Procedural vs Object Oriented Design
Bad code smells

Game Design Experience
Professor Jim Whitehead
January 23, 2009
Announcements

- You are expected to be in a project team by now
  - Please see me after class if you are not

- Craig Reynolds talk
  - Crowds and Emergent Teamwork
  - Today, 11am
  - Engineering 2, room 180 (Plaza level, “Simularium”)
  - You are welcome to attend
Upcoming Assignments

- **Monday: Game Concept Document**
  - A compelling document that sells your game concept
  - **Title page**
    - Title of game, name of group, name of team members, sample artwork
  - **Overview page**
    - Table at top: game genre, platform (PC/XBox), team size
    - Key points section
      - Bulleted list of important elements of gameplay
      - Goal of game, what makes game unique, main characters, main fictional elements
      - Sample artwork image to give feel of the game
  - **Biographies**
    - True, pocket biographies of each team member (1-2 paragraphs each) stressing experience that makes you a strong game designer
  - **1-3 pages giving a textual description of the game**
    - Fictional background, brief description of characters, goal of player in game, how does player interact with the game, brief description of levels, game audience, other important elements as needed.
  - **1-2 pages of sample conceptual artwork**
    - Hand-drawn sketches are fine
  - See template and evaluation criteria on course website
Homework #1

- Many of the submissions for homework #1 are
  - Functionally correct
  - Do not exhibit object-oriented design
- In many cases, the design is **procedural**
  - Even though implemented in C#, and OO language

- We were looking for an object-oriented design
- Even the object-oriented designs can have their design improved.
Homework #1 Revisited

• Improve the design of your Hunt the Wumpus
  ► New class assignment, due February 9
  ► If you had a procedural design, must make it object-oriented
    ► If you have an object-oriented design, must improve its structure via code refactoring

• Will add this assignment to website and syllabus over the weekend

• Will give some in-class instruction and readings on code refactoring
Procedural programming paradigm

• Programs are composed of a set of procedures
  ► Also known as functions, methods, etc.
  ► Specify a sequence of statements

• Each procedure specifies a set of steps in the overall computation
  ► Few firm criteria for how to decompose a problem into procedures

• Focus is on determining the steps in the computation, and the sequence of those steps

• Some data is passed as parameters, much data is left as global variables
  ► Data, and the procedures that operate on them, may be in different text files
Object-oriented programming paradigm

- Programs are composed of objects
- An object is composed of
  - Data
  - Methods that act on that data
- An object is expected to correspond to a significant abstraction in the application
- It is possible to specialize objects, via inheritance
  - E.g., square can inherit from the more generic shape
- Class/object distinction
  - Specification of a class, vs. instances of a class
Ways Procedural Designs are Expressed in C#

- Pure procedural
  - All methods are static, and found in the same class as main(). There is only one class.
  - Class variables act as global variables.

- Pure procedural with data abstraction
  - All methods are static, and found in the same class as main(). There are multiple classes, but other classes only have member variables, no methods (except a constructor).
  - Object-orientation only used to cluster related global variables together

- Procedural, with minor object-orientation
  - Design is strongly procedural, but classes have a small number of methods defined on them

- In the vast majority of cases (and until you have substantial experience as a software designer), you should avoid use of procedural design in object-oriented software.

- Demonstration of some of these from homework submissions
Bad Code Smells

• Once code has an object-oriented design, can then focus on improving its design
  ▶ If the design is procedural, can’t even begin to do this

• Refactoring literature has notion of “code smells”
  ▶ “If it stinks, change it” (M. Fowler, Refactoring)
  ▶ A characteristic of a design that is a strong indicator it has poor structure, and should be refactored
  ▶ Code smells are rules of thumb
    • It’s not always straightforward that a bad smell must lead to a refactoring. Have to use judgement.
    • Still, as new designers, bad code smells likely mean you should change your code.
Code Smells: Duplicated Code

- Duplicated code (code clones)
  - The same, or very similar code, appears in many places
  - Problem
    - A bug fix in one code clone may not be propagated to all
    - Makes code larger than it needs to be
  - Example from homework
    - Adjacency checks to print warnings (“I smell a Wumpus”, etc.)
  - Fix: *extract method* refactoring
    - Create new method that encapsulates duplicated code
    - Replace code clones with method call
Code smells: Long Method

• Long method
  ▶ A method that has too many lines of code
    • How long is too long? Depends.
    • Over 20 is usually a bad sign. Under 10 lines is typically good.
    • Still, no hard and fast rules.
  ▶ Problem
    • The longer a method, the harder it is to understand, change, and reuse
  ▶ Example from homework
    • Shooting logic, Main
  ▶ Fix: extract method
    • Take chunks of code from inside long method, and make a new method
    • Call new method inside the now-not-so-long method.
Code smells: Feature Envy

• Feature Envy
  ► A method in one class uses primarily data and methods from another class to perform its work
    • Seems “envious” of the capabilities of the other class
  ► Problem:
    • Indicates abstraction fault.
    • Ideally want data, and actions on that data, to live in the same class.
    • Feature Envy indicates the method was incorrectly placed in the wrong class
  ► Fix:
    • Move method
      – Move the method with feature envy to the class containing the most frequently used methods and data items
Code smells: Large class

- Large class
  - A class is trying to do too much
  - Many instance variables
  - Many methods
- Problem:
  - Indicates abstraction fault
    - There is likely more than one concern embedded in the code
    - Or, some methods belong on other classes
  - Associated with duplicated code
- Fix:
  - *Extract class* refactoring
    - Take a subset of the instance variables and methods and create a new class with them
    - This makes the initial (long) class shorter
  - *Move method* refactoring
    - Move one or more methods to other classes
- Example from homework:
  - Class containing Main() tends to have too much game logic
Code smells: switch statements

- Switch statements
  - The cases in a switch statement contain logic for different types of instances of the same class
  - In object-oriented code, this indicates new subclasses should be created

- Problem
  - The same switch/case structure appears in many places

- Fix
  - Create new subclasses
  - Extract method to move case block logic into methods on the new subclasses
Code smells: Data class

• Data class
  ► A class that has only class variables, getter/setter methods/properties, and nothing else
  ► Is just acting as a data holder

• Problem
  ► Typically, other classes have methods with feature envy
  ► That is, there are usually other methods that primarily manipulate data in the data class
  ► Indicates these methods should really be on the data class
  ► Can indicate the design is really procedural

• Fix
  ► Examine methods that use data in the data class, and use move method refactoring to shift methods
Homework

• Read Chapter 1 (Refactoring, a First Example) in Martin Fowler, *Refactoring* book
  ▶ Will post link on forum, and put link in syllabus on website

• Examine your Wumpus source code for code smells