Lecture 23: Design and Refactoring

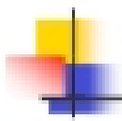
Kenneth M. Anderson
Software Methods and Tools
CSCI 3308 - Fall Semester, 2004

Credit where Credit is Due

- The material for this lecture is based on content from "Refactoring: Improving the Design of Existing Code" by Martin Fowler
- As such, some of this material is copyright © Addison Wesley, 1999

Goals for this lecture

- (Very) Briefly introduce the concept of design
- Introduce Refactoring and cover a few examples



Software Design (I)

- Software design is the process of creating a software system that meets a set of customer requirements
  - Designs require conceptual integrity
- Traditional software design consists of
  - high-level design (architecture, modules)
  - low-level design (interfaces, algorithms)
    - with these two pieces, implementation is often much simpler than it would be if you start coding from scratch

Software Design (II)

- Many different techniques to choose from
  - Structural
    - Stepwise Refinement; "Top Down" vs. "Bottom Up"
    - Abstractions used in design are often different from those used in requirements
    - Typically result in procedural solutions that share data structures; the shared data structures is how modules "communicate"
  - Object-Oriented
    - World consists of objects; Thus, systems should consist of objects that "model" their real-world counterparts
    - Objects appear in all phases (requirements; design; implementation)
    - Typically result in "federations" of objects that work together to achieve system functionality; data and algorithms "live" in objects; communication (data sharing) occurs via "message passing"

Software Design (III)

- Good design requires experience
  - also depends on talent (great designers ala Brooks)
- We can't teach experience (you just have to earn it); we can however teach good design techniques
- Example: Refactoring
  - Useful because its focus is on source code not a specific design notation (so you do not need to learn a new notation to learn this technique)

What is Refactoring

- Refactoring is the process of changing a software system such that
  - the external behavior of the system does not change
    - e.g. functional requirements are maintained
  - but the internal structure of the system is improved
- This is sometimes called
  - "Improving the design after it has been written"

Very Simple Example (I)

- What's wrong with this code?

```
if (isSpecialDeal()) {
    total = price * 0.95;
    send()
} else {
    total = price * 0.98;
    send()
}
```

Answer: Duplicated Code

- The call to send() appears twice, once in the true branch and once in the false branch
- What's wrong with that?
  - In a small project, probably not much
  - But as a project evolves, duplicated code can cause all sorts of problems
    - often associated with "cut and paste" bugs
      - you may copy this code to a context where a call to send() is not appropriate
    - or you may decide to change one branch and forget the other (especially if the code in each branch is long and you can't keep both branches on the screen at once)

Very Simple Example (II)

- How to fix? "Refactor the code"
- We'll use "Consolidate Duplicate Conditional Fragments" and the code becomes

```
if (isSpecialDeal()) {
    total = price * 0.95;
} else {
    total = price * 0.98;
}
send();
```


A Rose is a Rose…

- Why is it so important to give a stuffy sounding name to something so simple?
  - Answer: to improve the "state of practice" in the software development industry
  - As we add standardized vocabulary which all professional developers are required to know; we improve the professionalism of the entire field
  - Refactoring vocabulary is especially important since its improving developer's "design skills"
- Also: some refactorings are NOT simple and giving them a name, makes it easier to discuss the technique with other developers

Benefits of Refactoring

- The idea behind refactoring is to acknowledge that it will be difficult to get a design right the first time
  - and as a program's requirements change, the design may need to change
  - thus, refactoring provides techniques for evolving the design in small incremental steps
- Benefits
  - Often code size is reduced after a refactoring
  - Confusing structures are transformed into simpler structures
    - thus, these new structures are easier to maintain and understand