Lecture 13 & 14

Refactoring

What is Refactoring?

- Restructure software in some ways
- Semantics/ functionality / behavior preserving transformation.

Motivation for Refactoring?

- Make it easier to maintain
- Make it easier to comprehend
- Breakdown of modularity --- crosscutting modification (non-localized)
- Painful & tedious to write (to add code for a bug fix / feature addition)
- Generalize / keeping the program OO
- Design conformance
- Performance

Bad Smells of Code

- Duplicated Code
- Long method
- Large class
- Long parameter list
- Divergent change
- ...

- Too many parameters
- Mixed language code
- Non private members
- Declaring the same variables in multiple methods
- Renaming variables --- to explicitly capture the intent of using that variables.

Fowler's Refactoring Book

- A catalogue of refactoring techniques
- Class activity: I will show you several refactoring techniques and you will have a chance to apply refactoring yourself.

Refactoring

- The name/ type of refactoring
- Motivation
- Mechanics (how to apply that refactoring)

Extract Method

- Type: Extract Method---*Turn the fragment into a method whose name explains the purpose of the method*
- Motivation
  - a method is too long
  - what the code does does not match its name (purpose)

Extract Method

- Mechanics
  - Create a new method and name it after its intention
  - Copy the extracted code from the source method into the new target method
  - Scan the extracted code for references to any variables that are local in scope to the source method

Extract Method

- See whether any temporary variables are used only within this extracted code
- See whether any of these local-scope variables are modified by the extracted code; if more than one variable is modified, it is hard.
- Pass into the target methods parameters local-scope variables that are read from the extracted code
- Compile when you have dealt with all the locally-scoped variables
- Replace the extracted code in the source method with a call to the target method
- Compile and test

Class Activity

- Extract Method
- Replace Conditional with Polymorphism
- Extract Superclass

What kinds of difficulties did you face when applying refactoring?

- Redundant edits e.g. declaring overriding methods across similar sibling classes
- Deciding how much you want to generalize - e.g. factoring out print function
- Deciding how much refactoring is enough.
- Both of the two points above all require programmers to anticipate what are likely changes
- Visibility of refactored function.

What kinds of difficulties did you face when applying refactoring?

- Reasons for refactoring in CSW: if there are redundant work going on, you want to identify and merge similar code fragments
- API evolution: foo() -> bar()
- Deciding when to refactor
- Standard refactoring process

What kinds of research is needed in the area of refactoring in your opinion?

- Recommendation systems -- when to refactor
  - amount of accumulated changes over time
  - bad smell, breakdown modularity over time. on-line
- Recommendation systems - what to refactor
  - code coupling, information that are revealed unnecessarily
  - Bad-smell
- Mining refactoring patterns - UMLDiff

What kinds of research is needed in the area of refactoring in your opinion?

- How to refactor
  - Automated update of API evolution [Chow and Notkin 1999, Catch-up Diwan 2005, Z.Xing 2007, M. Robillard 2008]
  - Refactoring support tools in IDEs
- Refactoring detection

What are some reasons to avoid refactoring?

- Breaking other people's code --> sometimes make other people code not to compile
- Reduces program comprehensibility/ hard to trace original design intent
- Increase program complexity
- Unknowing changing behaviors -- break implicit assumptions about code behavior (invariants, precondition, post condition.)
- reduce program performance

On-going Research on Refactoring

- Refactoring Operation Definition [Griswold 1992] [Opdyke 1992]
- Tool Support for Automated Refactoring [Griswold 1992]
- Identifying Refactoring Opportunities (Bad smell detection)

- Automatic update of client component [Chow and Notkin 1999]
- Refactoring-aware version merging [Dig 2007]
- Refactoring reconstruction from program versions [Demeyer et al. 2000, Zou and Godfrey 2005, S.Kim et al. 2005, Xing and Stroulia 2005, Dig et al. 2006, Weissgerber and Deihl 2006, Kim et al. 2007, Fluri et al. 2007, etc.]

Preview for Next Monday

- Refactoring Reconstruction
  - What can you do if you have inferred refactorings?
  - What are challenges of inferring refactorings from two versions?
  - Xing & Stroulia 2005, Fluri et al 2007, Dig et al 2006.
- Crosscutting Concerns
  - Why some code changes require crosscutting changes?