

Refactoring Planning for Design Smell Correction in Object-Oriented Software

PhD Thesis

Francisco Javier Pérez García

supervised by
Yania Crespo González-Carvajal

Universidad de Valladolid - ETSII - Departamento de Informática - grupo GIRO

July 6th, 2011 - Valladolid

Outline

- ❖ Introduction
- ❖ State of the Art
- ❖ Thesis Proposal
 - ❖ Refactoring Strategies
 - ❖ Refactoring Planning
- ❖ Prototype
- ❖ Case Studies
- ❖ Conclusions

Introduction

Design Smells

Design smells

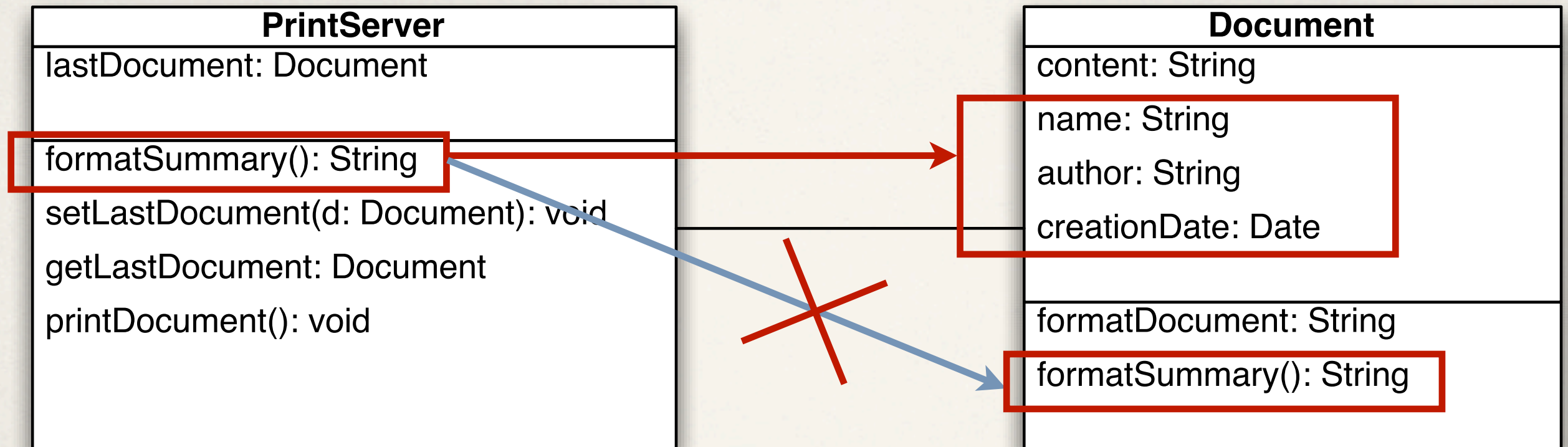
- ♦ Problems in software's structure
- ♦ Can be detected statically
- ♦ Do not produce compile-time or run-time errors
- ♦ Negatively affect software quality factors.
- ♦ Kent Beck and Martin Fowler. *Bad Smells in Code*, chapter 3. In *Refactoring: Improving the Design of Existing Code*, 1999.
- ♦ Referred as defects, flaws, disharmonies, antipatterns, by other authors.
- ♦ Design Smell is proposed as a unifying term.

Refactorings

Refactorings

- ♦ structural transformations
 - ♦ to perform design changes
 - ♦ without modifying the system's observable behaviour.
- ❖ **William Opdyke.** *Refactoring Object-Oriented Frameworks*. PhD Thesis, 1992
 - ♦ behaviour preserving invariants + preconditions + transformations
 - ❖ **Kent Beck and Martin Fowler.** *Refactoring: Improving the Design of Existing Code*, 1999.
 - ♦ bad smells as a guide for refactorings
 - ❖ **Low-level, composite and big refactorings**

The problem to solve



- ❖ `formatSummary()` of `PrintServer` suffers from **Feature Envy**
- ❖ Trivial strategy: apply **Move Method** from `PrintServer` to `Document`
- ❖ Refactoring **precondition violation**: same signature conflict
- ❖ **Additional refactorings** are needed to enable the precondition
- ❖ Different refactoring sequences **for each particular case**

The problem to solve

❖ What is the problem?

- ♦ Automated or semi-automated support
- ♦ To schedule sequences of refactorings (refactoring plans)
- ♦ To correct software design smells

❖ Why is it a problem?

- ♦ Preconditions can disable the application of a refactoring over the current system
- ♦ Refactoring sequences have to be planned ahead for each case

❖ Why it is an important problem?

- ♦ Design smells have a negative effect over software quality factors
- ♦ Strategic refactorings are complex refactoring processes

Thesis Objectives

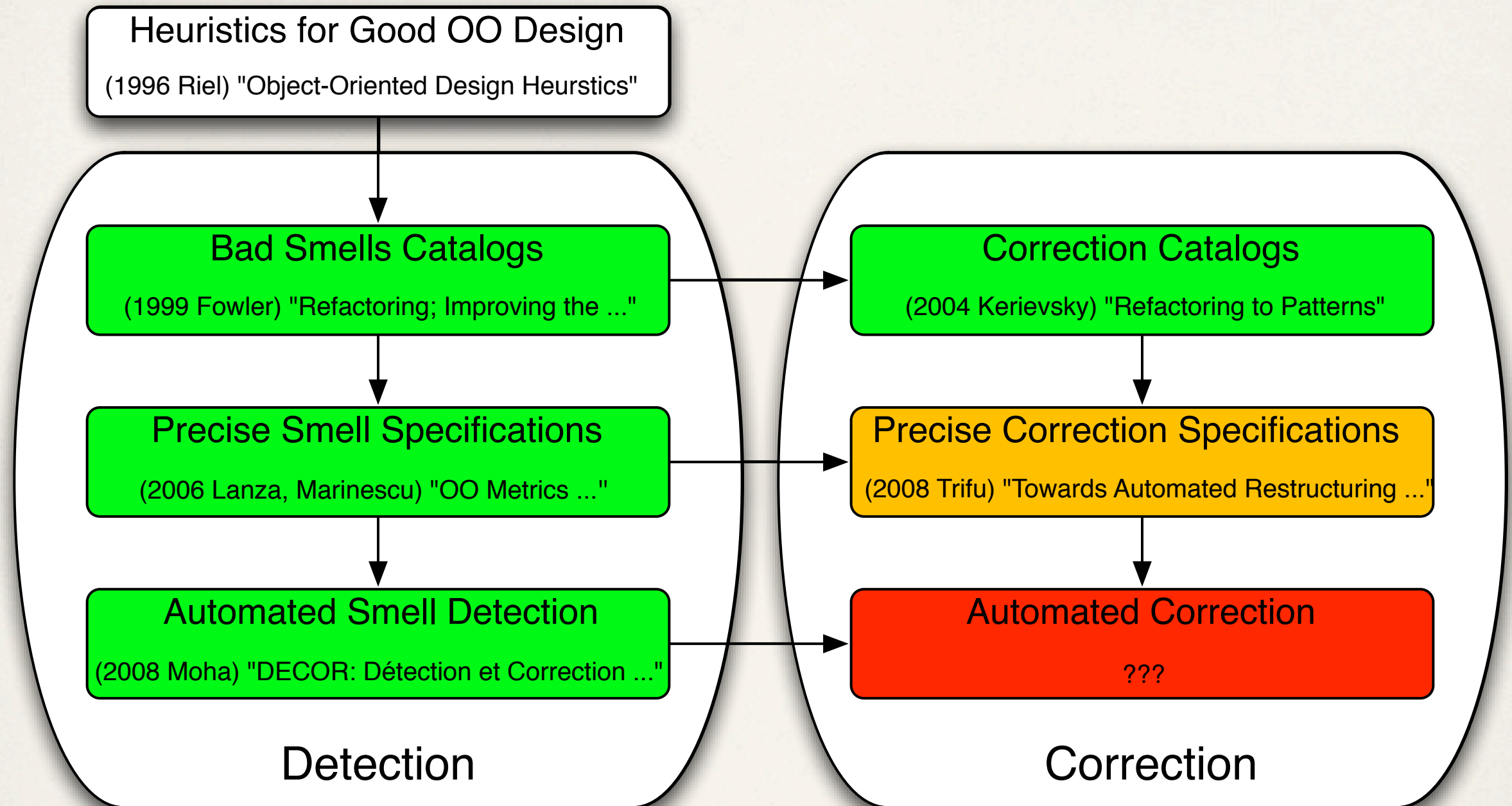
- ❖ Provide **support for computing Refactoring Plans**:
 - ♦ for enabling the precondition of a set of refactorings.
 - ♦ for applying design smell correction specifications.
- ❖ Provide a way to **help software developers use the techniques** elaborated in this PhD Thesis Dissertation.
- ❖ **Evaluate the effectiveness, efficiency and scalability** of the approach presented in this PhD Thesis Dissertation by developing a prototype.

Thesis Statement

The activity of refactoring, when complex refactoring sequences have to be applied as in the case of design smell correction in Object-Oriented software, can be assisted by means of refactoring plans that can be obtained automatically.

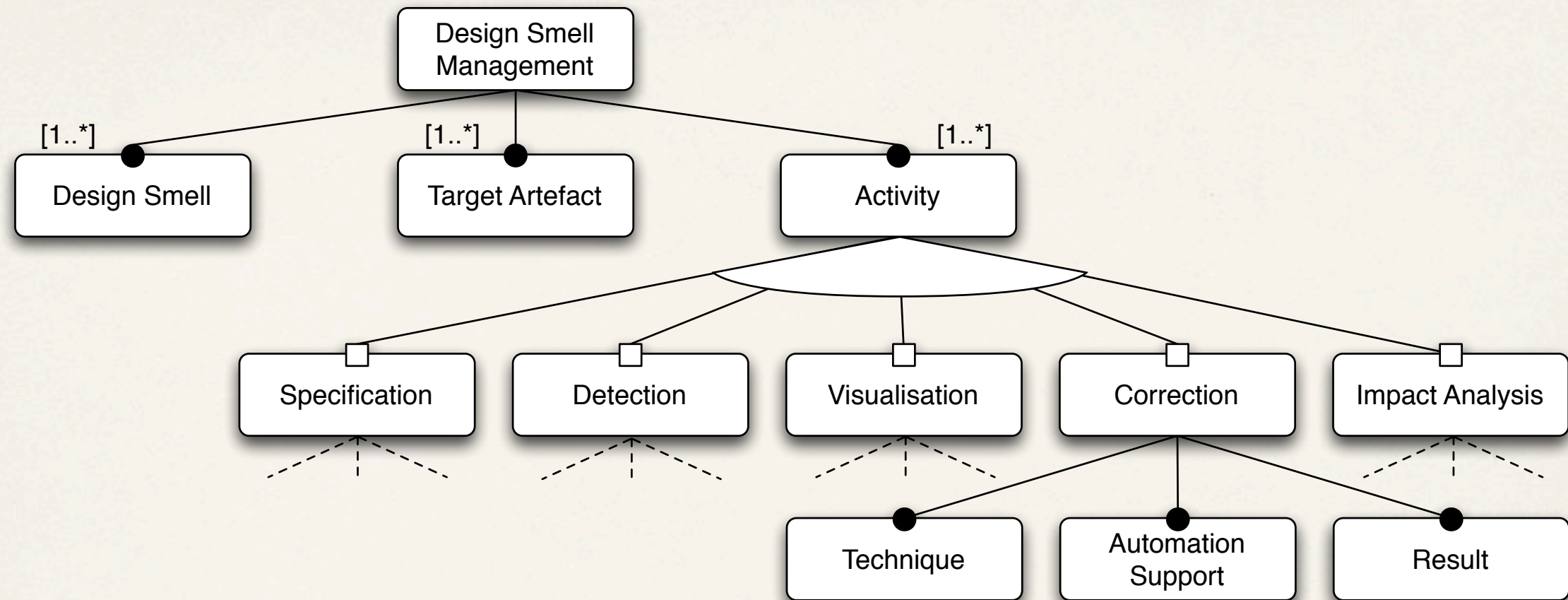
State of the Art

Brief History of Design Smell Management (DSM)



- ❖ More precise specifications lead to better automation
- ❖ Correction lacks formal specifications and automation support

Design Smell Management Survey and Taxonomy



- ❖ Comprehensive study with Tom Mens, Naouel Moha and Carlos López
- ❖ Design smell management taxonomy with feature modeling notation
- ❖ More than 100 references analysed
- ❖ 22 tool reviews available at <http://www.infor.uva.es/DesignSmells>
 - ❖ Analyst4j, ArgoUML, Argus CodeWatch, CodePro Analytix, Cultivate, Eclipse, FxCop, inCode, jDeodorant, JRefactory, M2 Resource StandarMetrics, PMD, Reek, RevJav, Roodi, SA4J, STAN, Structure101, StyleCop, Together, TRex, XRefactory

Analysis of 22 Design Smell Management Tools

Supported Activity

	Specification	Detection	Visualisation	Correction	Impact Analysis
#	9	22	7	9	2

Degree of Automation

	Manual	Interactive	Fully Automated
Detection	0	12	10
Correction	1	8	0

Type of Result

Correction	
Correction Suggestions	Refactorings / Actions
8	1

- ❖ Automated DSM is mature in detection, has to be improved in correction.
- ❖ Refactoring Suggestions for correction are not directly applicable.

Thesis Proposal

Proposal: Refactoring Strategies and Refactoring Plans

Refactoring Strategies

- ♦ heuristic-based specifications
- ♦ automation-suitable
- ♦ describe complex behaviour-preserving transformations
- ♦ aimed at a certain goal

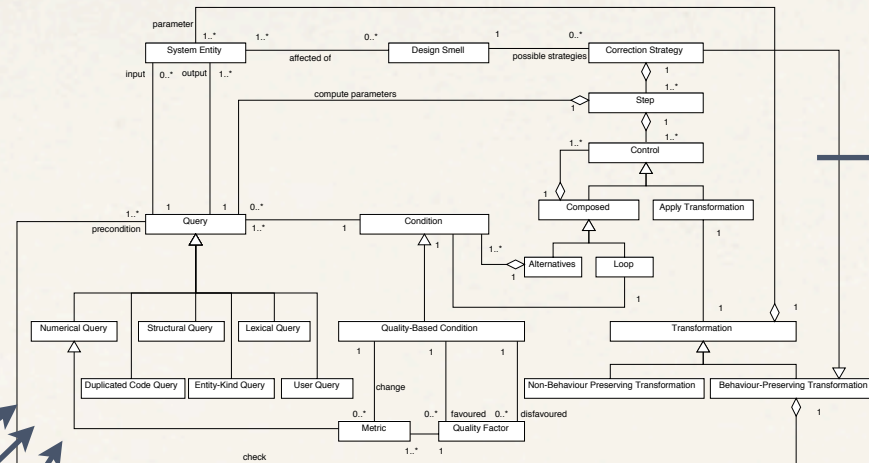
Refactoring Plans

- ♦ sequences of instantiated transformations
- ♦ achieve a certain goal
- ♦ can be applied over a system in its current state
- ♦ behaviour-preserving transformation sequence

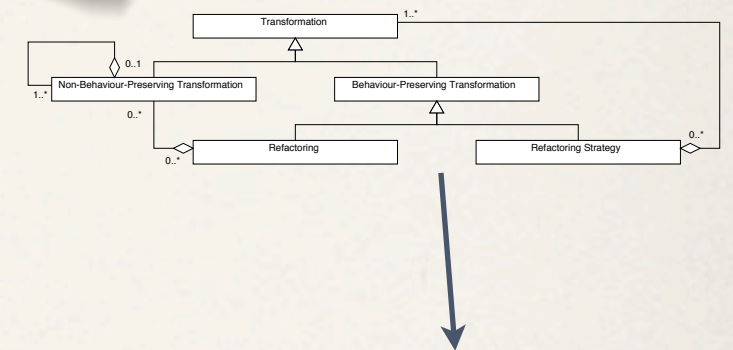
The Thesis in a Nutshell

Brown
Beck
Wake
Kerievsky
Marinescu
Demeyer
Trifu

Design Smell
Correction
Specifications



Refactoring
Strategies
Requirements



Refactoring
Strategies
Specification
Language

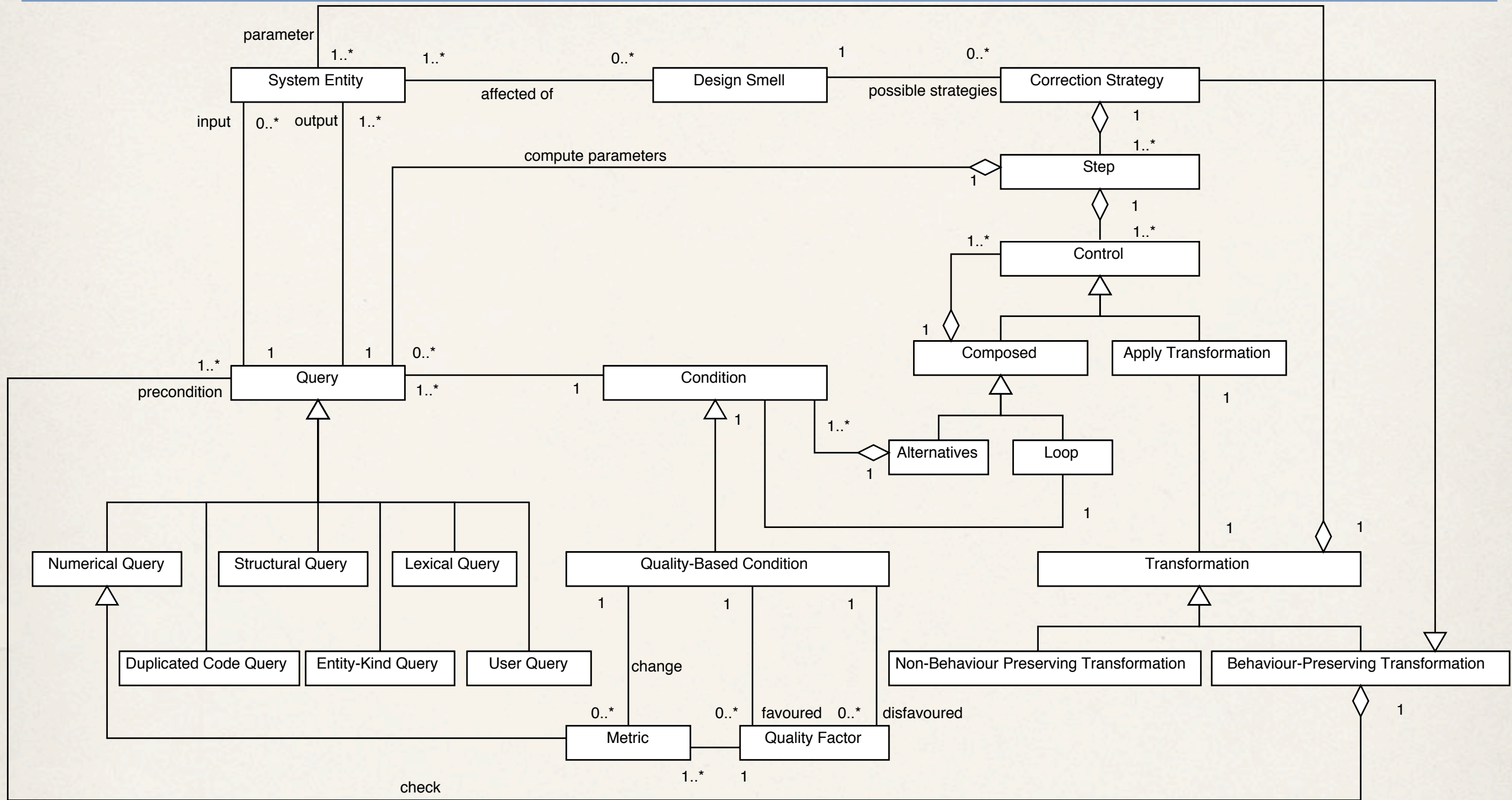
Refactoring Plan
Computation
Requirements

Evaluation
2 Case Studies

Prototype



Refactoring Strategies



[Brown et al., 1998], [Beck and Fowler, 1999], [Wake, 2003], [Kerievsky, 2004], [Lanza and Marinescu, 2006], [Demeyer et al., 2008], [Trifu, 2008]

Automation-Suitable Correction Specifications

❖ **Applicability of refactorings**

- ♦ representation of system entities at a low level of detail
- ♦ representation of refactoring preconditions
- ♦ support computing refactoring preconditions
- ♦ support computing refactoring effects

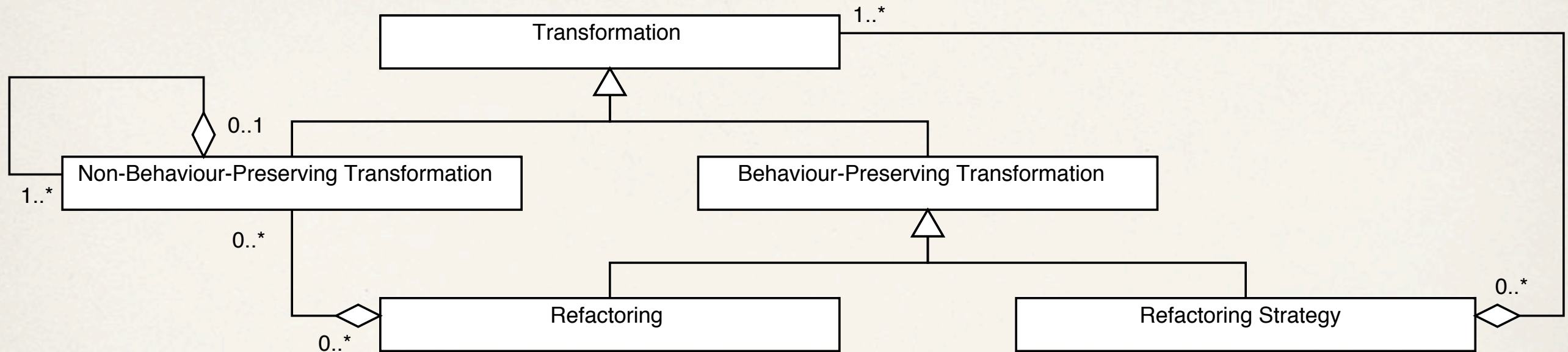
❖ **Heuristic descriptions of correction specifications**

- ♦ corrections knowledge is compiled in natural language, heuristics are hard to specify in an algorithmic language
- ♦ steps, loops and alternatives with unspecified order or without decision conditions
- ♦ incremental improvement of the available knowledge
- ♦ alternative correction strategies

❖ **Additional elements in correction strategies**

- ♦ invocation of strategies and substrategies
- ♦ calls for user interaction

Refactoring Strategies



- ❖ **NBP transformations**

- ♦ building blocks for other transformations

- ❖ **Refactorings**

- ♦ can include: simple queries, atomic AST changes, NBP transformations, deterministic control constructs

- ❖ **Refactoring strategies**

- ♦ can include: complex queries, atomic AST changes, any transformation, non-deterministic control constructs
 - ♦ aimed at a goal (apply big refactorings, remove smells)

Refactoring Strategy Specification Language

```

strategy remove-feature-envy move-method (method)
  body
    alt
      branch
        apply remove-feature-envy move-method-to-user-class (method)
      branch
        apply remove-feature-envy move-method-to-data-class (method)
      branch
        apply remove-feature-envy move-method-to-envied-class (method, env-class)
      end
    end
end

strategy remove-feature-envy move-method-to-envied-class (method, env-class)
  precondition
    get-envied-class (method, env-class)
    get-movemethod-reference (method, env-class, reference)
  body
    apply move-method all-sts (method, env-class, reference)
  end

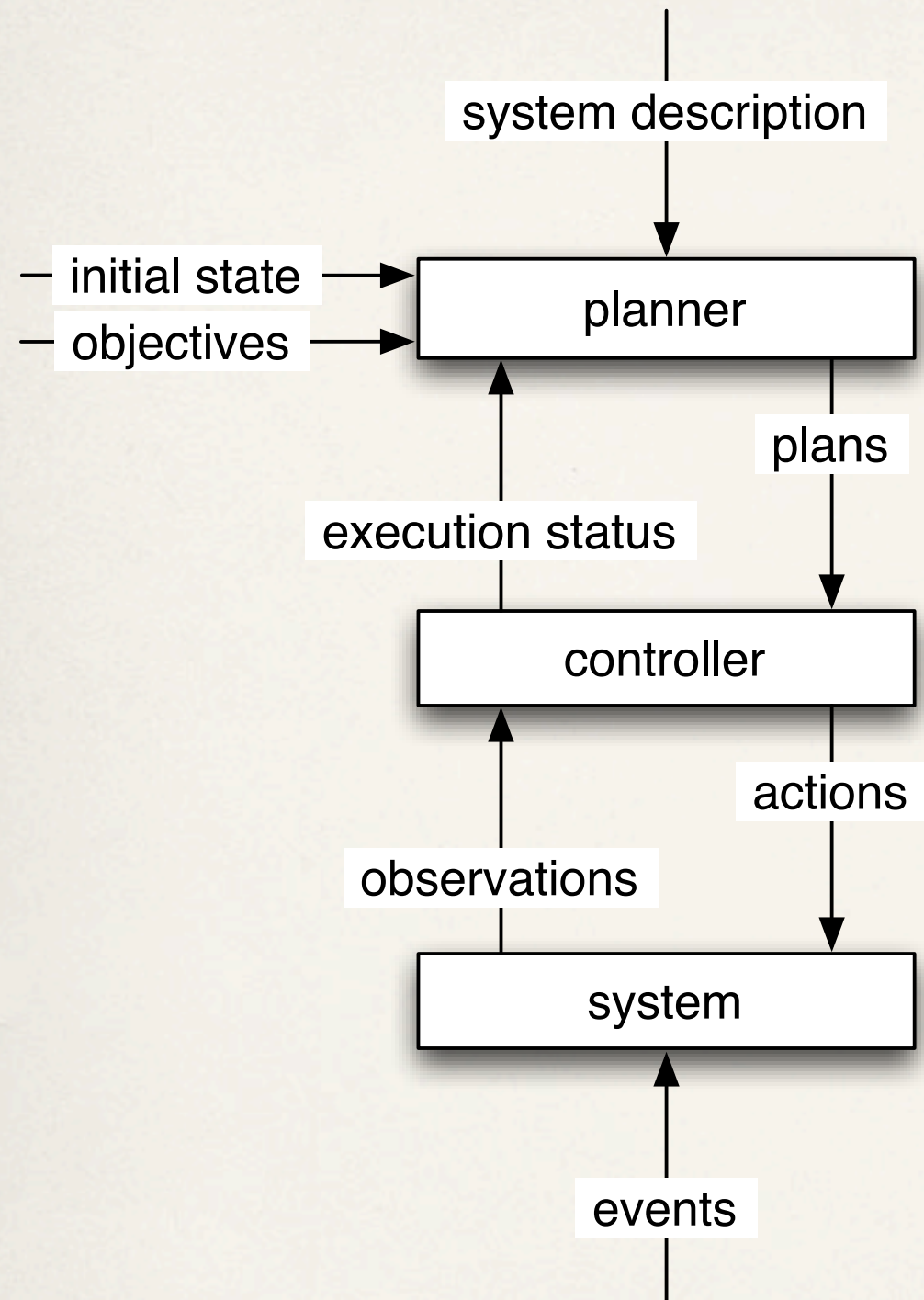
strategy move-method all-sts (method, tgt-class, reference)
  body
    alt
      branch apply move-method trivial (method, tgt-class, reference)
      branch apply move-method basic (method, tgt-class, reference)
    end
  end

```

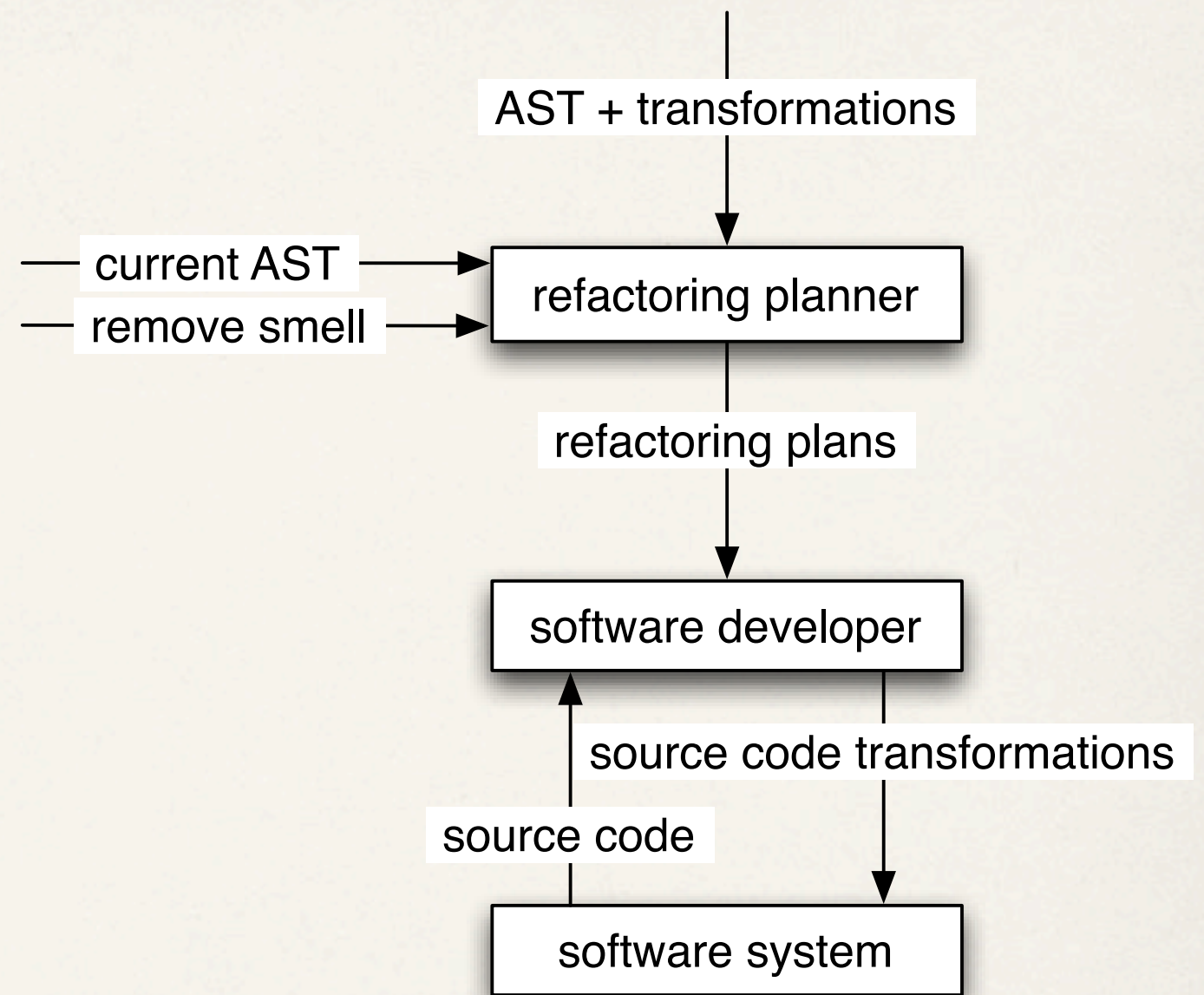
Refactoring Plans

- ❖ **Software Model**
 - ♦ low level of detail, representation of method's bodies
 - ♦ AST-based model
- ❖ **Computation of refactoring precondition and effects over the model**
- ❖ **Deterministic and non-deterministic control constructs**
- ❖ **Incomplete specifications**
 - ♦ Computation with the available knowledge
 - ♦ Incrementally improvement of the available knowledge
- ❖ **Support to represent and manage all elements of refactoring strategies**

Refactoring Planning as an Automated Planning Problem



Automated Planning

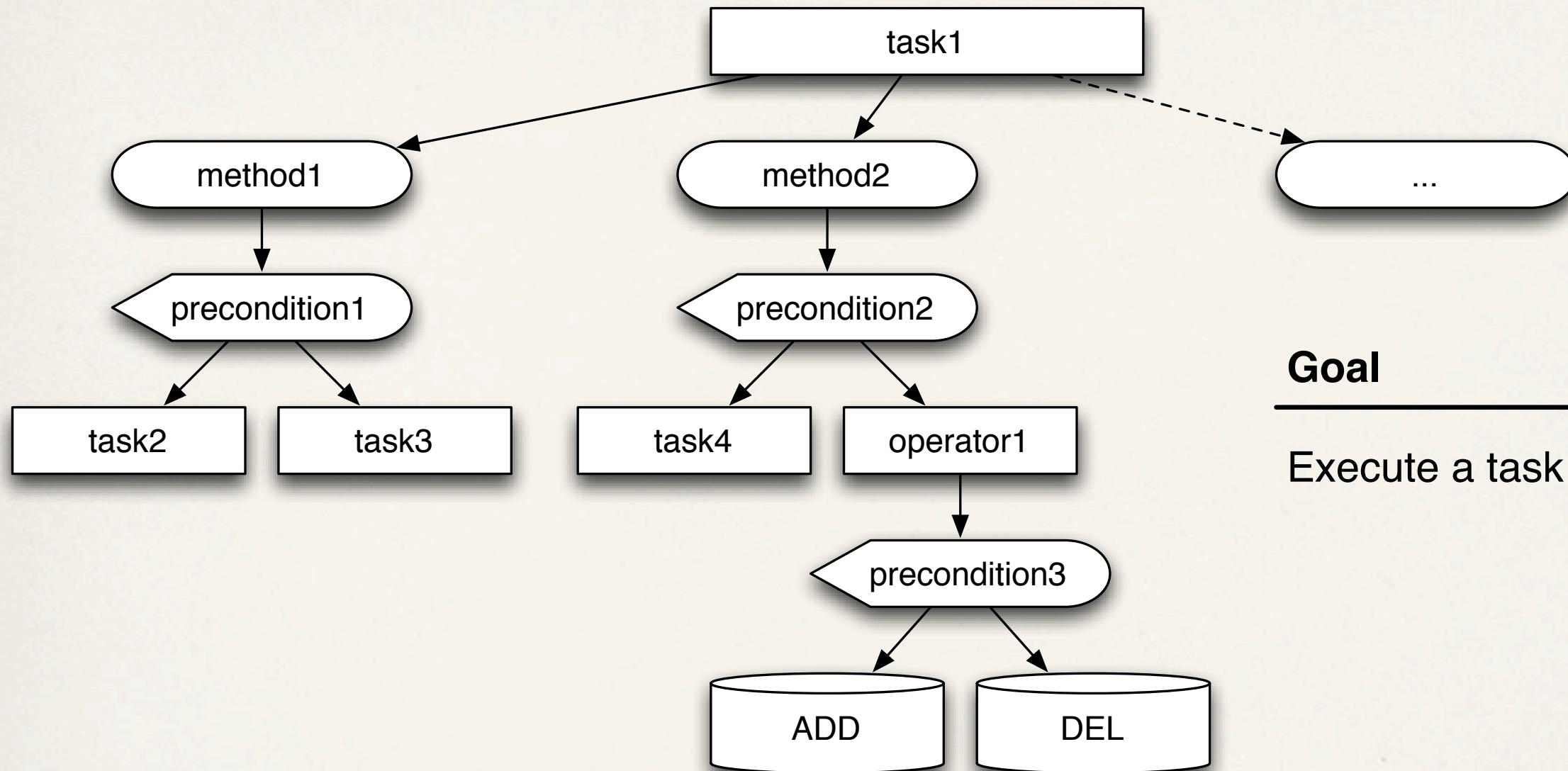


Refactoring Planning

Choosing the Right Planner

- ❖ **Many kinds of planning approaches**
 - ♦ select the most appropriate given the problem characteristics
- ❖ **Classical Planning** (8 assumptions - restricted model)
- ❖ **Neoclassical planning** (relaxed assumptions)
 - domain-configurable planners
 - Hierarchical Task Network (HTN) Planning
- ❖ **Refactoring Planning - characteristics of the problem**
 - ♦ huge states and search-spaces
 - ♦ heuristic knowledge available as “recipes”
 - ♦ some unmet assumptions

Hierarchical Task Network (HTN) Planning



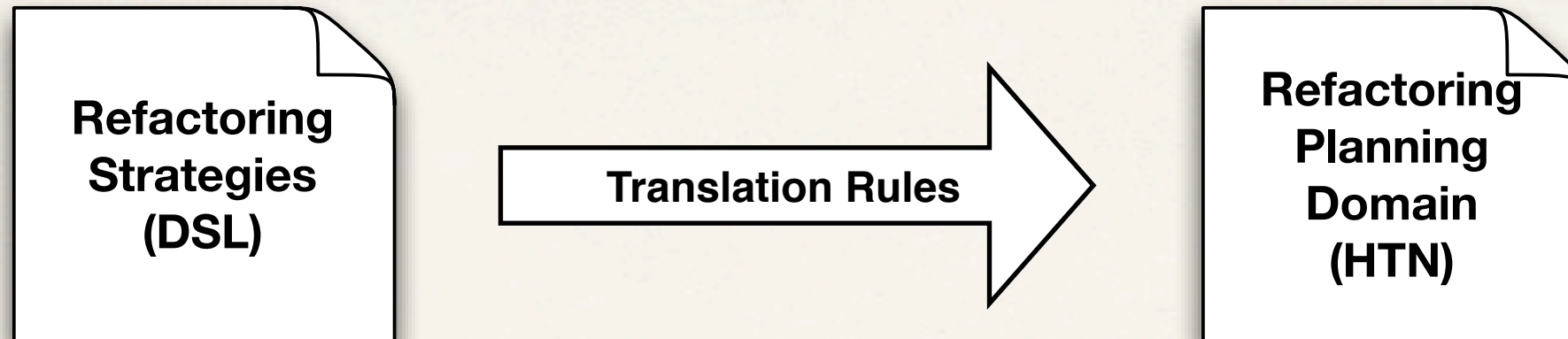
- ❖ **Tasks:** Methods and operators - “recipes” about how to execute a task
- ❖ **Preconditions:** first-order-logic queries - gather system information
- ❖ **Goal:** Execute a task

Refactoring Planning as an HTN Planning Problem

World's state:	AST represented by first-order logic predicates
Operators:	atomic changes to the AST (mainly add, delete and replace)
Tasks:	transformation parts refactoring parts non-behaviour preserving transformations refactorings refactoring strategies
Goal:	Executing a smell correction strategy Executing a refactoring application strategy
Planning problem:	Execute a particular refactoring strategy over a particular version of a system

- ❖ **JSHOP2:** the HTN planner selected
 - ♦ forward search in the same order as the plan should be executed
 - ♦ very expressive and efficient
 - ♦ first-order-logic inference engine
 - ♦ external queries
 - ♦ two stages planning process: precompilation + planning

Implementation of Refactoring Strategies: alternatives



Deterministic alternative

```
if COND1 then
  STEPS1
elseif COND2 then
  STEPS2
else
  STEPS3
end
```



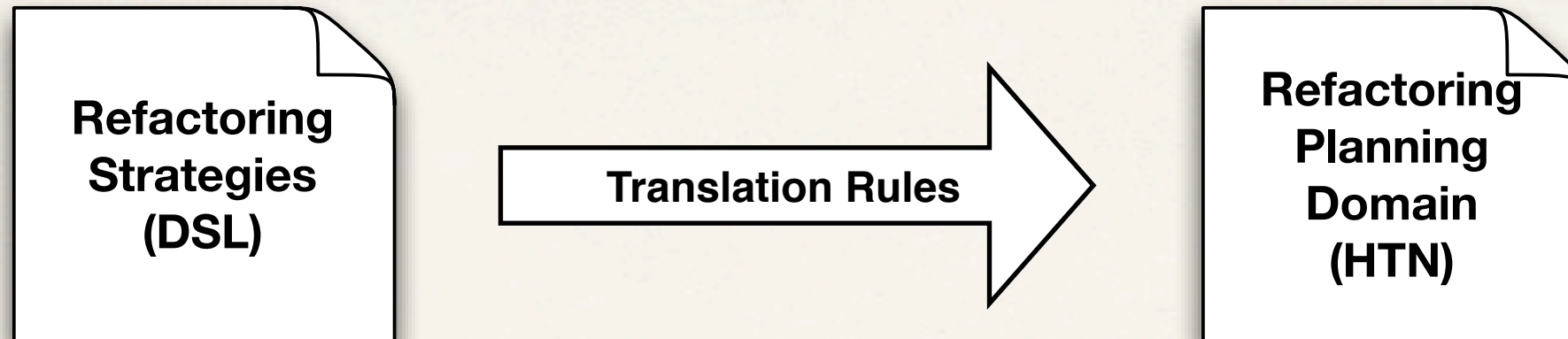
```
:: if invocation
(if-xx ARGS)

:: if definition
(:method (if-xx ARGS)
  (COND1)
  (STEPS1)

  (COND2)
  (STEPS2)

  ()
  (STEPS3)
)
```


Implementation of Refactoring Strategies: alternatives



Non-Deterministic alternative

```
alt
  branch COND1
    body
      STEPS1
  branch COND2
    body
      STEPS2
  branch
    body
      STEPS3
end
```



```
:: alt invocation
(alt-xx ARGS)

;; alt definition
(:method (alt-xx ARGS)
  (COND1)
  (STEPS1)
)
(:method (alt-xx ARGS)
  (COND2)
  (STEPS2)
)
(:method (alt-xx ARGS)
  ()
  (STEPS3)
)
```

Prototype

Refactoring Planning Domain

- ❖ **3 Refactoring Strategies:**

- ♦ for removing design smells
 - Remove Data Class and Remove Feature Envy
- ♦ for applying a complex refactoring
 - Move Method

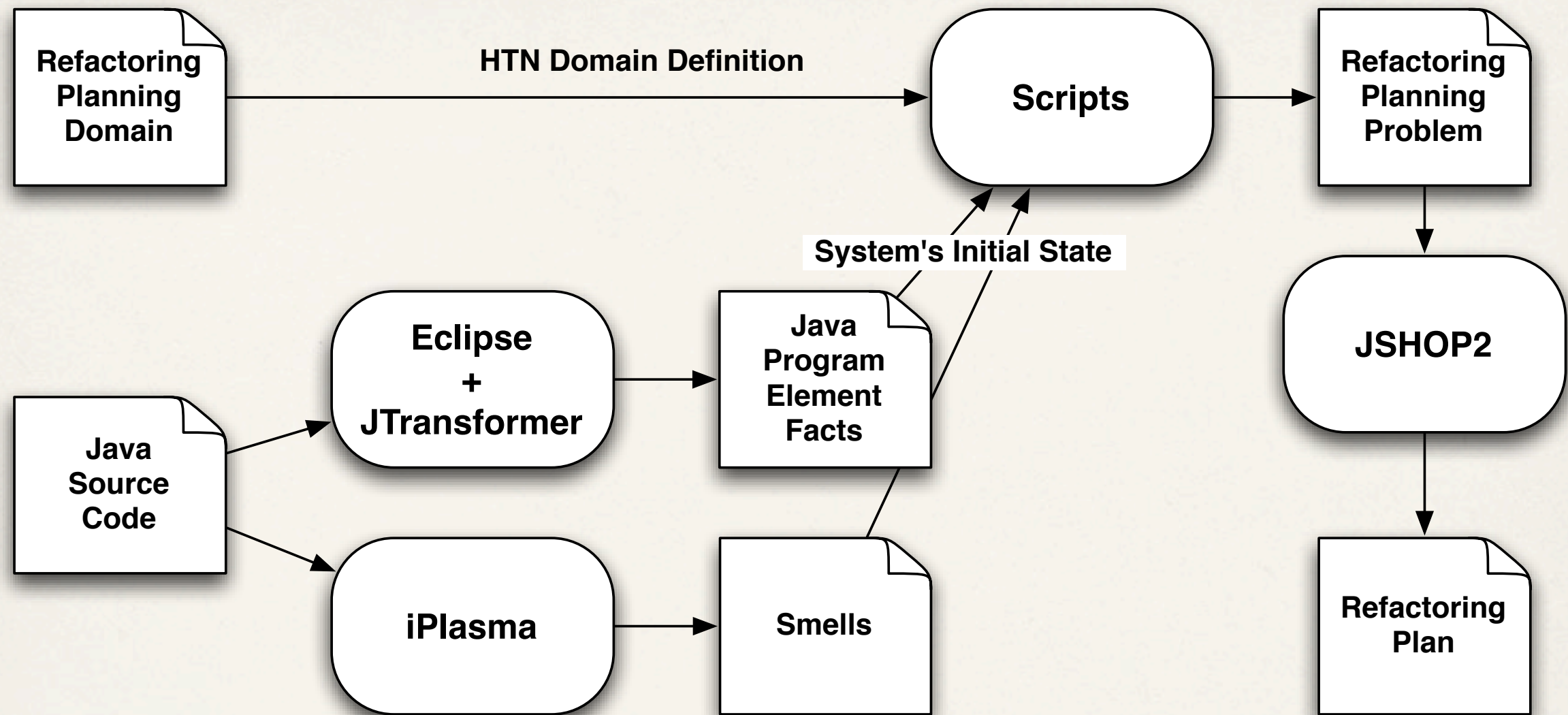
- ❖ **9 Refactorings:**

- ♦ Encapsulate Field, Move Method, Rename Method, Rename Field, Rename Parameter, Rename Local Variable, Remove Field, Remove Method and Remove Class.

- ❖ **> 150 system queries:**

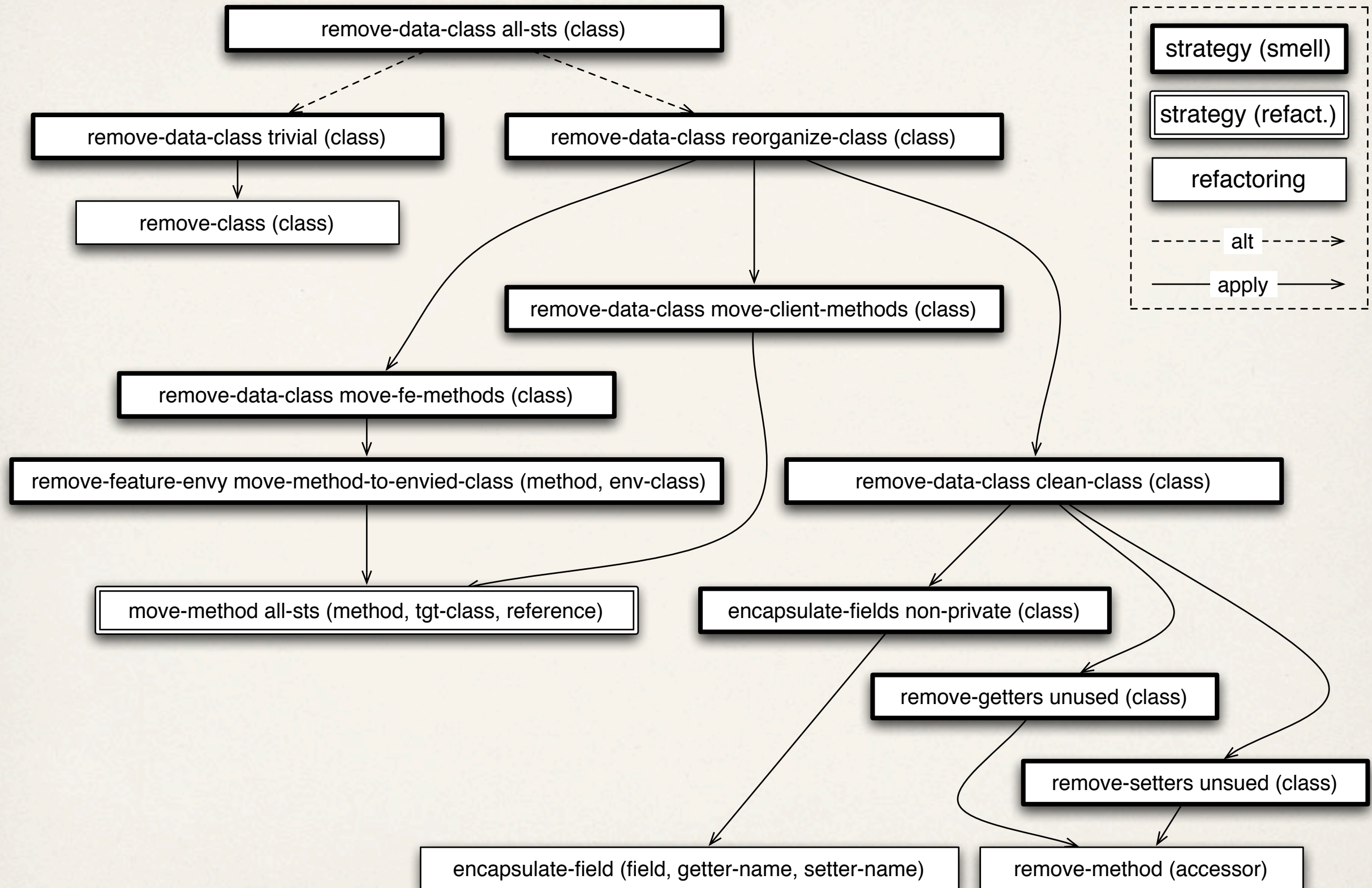
- ♦ structural, lexical, numerical (metrics), user queries
- ♦ 8 external Java procedures

Prototype Overview



- ❖ **Refactoring Planning Domain:** strategies + refactorings + NBPT + queries
- ❖ **JTransformer:** to obtain a predicate-based representation of the system
- ❖ **iPlasma:** to produce smell-entity reports
- ❖ **JSHOP2:** to compute refactoring plans from strategies for a particular case

Remove Data Class Strategy



Remove Data Class Strategy: example of a plan

```

apply-refactoring:      show-method (org.jwebap.asm.attrs, stackmapattribute,
                                gettypeinfolabels)
apply-refactoring:      move-method (from, org.jwebap.asm.attrs, stackmapattribute,
                                getframelabels, to, org.jwebap.asm.attrs, stackmapframe,
                                through, frame, keeping-delegate, false)
apply-refactoring:      show-method (org.jwebap.asm.attrs, stackmapattribute, writetypeinfo)
apply-refactoring:      move-method (from, org.jwebap.asm.attrs, stackmapattribute,
                                writeframe, to, org.jwebap.asm.attrs, stackmapframe,
                                through, frame, keeping-delegate, false)
apply-refactoring:      show-method (org.jwebap.asm.util.attrs, asmstackmapattribute, asmify)
apply-refactoring:      show-method (org.jwebap.asm.util.attrs, asmstackmapattribute,
                                declarelabel)
apply-refactoring:      show-method (org.jwebap.asm.util.attrs, asmstackmapattribute,
                                asmifytypeinfo)
apply-refactoring:      move-method (from, org.jwebap.asm.util.attrs, asmstackmapattribute,
                                asmify, to, org.jwebap.asm.attrs, stackmapframe,
                                through, f keeping-delegate, false)

ply-refactoring:        encapsulate-field (org.jwebap.asm.attrs, stackmapframe, label,
                                getlabel, setlabel)
apply-refactoring:      encapsulate-field (org.jwebap.asm.attrs, stackmapframe, locals,
                                getlocals, setlocals)
apply-refactoring:      encapsulate-field (org.jwebap.asm.attrs, stackmapframe, stack,
                                getstack, setstack)
apply-refactoring:      remove-method(org.jwebap.asm.attrs, stackmapframe, setlabel) (SETTER)
apply-refactoring:      remove-method(org.jwebap.asm.attrs, stackmapframe, setlocals)(SETTER)
apply-refactoring:      remove-method(org.jwebap.asm.attrs, stackmapframe, setstack) (SETTER)

apply-remove-smell:     remove-data-class (org.jwebap.asm.attrs, stackmapframe) cleanclass
  
```


Case Studies

Experiment Description

Experiment Goal (GQM template)

object of study	refactoring planning approach
purpose	characterisation and evaluation
focus	effectiveness, efficiency and scalability
stakeholders	researcher
context	our reference prototype

Experiment Procedure:

- ❖ execution of the refactoring planner
- ❖ obtaining refactoring plans from refactoring strategies
- ❖ two case studies: removing Feature Envy and Data Class
- ❖ for a set of 9 open-source systems.

Samples and Variables

	System	Version	LOC	PEF	FE	DC
1	Jtombstone	1.1.1	1938	32780	2	7
2	Groom	1.3	3699	35434	2	2
3	Lucene	1.9	17627	85064	18	16
4	Pounder	0.96	9410	98570	26	3
5	MyTelly	1.2	12625	133605	2	13
6	Jwebap	0.6.1	16417	141047	17	21
7	dbXML	2.0	25862	199400	21	40
8	GanttProject	2.0.10	40775	241095	36	24
9	JfreeChart	1.0.11	80668	354543	45	29

❖ Independent Variables:

- ❖ *PEF*: number of program element facts - system size (obtained with JTransformer)
- ❖ *FE*: number of *Feature Envy* design smells (according to iPlasma)
- ❖ *DC*: number of Data Class design smells (according to iPlasma)

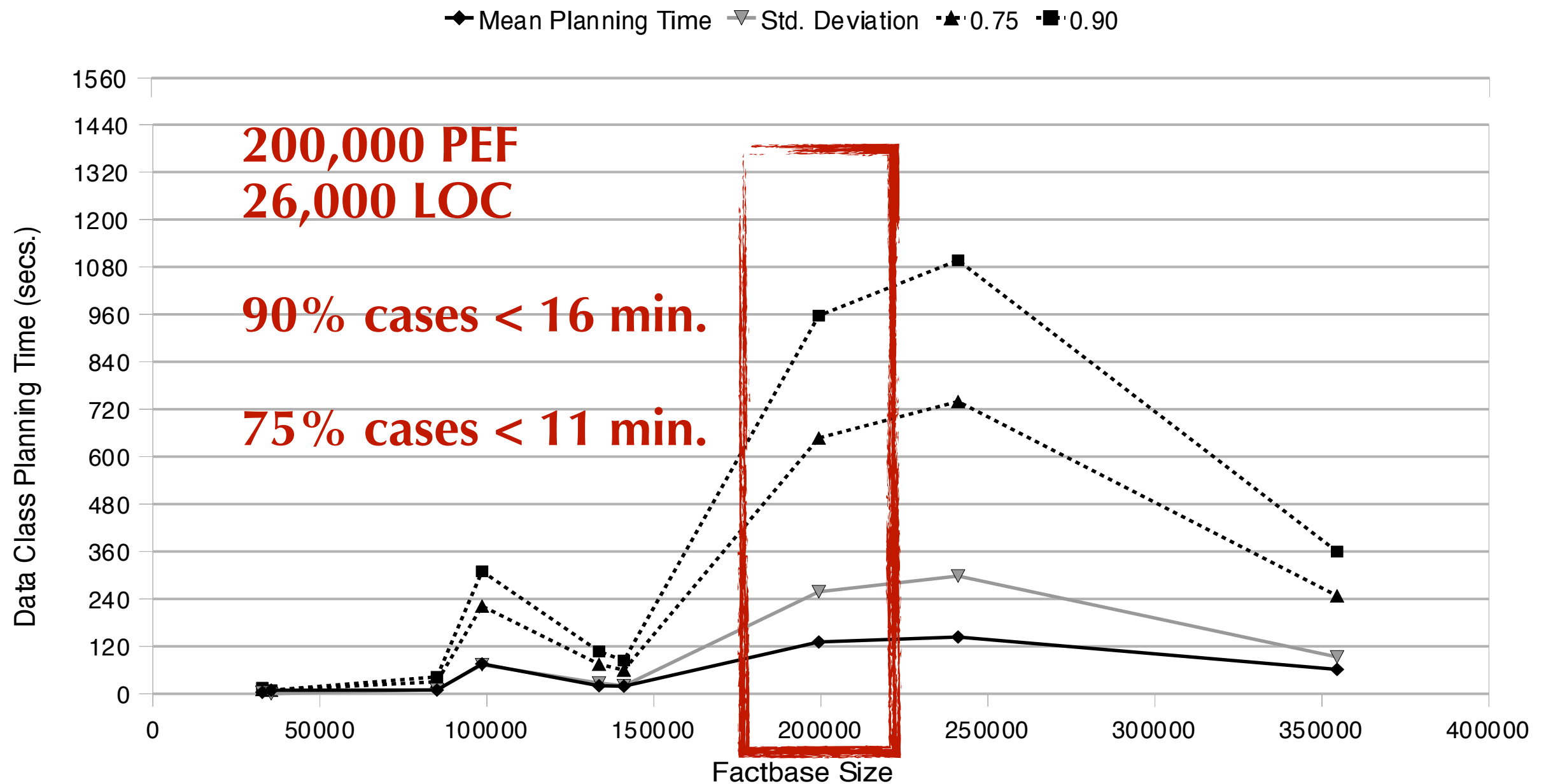
❖ Dependant Variables:

- ❖ *P*: Number of plans obtained for each system
- ❖ T_t : Total elapsed time (seconds) $\rightarrow T_t = T_c + T_p$
- ❖ T_c : Precompilation time (seconds)
- ❖ T_p : Planning time (seconds)

Summary of Results

Feature Envy					Time is given in seconds.		
	PEFs	Smells	Plans	%	Mean T_t	Mean T_c	Mean T_p
1	32780	2	1	50	29.52	23.02	6.49
2	35434	2	1	50	30.55	21.57	8.98
3	85064	18	3	16.67	64.1	53.71	10.4
4	98570	26	21	80.77	107.7	103.66	4.04
5	133605	2	0	0	253.69	182.85	70.84
6	141047	17	9	52.94	182.03	148.05	33.98
7	199400	21	11	52.38	413.21	303.29	109.92
8	241095	36	13	36.11	544.79	385.46	159.32
9	354543	45	24	53.33	1065.54	960.83	104.71
Totals							
Data Class							
1	32780	7	6	85.71	26.28	22.5	3.78
2	35434	2	2	100	30.56	21.63	8.93
3	85064	16	16	100	62.96	53.48	9.48
4	98570	3	3	100	179.91	104.28	75.63
5	133605	13	11	84.62	206.36	186.15	20.21
6	141047	21	20	95.24	167.55	148.36	19.2
7	199400	40	40	100	431.47	300.36	131.11
8	241095	27	24	88.89	530.18	386.57	143.62
9	354543	29	23	79.31	1021.41	959.76	61.65
Totals							

Probabilistic Upper Bounds for Planning Time



- ◆ Upper bounds for T_p computed with Chebishev's Inequality for 75% and 90% probability

Experiment Conclusions

- ❖ **Effectiveness:**

- ♦ 50% plans for Feature Envy and 92% for Data Class

- ❖ **Efficiency and scalability (analysis):**

- ♦ Precompilation time is tightly correlated to system size.
- ♦ Planning time is very disperse.
- ♦ Planning time does not follow a normal distribution.
- ♦ Planning time depends on system size.
- ♦ Results for the dependency between planning time and strategies are not conclusive.
- ♦ Probabilistic upper bounds of planning time are satisfactory.

- ❖ **Efficiency and scalability (conclusions):**

- ♦ Planning time is reasonable for a prototype.
- ♦ Precompilation time should be avoided.
- ♦ Scalability is hard to infer, good for the tested sizes, promising results.

Conclusions

Contributions

✧ Regarding Refactoring Strategies

- ✧ A review on the design smells' literature
- ✧ A survey on design smell management and a taxonomy
- ✧ Definition of refactoring strategies
- ✧ Definition of a refactoring strategy specification language

Contributions (2)

❖ **Regarding Refactoring Plans**

- ❖ Definition of refactoring plans
- ❖ Definition of the requirements to compute refactoring plans
- ❖ A technique to instantiate refactoring strategies into refactoring plans by means of automated planning.
- ❖ An initial refactoring planning domain, and a reference prototype for future research

Future Work

- ✧ Improve the refactoring planning domain
- ✧ Improve the prototype
- ✧ Explore different application scenarios for the approach

Refactoring Planning for Design Smell Correction in Object-Oriented Software

PhD Thesis

Francisco Javier Pérez García

supervised by
Yania Crespo González-Carvajal

Universidad de Valladolid - ETSII - Departamento de Informática - grupo GIRO

July 6th, 2011 - Valladolid