Systems, dieses auf einem höheren Abstraktionsniveau bzgl. seiner Architektur und seinen Strukturen formal zu beschreiben
- Komponenten, Services, Entitäten etc. werden vor der Umsetzung zuerst in einem UML Modell exakt beschrieben
- Auf Basis der formal im Modell beschriebenen Systemteile erfolgt anschließend die automatisierte Generierung von Teilen des Programmcodes



UML Modell



Entwicklungsumgebung

: Model Driven Software Development

Was bedeutet „**modellgetriebene Entwicklung**“ für die tägliche Arbeit?

- Das UML Modell steht in allen Phasen der Entwicklung (von der Analyse bis zur Implementierung) im Mittelpunkt
- Modell ist Master gegenüber Code. Es erfolgt kein Roundtrip Engineering.
- Optimaler weise erfolgt das gesamte Design einer Applikation auf Basis von UML Modellen
- Alle technischen und fachlichen Informationen über eine Applikation befinden sich an einer zentralen Stelle, dem UML Modell z.B. Architektur, Use Cases, Komponenten, Services, Entitäten, DB-Mapping, Deployment-Einheiten, ...
- Zur Synchronisation von Modell und Code kommt ein Generator zum Einsatz

■ Die Schritte vom Modell zum Code

- Erfassen der notwendigen Informationen im Modell (z.B. Service Interface inkl. Methoden und Parameter sowie Dokumentation)
- Abbildung der notwendigen Zusatzinformationen im UML Modell durch die Definition von entsprechenden Stereotypen
- Zwischenschritt XMI Export
Je nach eingesetztem Generator kann ein Zwischenschritt in Form eines XMI Export notwendig sein
- Ausführen des Generators (Plugin in der Entwicklungsumgebung). Input für Generator sind UML Modell und Transformationsregeln
- Generierter Code wird „ganz normal“ in der Versionsverwaltung eingchecked
- Die Ausführung des Generators ist ein obligatorischer Schritt beim Build der Software
So kann sichergestellt werden, dass keine manuellen Änderungen am generierten Code vorgenommen werden

- Zusammenspiel zw. generiertem und hand-made Code
- Generatoren erzeugen lediglich Code, um Entwickler von Standardtätigkeiten zu entlasten. Die spezielle fachliche Logik muss nach wie vor „konventionell“ / von Hand implementiert werden

Varianten

- Strikte Trennung zwischen generiertem und konventionellem Code
Aufteilung in abstrakte Basisklasse (vollständig generiert) und konkrete Klasse (hand-made)
- Generierter Code wird direkt mit fachlicher Logik angereichert
Generierter Code wird mit so genannten Code-Schutzblöcken versehen, in denen die fachliche Logik implementiert wird. Diese Teile werden auch bei erneuter Generierung nicht verändert.

: Vorteile modellgetriebener Entwicklung

Vorteile eines Modells

- Höheres Abstraktionsniveau beim Architekturfentwurf und Design, Komplexität wird dadurch besser beherrschbar
- Ein Bild sagt mehr als tausend Worte
- Beschreibung erfolgt in formaler Sprache, dadurch ist exakte und widerspruchsfreie Definition möglich
- Erhebliche Verbesserung der Qualität der Dokumentation

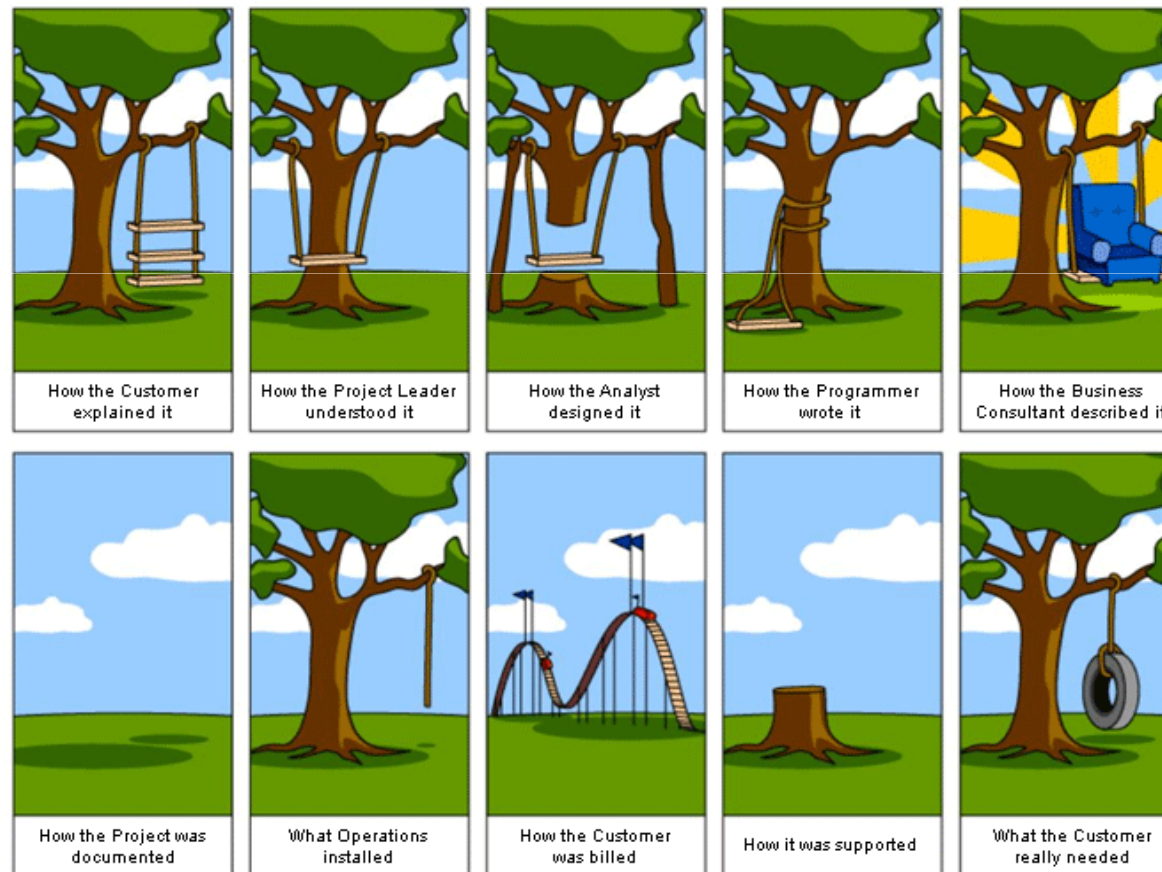
Vorteile durch Einsatz eines Generators

- Zwingt Entwickler zu strukturierter Vorgehensweise (erst Modell dann Code)
- Entlastung der Entwickler von fehleranfälligen Standardtätigkeiten
- Vereinheitlichung des erstellten Codes (stärkere Standardisierung)
- Implizite Qualitätssicherung für UML Modelle

- Nachteile modellgetriebener Entwicklungsprozesse
- Sauberes Configuration- und Change-Management auch für UML Modell notwendig
- Tendenziell größere Anlaufzeiten bei Projektbeginn
- Agilität im Entwicklungsprozess geht zumindest teilweise verloren
- Stärkere Abhängigkeit zu Tools
- Längere Laufzeiten im Build-Prozess

JEAF:

- Einführung von modellgetriebenen Entwicklungsprozessen



■ Notwendige Voraussetzungen

- **Akzeptanz im Entwicklungsteam**

Einsatz von Modell und Generatoren soll die Entwickler in ihrer Arbeit unterstützen und nicht behindern

- **Praxistaugliche Definition der Entwicklungsmethodik**

- **Best Practices**

Bilden die Basis für den aus dem Modell zu generierenden Code

- Design- und Implementierungsrichtlinien
- Modellierungsrichtlinien

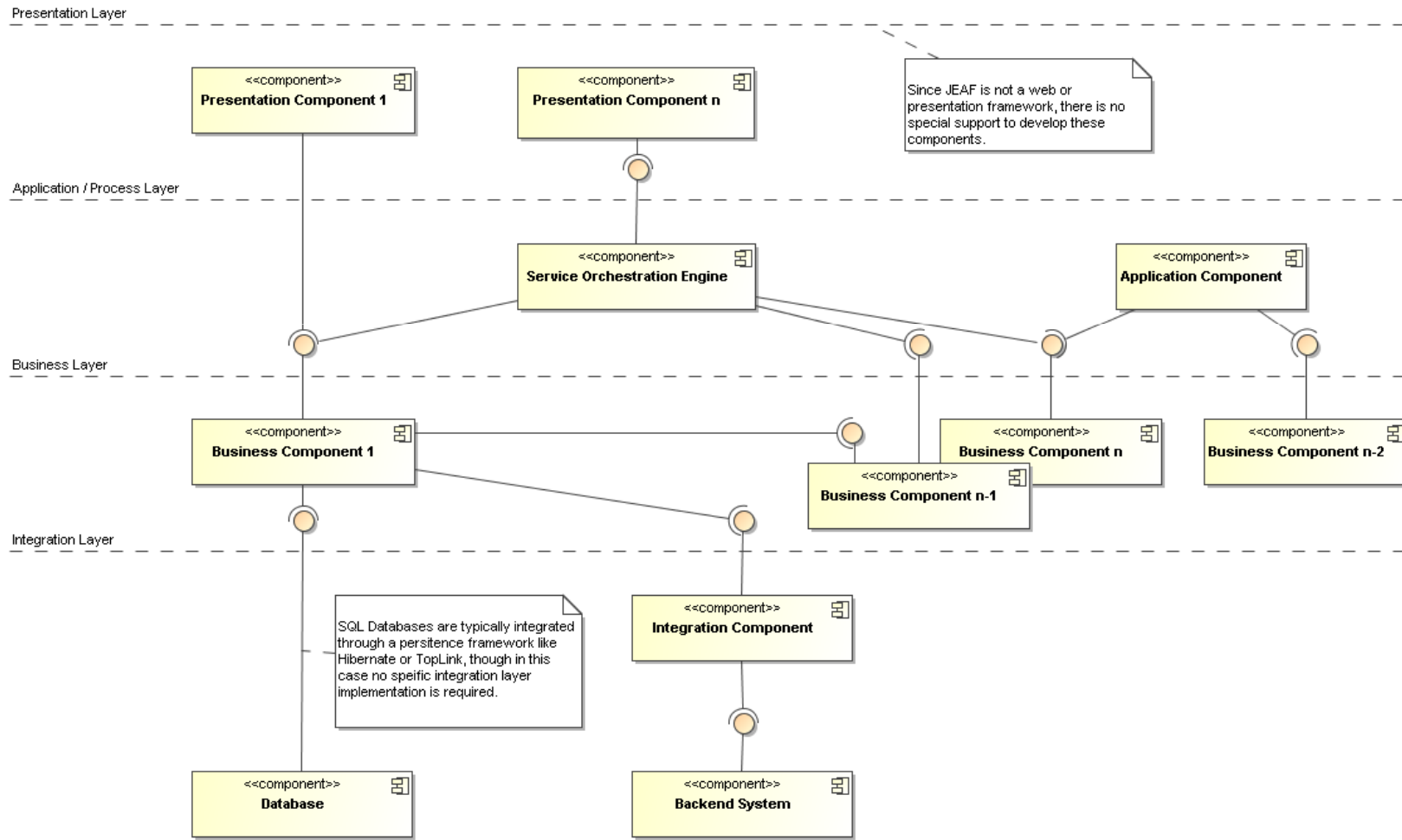
- **Architekturmodell**

Beschreibt unabhängig von einer Applikation / einem Projekt den grundsätzlichen Aufbau eines Systems

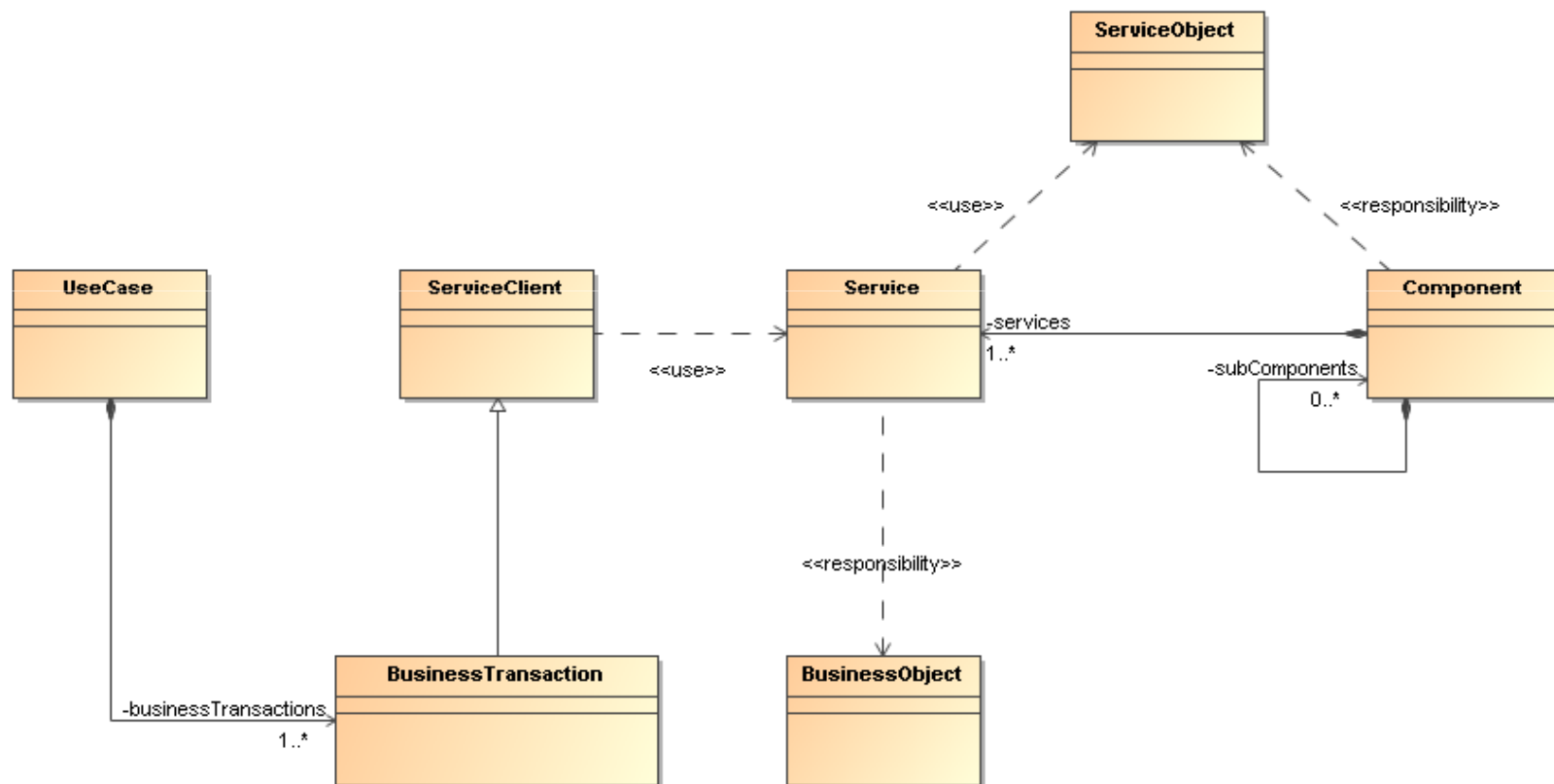
- **Metamodell für Aufbau von Applikationen**

Definiert welche Teile eines Systems mindestens im UML Modell beschrieben werden müssen

Architekturmodell



Metamodell für Aufbau von Applikationen



: Was soll modelliert werden?

Fachlich:

- Use Cases (eventuell Verfeinerung durch Aktivitätsdiagramme)
- Screenabläufe
- Fachliches Domänenmodell

Technisch:

- Komponenten und ihre Abhängigkeiten
- Services (Interface und Service Objekte)
- Entitäten (Attribute und fachliche Methoden)
- Deploymentseinheiten (logisch / physisch)
- Infrastruktur (logisch / physisch)
- Schnittstellen zu Fremd- und Umsystemen

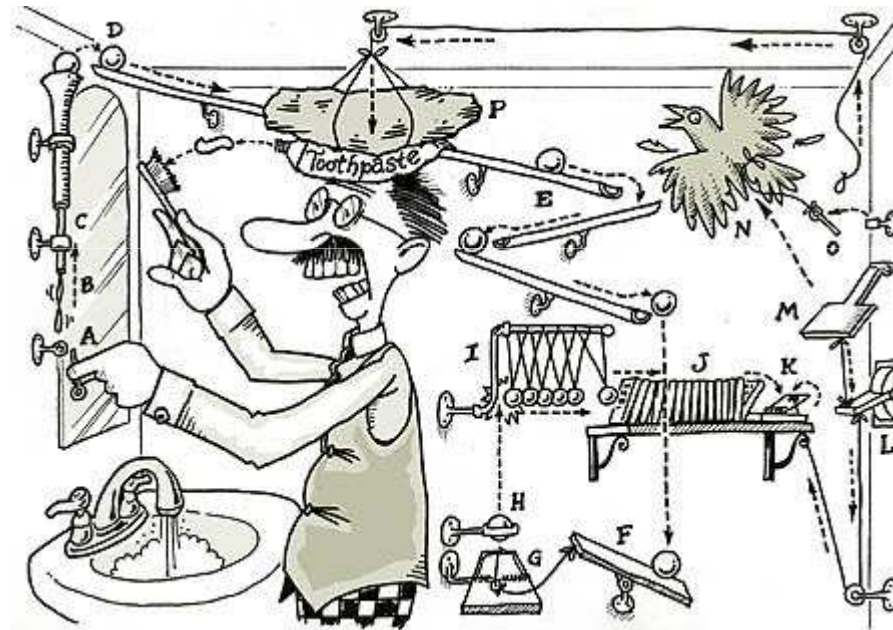
: Was soll generiert werden?

UML Modell	Generator Output
Komponente	<ul style="list-style-type: none">▪ Komponenten Implementierung (z.B. Java Klasse, Konfigurationsdatei etc.)
Service	<ul style="list-style-type: none">▪ Service Interface und Service Objekte▪ Rumpfklasse
Entitäten (Fachliches Domänenmodell)	<ul style="list-style-type: none">▪ Persistente Klasse (inkl. Zugriffsmethoden)▪ DB-Mapping (z.B. für Hibernate)
Use Case	<ul style="list-style-type: none">▪ Use Case Implementierung inkl.
Aktivitätsdiagramm	Ablaufsteuerung
Screenablauf	<ul style="list-style-type: none">▪ Dialog Ablaufsteuerung

- Was sollte nicht generiert werden?
- Grundsätzlich erscheint eine Generierung von dynamischen Aspekten des Systems nicht sinnvoll, hierfür wird die Intelligenz der Entwickler benötigt
- Beispiele
 - Validierungen für fachliche Attribute
 - Implementierungen von fachlichen Methoden
 - Prozess- oder Ablauflogik

JEAF:

- Erfahrungen aus der Praxis



▪ Fallstricke aus der Praxis

- Tools, Tools, Tools
 - Unterstützung für Teamarbeit bei UML Tools meist sehr bescheiden
 - Umgang mit großen Modellen
 - Usability von Modellierungswerkzeugen (→ Akzeptanz)
- Migration / Wechsel Modellierungswerkzeug
- Keep it simple!
- Erstellen eines Modelles bei bereits vorhandener Software (einmaliges Reverse Engineering etc.)
- Generierung von Spezifikationen aus dem UML Modell

: DO's und DONT's

DO's

- Proof of Concept mit kritischen Entwicklern
- Schrittweise Einführung bei bestehenden Systemen
- Team steht dahinter
- Aufbau UML- und Tool-Know-how (interne Schulungen)
- Generiertes Code-Volumen gering halten (auch sauberes Design für generierten Code)
- Generierter Code muss den selben Guidelines entsprechen wie hand-made Code

DONT's

- Methodik ist nicht klar definiert
- Roundtrip Engineering
- MDD wenn UML Tools keine geeignete Unterstützung bieten
- Einsatz von halb-fertigem Generator
- Effizienzsteigerung als Begründung gegenüber Management
- Manuelle Anpassung von generiertem Code zulassen

: Fazit

- In der **richtigen Dosierung** kann der Einsatz von modellgetriebenen Entwicklungsprozessen in Kombination mit einem Generator, das Leben deutlich erleichtern und die eingangs aufgeführten **Vorteile** erzielen
- Nach wie vor ist die **Tool-Unterstützung** im Bereich der UML Modellierungswerkzeuge nicht so wie sie sein sollte
- Die Idee das Modell als **zentrales Repository** für alle Entwicklungsartefakte zu verwenden und dann daraus neben Code auch alle Dokumente etc. zu generieren ist **akademisch** und in der Praxis weder umsetzbar noch sinnvoll
- Auch in **kleinen und mittleren Projekten** lassen sich die **Benefits** schon erzielen

JEAF:

anap|tecs

: Diskussion und Fragen



JEAF:

anap|tecs

: Einladung

Life Demo am Stand der
anap|tecs GmbH auf Basis der
JEAF Platform und
MagicDraw UML