b
UNIVERSITÄT
BERN

12. Program Transformation

Prof. O. Nierstrasz

Roadmap

- Program Transformation
- Refactoring
- Aspect-Oriented Programming
- Outlook

Links

- **Program Transformation:**
  - http://swerl.tudelft.nl/bin/view/Pt
  - http://www.program-transformation.org/
- **Stratego:**
  - http://strategoxt.org/
- **TXL:**
  - http://www.txl.ca/
- **Refactoring:**
  - http://www.ibm.com/developerworks/library/os-ecref/
  - http://recoder.sourceforge.net/wiki/
  - http://www.refactory.com/RefactoringBrowser/
- **AOP:**
  - http://www.eclipse.org/aspectj/

Roadmap

> **Program Transformation**
  — **Introduction**
  — Stratego/XT
  — TXL

> Refactoring

> Aspect-Oriented Programming

> Outlook

Thanks to Eelco Visser and Martin Bravenboer for their kind permission to reuse and adapt selected material from their Program Transformation course.
http://swerl.tudelft.nl/bin/view/Pt

What is “program transformation”?

- *Program Transformation* is the process of transforming one program to another.

- Near synonyms:
  - Metaprogramming
  - Generative programming
  - Program synthesis
  - Program refinement
  - Program calculation

Applications of program transformation

- **Translation**
  - Migration
  - Synthesis
    - Refinement
    - Compilation
  - Reverse Engineering
    - Decompilation
    - Architecture Extraction
    - Visualization
  - Program Analysis
    - Control flow
    - Data flow
- **Rephrasing**
  - Normalization
    - Simplification
    - Desugaring
    - Aspect Weaving
  - Optimization
    - Specialization
    - Inlining
  - Refactoring
    - Improvement
    - Obfuscation
  - Reengineering

Translation — compilation

Tiger

```
function fact(n : int) : int =
  if n < 1 then 1
           else (n * fact(n - 1))
```

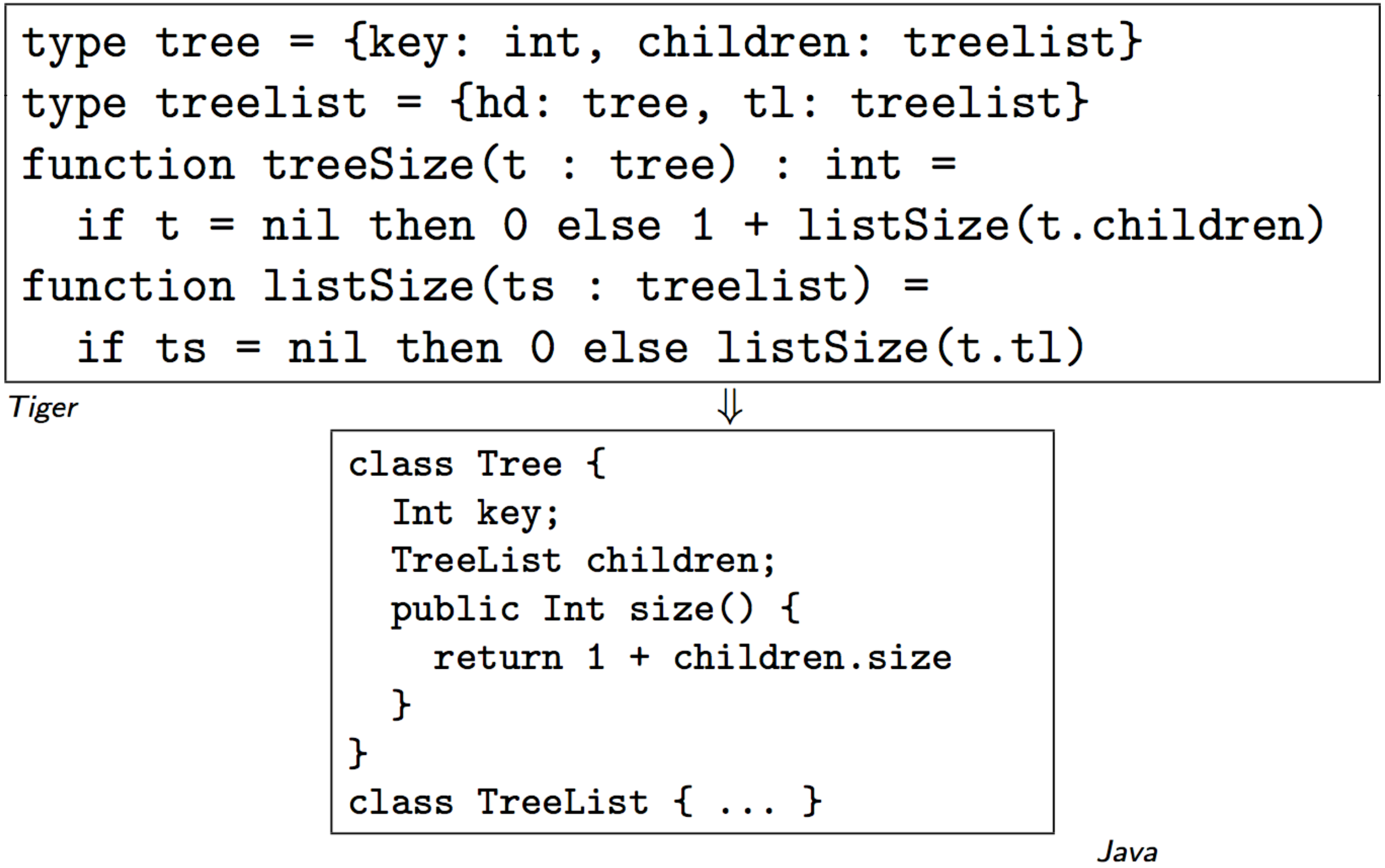
⇒

MIPS

```
fact:subu   $sp, $sp, 20
     sw     $fp, 8($sp)
     addiu  $fp, $sp, 20
     sw     $s2, -8($fp)
     sw     $ra, -4($fp)
     sw     $a0, 0($fp)
     move   $s2, $a1
     li     $t0, 1
     bge    $s2, $t0, c_0
     li     $v0, 1
     b      d_0
c_0: lw     $a0, ($fp)
     li     $t0, 1
     subu   $a1, $s2, $t0
     jal    fact_a_0
     mul    $v0, $s2, $v0
d_0: lw     $s2, -8($fp)
     lw     $ra, -4($fp)
     lw     $fp, 8($sp)
     addiu  $sp, $sp, 20
     jr     $ra
```

http://www.cs.uu.nl/docs/vakken/pt/slides/PT05-ProgramTransformation.pdf

Translation — migration from procedural to OO



```
type tree = {key: int, children: treelist}
type treelist = {hd: tree, tl: treelist}
function treeSize(t : tree) : int =
  if t = nil then 0 else 1 + listSize(t.children)
function listSize(ts : treelist) =
  if ts = nil then 0 else listSize(t.tl)
```

Tiger

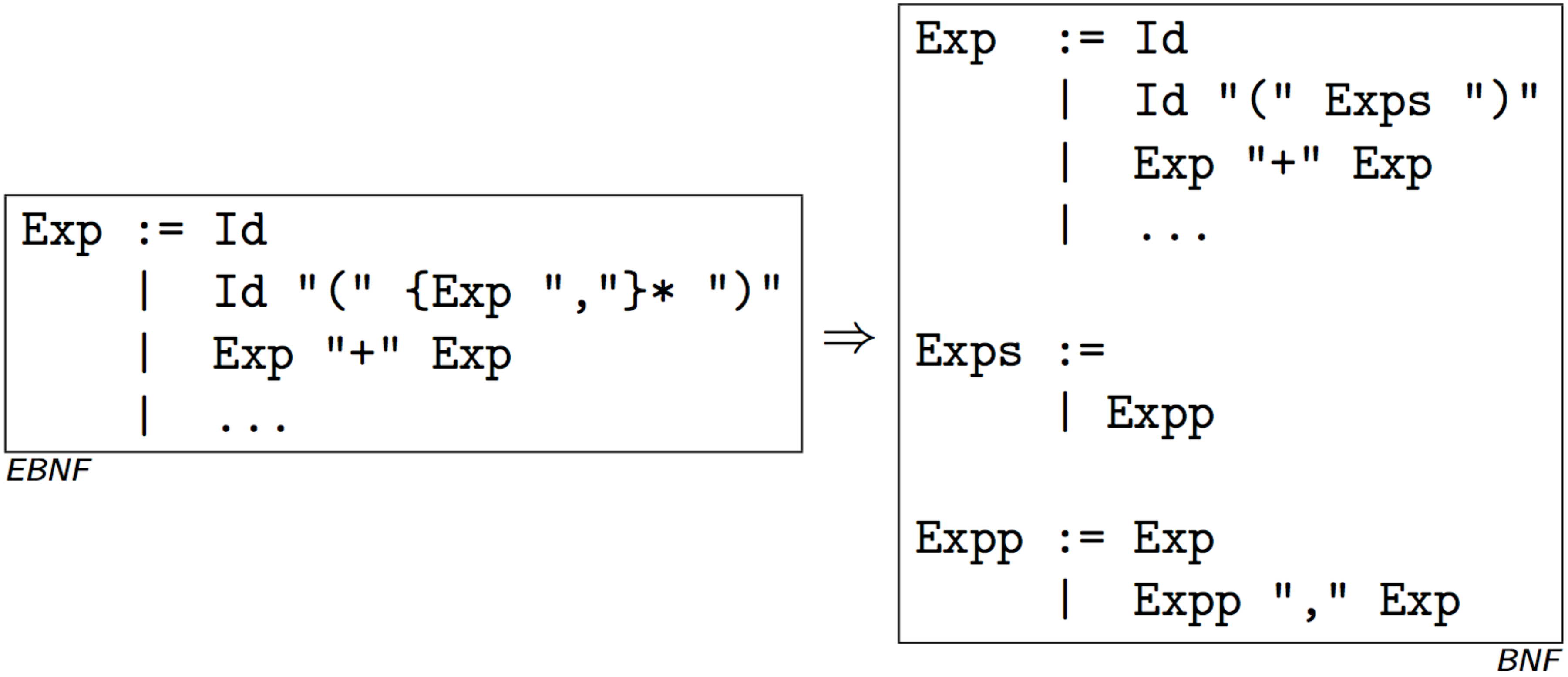
⇓

```
class Tree {
  Int key;
  TreeList children;
  public Int size() {
    return 1 + children.size
  }
}
class TreeList { ... }
```

Java

http://www.cs.uu.nl/docs/vakken/pt/slides/PT05-ProgramTransformation.pdf

Rephrasing — desugaring regular expressions



```
Exp := Id
     |  Id "(" {Exp ","}* ")"
     |  Exp "+" Exp
     |  ...
```

EBNF

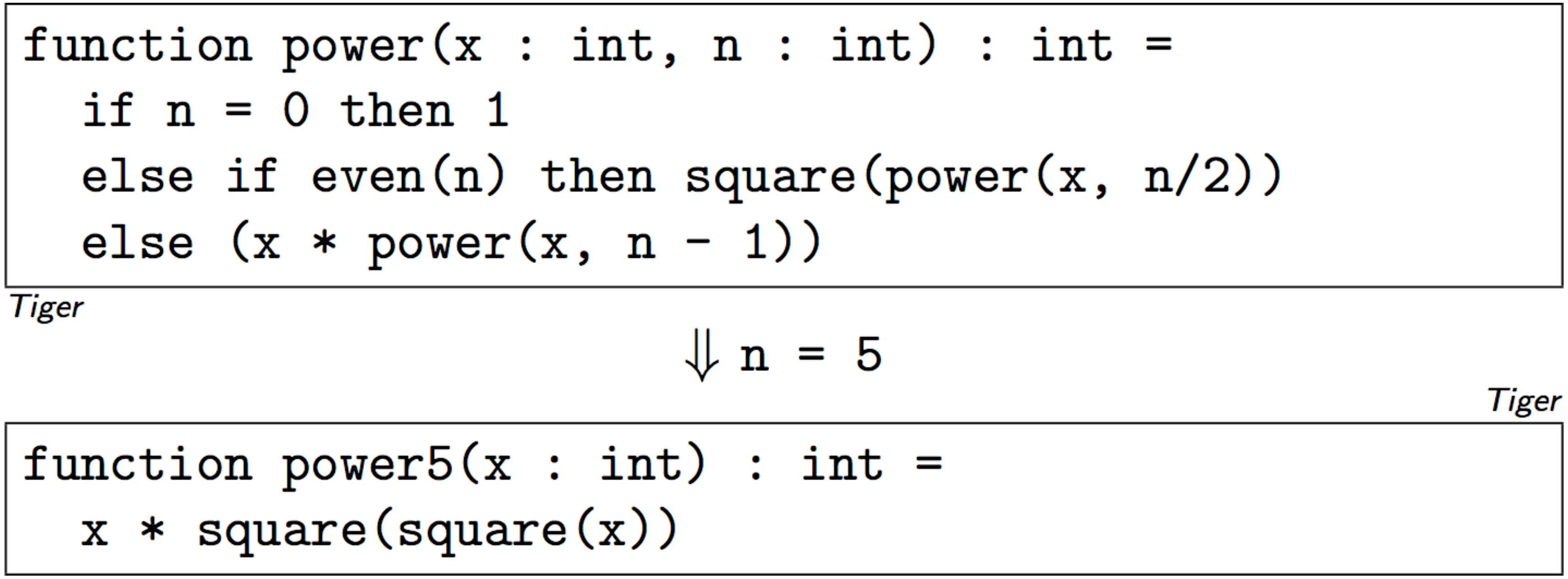
⇒

```
Exp   := Id
       |  Id "(" Exps ")"
       |  Exp "+" Exp
       |  ...

Exps  :=
       | Expp

Expp  := Exp
       |  Expp "," Exp
```

BNF

http://www.cs.uu.nl/docs/vakken/pt/slides/PT05-ProgramTransformation.pdf

Rephrasing — partial evaluation



```
function power(x : int, n : int) : int =
  if n = 0 then 1
  else if even(n) then square(power(x, n/2))
  else (x * power(x, n - 1))
```

Tiger

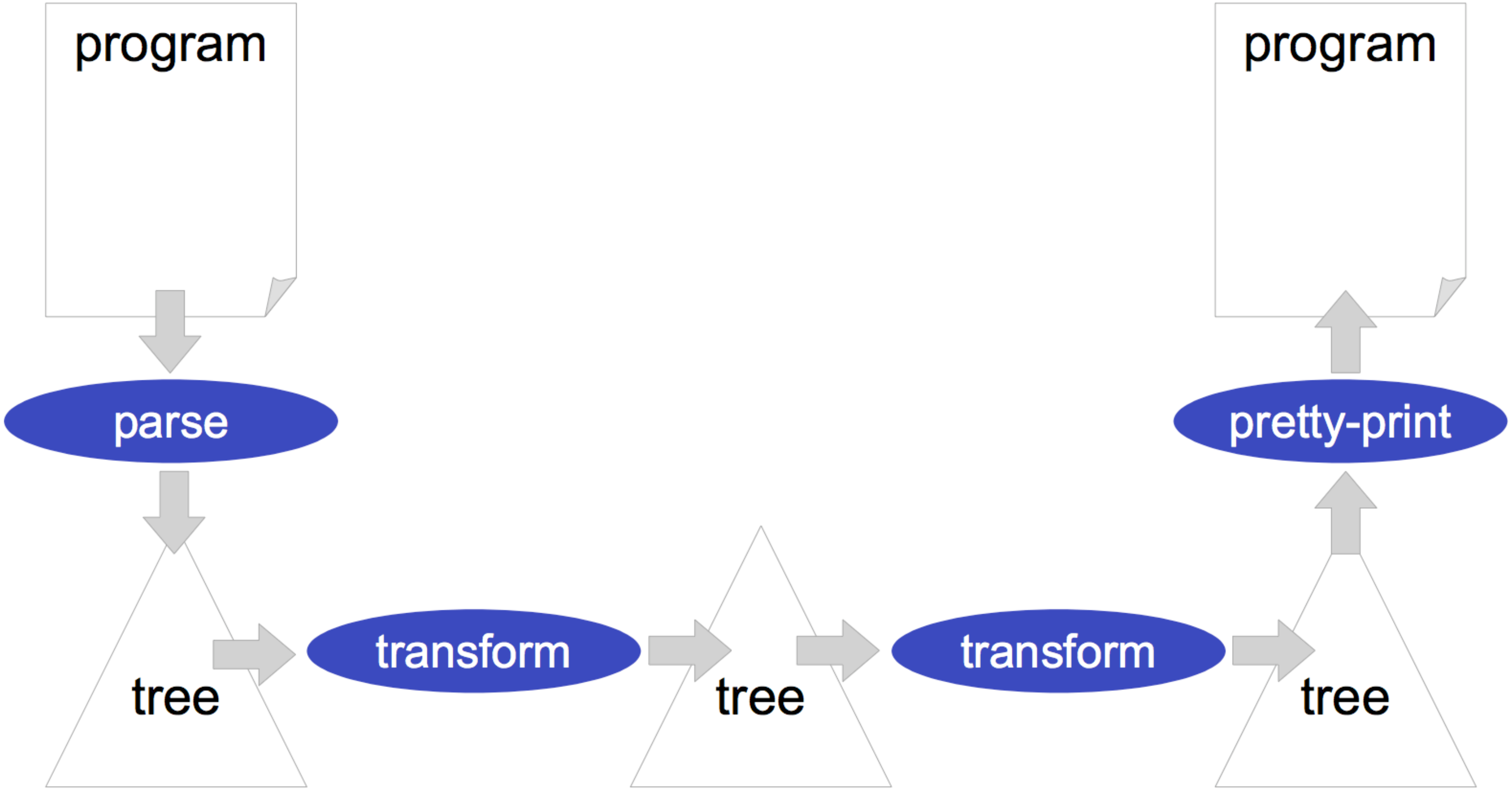
⇓ n = 5

Tiger

```
function power5(x : int) : int =
  x * square(square(x))
```

http://www.cs.uu.nl/docs/vakken/pt/slides/PT05-ProgramTransformation.pdf

Transformation pipeline



http://losser.st-lab.cs.uu.nl/~mbravenb/PT05-Infrastructure.pdf

Roadmap

- **Program Transformation**
  - Introduction
  - **Stratego/XT**
  - TXL
- Refactoring
- Aspect-Oriented Programming
- Outlook

Thanks to Eelco Visser and Martin Bravenboer for their kind permission to reuse and adapt selected material from their Program Transformation course. http://swerl.tudelft.nl/bin/view/Pt

Stratego/XT

> ***Stratego***

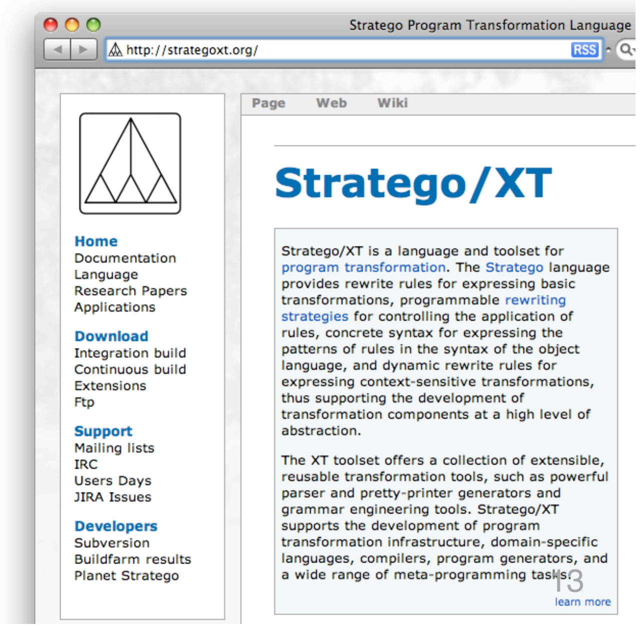
— A language for specifying program transformations

- term rewriting rules
- programmable rewriting strategies
- pattern-matching against syntax of object language
- context-sensitive transformations

> ***XT***

— A collection of transformation tools

- parser and pretty printer generators
- grammar engineering tools



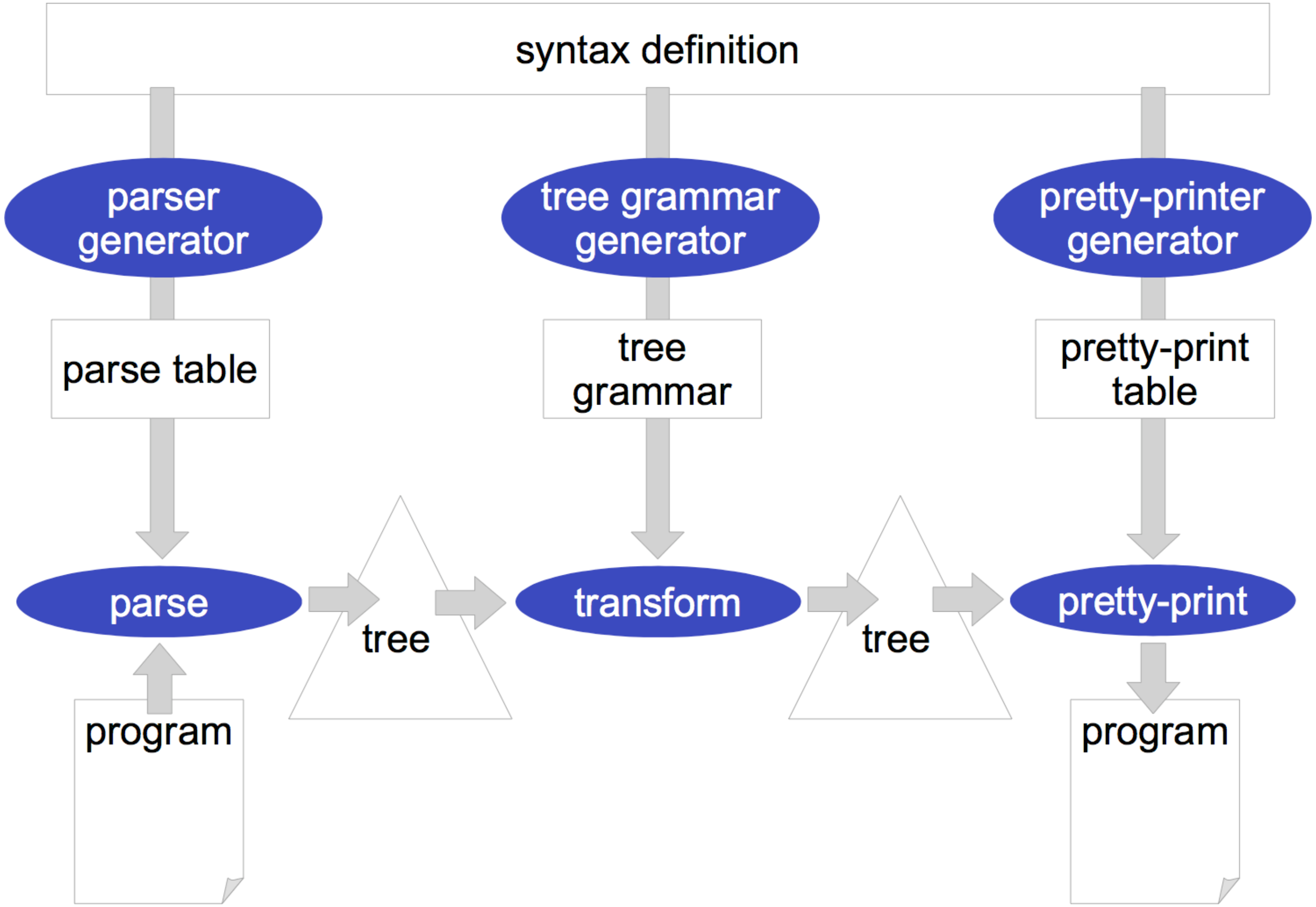
Stratego/XT

Stratego/XT is a language and toolset for program transformation. The Stratego language provides rewrite rules for expressing basic transformations, programmable rewriting strategies for controlling the application of rules, concrete syntax for expressing the patterns of rules in the syntax of the object language, and dynamic rewrite rules for expressing context-sensitive transformations, thus supporting the development of transformation components at a high level of abstraction.

The XT toolset offers a collection of extensible, reusable transformation tools, such as powerful parser and pretty-printer generators and grammar engineering tools. Stratego/XT supports the development of program transformation infrastructure, domain-specific languages, compilers, program generators, and a wide range of meta-programming tasks.

http://strategoxt.org/

Stratego/XT



http://losser.st-lab.cs.uu.nl/~mbravenb/PT05-Infrastructure.pdf

Parsing

Rules translate
terms to *terms*

Stratego parses any context-free language using Scannerless Generalized LR Parsing

File: Exp.sdf

```
module Exp
exports
  context-free start-symbols Exp
  sorts Id IntConst Exp

  lexical syntax
    [\ \t\n]   -> LAYOUT
    [a-zA-Z]+ -> Id
    [0-9]+    -> IntConst

  context-free syntax
    Id         -> Exp {cons("Var")}
    IntConst   -> Exp {cons("Int")}

    "(" Exp ")" -> Exp   {bracket}

    Exp "*"  Exp -> Exp  {left, cons("Mul")}
    Exp "/"  Exp -> Exp  {left, cons("Div")}
    Exp "%"  Exp -> Exp  {left, cons("Mod")}

    Exp "+"  Exp -> Exp  {left, cons("Plus")}
    Exp "-"  Exp -> Exp  {left, cons("Minus")}

  context-free priorities
    {left:
      Exp "*"  Exp -> Exp
      Exp "/"  Exp -> Exp
      Exp "%"  Exp -> Exp
    }
  > {left:
      Exp "+"  Exp -> Exp
      Exp "-"  Exp -> Exp
    }
```

Testing

File: Exp.testsuite

```
testsuite Exp
topsort Exp

test eg1 parse
  "1 + 2 * (3 + 4) * 3 - 1"
->
  Minus(
    Plus(
      Int("1")
    , Mul(
        Mul(Int("2"), Plus(Int("3"), Int("4")))
      , Int("3")
      )
    )
  , Int("1")
  )
```

Running tests

```
pack-sdf -i Exp.sdf -o Exp.def
  including ./Exp.sdf
```

Pack the definitions

```
sdf2table -i Exp.def -o Exp.tbl -m Exp
SdfChecker:error: Main module not defined
--- Main
```

Generate the parse table

```
parse-unit -i Exp.testsuite -p Exp.tbl
-----------------------------------------------------------------------
executing testsuite Exp with 1 tests
-----------------------------------------------------------------------
* OK   : test 1 (eg1 parse)
-----------------------------------------------------------------------
results testsuite Exp
successes : 1
failures  : 0
-----------------------------------------------------------------------
```

Run the tests

Interpretation example

File: ExpEval.str

```
module ExpEval

imports libstratego-lib
imports Exp

rules
  convert : Int(x) -> <string-to-int>(x)
  eval : Plus(m,n) -> <add>(m,n)
  eval : Minus(m,n) -> <subt>(m,n)
  eval : Mul(m,n) -> <mul>(m,n)
  eval : Div(m,n) -> <div>(m,n)
  eval : Mod(m,n) -> <mod>(m,n)

strategies
  main = io-wrap(innermost(convert <+ eval))
```

File: ultimate-question.txt

```
1 + 2 * (3 + 4) * 3 - 1
```

Stratego separates the specification of *rules* (transformations) from *strategies* (traversals). In principle, both are reusable.

Strategies

A *strategy* determines how a set of rewrite rules will be used to traverse and transform a term.

- innermost
- top down
- bottom up
- repeat
- …

Running the transformation

```
sdf2rtg -i Exp.def -o Exp.rtg -m Exp
SdfChecker:error: Main module not defined
--- Main
```

Generate regular tree grammar

```
rtg2sig -i Exp.rtg -o Exp.str
```

Generate signature

```
strc -i ExpEval.str -la stratego-lib
[ strc | info ] Compiling 'ExpEval.str'
[ strc | info ] Front-end succeeded          : [user/system] = [0.56s/0.05s]
[ strc | info ] Optimization succeeded -O 2 : [user/system] = [0.00s/0.00s]
[ strc | info ] Back-end succeeded           : [user/system] = [0.16s/0.01s]
 gcc -I /usr/local/strategoxt/include -I /usr/local/strategoxt/include -I /usr/local/
strategoxt/include -Wall -Wno-unused-label -Wno-unused-variable -Wno-unused-function -Wno-
unused-parameter -DSIZEOF_VOID_P=4 -DSIZEOF_LONG=4 -DSIZEOF_INT=4 -c ExpEval.c  -fno-common -
DPIC -o .libs/ExpEval.o
 gcc -I /usr/local/strategoxt/include -I /usr/local/strategoxt/include -I /usr/local/
strategoxt/include -Wall -Wno-unused-label -Wno-unused-variable -Wno-unused-function -Wno-
unused-parameter -DSIZEOF_VOID_P=4 -DSIZEOF_LONG=4 -DSIZEOF_INT=4 -c ExpEval.c -o ExpEval.o
>/dev/null 2>&1
gcc .libs/ExpEval.o -o ExpEval -bind_at_load  -L/usr/local/strategoxt/lib /usr/local/
strategoxt/lib/libstratego-lib.dylib /usr/local/strategoxt/lib/libstratego-lib-native.dylib /
usr/local/strategoxt/lib/libstratego-runtime.dylib -lm /usr/local/strategoxt/lib/
libATerm.dylib
[ strc | info ] C compilation succeeded      : [user/system] = [0.31s/0.36s]
[ strc | info ] Compilation succeeded        : [user/system] = [1.03s/0.42s]
```

Compile to C

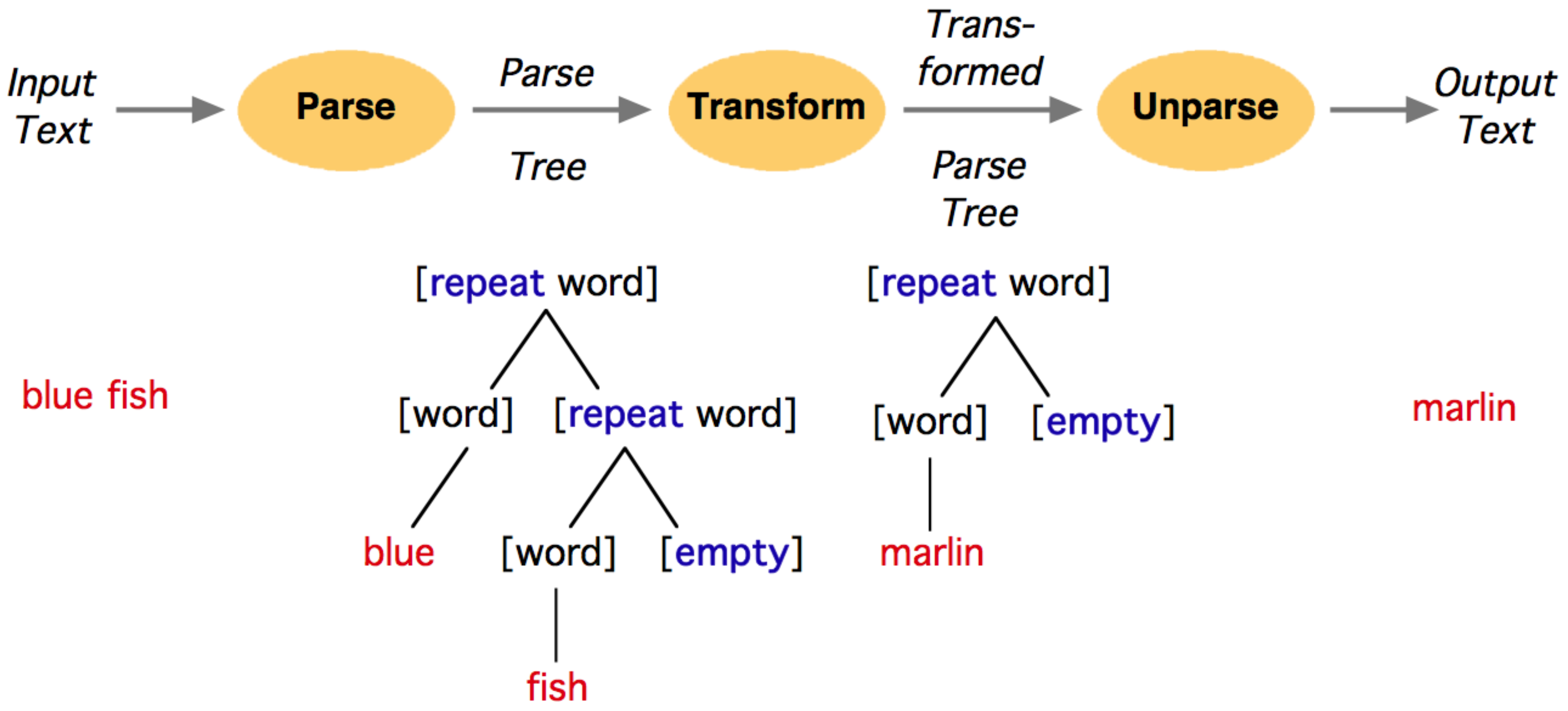
```
sglri -p Exp.tbl -i ultimate-question.txt | ./ExpEval
42
```

Parse and transform

Roadmap

- > **Program Transformation**
  - — Introduction
  - — Stratego/XT
  - — **TXL**
- > Refactoring
- > Aspect-Oriented Programming
- > Outlook

The TXL paradigm: *parse, transform, unparse*



http://www.txl.ca/docs/TXLintro.pdf

TXL programs

Base grammar	*defines tokens and non-terminals*
Grammar overrides	*extend and modify types from grammar*
Transformation rules	*rooted set of rules and functions*

Expression example

File: ExpEval.str

```
% Part I.  Syntax specification
define program
        [expression]
end define

define expression
        [expression] + [term]
    |   [expression] - [term]
    |   [term]
end define

define term
        [term] * [primary]
    |   [term] / [primary]
    |   [primary]
end define

define primary
        [number]
    |   ( [expression] )
end define
```

```
% Part 2.  Transformation rules
rule main
    replace [expression]
        E [expression]
    construct NewE [expression]
        E [resolveAddition]
          [resolveSubtraction]
          [resolveMultiplication]
          [resolveDivision]
          [resolveBracketedExpressions]
    where not
        NewE [= E]
    by
        NewE
end rule

rule resolveAddition
    replace [expression]
        N1 [number] + N2 [number]
    by
        N1 [+ N2]
end rule
...

rule resolveBracketedExpressions
    replace [primary]
        ( N [number] )
    by
        N
end rule
```

Running the example

File: ultimate-question.txt

```
1 + 2 * (3 + 4) * 3 - 1
```

```
txl Ultimate.Question
TXL v10.5d (1.7.08) (c)1988-2008 Queen's University at Kingston
Compiling Question.Txl ...
Parsing Ultimate.Question ...
Transforming ...
42
```

Example: TIL — a tiny imperative language

```
// Find all factors of a given input number
var n;
write "Input n please";
read n;
write "The factors of n are";
var f;
f := 2;
while n != 1 do
    while (n / f) * f = n do
        write f;
        n := n / f;
    end
    f := f + 1;
end
```

File: factors.til

http://www.program-transformation.org/Sts/TILChairmarks

TIL Grammar

File: TIL.Grm

```
% Keywords of TIL
keys
    var if then else while
    do for read write
end keys

% Compound tokens
compounds
    :=  !=
end compounds

% Commenting convention
comments
    //
end comments
```

All TXL parsers are also pretty-printers if the grammar includes formatting cues

```
define program
    [statement*]
end define

define statement
        [declaration]
    |   [assignment_statement]
    |   [if_statement]
    |   [while_statement]
    |   [for_statement]
    |   [read_statement]
    |   [write_statement]
end define

% Untyped variables
define declaration
    'var [id] ;                   [NL]
end define

define assignment_statement
    [id] := [expression] ;        [NL]
end define

define if_statement
    'if [expression] 'then        [IN][NL]
        [statement*]              [EX]
    [opt else_statement]
    'end                          [NL]
end define
...
```

Pretty-printing TIL

```
include "TIL.Grm"
function main
   match [program]
      _ [program]
end function
```

File: TILparser.Txl

```
txl factors.til TILparser.Txl
```

```
var n;
write "Input n please";
read n;
write "The factors of n are";
var f;
f := 2;
while n != 1 do
    while (n / f) * f = n do
        write f;
        n := n / f;
    end
    f := f + 1;
end
```

Generating statistics

File: TILstats.Txl

```
include "TIL.Grm"

function main
    replace [program]
        Program [program]

    % Count each kind of statement we're interested in
    % by extracting all of each kind from the program

    construct Statements [statement*]
        _ [^ Program]
    construct StatementCount [number]
        _ [length Statements] [putp "Total: %"]

    construct Declarations [declaration*]
        _ [^ Program]
    construct DeclarationsCount [number]
        _ [length Declarations] [putp "Declarations: %”]
...
    by
        % nothing
end function
```

```
Total: 11
Declarations: 2
Assignments: 3
Ifs: 0
Whiles: 2
Fors: 0
Reads: 1
Writes: 3
```

Tracing

File: TILtrace.Txl

```
include "TIL.Grm"
...
redefine statement
        ...
    |    [traced_statement]
end redefine

define traced_statement
    [statement] [attr 'TRACED]
end define

rule main
replace [repeat statement]
        S [statement]
        Rest [repeat statement]
...
    by
        'write QuotedS;     'TRACED
        S                   'TRACED
        Rest
end rule

...
```

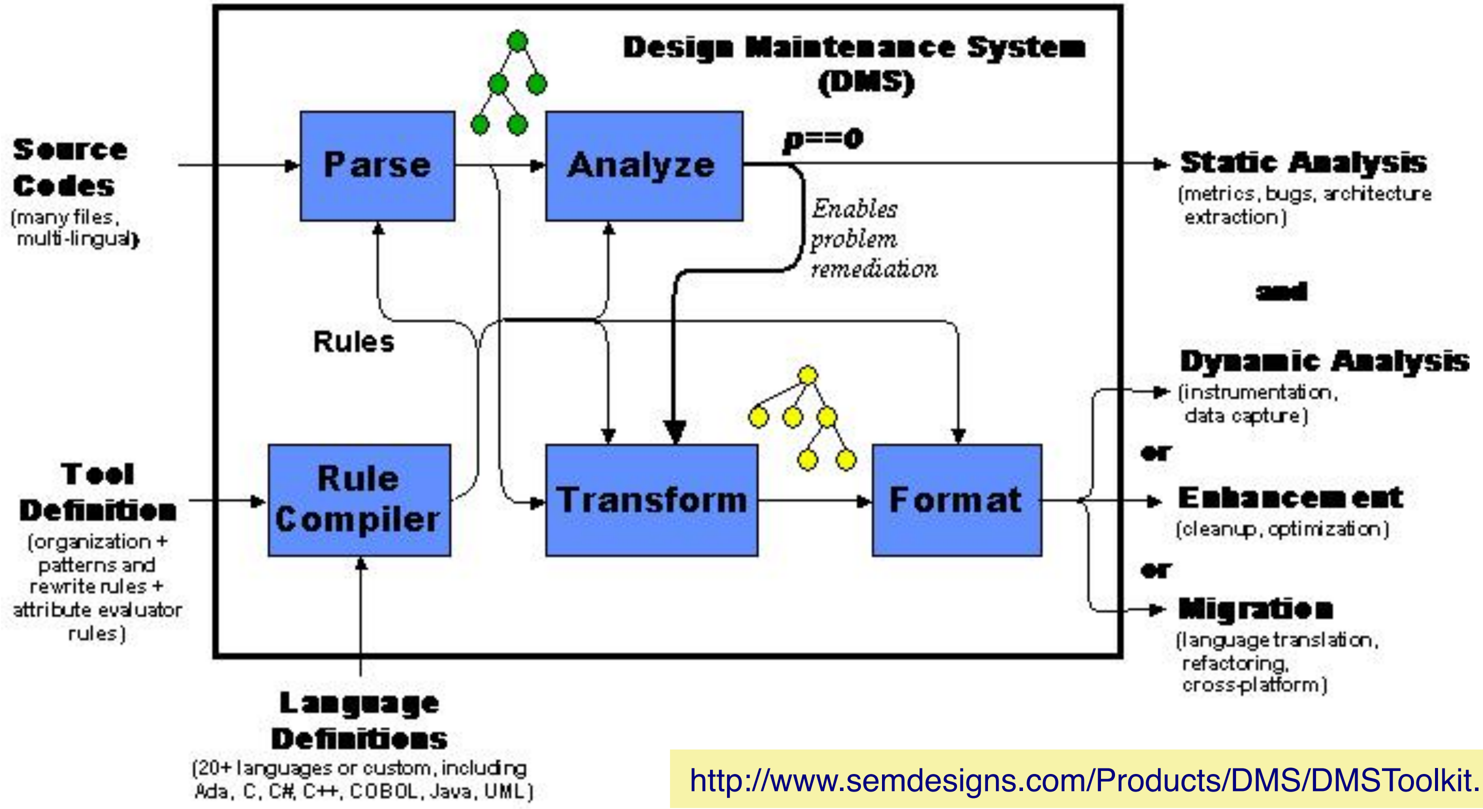
```
write "Trace: var n;";
var n;
write "Trace: write \"Input n please\";";
write "Input n please";
write "Trace: read n;";
read n;
...
```

TXL vs Stratego

Stratego	***TXL***
Scannerless GLR parsing	Agile parsing (top-down + bottom-up)
Reusable, generic traversal strategies	Fixed traversals
Separates rewrite rules from traversal strategies	Traversals part of rewrite rules

Commercial systems

“The DMS Software Reengineering Toolkit is a set of tools for automating customized source program analysis, modification or translation or generation of software systems, containing arbitrary mixtures of languages.”


Design Maintenance System
(DMS)
Source Codes
(many files, multi-lingual)
Parse
Analyze
p==0
Enables problem remediation
Static Analysis
(metrics, bugs, architecture extraction)
and
Dynamic Analysis
(instrumentation, data capture)
or
Enhancement
(cleanup, optimization)
or
Migration
(language translation, refactoring, cross-platform)
Rules
Tool Definition
(organization + patterns and rewrite rules + attribute evaluator rules)
Rule Compiler
Transform
Format
Language Definitions
(20+ languages or custom, including Ada, C, C#, C++, COBOL, Java, UML)

http://www.semdesigns.com/Products/DMS/DMSToolkit.html

Roadmap

- Program Transformation
- **Refactoring**
  - **Refactoring Engine and Code Critics**
  - Eclipse refactoring plugins
- Aspect-Oriented Programming
- Outlook

What is Refactoring?

> The process of *changing a software system* in such a way that it *does not alter the external behaviour* of the code, yet *improves its internal structure*.

— Fowler, et al., Refactoring, 1999.

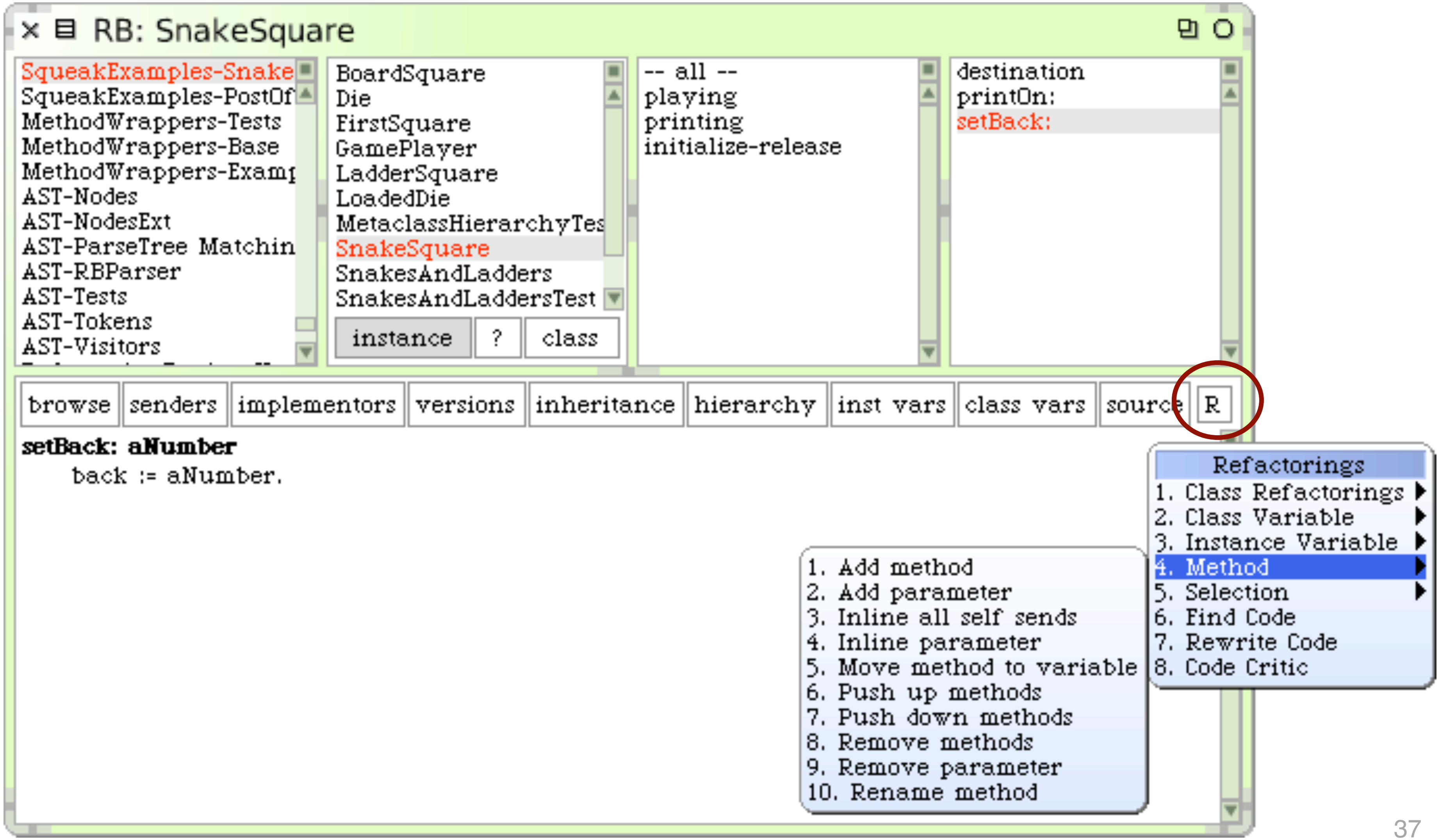
Rename Method — manual steps

- Do it yourself approach:
  - Check that no method with the new name already exists in any subclass or superclass.
  - Browse all the implementers (method definitions)
  - Browse all the senders (method invocations)
  - Edit and rename all implementers
  - Edit and rename all senders
  - Remove all implementers
  - Test
- Automated refactoring is better !

Rename Method

- Rename Method (method, new name)
- Preconditions
  - No method with the new name already exists in any subclass or superclass.
  - No methods with same signature as method outside the inheritance hierarchy of method
- PostConditions
  - method has new name
  - relevant methods in the inheritance hierarchy have new name
  - invocations of changed method are updated to new name
- Other Considerations
  - Typed/Dynamically Typed Languages => Scope of the renaming

The Refactoring Browser



```
setBack: aNumber
    back := aNumber.
```

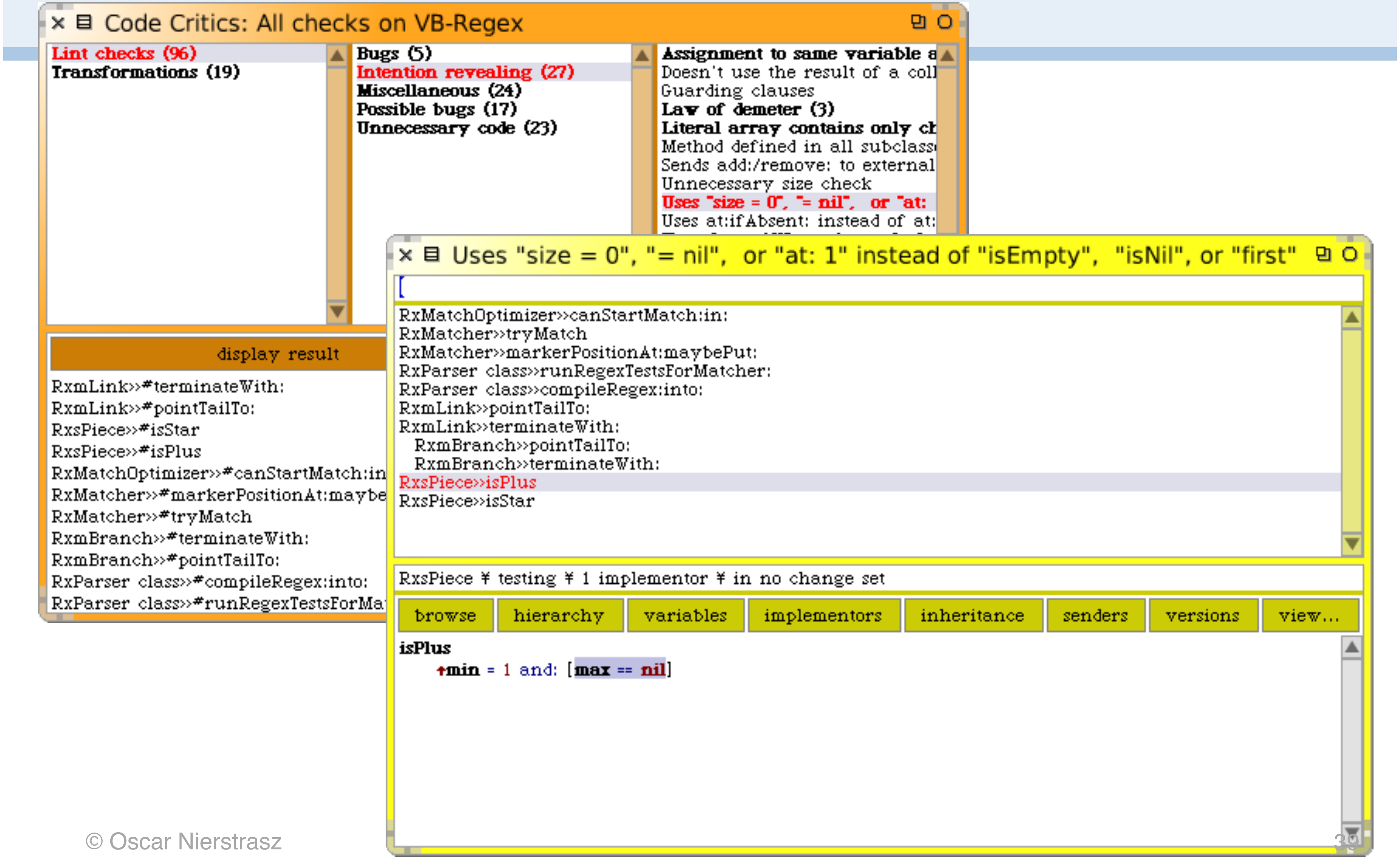
Typical Refactorings

Class Refactorings	Method Refactorings	Attribute Refactorings
add (sub)class to hierarchy	add method to class	add variable to class
rename class	rename method	rename variable
remove class	remove method	remove variable
	push method down	push variable down
	push method up	pull variable up
	add parameter to method	create accessors
	move method to component	abstract variable
	extract code in new method	

Bill Opdyke, “Refactoring Object-Oriented Frameworks,” Ph.D. thesis, University of Illinois, 1992.

Don Roberts, “Practical Analysis for Refactoring,” Ph.D. thesis, University of Illinois, 1999.

Code Critic — search for common errors



```
Lint checks (96)
Transformations (19)
```

```
Bugs (5)
Intention revealing (27)
Miscellaneous (24)
Possible bugs (17)
Unnecessary code (23)
```

```
Assignment to same variable a
Doesn't use the result of a coll
Guarding clauses
Law of demeter (3)
Literal array contains only ch
Method defined in all subclass
Sends add:/remove: to external
Unnecessary size check
Uses "size = 0", "= nil", or "at:
Uses at:ifAbsent: instead of at:
```

```
RxmLink>>#terminateWith:
RxmLink>>#pointTailTo:
RxsPiece>>#isStar
RxsPiece>>#isPlus
RxMatchOptimizer>>#canStartMatch:in
RxMatcher>>#markerPositionAt:maybe
RxMatcher>>#tryMatch
RxmBranch>>#terminateWith:
RxmBranch>>#pointTailTo:
RxParser class>>#compileRegex:into:
RxParser class>>#runRegexTestsForMa
```

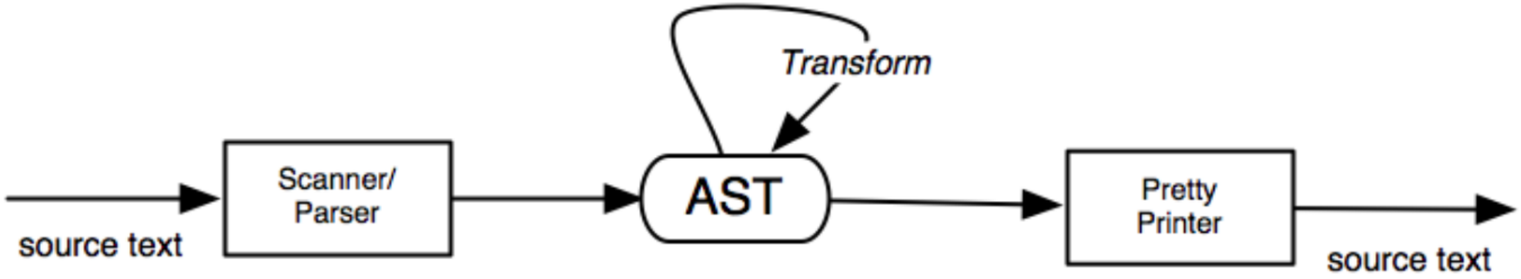
Uses "size = 0", "= nil", or "at: 1" instead of "isEmpty", "isNil", or "first"

```
RxMatchOptimizer>>canStartMatch:in:
RxMatcher>>tryMatch
RxMatcher>>markerPositionAt:maybePut:
RxParser class>>runRegexTestsForMatcher:
RxParser class>>compileRegex:into:
RxmLink>>pointTailTo:
RxmLink>>terminateWith:
  RxmBranch>>pointTailTo:
  RxmBranch>>terminateWith:
RxsPiece>>isPlus
RxsPiece>>isStar
```

RxsPiece ¥ testing ¥ 1 implementor ¥ in no change set

```
isPlus
    ↑min = 1 and: [max == nil]
```

Refactoring Engine — matching trees

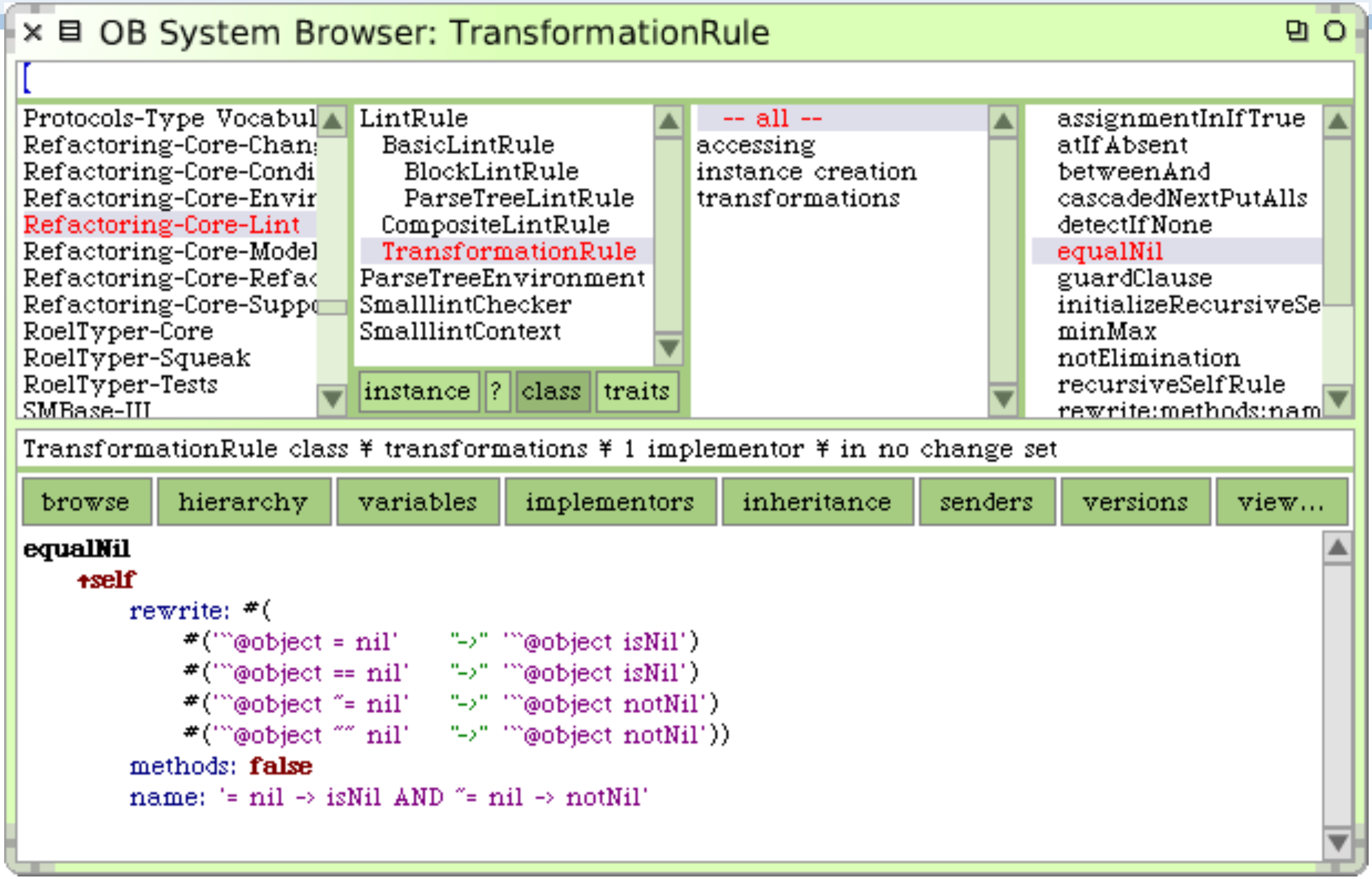


NB: All metavariables start with `

Syntax	Type
`	recurse
@	list
.	statement
#	literal

``@object halt	recursively match send of halt
`@.Statements	match list of statements
Class `@message: `@args	match all sends to Class

Rewrite rules



TransformationRule class ¥ transformations ¥ 1 implementor ¥ in no change set

```
equalNil
    ↑self
        rewrite: #(
            #('``@object = nil'    "->"  '``@object isNil')
            #('``@object == nil'   "->"  '``@object isNil')
            #('``@object ~= nil'   "->"  '``@object notNil')
            #('``@object ~~ nil'   "->"  '``@object notNil'))
        methods: false
        name: '= nil -> isNil AND ~= nil -> notNil'
```

Roadmap

- Program Transformation
- **Refactoring**
  - Refactoring Engine and Code Critics
  - **Eclipse refactoring plugins**
- Aspect-Oriented Programming
- Outlook

Thanks to Lea Hänsenberger for the plugin code.

A workbench action delegate

When the workbench action proxy is triggered by the user, it delegates to an instance of this class.

```
package astexampleplugin.actions;
...
import org.eclipse.ui.IWorkbenchWindowActionDelegate;

public class ChangeAction implements IWorkbenchWindowActionDelegate {
    ...
    public void run( IAction action ) {
        for ( ICompilationUnit cu : this.classes ) {
            try {
                ...
                parser.setSource( cu );
                ...
                CompilationUnit ast = (CompilationUnit)parser.createAST( null );
                ...
                StackVisitor visitor = new StackVisitor( ast.getAST() );
                ast.accept( visitor );
                ...

            } catch ...
        }
    }
    ...
}
```

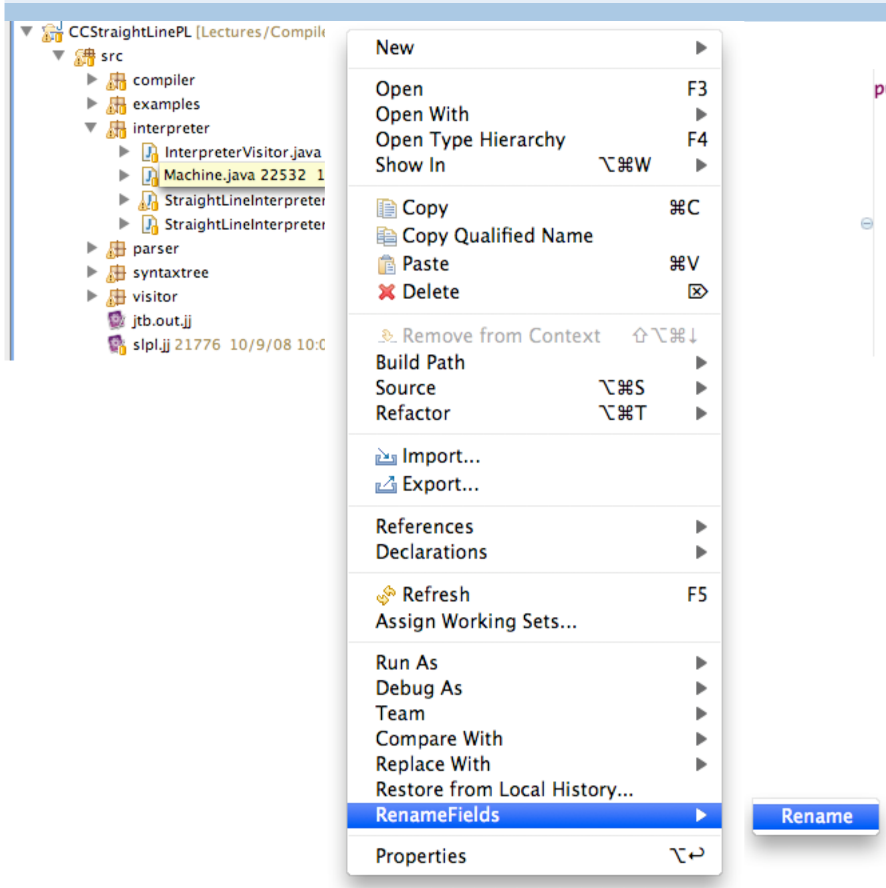
http://help.eclipse.org/ganymede/index.jsp?topic=/org.eclipse.jdt.doc.isv/guide/jdt_api_manip.htm

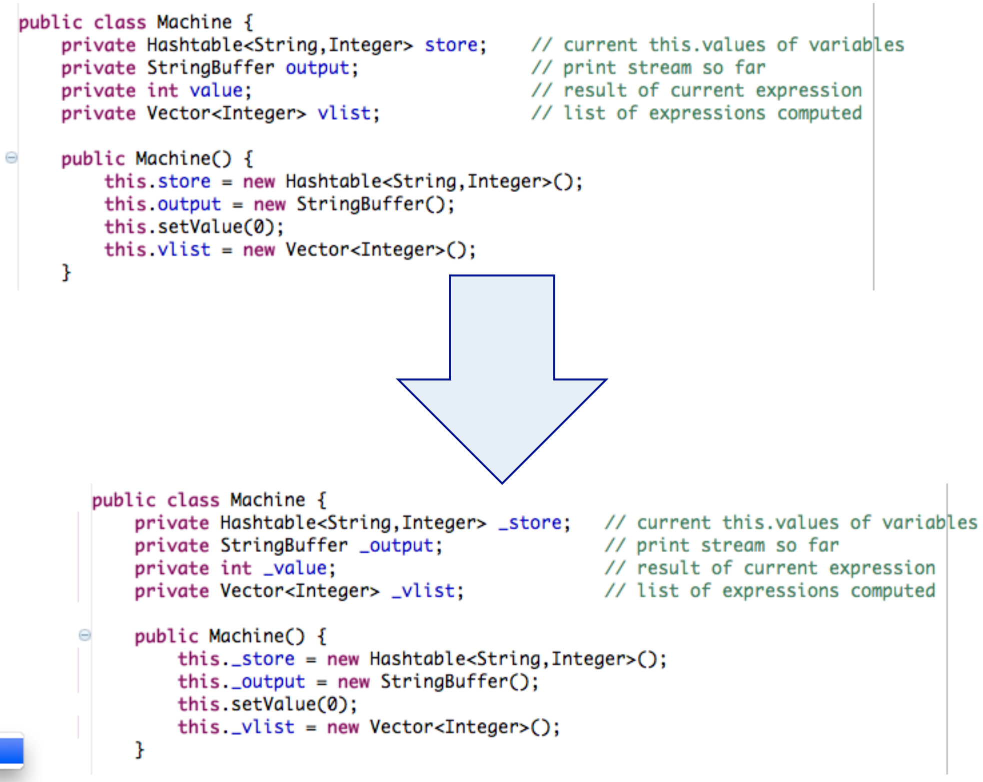
A field renaming visitor

```
package astexampleplugin.ast;
...
import org.eclipse.jdt.core.dom.ASTVisitor;

public class StackVisitor extends ASTVisitor {

    private static final String PREFIX = "_";
    ...
    public boolean visit(FieldDeclaration field){
        ...
    }

    public boolean visit(FieldAccess fieldAccess){
        String oldName = fieldAccess.getName().toString();
        String newName = this.fields.get( oldName );
        if(newName == null){
            newName = PREFIX + oldName;
            this.fields.put( oldName , newName );
        }
        fieldAccess.setName( this.ast.newSimpleName( newName ) );
        return true;
    }
}
```

The visitor simply implements the visit method for field declarations and accesses, and prepends an underscore.

Renaming fields





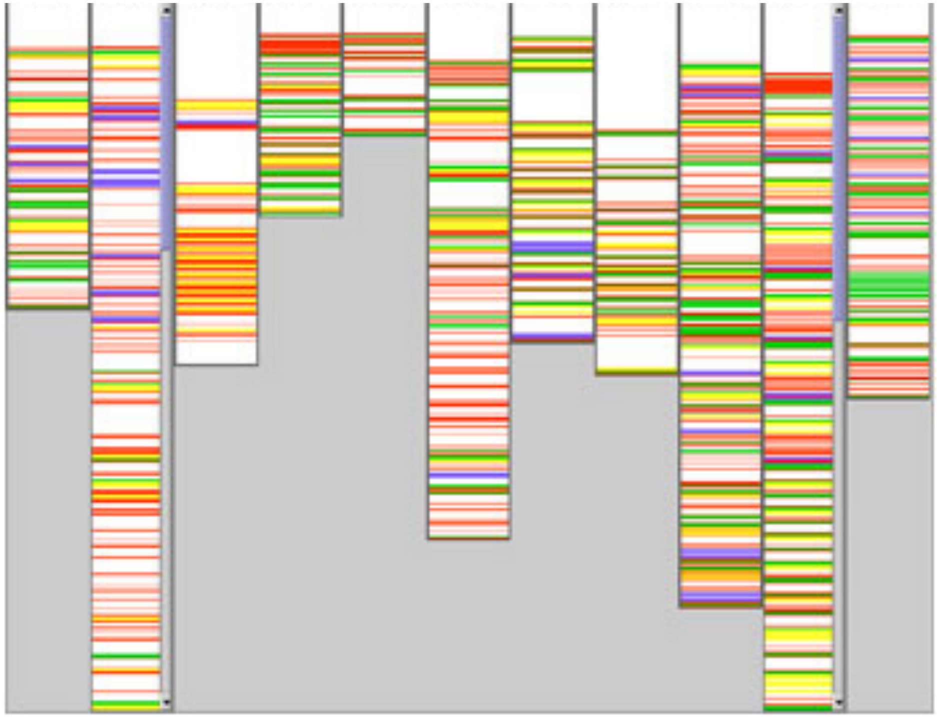
```
public class Machine {
    private Hashtable<String,Integer> store;    // current this.values of variables
    private StringBuffer output;                // print stream so far
    private int value;                          // result of current expression
    private Vector<Integer> vlist;              // list of expressions computed

    public Machine() {
        this.store = new Hashtable<String,Integer>();
        this.output = new StringBuffer();
        this.setValue(0);
        this.vlist = new Vector<Integer>();
    }
```

```
public class Machine {
    private Hashtable<String,Integer> _store;   // current this.values of variables
    private StringBuffer _output;               // print stream so far
    private int _value;                         // result of current expression
    private Vector<Integer> _vlist;             // list of expressions computed

    public Machine() {
        this._store = new Hashtable<String,Integer>();
        this._output = new StringBuffer();
        this.setValue(0);
        this._vlist = new Vector<Integer>();
    }
```

Roadmap

- Program Transformation
- Refactoring
- **Aspect-Oriented Programming**
- Outlook

Problem: cross-cutting concerns



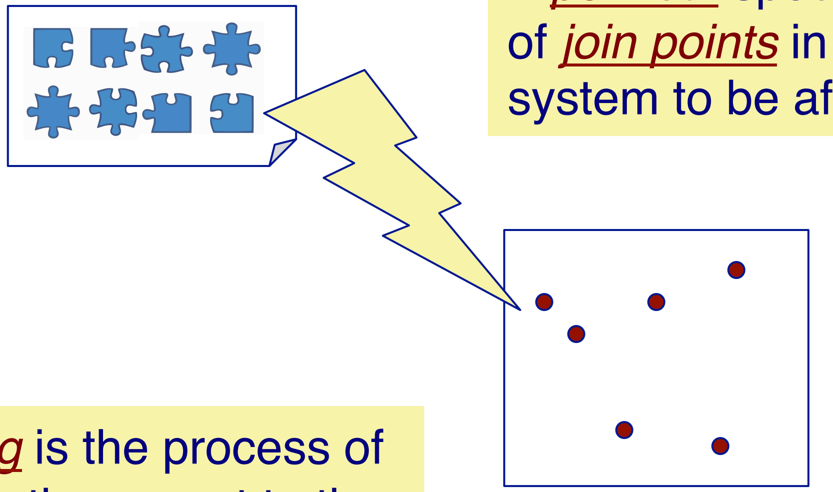
Certain features (like logging, persistence and security), cannot usually be encapsulated as classes. They *cross-cut* code of the system.

“Identifying Cross-Cutting Concerns in Embedded C Code”, Bruntink, van Deursen, Tourwé

Aspect-Oriented Programming

AOP improves modularity by supporting the separation of cross-cutting concerns.

An *aspect* packages cross-cutting concerns



A *pointcut* specifies a set of *join points* in the target system to be affected

Weaving is the process of applying the aspect to the target system

Canonical example — logging

```
package tjp;

public class Demo {
 static Demo d;
 public static void main(String[] args){
  new Demo().go();
 }
 void go(){
  d = new Demo();
  d.foo(1,d);
  System.out.println(d.bar(new Integer(3)));
 }
 void foo(int i, Object o){
  System.out.println("Demo.foo(" + i + ", " + o + ")\n");
 }
 String bar (Integer j){
  System.out.println("Demo.bar(" + j + ")\n");
  return "Demo.bar(" + j  + ")";
 }
}
```

```
Demo.foo(1, tjp.Demo@939b78e)
Demo.bar(3)
Demo.bar(3)
```

http://www.eclipse.org/aspectj/downloads.php

A logging aspect

Intercept execution within control flow of `Demo.go()`

Identify all methods within `Demo`

```
aspect GetInfo {

 pointcut goCut(): cflow(this(Demo) && execution(void go()));

 pointcut demoExecs(): within(Demo) && execution(* *(..));

 Object around(): demoExecs() && !execution(* go()) && goCut() {
  ...
 }

...
}
```

Wrap all methods except `Demo.go()`

A logging aspect

```
aspect GetInfo {
 ...
 Object around(): demoExecs() && !execution(* go()) && goCut() {
  println("Intercepted message: " +
    thisJoinPointStaticPart.getSignature().getName());
  println("in class: " +
    thisJoinPointStaticPart.getSi
  printParameters(thisJoinPoint)
  println("Running original meth
  Object result = proceed();
  println("  result: " + result
  return result;
 }
 ...
}
```

```
Intercepted message: foo
in class: tjp.Demo
Arguments:
  0. i : int = 1
  1. o : java.lang.Object = tjp.Demo@c0b76fa
Running original method:

Demo.foo(1, tjp.Demo@c0b76fa)
  result: null
Intercepted message: bar
in class: tjp.Demo
Arguments:
  0. j : java.lang.Integer = 3
Running original method:

Demo.bar(3)
  result: Demo.bar(3)
Demo.bar(3)
```

Making classes visitable with aspects

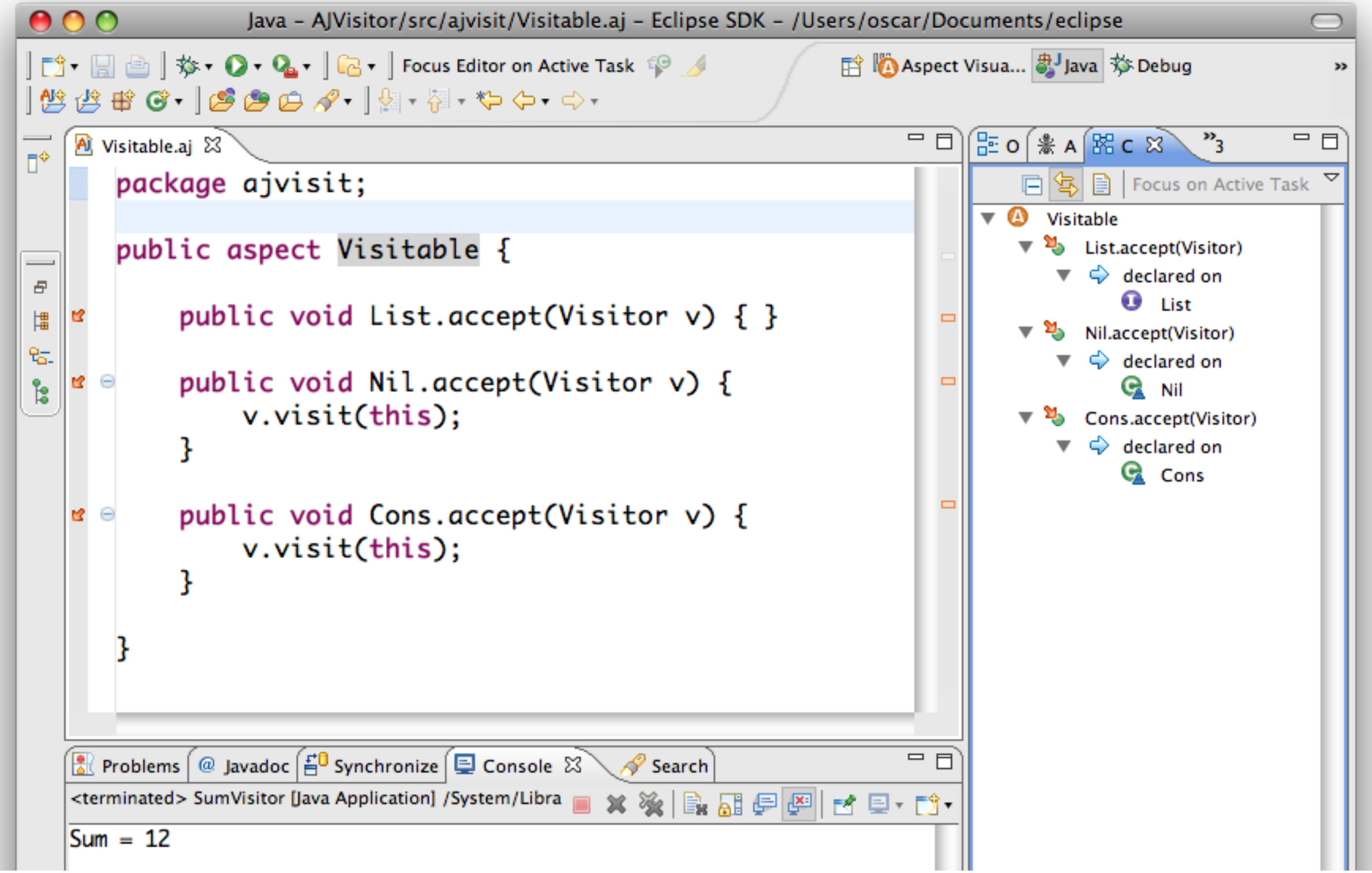
```
public class SumVisitor implements Visitor
{
 int sum = 0;
 public void visit(Nil l) { }

 public void visit(Cons l) {
  sum = sum + l.head;
  l.tail.accept(this);
 }

 public static void main(String[] args) {
  List l = new Cons(5, new Cons(4,
        new Cons(3, new Nil())));
  SumVisitor sv = new SumVisitor();
  l.accept(sv);
  System.out.println("Sum = " + sv.sum);
 }
}
public interface Visitor {
 void visit(Nil l);
 void visit(Cons l);
}
```

We want to write this

```
public interface List {}
public class Nil implements List {}
public class Cons implements List {
 int head;
 List tail;
 Cons(int head, List tail) {
  this.head = head;
  this.tail = tail;
 }
}
```

But we are stuck with this …

AspectJ



Java – AJVisitor/src/ajvisit/Visitable.aj – Eclipse SDK – /Users/oscar/Documents/eclipse
Focus Editor on Active Task
Aspect Visua...
Java
Debug
Visitable.aj
O
A
C
Focus on Active Task
Problems
Javadoc
Synchronize
Console
Search

```
package ajvisit;

public aspect Visitable {

    public void List.accept(Visitor v) { }

    public void Nil.accept(Visitor v) {
        v.visit(this);
    }

    public void Cons.accept(Visitor v) {
        v.visit(this);
    }

}
```

```
Visitable
    List.accept(Visitor)
        declared on
            List
    Nil.accept(Visitor)
        declared on
            Nil
    Cons.accept(Visitor)
        declared on
            Cons
```

```
<terminated> SumVisitor [Java Application] /System/Libra
Sum = 12
```

With aspects, who needs visitors?

This would be even cleaner

```
public class SumList {
 public static void main(String[] args) {
  List l = new Cons(5, new Cons(4, new Cons(3, new Nil())));
  System.out.println("Sum = " + l.sum());
 }
}
```

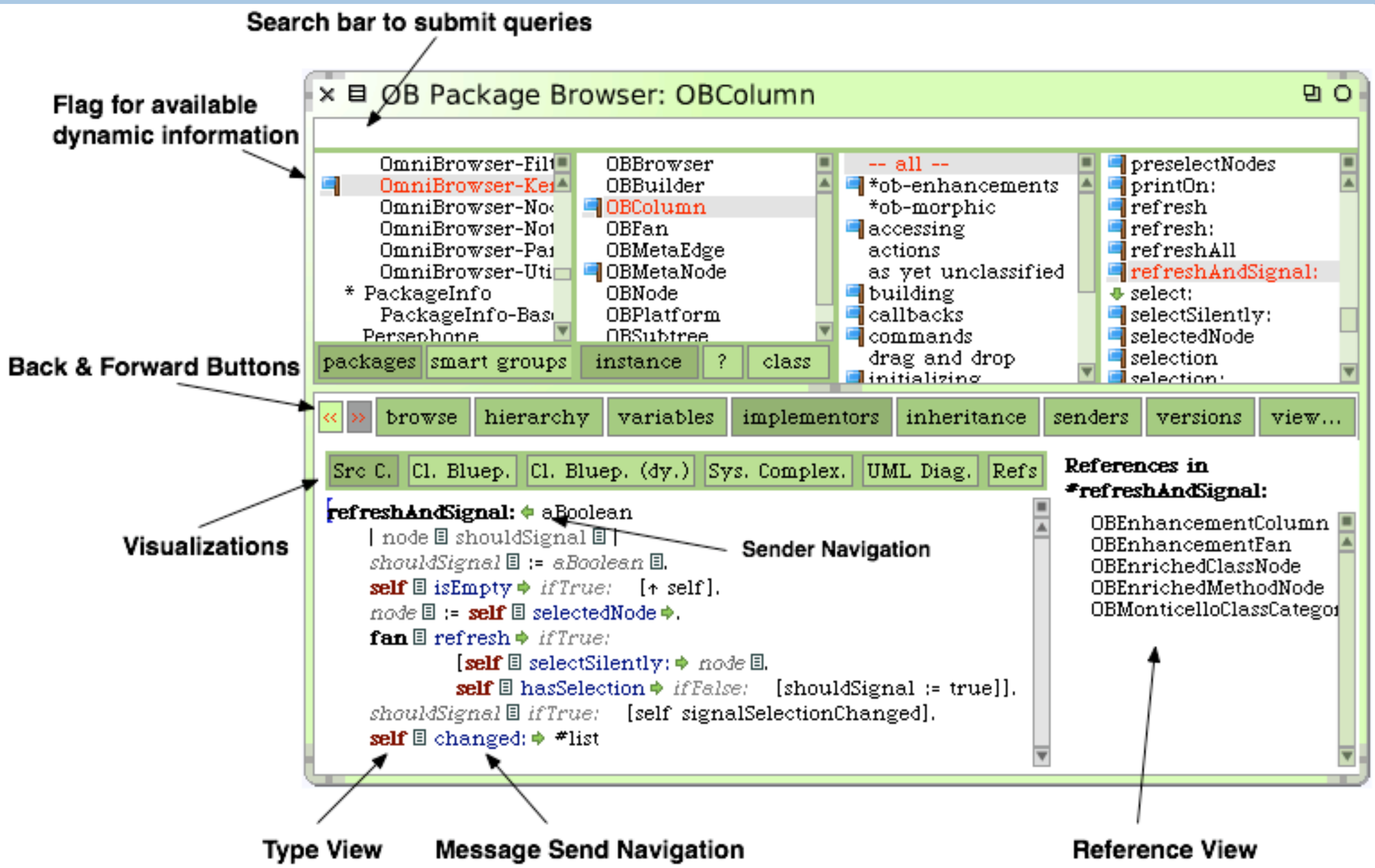
The missing method is just an aspect

```
public aspect Summable {
 public int List.sum() {
  return 0;
 }
 public int Nil.sum() {
  return 0;
 }
 public int Cons.sum() {
  return head + tail.sum();
 }
}
```

Roadmap

> Program Transformation
> Refactoring
> Aspect-Oriented Programming
> **Outlook**

Model-aware IDEs



Search bar to submit queries

Flag for available dynamic information

Back & Forward Buttons

Visualizations

Sender Navigation

Type View

Message Send Navigation

Reference View

OB Package Browser: OBColumn

```
OmniBrowser-Filt
OmniBrowser-Ker
OmniBrowser-No
OmniBrowser-Not
OmniBrowser-Pa
OmniBrowser-Uti
* PackageInfo
  PackageInfo-Bas
Persephone
```

```
OBBrowser
OBBuilder
OBColumn
OBFan
OBMetaEdge
OBMetaNode
OBNode
OBPlatform
OBSubtree
```

```
-- all --
*ob-enhancements
*ob-morphic
accessing
actions
as yet unclassified
building
callbacks
commands
drag and drop
initializing
```

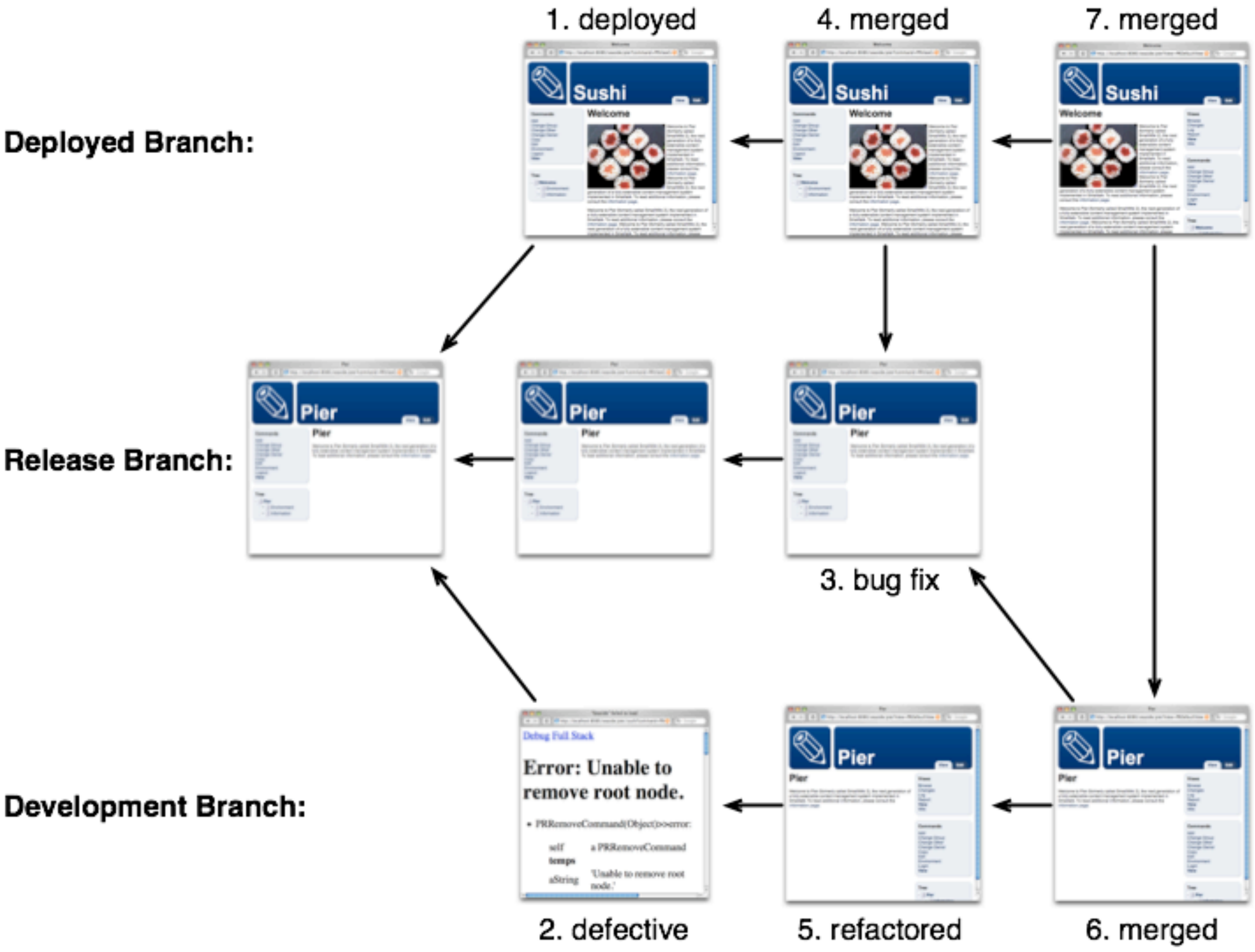
```
preselectNodes
printOn:
refresh
refresh:
refreshAll
refreshAndSignal:
select:
selectSilently:
selectedNode
selection
selection:
```

```
refreshAndSignal: aBoolean
    | node shouldSignal |
    shouldSignal := aBoolean.
    self isEmpty ifTrue: [↑ self].
    node := self selectedNode.
    fan refresh ifTrue:
            [self selectSilently: node.
            self hasSelection ifFalse: [shouldSignal := true]].
    shouldSignal ifTrue: [self signalSelectionChanged].
    self changed: #list
```

References in #refreshAndSignal:

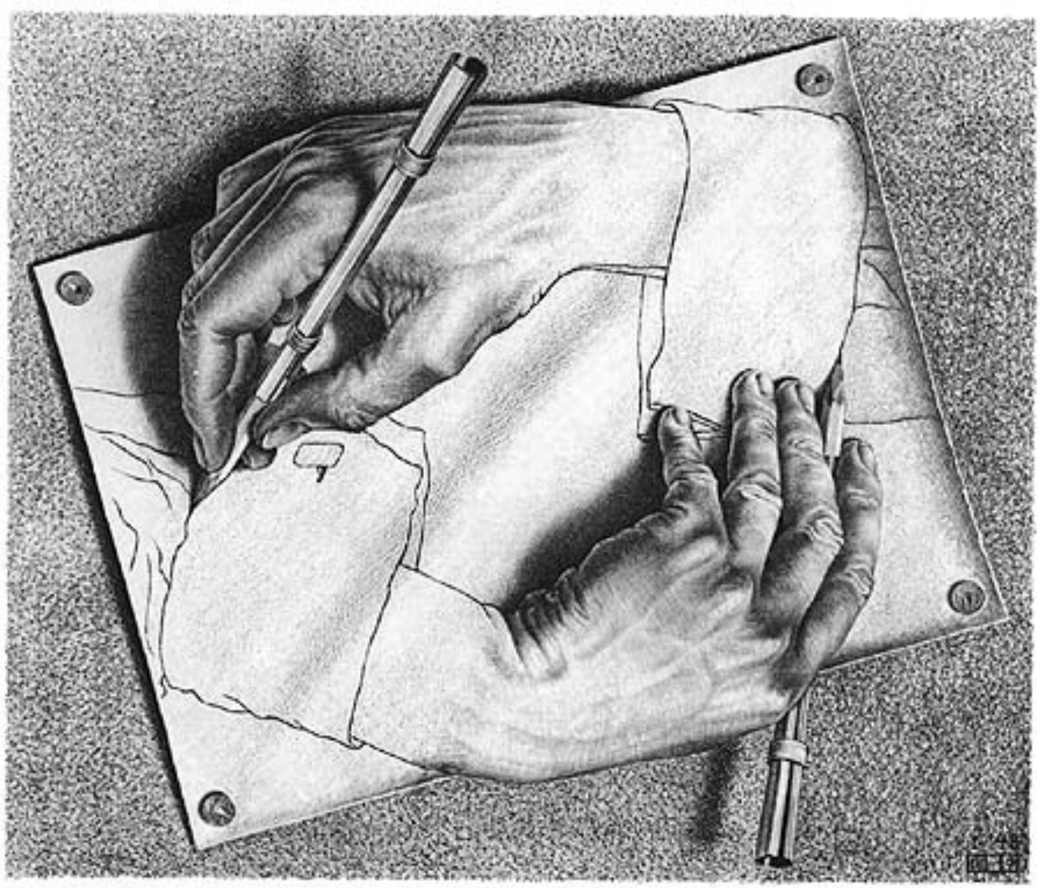
```
OBEnhancementColumn
OBEnhancementFan
OBEnrichedClassNode
OBEnrichedMethodNode
OBMonticelloClassCatego
```

Context-oriented programming with Changeboxes



Error: Unable to remove root node.

Model-centric development



Directly manipulate models without passing through source code …

What you should know!

- ✎ *What are typical program transformations?*
- ✎ *What is the typical architecture of a PT system?*
- ✎ *What is the role of term rewriting in PT systems?*
- ✎ *How does TXL differ from Stratego/XT?*
- ✎ *How does the Refactoring Engine use metavariables to encode rewrite rules?*
- ✎ *Why can’t aspects be encapsulated as classes?*
- ✎ *What is the difference between a pointcut and a join point?*

Can you answer these questions?

- ✎ *How does program transformation differ from metaprogramming?*
- ✎ *In what way is optimization a form of PT?*
- ✎ *What special care should be taken when pretty-printing a transformed program?*
- ✎ *How would you encode typical refactorings like “push method up” using a PT system like TXL?*
- ✎ *How could you use a PT system to implement AOP?*

License