SWEN 262
Engineering of Software Subsystems

Anti-Patterns
References

- An anti pattern is a common response to a recurring problem that is usually ineffective and risks being highly counterproductive.

- **Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis**
  - Brown, Malveau, McCormick III, & Mowbray
  - John Wiley & Sons, 1998

- **Software Anti-Patterns Cheat Sheet**
Anti Patterns

● A pattern of practice that is commonly found in use.
  ○ When practiced, anti patterns often result in negative consequences.

● A software engineer must develop and implement strategies to repair and remove anti patterns when they are encountered.
  ○ Solve through safely refactoring the code.
  ○ Work incrementally.
  ○ There are many alternatives to consider.
  ○ Beware of the cure being worse than the disease.

It is worth remembering that refactoring takes time (and money) and risks breaking code that works.

Keep this in mind when considering removing an antipattern.
The Blob

- Anecdotal Evidence:
  - Classes with names like “System,” “Manager,” or “Controller.”
  - Lots of little data classes.
  - UML class diagram that is larger than all of the other classes.

- Possible Causes:
  - “Just put it in main.”
  - Information Expert taken to the extreme.
  - No Pure Fabrication.

- Problems:
  - Too complex to test.
  - No hope for reuse.
  - Low cohesion.
  - High coupling.

WANTED

THE BLOB

A.K.A. The Winnebago, God Class, The Kitchen Sink

Like the blob in the movie, starts out small and grows over time. “This class is the heart of our system.”

If found: Categorize related attributes and operations, **extract class**. Apply **information expert** to data classes.
Copy-and-Paste Programming

- **Anecdotal Evidence:**
  - “I thought you already fixed this bug?”

- **Possible Causes:**
  - People unfamiliar with tools or technology copy and modify a working example.
  - Laziness/time pressure.
  - Low coupling and/or Law of Demeter taken to the extreme.

- **Problems:**
  - Code duplication (DRY!).
  - Same bugs occur multiple times.
  - Information expert broken.
  - Low cohesion.
  - Higher maintenance costs.

“Man, you guys work fast! Over 400,000 lines of code written in three weeks!”

If found: Apply **information expert** and **pure fabrication** to create a single, standard implementation. Replace duplication with method calls.
Lava Flow

- **Anecdotal Evidence:**
  - IDE flags unused code.
  - No one is sure what a method or class does, but is afraid to remove it.
  - Large blocks of commented code.

- **Possible Causes:**
  - Lack of confidence in refactored implementations (old code kept “just in case”).
  - Change in development team members.
  - Research code moved into production.
  - Time.

- **Problems:**
  - Classes with no relationships.
  - Code glut/increased lines of code.
  - Unused methods with duplicate behavior.

If found: Write characterization tests, then slowly remove dead code. Rerun tests to make sure that nothing is broken. Moving forward, safely refactor code (tests). 

Code, like lava, is fluid when it starts life but becomes hard and immovable later.
Poltergeists

- Anecdotal Evidence:
  - Transient associations.
  - Short-lived, stateless classes.
  - Classes created to invoke methods in other classes, then go out of scope.

- Possible Causes:
  - Using OO when it is not appropriate.
  - Poorly implemented Command Pattern.
  - Poorly implemented MVC.

- Problems:
  - Higher coupling.
  - Lots of little, extra classes.

If found: Remove poltergeists by moving controlling actions to related classes.
The Golden Hammer

- **Anecdotal Evidence:**
  - “Our database drives our architecture.”
  - “The entire program is implemented with Excel macros!”
  - Identical tools used for conceptually different problems.

- **Possible Causes:**
  - “The customer asked us to use .net, but I already know Python, so...”
  - Lack of training, outdated skills.
  - No diversity of background.

- **Problems:**
  - Tools determine the architecture.
  - Non-functional requirements ignored.
  - Customer/Product Owner ignored.
  - Poor performance & scalability.

---

WANTED

THE GOLDEN HAMMER

A.K.A. Old Yeller

When the only tool you have is a hammer, everything starts to look like a nail.

**If found:** Diversify team skills. Commit to professional development. Explore new technologies. Use open systems and architectures.
Functional Decomposition

- **Anecdotal Evidence:**
  - Classes have names that sound like methods, e.g. CalculateInterest, DisplayTable.
  - Classes with a single method.

- **Possible Causes:**
  - High cohesion taken to the extreme.
  - Lack of object oriented experience.

- **Problems:**
  - No hope of reuse.
  - Class explosion.
  - High coupling.

“Object oriented programming is easy! You just put everything in a class!”

**If found:** Perform domain analysis, use domain driven design. Apply information expert. Combine classes with related state/behavior. Use static functions when appropriate.