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.
- Referred as defects, flaws, disharmonies, antipatterns, by other authors.
- Design Smell is proposed as a unifying term.
Refactorings

✧ structural transformations
✧ to perform design changes
✧ without modifying the system’s observable behaviour.


✧ behaviour preserving invariants + preconditions + transformations


✧ bad smells as a guide for refactoring

✧ 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.
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)

- 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

* 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
# Analysis of 22 Design Smell Management Tools

## Supported Activity

<table>
<thead>
<tr>
<th></th>
<th>Specification</th>
<th>Detection</th>
<th>Visualisation</th>
<th>Correction</th>
<th>Impact Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>9</td>
<td>22</td>
<td>7</td>
<td>9</td>
<td>2</td>
</tr>
</tbody>
</table>

## Degree of Automation

<table>
<thead>
<tr>
<th></th>
<th>Manual</th>
<th>Interactive</th>
<th>Fully Automated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detection</td>
<td>0</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>Correction</td>
<td>1</td>
<td>8</td>
<td>0</td>
</tr>
</tbody>
</table>

## Type of Result

<table>
<thead>
<tr>
<th></th>
<th>Correction Suggestion</th>
<th>Refactorings / Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8</td>
<td>1</td>
</tr>
</tbody>
</table>

* 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

Prototype

Refactoring Strategies
Requirements

Design Smell
Correction
Specifications

Refactoring
Strategies
Specifications

Evaluation
2 Case Studies

Requirements

Refactoring
Plan
Computation
Requirements

Brown
Beck
Wake
Kerievsky
Marinescu
Demeyer
Trifu

Correction Strategy

Quality-based Condition

Transformation

Non-Behaviour Preserving Transformation

Behaviour-Preserving Transformation

1..* 0..*

Correction Strategy

1

Possible strategies

Step

Composed

Alternatives

Loop

Condition

Quality-Based Condition

Transformation

Non-Behaviour Preserving Transformation

Behaviour-Preserving Transformation

1..* 1

Output

1..* 1

1

Parameter

Check

Precondition

Compute Parameters

Apply Transformation

Control

Refactoring Strategies

Requirements

Evaluation

2 Case Studies

Refactoring Strategies

Specification Language
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

```plaintext
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
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
Refactoring Plans from Refactoring Strategies: Requirements

- **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

- initial state
- objectives
- execution status
- observations
- events

system description

planner

controller

system

plans

actions

Refactoring Planning

AST + transformations

current AST

remove smell

refactoring planner

refactoring plans

software developer

source code transformations

source code

software system
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
**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

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

- **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

```plaintext
if COND1 then
    STEPS1
elsif COND2 then
    STEPS2
else
    STEPS3
end
```

```plaintext
;; if invocation
(if-xx ARGS)

;; if definition
(method (if-xx ARGS)
    (COND1)
    (STEPS1)
    (COND2)
    (STEPS2)
    ()
    (STEPS3)
)
```
B.8 Loops: while

A while loop is translated with a single method that contains two branches. The first branch of the method is used for translating the loop’s elements, while the second branch has an empty precondition and an empty task decomposition list that works as the loop’s exit branch. The loop’s stay condition is translated as the first branch’s precondition. The block of steps in the loop’s body is translated into the first branch’s task decomposition list. These steps should be recursively translated if needed. The last task in the first branch’s decomposition list should be a recursive invocation of the loop’s translated method.

Variable bindings taking place within the loop’s method precondition are discarded between loop iterations by default. The variables in the loop VARS section should be kept between iterations. In order to translate it, the variables in VARS should be included in the ARGS argument list.

The method representing the while statement has to be named with a symbol with the prefix while- and a numeric suffix, so that the resulting symbol provides a unique name for the method in the refactoring planning domain. The while statement in the original sequence of steps should be then replaced by the invocation of the method. The method’s arguments ARGS should contain all the variables appearing in the method’s steps that are accessible from the

```logic
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
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 all-sts (class)
- remove-data-class trivial (class)
- remove-class (class)
- remove-data-class reorganize-class (class)
- remove-data-class move-client-methods (class)
- remove-data-class move-fe-methods (class)
- remove-feature-envy move-method-to-envied-class (method, env-class)
- move-method all-sts (method, tgt-class, reference)
- remove-data-class clean-class (class)
- encapsulate-fields non-private (class)
- remove-getters unused (class)
- remove-sets unused (class)
- encapsulate-field (field, getter-name, setter-name)
- remove-method (accessor)
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)

apply-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)

<table>
<thead>
<tr>
<th>object of study</th>
<th>refactoring planning approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>purpose</td>
<td>characterisation and evaluation</td>
</tr>
<tr>
<td>focus</td>
<td>effectiveness, efficiency and scalability</td>
</tr>
<tr>
<td>stakeholders</td>
<td>researcher</td>
</tr>
<tr>
<td>context</td>
<td>our reference prototype</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>System</th>
<th>Version</th>
<th>LOC</th>
<th>PEF</th>
<th>FE</th>
<th>DC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jtombstone</td>
<td>1.1.1</td>
<td>1938</td>
<td>32780</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>Groom</td>
<td>1.3</td>
<td>3699</td>
<td>35434</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Lucene</td>
<td>1.9</td>
<td>17627</td>
<td>85064</td>
<td>18</td>
<td>16</td>
</tr>
<tr>
<td>Pounder</td>
<td>0.96</td>
<td>9410</td>
<td>98570</td>
<td>26</td>
<td>3</td>
</tr>
<tr>
<td>MyTelly</td>
<td>1.2</td>
<td>12625</td>
<td>133605</td>
<td>2</td>
<td>13</td>
</tr>
<tr>
<td>Jwebap</td>
<td>0.6.1</td>
<td>16417</td>
<td>141047</td>
<td>17</td>
<td>21</td>
</tr>
<tr>
<td>dbXML</td>
<td>2.0</td>
<td>25862</td>
<td>199400</td>
<td>21</td>
<td>40</td>
</tr>
<tr>
<td>GanttProject</td>
<td>2.0.10</td>
<td>40775</td>
<td>241095</td>
<td>36</td>
<td>24</td>
</tr>
<tr>
<td>JfreeChart</td>
<td>1.0.11</td>
<td>80668</td>
<td>354543</td>
<td>45</td>
<td>29</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>PEFs</th>
<th>Smells</th>
<th>Plans</th>
<th>%</th>
<th>Mean $T_t$</th>
<th>Mean $T_c$</th>
<th>Mean $T_p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>32780</td>
<td>2</td>
<td>1</td>
<td>50</td>
<td>29.52</td>
<td>23.02</td>
</tr>
<tr>
<td>2</td>
<td>35434</td>
<td>2</td>
<td>1</td>
<td>50</td>
<td>30.55</td>
<td>21.57</td>
</tr>
<tr>
<td>3</td>
<td>85064</td>
<td>18</td>
<td>3</td>
<td>16.67</td>
<td>64.1</td>
<td>53.71</td>
</tr>
<tr>
<td>4</td>
<td>98570</td>
<td>26</td>
<td>21</td>
<td>80.77</td>
<td>107.7</td>
<td>103.66</td>
</tr>
<tr>
<td>5</td>
<td>133605</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>253.69</td>
<td>182.85</td>
</tr>
<tr>
<td>6</td>
<td>141047</td>
<td>17</td>
<td>9</td>
<td>52.94</td>
<td>182.03</td>
<td>148.05</td>
</tr>
<tr>
<td>7</td>
<td>199400</td>
<td>21</td>
<td>11</td>
<td>52.38</td>
<td>413.21</td>
<td>303.29</td>
</tr>
<tr>
<td>8</td>
<td>241095</td>
<td>36</td>
<td>13</td>
<td>36.11</td>
<td>544.79</td>
<td>385.46</td>
</tr>
<tr>
<td>9</td>
<td>354543</td>
<td>45</td>
<td>24</td>
<td>53.33</td>
<td>1065.54</td>
<td>960.83</td>
</tr>
<tr>
<td><strong>Totals</strong></td>
<td><strong>169</strong></td>
<td><strong>83</strong></td>
<td></td>
<td><strong>49.11</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Data Class

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>Mean $T_t$</th>
<th>Mean $T_c$</th>
<th>Mean $T_p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>32780</td>
<td>7</td>
<td>6</td>
<td>85.71</td>
<td>26.28</td>
<td>22.5</td>
</tr>
<tr>
<td>2</td>
<td>35434</td>
<td>2</td>
<td>2</td>
<td>100</td>
<td>30.56</td>
<td>21.63</td>
</tr>
<tr>
<td>3</td>
<td>85064</td>
<td>16</td>
<td>16</td>
<td>100</td>
<td>62.96</td>
<td>53.48</td>
</tr>
<tr>
<td>4</td>
<td>98570</td>
<td>3</td>
<td>3</td>
<td>100</td>
<td>179.91</td>
<td>104.28</td>
</tr>
<tr>
<td>5</td>
<td>133605</td>
<td>13</td>
<td>11</td>
<td>84.62</td>
<td>206.36</td>
<td>186.15</td>
</tr>
<tr>
<td>6</td>
<td>141047</td>
<td>21</td>
<td>20</td>
<td>95.24</td>
<td>167.55</td>
<td>148.36</td>
</tr>
<tr>
<td>7</td>
<td>199400</td>
<td>40</td>
<td>40</td>
<td>100</td>
<td>431.47</td>
<td>300.36</td>
</tr>
<tr>
<td>8</td>
<td>241095</td>
<td>27</td>
<td>24</td>
<td>88.89</td>
<td>530.18</td>
<td>386.57</td>
</tr>
<tr>
<td>9</td>
<td>354543</td>
<td>29</td>
<td>23</td>
<td>79.31</td>
<td>1021.41</td>
<td>959.76</td>
</tr>
<tr>
<td><strong>Totals</strong></td>
<td><strong>158</strong></td>
<td><strong>145</strong></td>
<td></td>
<td><strong>91.77</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Time is given in seconds.
Probabilistic Upper Bounds for Planning Time

- Upper bounds for $T_P$ computed with Chebyshev’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