Grupo de Investigación en Reutilización y Orientación a Objeto

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

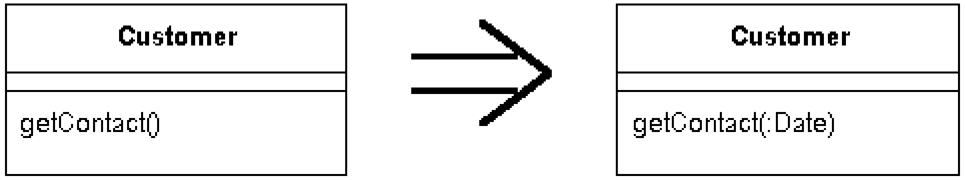
ICSOFT 2011
6th International Conference on Software and Data Technologies
Seville, Spain
18 - 21 July

Outline

- Context
- Problem
- Goal
- Refactoring Characterization
- How to use the characterization
- Conclusions and Future Work

Context

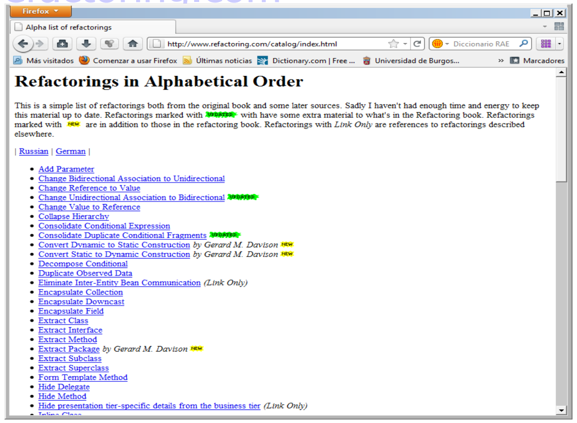
- Refactoring [Fowler, 2000]
  - *"Process of changing a software system in such a way that it does not alter external behavior of the code yet improve its internal structure"*
  - *Example: Add Parameter* *(275)*



- 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



Refactorings in Alphabetical Order

This is a simple list of refactorings both from the original book and some later sources. Sadly I haven't had enough time and energy to keep this material up to date. Refactorings marked with UPDATED with have some extra material to what's in the Refactoring book. Refactorings marked with NEW are in addition to those in the refactoring book. Refactorings with *Link Only* are references to refactorings described elsewhere.

| Russian | German |

- Add Parameter
- Change Bidirectional Association to Unidirectional
- Change Reference to Value
- Change Unidirectional Association to Bidirectional UPDATED
- Change Value to Reference
- Collapse Hierarchy
- Consolidate Conditional Expression
- Consolidate Duplicate Conditional Fragments UPDATED
- Convert Dynamic to Static Construction *by Gerard M. Davison* NEW
- Convert Static to Dynamic Construction *by Gerard M. Davison* NEW
- Decompose Conditional
- Duplicate Observed Data
- Eliminate Inter-Entity Bean Communication *(Link Only)*
- Encapsulate Collection
- Encapsulate Downcast
- Encapsulate Field
- Extract Class
- Extract Interface
- Extract Method
- Extract Package *by Gerard M. Davison* NEW
- Extract Subclass
- Extract Superclass
- Form Template Method
- Hide Delegate
- Hide Method
- Hide presentation tier-specific details from the business tier *(Link Only)*

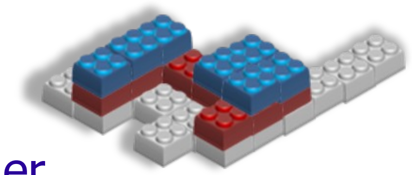
- Main task
  - Build a refactoring tool
- Questions

» How to begin the implementation:
- What criteria should I use to select them?

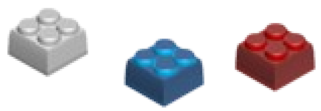
» Reuse building GUI

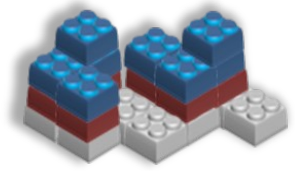
Problem

- 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



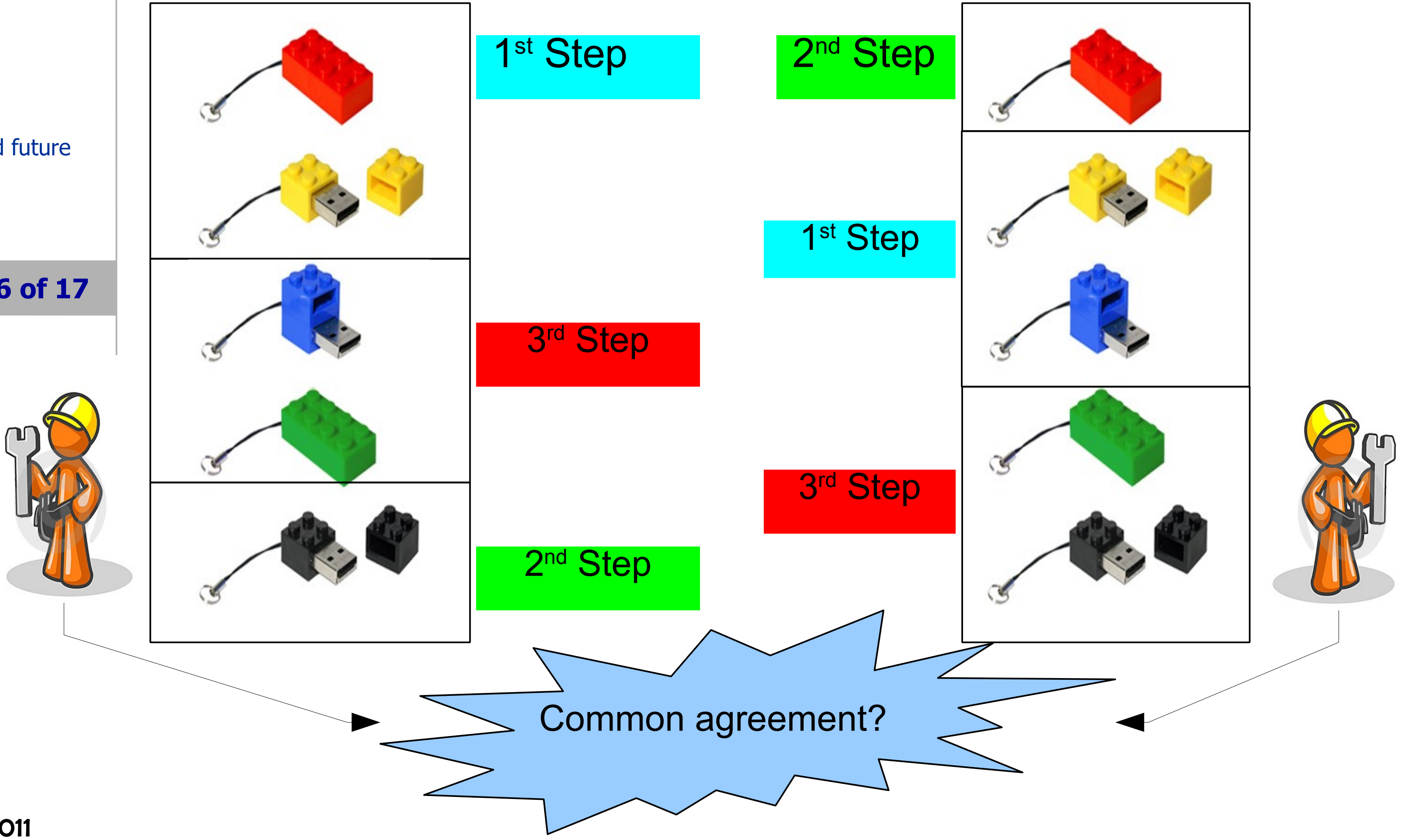
- Lack of guidelines:
  - How to face their implementation order
  - How to group refactorings on the basis of common implementation issues
  - Reuse previous efforts





Problem

- Subjective problem
  - Same refactorings but different points of view



Goal

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

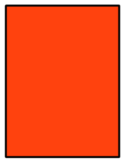
- Selected features
  - Design and language issues
  - Scope
  - Inputs
  - Actions

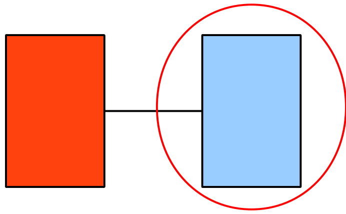
Refactoring Characterization

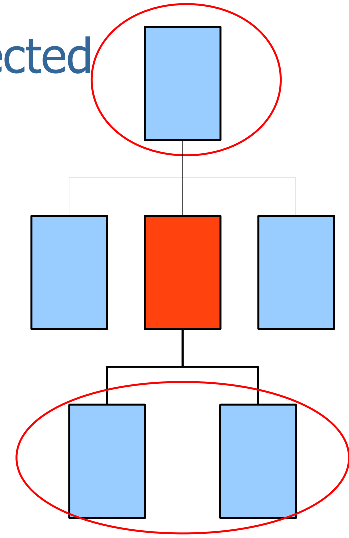
- Design and language issues (DLI): programmers applying the refactoring should know
  - Basic (B)
    - Basic programming concepts
      - e.g. In OOP: classes, inheritance, generics, etc.
  - Advanced (A)
    - Advanced programming concepts
      - e.g. In OOP: exceptions, Design by contract, annotations/attributes, 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

Refactoring Characterization

- Action:
  - Select <u>one and just one action</u> 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

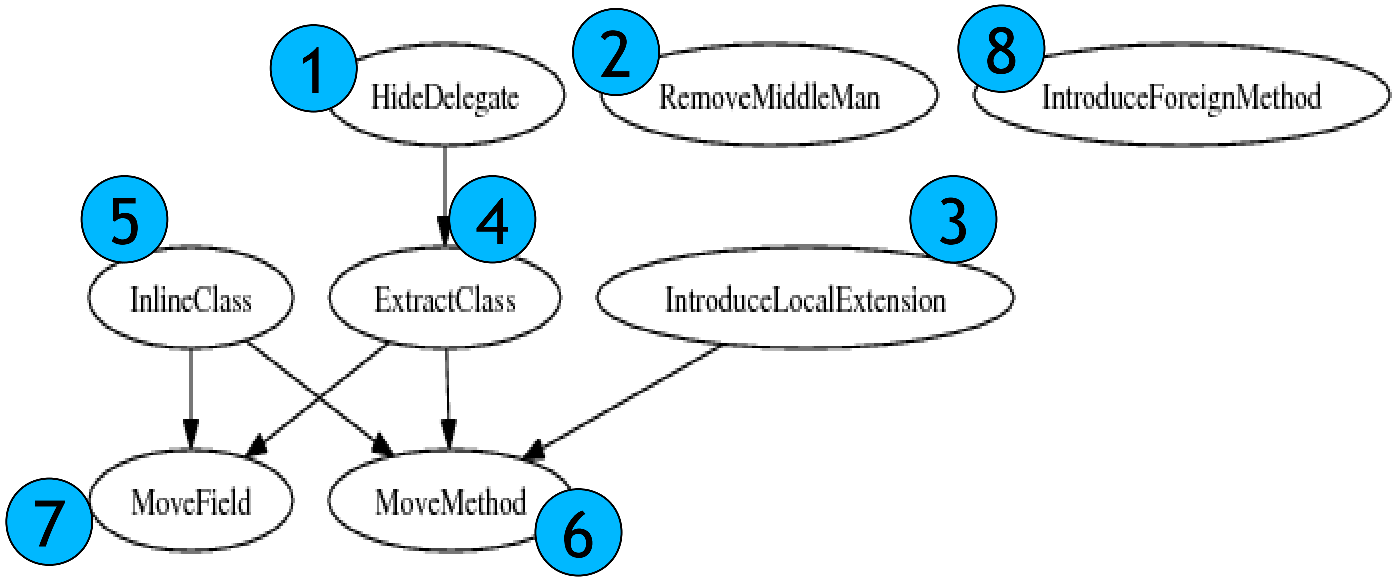
How to use the characterization

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

Refactoring	DLI	Scope	Root input	Additional Inputs	Action
Hide Delegate	Design Pattern	ICH	1 Class	N Clsses	Move
Remove Middle Man	Design Pattern	ICH	1 Class	N Classes	Remove
Introduce Local Extension	Basic	ICH	1 Class	1 Class N Methods	Add
Extract Class	Basic	ICH	1 Class	N Attributes	Move
Inline Class	Basic	ICH	1 Class	1 Class	Move
Move Method	Basic	ICH	1 Method	1 Class	Move
Move Field	Basic	ICH	1 Attribute	1 Class	Move
Introduce Foreign Method	Basic	IC	1 Class	N Instructions	Add

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...

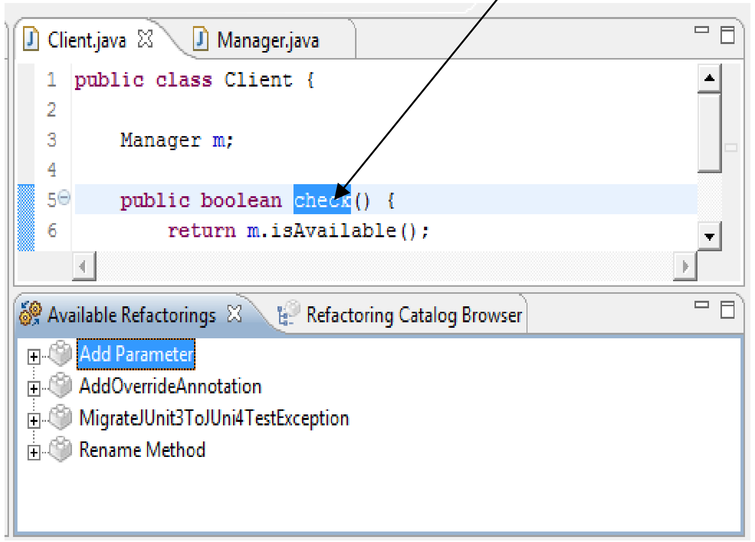
How to use the characterization

- 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

Method root input

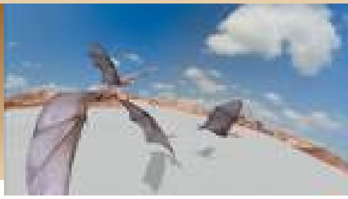
Refactorings with same root input

e.g. Method

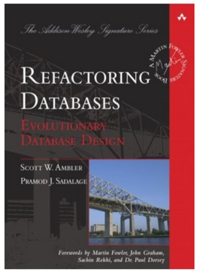


```
public class Client {

    Manager m;

    public boolean check() {
        return m.isAvailable();
```

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



ASSISTING REFACTORING TOOL DEVELOPMENT THROUGH REFACTORING CHARACTERIZATION

Thank you very much

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