“AntiPatterns”
What are AntiPatterns?

- Recall Design Patterns.

- AntiPatterns are observable phenomena that are signs of development problems.

- The purpose of cataloguing antiPatterns is so they can be recognized and remedied (typically by some form of refactoring).
Representative Sources

*Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis*

- William H. Brown
- Raphael C. Maloueu
- Hays W. "Skip" McCormick III
- Thomas J. Hovbrig

*AntiPatterns and Patterns in Software Configuration Management*

- William J. Brown
- Hays W. "Skip" McCormick III
- Scott W. Thomas
AntiPatterns

- Begin with a **problematic attempt** to solve a problem.

- Abstract **symptoms** and consequences, similar to the context of a design pattern.

- Once identified, the antiPattern’s **refactored solution** can be used to resolve or lessen the problem.
“refactored” is a code word meaning:

“changed for the better”,

“improved”,

“re-engineered”
Patterns vs. AntiPatterns

Patterns:
- Recurrent Problems
- Solution (Pattern)

Anti-Patterns:
- Recurrent Problems
- "Solution" introducing problems of its own (Anti-Pattern)
- Refactored Solution
The preceding is my own refactoring of this “cloud diagram”
Software vs. Software Development

- When authors say “software” anti-Patterns they often mean “software development” antiPatterns.

- Software can have its own antiPatterns, also referred to as “design flaws”:
  - fascistic features
  - gratuitous sound effects
  - unnecessary font transformations
AntiPattern Categories

- Software Development/Architecture
- Project Management
- Project Team
**AntiPattern: Spaghetti Code**

- **Scale**: Application
- **Symptoms**:
  - Single body of code supporting > 1 function
  - Easier to rewrite code than modify
  - Lack of documentation
- **Cause**: sloth, ignorance, time pressure
- **Refactored Solution**: Code cleanup, code factoring, higher-order functions, object hierarchy
AntiPattern: The Blob
(aka “God Class”)

- **Scale**: Application
- **Symptoms**:
  - One big class, hundreds of unrelated methods
  - Many methods with no arguments
- **Cause**: lack of design experience
- **Refactored Solution**: Split into smaller classes, avoid transitive associations
- **Similar to**: Swiss army knife, kitchen sink
AntiPattern: Poltergeists

- **Scale**: Application
- **Symptoms**:  
  - Lots of small, non-descript, classes  
  - Classes have limited use  
  - Classes have overlapping uses
- **Cause**: lack of design
- **Refactored Solution**: Create fewer and more coherent classes
AntiPattern: Cut-and-Paste

- **Scale:** Application

- **Symptoms:**
  - Over 10000 lines of code produced in a week
  - Having to make multiple identical edits to correct a single problem

- **Cause:** sloth, ignorance, time pressure

- **Refactored Solution:** Procedures, macros, methods, higher-order functions
AntiPattern: Input Kludge

- **Scale:** Application
- **Symptoms:**
  - Software fails on straightforward input tests
- **Cause:** sloth, ignorance, time pressure
- **Refactored Solution:** Construct a proper parser and error-check the input.
AntiPattern: Lava Flow
(dead code and forgotten design)

- **Scale:** Application
- **Symptoms:**
  - Code that nobody understands
  - Undocumented design
  - Code’s author long since departed
  - Code that can’t be tested
- **Cause:** failure to use revision control system
- **Refactored Solution:** Use revision control, get rid of dead code, redesign
AntiPattern: Stovepipe System
(irregular system parts hooked together)

- **Scale**: Application
- **Symptoms**:
  - Inordinately large system
  - Many components with similar functions
  - Many different interfaces
- **Cause**: too much reliance on pre-existing components
- **Refactored Solution**:
  - Use abstraction to coalesce components.
  - Use layered architecture, use common interfaces.
AntiPattern: Vendor Lock-in

- **Scale**: Application
- **Symptoms**:
  - “Our architecture is DCOM” (or whatever) (which is code for “We don’t have an architecture.”)
  - Missing features, because vendor doesn’t support them
  - Difficult to upgrade product
- **Cause**: excessive reliance on a vendor
- **Refactored Solution**: layer that isolates vendor-specific modules from other code
AntiPattern: Golden Hammer

- **Scale**: Application

- **Symptoms**:
  - Contrived code
  - Database driving the architecture
  - Unusual language choices (Hypertalk, Excel macros, XML)

- **Cause**: using one tool for everything

- **Refactored Solution**: find tools best suited to the problem
**AntiPattern: Reinventing the Wheel**

- **Scale:** Application
- **Symptoms:**
  - “Our problem is unique”
  - “Not invented here” (N.I.H. factor)
- **Cause:** failure to use the work of others or buy available components
- **Refactored Solution:** Use existing patterns and components
Some Project Management AntiPatterns

- Design by Committee
- Analysis Paralysis
- Death by Planning
- Viewgraph Engineering
- Corncob
- Death March Projects
- Irrational Management
- Throw it over the wall
- Fire Drill
Some Observed Software Team Anti-Patterns

- **Reinvent the wheel**: Ignore existing patterns.
- **Out-to-lunch**: Team member doesn’t read email or voice-mail (Particularly bad if it’s the leader.)
- **Things-to-do, places-to-go, ...**: Team member doesn’t attend meetings.
- **Golden hammer**: Team member too enamored with using specific tool, forgets about the main problem.
- **Client starvation**: Team does not interact with client, ends up with unusable product.
- **I’d rather do it myself**: Team doesn’t ask for advice on difficult issue until it’s too late.
- **Blackhole**: Team gets committed to an unusable software library.
- **GroupThink**: Fear of expressing individual views outside of consensus.