ANTIPATTERNS

Ingeniería de Software I

Agenda

- Antipatterns
  - Definición y Fundamentos
  - Template
  - Grupos
    - en Desarrollo
    - en Arquitectura
    - en Gerenciamiento

Well-known AntiPatterns

AntiPatterns are all around us. They're often used as tools for social control.

SOCIAL AntiPatterns

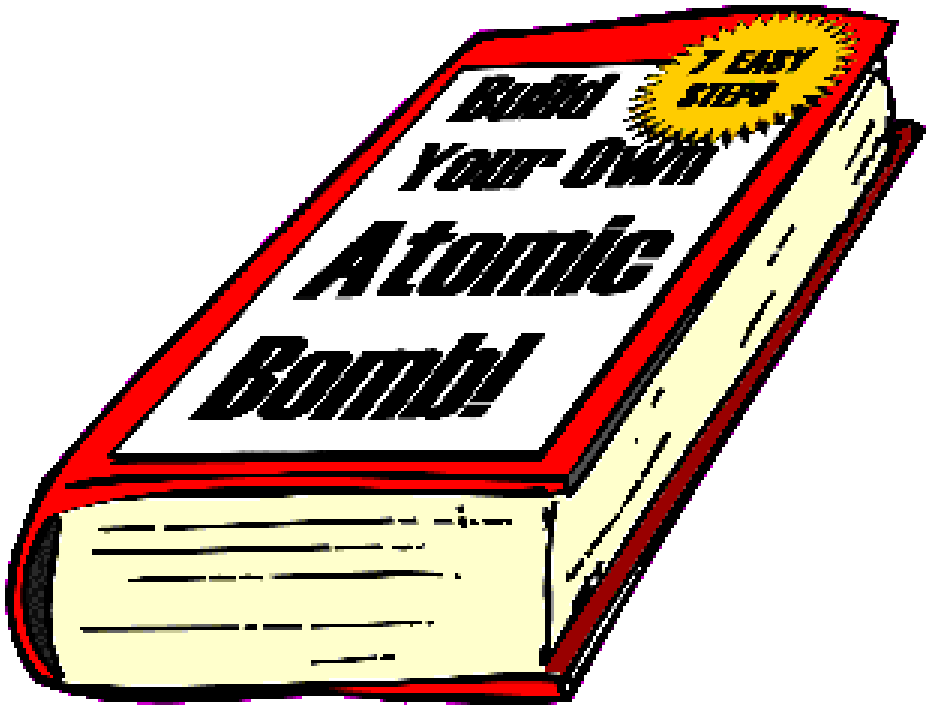
- Criminal
- Terrorist
- Pervert
- Drug Addict/Pusher
- Heretic/Witch
- Dilbert's Pointy-Haired Manager

SOFTWARE AntiPatterns

- Spaghetti Code
- Stovepipe System
- Analysis Paralysis
- Design By Committee
- God Class
- Mythical Man Month
- Death March Project

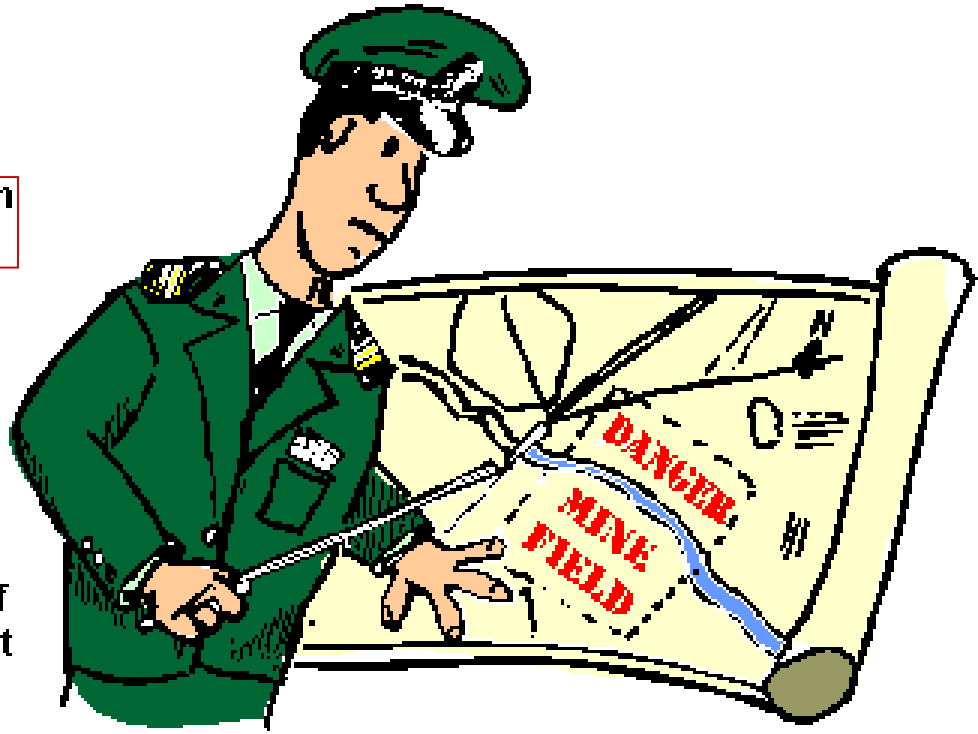
What is an AntiPattern (and why should I care)?

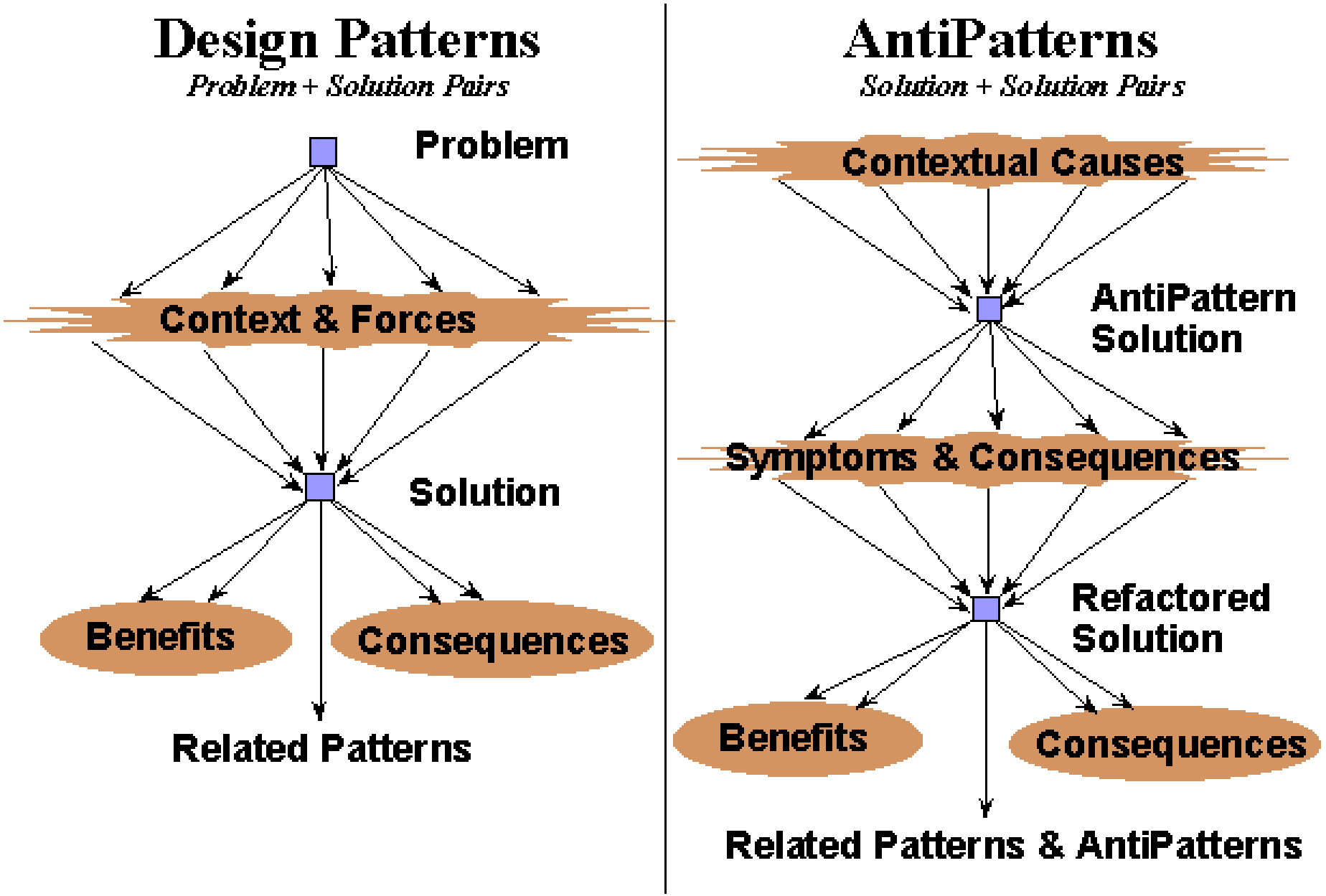
- AntiPatterns are ***Negative*** Solutions that present more problems than they address
- AntiPatterns are a natural extension to design patterns
- AntiPatterns bridge the gap between architectural concepts and real-world implementations.
- Understanding AntiPatterns provides the knowledge to prevent or recover from them



Why Study AntiPatterns?

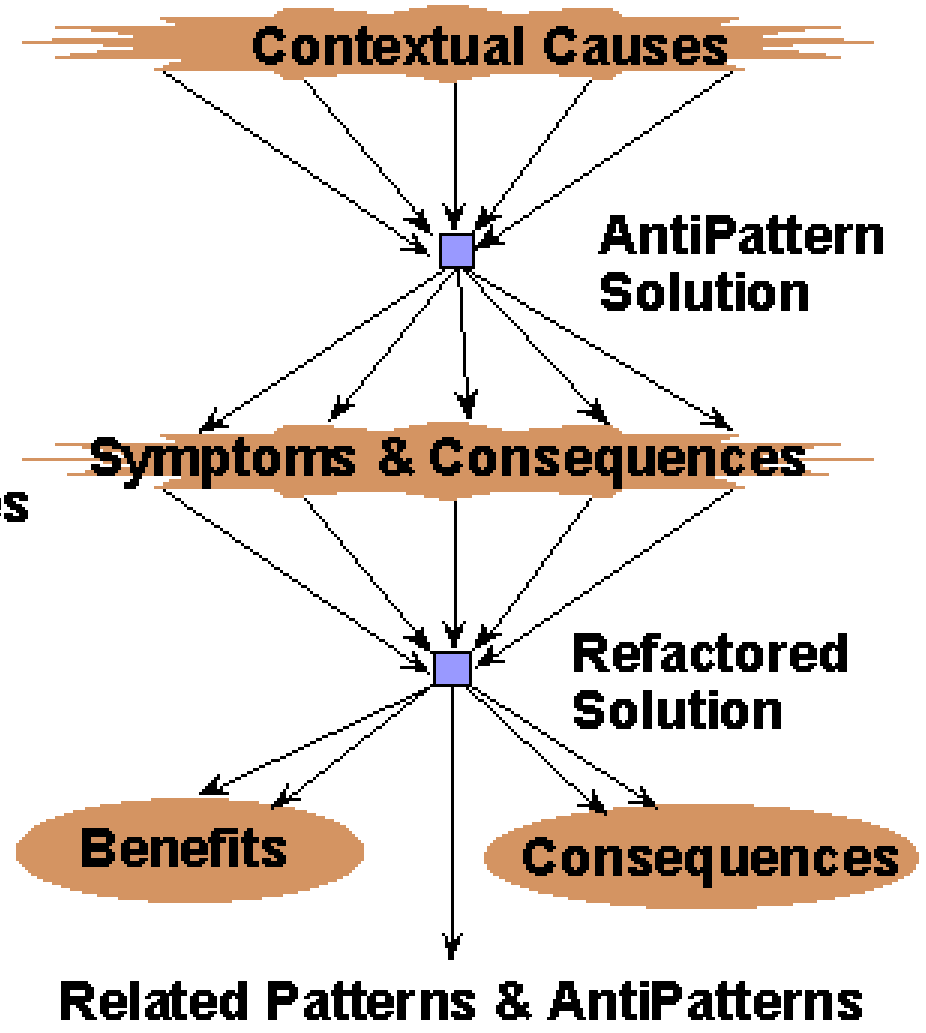
- AntiPatterns are a method of efficiently mapping a general situation to a specific class of solutions.
- AntiPatterns provide real world experience in recognizing recurring problems in the software industry, providing a detailed remedy for the most common predicaments.
- AntiPatterns provide a common vocabulary for identifying problems and discussing solutions.
- AntiPattern support the holistic resolution of conflicts utilizing organizational resources at several levels, where possible.
- AntiPatterns provide stress release in the form of shared misery for the most common pitfalls in the software industry.




Design Patterns
Problem + Solution Pairs
Problem
Context & Forces
Solution
Benefits
Consequences
Related Patterns
AntiPatterns
Solution + Solution Pairs
Contextual Causes
AntiPattern Solution
Symptoms & Consequences
Refactored Solution
Benefits
Consequences
Related Patterns & AntiPatterns

AntiPattern Template

- AntiPattern Name & AKA
- Reference Model Keywords
- Background
- Anecdotal Evidence
- AntiPattern Solution (General Form)
- Symptoms and Consequences
- Typical Causes
- Refactored Solution
- Variations
- Example
- Related Solutions

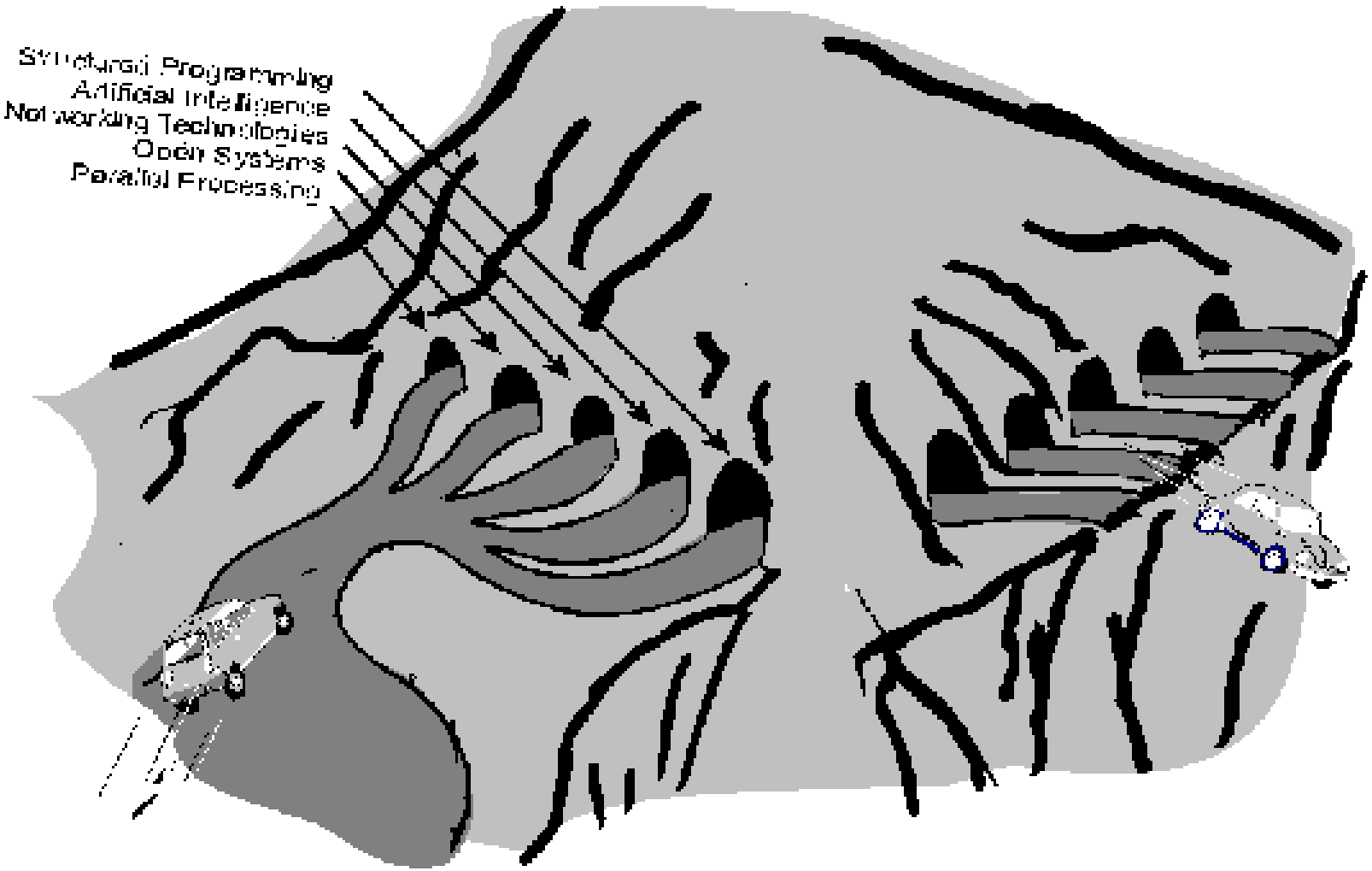


AKA = also known as

AntiPattern Template

- Possible templates:
  - *Mini-AntiPattern* and *Full AntiPattern* template
- Mini-AntiPattern template consists of:
  - compact, unique and unambiguous name
  - description of the AntiPattern solution
  - description of the refactored solution
- Full AntiPattern template includes also for instance:
  - scale and scope of the AntiPattern
  - root causes; practical description of the typical causes
  - misused primal forces (mal uso de las fuerzas originales)
  - symptoms of the problem; (negative) consequences
  - variation

Yesterday's hot solution can become today's AntiPattern



Background for AntiPatterns

- *"AntiPattern es una forma literaria de describir una solución común a un problema que genera decididamente consecuencias negativas."*

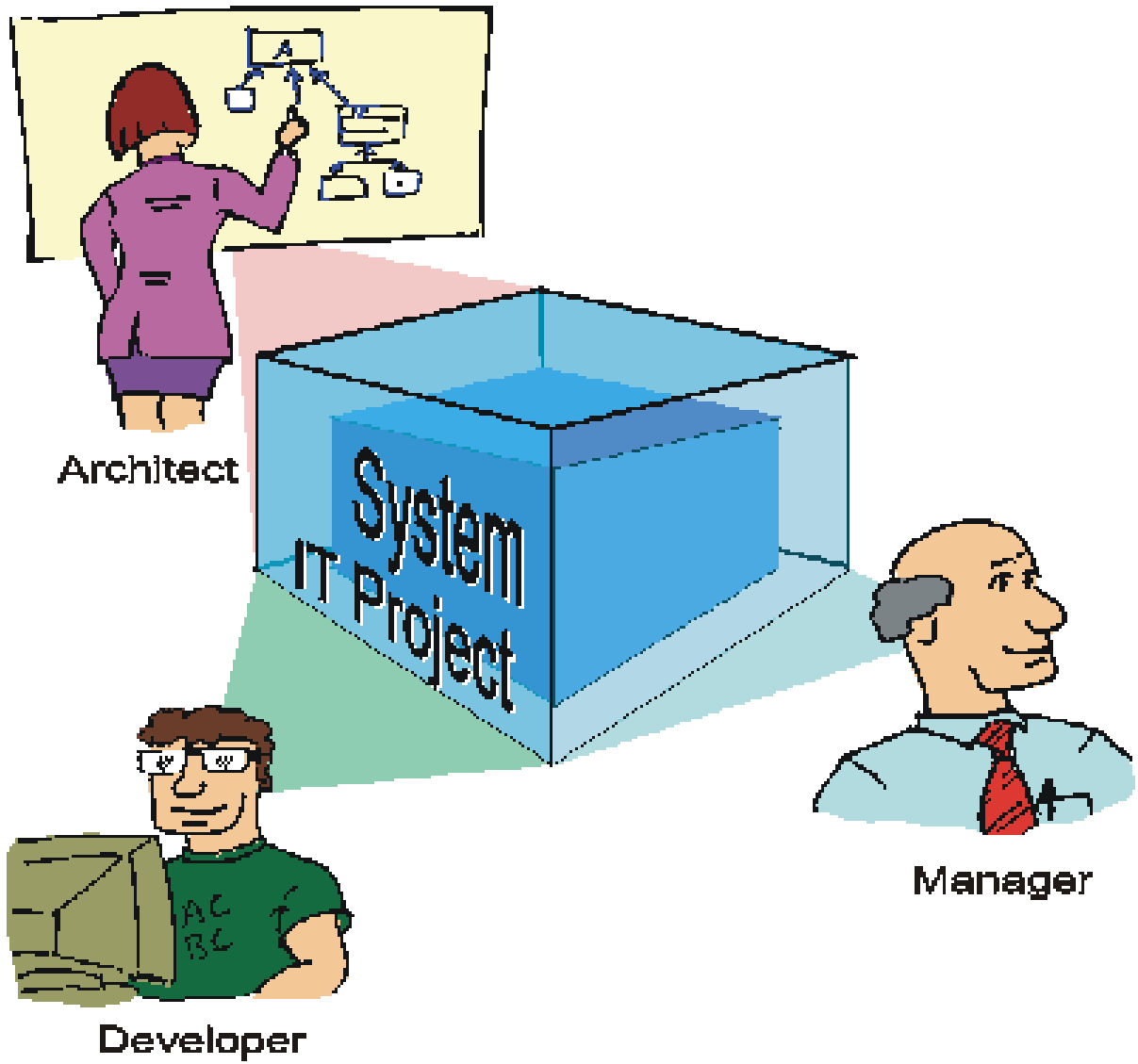
- Design Patterns definen una solución ideal a un problema, mientras que AntiPatterns describen *dos* soluciones:
  - AntiPattern solución: la no querida, problemática, solución existente
  - solución refactoreada : la buena solución (meta del AntiPattern)

- El proposito es dar a la industria IT un vocabulario comun y la forma de discutir y mitigar los problemas comunes

AntiPattern Fundamentals

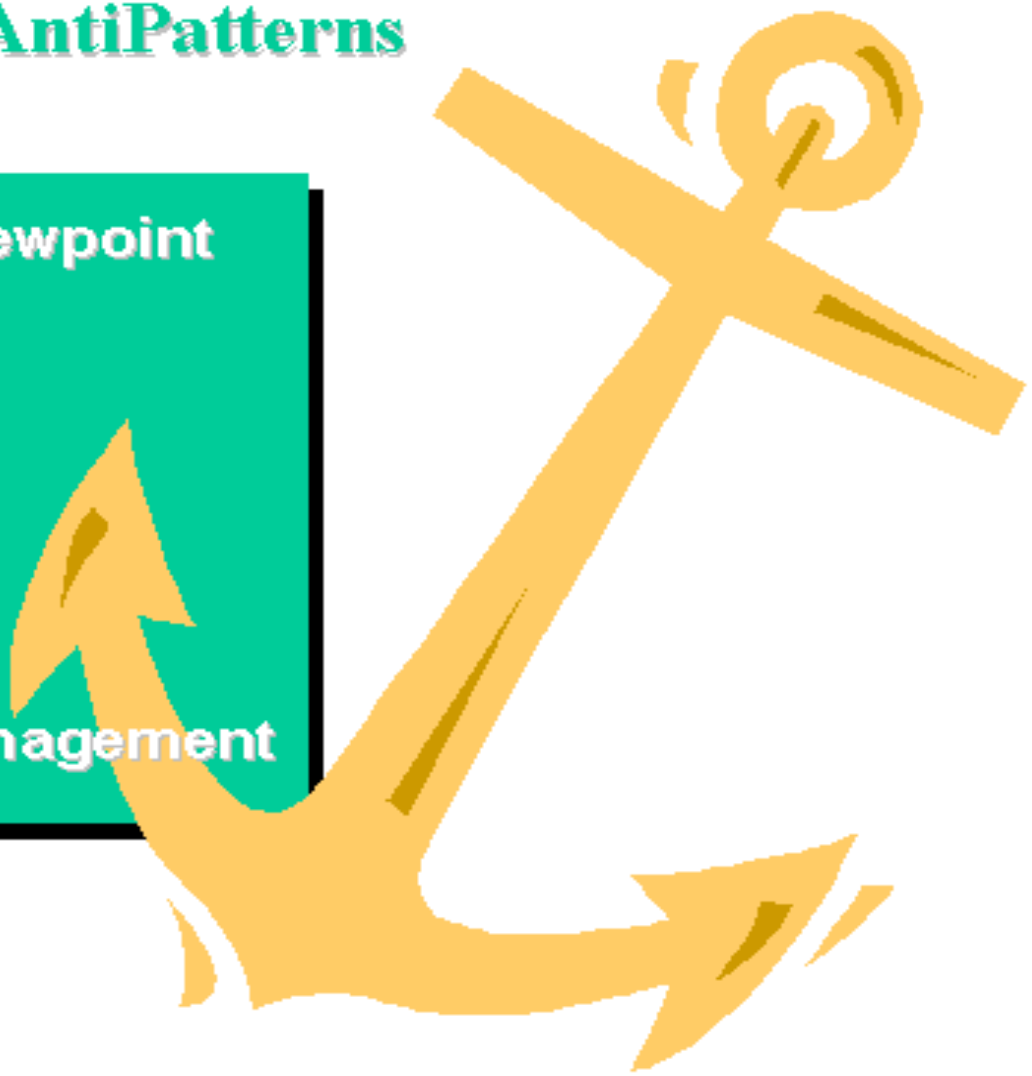
- Three **groups** of AntiPatterns
  - Development, Architecture, Management
- **Root causes** for AntiPatterns
  - Prisa, apatía, mente estrecha, pereza, avaricia, ignorancia, orgullo
- **Primal forces** — management of
  - Funcionalidad, desempeño, complejidad, cambio, transferencia tecno
- **Software Design-Level Model**
  - Nivel de objetos, framework, aplicación, sistema, empresa, industria

AntiPattern Viewpoints



Development Mini-AntiPatterns

- Ambiguous Viewpoint
- Boat Anchor
- Continuous Obsolescence
- Dead End
- Input Kludge
- Mushroom Management



Development AntiPatterns

- The Blob - AKA: God Class
- Cut and Paste Programming
- Functional Decomposition
- Golden Hammer

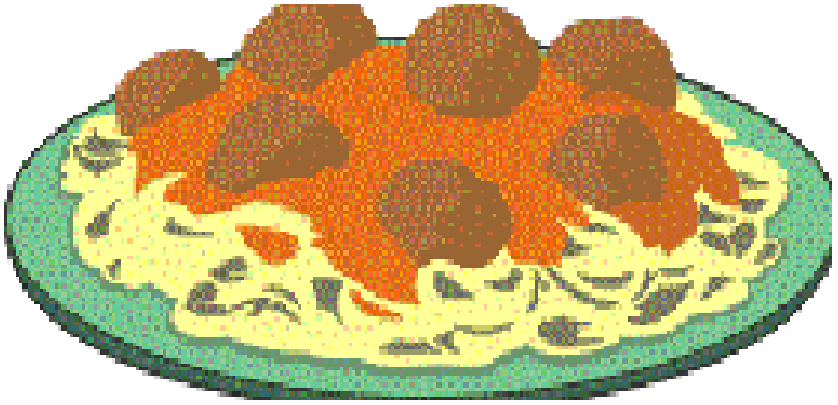
- Lava Flow
- Poltergeists - AKA: Proliferation of Classes
- Spaghetti Code

Walking Through a Minefield
Cut and Paste Programming

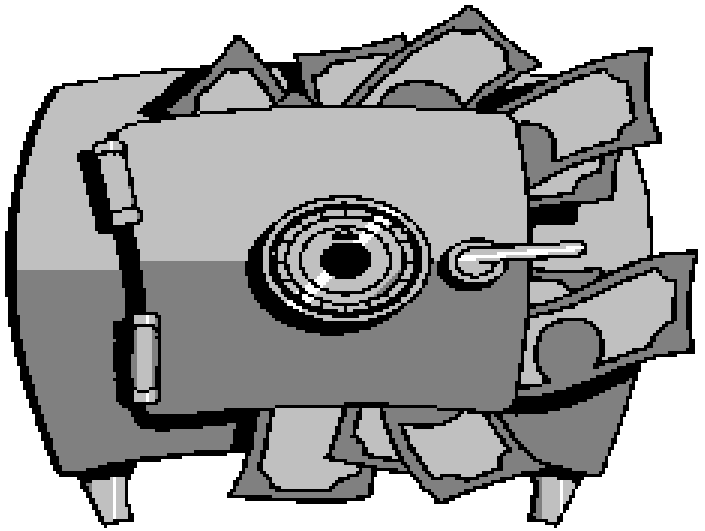
Development AntiPattern:

Spaghetti Code

- **spa·ghet·ti code** [Slang] an undocumented piece of software source code that cannot be extended or modified without extreme difficulty due to its convoluted structure.



Un-structured code
is a liability



Well structured code
is an investment.

Development AntiPattern:

Spaghetti Code

Symptoms

- Quick demonstration code that became operational
- "Lone Ranger" programmer *(who was that masked man?)*
- Obsolete or scanty documentation
- 50% of maintenance spent on system rediscovery
- Hesitant Programmer Syndrome
  - More likely to break it than extend it
  - Easier to just rewrite it
- Cannot be reused
  - System software and COTS packages can't be upgraded
  - Performance cannot be optimized
- User work arounds

Development AntiPattern:

Spaghetti Code

Object-Oriented Spaghetti Code

- Many object methods with no parameters
- Suspicious class or global variables
- Intertwined and unforeseen relationships between objects
- Process-oriented methods, objects with process-oriented names.
- OO advantage lost -- inheritance cannot be used to extend the system, polymorphism not effective either.

Bottom Line: Software has reached point of diminishing returns here the effort involved to maintain existing code exceeds the cost of developing a new "ground up" solution

Development AntiPattern:

Spaghetti Code - Refactored 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

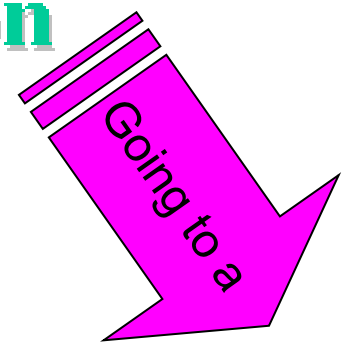
- **Refactor to combine: *Capture aggregations and components***

  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

Source: [Opdyke 92]

Development AntiPattern:

Spaghetti Code - Refactored Solution



RavioliCode

Strategy: Reform the software process

- **Refactoring as you program [Beck 96]**
  - Incremental development
  - Refactoring to improve structure
  - Incremental test
  - Iterate
- **Use programming discipline [Humphrey 95]**
  - Keep track of defects (metrics)
  - Learn to avoid programming defects
- **Use Architecture-Centered development [Booch 96]**
  - Define enforceable system boundaries
  - Use design patterns to document software

Development AntiPattern:

The Blob

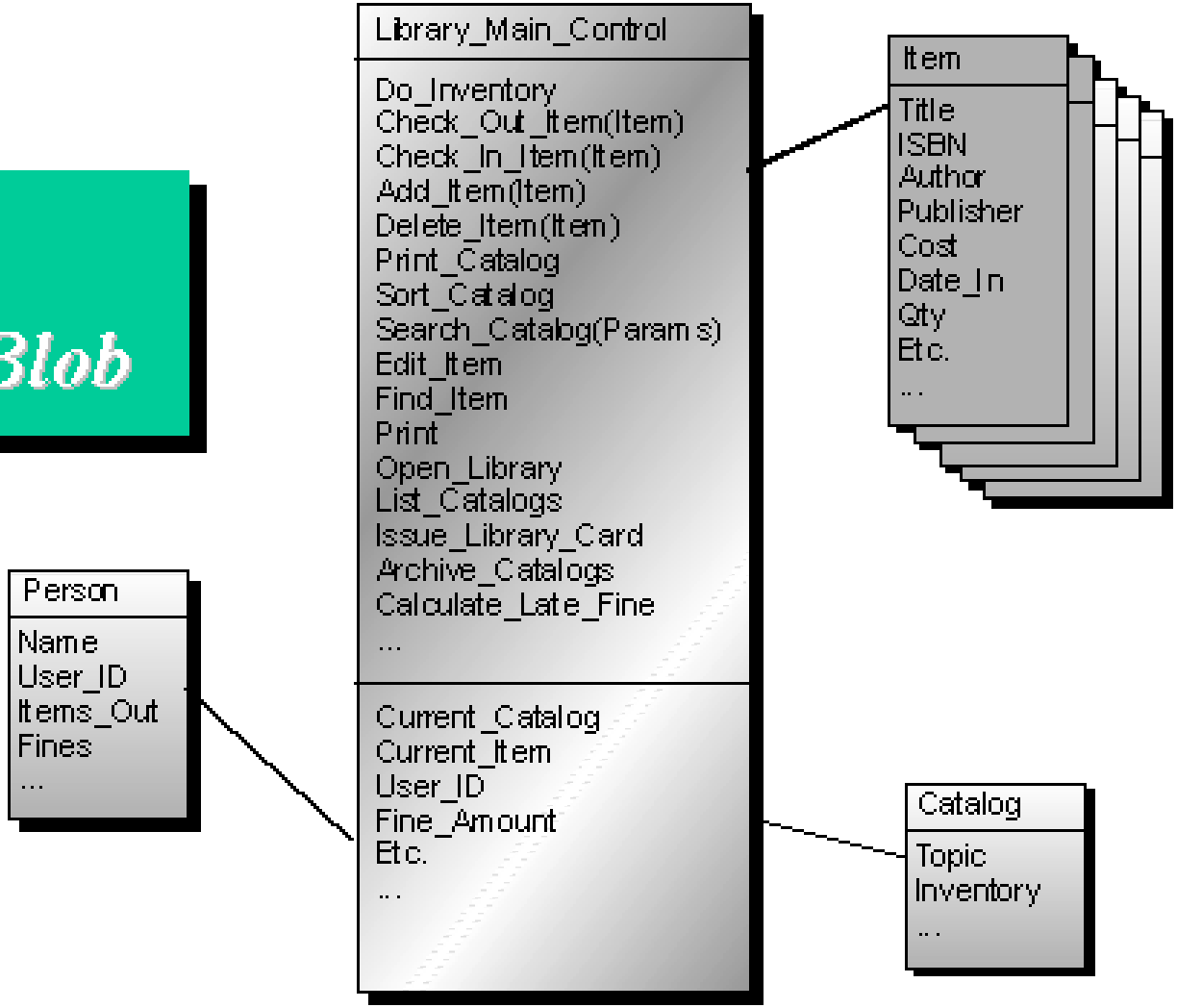
- **Symptoms**
  - Single class with many attributes & operations
  - Controller class with simple, data-object classes.
  - Lack of OO design.
  - A migrated legacy design
- **Consequences**
  - Lost OO advantage
  - Too complex to reuse or test.
  - Expensive to load

Development AntiPattern:

The Blob

Example:

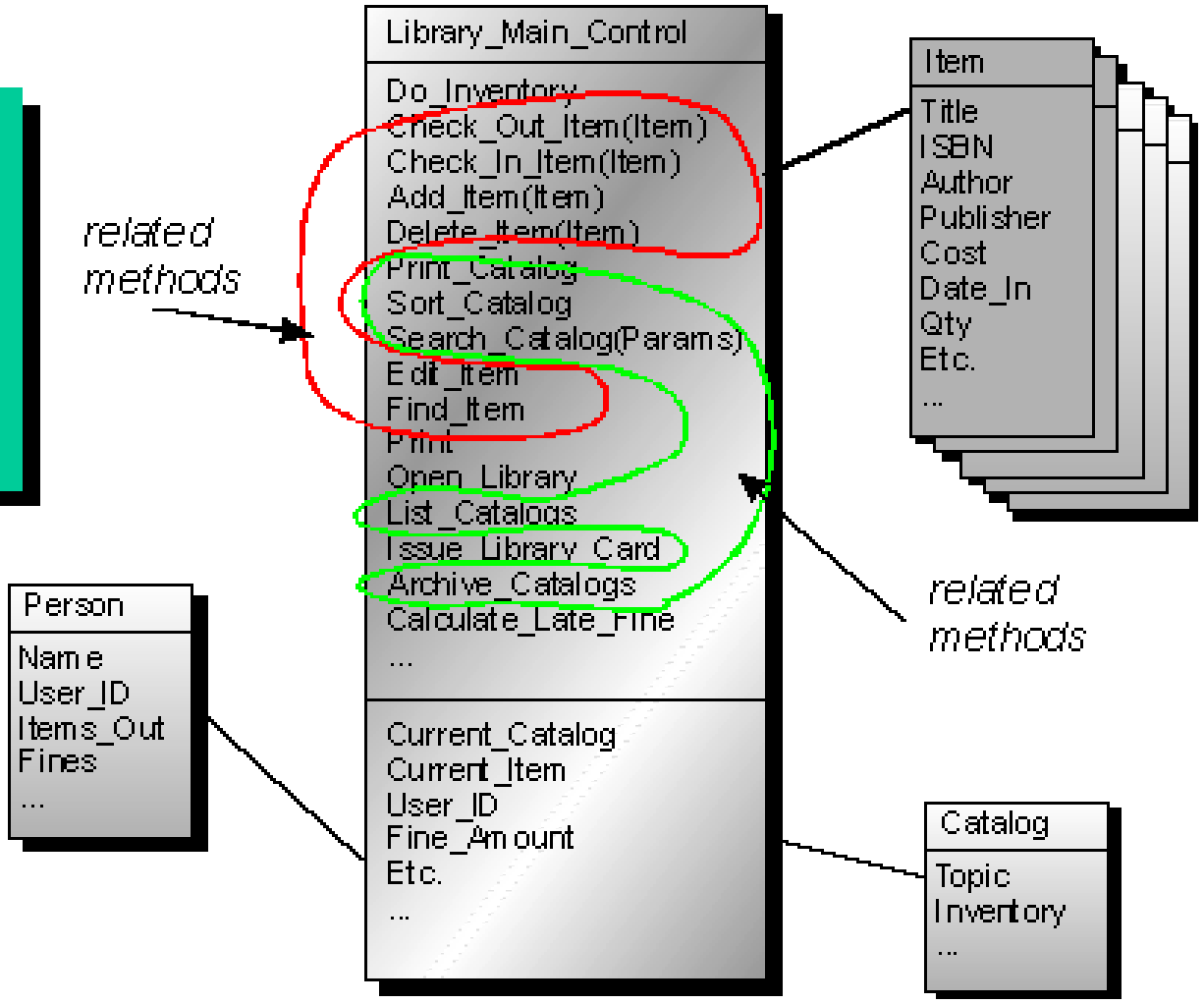
The Library Blob



Development AntiPattern:

The Blob - Refactoring

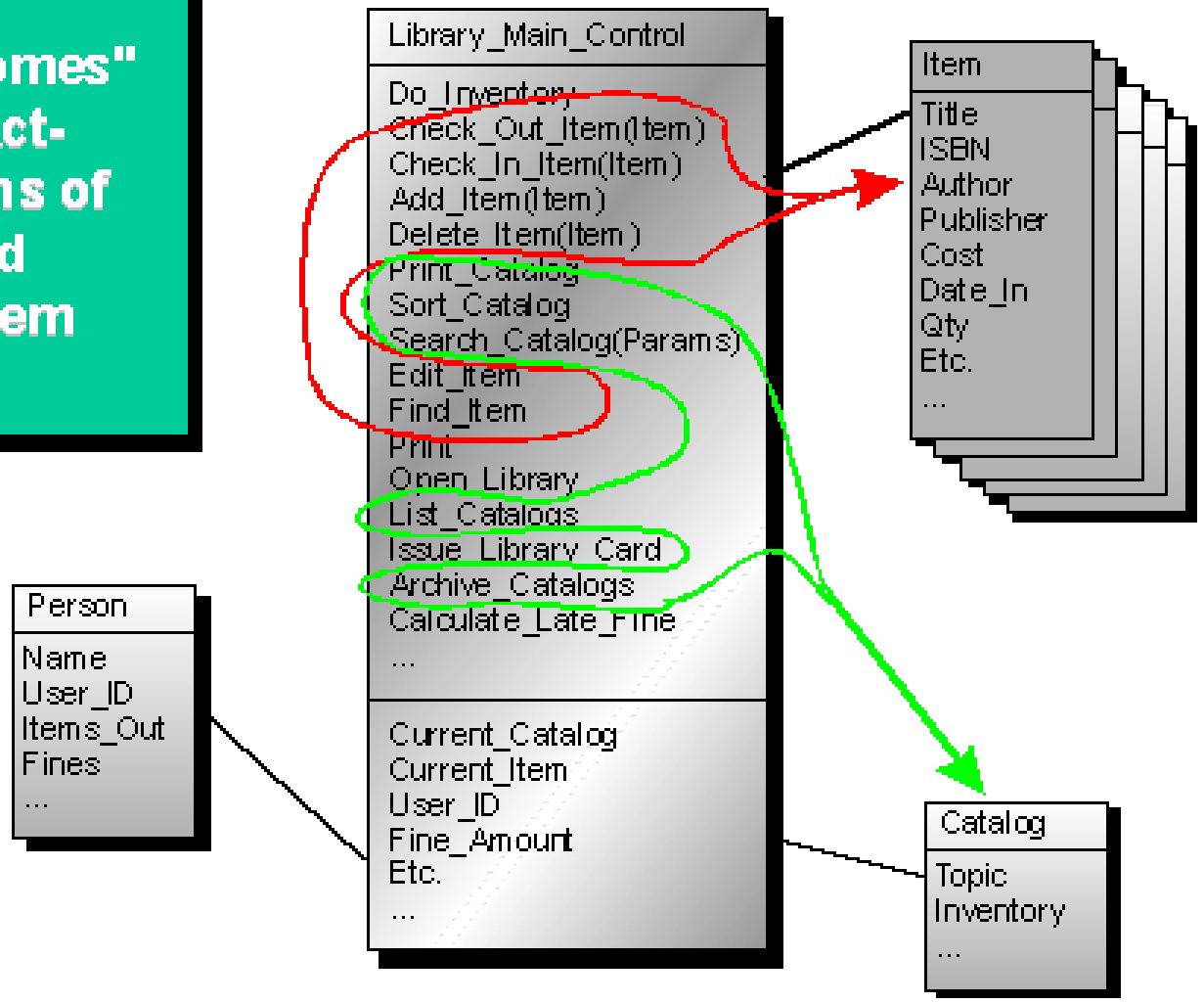
Step 1:
Identify or categorize related attributes and operations according to contracts.



Development AntiPattern:

The Blob - Refactoring

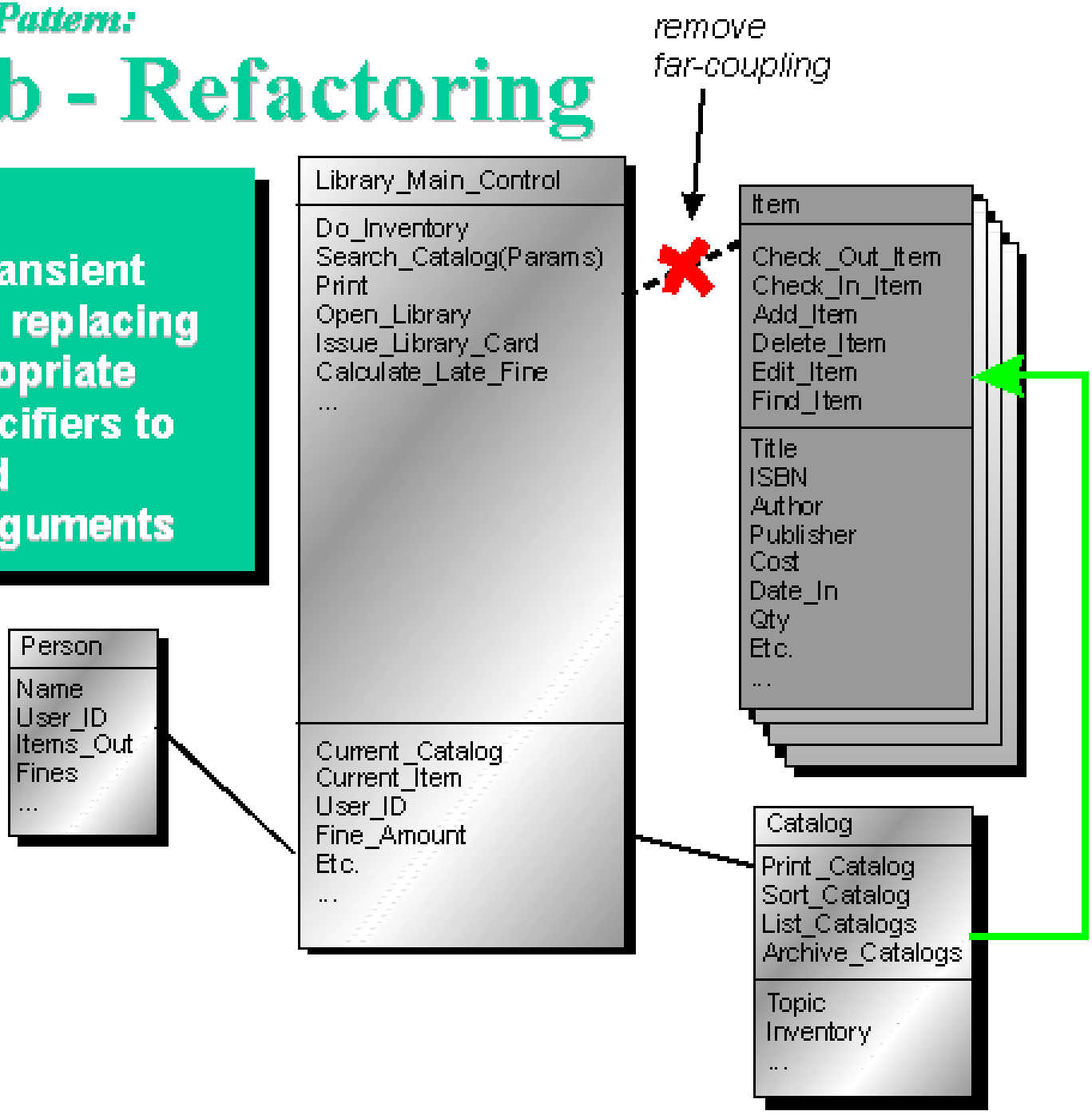
Step 2:
Find "natural homes" for these contract-based collections of functionality and them migrate them there



Development AntiPattern:

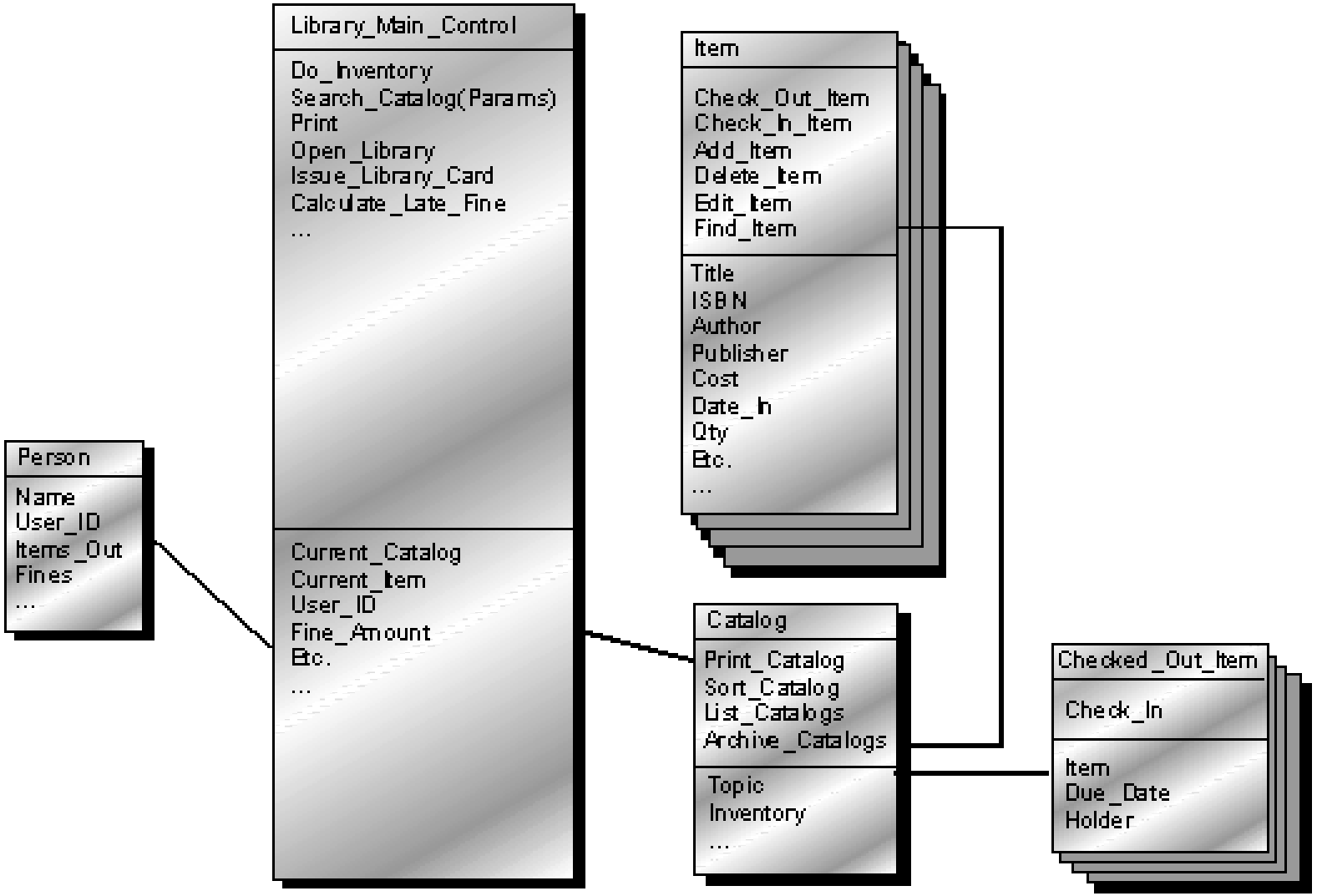
The Blob - Refactoring

Final Step:
Remove all transient associations, replacing them as appropriate with type specifiers to attributes and operations arguments



Development AntiPattern:

The Blob Refactored



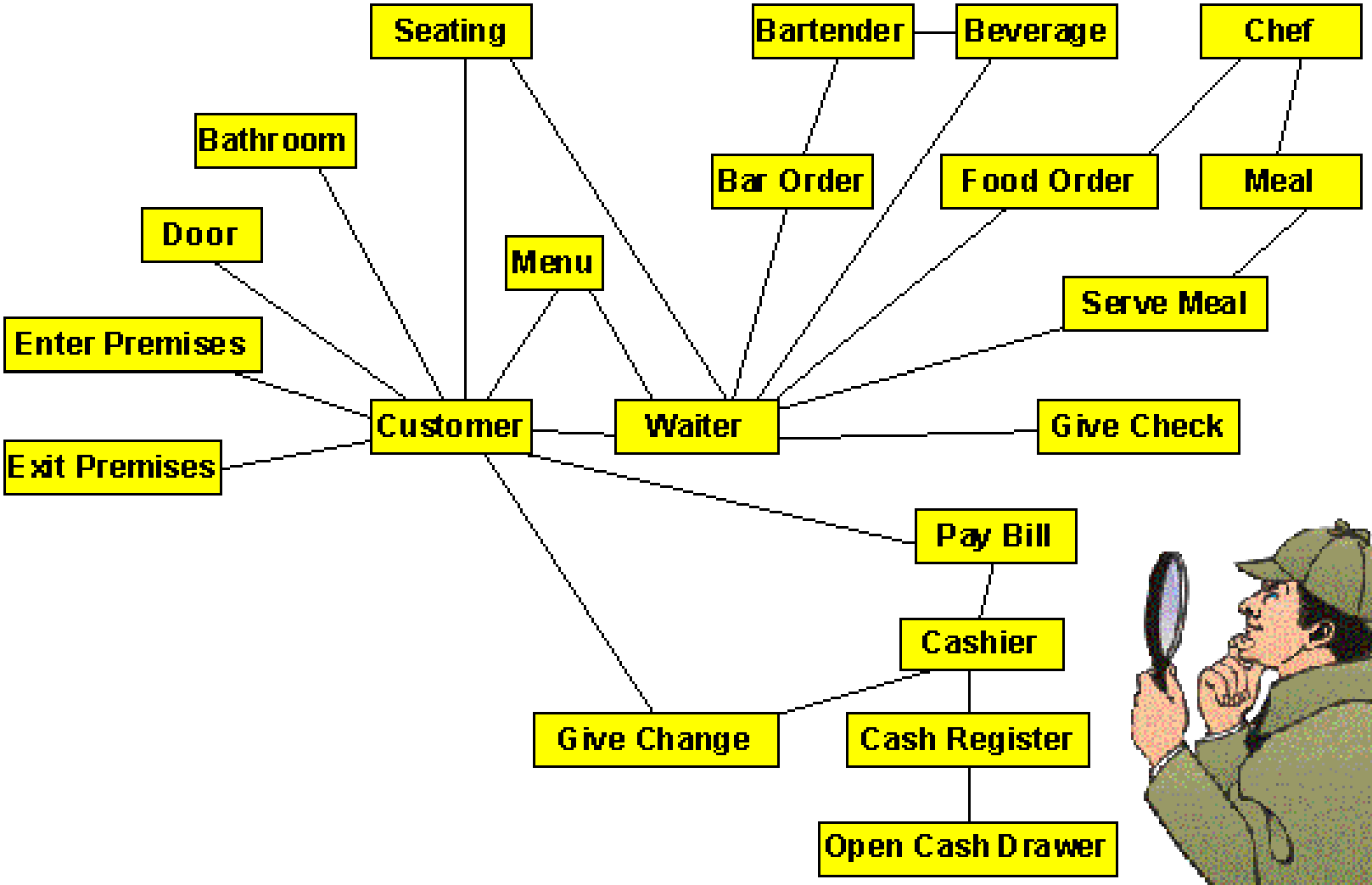
Development AntiPattern:

Poltergeists

- Proliferation of classes [Riel 96]
- Spurious classes and associations
  - Stateless, short-lifecycle classes
  - Classes with few responsibilities
  - Transient associations
- Excessive complexity
- Unstable analysis and design models
- Analysis paralysis
- Divergent design and implementation
- Poor system performance
- Lack of system extensibility

Development AntiPattern:

Poltergeists Example



Development AntiPattern:

Poltergeists Refactored Solution

- **Refactor to eliminate irrelevant classes**
  - Delete external classes (outside the system)
  - Delete classes with no domain relevance
- **Refactor to eliminate transient "data classes"**
- **Refactor to eliminate "operation classes"**
- **Refactor other classes with short lifecycles or few responsibilities**
  - Move into collaborating classes
  - Regroup into cohesive larger classes

Development AntiPatterns

- The Blob - AKA: God Class
- Cut and Paste Programming
- Functional Decomposition
- Golden Hammer

- Lava Flow
- Poltergeists - AKA: Proliferation of Classes
- Spaghetti Code

Walking Through a Minefield
Cut and Paste Programming

Architecture AntiPatterns

- Architecture By Implication
- Design By Committee
- Reinvent the Wheel
- Stovepipe Enterprise
- Stovepipe System
- Vendor Lock-In

AntiPattern
(Intended Solution)

Resulting
Negative
Consequence

Architecture AntiPattern:

Vendor Lock-In

"Our ***architecture*** is Oracle / Microsoft / CORBA"

really means:

"We don't have an architecture"

or:

"We're completely dependent on vendor X."

Architecture AntiPattern:

Vendor Lock-In

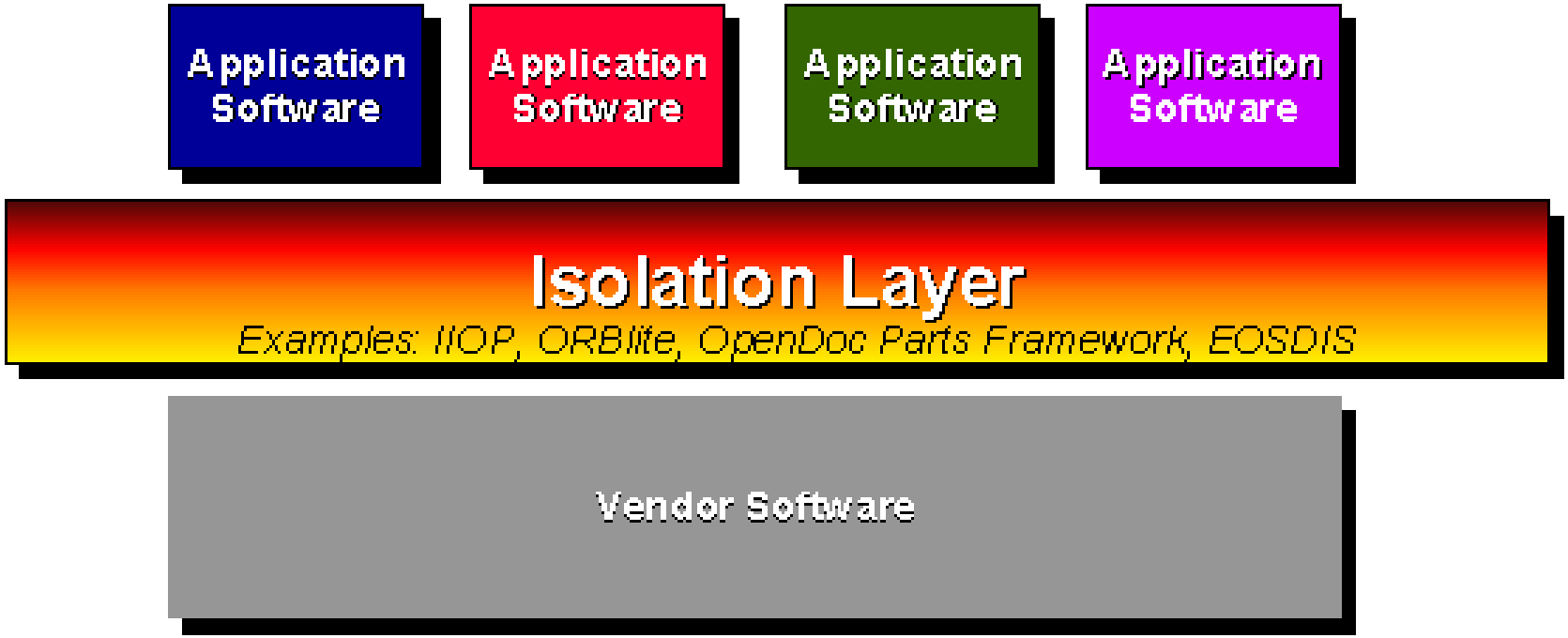
Who's profiting from your misfortune?

- **Loss of control**
  - The product does not live up to expectations
  - The features you need are always 6 months away
  - The vendor changed the product and broke your software
- **The Connector Conspiracy**
  - Vendors products hardly interoperate
  - Product versions proliferate
  - Only certain versions work together, but not the ones you bought
  - Upgrade lifecycles are decreasing: Word97, Word98, ...

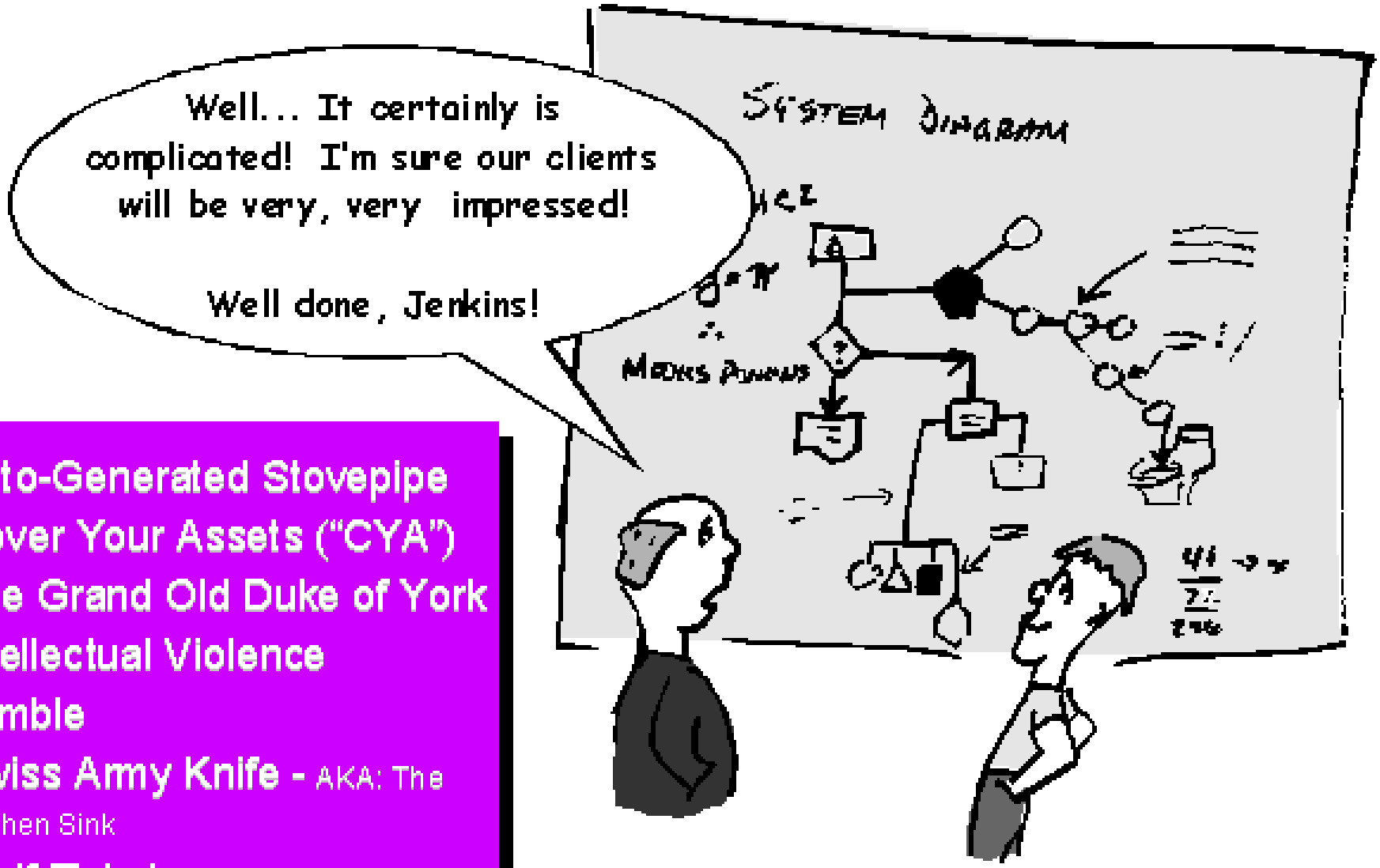
Architecture AntiPattern:

Vendor Lock-In - Refactored Solution

Solution Strategy: Isolation Layer



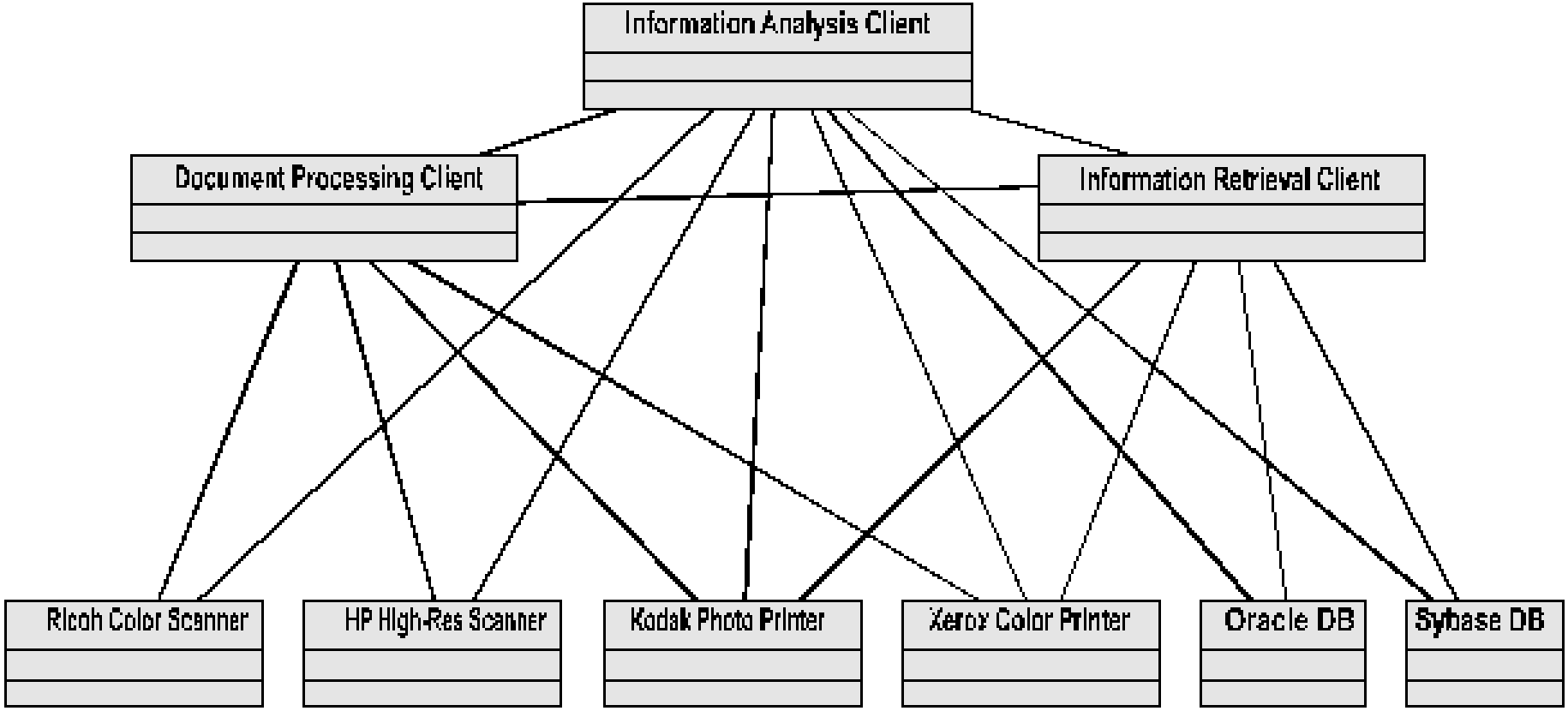
Architecture Mini-AntiPatterns



- Auto-Generated Stovepipe
- Cover Your Assets ("CYA")
- The Grand Old Duke of York
- Intellectual Violence
- Jumble
- Swiss Army Knife - AKA: The Kitchen Sink
- Wolf Ticket

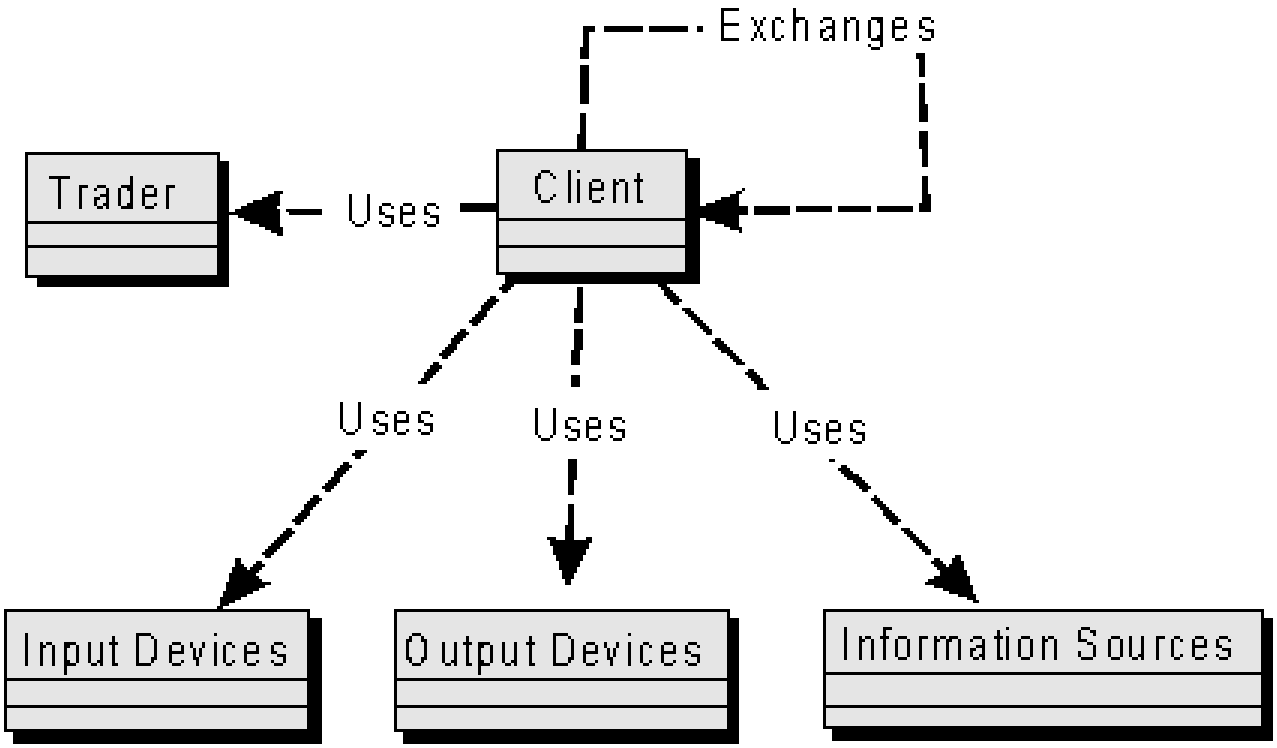
Architecture AntiPattern:

Stovepipe System - Example



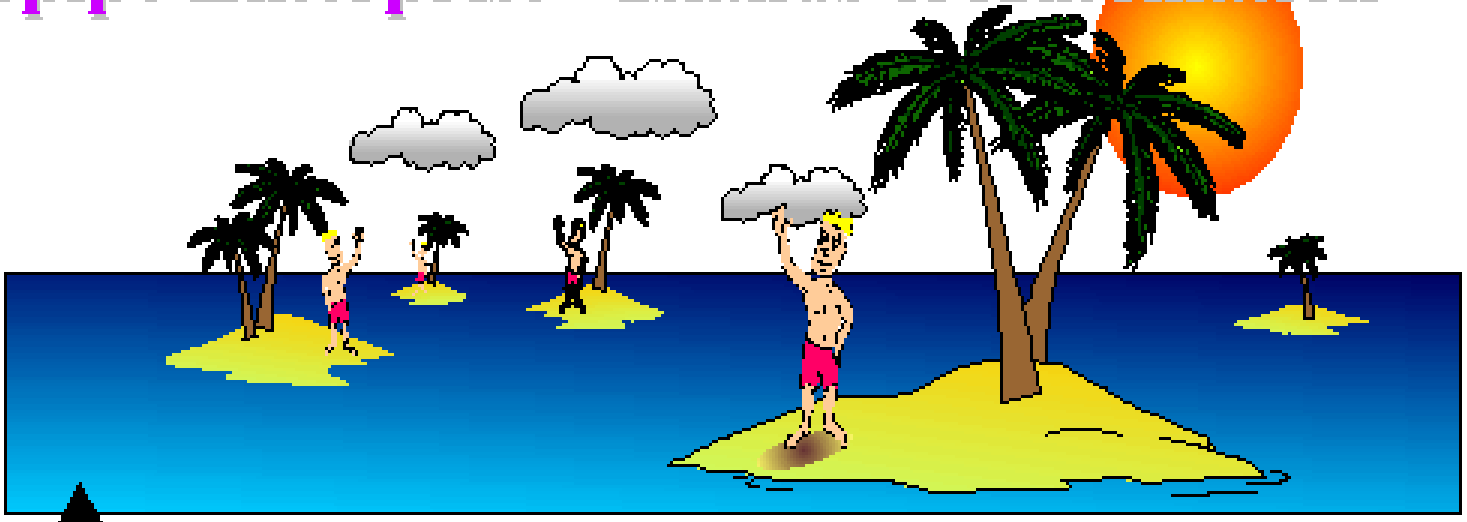
Architecture AntiPattern:

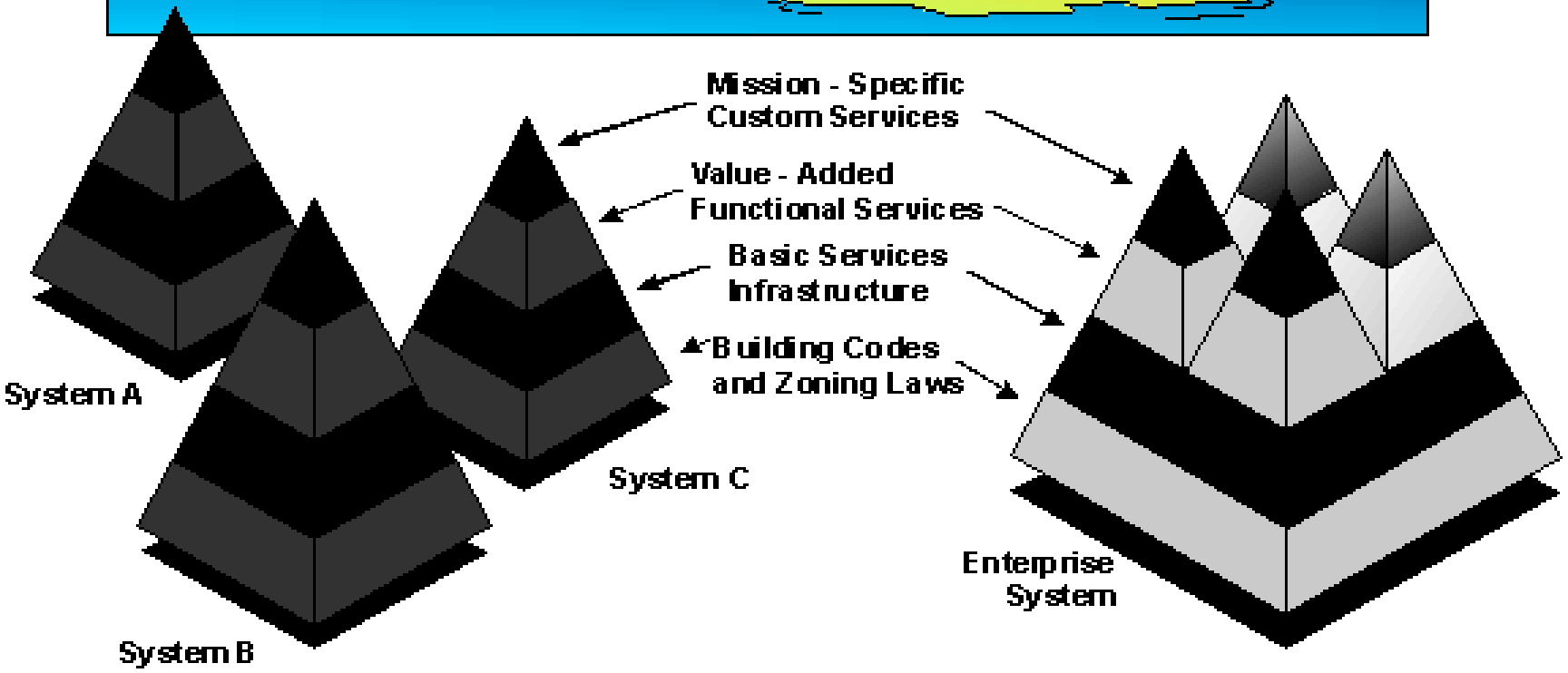
Stovepipe System - Refactored Solution



Architecture AntiPattern:

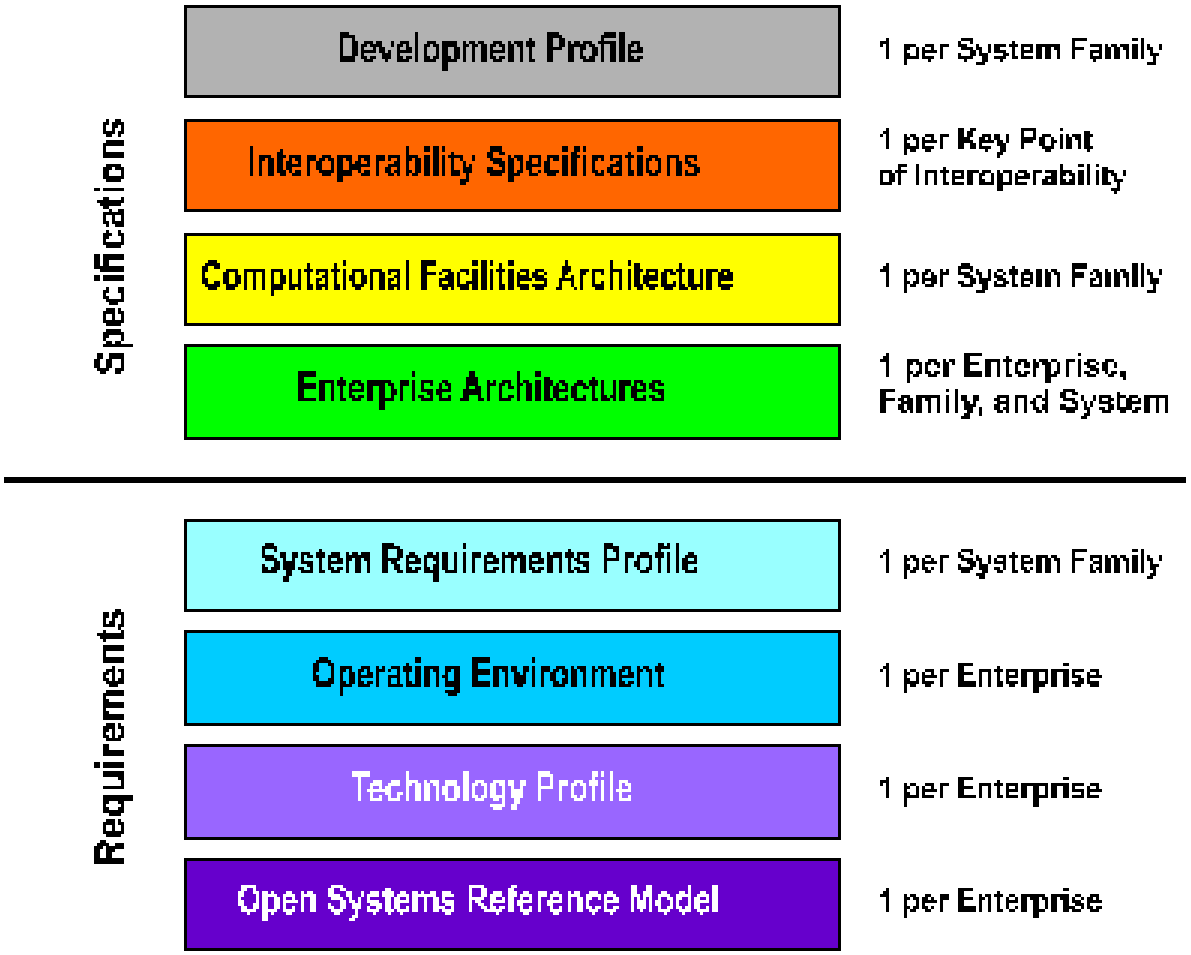
Stovepipe Enterprise - Islands of Automation





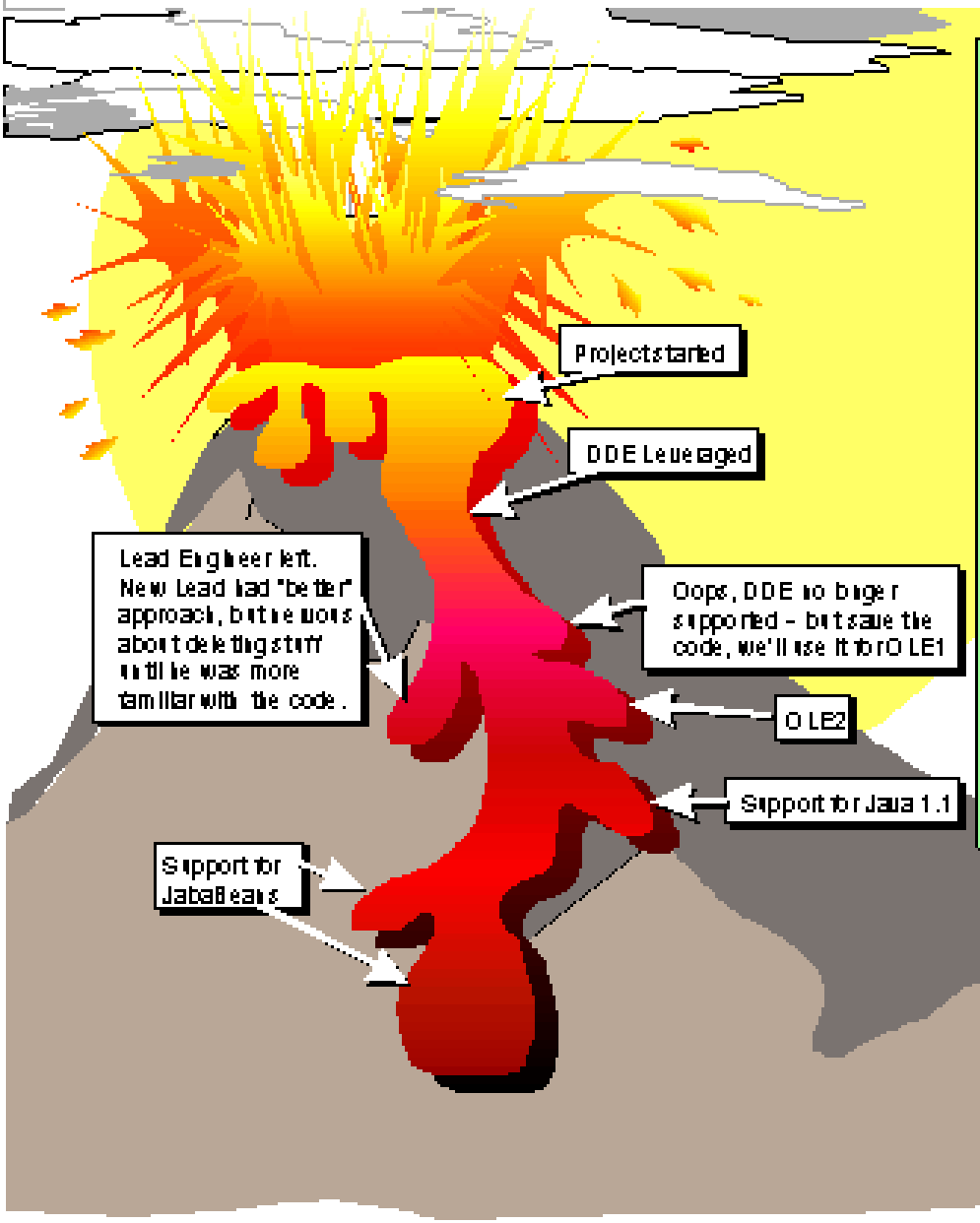
Architecture AntiPattern:

Stovepipe Enterprise - Refactoring Strategy



Architecture AntiPattern:

Vendor Lock-In - Related AntiPattern: Lava Flow

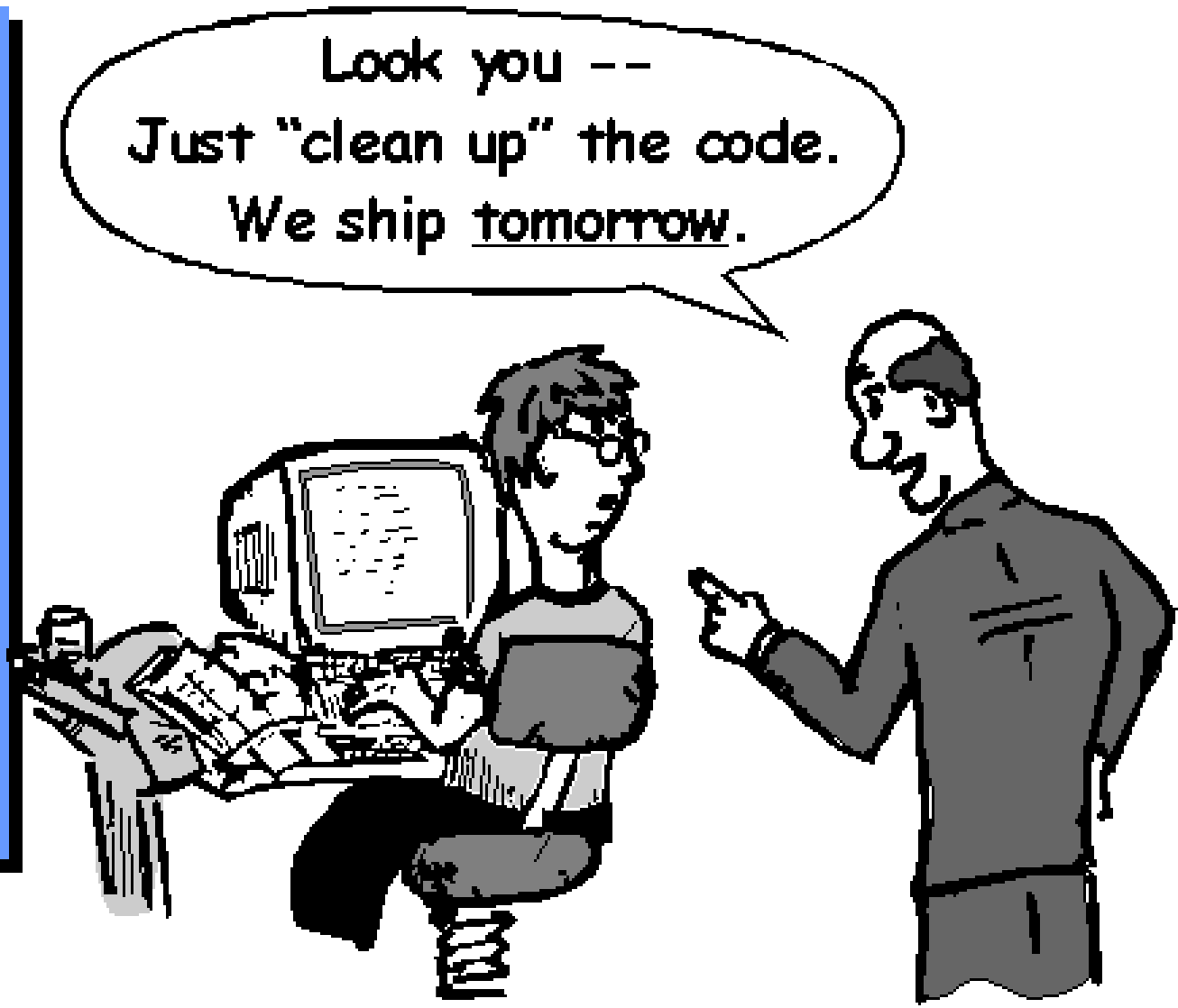



```
...
// This class was written by someone earlier (Alex?) to manage the
//  indexing or something (maybe).  It's probably important.
//
// ****DO NOT DELETE!****
//
// I don't think this is used anywhere - at least not in the new
// Macro_Indexer module which may actually replace whatever
// this was used for, but I'm not sure, so it's best to just leave it
// here for now...  (J.P. - 4/89)

class IndexFrame extends Frame
{
  // IndexFrame constructor
  //-----------------------------------------------
  public IndexFrame(String index_parameter_1)
  {
            // Note: need to add additional stuff here...
            super (str);
  }
  //-----------------------------------------------
```

Management AntiPatterns

- Analysis Paralysis
- Corncob
- Death By Planning
- Irrational Management *AKA: Pathological Supervisor*
- Project Mis-Management



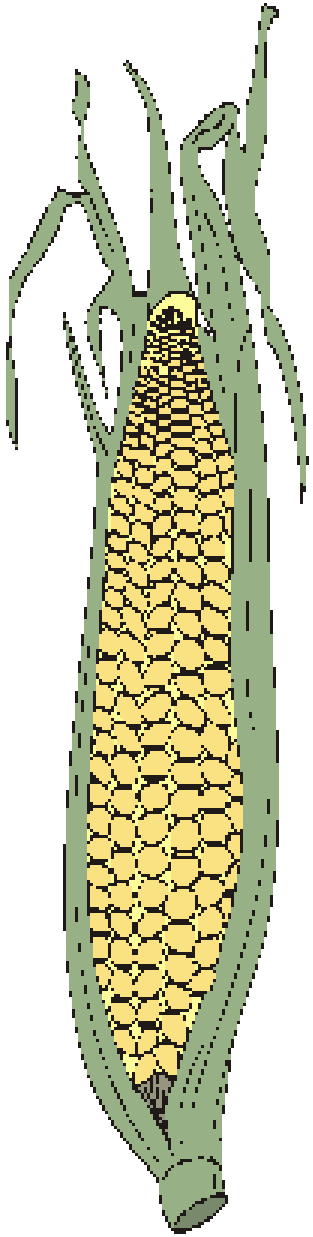
Management Mini-AntiPatterns

- Blowhard Jamboree
- Email is Dangerous
- Fear of Success
- The Feud
- Smoke and Mirrors
- Throw It Over the Wall
- Viewgraph Engineering
- Warm Bodies

Management AntiPattern:

Corncob

The worse things get, the more they come out of the woodwork



Synopsis: Frequently, "difficult people" obstruct and divert the software development process

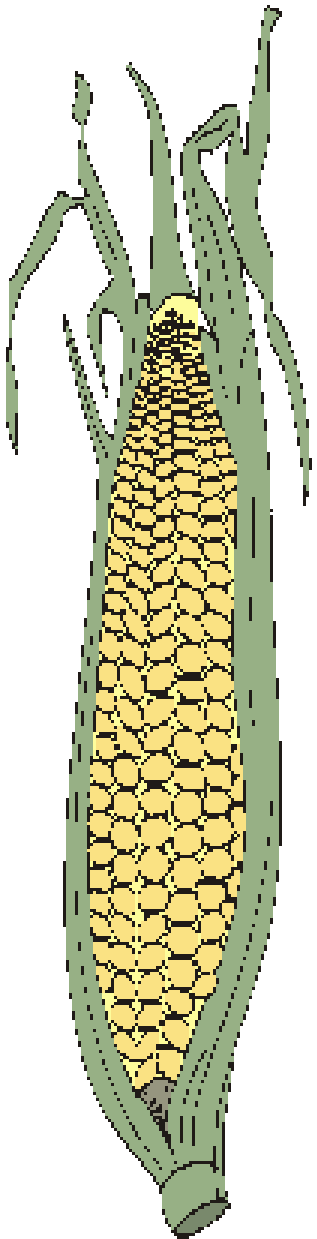
AKA: Corporate Shark, Loose Cannon, Complete Jerk, TWIT

Common Causes:

- Stress and/or personality (see Irrational Management)
- Hidden agendas
  - On Wall Street, up to 75% of programmer compensation is incentive bonus
  - In all industries, most senior IT managers are incentivized and are competing against other IT managers
- Negative training or background
- Defensiveness: Fear of the unknown
- Intellectual Arsenic = Obsession with a pet idea

Management AntiPattern:

Corncob - Refactored Solutions



- **"Bill Gates Special" - Pie in the face!**
- **Responsibility**
  - "You raised the issue, you own/fix the problem"
- **Corrective interview**
- **Pizza Party -** "There is no problem so serious that a pizza party can't resolve" - *Randall Oakes, MD and Healthcare IT Evangelist*
- **Stress reduction**
- **Reform the policies and procedures**
- **Reorganization - Corncob support groups**
- **Termination**

AntiPatterns Ground Rules

Use AntiPatterns to quickly move through negative issues and onto positive solutions

- **AntiPatterns are normal, status-quo solutions, like "dysfunctional families"**

  Some AntiPatterns must be tolerated:

  "Accept those things you cannot change; have the courage to change those things you can, and the wisdom to know the difference"

  - Serenity Prayer

- **Avoid the Urinary Olympics**

  –Don't use AntiPatterns to exacerbate negativity

- **Avoid the Golden Hammer = Obsessive use of 1 pattern**

  There are 191 fundamental software patterns

  (23 Gamma Patterns + 17 Buschmann Patterns + 72 Analysis Patterns + 38 CORBA Design Patterns + 121 AntiPatterns)

- **Be sophisticated, use a range of solutions**

Conclusions

AntiPatterns are a more compelling form of patterns

Each AntiPattern includes a solution + solution pair

- AntiPattern Solution Generates mostly negative consequences
- Refactored Solution Generates mostly positive benefits

AntiPatterns are useful for refactoring, migration, upgrade, and reengineering

Bibliografía

- **Links**
  - *http://www.antipatterns.com/briefing/sld001.htm*
  - *http://c2.com/cgi/wiki?AntiPatternsCatalog*
- **Libros**
  - "*J2EE Antipatterns*". Dudney, Asbury, Krozak, Wittkopf. *Wiley & Sons, 2003*
  - "*Bitter Java*". Bruce Tate. *Manning, 2002*
  - "*Antipatterns – Refactoring Software, Architectures, and Projects in Crisis*". Brown, Malveau, McCormick III y Mowbray. *Wiley & Sons, 1998*
  - *"Refactoring to Patterns"* Joshua Kerievsky, *Addison Wesley* , 2004