Bad code smells
Refactoring Example

Game Design Experience
Professor Jim Whitehead
January 26, 2009
Announcements

• Game concept document due today
  ► Please let us know immediately if your partner has dropped the class
  ► Or, did not participate in the assignment (you couldn’t reach them, for example)

• Bob Mitchell talk
  ► CTO of Sony Online
  ► Today, 2pm, speaking to CS 80K
  ► Media Theater
  ► You are welcome to attend
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 had gotos, you must remove them
    • Except for use in case statements, and even here they are strongly discouraged
  ► If you have an object-oriented design, must improve its structure via code refactoring
    • At least four refactorings to fix at least two different types of code smells
    • Must mark up original code with location of bad code smells, and indicate type of refactoring
  ► See lecture notes from Friday on difference between procedural and object-oriented code
    • Ask in section for help on assessing your own design
Homework #1 Revisited

• Submission instructions
  • First HW#1 is Procedural (or non-OO) design
    ▶ Submit original code printout
    ▶ Submit printout of entire new program
    ▶ Submit all new code electronically
  • First HW#1 is Object-Oriented design
    ▶ Submit original code printout marked up with location of bad code smells, and refactorings
    ▶ Submit printout of only changed code in new version
    ▶ Submit all new code electronically
• Details on web site
Goto Statement

• We were surprised to receive a number of homework #1 submissions containing goto statements
  ▶ Possibly due to access of BASIC source code for Wumpus
  ▶ Possibly due to C# having a goto construct that can be used for case statements

• Use of goto statement (outside of case) is unacceptable
  ▶ Core problem: with goto, control flow within a method does not even remotely follow the linear arrangement of statements on a page
  ▶ Leads to code that is substantially more difficult to understand, debug, and get correct
  ▶ Edsger Dijkstra's letter, Go To Statement Considered Harmful, published in the March 1968 Communications of the ACM (CACM)
    • Major milestone in move towards structured programming
  ▶ In the year 2009, use of goto in professional settings leads to immediate (and perhaps irrevocable) loss of respect from peers, and casts tremendous doubt over your ability to program. Depending on the setting, could lead to immediate termination.
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
Refactoring Example

- Walk through refactoring example from Chapter 3 of Martin Fowler, Refactoring.
Homework

• Read Chapter 3 of *Refactoring*, by Martin Fowler
  ► Username: cs116
  ► Password: cs116
  ► Same for all chapters