Overview of the Refactoring Discovering Problem

Javier Pérez
jperez@infor.uva.es

Universidad de Valladolid

ECOOP 2006, Doctoral Symposium and PhD Students Workshop

Introduction

Refactorings are structural transformations that can be applied to a software system to perform design changes without modifying its behaviour.

Current approaches to improve a system design with refactorings focus in:

- Individual refactoring steps.
- Detecting refactoring opportunities.
- Executing the refactoring with a tool.

Some techniques can suggest **wider design changes**:

- Formal Concept Analysis to propose hierarchy reorganization (Prieto et al., 2003).
- Metrics to detect Bad Smells (Crespo et al., 2005).

They require **sequences of refactorings** to perform the proposed change.

Introduction

Refactorings are structural transformations that can be applied to a software system to perform design changes without modifying its behaviour.

Current approaches to improve a system design with refactorings focus in:

- Individual refactoring steps.
- Detecting refactoring opportunities.
- Executing the refactoring with a tool.

Some techniques can suggest **wider design changes**:

- Formal Concept Analysis to propose hierarchy reorganization (Prieto et al., 2003).
- Metrics to detect Bad Smells (Crespo et al., 2005).

They require **sequences of refactorings** to perform the proposed change.

Introduction

Refactorings are structural transformations that can be applied to a software system to perform design changes without modifying its behaviour.

Current approaches to improve a system design with refactorings focus in:

- Individual refactoring steps.
- Detecting refactoring opportunities.
- Executing the refactoring with a tool.

Some techniques can suggest **wider design changes**:

- Formal Concept Analysis to propose hierarchy reorganization (Prieto et al., 2003).
- Metrics to detect Bad Smells (Crespo et al., 2005).

They require **sequences of refactorings** to perform the proposed change.

Introduction

Refactorings are structural transformations that can be applied to a software system to perform design changes without modifying its behaviour.

Current approaches to improve a system design with refactorings focus in:

- Individual refactoring steps.
- Detecting refactoring opportunities.
- Executing the refactoring with a tool.

Some techniques can suggest **wider design changes**:

- Formal Concept Analysis to propose hierarchy reorganization (Prieto et al., 2003).
- Metrics to detect Bad Smells (Crespo et al., 2005).

They require **sequences of refactorings** to perform the proposed change.

Main Goals

1. **To automatically and dinamically generate refactoring sequences (refactoring plans)** that can transform a system, following a redesing proposal, and preserving the system's behaviour.
2. **To provide very high level (big) refactorings** for design improvement, using refactoring plan generation altogether with analysis techniques that suggest redesign proposals.
3. **To support this "refactoring plan" technique with tool prototypes** to provide it as a regular design improving development technique.

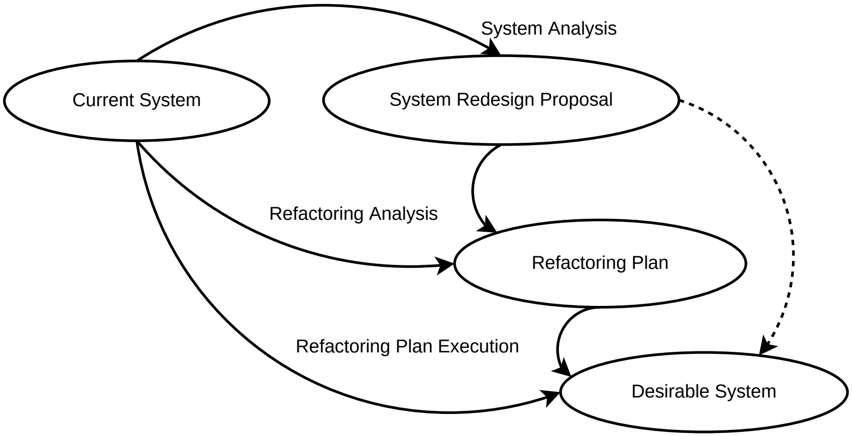
Main Goals

1. **To automatically and dinamically generate refactoring sequences (refactoring plans)** that can transform a system, following a redesing proposal, and preserving the system's behaviour.
2. **To provide very high level (big) refactorings** for design improvement, using refactoring plan generation altogether with analysis techniques that suggest redesign proposals.
3. **To support this "refactoring plan" technique with tool prototypes** to provide it as a regular design improving development technique.

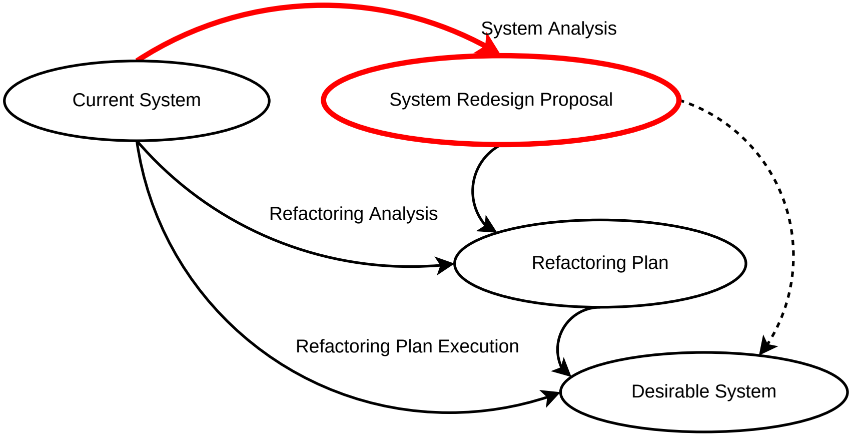
Main Goals

1. **To automatically and dinamically generate refactoring sequences (refactoring plans)** that can transform a system, following a redesing proposal, and preserving the system's behaviour.
2. **To provide very high level (big) refactorings** for design improvement, using refactoring plan generation altogether with analysis techniques that suggest redesign proposals.
3. **To support this "refactoring plan" technique with tool prototypes** to provide it as a regular design improving development technique.

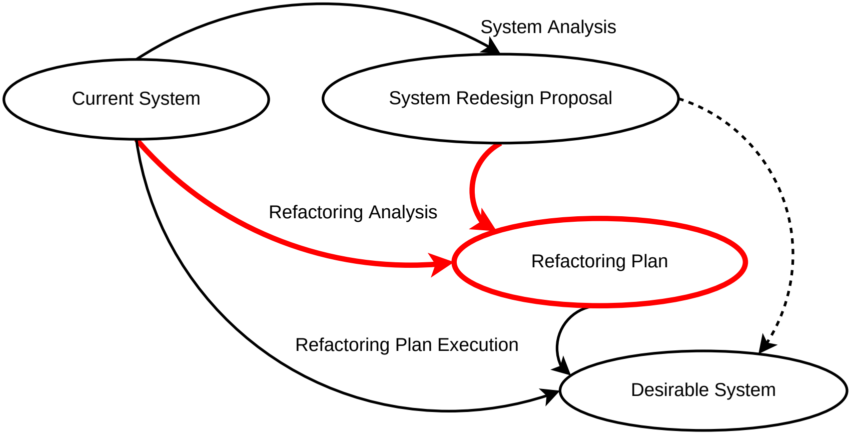
Refactoring Sequences Problem Overview



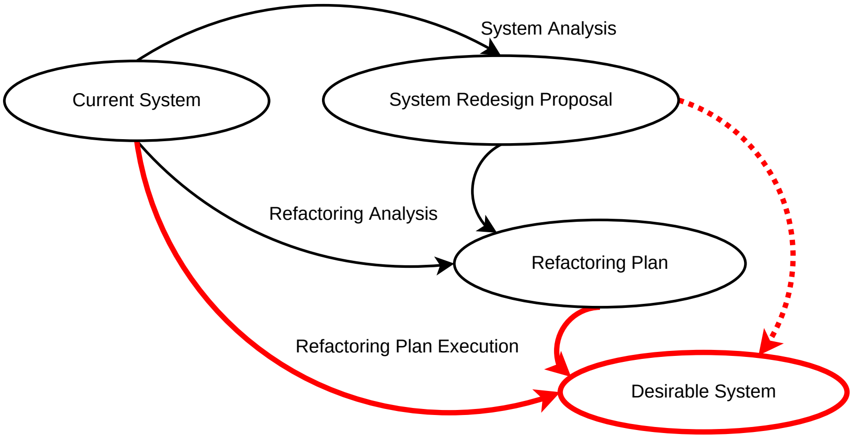
Refactoring Sequences Problem Overview



Refactoring Sequences Problem Overview



Refactoring Sequences Problem Overview



Refactoring Plans

We pretend to introduce a new concept: **Refactoring Plans**

Definition

A **Refactoring Plan** will be a specification of a refactoring sequence which matches a system redesign proposal, so that it can be automatically executed to modify the system in order to obtain that desirable system redesign without changing the system's behaviour.

Refactoring Plans

We pretend to introduce a new concept: **Refactoring Plans**

Definition

A **Refactoring Plan** will be a specification of a refactoring sequence which matches a system redesign proposal, so that it can be automatically executed to modify the system in order to obtain that desirable system redesign without changing the system's behaviour.

Refactoring Plans

We pretend to introduce a new concept: **Refactoring Plans**

Definition

A **Refactoring Plan** will be a specification of a refactoring sequence which matches a system redesign proposal, so that it can be automatically executed to modify the system in order to obtain that desirable system redesign without changing the system's behaviour.

Refactoring Plans

We pretend to introduce a new concept: **Refactoring Plans**

Definition

A **Refactoring Plan** will be a specification of a refactoring sequence which matches a system redesign proposal, so that it can be automatically executed to modify the system in order to obtain that desirable system redesign without changing the system's behaviour.

Refactoring Plans

We pretend to introduce a new concept: **Refactoring Plans**

Definition

A **Refactoring Plan** will be a specification of a refactoring sequence which matches a system redesign proposal, so that it can be automatically executed to modify the system in order to obtain that desirable system redesign without changing the system's behaviour.

Refactoring Plans

We pretend to introduce a new concept: **Refactoring Plans**

Definition

A **Refactoring Plan** will be a specification of a refactoring sequence which matches a system redesign proposal, so that it can be automatically executed to modify the system in order to obtain that desirable system redesign without changing the system's behaviour.

Refactoring Plans

We pretend to introduce a new concept: **Refactoring Plans**

Definition

A **Refactoring Plan** will be a specification of a refactoring sequence which matches a system redesign proposal, so that it can be automatically executed to modify the system in order to obtain that desirable system redesign without changing the system's behaviour.

Refactoring Plan Questions

Given **a software system** as the source of the transformation, **a redesign proposal**, and **a set of refactorings** that can be used as transformation operations:

1. Does a refactoring plan, which transforms the source, according to the redesign proposal, using the provided refactorings, exist?
2. When a refactoring plan exists, can it be generated and executed automatically?

Refactoring Plan Questions

Given **a software system** as the source of the transformation, **a redesign proposal**, and **a set of refactorings** that can be used as transformation operations:

1. Does a refactoring plan, which transforms the source, according to the redesign proposal, using the provided refactorings, exist?
2. When a refactoring plan exists, can it be generated and executed automatically?

Refactoring Plan Questions

Given **a software system** as the source of the transformation, **a redesign proposal**, and **a set of refactorings** that can be used as transformation operations:

1. Does a refactoring plan, which transforms the source, according to the redesign proposal, using the provided refactorings, exist?
2. When a refactoring plan exists, can it be generated and executed automatically?

Subproblems

We have divided the problem of **automatic generation of refactoring plans** in:

- Definition and formalization of the “Refactoring Plan” concept
- Representation of Software
- Formalization of Refactorings
- Elaboration of a “refactoring sequences” discovering algorithm
- Validation of proposed subproblem solutions through existing tools and prototype implementation

Research Strategy

To reduce the problem complexity given by:

- differences between source and redesign proposal representation type, abstraction level, description language, . . .
- uncertainty about what kind of redesign proposal descriptions will we be capable to deal with.

We have planned the **research** to progress **through two stages**:

- First, we will apply some restrictions to the problem and propose a restricted solution.
- Then, we will “open up” those restrictions to find a more general solution.

Research Strategy

To reduce the problem complexity given by:

- differences between source and redesign proposal representation type, abstraction level, description language, . . .
- uncertainty about what kind of redesign proposal descriptions will we be capable to deal with.

We have planned the **research** to progress **through two stages**:

- First, we will apply some restrictions to the problem and propose a restricted solution.
- Then, we will "open up" those restrictions to find a more general solution.

Research Strategy

To reduce the problem complexity given by:

- differences between source and redesign proposal representation type, abstraction level, description language, . . .
- uncertainty about what kind of redesign proposal descriptions will we be capable to deal with.

We have planned the **research** to progress **through two stages**:

- First, we will apply some restrictions to the problem and propose a restricted solution.
- Then, we will “open up” those restrictions to find a more general solution.

Research Strategy

To reduce the problem complexity given by:

- differences between source and redesign proposal representation type, abstraction level, description language, . . .
- uncertainty about what kind of redesign proposal descriptions will we be capable to deal with.

We have planned the **research** to progress **through two stages**:

- First, we will apply some restrictions to the problem and propose a restricted solution.
- Then, we will “open up” those restrictions to find a more general solution.

Research Strategy

To reduce the problem complexity given by:

- differences between source and redesign proposal representation type, abstraction level, description language, . . .
- uncertainty about what kind of redesign proposal descriptions will we be capable to deal with.

We have planned the **research** to progress **through two stages**:

- First, we will apply some restrictions to the problem and propose a restricted solution.
- Then, we will “open up” those restrictions to find a more general solution.

Research Strategy

To reduce the problem complexity given by:

- differences between source and redesign proposal representation type, abstraction level, description language, . . .
- uncertainty about what kind of redesign proposal descriptions will we be capable to deal with.

We have planned the **research** to progress **through two stages**:

- First, we will apply some restrictions to the problem and propose a restricted solution.
- Then, we will “open up” those restrictions to find a more general solution.

Research Strategy: First Stage

We apply some **restrictions to reduce complexity**:

- The transformation source is the current system code.
- The redesign proposal is the modified system code.

Goals of this stage are:

- To propose a solution to the restricted refactoring plan problem.
- To validate this solution.

Results obtained so far **can be applied** to other problems:

- testing if an evolved system preserves the behaviour of the original system.
- documenting refactoring changes performed in a redesigned system.

Research Strategy: First Stage

We apply some **restrictions to reduce complexity**:

- The transformation source is the current system code.
- The redesign proposal is the modified system code.

Goals of this stage are:

- To propose a solution to the restricted refactoring plan problem.
- To validate this solution.

Results obtained so far **can be applied** to other problems:

- testing if an evolved system preserves the behaviour of the original system.
- documenting refactoring changes performed in a redesigned system.

Research Strategy: First Stage

We apply some **restrictions to reduce complexity**:

- The transformation source is the current system code.
- The redesign proposal is the modified system code.

Goals of this stage are:

- To propose a solution to the restricted refactoring plan problem.
- To validate this solution.

Results obtained so far **can be applied** to other problems:

- testing if an evolved system preserves the behaviour of the original system.
- documenting refactoring changes performed in a redesigned system.

Research Strategy: First Stage

We apply some **restrictions to reduce complexity**:

- The transformation source is the current system code.
- The redesign proposal is the modified system code.

Goals of this stage are:

- To propose a solution to the restricted refactoring plan problem.
- To validate this solution.

Results obtained so far **can be applied** to other problems:

- testing if an evolved system preserves the behaviour of the original system.
- documenting refactoring changes performed in a redesigned system.

Research Strategy: First Stage

We apply some **restrictions to reduce complexity**:

- The transformation source is the current system code.
- The redesign proposal is the modified system code.

Goals of this stage are:

- To propose a solution to the restricted refactoring plan problem.
- To validate this solution.

Results obtained so far **can be applied** to other problems:

- testing if an evolved system preserves the behaviour of the original system.
- documenting refactoring changes performed in a redesigned system.

Research Strategy: First Stage

We apply some **restrictions to reduce complexity**:

- The transformation source is the current system code.
- The redesign proposal is the modified system code.

Goals of this stage are:

- To propose a solution to the restricted refactoring plan problem.
- To validate this solution.

Results obtained so far **can be applied** to other problems:

- testing if an evolved system preserves the behaviour of the original system.
- documenting refactoring changes performed in a redesigned system.

Research Strategy: Second Stage

After validating the first stage restricted solution, we will reduce restrictions to **generalize the problem**:

- The transformation source will still be the current system code.
- The redesign proposal will be allowed to be of different type, abstraction level, description language, . . . , than the source.
- Redesign proposals in the second stage will contain less information than redesign proposal in the first stage (modified system code).

Goals of this stage are:

- To specify which kind of descriptions will we be capable to deal with and would be allowed as redesign proposals.
- To propose a general solution to the problem of automatic elaboration of refactoring plans.
- To develop tool prototypes.

Research Strategy: Second Stage

After validating the first stage restricted solution, we will reduce restrictions to **generalize the problem**:

- The transformation source will still be the current system code.
- The redesign proposal will be allowed to be of different type, abstraction level, description language, . . . , than the source.
- Redesign proposals in the second stage will contain less information than redesign proposal in the first stage (modified system code).

Goals of this stage are:

- To specify which kind of descriptions will we be capable to deal with and would be allowed as redesign proposals.
- To propose a general solution to the problem of automatic elaboration of refactoring plans.
- To develop tool prototypes.

Research Strategy: Second Stage

After validating the first stage restricted solution, we will reduce restrictions to **generalize the problem**:

- The transformation source will still be the current system code.
- The redesign proposal will be allowed to be of different type, abstraction level, description language, . . . , than the source.
- Redesign proposals in the second stage will contain less information than redesign proposal in the first stage (modified system code).

Goals of this stage are:

- To specify which kind of descriptions will we be capable to deal with and would be allowed as redesign proposals.
- To propose a general solution to the problem of automatic elaboration of refactoring plans.
- To develop tool prototypes.

Research Strategy: Second Stage

After validating the first stage restricted solution, we will reduce restrictions to **generalize the problem**:

- The transformation source will still be the current system code.
- The redesign proposal will be allowed to be of different type, abstraction level, description language, . . . , than the source.
- Redesign proposals in the second stage will contain less information than redesign proposal in the first stage (modified system code).

Goals of this stage are:

- To specify which kind of descriptions will we be capable to deal with and would be allowed as redesign proposals.
- To propose a general solution to the problem of automatic elaboration of refactoring plans.
- To develop tool prototypes.

Research Strategy: Progress so far

We are at the first stage of the research, and working on these subproblems:

- Software representation
- Refactoring formalization
- “refactoring sequences discovering” algorithm

This presentation introduces the current progress of each one.

Refactoring Formalization

Any refactoring formalization method must allow:

- to deal with **system structure**.
- to **check** behaviour preserving **conditions**.

We will use **Graph Transformations** because:

- Representing and managing structural information is straightforward with graphs.
- This approach has already been validated (Mens et al., 2005).

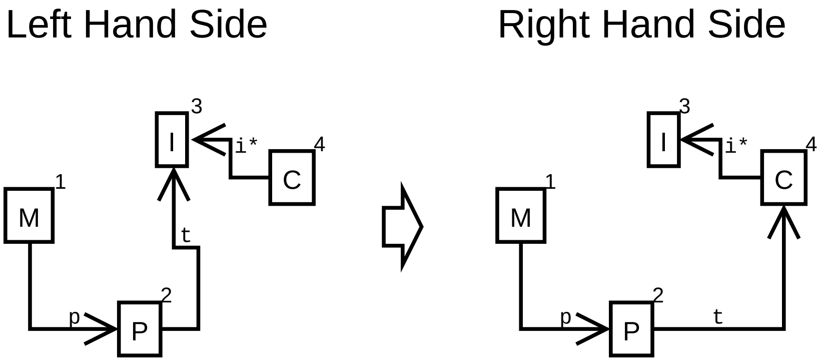
With Graph Transformation:

- **Software** is represented as **graphs**.
- **Refactorings** are represented as **graph transformation rules**.

Other refactoring formalization approaches:

- First Order Logic (Köch, 2002).

Example of a Graph Transformation Rule



Refactoring and Graph Transformation

We have found two main directions within the field of graph transformations, that can be useful:

- Rule Driven Systems
- Programmed Graph Rewriting Systems

Rule Driven Systems

Rule Driven Systems are graph rewriting systems where:

- transformation rules are described by a graph grammar
- transformations follow known derivation sequences
- rules are randomly selected to automatically transform a graph

The **problem** was **modeled as a formal language problem**:

- Current system $\Rightarrow$ starting node
- Refactorings $\Rightarrow$ production rules
- Desirable system $\Rightarrow$ final node
- Does the plan exist? $\Rightarrow$ membership problem
- Refactoring plan $\Rightarrow$ derivation sequence

Rule Driven Systems

Rule Driven Systems are graph rewriting systems where:

- transformation rules are described by a graph grammar
- transformations follow known derivation sequences
- rules are randomly selected to automatically transform a graph

The **problem** was **modeled as a formal language problem**:

- Current system $\Rightarrow$ starting node
- Refactorings $\Rightarrow$ production rules
- Desirable system $\Rightarrow$ final node
- Does the plan exist? $\Rightarrow$ membership problem
- Refactoring plan $\Rightarrow$ derivation sequence

Rule Driven Systems

We explored this approach and found some issues:

- This approach deals well with problems for which derivation paths are well defined (e.g. visual language parsing).
- A well known derivation tree for refactoring sequences should be needed
- The number of refactorings which can be applied for each derivation step is unpredictable

Programmed Graph Rewriting Systems

PGR Systems present a more general graph rewriting approach:

- graph grammars are also used
- they include structured programming
- transformations can be programmed and organized in modules

The **problem** can be **seen as a state space search problem**:

- Current system $\Rightarrow$ starting state
- Refactorings $\Rightarrow$ state changing operations
- Desirable System $\Rightarrow$ a desired state
- Does the plan exist? $\Rightarrow$ reachability of the desired state
- Refactoring Plan $\Rightarrow$ path to the final state

Issues found so far:

- the combinatorial explosion
- the need to develop heuristics

Programmed Graph Rewriting Systems

PGR Systems present a more general graph rewriting approach:

- graph grammars are also used
- they include structured programming
- transformations can be programmed and organized in modules

The **problem** can be **seen as a state space search problem**:

- Current system $\Rightarrow$ starting state
- Refactorings $\Rightarrow$ state changing operations
- Desirable System $\Rightarrow$ a desired state
- Does the plan exist? $\Rightarrow$ reachability of the desired state
- Refactoring Plan $\Rightarrow$ path to the final state

Issues found so far:

- the combinatorial explosion
- the need to develop heuristics

Programmed Graph Rewriting Systems

PGR Systems present a more general graph rewriting approach:

- graph grammars are also used
- they include structured programming
- transformations can be programmed and organized in modules

The **problem** can be **seen as a state space search problem**:

- Current system $\Rightarrow$ starting state
- Refactorings $\Rightarrow$ state changing operations
- Desirable System $\Rightarrow$ a desired state
- Does the plan exist? $\Rightarrow$ reachability of the desired state
- Refactoring Plan $\Rightarrow$ path to the final state

Issues found so far:

- the combinatorial explosion
- the need to develop heuristics

Refactoring Formalization with PGR Systems

We have chosen Programmed Graph Rewriting Systems because:

- they offer **programmable control** over the graph transformation process
- they offer **more expressiveness** than the grammar based systems
- **refactorings which take multiple transformation steps** are very difficult to describe with Rule Driven Systems, and they can be described easily with PGR Systems

In the first stage of the research we are addressing the already mentioned subproblems according to the PGR systems paradigm.

Refactoring Formalization with PGR Systems

We have chosen Programmed Graph Rewriting Systems because:

- they offer **programmable control** over the graph transformation process
- they offer **more expressiveness** than the grammar based systems
- **refactorings which take multiple transformation steps** are very difficult to describe with Rule Driven Systems, and they can be described easily with PGR Systems

In the first stage of the research we are addressing the already mentioned subproblems according to the PGR systems paradigm.

Software Representation: Program Graphs

A graph representation for Object-Oriented Software is needed. We must represent:

- elements of OO paradigm (classes, fields, methods, ...)
- structural relationships
- method bodies

We have chosen the software representation part from the refactoring formalization of (Mens et al., 2005). This representation:

- uses directed type graphs.
- is language independent, lacking specific language constructions.
- has been simplified to be as flexible as possible.

Software Representation: Program Graphs

A graph representation for Object-Oriented Software is needed. We must represent:

- elements of OO paradigm (classes, fields, methods, ...)
- structural relationships
- method bodies

We have chosen the software representation part from the refactoring formalization of (Mens et al., 2005). This representation:

- uses directed type graphs.
- is language independent, lacking specific language constructions.
- has been simplified to be as flexible as possible.

Software Representation: Java Program Graphs

For real systems, it is necessary to extend the graph format, adding:

- elements for specific languages
- more detailed representation of method bodies

We have extended program graphs for Java: **Java Program Graphs**.

Our graph representation format adds:

- Java concepts such as visibility, interfaces, packages, . . .
- More detailed representation of method bodies, with new node types, attributes and relationships.

Software Representation: Java Program Graphs

For real systems, it is necessary to extend the graph format, adding:

- elements for specific languages
- more detailed representation of method bodies

We have extended program graphs for Java: **Java Program Graphs**.

Our graph representation format adds:

- Java concepts such as visibility, interfaces, packages, . . .
- More detailed representation of method bodies, with new node types, attributes and relationships.

Software Representation: Java Program Graphs

For real systems, it is necessary to extend the graph format, adding:

- elements for specific languages
- more detailed representation of method bodies

We have extended program graphs for Java: **Java Program Graphs**.
Our graph representation format adds:

- Java concepts such as visibility, interfaces, packages, . . .
- More detailed representation of method bodies, with new node types, attributes and relationships.

Refactoring Rules Description

Refactorings will be formalized for PGR, by splitting its descriptions:

- pre and postconditions
- transformation process (the refactoring itself)
- transformation relationships between pre and post conditions.

Programmed Graph Rewritings will allow to:

- Use each refactoring description part when it's needed.
- Describe structured conditions:
  `if...then...else`
- Algorithmically describe refactorings that take multiple transformation steps:
  Pull Up Method

Refactoring Rules Description

Refactorings will be formalized for PGR, by splitting its descriptions:

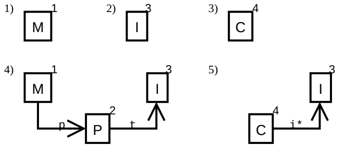
- pre and postconditions
- transformation process (the refactoring itself)
- transformation relationships between pre and post conditions.

Programmed Graph Rewritings will allow to:

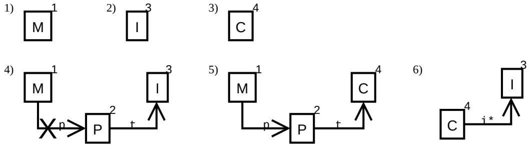
- Use each refactoring description part when it's needed.
- Describe structured conditions:
  `if...then...else`
- Algorithmically describe refactorings that take multiple transformation steps:
  Pull Up Method

Example: Change Interface Type Parameter

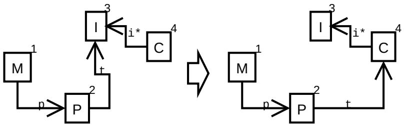
Preconditions:



Postconditions:



Transformation:



pre	→	post
1	→	1
2	→	2
3	→	3
4	→	4,5
5	→	6

Refactoring Sequences Discovering Algorithm

- A basic state space search algorithm is being developed.
- It needs refactoring descriptions to be expressed in terms of preconditions, transformations and postconditions.
- It needs the expressiveness and execution control of programmed graph rewriting.
- The **algorithm main keys** are:
  - The Algorithm is **guided by pre and postconditions**.
  - **To look for preconditions in the source graph** in order to reveal which refactorings can be applied.
  - **To look for postconditions in the target graph** in order to fi nd which refactorings are more likely to be part of the sequence.
  - The **source graph** gets **transformed** progressively **into the target graph**.

Refactoring Sequences Discovering Algorithm

- A basic state space search algorithm is being developed.
- It needs refactoring descriptions to be expressed in terms of preconditions, transformations and postconditions.
- It needs the expressiveness and execution control of programmed graph rewriting.
- The **algorithm main keys** are:
  - The Algorithm is **guided by pre and postconditions**.
  - **To look for preconditions in the source graph** in order to reveal which refactorings can be applied.
  - **To look for postconditions in the target graph** in order to fi nd which refactorings are more likely to be part of the sequence.
  - The **source graph** gets **transformed** progressively **into the target graph**.

Refactoring Sequences Discovering Algorithm

- A basic state space search algorithm is being developed.
- It needs refactoring descriptions to be expressed in terms of preconditions, transformations and postconditions.
- It needs the expressiveness and execution control of programmed graph rewriting.
- The **algorithm main keys** are:
  - The Algorithm is **guided by pre and postconditions**.
  - **To look for preconditions in the source graph** in order to reveal which refactorings can be applied.
  - **To look for postconditions in the target graph** in order to fi nd which refactorings are more likely to be part of the sequence.
  - The **source graph** gets **transformed** progressively **into the target graph**.

Refactoring Sequences Discovering Algorithm

- A basic state space search algorithm is being developed.
- It needs refactoring descriptions to be expressed in terms of preconditions, transformations and postconditions.
- It needs the expressiveness and execution control of programmed graph rewriting.
- The **algorithm main keys** are:
  - The Algorithm is **guided by pre and postconditions**.
  - **To look for preconditions in the source graph** in order to reveal which refactorings can be applied.
  - **To look for postconditions in the target graph** in order to fi nd which refactorings are more likely to be part of the sequence.
  - The **source graph** gets **transformed** progressively **into the target graph**.

Refactoring Sequences Discovering Algorithm

- A basic state space search algorithm is being developed.
- It needs refactoring descriptions to be expressed in terms of preconditions, transformations and postconditions.
- It needs the expressiveness and execution control of programmed graph rewriting.
- The **algorithm main keys** are:
  - The Algorithm is **guided by pre and postconditions**.
  - **To look for preconditions in the source graph** in order to reveal which refactorings can be applied.
  - **To look for postconditions in the target graph** in order to fi nd which refactorings are more likely to be part of the sequence.
  - The **source graph** gets **transformed** progressively **into the target graph**.

Refactoring Sequences Discovering Algorithm

- A basic state space search algorithm is being developed.
- It needs refactoring descriptions to be expressed in terms of preconditions, transformations and postconditions.
- It needs the expressiveness and execution control of programmed graph rewriting.
- The **algorithm main keys** are:
    - The Algorithm is **guided by pre and postconditions**.
    - **To look for preconditions in the source graph** in order to reveal which refactorings can be applied.
    - **To look for postconditions in the target graph** in order to fi nd which refactorings are more likely to be part of the sequence.
    - The **source graph** gets **transformed** progressively **into the target graph**.

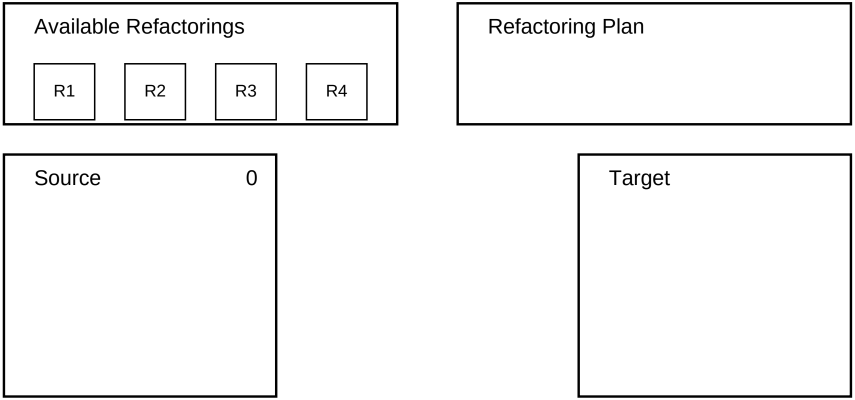
Refactoring Sequences Discovering Algorithm

- A basic state space search algorithm is being developed.
- It needs refactoring descriptions to be expressed in terms of preconditions, transformations and postconditions.
- It needs the expressiveness and execution control of programmed graph rewriting.
- The **algorithm main keys** are:
  - The Algorithm is **guided by pre and postconditions**.
  - **To look for preconditions in the source graph** in order to reveal which refactorings can be applied.
  - **To look for postconditions in the target graph** in order to fi nd which refactorings are more likely to be part of the sequence.
  - The **source graph** gets **transformed** progressively **into the target graph**.

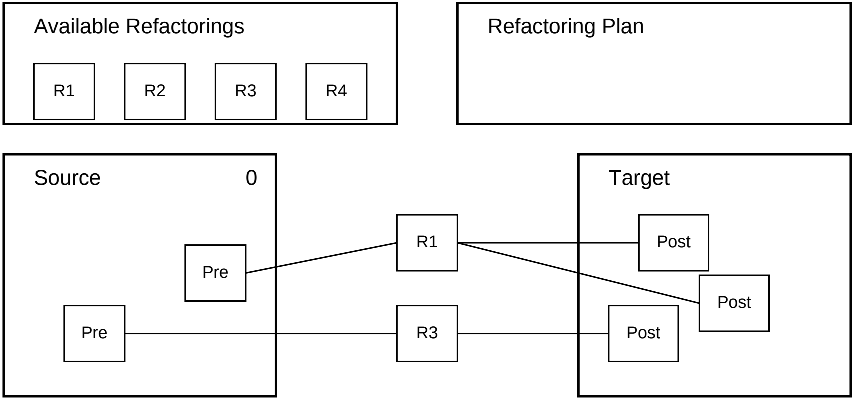
Refactoring Sequences Discovering Algorithm

- A basic state space search algorithm is being developed.
- It needs refactoring descriptions to be expressed in terms of preconditions, transformations and postconditions.
- It needs the expressiveness and execution control of programmed graph rewriting.
- The **algorithm main keys** are:
  - The Algorithm is **guided by pre and postconditions**.
  - **To look for preconditions in the source graph** in order to reveal which refactorings can be applied.
  - **To look for postconditions in the target graph** in order to fi nd which refactorings are more likely to be part of the sequence.
  - The **source graph** gets **transformed** progressively **into the target graph**.

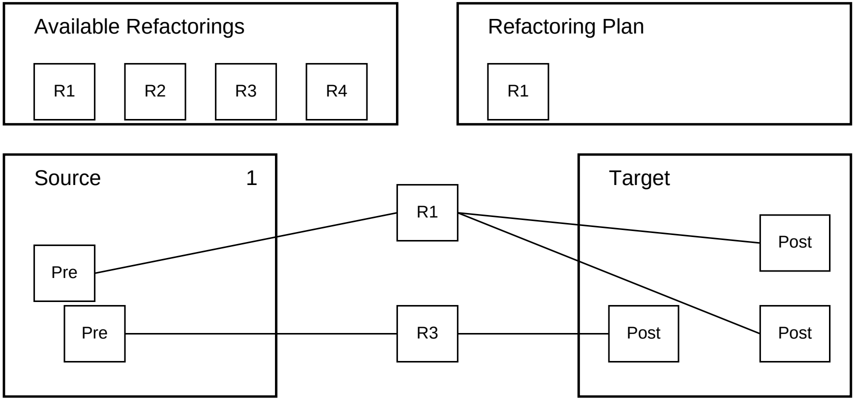
Refactoring Sequences Discovering Algorithm



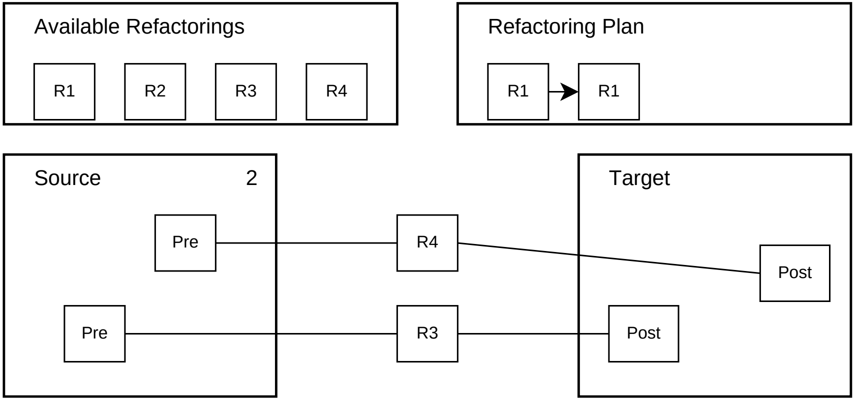
Refactoring Sequences Discovering Algorithm



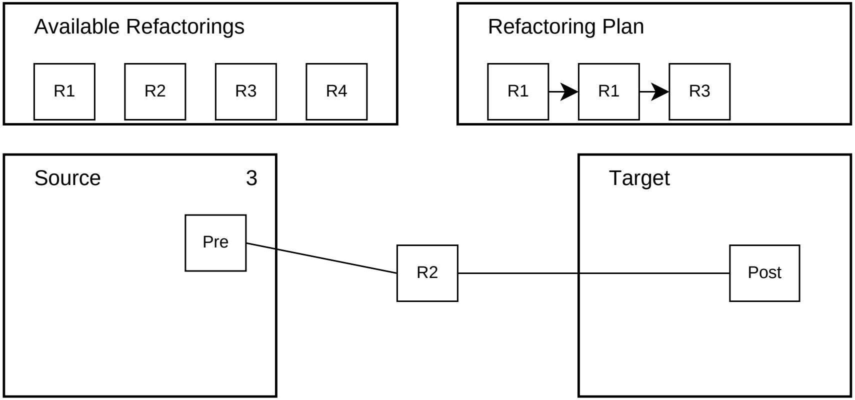
Refactoring Sequences Discovering Algorithm



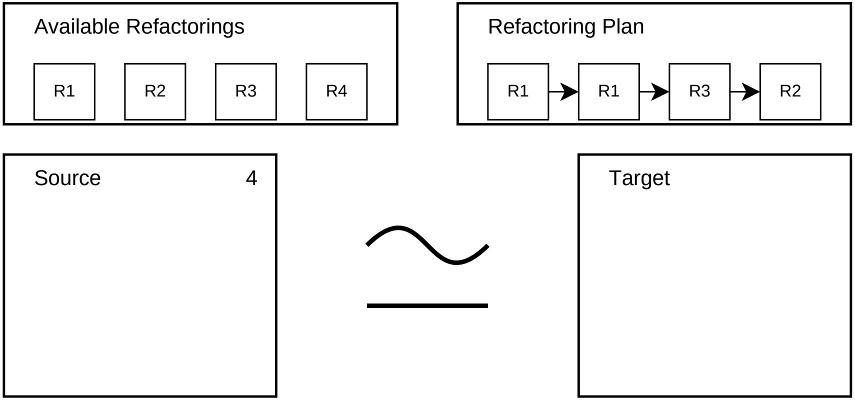
Refactoring Sequences Discovering Algorithm



Refactoring Sequences Discovering Algorithm

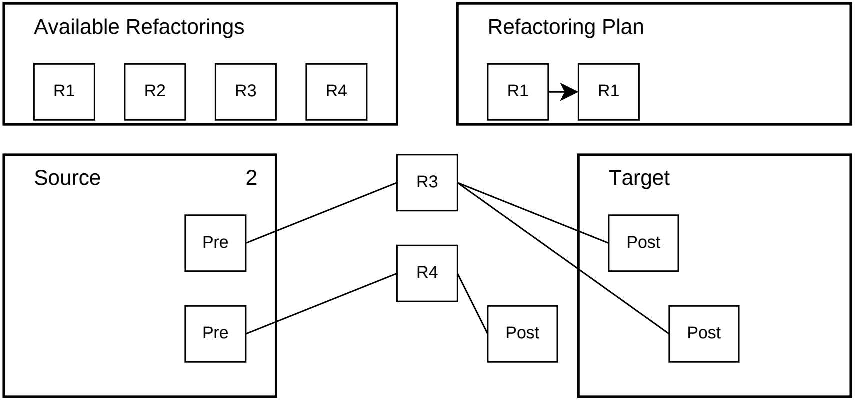


Refactoring Sequences Discovering Algorithm



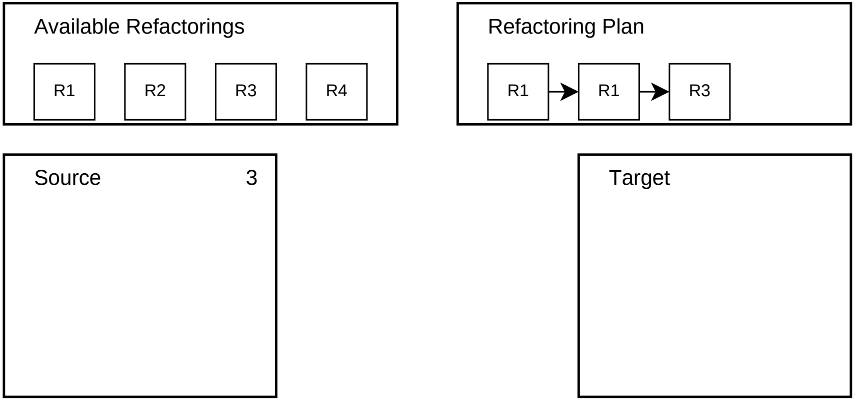
“Postcondition” Heuristic

- Prioritize refactorings whose postconditions hold on the target graph.



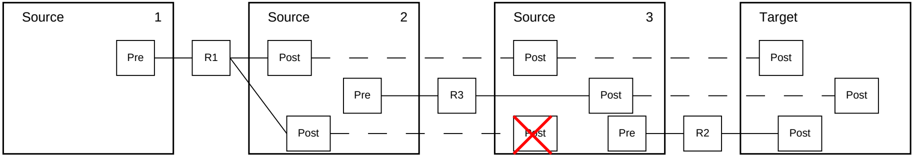
“Postcondition” Heuristic

- Prioritize refactorings whose postconditions hold on the target graph.



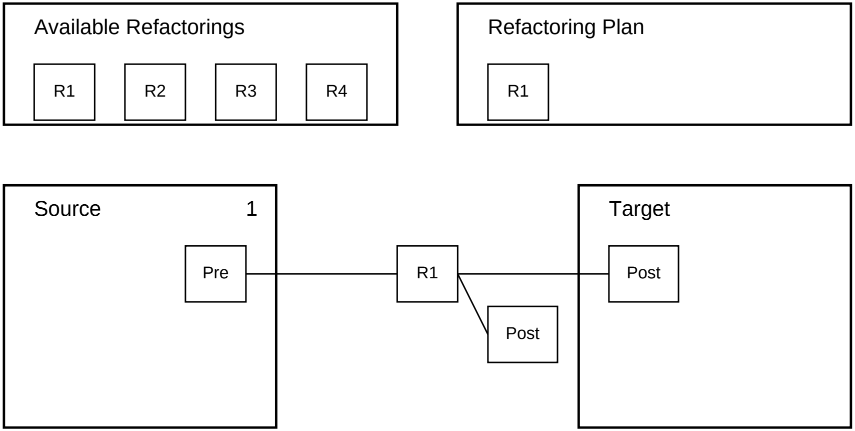
“Awaiting Postcondition” Heuristic

- Prioritize refactorings which make that the previous selected. refactorings can hold their awaiting postconditions



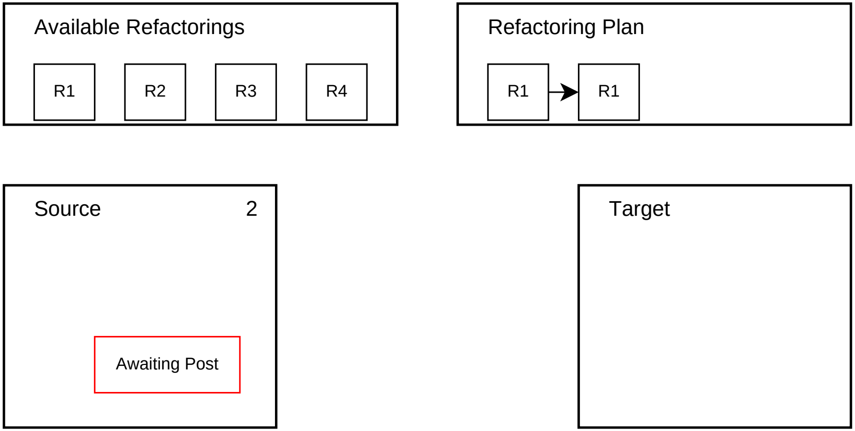
“Awaiting Postcondition” Heuristic

- Prioritize refactorings which make that the previous selected. refactorings can hold their awaiting postconditions



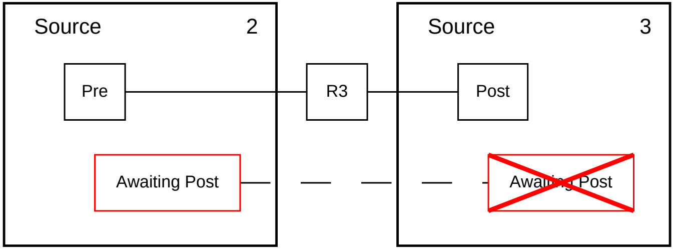
“Awaiting Postcondition” Heuristic

- Prioritize refactorings which make that the previous selected. refactorings can hold their awaiting postconditions



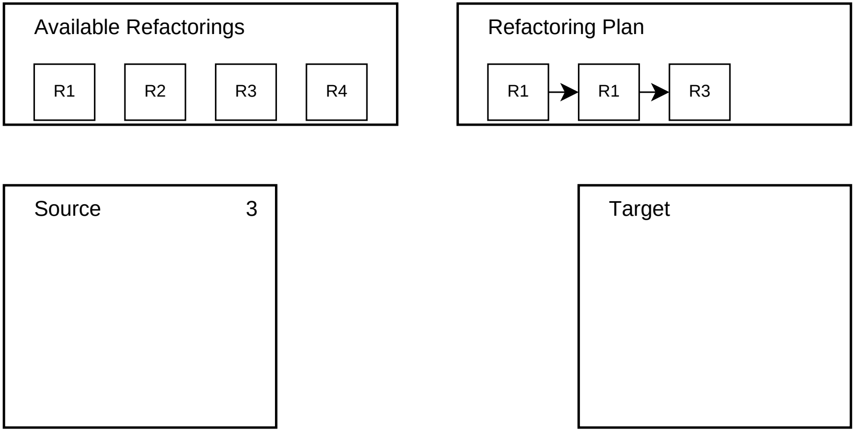
“Awaiting Postcondition” Heuristic

- Prioritize refactorings which make that the previous selected. refactorings can hold their awaiting postconditions



"Awaiting Postcondition" Heuristic

- Prioritize refactorings which make that the previous selected. refactorings can hold their awaiting postconditions



Conclusions

- Automatic generation of refactoring plans will provide very high level refactorings to improve the design of existing code.
- The Main subproblems and the research strategy have been introduced.
- Graph transformation can be used as the underlying formalism, specifically the programmed graph rewriting approach.
  - Representing Java programs with Java Program Graphs.
  - Describing refactoring rules with programmed graph transformation rules in terms of pre, postconditions and transformations.
- The problem can be modeled as a state space search problem.
  - Using a “refactoring sequences discovering” algorithm guided by pre and postconditions, to fi nd a refactoring plan, .
  - Using heuristics to guide the algorithm.

Conclusions

- Automatic generation of refactoring plans will provide very high level refactorings to improve the design of existing code.
- The Main subproblems and the research strategy have been introduced.
- Graph transformation can be used as the underlying formalism, specifically the programmed graph rewriting approach.
  - Representing Java programs with Java Program Graphs.
  - Describing refactoring rules with programmed graph transformation rules in terms of pre, postconditions and transformations.
- The problem can be modeled as a state space search problem.
  - Using a “refactoring sequences discovering” algorithm guided by pre and postconditions, to fi nd a refactoring plan, .
  - Using heuristics to guide the algorithm.

Conclusions

- Automatic generation of refactoring plans will provide very high level refactorings to improve the design of existing code.
- The Main subproblems and the research strategy have been introduced.
- Graph transformation can be used as the underlying formalism, specifically the programmed graph rewriting approach.
    - Representing Java programs with Java Program Graphs.
    - Describing refactoring rules with programmed graph transformation rules in terms of pre, postconditions and transformations.
- The problem can be modeled as a state space search problem.
    - Using a “refactoring sequences discovering” algorithm guided by pre and postconditions, to fi nd a refactoring plan, .
    - Using heuristics to guide the algorithm.

Conclusions

- Automatic generation of refactoring plans will provide very high level refactorings to improve the design of existing code.
- The Main subproblems and the research strategy have been introduced.
- Graph transformation can be used as the underlying formalism, specifically the programmed graph rewriting approach.
    - Representing Java programs with Java Program Graphs.
    - Describing refactoring rules with programmed graph transformation rules in terms of pre, postconditions and transformations.
- The problem can be modeled as a state space search problem.
    - Using a “refactoring sequences discovering” algorithm guided by pre and postconditions, to fi nd a refactoring plan, .
    - Using heuristics to guide the algorithm.

Detected Problems

Correction and completeness of the algorithm:

- Heuristics could not be enough to prevent the algorithm getting lost in the search.
- The number of applicable refactorings on each step is expectable to be very big.
- The algorithm can get stuck selecting “wrong” refactorings.

Future Work

Main future tasks will be directed to:

- Further definition of the “Refactoring Plan” concept.
- Extend the available set of formalized refactorings.
- Validate proposals implementing the algorithm with a PGR tool.
- Analyse termination and correctness conditions of the refactoring discovering algorithm.
- Analyse, improve and extend heuristics of the state space search algorithm.

Overview of the Refactoring Discovering Problem

Javier Pérez
`jperez@infor.uva.es`

Universidad de Valladolid

ECOOP 2006, Doctoral Symposium and PhD Students Workshop