Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

The Twelfth International Conference on Advances in Databases, Knowledge, and Data Applications
DBKDA 2020
September 27, 2020 to October 01, 2020 - Lisbon, Portugal

Tutorial: Codegeneration for Database Developers

Andreas Schmidt[1,2] and Steffen G. Scholz[2]

(1)
andreas.schmidt@hs-karlsruhe.de
Faculty of Computer Science and Business Information Systems
University of Applied Sciences Karlsruhe
Germany

(2)
{ schmidt | scholz }@kit.edu
Institute for Automation and Applied Informatics
Karlsruhe Institute of Technologie
Germany

Karlsruhe Institute of Technology

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

A short Resume of the Presenters

Prof. Dr. **Andreas Schmidt** is a professor at the Department of Computer Science and Business Information Systems of the Karlsruhe University of Applied Sciences (Germany). He is lecturing in the fields of database information systems, data analytics and model-driven software development. Additionally, he is a senior research fellow in computer science at the Institute for Applied Computer Science of the Karlsruhe Institute of Technology (KIT). His research focuses on database technology, knowledge extraction from unstructured data/text, Big Data, and generative programming. Andreas Schmidt was awarded his diploma in computer science by the University of Karlsruhe in 1995 and his PhD in mechanical engineering in 2000. Dr. Schmidt has numerous publications in the field of database technology and information extraction. He regularly gives tutorials on international conferences in the field of Big Data related topics and model driven software development. Prof. Schmidt followed sabbatical invitations from renowned institutions like the Systems-Group at ETH-Zurich in Switzerland, the Database Group at the Max-Planck-Institute for Informatics in Saarbrucken/Germany and the Data-Management-Lab at the University of Darmstadt.

Dipl.-Ing Dr. **Steffen G. Scholz** has more than 18 years of R&D experience in the field of polymer micro & nano replication with a special focus on injection moulding and relevant tool-making technologies. He is an expert in process optimization and algorithm design and development for micro replication processes. He studied mechanical engineering with special focus on plastic processing and micro injection moulding and obtained his degree as from the University of Aachen (RWTH). He obtained his PhD from Cardiff University in the field of process monitoring and optimization in micro injection moulding and led a team in micro tool making and micro replication at Cardiff University. Dr. Scholz joined KIT in 2012, where he is now leading the group for process optimization, information management and applications (PIA)

Karlsruhe Institute of Technology

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Research Interests

- For PIA Group at KIT see https://www.iai.kit.edu/english/941.php
- Additionally, all sort of database related stuff, like
  - Database Implementation
  - Graph databases
  - Semantic Text Analysis
  - Information Retrieval
  - ...

Karlsruhe Institute of Technology

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Purpose

- Present the principal architecture and functioning of a code generator
- Enable the participants to
  - ... use existing code generators
  - ... build generators on their own

Karlsruhe Institute of Technology

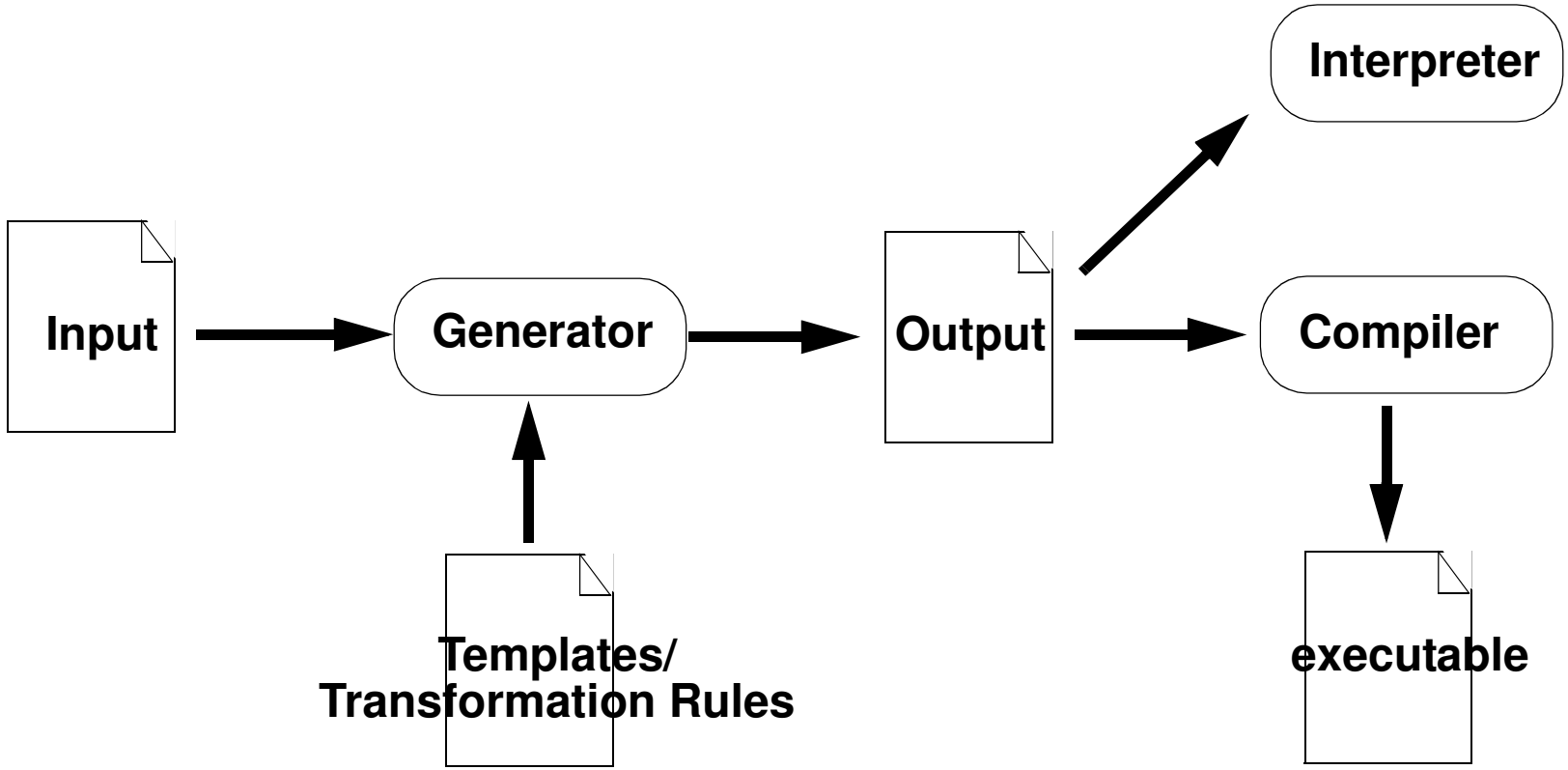
Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Outline

- Introduction and Motivation
- Regular Expressions
- Overview of different generator technologies
- Steps towards a general purpose generator
- Summary and next steps

What is a Software Generator

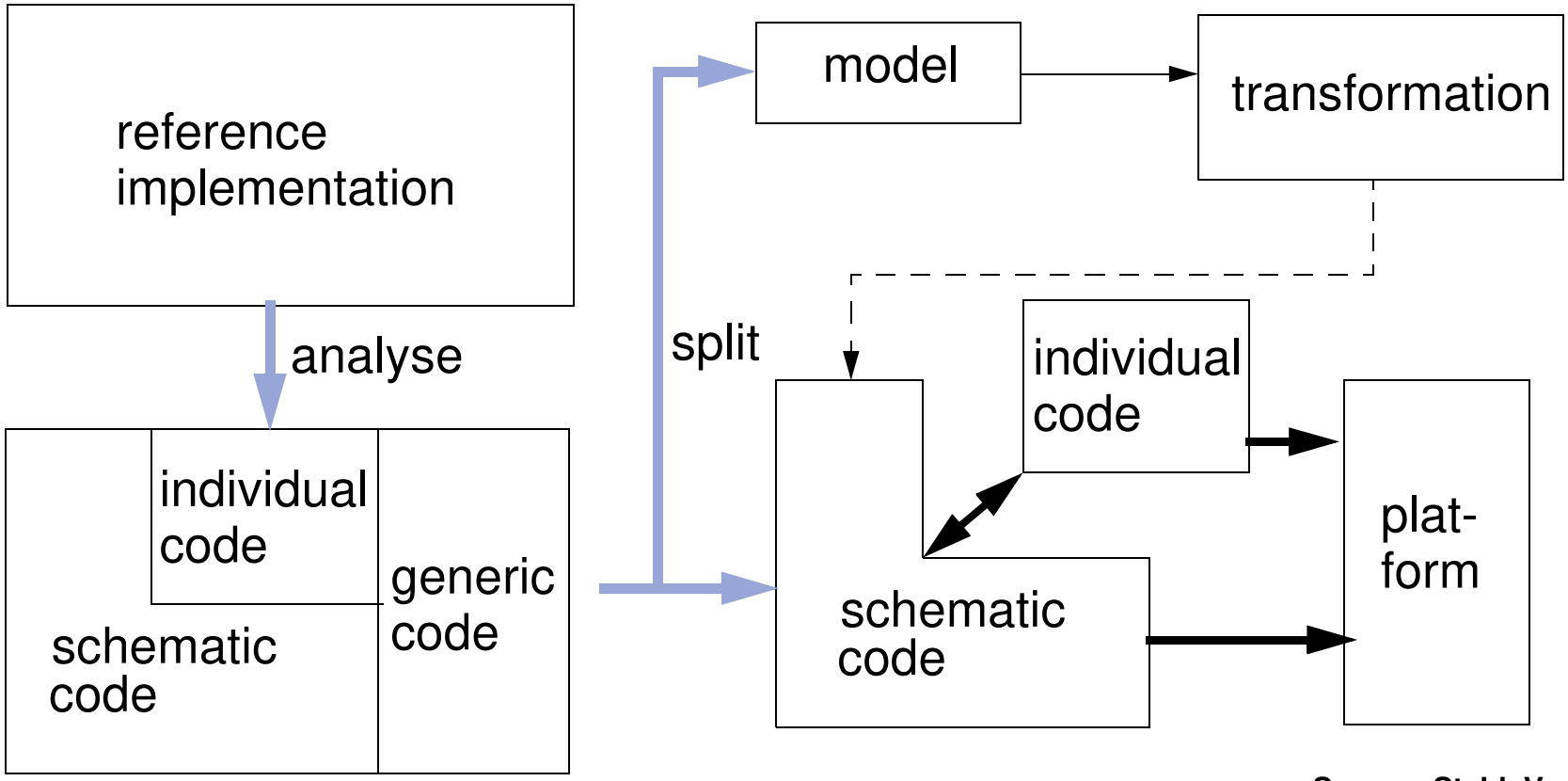
Principle



Model Driven Software Development

- Partial or whole generation of programs, based on a formal model
- Model represents the problem space of the application
- Models could be transformed in other models or into source code
- Model representation:
  - Abstract and formal description (without implementation details) of a problem space
  - Notation:
    - text
    - xml
    - graphical representation

Concept of MDSD



Source: Stahl, Voelter, 2005

What can be Generated?

- Database Schema
- Data Access Layer
- User Interfaces
- Whole or part of the application logic
- Documentation
- Configurations
- Tests
- Wrapper
- Import/Export modules
- ...

Advantages of Software Generation

- Higher productivity
  - Tedious parts can be automated
  - Reduced reaction time on design changes/change requirements
- Improved quality
  - The transformation (template) is responsible for the quality of the code
  - Integrated architecture in templates defined
  - Automatic transformations (no careless errors)

Advantages of Software Generation

- Higher abstraction
  - Model represents an abstract description of the application
  - Business rules can be review by domain experts
  - easier change to new technology (change templates)
  - reuse of already developed transformation rules (software factories)
  - better handling of complexity (reduction to essential)

- consistency of application
  - code generated based on rules is very consistent (naming conventions, parameter passing, ...) and so easy to understand and use
  - cross cutting concerns bundled in a central place (template/rule)

Karlsruhe Institute of Technology

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Part I

Regular Expressions

Regular Expressions

- Powerful text pattern language
- Allows the filtering/substitution of text patterns
- Implementation in many computer languages
- consists of
  - literal characters (A...Z, a...z 0...9 _, ...)
  - meta characters ([ ] ( ) { } | ? + - * ^ $ \ . \b)
  - character classes:
    - predefined: . \w \d \s \W \D \S
    - user defined: [A-Z] [aeiou] [0-9A-Fa-f] ...

Concepts

- Quantifier: define how many times the token before should be matched
  - * : zero or more (greedy)
  - *? : zero or more (ungreedy or lazy)
  - + : one or more (greedy)
  - +? : one or more (ungreedy or lazy)
  - ? : zero or one
  - {3,5} : three to five
  - {3,} : three or more
  - {,5} : less or equal five

Concepts

- Backreferences: if parts of a matched text should be used later, use round brakkets to mark these parts (referenced[1] later by $1, $2, $3, ...)

- Position in pattern
  - ^ : Start of pattern
  - $: End of pattern
  - \b : Word boundary

1. or \1, \2, \3, ... depening on usd tool/implementation

Examples of Regular Expressions

- Matching:
  - IP-Address: `\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}`
    `(\d{1,3}\.){3}\d{1,3}`
  - Number between 100 and 9999 : `[1-9][0-9]{2,3}`
  - Extract headings from a HTML-Document: `<h([0-3])>(.*?)</h$1>`
  - A number (i.e. 1, -2.564, 0.1, ...) : `-?\d+(\.\d+)?`

- Replacing (perl syntax)
  - Change your winter hobby: `s#\bSki\b#Snowboard#g`
  - Remove markup from HTML: `s#<.*?>##g`
  - Make Hyperlink from URL:
    `s#\bhttps?://(.*?)\s#<a href="$1">$1</a>#g`

Embedding Regexes in a Programming Language

- The perl way
  - Regexes are integral part of the language
  - Examples:

```
$text = "The computers with ip-addresses 123.34.45.234 and 123.34.32.1 are infected";

if ($text =~ /((\d{1,3}\.){3}\d{1,3})/) {
  print "IP-address $1 found\n";
}
print "All IP-Addresses:\n";
while ($text =~ /((\d{1,3}\.){3}\d{1,3})/g) {
  print $1,"\n";
}

$text = "The URL of my institute is http://www.iai.kit.edu";
$text=~s#\b(http://([\w./]+))#<a href="$1">$2</a>#g;
print $text;
```

- Output:

```
IP-Address 123.34.45.234 found

All IP-Addresses:
123.34.45.234
123.34.32.1

The URL of my institute is
<a href="http://www.kit.edu">\
www.kit.edu</a>"
```

Karlsruhe Institute of Technology

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Embedding regexes in a Programming Language

- The other way (e.g. PHP, Java, awk, ...)
  - Integration via library
  - Regular expressions are handled as Strings
  - Examples (PHP)

```
$text = "The computers with ip-addresses 123.34.45.234 and 123.34.32.1 are infected";

if (preg_match('/((\d{1,3}\.){3}\d{1,3})/',$text, $match)) {
  print "IP-address ".$match[1]." found\n";
}

print "All ip-addresses:\n";
if (preg_match_all('/((\d{1,3}\.){3}\d{1,3})/',$text, $all_matches))
  foreach ($all_matches[0] as $match)
    print "$match\n";

$text = "The URL of my institute is http://www.iai.fzk.de";
$text= preg_replace ('#\b(http://([\w./]+))#','<a href="\1">\2</a>', $text);
print $text;
```

Karlsruhe Institute of Technology

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Part II

Generator Models

Karlsruhe Institute of Technology

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

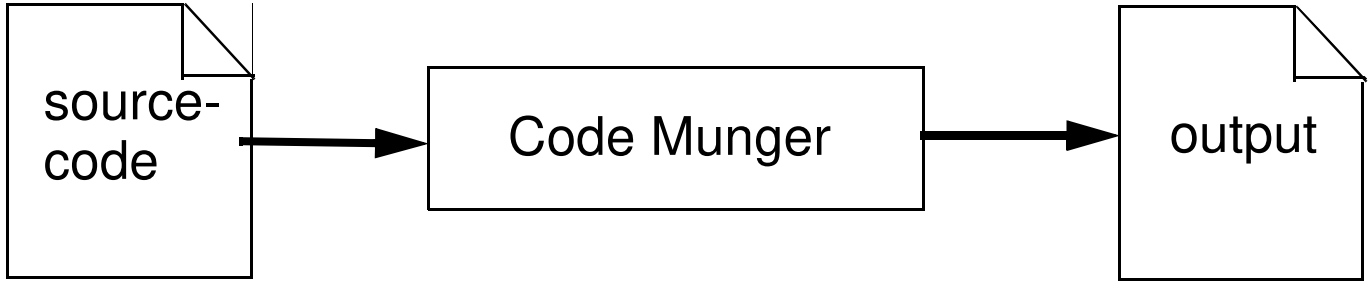
Generator Models

- Code Munger
- Inline Code Expander
- Mixed Code Generator
- Partial Class Generator
- Full Tier Generator
- Domain Specific Language

Source: [Herrington, 2003]

Code Munger

- Workflow



- Functionality:
  - Input is source code
  - Extraction of relevant aspects from the code
  - Transformation to output format
  - Extraction is commonly based on regular expressions
- Examples:
  - Javadoc
  - XDoclet / Java annotations

Karlsruhe Institute of Technology

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Code Munger

Further examples:

- Generation of a class/method index in HTML
- Generation of base classes
- Generation of SQL statements from CSV data
- Generation of base classes from DDL Statements
- Source code analysis
- Modification of xml documents (without XSLT/DOM)

Code Munger - Example (1)

- Extraction of a class-/method index of library file DB.php

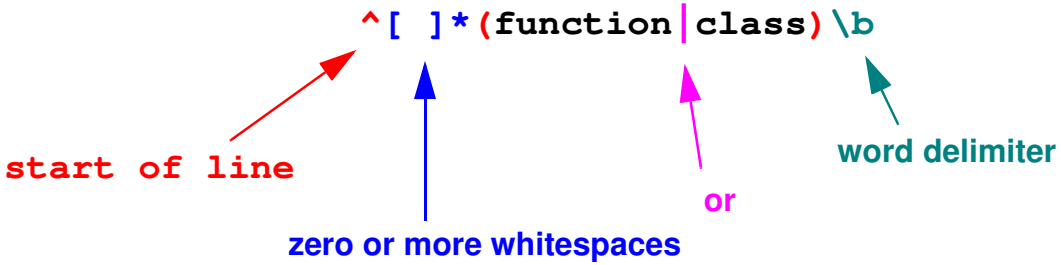
```
$ egrep -e '^[ ]*(function|class)\b' d:/Programme/php/PEAR/DB.php
class DB
    function factory($type, $options = false)
    function connect($dsn, $options = array())
    function apiVersion()
    function isError($value)
    function isConnection($value)
    function isManip($query)
    function errorMessage($value)
    function parseDSN($dsn)
class DB_Error extends PEAR_Error
    function DB_Error($code = DB_ERROR, $mode = PEAR_ERROR_RETURN, ...)
class DB_result
    function DB_result(&$dbh, $result, $options = array())
    function setOption($key, $value = null)
    function fetchRow($fetchmode = DB_FETCHMODE_DEFAULT, $rownum = null)
    function fetchInto(&$arr, $fetchmode = DB_FETCHMODE_DEFAULT, $rownum = null)

    function numCols()
    ...
```

Output Formatation (in HTML)

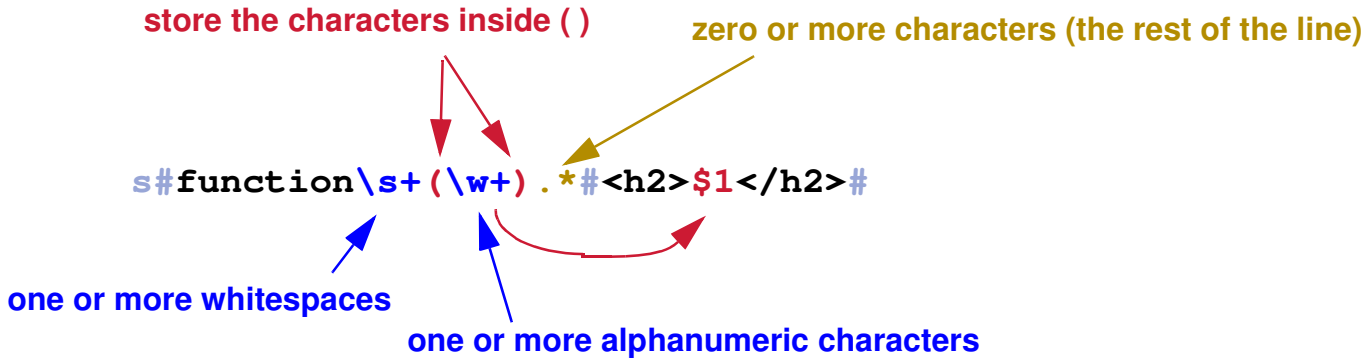
```
$ egrep -e '^[ ]*(function|class)\b' ../Programme/php/PEAR/DB.php | \
    perl -pe 's#(class\s+\w+).*#<h2>\1</h2>#;s#function\s+(\w+)(\(.*\))#<b>\1</b> \2#'
<h2>class DB</h2>
    <b>factory</b> ($type, $options = false)
    <b>connect</b> ($dsn, $options = array())
    <b>apiVersion</b> ()
    <b>isError</b> ($value)
    <b>isConnection</b> ($value)
    <b>isManip</b> ($query)
    <b>errorMessage</b> ($value)
    <b>parseDSN</b> ($dsn)
    <b>getDSNString</b> ($dsn, $hidePassword) {
<h2>class DB_Error</h2>
    function DB_Error($code = DB_ERROR, $mode = PEAR_ERROR_RETURN, ...)
<h2>class DB_result</h2>
    <b>DB_result</b> (&$dbh, $result, $options = array())
    <b>setOption</b> ($key, $value = null)
    <b>fetchRow</b> ($fetchmode = DB_FETCHMODE_DEFAULT, $rownum = null)
    <b>fetchInto</b> (&$arr, $fetchmode = DB_FETCHMODE_DEFAULT, $rownum = null)
    <b>numCols</b> ()
    <b>numRows</b> ()
```

Explanation Regular Expressions

- pattern matching (egrep-syntax)



- pattern substitution (`s # text # relacement #`) (perl-syntax)



Output

class DB

factory($type, $options = false)
connect($dsn, $options = array())
apiVersion()
isError($value)
isConnection($value)
isManip($query)
errorMessage($value)
parseDSN($dsn)
getDSNString($dsn, $hidePassword)

class DB_Error

DB_Error($code = DB_ERROR, $mode = PEAR_ERROR_RETURN, $level = E_USER_NOTICE, $debuginfo = null)

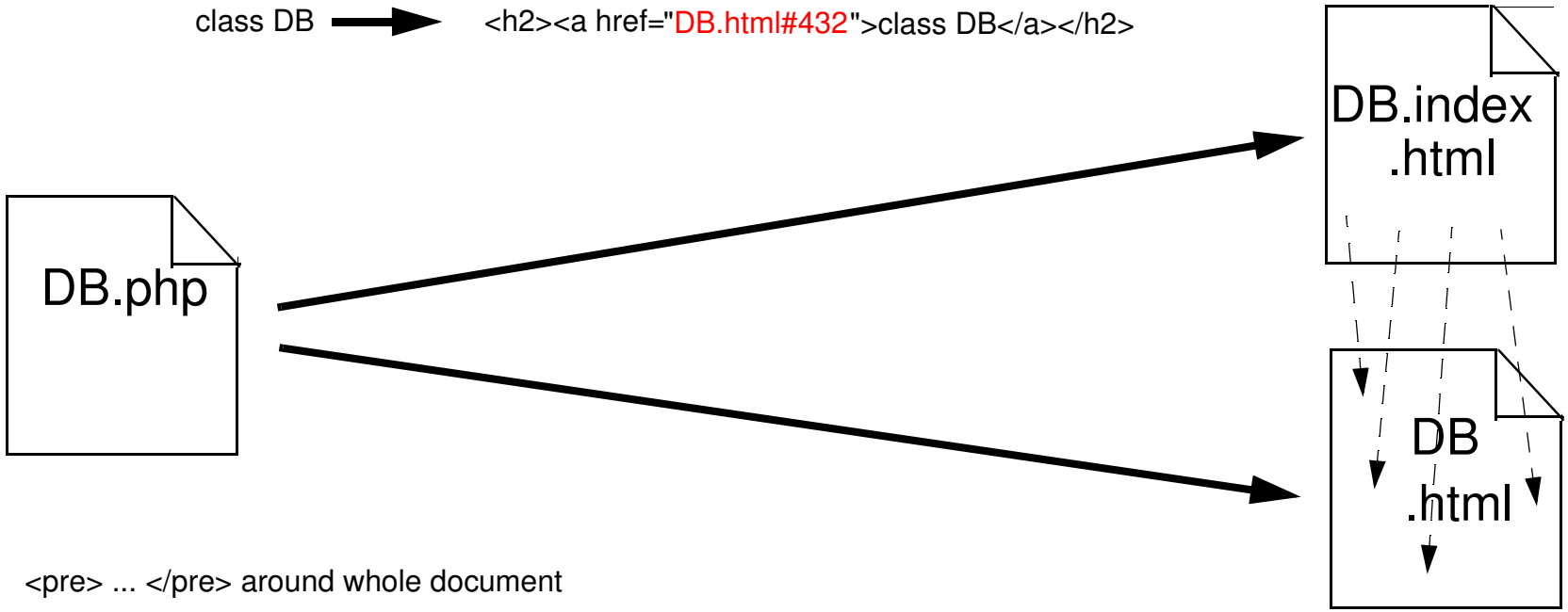
class DB_result

DB_result(&$dbh, $result, $options = array())
setOption($key, $value = null)

Possible extension

function isError($value) → function isError($value)

class DB → <h2>class DB</h2>



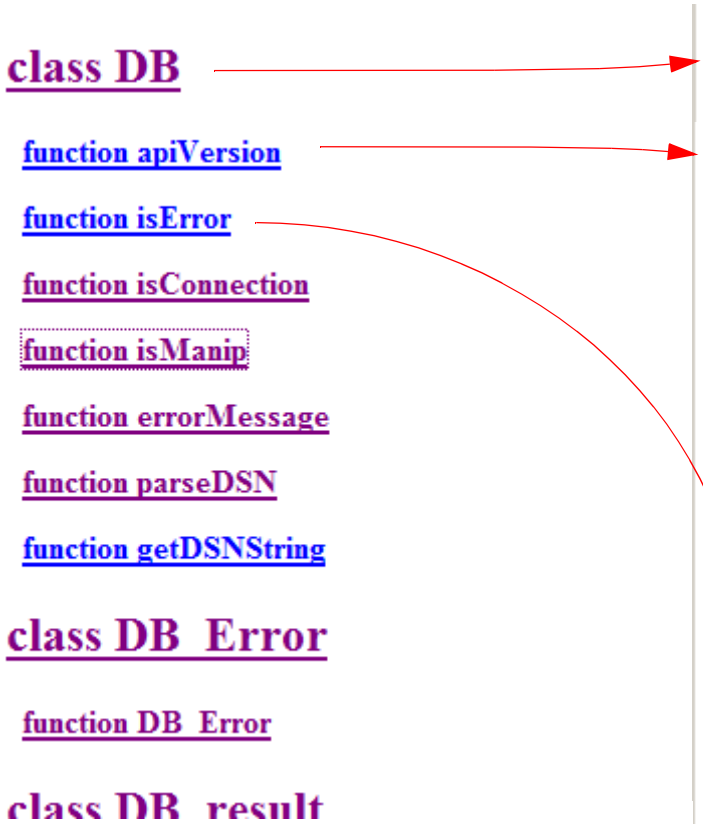
<pre> ... </pre> around whole document

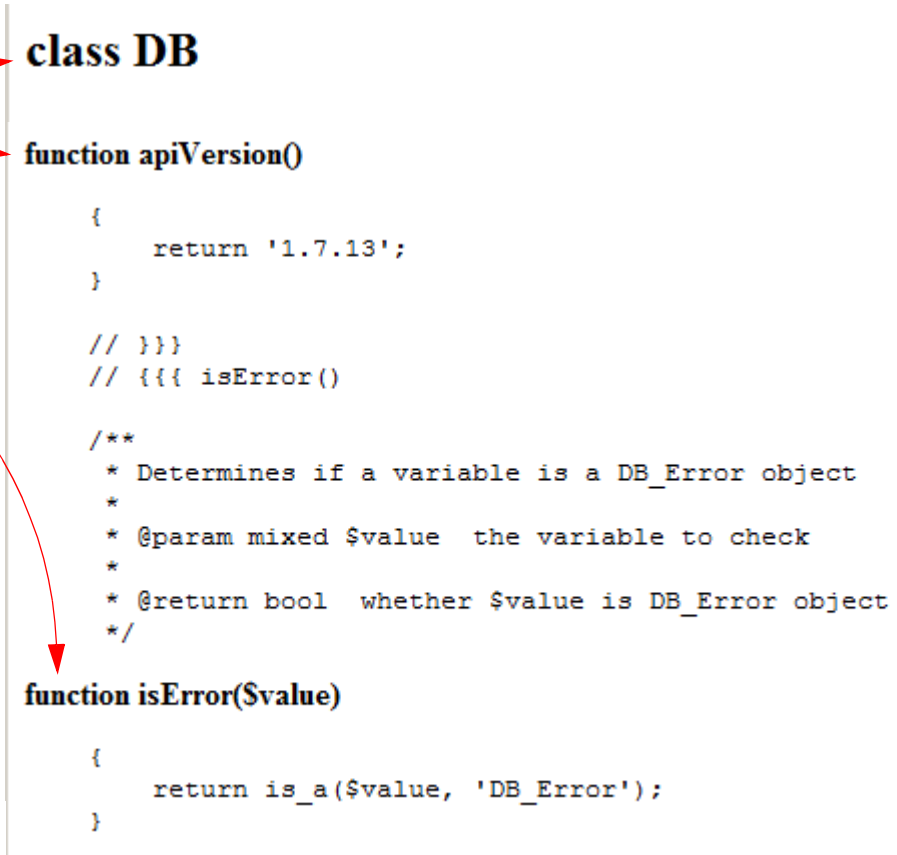
function isError($value) → </pre>function isError($value)
<pre>

class DB → </pre><h2>class DB</h2><pre>

rest → ...

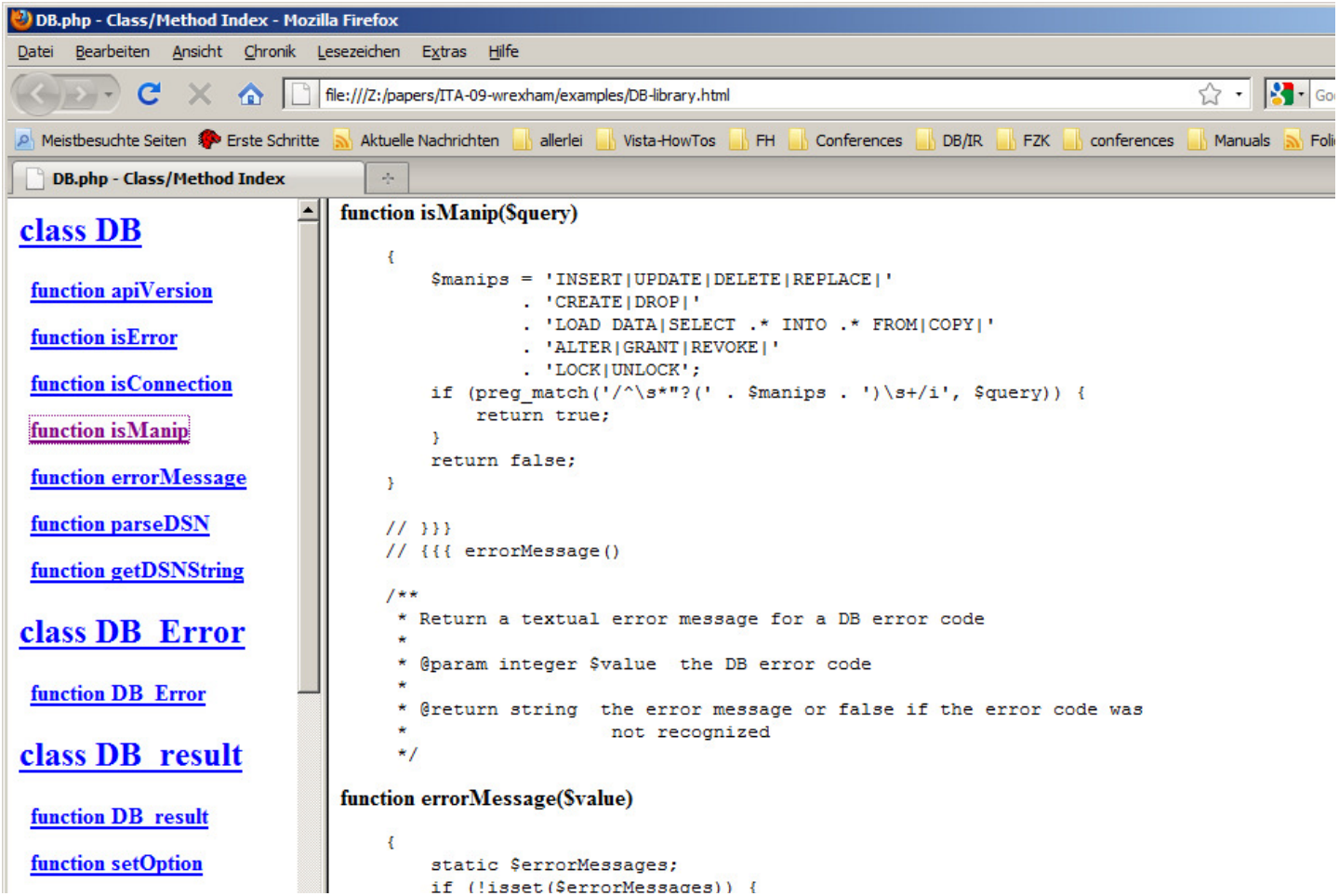
- DB.index.html
- DB.html





- Overall effort:
- 2 regular expressions for index (one line each)
- 2 regular expressions for code (one line each)
- 3 command line calls
- 1 frameset

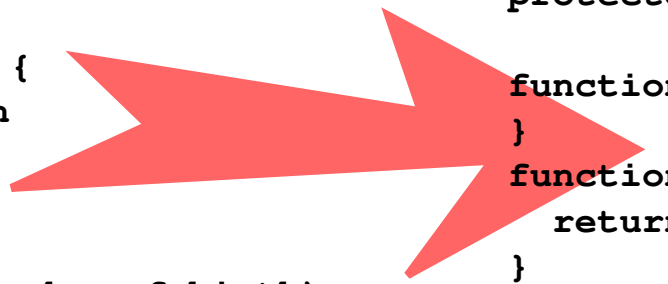
easily adaptable for many languages !!



Code Munger - Example (2)

File: Person.php

```
<?php

include 'BasePerson.php';


class Person extends BasePerson {
  // @fields: name, day_of_birth

  function age() {
    $now = time();
    $birthday = strtotime($this->day_of_birth);
    return floor(($now - $birthday)/(3600*24*365));
  }
}
```



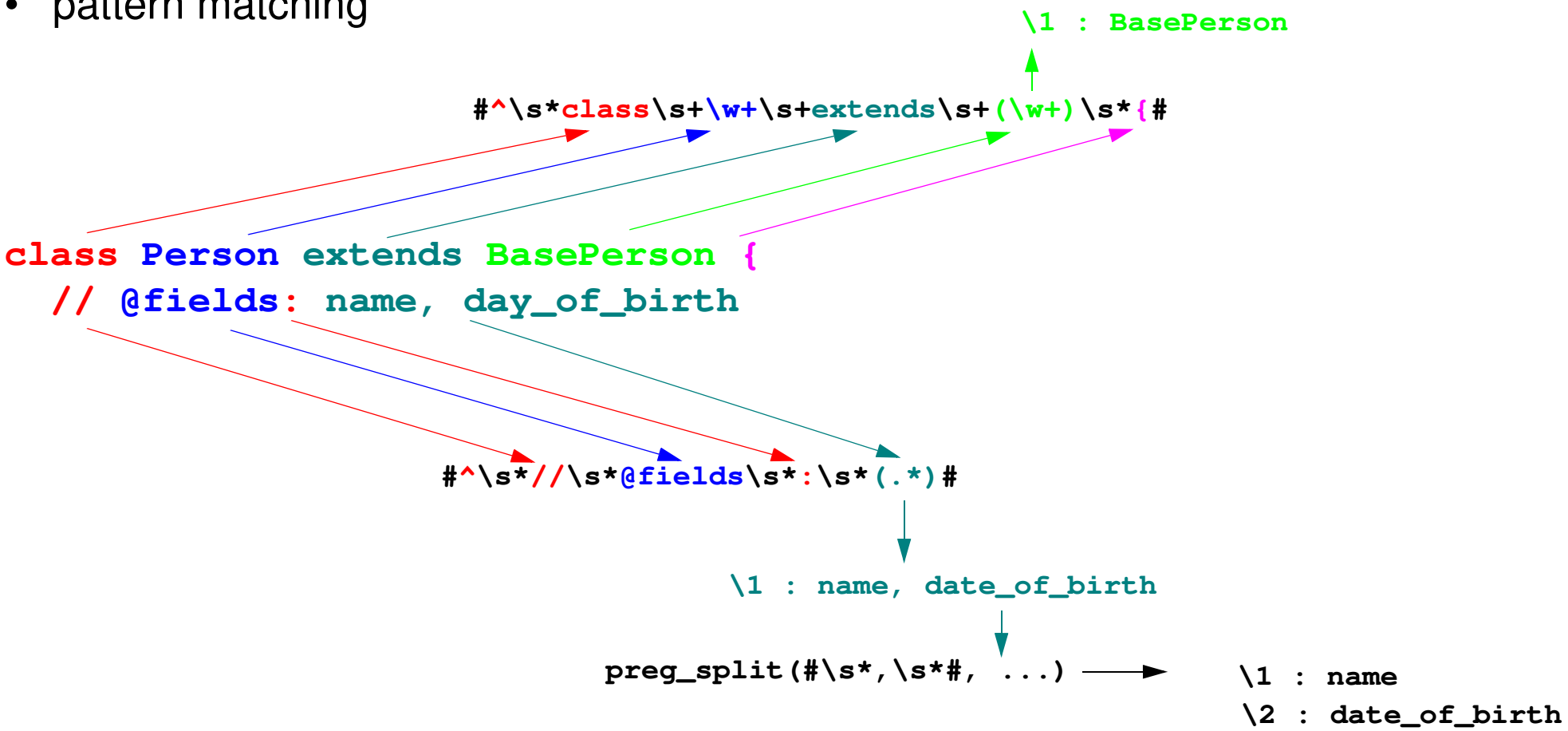
```
<?php

class BasePerson {

  protected $name;
  protected $day_of_birth;

  function __construct() {
  }
  function get_name() {
    return $this->name;
  }
  function get_day_of_birth() {
    return $this->day_of_birth;
  }
  function set_name($value) {
    return $this->name = $value;
  }
  function set_day_of_birth($value) {
    return $this->day_of_birth = $valu
  }
}
```

Example (Implementation)

File: generate-base-classes.php

```
<?php

$content = file_get_contents($argv[1]);
$lines = split("\n", $content);
foreach ($lines as $line) {

  if (preg_match('#^\s*class\s+\w+\s+extends\s+(\w+)\s*{#', $line, $match)) {

    $base_class = $match[1];

} else if (preg_match('#^\s*//\s*@fields\s*:\s*(.*)#',$line,$match)) {

    $attributes = preg_split('#\s*,\s*#', $match[1]);
    $content = create_base_class($base_class, $attributes);
    file_put_contents($base_class.".php", $content);
  }
}
```

- pattern matching



```
function create_base_class($classname, $attributes) {
  $content = "<?php\n
class $classname {\n";
  foreach ($attributes as $attribute) {
    $content .= "  protected \$$attribute;\n";
  }
  $content .= "\n function __construct() {
  }\n";
  foreach ($attributes as $attribute) {
    $content .= "\n function get_$attribute() {
    return \$this->$attribute;
  }\n";
  }
  foreach ($attributes as $attribute) {
    $content .= "\n function set_$attribute(\$value) {
    return \$this->$attribute = \$value;
  }\n";
  }
  $content .="\n}";
  return $content;
}
```

instance variables

getter

setter

Run the test ...

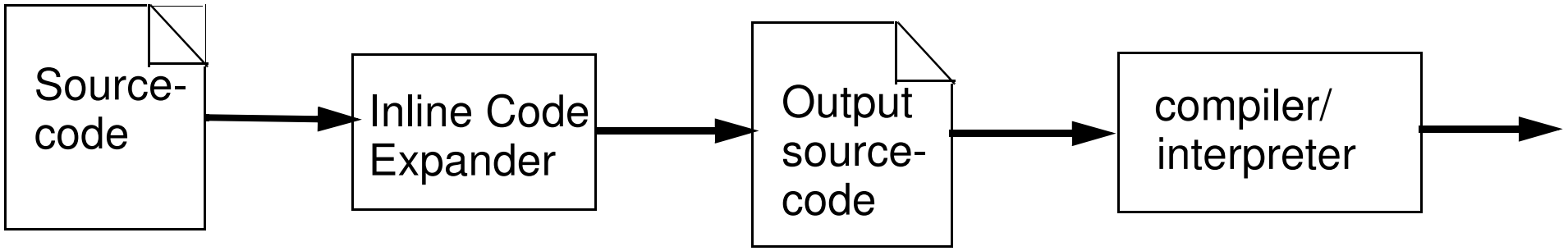
- File: test.php

```
<?php

include 'Person.php';

print "a little test:\n";

$p = new Person();
$p->set_name('Andreas Schmidt');
$p->set_day_of_birth('9 September 1965');

print  $p->get_name()." is ".
       $p->age()." years old\n";
```

```
$ php.exe generate-base-classes.php Person.php
$ php.exe test.php
a little test:
Andreas Schmidt is 54 years old
```

Inline Code Expander

- Workflow



- Functionality:
  - Implicit definition of a new language (extension of existing language)
  - Simplifies the writing of source code,
  - Outout is input for a compiler/interpreter
- Examples:
  - SQLJ
  - Generation of classes

Example

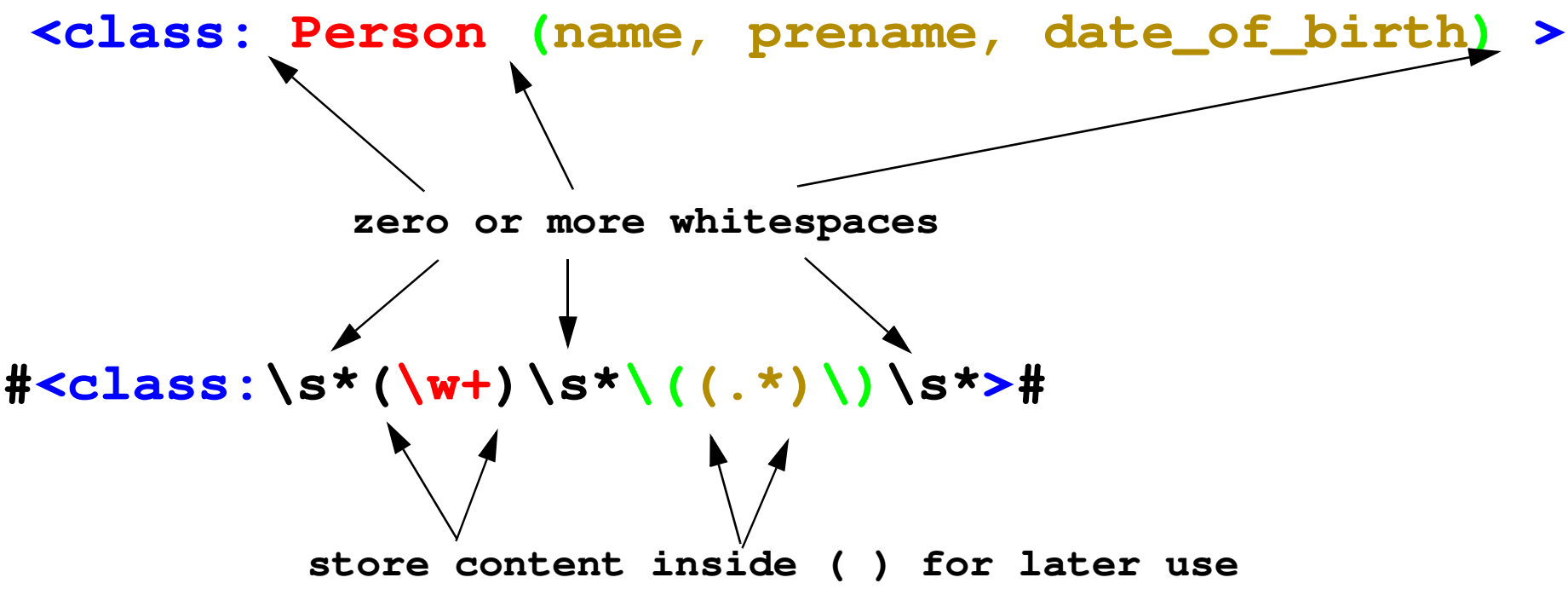
- Sourcecode ice-test.phpi

```
<?php

<class: Person (surname, firstname, day_of_birth) >

<class: Film (title, year, regisseur) >

$p1 = new Person('Waits', 'Tom', '9.9.1949');
$f1 = new Film('Short Cuts', 1989, 'Jim Jarmusch');

echo "a little test:\n--------------\n";
echo $p1->get_surname()." ".$p1->get_firstname()." ".
     $p1->get_day_of_birth()."\n";
echo $f1->get_title()." ".$f1->get_year()."\n";

?>
```

Example

```
<?php

    class Person { ... }

    class Film {
         private $title;
         private $year;
         private $regisseur;

        function  __construct($title,$year,$regisseur) {
         $this->title = $title;
         $this->year = $year;
         $this->regisseur = $regisseur;


     function get_title() {
         return $this->title;
     }

     function get_year() {
         return $this->year;
     }

     function get_regisseur() {
         return $this->regisseur;
     }
   }

$p1 = new Person('Waits', 'Tom', '9.9.1949');

$f1 = new Film('Short Cuts', 1989, 'Jim Jarmusch');

?>
```

Pattern Matching for Extended Syntax



Implementation Inline Code Expander

- inline-code-expander.php

```
<?php

$lines = file($argv[1]);
foreach ($lines as $line) {
  if (preg_match('#<class:\s*(\w+)\s*\((.*)\)\s*>#', $line, $match)) {
    $class_name = $match[1];
    $att_list = preg_split("#\s*,\s*#", $match[2]);
    print_class_definition($class_name, $att_list);
  } else {
    print $line;
  }
}
```

look for a line with special syntax (extended language syntax)

otherwise, do nothing (just output the line)

```
function print_class_definition($class_name, $att_list) {
    ?>
    class <?= $class_name ?> {
       <?php foreach ($att_list as $a) { ?>
        private $<?= $a ?>;
       <?php } ?>

       function  __construct(<?= join(",",add_dollar_sign($att_list)) ?>) {
       <?php foreach ($att_list as $a) { ?>
        $this-><?= $a ?> = $<?= $a ?>;
       <?php } ?>
       }

       <?php foreach ($att_list as $a) { ?>
       function get_<?= $a ?>() {
           return $this-><?= $a ?>;
       }
       <?php } ?>
    }
    <?php
}
```

Example Workflow

```
$ php.exe inline-code-expander.php ice-test.phpi > gen_ice.php
```

```
$ php.exe gen_ice.php
a little test:
--------------
Waits Tom 9.9.1949
Short Cuts 1989
```

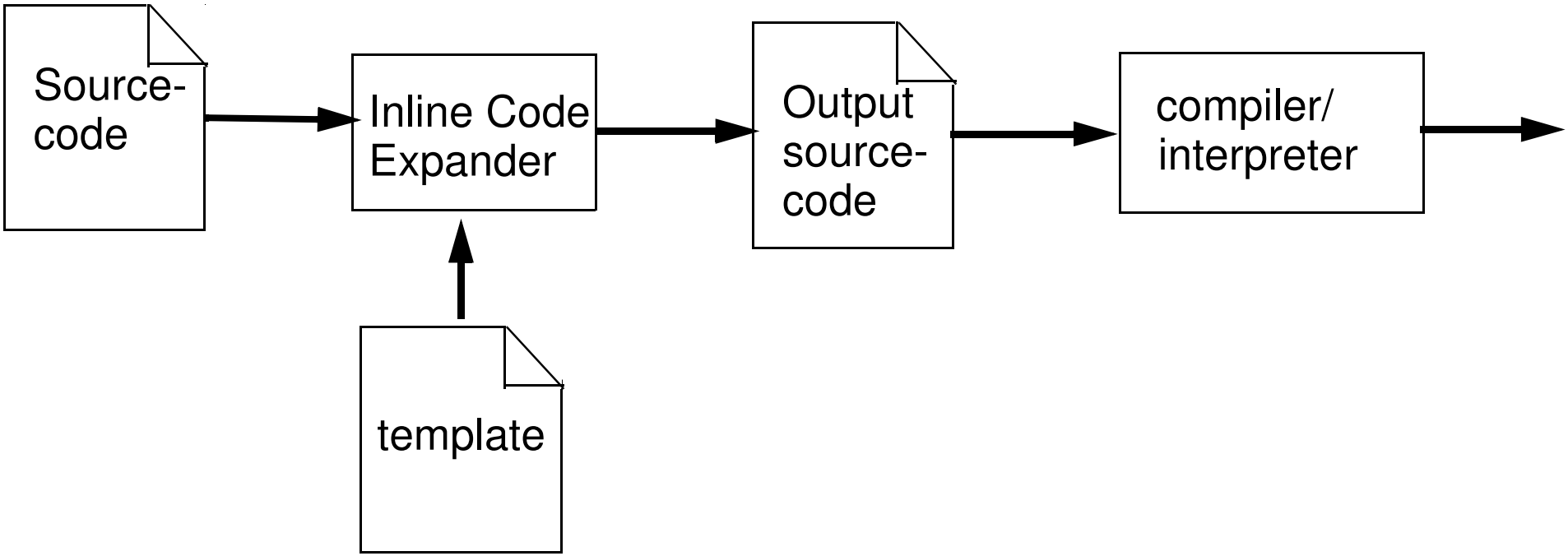
Using a separate Template

```
<?php

$lines = file($argv[1]);
$template_file = $argv[2];

foreach ($lines as $line) {
  if (preg_match('#<class:\s*(\w+)\s*\((.*)\)\s*>#',
$line, $match)) {
     $class_name = $match[1];
     $att_list = preg_split("#\s*,\s*#", $match[2]);
    print_class_definition($class_name, $att_list);
  } else {
    print $line;
  }
}

function print_class_definition($class_name,
                                $att_list) {
  global $template_file;
  include( $template_file );
}
```

- Template File:

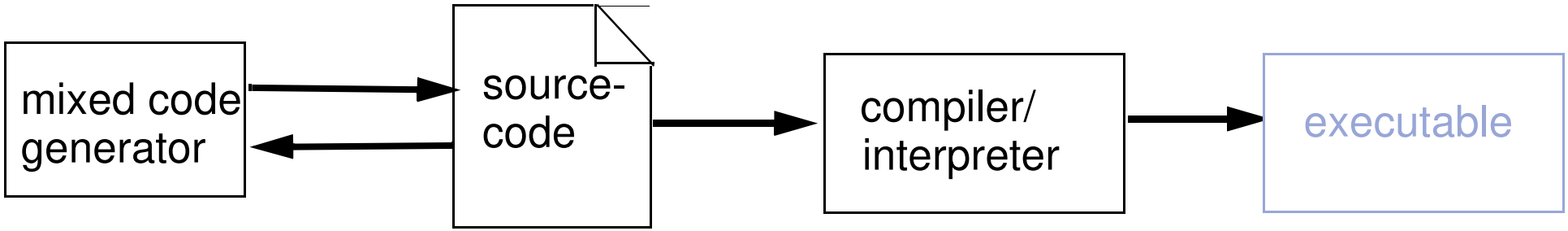
```
class <?php echo $class_name ?> {
  <?php foreach ($att_list as $a) { ?>
    private $<?php echo $a ?>;
  <?php } ?>

  function __construct(<?php echo
    join(",",add_dollar_sign($att_list)) ?>) {
  <?php foreach ($att_list as $a) { ?>
    $this-><?php echo $a ?> = $<?php echo $a ?>;
  <?php } ?>
  }

  <?php foreach ($att_list as $a) { ?>
  function get_<?php echo $a ?>() {
      return $this-><?php echo $a ?>;
  }
  <?php } ?>
}
```

- Workflow (with separate template file)



Mixed Code Generator

- Workflow

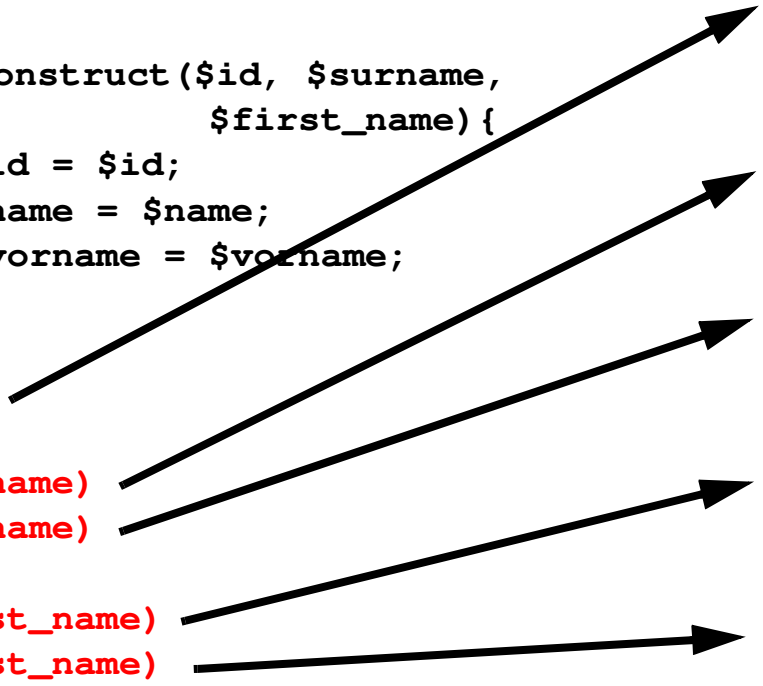


- Functionality
  - Special implementation of the inline code expander
  - The content of the input file is replaced by the output
  - special syntax is hidden as comments
- Examples
  - Like Inline code expander
  - ...

Mixed Code Generator - Example

- file: source.php

```
class person {
    protected $id;
    private $surname;
    private $first_name;

    function __construct($id, $surname,
                         $first_name){
        $this->id = $id;
        $this->name = $name;
        $this->vorname = $vorname;
    }

    // get($id)

    // get($surname)
    // set($surname)

    // set($first_name)
    // get($first_name)

}
```



```
class person {

    ...

    function get_id() {
        return $this->id;
    }

    function get_name() {
        return $this->name;
    }
    function set_name($name) {
        $this->name = $name;
    }

    function set_vorname($vorname) {
        $this->vorname = $vorname;
    }
    function get_vorname() {
        return $this->vorname;
    }
```

Mixed Code Generator - Implementation

Implementation with regular expressions in perl (as command line tool)

- call:

```
perl -pi.bak transformation.pl source.php
```

- file: transformation.pl

```
s#^(\s*)//\s*set\(\$(\w+)\)#
$1function set_$2($$2) {
$1    \$this->$2 = \$$2;
$1}#;

s#^(\s*)//\s*get\(\$(\w+)\)#
$1function get_$2() {
$1    return \$this->$2;
$1}#;


# explanations:
# $1: indent (whitespaces)
# $2: name of variable
# \$: print a $-sign
# \(: matches a (-sign
```

Regex - Replacement

```
// set($name)
```

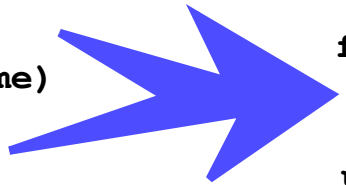
```
s#^(\s*)//\s*set\(\$(\w+)\)#$1function set_$2(\$$2) {
                           $1    \$this->$2 = \$$2;
                           $1}#;
```

```
function set_name($name) {
    $this->name = $name;
}
```

Extension

- User defined constructors

```
// __construct($surname, $firstname)
```



```
function __construct($surname, $firstname)
    $this->surname = $surname;
     $this->firstname =  $firstname;
}
```

- Rule:

```
s#^(\s*)//\s*__construct\s*\((.*)\)#
"$1function __construct($2) {".
constructor_body(split(',',$2)).
"\n$1}"#ge;

sub constructor_body {
  $str = "";
  foreach (@_) {
    $left_side = $_;
    $left_side =~ s/\$/\$this->/;
    $str.= "\n\t$left_side = $_;";
  }
  return $str;
}
```

call of external procedure

eval mode

definition of external procedure

Karlsruhe Institute of Technology

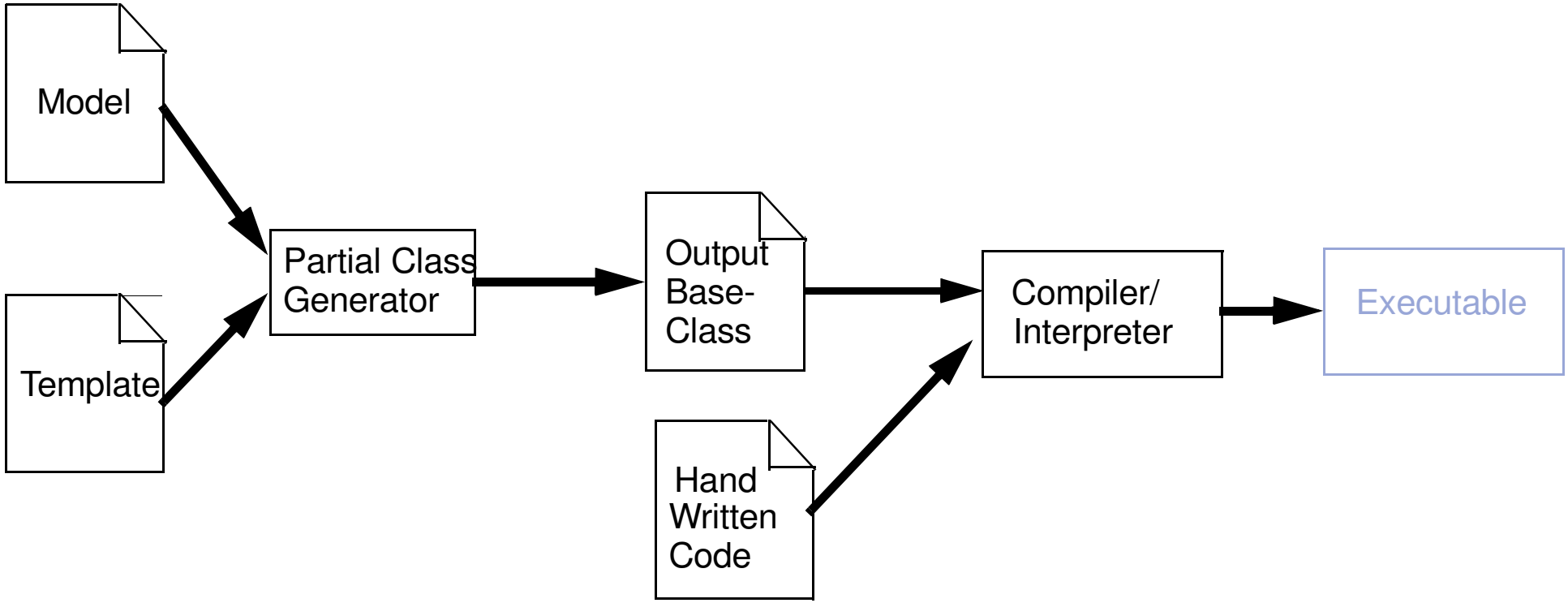
Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Partial Class Generator

- Functionality:
  - based on an explicit definition file (an abstract model)
  - generates a number of base classes
  - Manual extensions in derived classes or „protected areas“
  - Initial point for building a „Tier Generator“

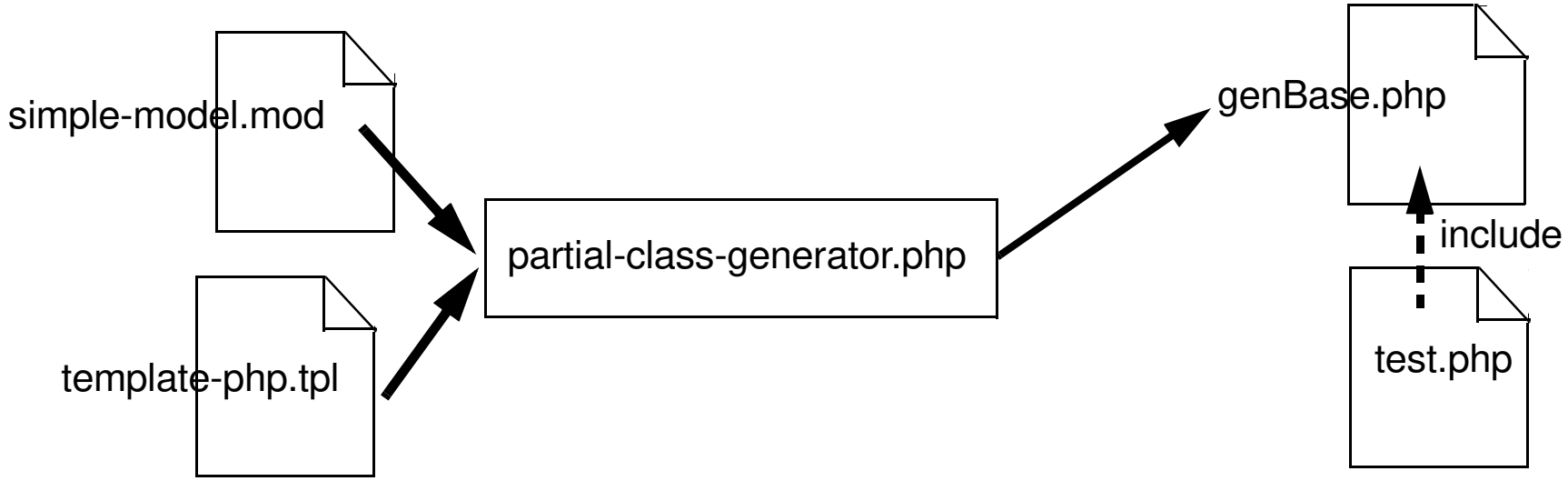
- Examples:
  - Data access layer
  - Database schema
  - User interfaces
  - Import-/export filters

- Workflow


Model
Template
Partial Class Generator
Output Base-Class
Hand Written Code
Compiler/ Interpreter
Executable

Example

- Structure



- simple-model.mod

```
Person(name:string, first_name:string, date_of_birth:date)
Film(title:string, year:integer, director:Person)
```

Partial Class Generator

```
<?php

$lines = file($argv[1]);
$template = $argv[2];

$classes = array();

foreach ($lines as $line) {
  if (! preg_match('#(\w+)\((.*)\)#', $line, $match))
    die("illegal line in model file '$argv[1]':\n>>$line<<");
  $class_name = $match[1];
  $attribute_string = $match[2];
  $attribute_list = preg_split("#\s*,\s*#",  $attribute_string);

  foreach ($attribute_list as $attribute) {

    if (! preg_match('#(\w+):(\w+)#', $attribute, $match))
      die("illegal attribute definition in model file
          '$argv[1]':\n>>$line<<\n>>>>$attribute<<<<");
    $attribut_name = $match[1];
    $attribut_type = $match[2];
    $classes[$class_name][] = array('name'=>$attribut_name,
                                    'type'=>$attribut_type);

}
}

include $template;
```

model file

template file

split name and attributes

split attributes

split name and type

build datastructure for template

call template

Partial Class Generator

- generated datastructure (variable $classes):

```
$classes = Array
(
    [Person] => Array (
            [0] => Array (
                    [name] => name
                    [type] => string )
            [1] => Array (
                    [name] => prename
                    [type] => string )
            [2] => Array (
                    [name] => date_of_birth
                    [type] => date )
        )
    [Film] => Array (
            [0] => Array (
                    [name] => title
                    [type] => string )
            [1] => Array (
                    [name] => year
                    [type] => integer )
            [2] => Array (
                    [name] => director
                    [type] => Person )
        )
)
```

Template File (1)

- template-php.php

„<?php“ at start of generated file

```
<?php print "<?php\n"; ?>

<?php foreach ($classes as $class_name=>$class_attributes)  { ?>

class Base<?php echo $class_name ?> {
    // instance variables
    <?php foreach ($class_attributes as $a) { ?>
    private $<?php echo $a['name'] ?>;
    <?php } ?>

    function __construct($dic) {
    <?php foreach ($class_attributes as $a) { ?>
        $this-><?php echo $a['name'] ?> = $dic['<?php echo $a['name'] ?>'];
    <?php } ?>
    }

    <?php foreach ($class_attributes as $a) { ?>
      function get_<?php echo $a['name'] ?>() {
        return $this-><?php echo $a['name'] ?>;
      }
    <?php } ?>
```

private instance variables

constructor

getter methods

Template file (2)

```
  // continued from previous slide ...

    <?php foreach ($class_attributes as $a) { ?>
      function set_<?php echo $a['name'] ?>($value) {
        $this-><?php echo $a['name'] ?> = $value;
      }
    <?php } ?>

    function __toString() {
       return get_class($this)." [".
    <?php foreach ($class_attributes as $a) { ?>
        " ".trim($this-><?php echo $a['name'] ?>).
    <?php } ?>"]\n";
    }
}
<?php } ?>
```

setter methods

__toString method

Generated Code

```
<?php
class BasePerson {
    // instance variables
    private $name;
    private $prename;
    private $date_of_birth;

    function __construct($dic) {
        $this->name = $dic['name'];
        $this->prename = $dic['prename'];
        $this->date_of_birth =
                    $dic['date_of_birth'];
     }
    function get_name() {
        return $this->name;
    }
    function get_prename() {
        return $this->prename;
    }
    function get_date_of_birth() {
        return $this->date_of_birth;
    function set_name($value) {
        $this->name = $value;
    }
    function set_prename($value) {
        $this->prename = $value;
    }
    function set_date_of_birth($value) {
        $this->date_of_birth = $value;
    }
    function __toString() {
       return get_class($this) . " [" . " " .
trim($this->name) . " " . trim($this->prename)
. " " . trim($this->date_of_birth) . "]\n";
    }
}

class BaseFilm {
   ...
```

Test Application

```
<?php

include 'genBase.php';

class Person extends BasePerson {
    // manual coded logic
}

class Film extends BaseFilm {
     // manual coded logic
}

print "a little test:\n";
print "--------------\n";
$person = new Person(array('name' => 'Smith', 'prename' => 'Kevin'));
$film = new Film(array('title' => 'The clerks II', 'year' => 2006));
$film->set_director($person);
echo $person;
echo $film;
echo "regisseur: ";
echo $film->get_director();
```

- usage:

```
$ php.exe partial-class-generator.php simple-model.mod template-php.tpl > genBase.php
$ php.exe -l genBase.php
$ php.exe test.php
```

- output:

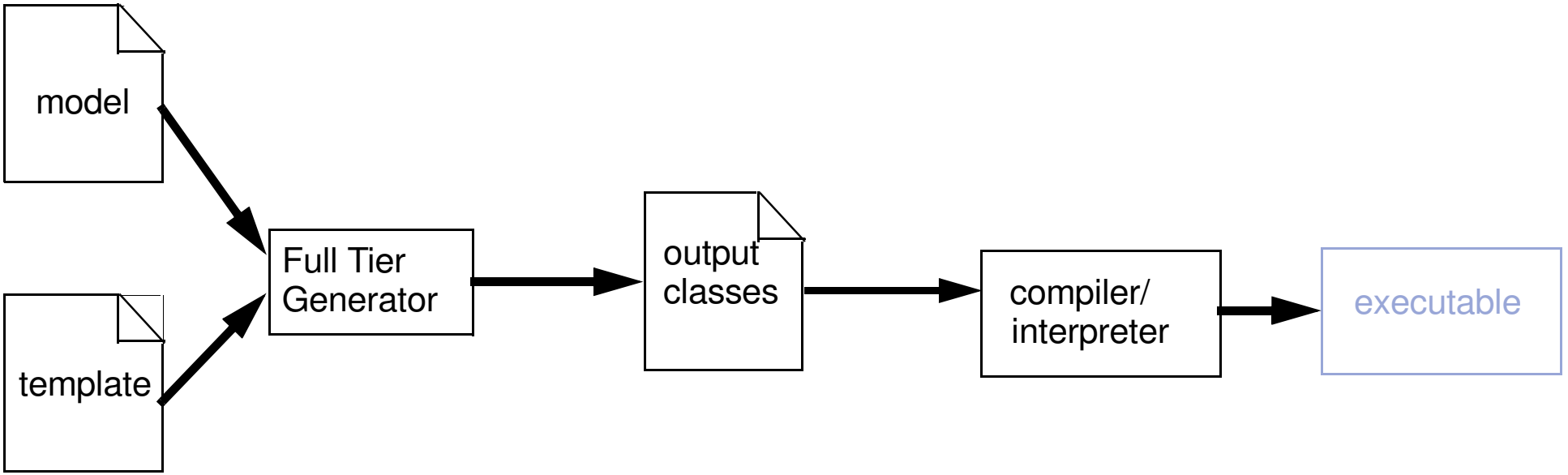
```
a little test:
--------------
Person [ Smith Kevin ]
Film [ The clerks II 2006 Person [ Smith Kevin ]]
regisseur: Person [ Smith Kevin ]
```

Tier Generator Model

- Functionality
  - Like „Partial Class Generator“, but it generates the code of a tier of an application
  - Whole application logic outside of codebase
  - „Partial Class Generator“ is a good starting point for building a „Tier Generator model“
- Examples
  - Database Access layer
  - Web client layer
  - Data Import/Export

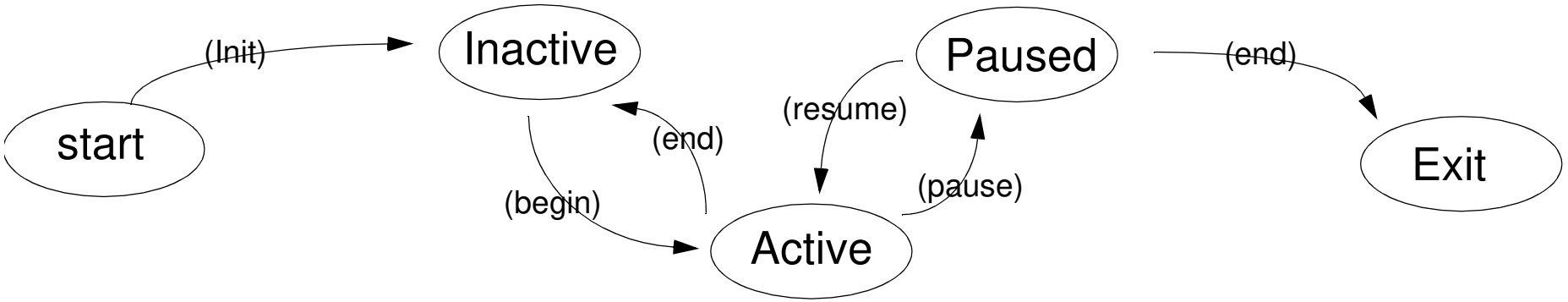
Tier Generator Model - Workflow

Workflow



Tier Generator Model - Model

- State Diagram



- Model File:

```
start   -(init)-> Inactive
                  Inactive -(begin)-> Active
                  Inactive <-(end)-   Active
                                      Active  -(pause)->  Paused
                                      Active <-(resume)-  Paused
                  Inactive <-(end)-                       Paused
          Inactive             -(exit)->                              Exit
```

Tier Generator Model - Generator

```
include 'Statechart.php';

$lines = file($argv[1]);
$template_file = $argv[2];
$statechart = new Statechart();
foreach ($lines as $line) {
  if (preg_match('#^\s*(\w+)\s*-+\((\w+)\)-+(>)\s*(\w+)\s*$#', $line, $match) or
      preg_match('#^\s*(\w+)\s*<-+\((\w+)\)(-)+\s*(\w+)\s*$#', $line, $match)) {
    $direction = $match[3];
    $state[0] = $match[1];
    $state[1] = $match[4];
    $event = $match[2];
    if ($direction=='-')
      $state = array_reverse($state);
    $statechart->createTransition($state[0], $event, $state[1]);
  } else {
    die("Illegal content in model file ($line)");
  }
}
include $template_file;
```

determines direction ($match[3])

Tier Generator Model - Internal Model

```
class Statechart {

  private $states = array();

  // constructor
  function __construct() {
      ...
  }

  // inserts a new transition and if not already
  // known the start and end state of this
  // transition
  function createTransition($s0,
                            $event,
                            $s1) {
      ...
  }

  // returns an array with all states
  function getStates() {
     ...
  }

}
```

```
class State {

  private $name;
  private $transitions;

  function __construct($name) {
     ...
  }

  function getName() {
     ...
  }

  // adds a transition to the state
  function addTransition($e, $state) {
      ...
  }

  // returns an dictionary with the events as
  // key elements and the target state
  // instances as values
  function getTransitions() {
     ...
  }
}
```

Karlsruhe Institute of Technology

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Tier Generator Model - Template

```
...
function transition($event) {
  <?php foreach ($statechart->getStates() as $state) { ?>
    if ($this->actual_state == '<?php echo $state->getName() ?>') {
     <?php foreach ($state->getTransitions() as $t_event=>$t_state) { ?>
       if ($event=='<?php echo $t_event ?>')
         $new_state = '<?php echo $t_state->getName() ?>';
       else
      <? } ?>
          die("Illegal event ($event) in state '$this->actual_state'");
    } else
  <? } ?>
      die("statemachine is in unknowm state ($this->actual_state)");
     $this->actual_state = $new_state;
    return $new_state;
}
```

Tier Generator Model - Generated Code

```
function transition($event) {
        if ($this->actual_state == 'start') {
          if ($event=='init')
         $new_state = 'Inactive';
        else
                  die("Illegal event ($event) in state '$this->actual_state'");
    } else
           if ($this->actual_state == 'Inactive') {
         if ($event=='begin')
         $new_state = 'Active';
        else
        if ($event=='exit')
         $new_state = 'Exit';
        else
                  die("Illegal event ($event) in state '$this->actual_state'");
    } else
           if ($this->actual_state == 'Active') {
         if ($event=='end')
         $new_state = 'Inactive';
        else
      ...
```

Tier Generator Model - PrettyPrinted Code

```
function transition($event) {
        if ($this->actual_state == 'start') {
            if ($event == 'init') $new_state = 'Inactive';
            else die("Illegal event ($event) in state '$this->actual_state'");
        } else if ($this->actual_state == 'Inactive') {
            if ($event == 'begin') $new_state = 'Active';
            else if ($event == 'exit') $new_state = 'Exit';
            else die("Illegal event ($event) in state '$this->actual_state'");
        } else if ($this->actual_state == 'Active') {
            if ($event == 'end') $new_state = 'Inactive';
            else if ($event == 'pause') $new_state = 'Paused';
            else die("Illegal event ($event) in state '$this->actual_state'");
        } else if ($this->actual_state == 'Paused') {
            if ($event == 'resume') $new_state = 'Active';
            else if ($event == 'end') $new_state = 'Inactive';
            else die("Illegal event ($event) in state '$this->actual_state'");
        } else if ($this->actual_state == 'Exit') {
            die("Illegal event ($event) in state '$this->actual_state'");
        } else die("statemachine is in unknowm state ($this->actual_state)");
        $this->actual_state = $new_state;
        return $new_state;
    }
```

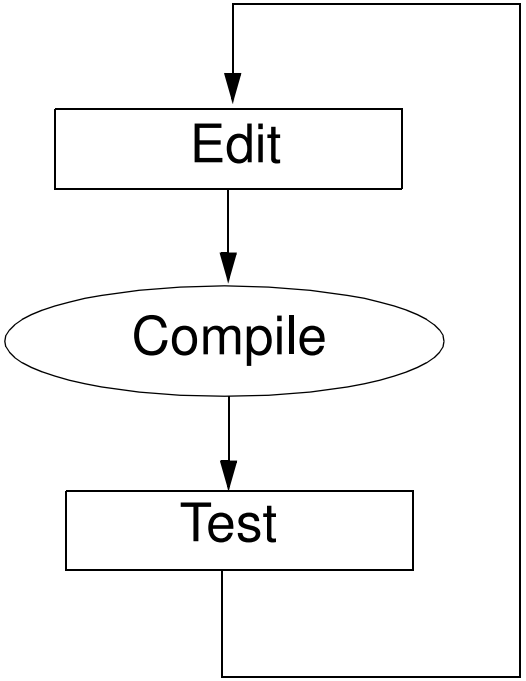
Domain Specific Language (DSL)

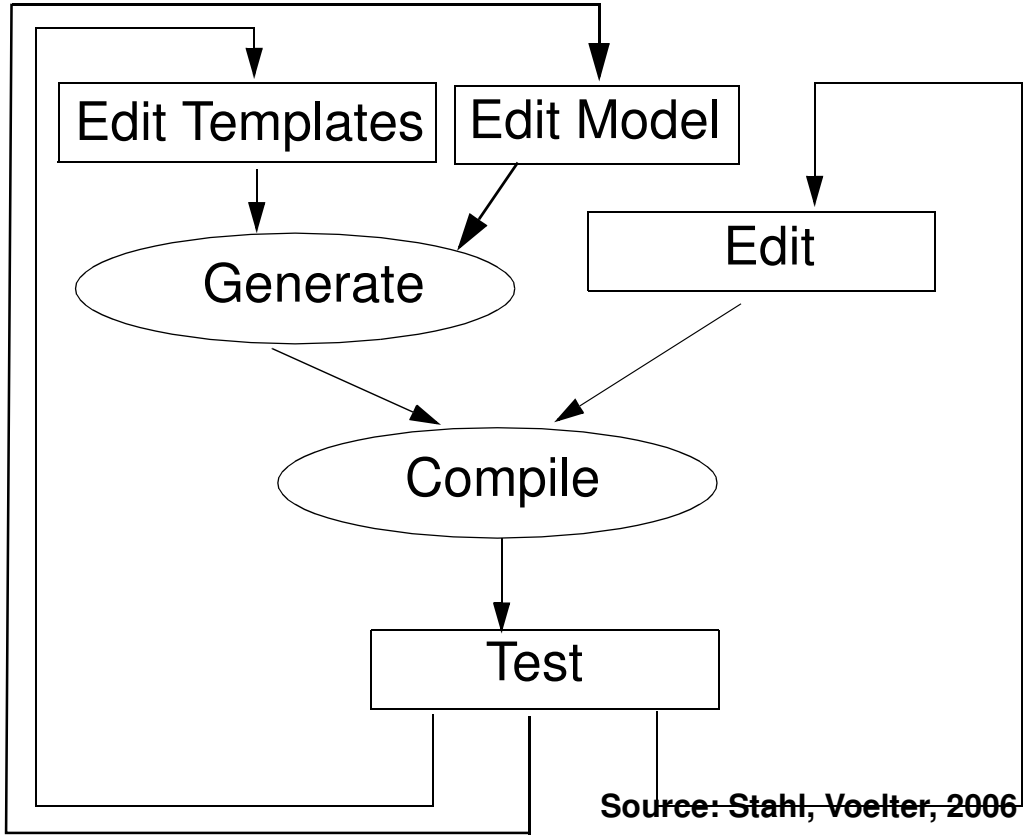
- Functionality
  - A language, which is closely related to your problem domain
  - Used to build the application logic of a program in that domain
  - Special tools to build the parser for your language (lex, yacc, bison, antlr, ...)

- Examples
  - mathematica, matlab
  - make, ant
  - SQL
  - ...

Software Development: Comparision of Workflow

- Traditional Software Engineering
- MDSD





Source: Stahl, Voelter, 2006

Karlsruhe Institute of Technology

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

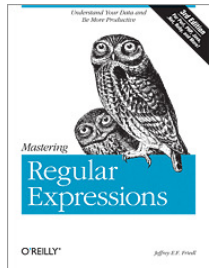
Conclusion

- Codegeneration deals with the partial or complete generation of programs, based on a formal model
- A model could be written in a specific language (model language), existing source code or also available meta information (Database Metadata, XML-Schema, ...)
- Regular Expressions are a powerful language to extract information from code or a formal model
- Lightweight software generators consist often only about a dozens of lines
- Code generation yields to higher abstraction, higher productivity, improved quality and a higher consistence of your application

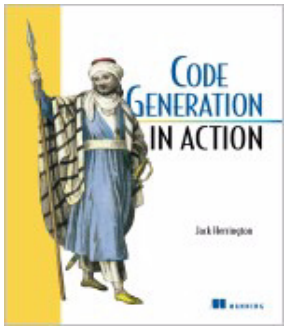
Resume: It's much more interesting to write programs that write programs than to write programs oneself

Resources

- Jeffrey E. F. Friedl, Mastering Regular Expressions, Third Edition, O'Reilly, August 2006



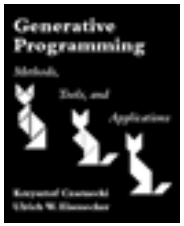
- Jack Herrington: *Code Generation in Action*. Manning Verlag, 2003, 350 Seiten, ISBN: 1930110979



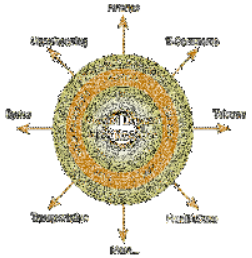
- http://www.codegeneration.net/

Resources

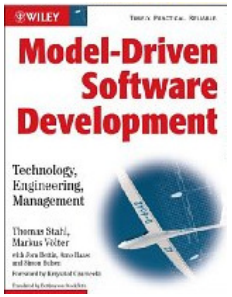
- Krzysztof Czarnecki, Ulrich Eisenecker: *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley Professional; 1. Auflage, 2000



- http://www.omg.org/mda/



- Markus Völter, Thomas Stahl: *Model-Driven Software Development - Technology, Engineering, Management.* Wiley & Sons, May 2006



- Homepage Markus Völter: http://www.voelter.de/services/mdsd.html