Enabling Refactoring with HTN Planning to Improve the Design Smells Correction Activity

Javier Pérez
jperez@infor.uva.es
www.infor.uva.es/~jperez

Universidad de Valladolid

BENEVOL 2008
Dec 11-12 2008, Eindhoven
Design Smell Correction
Design Smells

Problems encountered in the software’s structure (code or design), that can be detected statically, that do not produce compile or run-time errors, but negatively affect software quality factors. In fact, this negative effect on quality factors could lead to real compile and run-time errors in the future.

- In the context of software inconsistencies:
  - consistency maintenance (keeping models consistent)
  - inconsistency management (detect and resolve inconsistencies)
  - co-evolution (manage consistency between different artefacts)

- Design smells are corrected with refactorings
Design Smell Correction

Object-Oriented Software Design Smells

Design Smells
Problems encountered in the software’s structure (code or design), that can be detected statically, that do not produce compile or run-time errors, but negatively affect software quality factors. In fact, this negative effect on quality factors could lead to real compile and run-time errors in the future.

- In the context of software inconsistencies:
  - consistency maintenance (keeping models consistent)
  - inconsistency management (detect and resolve inconsistencies)
  - co-evolution (manage consistency between different artefacts)

- Design smells are corrected with refactorings
Design Smell Correction

Object-Oriented Software Design Smells

Design Smells

Problems encountered in the software’s structure (code or design), that can be detected statically, that do not produce compile or run-time errors, but negatively affect software quality factors. In fact, this negative effect on quality factors could lead to real compile and run-time errors in the future.

- In the context of software inconsistencies:
  - consistency maintenance (keeping models consistent)
  - inconsistency management (detect and resolve inconsistencies)
  - co-evolution (manage consistency between different artefacts)

- Design smells are corrected with refactorings
Design Smells

Problems encountered in the software’s structure (code or design), that can be detected statically, that do not produce compile or run-time errors, but negatively affect software quality factors. In fact, this negative effect on quality factors could lead to real compile and run-time errors in the future.

- In the context of software inconsistencies:
  - consistency maintenance (keeping models consistent)
  - inconsistency management (detect and resolve inconsistencies)
  - co-evolution (manage consistency between different artefacts)

- Design smells are corrected with refactorings
Design Smells

Problems encountered in the software’s structure (code or design), that can be detected statically, that do not produce compile or run-time errors, but **negatively affect software quality factors**. In fact, this negative effect on quality factors could lead to real compile and run-time errors in the future.

- In the context of software inconsistencies:
  - consistency maintenance (keeping models consistent)
  - inconsistency management (detect and resolve inconsistencies)
  - co-evolution (manage consistency between different artefacts)

- Design smells are corrected with refactorings
Object-Oriented Software Design Smells

Design Smells

Problems encountered in the software’s structure (code or design), that can be detected statically, that do not produce compile or run-time errors, but negatively affect software quality factors. In fact, this negative effect on quality factors could lead to real compile and run-time errors in the future.

- In the context of software inconsistencies:
  - consistency maintenance (keeping models consistent)
  - inconsistency management (detect and resolve inconsistencies)
  - co-evolution (manage consistency between different artefacts)

Design smells are corrected with refactorings
Design Smells

Problems encountered in the software’s structure (code or design), that can be detected statically, that do not produce compile or run-time errors, but negatively affect software quality factors. In fact, this negative effect on quality factors could lead to real compile and run-time errors in the future.

- In the context of software inconsistencies:
  - consistency maintenance (keeping models consistent)
  - inconsistency management (detect and resolve inconsistencies)
  - co-evolution (manage consistency between different artefacts)

- Design smells are corrected with refactorings
Design Smells

Problems encountered in the software’s structure (code or design), that can be detected statically, that do not produce compile or run-time errors, but negatively affect software quality factors. In fact, this negative effect on quality factors could lead to real compile and run-time errors in the future.

- In the context of software inconsistencies:
  - consistency maintenance (keeping models consistent)
  - inconsistency management (detect and resolve inconsistencies)
  - co-evolution (manage consistency between different artefacts)

- Design smells are corrected with refactorings
Brief History of Design Smell Management

- Heuristics for Good OO Design
  (1996 Riel) "Object-Oriented Design Heuristics"

- Bad Smells Catalogs
  (1999 Fowler) "Refactoring; Improving the ..."

- Precise Smell Specifications
  (2006 Lanza, Marinescu) "OO Metrics ..."

- Automated Smell Detection
  (2008 Moha) "DECOR: Détection et Correction ..."

- Correction Catalogs
  (2004 Kerievsky) "Refactoring to Patterns"

- Precise Correction Specifications
  (2008 Trifu) Towards Automated Restructuring ..."

- Automated Correction
  ???

Detection

Correction
Brief History of Design Smell Management

**Heuristics for Good OO Design**
(1996 Riel) "Object-Oriented Design Heuristics"

**Bad Smells Catalogs**
(1999 Fowler) "Refactoring: Improving the ..."

**Precise Smell Specifications**
(2006 Lanza, Marinescu) "OO Metrics ..."

**Automated Smell Detection**
(2008 Moha) "DECOR: Détectie et Correction ..."

**Correction Catalogs**
(2004 Kerievsky) "Refactoring to Patterns"

**Precise Correction Specifications**
(2008 Trifu) Towards Automated Restructuring ...

**Automated Correction**
???
Smell Example: Feature Envy
Feature Envy

“... a method that seems more interested in a class other than the one it actually is in.” (Fowler et al., 1999)
**Feature Envy Example**

- `formatSummary()` uses many attributes from `Document` and none from its own class.

- The strategy is to **move** the **method** to `Document` but a method with the same signature already exists.
**Feature Envy Example**

- `formatSummary()` uses many attributes from `Document` and none from its own class.
- The strategy is to move the method to `Document` but a method with the same signature already exists.
**Feature Envy Example**

- `formatSummary()` uses many attributes from `Document` and none from its own class.

- The strategy is to move the method to `Document` but a method with the same signature already exists.
Feature Envy Example

- `formatSummary()` uses many attributes from `Document` and none from its own class.
- The strategy is to move the method to `Document` but a method with the same signature already exists.
**Feature Envy Example**

- `formatSummary()` uses many attributes from `Document` and none from its own class.
- The strategy is to **move** the method to `Document` but a method with the same signature already exists.
Smell Example: Feature Envy

Problems in Automated Correction

- Which is the strategy to correct a smell?
  - Feature Envy (m) ⇒ move method m close to data, from class s to class t

- Which is the precise strategy instance to use?
  - Feature Envy (formatSummary) ⇒ move method formatSummary from Printserver to Document

- How to apply the strategy instance?
  - move method formatSummary from Printserver to Document ⇒
    1. first remove formatSummary in Document or rename formatSummary in Document or rename formatSummary in Printserver
    2. then move method formatSummary from Printserver to Document
Refactoring Planning
Refactoring Plans

The objective: Instantiate smell correction strategies into a correction plan which could be effectively applied, or at least could guide the developer through the process.

Refactoring Plan

Specification of a refactoring sequence which matches a system redesign proposal, and that can be immediately executed to modify the system, without changing the system’s behaviour, in order to obtain that desirable system redesign.
The objective: Instantiate smell correction strategies into a correction plan which could be effectively applied, or at least could guide the developer through the process.

Refactoring Plan

Specification of a refactoring sequence which matches a system redesign proposal, and that can be immediately executed to modify the system, without changing the system’s behaviour, in order to obtain that desirable system redesign.
The objective: Instantiate smell correction strategies into a correction plan which could be effectively applied, or at least could guide the developer through the process.

Refactoring Plan

Specification of a refactoring sequence which matches a system redesign proposal, and that can be immediately executed to modify the system, without changing the system’s behaviour, in order to obtain that desirable system redesign.
The objective: Instantiate smell correction strategies into a correction plan which could be effectively applied, or at least could guide the developer through the process.

Refactoring Plan

Specification of a refactoring sequence which matches a system redesign proposal, and that can be immediately executed to modify the system, without changing the system’s behaviour, in order to obtain that desirable system redesign.
The objective: Instantiate smell correction strategies into a correction plan which could be effectively applied, or at least could guide the developer through the process.

Refactoring Plan

Specification of a refactoring sequence which matches a system redesign proposal, and that can be immediately executed to modify the system, without changing the system’s behaviour, in order to obtain that desirable system redesign.
Refactoring Plans

The objective: Instantiate smell correction strategies into a correction plan which could be effectively applied, or at least could guide the developer through the process.

Refactoring Plan

Specification of a refactoring sequence which matches a system redesign proposal, and that can be immediately executed to modify the system, without changing the system’s behaviour, in order to obtain that desirable system redesign.
The objective: Instantiate smell correction strategies into a correction plan which could be effectively applied, or at least could guide the developer through the process.

Refactoring Plan

Specification of a refactoring sequence which matches a system redesign proposal, and that can be immediately executed to modify the system, without changing the system’s behaviour, in order to obtain that desirable system redesign.
Refactoring plans can be computed with automated planning.
Refactoring plans can be computed with automated planning
Automated Planning

Definition

Automated planning is an artificial intelligence technique to generate sequences of actions that will achieve a certain goal when they are performed.

Example: Getting apples and a book.

The state of the world: at (grocery) AND not (have (apples))
Actions: buy (apples); moveTo (bookstore)
Goals: have (book) AND have (apples)
Plan

O₁ → S₀ → S₁ → S₂ → Sₙ → S_G → O₂ → Oₙ → O_G
Classical Planning Operators (STRIPS)

- **World’s state:** list of terms

- **Operators:**
  - definition: name + arguments
  - precondition
  - effect list (add): terms to add to the state
  - effect list (deletes): terms to remove from the state

- **Problem:**
  - initial state
  - goal: list of terms

- **General planning approach:** chain operators by matching their effects and preconditions
Some Types of Planners

- Depending on the **planning space**:
  - state space planning
  - plan space planning

- Depending on the **search direction**:
  - forward searching
  - backwards searching

- Depending on when the **operator ordering** is committed:
  - total-order planning
  - partial-order planning

- I’m using Hierarchical Task Network (HTN) planning.
Some Types of Planners

- Depending on the **planning space**:
  - state space planning
  - plan space planning

- Depending on the **search direction**:
  - forward searching
  - backwards searching

- Depending on when the **operator ordering** is committed:
  - total-order planning
  - partial-order planning

- I’m using **Hierarchical Task Network (HTN) planning**.
Hierarchical Task Network (HTN) Planning

- **Task 1**
  - **Method 1**
    - **Precondition 1**
      - Task 2
      - Task 3
  - **Method 2**
    - **Precondition 2**
      - Task 4
      - Operator 1
        - **Precondition 3**
          - ADD
          - DEL

- ...
Hierarchical Task Network (HTN) Planning

- **method1**
  - **precondition1**
    - **task2**
    - **task3**
- **method2**
  - **precondition2**
    - **task4**
    - **operator1**
      - **precondition3**
        - **ADD**
        - **DEL**

...
Hierarchical Task Network (HTN) Planning

- method1
  - precondition1
    - task2
    - task3
- method2
  - precondition2
    - task4
    - operator1
      - precondition3
        - ADD
        - DEL
- ...
Hierarchical Task Network (HTN) Planning

- method1
  - precondition1
    - task2
    - task3
- method2
  - precondition2
    - task4
    - operator1
  - precondition3
    - ADD
    - DEL
- task1
- ...
Hierarchical Task Network (HTN) Planning

- method1
  - precondition1
    - task2
    - task3
  - method2
    - precondition2
      - task4
      - operator1
        - precondition3
          - ADD
          - DEL
- task1
- ...
Hierarchical Task Network (HTN) Planning

- **task1**
  - **method1**
    - **precondition1**
      - **task2**
      - **task3**
  - **method2**
    - **precondition2**
      - **task4**
      - **operator1**
        - **precondition3**
          - **ADD**
          - **DEL**

Goal
Hierarchical Task Network (HTN) Planning

Goal
Execute a task
Smell Correction with HTN Planning

- **World's state:** AST represented by first order logic formulas

- **Operators:** refactoring substeps

- **Tasks:**
  - refactorings strategies
  - smell correction strategies

- **Goals:** Execute a smell correction strategy

- **Planning Problem:** Execute a particular smell correction strategy over a particular version of a system
Planning for “Feature Envy”
HTN for “move method”

1. move_method (Class source, Method m, Class target)

   - everything_ok
   - name_conflict

   - not (isStatic(m))
   - isReachable(target, m)
   - not (callsSuper(m))
   - not (isNameConflict(m, target))

2. move_method_definition (Class source, Method m, Class target)

3. solve_name_conflict (Class source, Method m, Class target)

4. update_method_calls (Class source, Method m, Class target)

5. move_method (Class source, Method m, Class target)
Planning for “Feature Envy”

HTN for “solve conflict”

solve_name_conflict (Class source, Method m, Class target)
  ↓
rename_source
  ↓
rename_method (Class source, Method m)
  ↓
rename_target
  ↓
rename_method (Class target, Method m)

remove_target
  ↓
remove_unreferenced_method (Class target, Method m)
Planning for “Feature Envy”

HTN for “feature envy”

1. `correct_feature_envy (Class c, Method m)`
2. `move_method_close_to_data`
3. `Class target = whereToMove (c, m)`
4. `move_method (Class c, Method m, Class target)`

Javier Pérez (UVa)
Refactoring Planning
11-12 Dec 2008
Planning for “feature envy”

- `correct_feature_envy (Printserver, formatSummary)`
- `move_method_close_to_data`
- `whereToMove (Printserver, formatSummary) -> Document`
- `move_method (Printserver, formatSummary, Document)`
Planning for “move method” 1

move_method (Printserver, formatSummary, Document)

- everything_ok
  - not (isStatic(formatSummary))
  - isReachable(Document, formatSummary)
  - not (callsSuper(formatSummary))
  - not (isNameConflict(formatSummary, Document))

- name_conflict
  - not (isStatic(formatSummary))
  - isReachable(Document, formatSummary)
  - not (callsSuper(formatSummary))
  - isNameConflict(formatSummary, Document)

- solve_name_conflict (Printserver, formatSummary, Document)

move_method (Printserver, formatSummary, Document)
Planning for "move method" 2

- solve_name_conflict (Printserver, formatSummary, Document)
- remove_target
- remove_unreferenced_method (Document, formatSummary)
- move_method (Printserver, formatSummary, Document)
- everything_ok
- move_method_definition (Class source, Method m, Class target)
- update_method_calls (Class source, Method m, Class target)

- not (isStatic (formatSummary))
- isReachable (Document, formatSummary)
- not (callsSuper (formatSummary))
- not (isNameConflict (formatSummary, Document))
Planning for "move method" 2

- solve_name_conflict (Printserver, formatSummary, Document)
- remove_target
- move_method (Printserver, formatSummary, Document)
- remove_unreferenced_method (Document, formatSummary)
- move_method_definition (Class source, Method m, Class target)
- update_method_calls (Class source, Method m, Class target)

everything_ok

- not (isStatic (formatSummary))
- isReachable (Document, formatSummary)
- not (callsSuper (formatSummary))
- not (isNameConflict (formatSummary, Document))
Conclusions and Future Work
Conclusions

Design smell management can keep being improved, working on specification and automation of the correction activity.

To do that, correction strategies must be planned ahead for each specific case.

This can be done with automated planning and specifically with HTN planning:

- HT networks can accommodate correction strategies, combining procedural and non-deterministic searching.
- HTN planning offers good balance between procedural execution and non-determinism.
- The planner can be incrementally extended, adding new methods and improving the existing ones.
Conclusions and Future Work

Conclusions

- Design smell management can keep being improved, working on specification and automation of the correction activity.
- To do that, correction strategies must be planned ahead for each specific case.
- This can be done with automated planning and specifically with HTN planning:
  - HT networks can accommodate correction strategies, combining procedural and non-deterministic searching.
  - HTN planning offers good balance between procedural execution and non-determinism.
  - The planner can be incrementally extended, adding new methods and improving the existing ones.
Conclusions and Future Work

Conclusions

- Design smell management can keep being improved, working on specification and automation of the correction activity.
- To do that, correction strategies must be planned ahead for each specific case.
- This can be done with automated planning and specifically with HTN planning:
  - HT networks can accommodate correction strategies, combining procedural and non-deterministic searching.
  - HTN planning offers good balance between procedural execution and non-determinism.
  - The planner can be incrementally extended, adding new methods and improving the existing ones.
Conclusions

- Design smell management can keep being improved, working on specification and automation of the correction activity.
- To do that, correction strategies must be planned ahead for each specific case.
- This can be done with automated planning and specifically with HTN planning:
  - HT networks can accommodate correction strategies, combining procedural and non-deterministic searching.
  - HTN planning offers good balance between procedural execution and non-determinism.
  - The planner can be incrementally extended, adding new methods and improving the existing ones.
Conclusions and Future Work

Conclusions

- Design smell management can keep being improved, working on specification and automation of the correction activity.
- To do that, correction strategies must be planned ahead for each specific case.
- This can be done with automated planning and specifically with HTN planning:
  - HT networks can accommodate correction strategies, combining procedural and non-deterministic searching.
  - HTN planning offers good balance between procedural execution and non-determinism.
  - The planner can be incrementally extended, adding new methods and improving the existing ones.
Conclusions

- Design smell management can keep being improved, working on specification and automation of the correction activity.
- To do that, correction strategies must be planned ahead for each specific case.
- This can be done with automated planning and specifically with HTN planning:
  - HT networks can accommodate correction strategies, combining procedural and non-deterministic searching.
  - HTN planning offers good balance between procedural execution and non-determinism.
  - The planner can be incrementally extended, adding new methods and improving the existing ones.
Future Work

- Implement refactoring specifications
- Implement design smell correction strategies
- Run experiments on real systems
- Integrate the planer with other tools for:
  - refactoring dependencies computation
  - metrics computation
  - ...

Javier Pérez (UVa)
Enabling Refactoring with HTN Planning to Improve the Design Smells Correction Activity

Javier Pérez
jperez@infor.uva.es
www.infor.uva.es/~jperez

Universidad de Valladolid

BENEVOL 2008
Dec 11-12 2008, Eindhoven