Refactoring

Eduardo Figueiredo

http://www.dcc.ufmg.br/~figueiredo

Refactoring

- Refactoring is a change made to the internal structure of software without changing its observable behavior
  - Its goal is to make the software easier to understand and cheaper to modify
- When is refactoring needed?
  - “If it stinks, change it.”

Grandma Beck

Why Should You Refactor?

- To improve design of code
  - Without refactoring, code quality will decay
- To make software easy to understand
  - Refactoring makes your code more readable
- To help you find bugs
  - Easy understand the code, easy to find bugs
- To help you program faster
  - Essential for agile software development

Refactoring Classification

- Composing Methods
- Moving Features Between Objects
- Organizing Data
- Simplifying Conditional Expressions
- Making Method Calls Simpler
- Dealing with Generalization
- Big Refactorings

Composing Methods

Extract Method	Inline Method	Inline Temp
Replace Temp with Query	Introduce Explaining Variable	
Split Temporary Variable	Remove Assignments to Parameters	
Replace Method with Method Object	Substitute Algorithm	

Moving Features Between Objects

Move Method	Move Field
Extract Class	Inline Class
Hide Delegate	Remove Middle Man
Introduce Foreign Method	Introduce Local Extension

Organizing Data

Self Encapsulate Field	Replace Data Value with Object
Change Value to Reference	Change Reference to Value
Replace Array with Object	Duplicate Observed Data
Replace Magic Number with Symbolic Constant	Encapsulate Field
Encapsulate Collection	Replace Record with Data Class
Replace Type Code with Class	Replace Subclass with Fields

Simplifying Conditional Expressions

Decompose Conditional	Consolidate Conditional Expression
Consolidate Duplicate Conditional Fragments	Remove Control Flag
Replace Nested Conditional with Guard Clauses	Replace Conditional with Polymorphism
Introduce Null Object	Introduce Assertion

Making Method Calls Simpler

Rename Method	Add Parameter	Remove Parameter
Separate Query from Modifier	Parameterize Method	
Replace Parameter with Explicit Methods	Preserve Whole Object	
Replace Parameter with Method	Introduce Parameter Object	
Remove Setting Method	Hide Method	
Replace Constructor with Factory Method	Encapsulate Downcast	

Dealing with Generalization

Pull Up Field	Pull Up Method
Push Down Method	Push Down Field
Pull Up Constructor Body	Extract Subclass
Extract Superclass	Extract Interface
Collapse Hierarchy	Form Template Method
Replace Inheritance with Delegation	Replace Delegation with Inheritance

Big Refactorings

- Tease Apart Inheritance
- Convert Procedural Design to Objects
- Separate Domain from Presentation
- Extract Hierarchy

Bibliography

- Martin Fowler. **Refactoring: Improving the Design of Existing Code**. Addison-Wesley Professional, 1st edition, 1999.
  - Chapters 2, 6, 7, 8, 9, 10, 11, and 12