ASSISTING REFACTORING TOOL DEVELOPMENT THROUGH REFACTORING CHARACTERIZATION

Authors:  Raúl Marticorena  rmartico@ubu.es
Carlos López  clopezno@ubu.es
Javier Pérez  jperez@infor.uva.es
Yania Crespo  yania@infor.uva.es
Outline

- Context
- Problem
- Goal
- Refactoring Characterization
- How to use the characterization
- Conclusions and Future Work
Refactoring [Fowler, 2000]

“Process of changing a software system in such a way that it does not alter external behavior of the code yet improve its internal structure”

Example: Add Parameter

| Customer |  |
|----------|--|---|
| getContact() |  |  |

| Customer |  |
|----------|--|---|
| getContact(Date) |  |  |

Open Research Trends

- Define new refactorings
- Identify code defects (*Bad Code Smells*)
- Apply refactorings
  - Tool support
  - Certain language independence
  - etc.
Problem

- **Refactorings**
  - Great number of refactorings
  - e.g. Fowler's catalog as the "standard" catalog
    - Initially in [Fowler,2000]: 68 "medium" refactorings
    - Currently in the web catalog: 93 (and growing!)
      - [http://www.refactoring.com](http://www.refactoring.com)

- **Main task**
  - Build a refactoring tool

- **Questions**
  - How to begin the implementation:
    - What criteria should I use to select them?
  - Reuse building GUI
Assisting Refactoring Tool Development through Refactoring Characterization

Problem

- Refactorings are grouped by some criteria
  - e.g. Functionality (aim) [Fowler,00]
    - Composing methods
    - Moving features between objects
    - Organizing data
    - Etc.
  - Design defects / smells can suggest refactorings
    - Grouped by taxonomies of design defects

- Lack of guidelines:
  - How to face their implementation order
  - How to group refactorings on the basis of common implementation issues
  - Reuse previous efforts
Problem

- **Subjective problem**
  - Same refactorings but different points of view

**1\(^{st}\) Step**

**2\(^{nd}\) Step**

**3\(^{rd}\) Step**

Common agreement?
Goal

1. Simple criteria
2. Avoid subjective criteria
3. From basic observation
4. Low/medium number of features

Selected features

- Design and language issues
- Scope
- Inputs
- Actions
Refactoring Characterization

- **Design and language issues (DLI):** programmers applying the refactoring should know
  - Basic (B)
    - Basic programming concepts
      - e.g. In OOP: classes, inheritance, generics, etc.
  - Advanced (A)
    - Advanced programming concepts
      - e.g. In OOP: exceptions, Design by contract, annotations/attributes, delegates, etc.
  - Design patterns (DP)
    - Well-known patterns
      - e.g. In OOP: Factory Method, Adapter, Command, etc.
Refactoring Characterization

**Scope**: elements affected

- **Intraclass (I)**
  - Do not affect other classes

- **Clients (C)**
  - Client classes suffer its effects

- **Inheritance (H)**
  - Ancestors or descendants are affected
Refactoring Characterization

Inputs:

- Root input
  - Selected item in current code
    - e.g. Class, method, attribute, etc.
  - It determines the available refactoring set:
    - e.g. Method → Rename Method, Move Method, etc.

- Additional inputs
  - Extra information provided by the refactoring user to drive the refactoring execution
  - The greater size of inputs, the more complicated GUI is
Action:

- Select one and just one action that characterizes the refactoring in terms of the changes to the current state of the code

- Small set (from lower to higher complexity):
  - Add → +
  - Rename → n → n'
  - Remove → -
  - Replace → – & + to the same element
  - Move → - & + to different elements

- Although more actions can be identified the main goal is to select the most representative action
How to use the characterization

1st step
- Main features ordered:
  - DLI → Scope → Inputs → Action

2nd step
- Sub-features are also ordered as decreasing complexity:
  - e.g. DLI: DP → Advanced → Basic

3rd step
- Order the refactorings in descending complexity using main features as first criteria and subfeatures as second
**Example:**

- Grouped by Fowler as *Move features between objects*  
  - 8 refactorings in this group  
- Order to face the implementation?

<table>
<thead>
<tr>
<th>Refactoring</th>
<th>DLI</th>
<th>Scope</th>
<th>Root input</th>
<th>Additional Inputs</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hide Delegate</td>
<td>Design Pattern</td>
<td>ICH</td>
<td>1 Class</td>
<td>N Classes</td>
<td>Move</td>
</tr>
<tr>
<td>Remove Middle Man</td>
<td>Design Pattern</td>
<td>ICH</td>
<td>1 Class</td>
<td>N Classes</td>
<td>Remove</td>
</tr>
<tr>
<td>Introduce Local Extension</td>
<td>Basic</td>
<td>ICH</td>
<td>1 Class</td>
<td>1 Class N Methods</td>
<td>Add</td>
</tr>
<tr>
<td>Extract Class</td>
<td>Basic</td>
<td>ICH</td>
<td>1 Class</td>
<td>N Attributes</td>
<td>Move</td>
</tr>
<tr>
<td>Inline Class</td>
<td>Basic</td>
<td>ICH</td>
<td>1 Class</td>
<td>1 Class</td>
<td>Move</td>
</tr>
<tr>
<td>Move Method</td>
<td>Basic</td>
<td>ICH</td>
<td>1 Method</td>
<td>1 Class</td>
<td>Move</td>
</tr>
<tr>
<td>Move Field</td>
<td>Basic</td>
<td>ICH</td>
<td>1 Attribute</td>
<td>1 Class</td>
<td>Move</td>
</tr>
<tr>
<td>Introduce Foreign Method</td>
<td>Basic</td>
<td>IC</td>
<td>1 Class</td>
<td>N Instructions</td>
<td>Add</td>
</tr>
</tbody>
</table>
How to use the characterization

Example:

- Fowler defines a “use” relationship that can be depicted as a graph [Fowler, 2000]
- Graph extracted for this refactoring group:

More fine grained partial order, while in the graph this decision is more subjective...
How to use the characterization

- Building tools with GUI
  - Root inputs
    - Allow to filter the set of available refactorings
    - Help to the user
    - Provide dynamic menus
    - e.g.

- Same GUI
  - Reuse same graphical interface in case of common additional inputs
Conclusions and Future Work

- **Conclusions**
  - Characterization easy to use
  - Helpful to take decisions before beginning refactoring implementation

- **Future work**
  - Validate the characterization with different programmers
    - How each programmer understands one concrete refactoring?
  - Check the characterization with more refactorings
  - Apply this idea to other contexts
    - e.g. Refactoring databases catalog
Thank you very much

Authors:
Raúl Marticorena
Carlos López
Javier Pérez
Yania Crespo

rmartico@ubu.es
clopezno@ubu.es
jperez@infor.uva.es
yania@infor.uva.es