Refactoring:
Improving The Design of Existing Code
(Discussion)

Ruchira Datta

CS 294, Section 1: Software Development
University of California
Berkeley, California

April 10, 2002

What Is Refactoring?

- A particular kind of source-to-source program transformation
- Goal: Simplify and clarify code, *without* changing program's behavior
  - Question: Tree falling in the woods? Is there any point to changes that don't change behavior?
  - Question: "If it ain't broke, don't fix it." My first reaction is "Danger, Will Robinson!" Won't you introduce more bugs by refactoring?

Why Refactor? I

- Refactoring improves design
  - Pro: Very hard to get design right the first time—take advantage of "second system effect"
  - Pro: Modularization is hardest—need to fluidly adapt module boundaries
  - Con: Is this an excuse to postpone design decisions?

Why Refactor? II

- Refactoring Makes Software Easier to Understand
  - Question: But can this scale to large projects?
  - What's easier for me may not be easier for you
- Refactoring Helps You Find Bugs
  - But how do you know you're seeing the whole picture?
  - If not, won't you introduce more bugs?
- Refactoring Helps You Program Faster
  - Is this true? What evidence?

When to Refactor?

- To eliminate triplication
- When adding function
  - A recipe for chaos?
  - Definitely need version control!
- When fixing bugs
- During code review

What to Tell Managers?

- Explain quality orientation
- If ineffective, don't tell
  - Is this professional behavior?
  - How will we ever show that it's economically justified if we never admitted doing it in the first place?

Problems with Refactoring

- Databases
  - The message seems to be: don't
  - Or is it: come up with a refactorable database infrastructure?
- Changing Interfaces
  - Support both
  - Don't publish interfaces
    - The end of modularization?

When Not To Refactor?

- When rewriting is better
  - Most people would rather rewrite, most of the time
  - Coupled with hiding from managers, does this promote bad practice?
- When up against a deadline
  - Refactoring as debt
  - Refactoring: consider code as capital

Other Issues

- Refactoring As Design
  - Can refactoring eliminate Parnas's barnacle-encrusted code
- Strong emphasis on testing to keep from making new bugs
  - Will this actually work?
  - How much time does one spend making the tests in the first place?
  - One needs another book, maybe to explain how to design the tests
- Does refactoring make sense without tool support?
- To what extent is refactoring an excuse to hide flaws in the OO paradigm?