Automatisierte Architekturanalyse mittels UML2.0 Diagrammen

AW2 SoSe09

Person

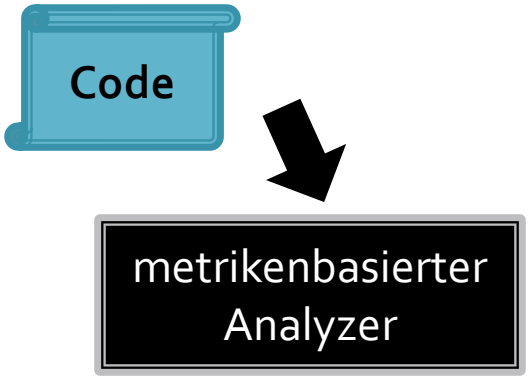
- Vortragender: Thorben Pergande
- Vertiefungsgebiete:
  - kollaboratives Arbeiten im Softwareentwicklungsprozess am Beispiel Microsoft Surface
  - Automatisierte Architekturanalyse, Verständlichkeit der Ergebnisse
- Fragen sind jederzeit erwünscht!

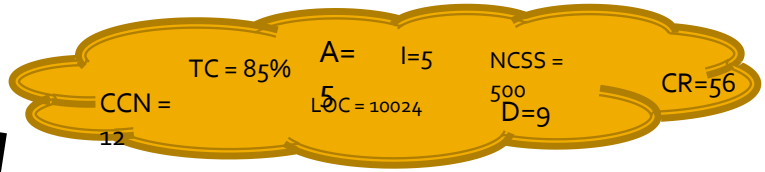
Agenda

- Vorstellung
- Einführung in das Themengebiet
- vergleichbare Arbeiten
  - Vorstellung
  - Bewertung der Arbeiten für diesen Kontext
  - Abgrenzung
- Zusammenfassung und Ausblick
- Quellen
- Fragen und Antworten

Einführung in das Themengebiet

- Warum Architekturanalyse?
  - Vergleich SOLL- mit IST-Architektur
  - Verfall der Architektur entgegenwirken
- Warum mit UML Diagrammen? (1)
  - Gegensatz → **metrikenbasierte** Analyse





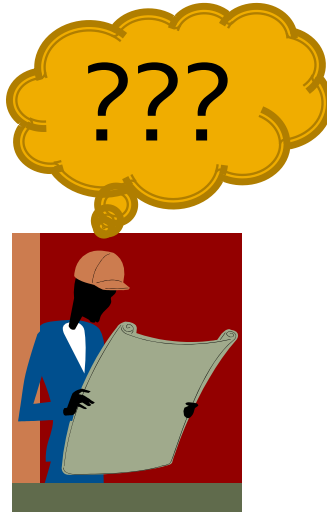


Abb. 1 metrikenbasierte Analyse

Einführung in das Themengebiet

- Warum mit UML Diagrammen? (2)
  - Vision: Output ist selbst ein Diagramm mit akzentuierten Hinweisen auf Abweichungen
  - Interpretation leichter!
  - Kommunikationsmedium über verschiedene Rollen im Projekt hinweg

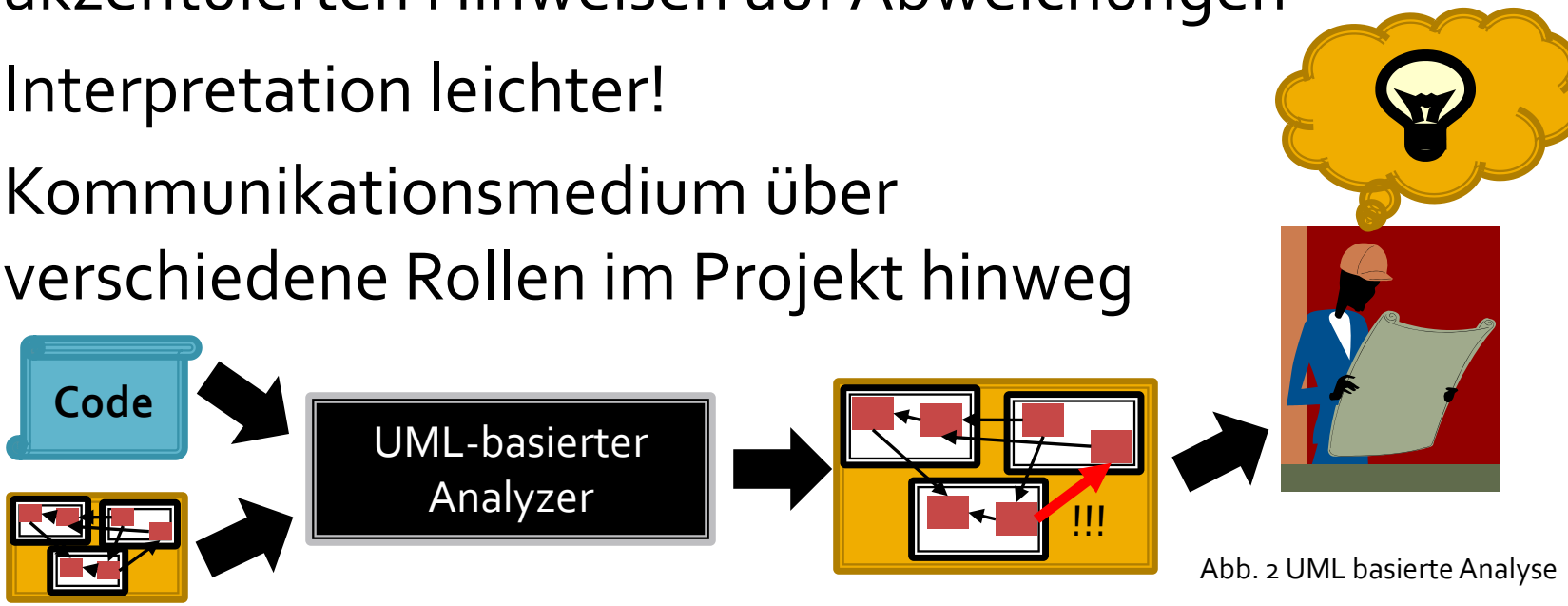


Abb. 2 UML basierte Analyse

Einführung in das Themengebiet

- **Top-Down** Ansatz zur diagrammbasierten Architekturanalyse

1. **Herleiten von Quelltext aus Diagramm**
2. Vergleich von Quelltext
3. Kopie von Diagramm erstellen
4. Differenzen in kopiertes Diagramm einzeichnen
5. Akzentuiertes Diagramm als Ausgabe veröffentlichen

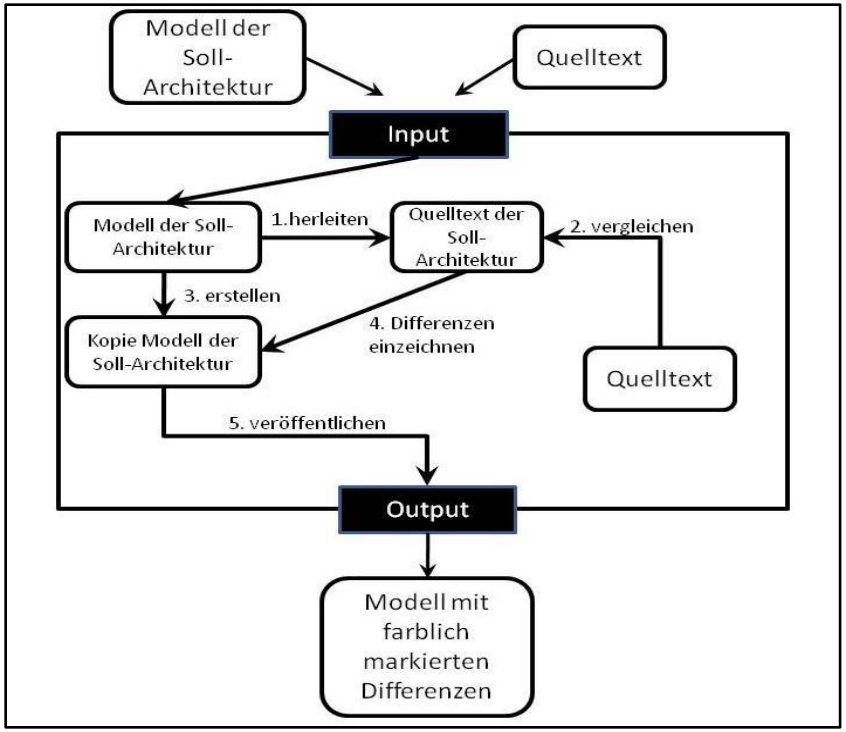


Abb. 3 Top-Down Methode

Einführung in das Themengebiet

- **Bottom-Up** Ansatz zur diagrammbasierten Architekturanalyse

1. **Herleiten von Diagramm aus Quelltext**
2. Vergleich von Diagramm
3. Differenzen in erstelltes Diagramm einzeichnen
4. Akzentuiertes Diagramm als Ausgabe veröffentlichen

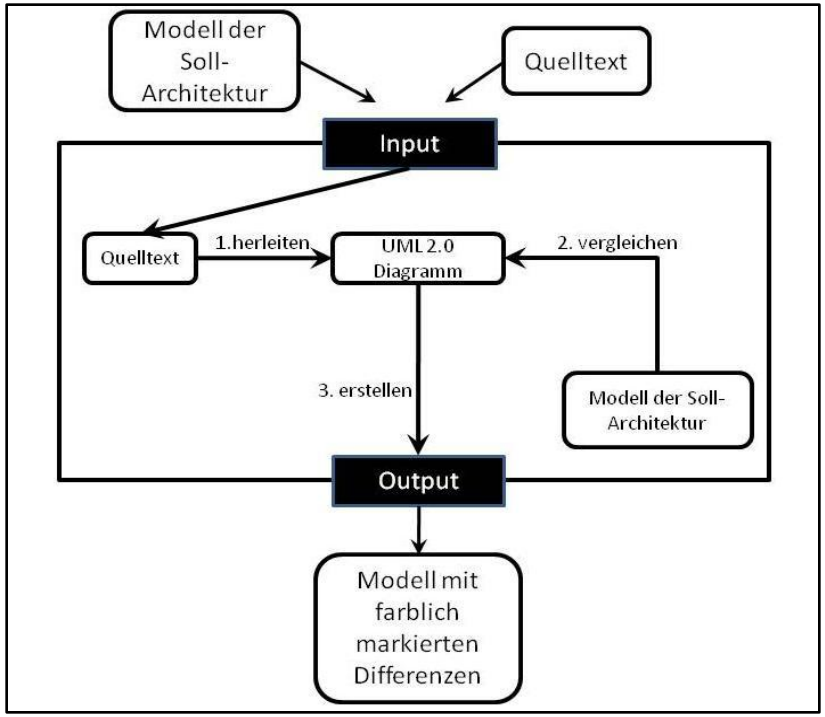


Abb. 4 Bottom-Up Methode

Einführung in das Themengebiet

- Probleme und Fragestellungen
  - allg. wie **vergleicht** man Quelltext mit Diagrammen?
    - Wie erstellt man aus Quelltext ein UML-Diagramm?
    - Wie erstellt man aus einem Diagramm Quelltext?
  - Welche Möglichkeiten gibt es in der **Code-Generierung**
  - Inwiefern können **Werkzeuge/Techniken** z.B. aus dem **MDA-Kontext** eingesetzt werden?
  - Wie **erstellt man Diagramme** oder ändert diese **automatisch**

Vorstellung vergleichbarer Arbeiten

- *„UML-based Desing Test Generation"* [1]

  Autoren: W. Pires, J. Brunet und F. Famalho

- *„On reverse engineering an object-oriented code into UML class diagram incorporating extensible mechanisms"* [2]

  Autoren: Vinita, A.Jain und D. Tayal

- *„Metamodel Approach on Model Conformance and Multiview Consistency Checking"* [3]

  Autoren: ChenShu, QuGuoQing und XiaoJing

- *„Automatic Code Generation: A Practical Approach"* [4]

  Autor: G. Papadopoulus

„UML-based Design Test Generation"

- Inhalt: **automatisierter** Abgleich von Klassendiagrammen und Java-Quelltext
- „*Design Wizard*" – Junit-Design Tests
  - API in Java, sodass dieselbe Entwicklungssprache genutzt wird

Code 1: DesignTest

```
1  public void testCommunication() {
2    DesignWizard dw;
3    dw = new DesignWizard(``project.jar'');
4    designwizard.ui.Class clazz;
5    clazz = dw.getClass(``A'');
6    Set<String> usedBy;
7    usedBy = clazz.getClassesUsedBy();
8    assertFalse(usedBy.contains(``B''))
9  }
```

Abb. 5 Design-Test Beispiel [1]

„UML-based Design Test Generation"

- **MDA** Techniken für automatisierte Generierung der Tests

• **ATL** als MDA-Framework eingesetzt → Transformationsregeln, müssen manuell erstellt werden

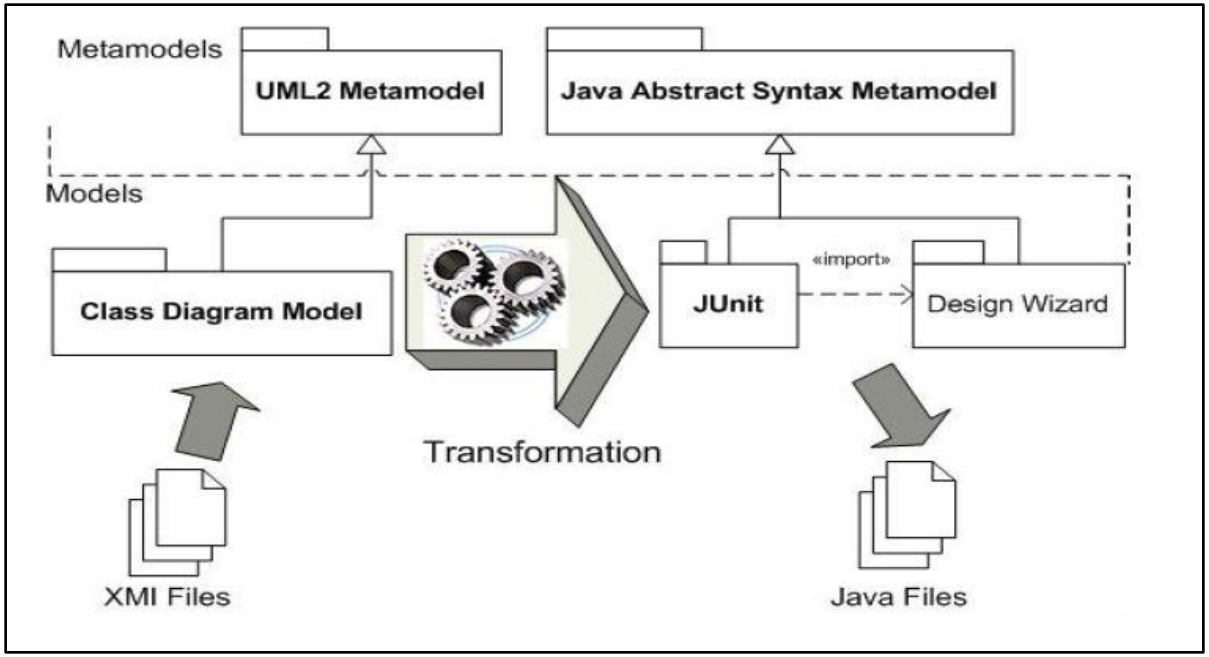


Abb. 6 MDA-Architektur[1]

„UML-based Design Test Generation"

- Bisher nur für **Java** und **Klassendiagramme** anwendbar
  - Weitere Modelle sind geplant
- **Output** ist erfolgreicher oder nicht erfolgreicher Test
- Für Ausarbeitung relevant:
  - Ähnlich dem **Top-Down** Ansatz
  - Techniken zur Code-Generierung
  - MDA-Techniken
  - Import via **XMI-Format**, evtl. Output als XMI-Format!?

„On reverse engineering an object-oriented code into UML class diagram incorporating extensible mechanisms"

- **Inhalt**: beliebige OO-Sprache in UML-Klassendiagramm überführen
- Arbeit stellt einen **generischen Algorithmus** mit insgesamt 15 Regeln in 10 Schritten vor
- Arbeit hebt hervor, dass **viele Möglichkeiten** des Klassendiagramms in UML 2.0 genutzt werden (z.B. tagged values)

„On reverse engineering an object-oriented code into UML class diagram incorporating extensible mechanisms"

- **Algorithmus-Struktur**:
  - Schritt 1
    - **Regel1**: Finde Stichwort „**Klasse**" und die Attribute und Methoden
    - **Regel2**: **abstrakte** Klasse → Schlüsselworte „abstract" oder „virtual"
    - **Regel3**: **Template** Klassen → Schlüsselworte „template"
    - **Regel4**: **Strukturen** → Schlüsselworte „struct"
  - Schritt 2…
    - Regel…

„On reverse engineering an object-oriented code into UML class diagram incorporating extensible mechanisms"

- Die für diesen Kontext interessantesten Regeln:
  - **Regel6**: Finde **gruppierende** Elemente→ Stichworte: „package", „namespace"…
  - **Regel10,11,12**: Finde **Assoziationen, Aggregation oder Kompositionen** → Attribute untersuchen, ob andere Klassen als Typ genutzt werden
  - **Regel13**: Finde **Vererbung** → Stichworte: „:" oder „implements"
  - **Regel15**: Identifiziere **Erweiterungs**mechanismen (z.B. tagged values)

„On reverse engineering an object-oriented code into UML class diagram incorporating extensible mechanisms"

- Bisher auf **Klassendiagramme** beschränkt, aber mit den meisten genutzten **Features** der UML Klassendiagramme
- Bietet genau **eine** Lösung: *Wie erstellt man aus Quelltexten ein Diagramm?*
- Interessanter Ansatz, Portierung auf **Komponentendiagramme** möglich!?

„Metamodel Approach on Model Conformance and Multiview Consistency Checking"

- **Inhalt**: Erstellung eines Metamodells zu einem konkreten Diagramm, anhand dessen die Einhaltung des Metamodells und verschiedener Sichten darauf realisiert wird
- **BON** als Modellierungssprach
- B als **Framework** für Metamodelle von BON
- Ein **Schwerpunkt**: formal beweisbar

„Metamodel Approach on Model Conformance and Multiview Consistency Checking"

- **Metamodell** von BON bekannt und spezifiziert
- Metamodell wird in einem Framework als logische **Constraints** hinterlegt
- Dadurch kann geprüft werden, ob ein **konkretes** Modell dem Metamodell genügt

$$\textbf{Unique_Feature_Name_Assertion} \widehat{=}$$
$$(\forall c \in CLASS)((f_1 \in Class_Features\ (c)$$
$$\wedge f_2 \in Class_Features\ (c) \Rightarrow$$
$$Feature_Name(f_1) = Feature_Name(f_2)) \Rightarrow (f_1 = f_2))$$

Abb. 7 Constraint Beispiel [3]

„Metamodel Approach on Model Conformance and Multiview Consistency Checking"

- **Multiview Consistency Checking:**
  - Dynamische Sicht mit statischer Sicht vergleichen

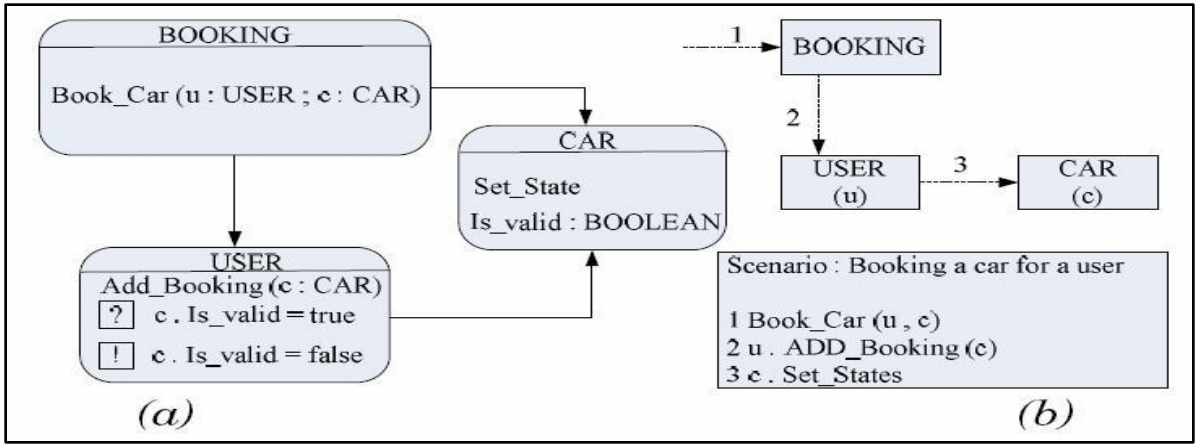


Abb. 8 Vergleich von Diagrammen [3]

$$\text{Class_and_Object_Assertion} \triangleq (\quad o \in \text{OBJECT})(o \in \text{union}(\text{CLASS})).$$

Abb. 9 Constraint Beispiel für Modelvergleich [3]

„Metamodel Approach on Model Conformance and Multiview Consistency Checking"

- Ausarbeitung nimmt **nicht UML** zur Modellierung und **keine** automatisierten Ansätze
- Interessant: ein Metamodell für Diagramm erstellen → **Einhaltung** konkrete Diagramme **prüfen**
- Weiterführung für die Frage: *Wie vergleicht man Diagramme miteinander?*

„Automatic Code Generation: A Practical Approach“

- **Inhalt**: Fallstudie für Code-Generierung aus UML 2.0 Diagrammen in eine Programmiersprache namens „**Manifold**“
- **Anleitung/Ratschläge** für:
  - Architekturmodellierung
  - Regeln für Code-Generierung

„Automatic Code Generation: A Practical Approach"

- Anleitung für Diagrammerstellung sehr **prozesslastig**
- Mapping Diagramm ←→ Quelltext **manuell** zu erstellen

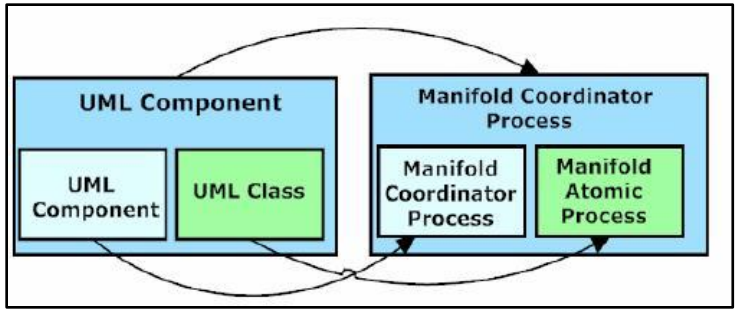


Abb. 10 Mapping [4]

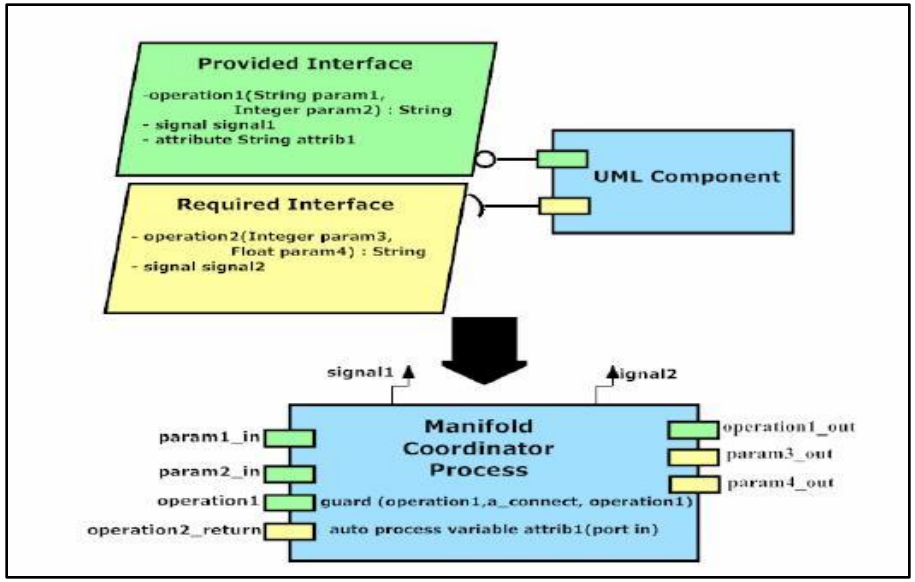


Abb. 11 Transformation[4]

„Automatic Code Generation: A Practical Approach"

- Verhaltensmodelle
  - Anleitung, wie ein Verhaltensmodell **gestaltet** werden soll
  - Beschreibt Initialisierung von **Klassen** und Inhalte von **Methoden**
- Code-Generation
  - MDA-Ansätze und **konkrete** Realisierung der Transformationen (PIM → PSM etc.)
  - Transformationsregeln **nicht** angegeben

„Automatic Code Generation: A Practical Approach"

- Interessant für die Frage: *„ Wie erstellt man aus einem Diagramm einen Quelltext"*
- Interessant sind die Ansätze, wie ein UML 2.0 Diagramm **verarbeitet** werden kann
  - **XMI** als Austausch- und Verarbeitungsformat
- Auffällig ist enger Rahmen, wie die Modelle **erstellt werden müssen**

Zusammenfassung und Ausblick

- **XMI** als Format für Diagramme
  - Erleichtert Verarbeitung, Manipulation, Vergleiche
- Paper Pires [1] auch **Architekturanalyse**!
  - Auch für Entwickler!
  - **Adaption** der Mechanismen!?
- Paper Vinita [2] **Algorithmus** für IST- zu SOLL-Architektur (Klassendiagramme)
  - Adaption auf **Komponentendiagramme**!?

Zusammenfassung und Ausblick

- Paper von [4] und [1] zeigen auch aktuellen Stand **MDA-Realisierung**
  - Modeltransformationen
  - **Code-Generierung**
  - **Manuelle** Regeln nötig
- Paper von ChenShu [3] -> Anleitung **formale** Vergleiche von Diagrammen
  - Ein Vergleich über **XMI** könnte aber einfacher zu realisieren sein
  - Automatisierte Elemente fehlen noch gänzlich!

Quellen

- [1]. *UML-based Design Test Generation.* **Waldemar Pires, Jo˜ao Brunet, Franklin Ramalho.** Fortaleza, Cear´a, Brazil : s.n., 2008. SAC 08.
- [2]. *On reverse engineering an object-oriented code into UML class diagrams incorporating.* **Vinita, Jain, Amita und Tayal, Devendra K.** 2008.
- [3]. *Metamodel Approach on Model Conformance and Multiview.* **ChenShu.** China : s.n., 2008. 863 National High-Tech Research and Development Plan.
- [4]. *Automatic Code Generation: A Practical Approach.* **Papadopoulos, George A.** Cavtat, Croatia : s.n., 2008. ITI 2008.

Fragen…

- … und bestimmt Antworten!

Q&A