Anti-Patterns

IOWA STATE UNIVERSITY

Com S 362: Object-Oriented Analysis and Design

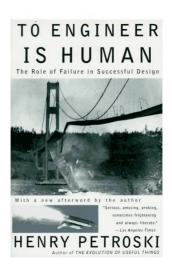
Recap: Refactoring

- Improve the structure of code
 - No value gain at the moment, but
 - Easier to add features later
 - Less chances of errors in maintenance tasks
- Key is to preserve semantics
 - Imprecisely ensure by developing tests
 - Also, by code inspection
- Often automated support for common refactorings
 - Automated support, less error prone
 - Often most general case
 - e.g. Eclipse extract method makes all variables parameters
 - Limitation of current program analysis techniques

2

IOWA STATE UNIVERSITY

Anti-Patterns



Lessons Learned from failures and their remedies.

AntiPatterns: Vaccinations against Object Misuse" [Akroyd 96]

IOWA STATE UNIVERSITY

Com S 362: Object-Oriented Analysis and Design

Example: Spaghetti Code

- An undocumented piece of source code
- Cannot be extended or modified
 - ■Reason: convoluted structure
 - ■Effect: significant cost in modification

4

IOWA STATE UNIVERSITY

Symptoms

- Quick demonstration code integrated in the running system
- Obsolete or scanty documentation
- 50% time spent learning what the code does
- "Hesitant programmer syndrome"
 - Perhaps easier to rewrite this code
 - More likely to break it then extend it
- Cannot be reused
 - Cannot change the used library/components
 - Cannot optimize performance
- Duplication
 - "I don't know what that piece of code was doing, so I rewrote what I thought should happen, but I cannot remove the redundant code because it breaks the system."

5

IOWA STATE UNIVERSITY

Com S 362: Object-Oriented Analysis and Design

Symptoms in an OO program

- Many OO method with no parameters
- Suspicious class or global variable
- Strange relationships between classes
- Process-oriented methods
 - ■Objects with process-oriented names
- ■OO advantage lost
 - Inheritance cannot be used to extend
 - ■Polymorphism cannot be used

6

IOWA STATE UNIVERSITY

Net Results

- Reached point of diminishing returns
- Efforts to maintain >> Efforts to redevelop

IOWA STATE UNIVERSITY

Com S 362: Object-Oriented Analysis and Design

Solution

• Refactor to generalize: Create an abstract superclass

- 1. Make subclass function signatures compatible
- 2. Add function signatures to the superclass
- 3. Make function bodies and variables compatible
- 4. Migrate common code to the superclass

• Refactor to specialize: Simplify conditionals

- 1. For each condition, create a subclass with matching invariant
- 2. Copy the code into the subclass
- 3. Simplify code based upon invariant
- 4. Specialize the superclass constructor

Refactor to combine: Capture aggregations and components

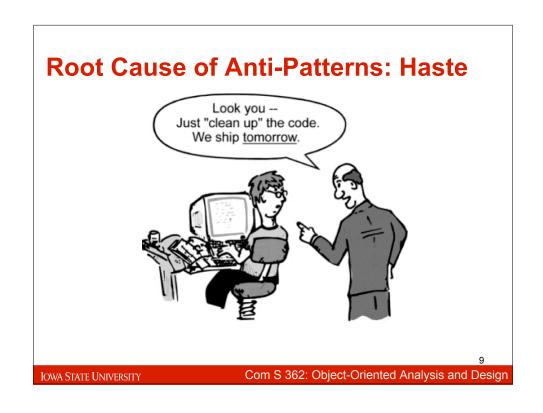
Type A. Move members from an aggregate class to a components class

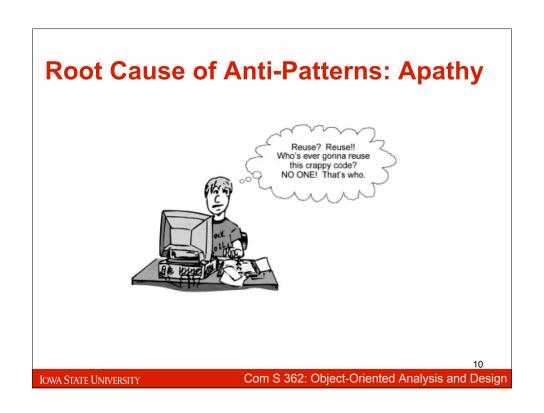
Type B. Move members from component classes to aggregate class

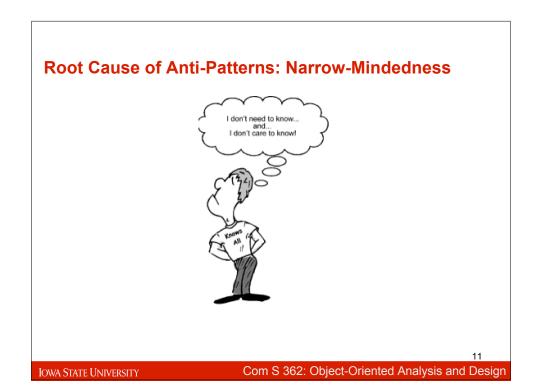
Type C. Convert inheritance into an aggregation

3

IOWA STATE UNIVERSITY







Root Cause of Anti-Patterns: Sloth



IOWA STATE UNIVERSITY

Root Cause of Anti-Patterns: Avarice Well...R certainly is complicated! I'm sure our clients will be very, very impressed! Well done, Jenkins!

IOWA STATE UNIVERSITY

13

Com S 362: Object-Oriented Analysis and Design

Root Cause of Anti-Patterns: Ignorance

Look, Roy... I don't really have time to actually read this...
Could you put together a one-page Executive Summary...

Dummy...

14

IOWA STATE UNIVERSITY

Com S 362: Object-Oriented Analysis and Design

Root Cause of Anti-Patterns: Pride



IOWA STATE UNIVERSITY

Com S 362: Object-Oriented Analysis and Design

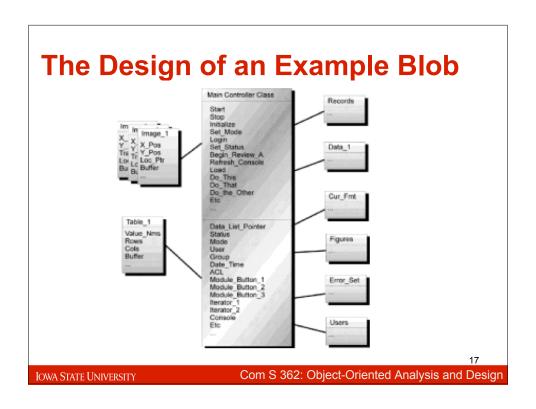
Another Example: The BLOB

- Also known as
 - ■Winnebago and the God class
- Scale: Entire application

- ■General Form:
 - ■One class monopolizes the processing
 - ■Other classes are data classes

16

IOWA STATE UNIVERSITY



Symptoms of a Blob

- Single Class
 - ■Large number of attributes
 - ■Large number of operations
- Unrelated attributes and operations
 - ■Overall lack of cohesiveness
- ■Too complex to reuse and test
- Expensive to load into memory
 - ■Exercise: Explain why?

18

IOWA STATE UNIVERSITY

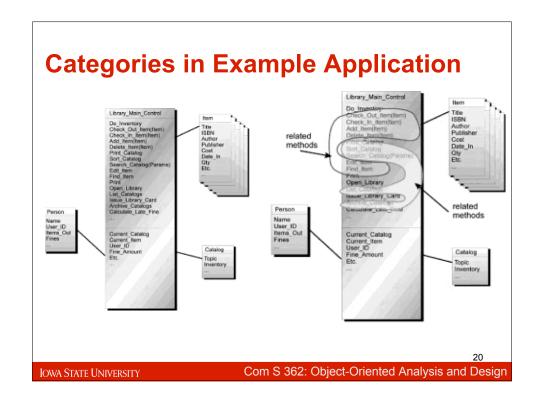
Refactored Solution

- Identify or categorize related things
 - ■Attributes, Operations
- ■Where do these categories naturally belong?
 - Apply move method, move field refactorings
- Remove redundant associations

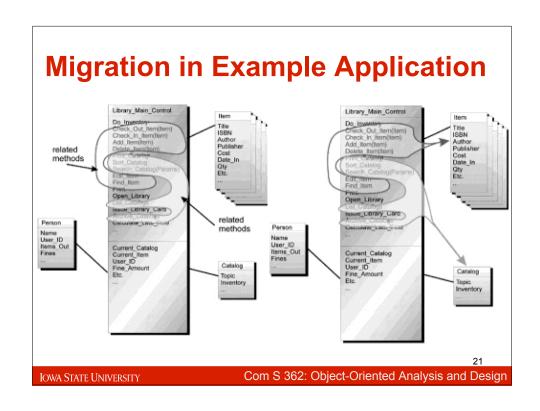
19

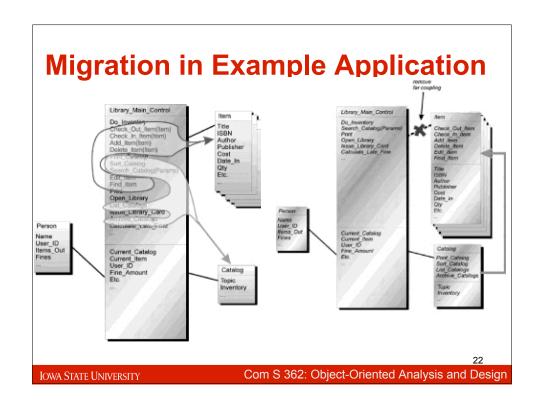
IOWA STATE UNIVERSITY

Com S 362: Object-Oriented Analysis and Design



10

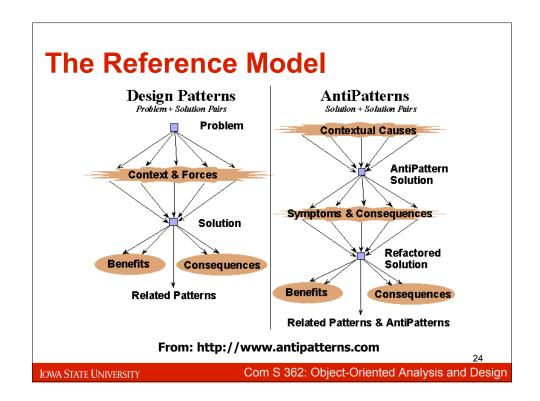




Why Study AntiPatterns?

- Provide a method of efficiently mapping a general situation to a specific class of solutions
- Provide real world experience in recognizing recurring problems in the software industry
- Provide a common vocabulary for identifying problems and discussing solutions.

IOWA STATE UNIVERSITY



Anti-Patterns

- Describes:
 - Commonly occurring solution to a problem
 - Solution often leads to negative consequences
- Results from ignorance, lack of experience, applying good patterns to wrong context, etc
- Purpose of cataloguing:
 - Recognize
 - Remedy, often by refactoring

25

IOWA STATE UNIVERSITY

Com S 362: Object-Oriented Analysis and Design

Describing an Anti-Pattern

- General Form
- Symptoms to recognize general form
 - How to identify
 - Example: One big class, a lot unrelated methods
 - Example: Many methods with no arguments
- Causes that lead to the general form
 - lack of design experience
- Refactored solution:
 - How to change into a healthier solution
 - Split into smaller classes
 - Identify or categorize attributes and operations

26

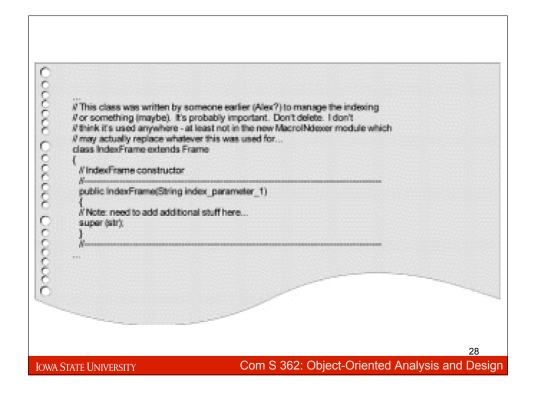
IOWA STATE UNIVERSITY

Mini Anti-Pattern: Lava Flow

- Also Known As: Dead Code
- Scale: Application
- Refactored Solution Name: Architectural Configuration Management
- Refactored Solution Type: Process
- Root Causes: Avarice, Greed, Sloth
- Unbalanced Forces: Management of Functionality, Performance, Complexity
- Anecdotal Evidence: "Oh that! Well Ray and Emil (they're no longer with the company) wrote that routine back when Jim (who left last month) was trying a workaround for Irene's input processing code (she's in another department now, too). I don't think it's used anywhere now, but I'm not really sure. Irene didn't really document it very clearly, so we figured we would just leave well enough alone for now. After all, the bloomin' thing works doesn't it?!"

IOWA STATE UNIVERSITY

Com S 362: Object-Oriented Analysis and Design



14

Poor Design

- Expensive to analyze, verify, and test. All such effort is expended entirely in vain and is an absolute waste. In practice, verification and test are rarely possible.
- Expensive to load into memory, wasting important resources and impacting performance.
- Many of the inherent advantages of an objectoriented design lost. In this case, you lose the ability to leverage modularization and reuse without further proliferating the Lava Flow globules.

29

IOWA STATE UNIVERSITY

Com S 362: Object-Oriented Analysis and Design

Symptoms of Lava Flow

- Frequent unjustifiable variables and code fragments
- Undocumented complex code segments
 - ■important-looking functions, classes,
 - ■These segments don't clearly relate to the system architecture.
- Very loose, "evolving" system architecture.
- Whole blocks of commented-out code with no explanation or documentation.
- Lots of "in flux" or "to be replaced" code areas.

30

IOWA STATE UNIVERSITY

Symptoms of Lava Flow

- Unused (dead) code, just left in.
- Unused, inexplicable, or obsolete interfaces
- If existing Lava Flow code is not removed, it can continue to proliferate as code is reused in other areas.
- If the process that leads to Lava Flow is not checked, there can be exponential growth as succeeding developers, too rushed or intimidated to analyze the original flows, continue to produce new, secondary flows as they try to work around the original ones, this compounds the problem.
- As the flows compound and harden, it rapidly becomes impossible to document the code or understand its architecture enough to make improvements.

31

IOWA STATE UNIVERSITY

Com S 362: Object-Oriented Analysis and Design

Cause

- R&D code placed into production [no configuration management]
- Uncontrolled distribution of unfinished code. Implementation of several trial approaches toward implementing some functionality. Often single-developer (lone wolf) written code.
- Lack of architecture
- Repetitive development process
 - Goals not clear
 - Design decisions not hidden
 - Rework, backtrack, and develop prototypes
 - Hasty changes, no refactoring
- Architectural scars
- Too costly to analyze the existing code base

32

IOWA STATE UNIVERSITY

How to fix it?

Exercise:

33

IOWA STATE UNIVERSITY

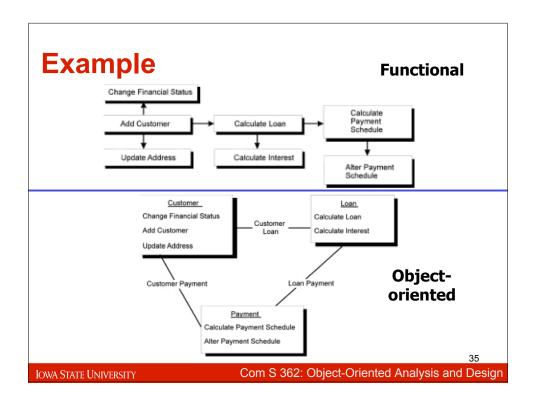
Com S 362: Object-Oriented Analysis and Design

Functional Decomposition: Exercise

- Also Known As: No Object-Oriented AntiPattern "No OO" [Akroyd 96]
- Most Frequent Scale: Application
- Refactored Solution Name: Object-Oriented Reengineering
- Refactored Solution Type: Process
- Root Causes: Avarice, Greed, Sloth
- Unbalanced Forces: Management of Complexity, Change
- Anecdotal Evidence: "This is our 'main' routine, here in the class called LISTENER."

34

IOWA STATE UNIVERSITY



Boat Anchor

- Piece of software or hardware that serves no useful purpose on the current project
- Often a costly acquisition, which makes the purchase even more ironic
- At acquisition pitch to "decision makers"
- No technical evaluation of the product
- Significant effort to make it work
- After efforts found to be useless

36

IOWA STATE UNIVERSITY

