

# 45. Werkzeuge zur Programmüberführung (Codegenerierung und Round-Trip Engineering)

Prof. Dr. Uwe Aßmann  
Technische Universität Dresden  
Institut für Software- und  
Multimediatechnik  
<http://st.inf.tu-dresden.de>  
Version 11-0.1, 29.12.11

- 1) Codegenerierung
  - 1) Beispielwerkzeuge
- 2) Codegenerierungstechniken
  - 1) Schablonenbasierte Codegenerierung
- 3) Round-Trip Engineering

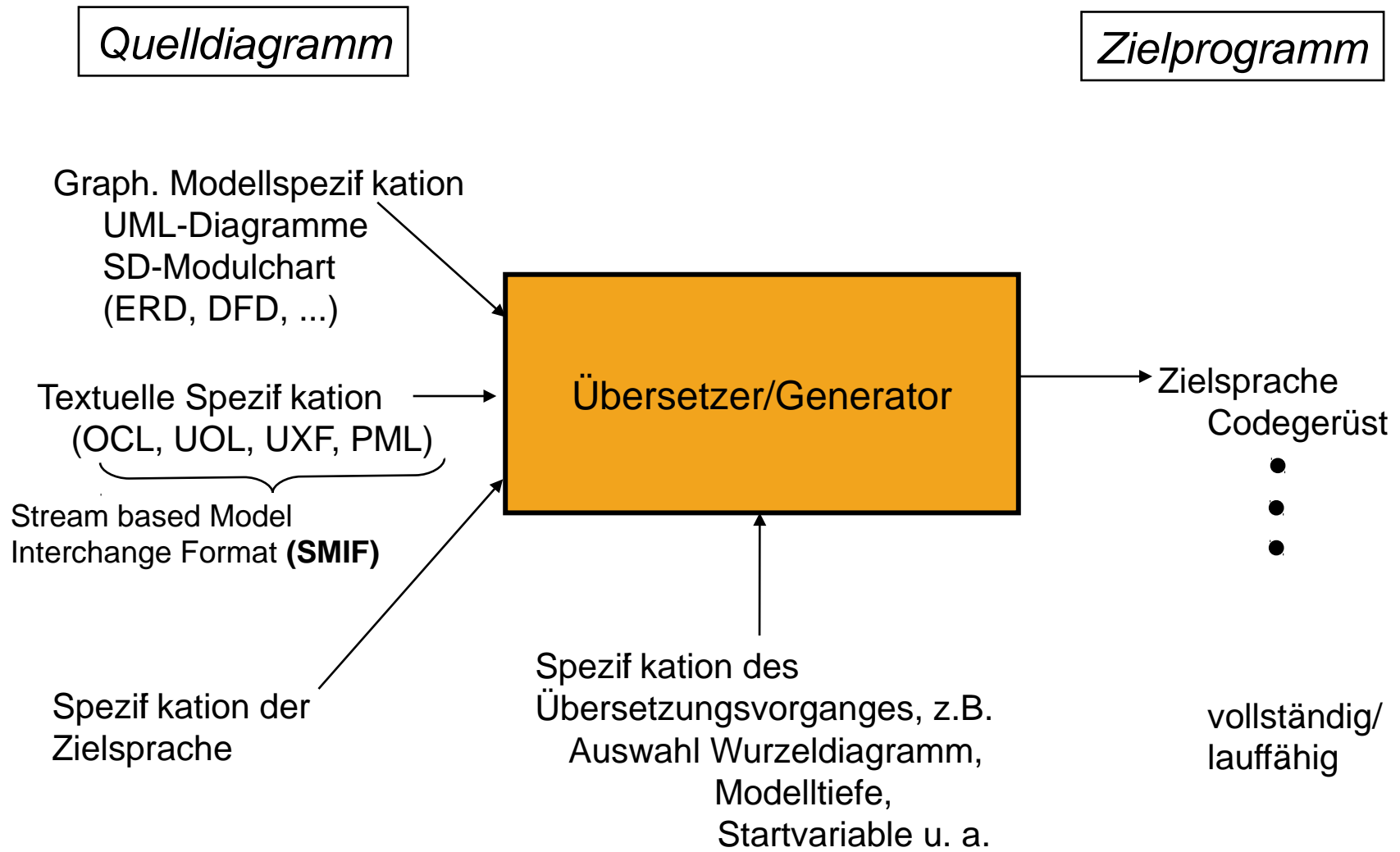
# Literatur

- ▶ <http://www.codegeneration.net/>
- ▶ [www.programtransformation.org](http://www.programtransformation.org)
- ▶ [http://www.codegeneration.net/tiki-read\\_article.php?articleId=65](http://www.codegeneration.net/tiki-read_article.php?articleId=65)
- ▶ Optional
  - Völter, Stahl: Modell-Driven Software Development, AWL 2005.

# 45.1 Codegenerierung

Überführung von Modellen in Programme  
(Programmüberführung)

# CASE-Code-Generatoren



# Phasen eines Compilers, hier Code-Generatoren

*typische Werkzeuge*

Quelldiagramm

Scanner **lex, flex**

lexikalische Analyse

der kleinsten syntakt. Einheiten  
des Quelldiagramms  
(reguläre Ausdrücke)

**Front-End** Parser **antlr, yacc, bison**

syntaktische Analyse

der Modellsyntax  
(kontextfreie Grammatiken)

*AG-Werkzeug*

**ELI, MAX, MUG, JastAdd,  
Silver**

semantische Analyse

Prüfg. auf semantische Fehler  
(attributierte Grammatiken)

**Back-End**

**(Codegenerator)** *Optimierergener.*

Zwischencode

interne Darstellg. im aktuellen Zustand  
(Bäume, Listen, Tabellen)

(Ziel-  
sprachen-  
abhängig)

**PAG, Optimix,  
grgen**

Codeoptimierung

Codeverbesserung  
(Effizienz, Laufzeit, Speicher)

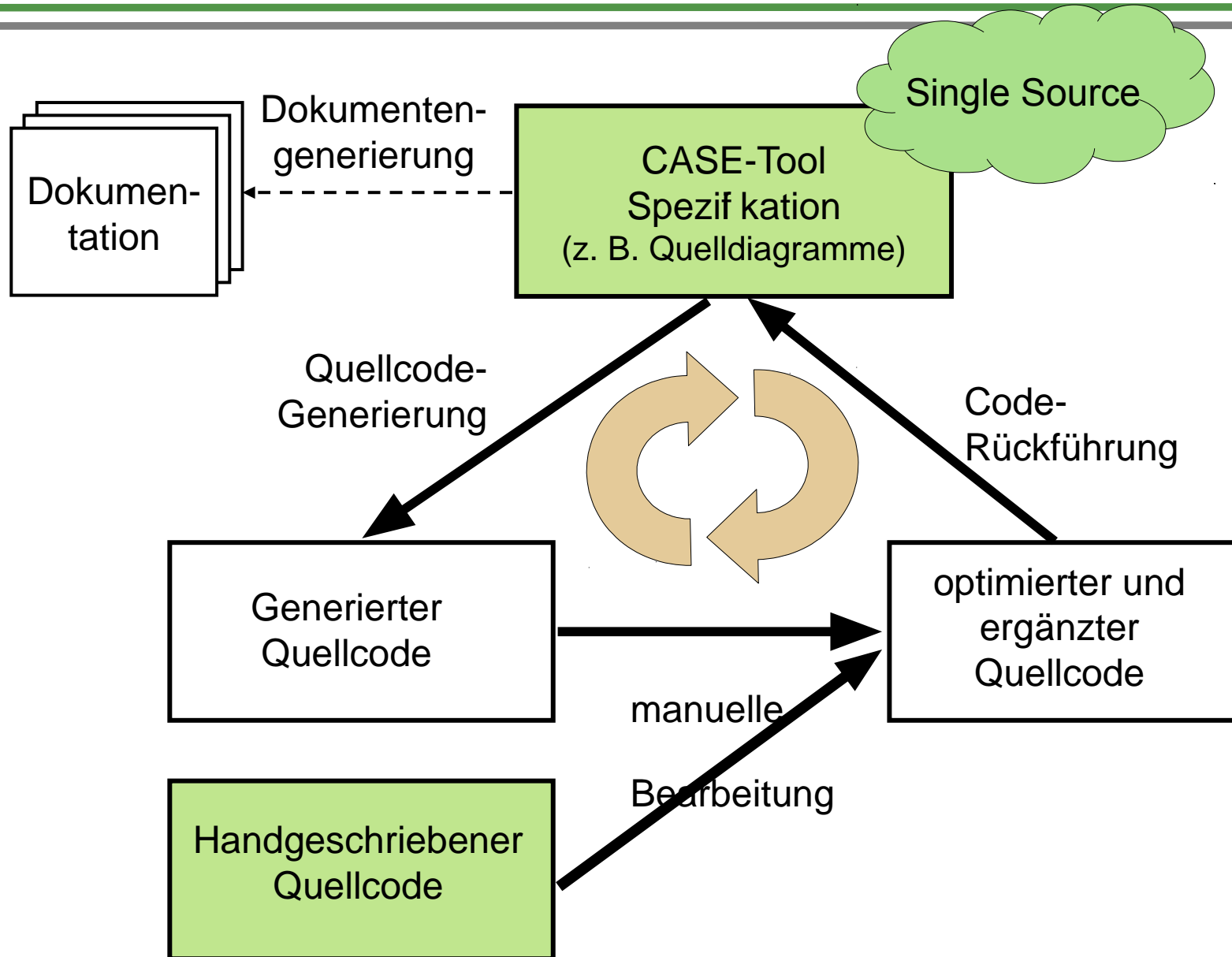
*Codegeneratorgener.*  
**BEG, cggg**

Codegenerierung

Endgültige Zuordnung des  
Codes der Zielsprache

Zielprogramm

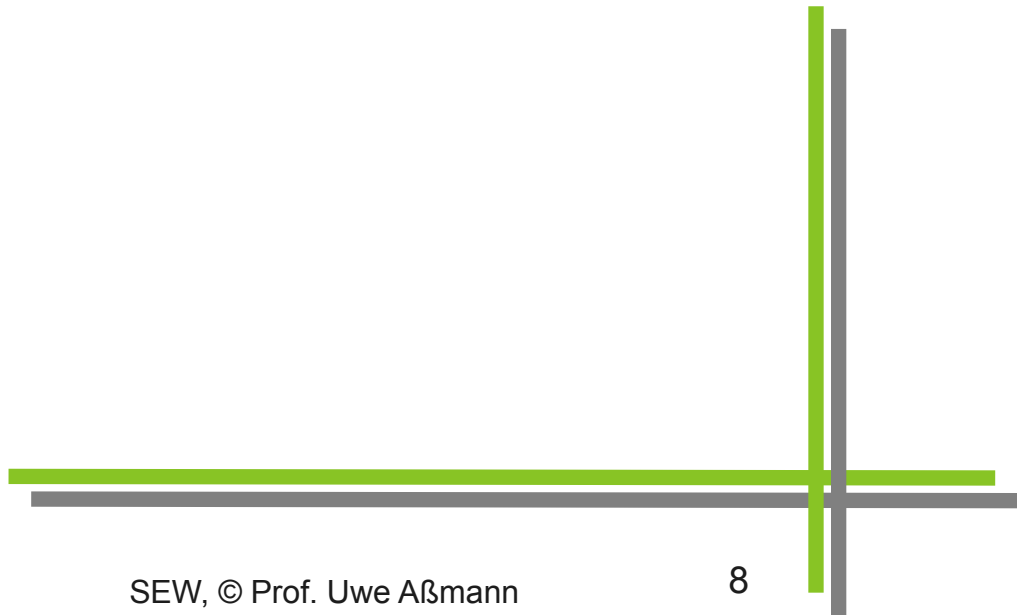
# Single-Source-Prinzip und Round-Trip Engineering



# Single Source Prinzip

- ▶ Eine **Single-Source-Technologie** gewährleistet zu jeder Zeit an jedem Ort absolute Konsistenz zwischen Modell, Code und Dokumentation
  - (ursprünglich von Peter Coad, Together-CASE-Werkzeug, jetzt Borland):
  - Vorhandensein nur einer definierten Quelle für alle Arbeiten
  - einheitlicher Ausgangspunkt für Spezifikation, Quellcode und Dokumentation (einschließlich Handbücher)
  - durch Bezeichner, Kommentare, Attribute, Markup oder ähnliches vorgegebene Struktur der Single Source
- ▶ Das Single-Source-Prinzip setzt **Round-Trip-Engineering** (RTE) voraus, mit
- ▶ **Codegenerierung**: automatisierte Codegenerierung in eine oder auch mehrere Programmiersprachen
  - Erzeugung von: Progr.-struktur, Bedingungen, Steuerfluss und Datendeklarationen
  - terminale Codeteile
- ▶ Templatebasierte Codegenerierung:
  - Einsetzen von Code-Fragmenten in Code-Schablonen
- ▶ **Coderückführung** (Re-Parsing) des geänderten Source-Codes des Quellprogramms in die Code-Teile der Spezifikation

# 45.1.1 Beispiele

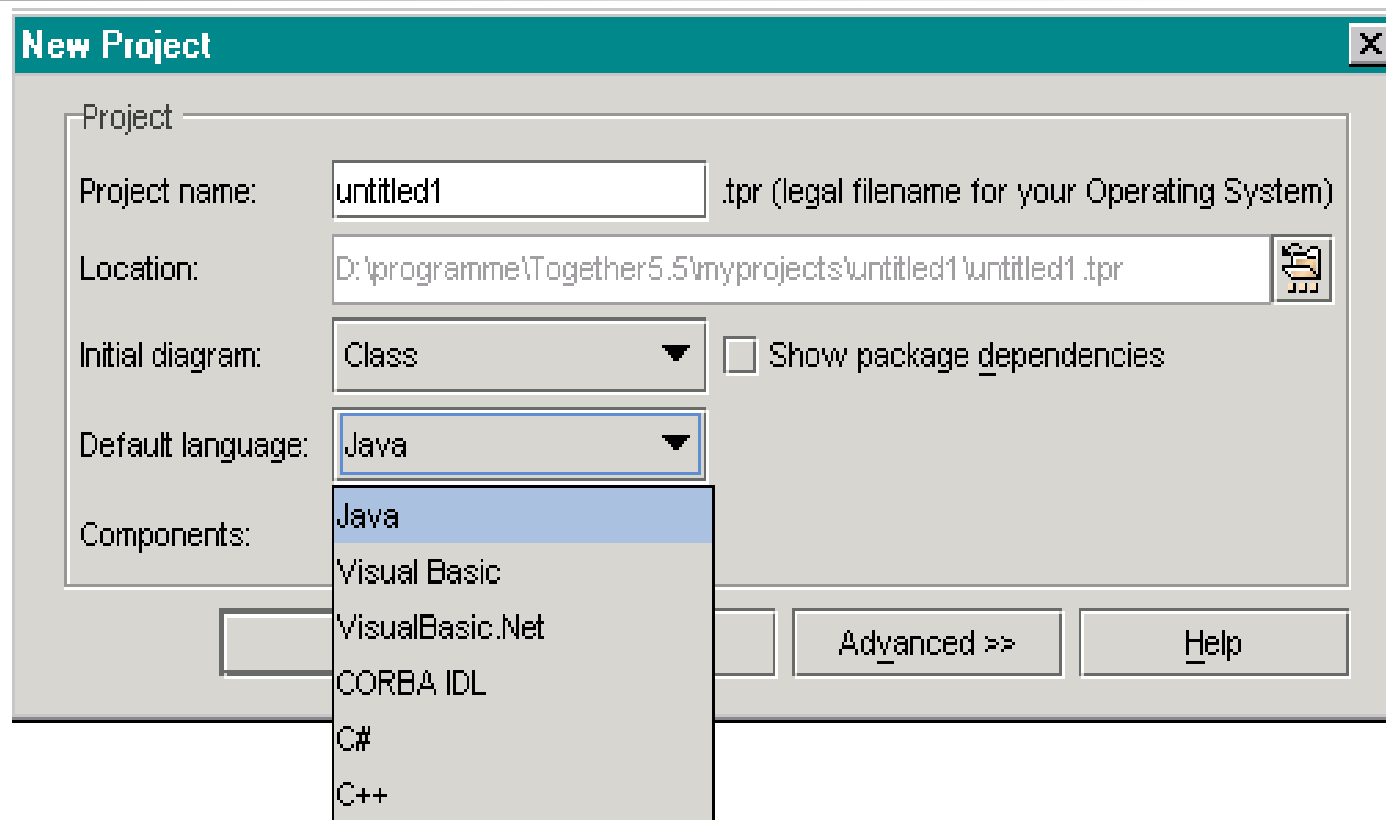




# Programmüberführung in Together (Coad)

- ▶ **Prinzip:** Single-Source-Technologie durch vollautomatische Synchronisation und vollständige Konsistenz zwischen Modell, Code und Dokumentation.
- ▶ **Zielsprachen:** Java, Visual Basic, VisualBasic.Net, CORBA IDL, C++, C#
- ▶ **Umsetzung:**
  - Multi-Language-Support durch Auswahl einer Programmiersprache(6) zu Beginn der Initialisierung eines neuen Projektes
  - UML-Modelling Editor ist nicht nur Werkzeug für den Entwurf von UML-Spezifikationen, sondern gleichzeitig werden diese inkrementell in die Syntax einer objektorientierten Programmiersprache überführt
  - Synchronisation erfolgt über Parser, nicht über Repository.
- ▶ **Bedienung:**
  - Simultanes Round-trip Engineering:**
    - Änderungen im Klassendiagramm werden unmittelbar im relevanten Source-Code angezeigt und umgekehrt
    - Reverse Engineering existierender Projekte zeigt die darin enthaltenen Programmstrukturen auch als UML-Diagramm

# Programmiersprachenauswahl in Together



- Basierend auf den Rollen: Business Modeler, Designer, Developer und Programmierer werden Sichten auf Arbeitsbereich automatisch konfiguriert (View-Management).
- Das Einbinden von Patterns, Templates und vorgefertigten source-basierten Frameworks (Komponenten incl. EJBs) wird unterstützt.
- Zur Qualitätssicherung werden Metriken und Audits angeboten.

# Together-Arbeitsbereiche

The screenshot displays the Together IDE interface. At the top is a menu bar with 'File', 'Edit', 'Object', 'Search', 'View', 'Options', 'Tools', and 'Help'. Below the menu is a toolbar with various icons for file operations and editing. On the left side, there is a project browser showing a tree structure under 'Demo' with sub-elements '<default>', 'Teilnehmer', and 'Person'. Below the project browser is the 'Properties of <default>' inspector, which has tabs for 'Description', 'HTMLdoc', and 'Requirements'. The 'Properties' tab is active, showing a table with columns 'Name' and 'Value'. The table contains the following entries:

Name	Value
diagram type	Class Diagram
name	<default>
package	<default>
stereotype	
alias	

At the bottom of the inspector, it says 'Press Ctrl+Enter to finish editing and close Inspector'. The main workspace is divided into two panes. The top pane shows a UML class diagram with two classes: 'Person' and 'Teilnehmer'. 'Person' has a private attribute '-attribute1:int' and a public operation '+operation1:void'. 'Teilnehmer' also has a private attribute '-attribute1:int' and a public operation '+operation1:void'. A solid line with an open arrowhead points from 'Teilnehmer' to 'Person', indicating inheritance. The bottom pane shows the Java source code for 'Teilnehmer.java':

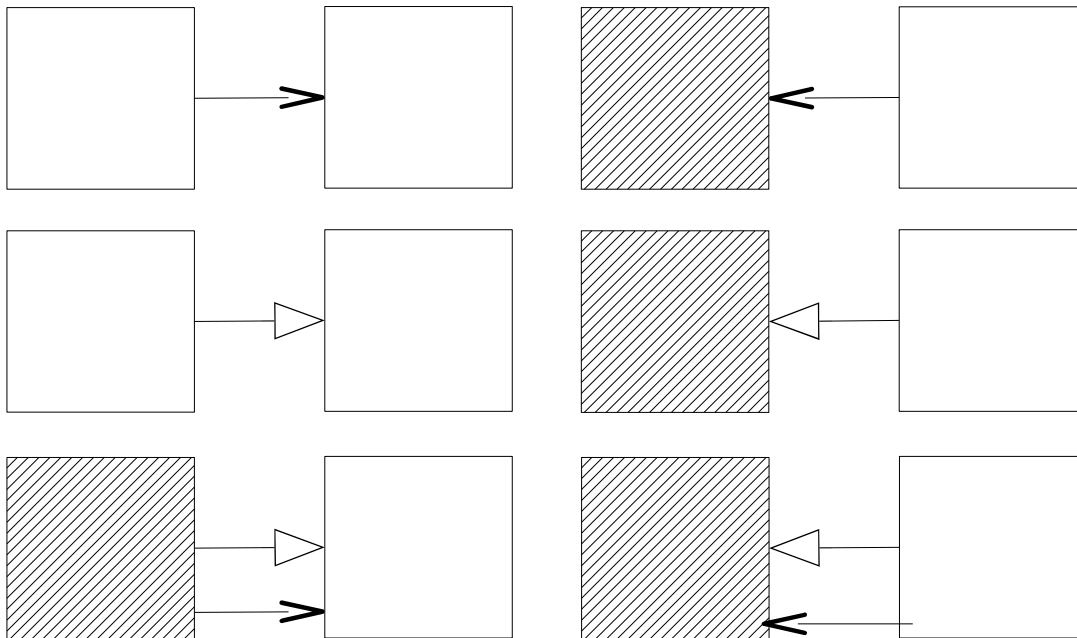
```
/* Generated by Together */  
  
public class Teilnehmer extends Person {  
    public void operation1() {  
    }  
  
    private int attribute1;  
}
```

# *45.2 Codegenerierungstechnologien*

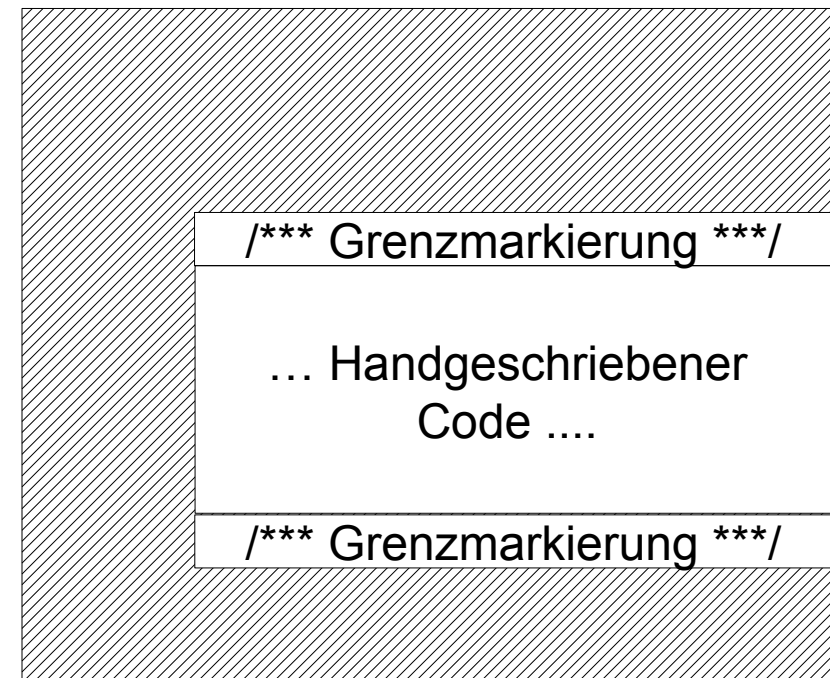


# Trennung von handgeschriebenem und generiertem Code

Kopplung mit Entwurfsmuster [Völter/Stahl]  
(separate Dateien, Klassenverknüpfungen  
wie Delegation, Vererbung, Composite,  
Decorator, etc)



Kopplung mit Trennmarkierung  
(hedge)



# Weitere Prinzipien der Codeselektion

- ▶ Ein **Codeselektor** ist ein Transformationssystem aus Term- oder Graphersetzungsregeln, der das Ausgangsprogramm oder -modell *abdeckt*, also jeden Knoten und Kante aus dem Ausgangsprogramm genau einmal transformiert
- ▶ Einsatz
  - Innerhalb der Zwischen- oder Assembler-Codegenerierung des Übersetzers
  - Als Back-End von CASE-Werkzeugen
- ▶ Ein **Codeanordner (code scheduler)** ordnet Befehle für den Chip in optimierter Reihenfolge an
  - Codeanordnung erfolgt meist nach der Codeselektion
- ▶ Ein **Schablonen-Expandierer (template expander)** generiert Code, in dem er Schablonen (templates) mit Werten aus der Programmrepräsentation oder dem Modell füllt (template-gesteuerte Codegenerierung)
- ▶ Ein **Invasiver Fragmentkompositor (invasive software composition)** komponiert Schablonen unter der Berücksichtigung von Fragmenttypen (s. CBSE)

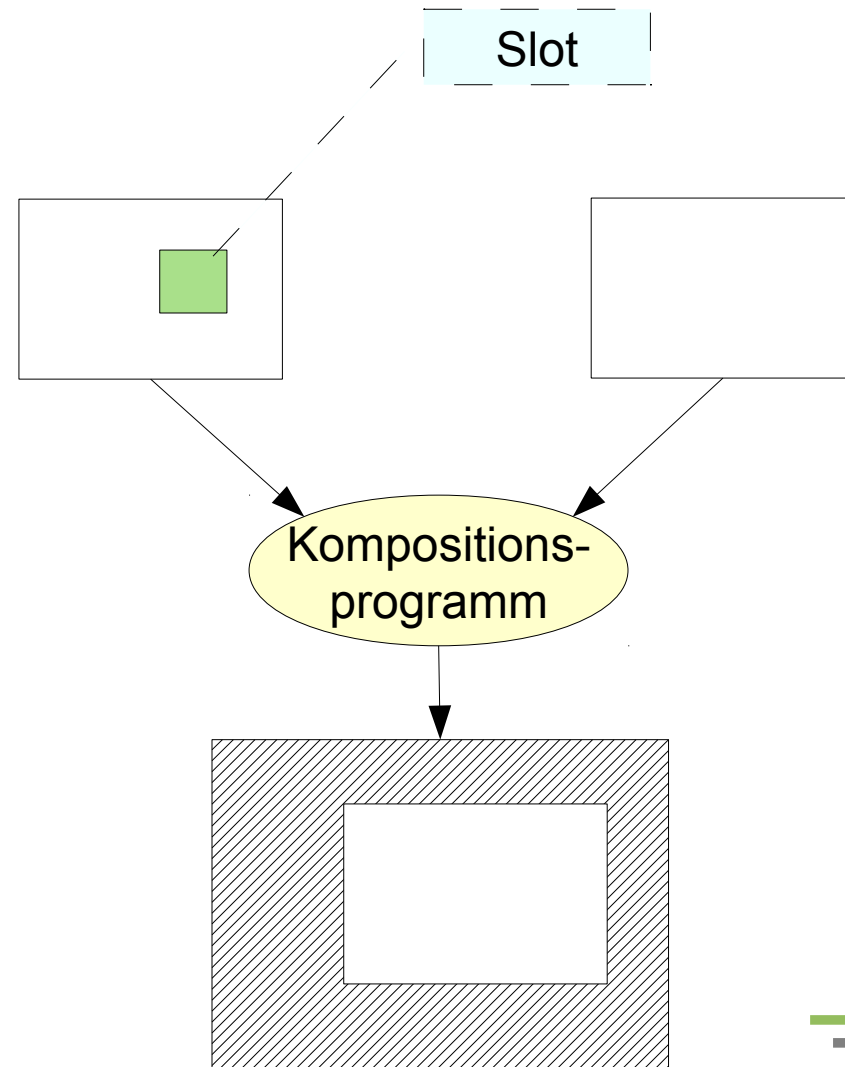
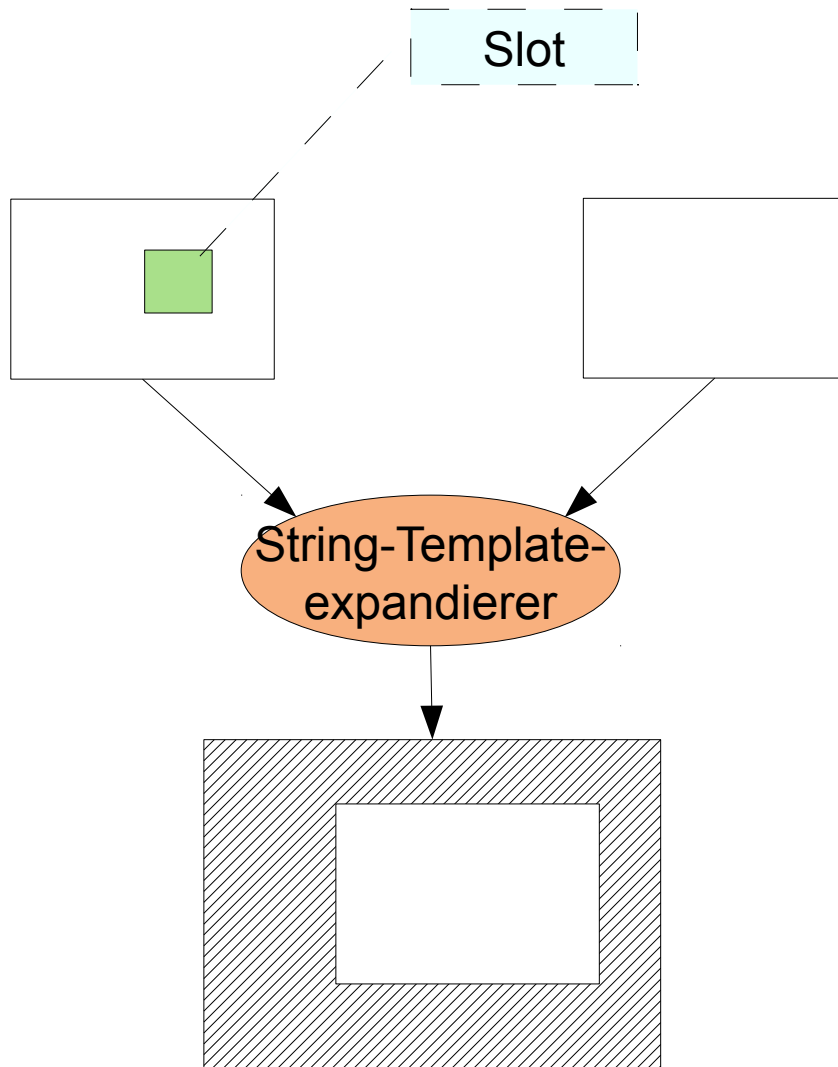
# ***45.2.1 Schablonenbasierte Programmüberführung (Template- based code generation)***



# Trennung von handgeschriebenem und generiertem Code

Kopplung durch Stringexpansion

Kopplung mit Kompositionsprogramm





## Slots are marked by Hedges

- ▶ Hedges are delimiters that do not occur in the base nor in the slot language

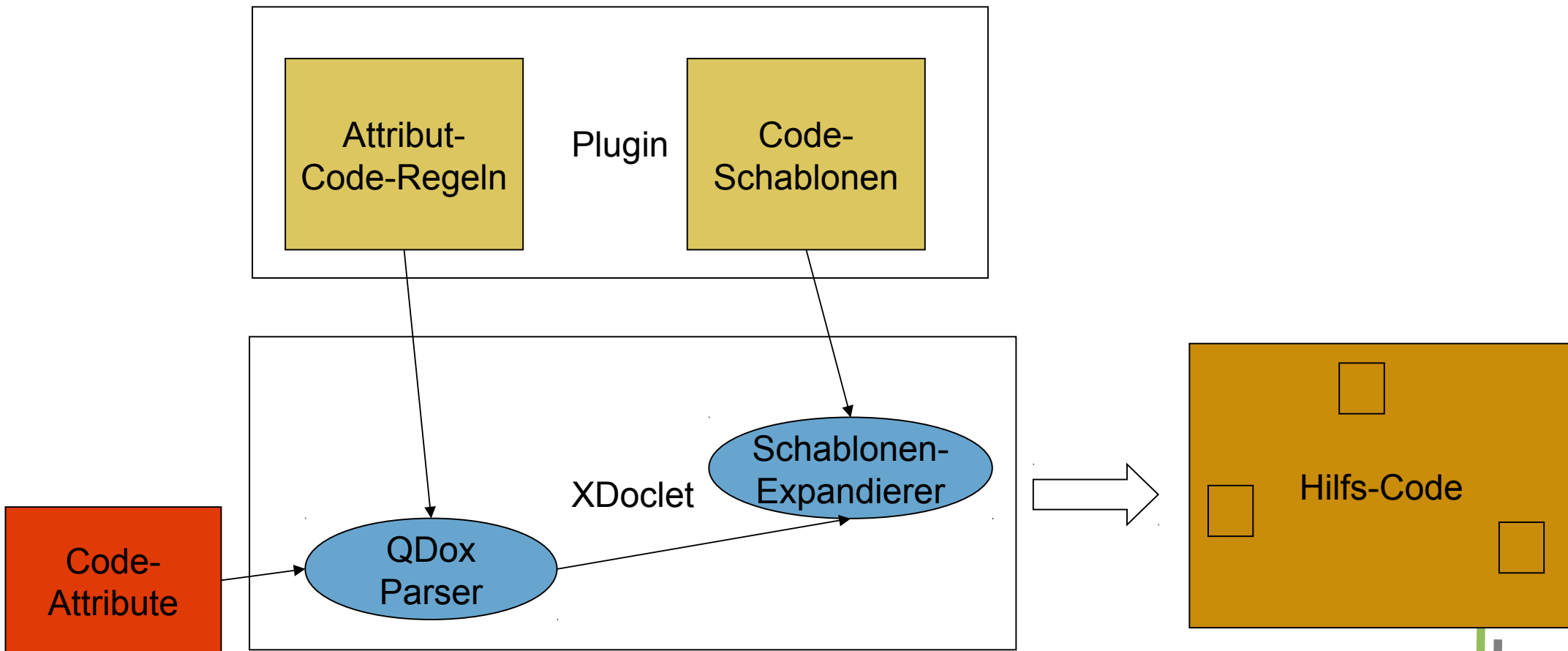
```
Template (superclass:CLASS, t:TYPE) {  
  class Worker extends << superclass >> {  
    << t>> attr = new <<t>>();  
    <<t>> getAttr();  
    void setAttr(<<t>>);  
  }  
}
```

# Tools für Schablonenexpansion

- ▶ Untypisierte Schablonen-Expansion:
  - Frame processing (Bassett)
    - C Holmes, A Evans. A review of frame technology. University of York, Dept. of Computer Science, 2003
    - <https://www.cs.york.ac.uk/ftpdireports/YCS-2003-369.pdf>
  - String template engines
    - Apache Velocity
    - Parr's template engine StringTemplate
    - Jenerator for Java <http://www.voelter.de/data/pub/jeneratorPaper.pdf>
- ▶ Metamodel-controlled template engines
  - Open Architecture Ware's Scripting language
- ▶ Invasive Softwarekomposition bietet volltypisierte Schablonenexpansion (siehe CBSE)
  - Getypte Schablonen-Expansion und -erweiterung
  - Kann für beliebige Programmiersprachen instantiiert werden
  - <http://www.the-compost-system.org>
  - <http://www.reuseware.org>

# Xdoclet ([xdoclet.sf.net](http://xdoclet.sf.net))

- ▶ Xdoclet wandet Attribute (Metadaten) in Code um
  - Schablonen-gesteuerte Codegenerierung



## ***45.3 Codemodifikation und -rückführung***



# Vorgehen der Coderückführung

- ▶ **Aufgabe:** Erkennen geänderter „Code“-Teile und Rückführung in die Entwurfsmodelle
- ▶ **Prinzip:** Die modifizierte Quellcodedatei stammt in jedem Fall aus der Single-Source-Spezifikation eines CASE-Tools, in die der geänderte Programmcode zurückgeführt werden soll
  - Kennzeichnungen der Single Source-Spezifikation sind noch vorhanden.
  - Strukturierung der Quellcodefiles ist so, dass Abschnitte erkennbar sind und ihnen eindeutig die Objekte der Entwurfsspezifikation zugeordnet werden können, beispielsweise durch:
    - Trennmarkierungen (-kommentare oder -attribute, hedges) zwischen den Abschnitten (Markup) wird zum Erkennen der Grenzen benutzt
    - Vorhandensein von „Code“-Teilen als zielsprachenspezifische Freiräume (hooks)
    - Weitere Rückführinformationen gegebenenfalls aus dem Quellfilekopf oder -kommentaren

# Beispiel-Folien

- ▶ Beispiel aus der Codegenerierung und -Rückführung von Fujaba:  
[http://www.fokus.fraunhofer.de/en/fokus\\_events/motion/ecmda2008/\\_docs/rs01\\_t03\\_ManuelBork\\_EMCCA2008\\_slides.pdf](http://www.fokus.fraunhofer.de/en/fokus_events/motion/ecmda2008/_docs/rs01_t03_ManuelBork_EMCCA2008_slides.pdf)
- ▶ Paralleles Parsen von Template und Generat, mit Vergleich zum Auflösen der Indeterminismen der Rückführung



*The End*