

# 63. Werkzeuge zur Programmüberführung aus Modellen (Metaprogramming, Code Generation and Round-Trip Engineering)

1

Prof. Dr. Uwe Aßmann  
Technische Universität Dresden  
Institut für Software- und  
Multimediatechnik  
<http://st.inf.tu-dresden.de>  
Version 12-1.0, 23.01.13

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

- ▶ <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)
- ▶ Paul Bassett. Frame-based software engineering. IEEE Software, 4(4):9-16, 1987.
  - <http://doi.ieeecomputersociety.org/10.1109/MS.1987.231057>
- ▶ Chris Holmes, Andy Evans. A review of frame technology. University of York, Dept. of Computer Science, 2003  
<ftp://www-users.cs.york.ac.uk/reports/2003/YCS/369/YCS-2003-369.pdf>
- ▶ Daniel Weise and Roger Crew. Programmable syntax macros. In Proceedings of the ACM SIGPLAN '93 Conference on Programming Language Design and Implementation, pages 156-165, Albuquerque, New Mexico, June 23-25, 1993.
- ▶ Optional
  - Völter, Stahl: Model-Driven Software Development, AWL 2005.

# 63.1 Model2Code Translation (Code Generation)

3

Überführung von Modellen in Programme  
(Programmüberführung)



# CASE-Code-Generatoren

4

Quelldiagramm

Zielprogramm

Graph. Modellspezifikation  
UML-Diagramme  
SD-Modulchart  
(ERD, DFD, ...)

Textuelle Spezifikation  
(OCL, Skripte, Templates)

Spezifikation der  
Zielsprache



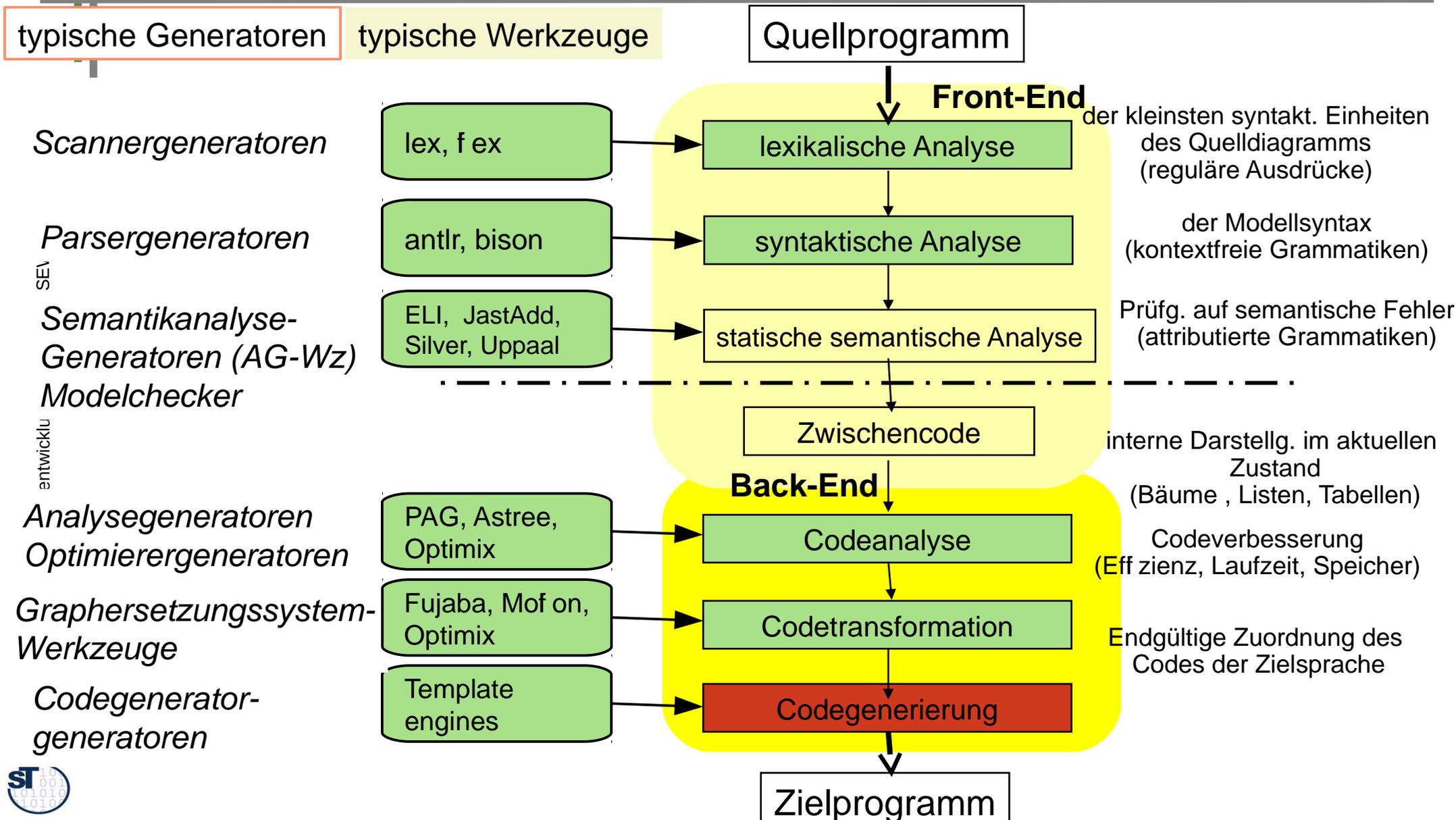
Spezifikation des  
Übersetzungsvorganges, z.B.  
Auswahl Wurzelendiagramm,  
Modelltiefe,  
Startvariable u. a.

Zielsprache  
Codegerüst  
•  
•  
•

vollständig/  
lauffähig



# Phasen eines Werkzeugs und ihre erzeugenden Werkzeuge



# Kinds of Code Generators

6

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

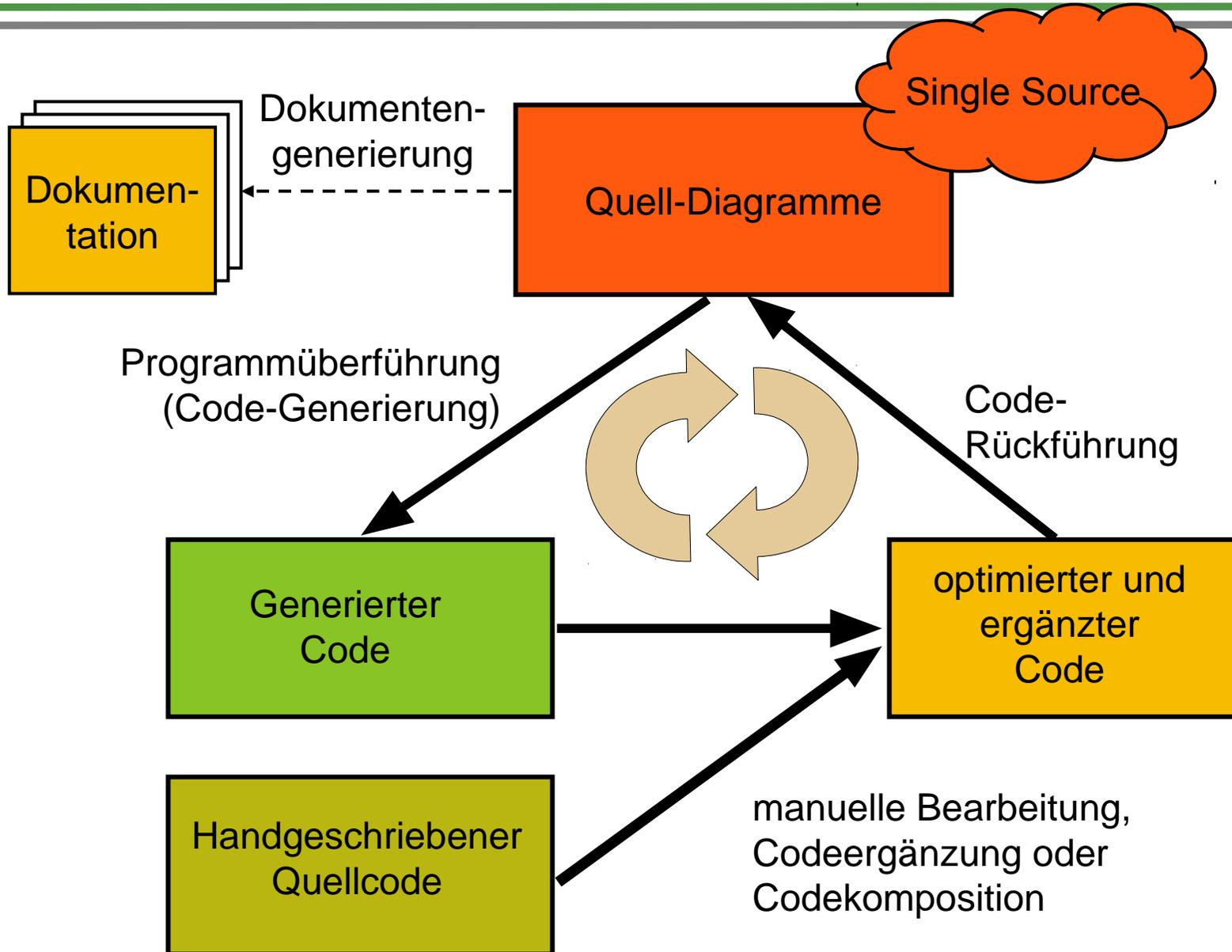
# 63.1.1 Single-Source Principle



7

# Single-Source-Prinzip, Code-Ergänzung und Round-Trip Engineering

8



- ▶ 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

# Programmüberführung in *Together* (Coad)

10

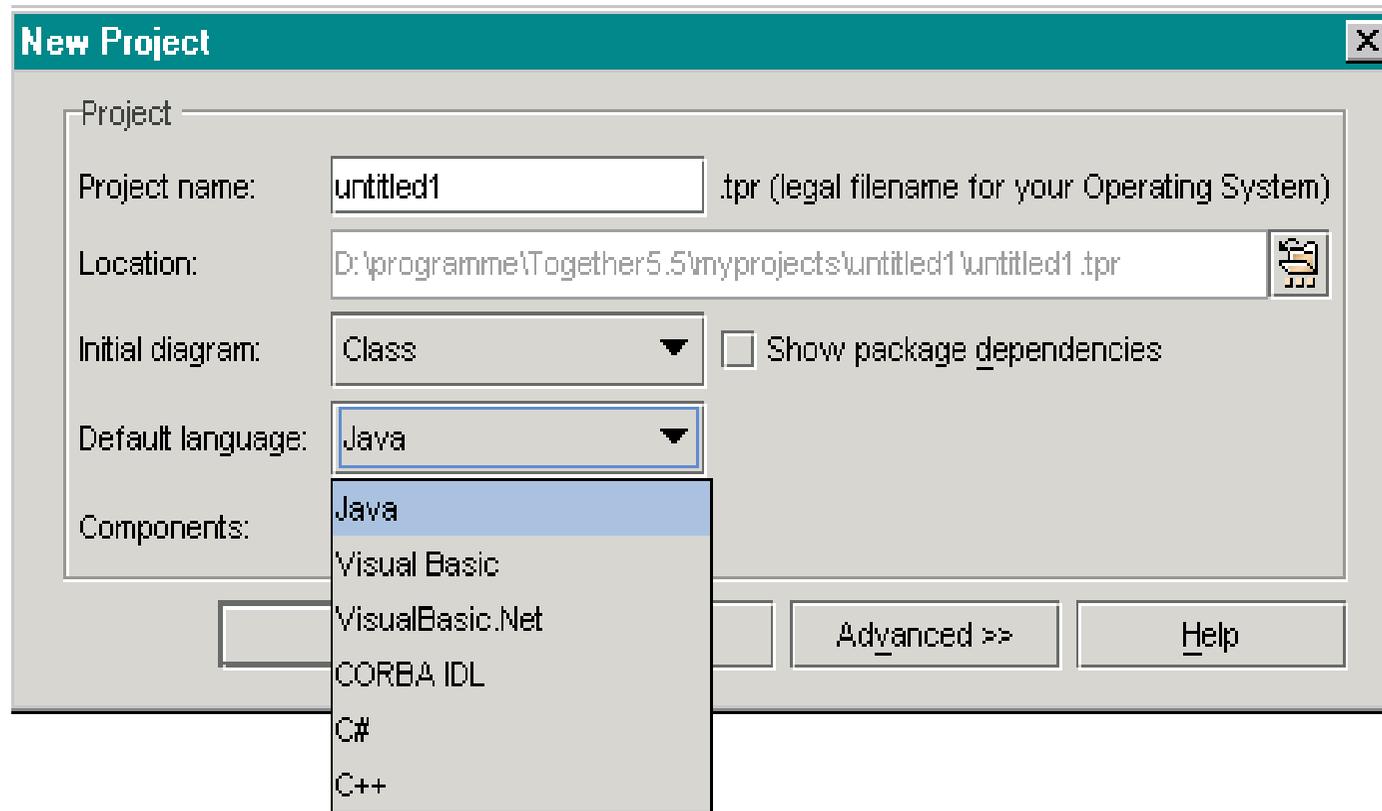
- ▶ Prinzip: Single-Source-Technologie durch vollautomatische Synchronisation und vollständige Konsistenz zwischen Modell, Code und Dokumentation.
- ▶ Zielsprachen: Java, Visual Basic, VisualBasic.Net, CORBA IDL, C++, C#
- ▶ Umsetzung:
  - Multi-Language-Support durch Auswahl einer Programmiersprache(6) 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
- [http://www.borland.com/downloads/download\\_together.aspx](http://www.borland.com/downloads/download_together.aspx)

# Programmiersprachenauswahl in Together

11



- 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

12

The screenshot displays the Together IDE interface. On the left, a project tree shows a package named 'Demo' containing two classes: '<default>' and 'Teilnehmer'. Below the tree is the 'Properties of <default>' inspector, which shows details for the selected class diagram.

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

The main workspace 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 generalization arrow points from 'Teilnehmer' to 'Person', indicating that 'Teilnehmer' inherits from 'Person'.

Below the diagram, the source code for 'Teilnehmer.java' is shown, which is automatically generated by Together:

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

# 63.2 Codegenerierungs-Technologien



13

- ▶ **Metaprogramme (Reflektive Programme)** liefern Code, besitzen also ein Metamodell oder Grammatik als Typsystem
- ▶ **Metaprogramm-Prozeduren** (Semantische Macros, Programmable Macros [Weise/Crew]) sind durch das Metamodell bzw. die Grammatik typisiert:
  - Ihre Parametertypen sind Metaklassen oder Nicht-Terminale
  - Sie haben als Rückgabewert eine Metaklasse oder Nicht-Terminal

# Trennung von handgeschriebenem und generiertem Code mit anschließender Code-Composition

15

- ▶ In separaten Dateien:
- ▶ Kopplung mit Entwurfsmuster [Völter/Stahl]
- ▶ Hier werden Klassenverknüpfungen wie Delegation, Vererbung, Composite, Decorator, etc als Code-Kompositionsoperatoren benutzt

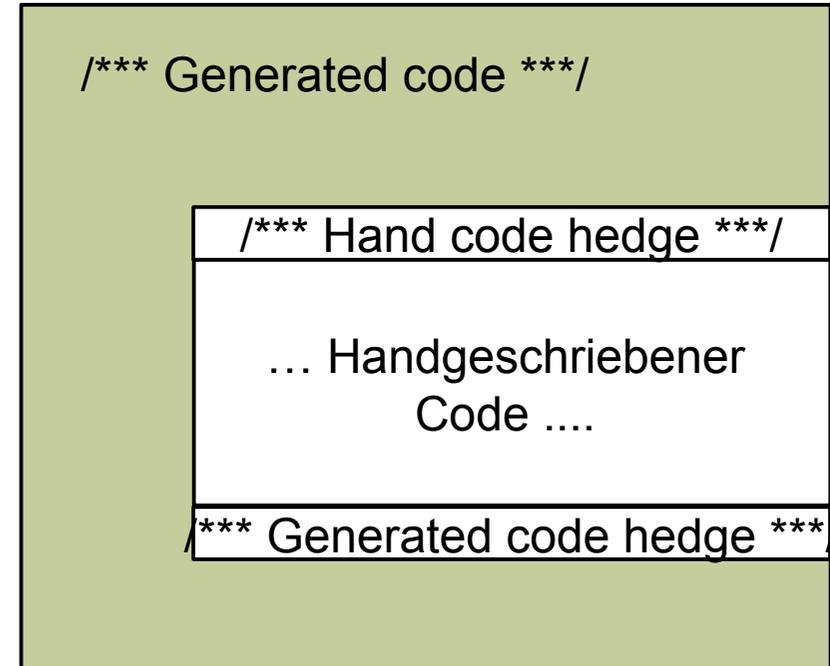
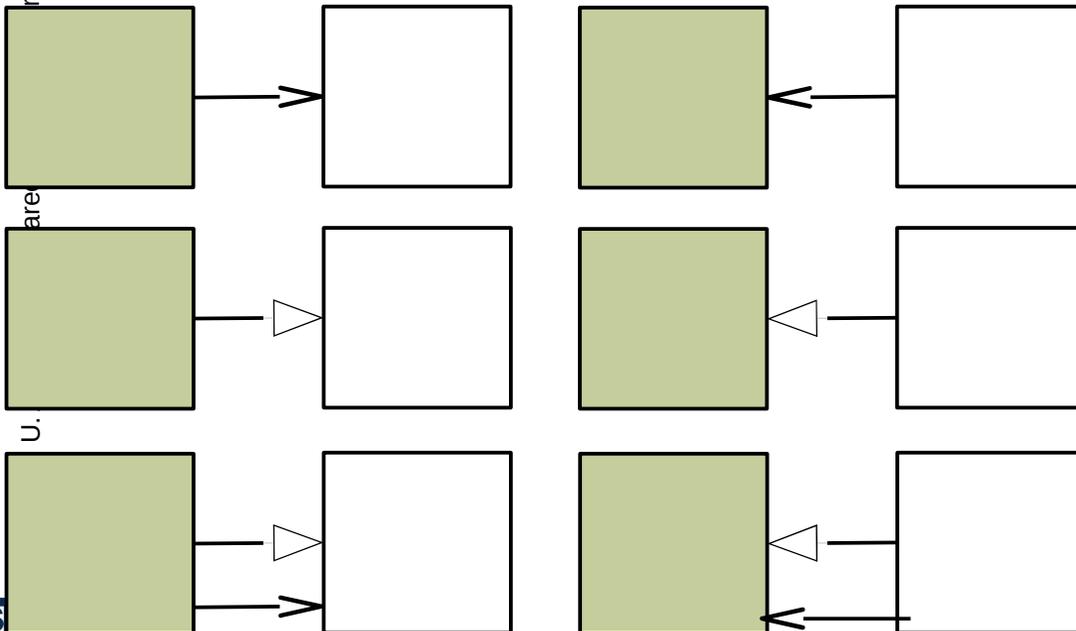
In einer einzigen Datei:

Kopplung mit **Trennmarkierung (hedge)**

Werkzeuge (SEW)

are

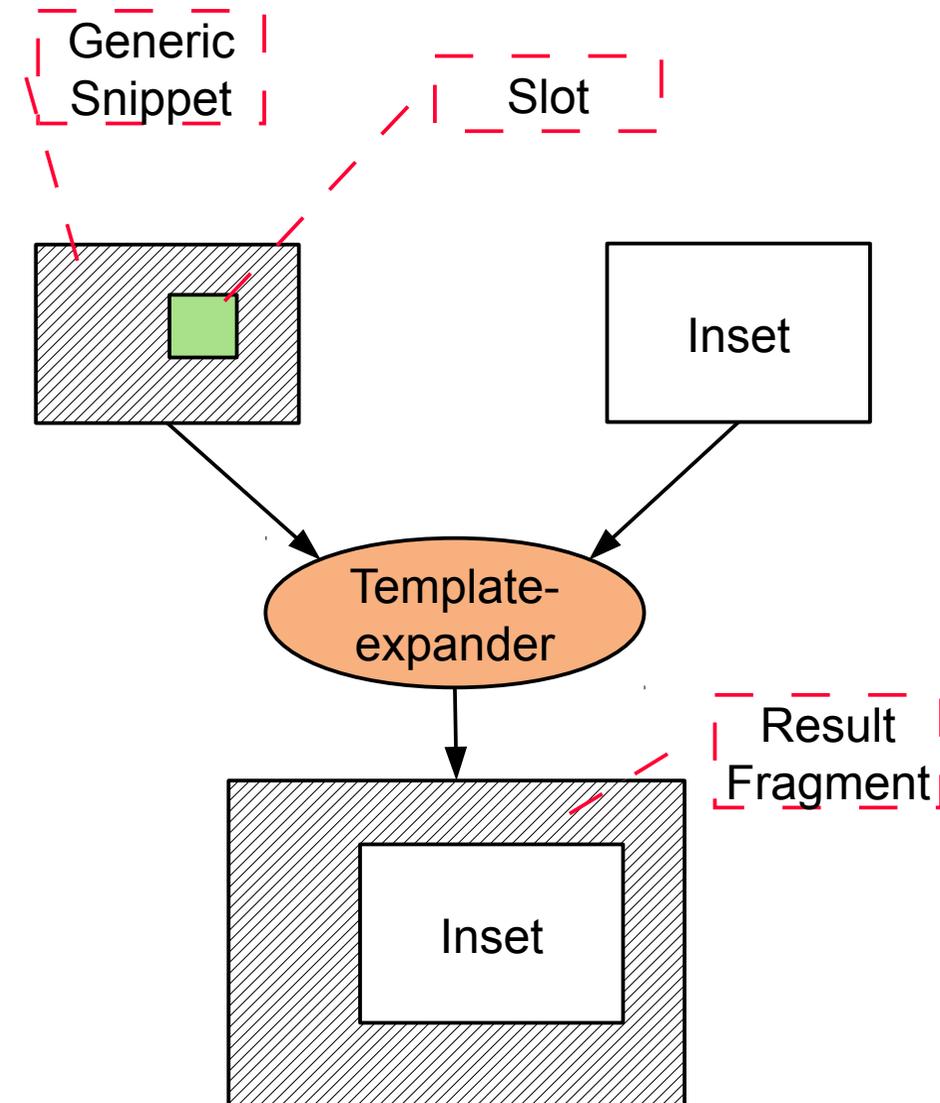
u.



# Snippet Programming

16

- ▶ A **fragment (snippet)** is an incomplete sentence of a language, derived from a nonterminal of the grammar, or described by a metaclass
- ▶ A **generic fragment (template, form, frame)** is a fragment with **slots (holes, code parameters, variation points)**, which can be *bound (filled, expanded)* with an **inset fragment** to a **result fragment**
- ▶ An **extensible fragment** is a fragment with **hooks (extension points)**, which can be *extended* to a fragment
- ▶ **Generic programming** is programming with generic fragments (templates).
- ▶ **Invasive programming** is programming with generic and extensible fragments (templates with hooks)



# 63.2.1 Schablonenbasierte Programmüberführung (Template-based code generation)

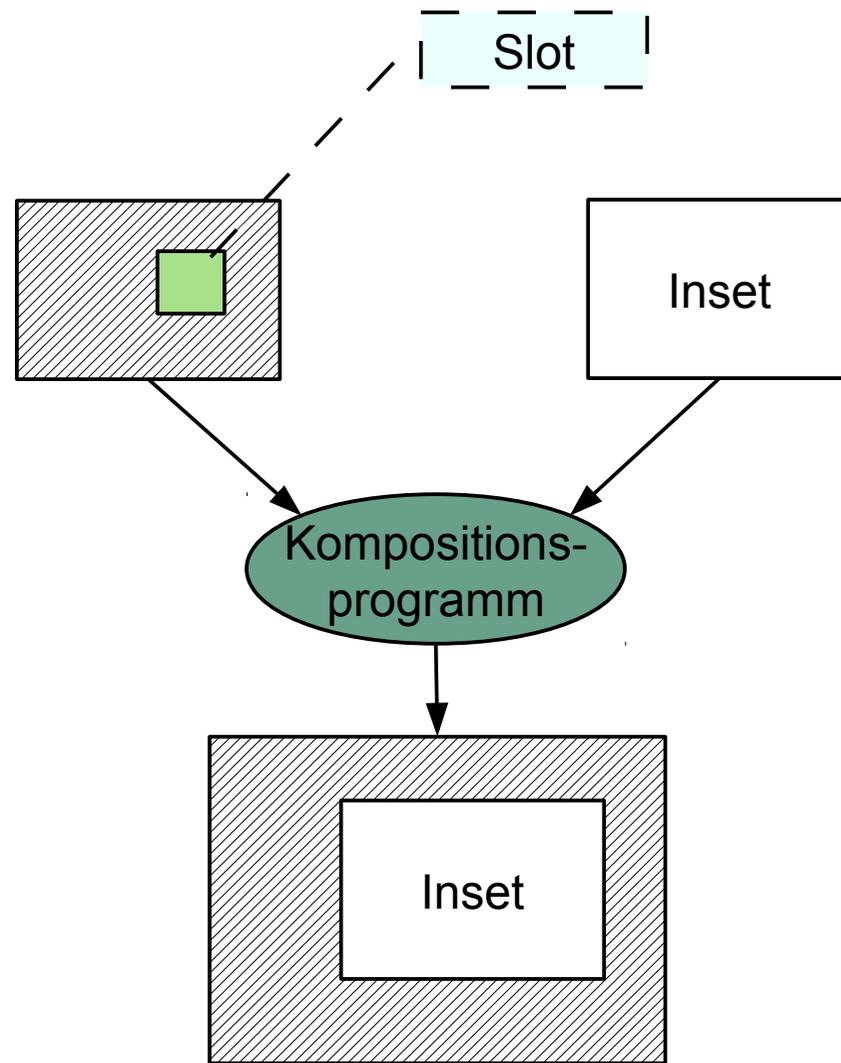
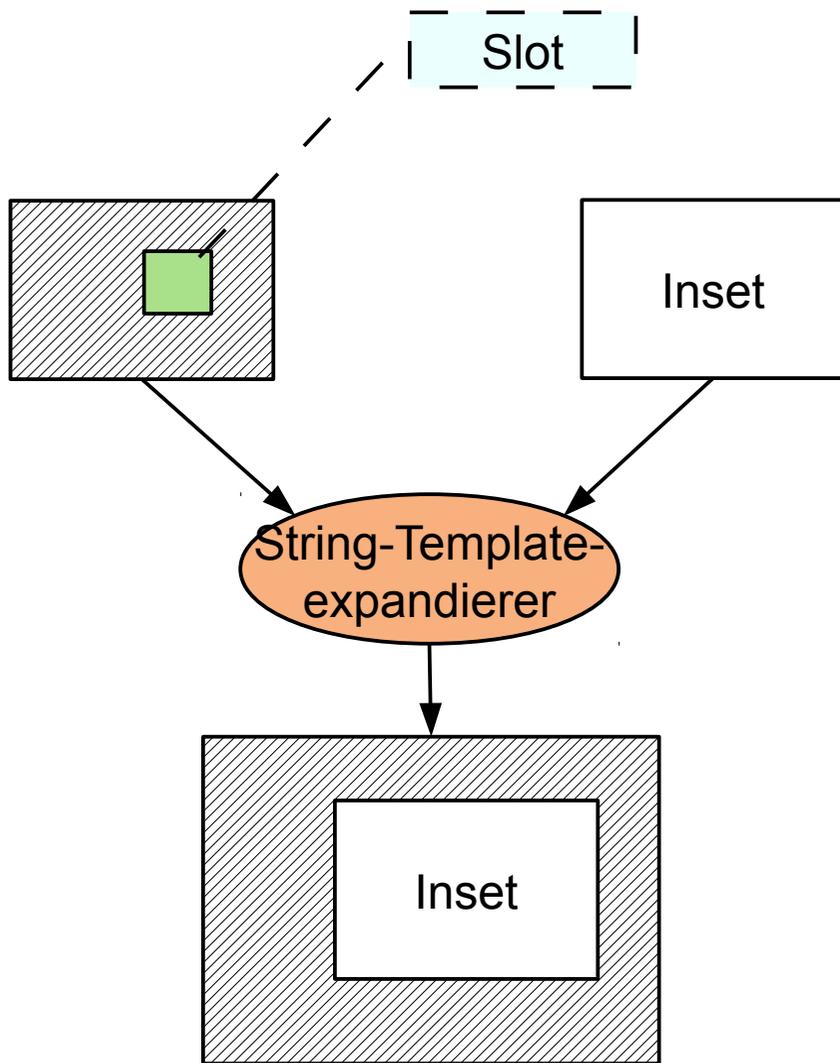
17

# Trennung von handgeschriebenem und generiertem Code

18

Kopplung durch Stringexpansion

Kopplung mit Kompositionsprogramm



# Slots are marked by Hedges

19

- ▶ **Hedges** are delimiters that do not occur in the base nor in the slot language
- ▶ **Slot hedges** are template2slot hedges marking the transition from the code language to the slot language
- ▶ **Inset hedges** are metaprogramming2code hedges marking the transition from the metaprogramming language to the code language

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

# Tools for Untyped Template Expansion

20

- ▶ **Frame processing** was invented in [P. Bassett] as an *untyped string template expansion technology*, universal for all textual languages [Holmes/Evans]
  - Frame processing is the main technology for web engineering today: it organizes reuse of page templates
  - The original frame processor used \$ as a hedge symbol for slots (slot variables)
- ▶ **Macro processing** is not much different
  - Because only slot variables hold insets, macro parameters correspond to slot variables
- ▶ XML template engine XVCL [Jarzabek] is an XML-controlled frame processor
  - <http://sourceforge.net/projects/fxvcl/files/XVCL%20Specification/Version%202.10/>
- ▶ String template engines in use today
  - Apache Velocity <http://velocity.apache.org/>
  - Parr's template engine StringTemplate
  - Jenerator for Java <http://www.voelter.de/data/pub/jeneratorPaper.pdf>

# Velocity String Template Language

21

- ▶ Velocity Template Language (VTL) is a frame processing language with metaprograms in slots
- ▶ {#, \$} are slot hedges
- ▶ < (from XML) is the inset hedge

```
<html>
<body>
#set( $foo = "Velocity" )
Hello $foo World!
</body>
</html>
```

```
<HTML>
<BODY>
Hello $customer.Name!
<table>
#foreach( $mud in $mudsOnSpecial )
  #if ( $customer.hasPurchased($mud) )
    <tr>
      <td>
        $flogger.getPromo( $mud )
      </td>
    </tr>
  #end
#end
</table>
```

# Velocity Template Language

22

- ▶ Velocity Template Language (VTL) is a simple scripting language in the spirit of TCL
- ▶ It has control structures (if, switch, foreach), assignments (set), and macros

<http://velocity.apache.org/engine/releases/velocity-1.7>

```
#macro( inner $foo )
  inner : $foo
#end

#macro( outer $foo )
  #set($bar = "outerlala")
  outer : $foo
#end

#set($bar = 'calltimelala')
#outer( "#inner($bar)" )
```

**Problem: the result of string template expansion may not be syntactically correct, nor well-formed, target language (error-prone)**

# Typed Template Expansion

23

- ▶ 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>

# Semantic Macros

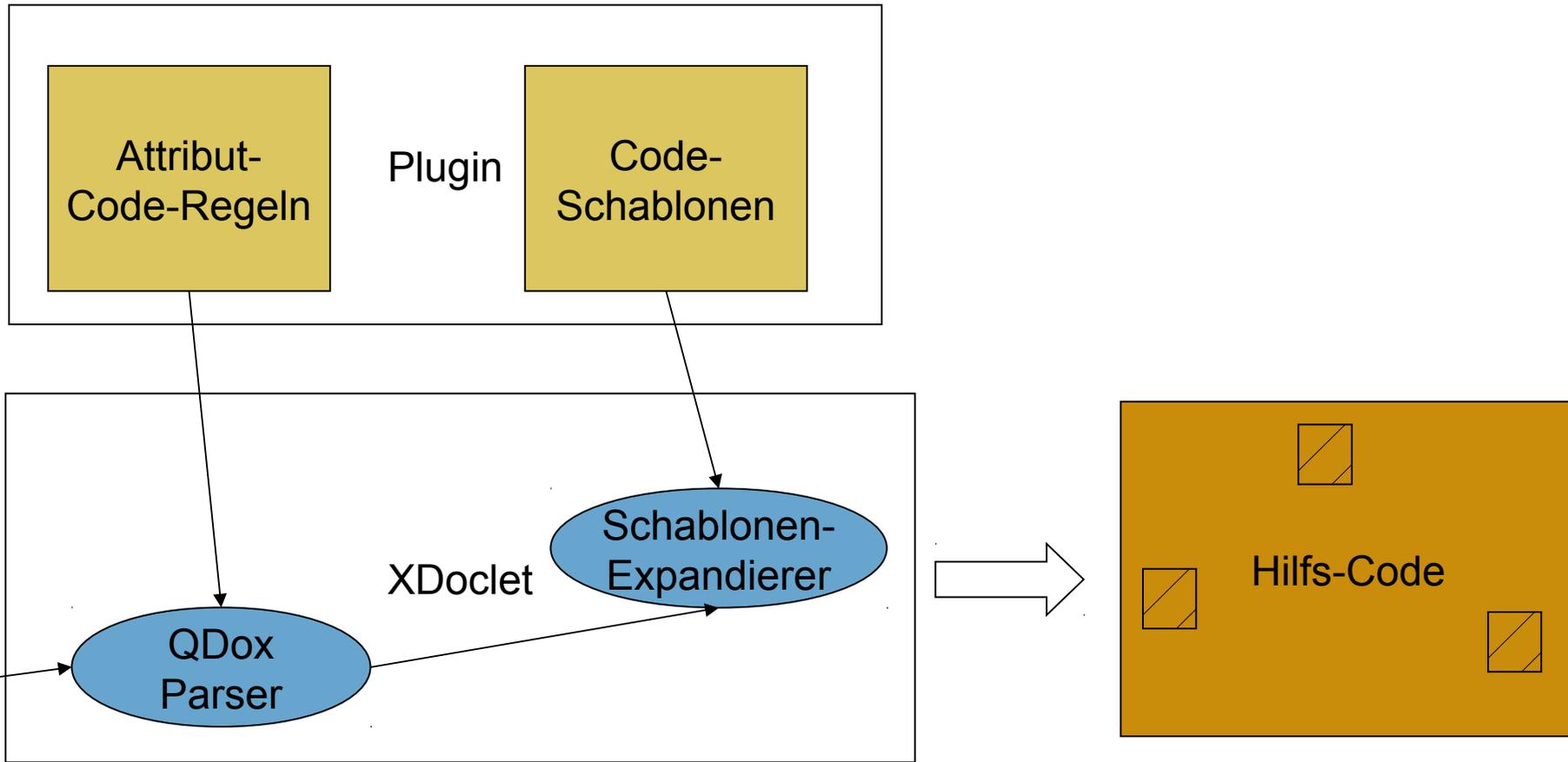
24

- ▶ **Semantic Macros** are metaprogramming procedures which are typed parameters and results.
  - They allow for type-safe static metaprogramming.
- ▶ Examples:
  - Scheme

# Xdoclet (xdoclet.sf.net)

25

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



## 63.3 Codemodifikation und -rückführung

27

# Vorgehen der Coderückführung

28

- ▶ **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

- ▶ **Trace hedges** are hedge symbols inserted by a template expander to demarcate the template from the inset.

- ▶ Beispiel aus der Codegenerierung und parser-basierten Code-Rückführung von Fujaba:

[http://www.fokus.fraunhofer.de/en/fokus\\_events/motion/ecmda2008/\\_docs/rs01\\_t03\\_ManuelBork\\_EMCD2008\\_slides.pdf](http://www.fokus.fraunhofer.de/en/fokus_events/motion/ecmda2008/_docs/rs01_t03_ManuelBork_EMCD2008_slides.pdf)

- ▶ Paralleles Parsen von Template und Generat, mit Vergleich zum Auflösen der Indeterminismen der Rückführung

**The End**