Modellistica Medica

Maria Grazia Pia

INFN Genova

Scuola di Specializzazione in Fisica Sanitaria
Genova
Anno Accademico 2002-2003

Lezione 12-13

OO modeling

Antipatterns

Antipatterns

- The majority of published works in software sciences have focused on positive and constructive solutions
- AntiPatterns are derived by looking at the negative solutions
- An AntiPattern describes a commonly occurring solution to a problem that generates decidedly negative consequences

Origin of antipatterns

A manager or developer

- does not know any better
- does not have sufficient knowledge or experience solving a particular problem
- applied a perfectly good design pattern in the wrong context

State of Affair

- 5 out of 6 software projects are considered unsuccessful
- 1/3 of all software projects are canceled
- For delivered systems the actual budget and time is often double than expected

Why is it important to identify antipatterns?

Antipattern view-points:

- Developer
  - Situations encountered by programmers
- Architect
  - Common problems in system structure
- Manager
  - Affect people in all software roles

Antipattern Categories

- Development AntiPatterns
- Architectural AntiPatterns
- Managerial AntiPatterns

- AntiPatterns apply to software construction as well as software evolution

Relation between Patterns and AntiPatterns

- Sometimes an antipattern is a pattern in an inappropriate context
- Design patterns often evolve into an AntiPattern
  - Procedural programming was a great design pattern in the 60's and 70's
  - Today it is an AntiPattern
- Object-oriented programming is today a practiced pattern...

Reference Model

- Root causes
  - provide fundamental context for the AntiPattern
- Primal forces
  - are the key motivators for decision making
- Software design-level model
  - define architectural scales

Root Causes

- Haste
  - hasty decisions compromise quality
  - code that appears to work is acceptable
  - testing is ignored
- Apathy
  - lack of partitioning
  - ignoring the separation of concerns (e.g., *stable* vs. *replaceable* design)

Root Causes

- Narrow-mindedness
  - refusal of known or accepted solutions
- Sloth
  - poor decision based on an easy answer
  - frequent interface changes
  - lack of configuration control
  - reliance on generating stubs and skeletons

Root causes

- Greediness
  - architectural cupidity: modeling of excessive details
  - excessive complexity due to insufficient abstraction

    overly complex systems are difficult to

    - develop,
    - integrate,
    - test,
    - maintain,
    - extend

Root Causes

- Ignorance
  - failing to seek understanding
  - antonym of “analysis paralysis”
  - focusing on code interfaces rather than system interfaces
  - no layering
  - no levels of indirection
  - no wrapping to isolate details

Root Causes …

- Pride
  - not- invented- here syndrome
  - unnecessary invention of new designs
  - reinventing the wheel
  - rewrite from scratch
  - ignoring requirements
  - ignoring COTS, freeware, existing legacy system

Primal Forces

- Present in nearly all design or reengineering situations
- Keep architecture and development on track or synchronized
- A fundamental value system for software architects

- Management of functionality
  - Meeting the requirements
- Management of performance
  - Meeting required speed and operation
- Management of complexity
  - Defining abstractions
- Management of change
  - Controlling the evolution of the software
- Management of resources
  - People and IT artifacts
- Management of technology
  - Controlling technology evolution

Software Development AntiPatterns

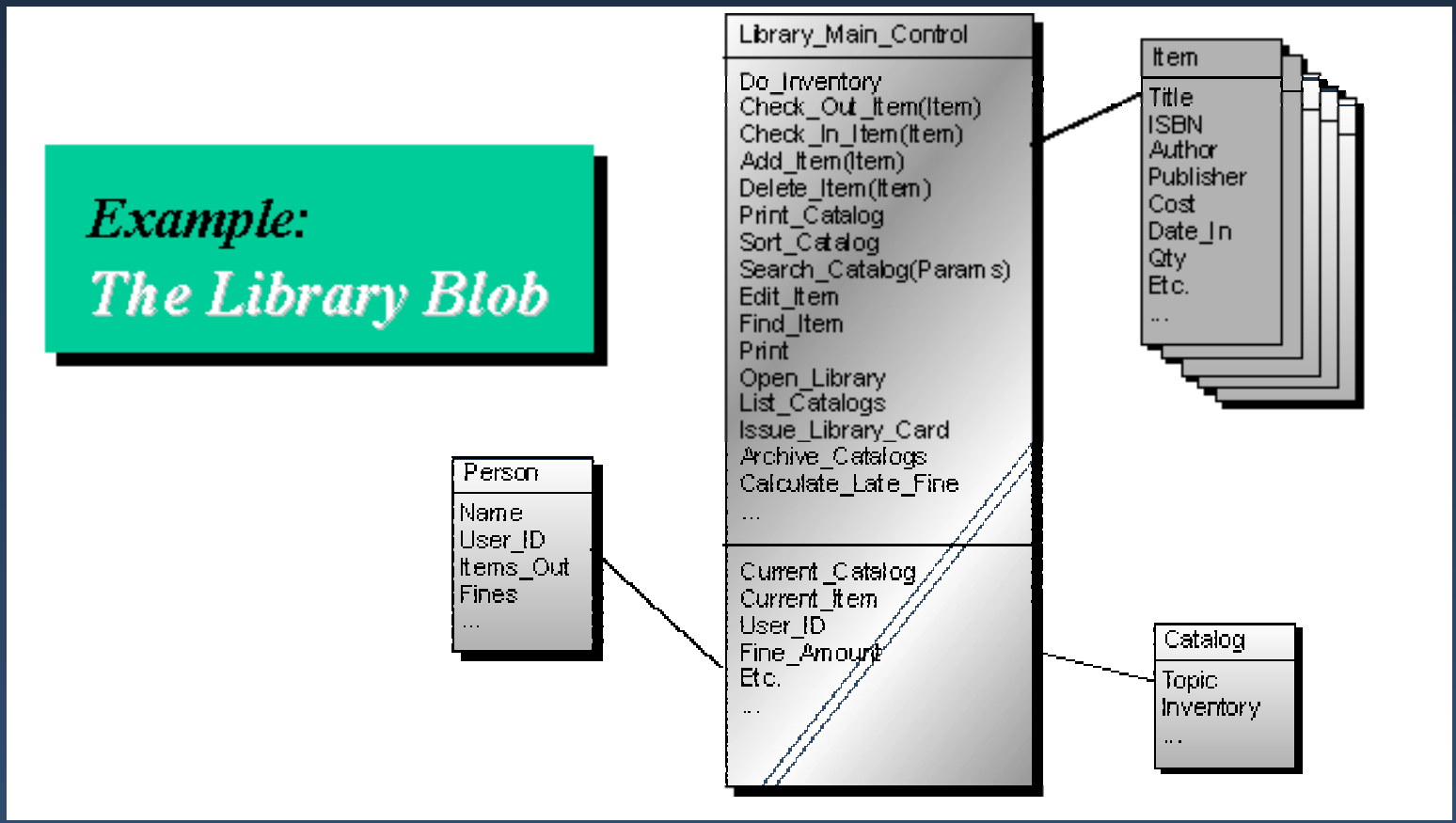
- The Blob
- Continuous obsolescence
- Lava Flow
- Ambiguous viewpoint
- Functional decomposition
- Poltergeists
- Boat Anchor
- Golden Hammer
- Dead End
- Spaghetti Code
- Input Kludge
- Walking through a Minefield
- Cut-and-Paste Programming
- Mushroom Management

The Blob

Problem

- Procedural style design leads to one object with a lion's share of the responsibilities
- Most other objects only hold data
- *"This is the class that is really the heart of our architecture"*
- One class monopolizes the processing and the others encapsulate data

The Blob


Example:
The Library Blob
Library_Main_Control
Do_Inventory
Check_Out_Item(Item)
Check_In_Item(Item)
Add_Item(Item)
Delete_Item(Item)
Print_Catalog
Sort_Catalog
Search_Catalog(Params)
Edit_Item
Find_Item
Print
Open_Library
List_Catalogs
Issue_Library_Card
Archive_Catalogs
Calculate_Late_Fine
...
Current_Catalog
Current_Item
User_ID
Fine_Amount
Etc.
...
Item
Title
ISBN
Author
Publisher
Cost
Date_In
Qty
Etc.
...
Person
Name
User_ID
Items_Out
Fines
...
Catalog
Topic
Inventory
...

The Blob

- Causes
  - Lack of an object- oriented architecture
  - Lack of architecture enforcement
  - Procedural design expert are chief architects
- Solution
  - Distribute responsibilities more uniformly
  - Isolate the effect of changes
  - Identify or categorize attributes and operations
  - Find “natural homes” for the identified classes
  - Remove outliers

Lava Flow

- Problem
  - Dead- code and forgotten design information is frozen in an ever- changing design

- Oh that! Well, Alessandra and Giovanni (they're no longer with the group) wrote that routine back when Riccardo (who left last month) was trying a workaround for Elena's input processing code (she's in another group now)…

Lava flow

- Problem

  - Lead engineer left
  - New lead had better approach but was nervous about deleting stuff until he was more familiar with the code

  - Each volcanic eruption leaves lava streams

Lava Flow

Causes

- R&D code moved to production
- Uncontrolled distribution of unfinished or unpolished code
- Trial approaches have not been eliminated from the code
- Architectural scars due to old middleware

Lava Flow …

Solution

- Configuration management system which identifies and eliminates dead code
- Evolve or refactor design
- Sound architecture must be at the basis of production code development
- Establish stable system level interfaces

Continuous Obsolescence

- **Problem**
  - Technology is changing rapidly
  - Developers have difficulty keeping up
  - Product releases don't work together
- **Solution**
  - Open systems standards
  - Use consortium standards since they represent industry consensus
  - Stable system interfaces to separate concerns

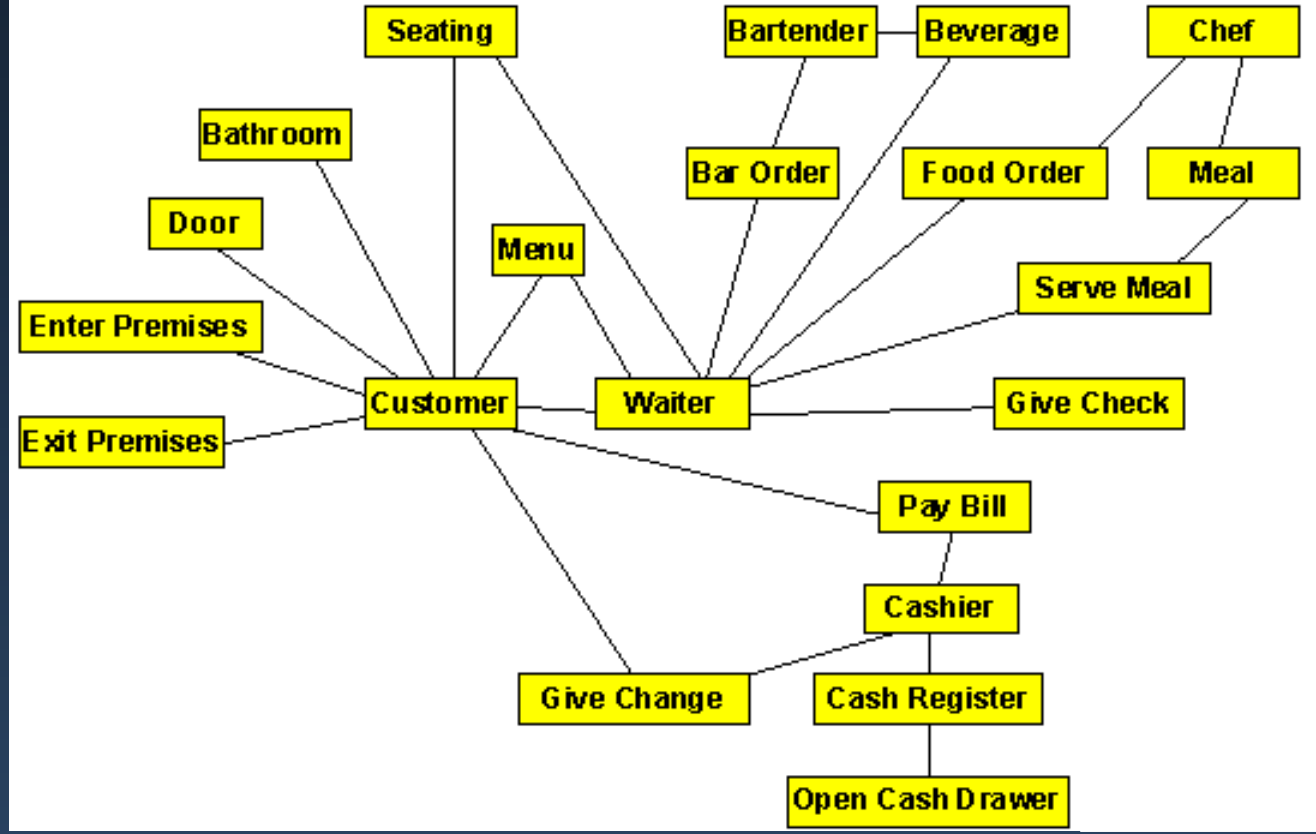
Ambiguous Viewpoint

- **Problem**
  - OOA&D models often do not explain their viewpoint
  - Often implementation view (least useful)
- **Solution**
  - Provide different viewpoints
  - Separation of concerns
  - Interfaces, db, application code

Functional Decomposition

- **Problem**
  - Result of experienced, non-OO developers
  - Procedural design in an OO language
  - Class-based versus object-oriented code
  - Complex and clever code
- **Solution**
  - Object-oriented redesign
  - Package data and methods
  - Separation of concerns

Poltergeists



- Problem
  - Classes with limited roles or life cycles
  - Start a process for another object
- Solution
  - Refactor into longer- lived objects
  - Package data and methods

Boat Anchor

- Problem
  - A piece of software that does not serve a useful purpose on the current project
  - Often a costly acquisition which management is reluctant to let go
- Solution
  - Ditch the anchor

Golden Hammer

- Problem
  - A familiar and proven technology or concept that is applied obsessively to many software problems
- Solution
  - Expand the knowledge of developers through courses, training, books
  - Expose developers to alternative technologies and approaches

Dead End

- Problem
  - Modifying a reusable component even if it is no longer maintained or supported by the supplier
  - Amount of maintenance increases significantly
- Solution
  - Outsource rather than import maintenance

Spaghetti Code

- Problem
  - Most famous AntiPattern
  - Many complexity measure have been invented to assess it
  - Common for programmer who cannot abstract
- Solution
  - Many automatic tools available

Cute and Paste Programming or Cloning

- Problem
  - Software clones
  - *"Hey, I thought you fixed that bug already, so why is it doing this again?"*
  - *"Wow, you guys work fast. Over 400K LOC in three weeks is amazing!"*
  - Degenerate form of reuse
  - Very common…
- Solution
  - Clone detection
  - Parameterize types
  - Introduce an additional level of indirection
  - Exploit polymorphism
  - Dynamic schemas

Software Architecture AntiPatterns

- Autogenerated Stovepipe
- Stovepipe Enterprise
- Jumble
- Stovepipe System
- Cover Your Assets
- Vendor Lock-in
- Wolf Ticket
- Architecture By Implication
- Warm Bodies
- Design By Committee
- Swiss Army Knife
- Reinvent the Wheel
- The Grand Old Duke of York

Swiss Army Knife or Kitchen Sink

- **Problem**

  - Excessively complex class interface
  - Designer attempts to provide for all possible uses of the class
  - Complicated interface
  - Many overloaded names
  - Excessive regression test suites
  - Several Swiss Army Knifes in a single design

Swiss Army Knife or Kitchen Sink

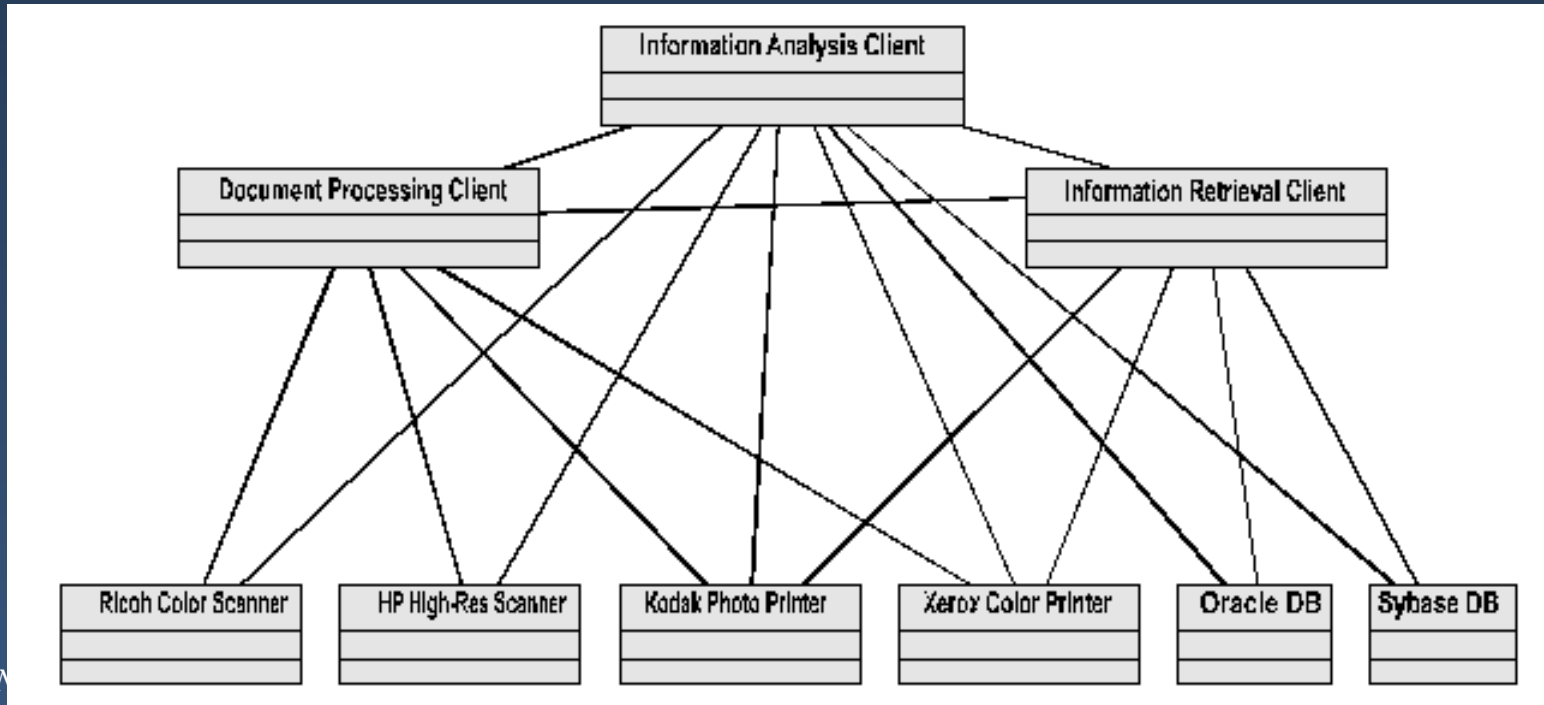
- **Refactored solution**
  - Provide guidelines for using complicated standards or interfaces
  - Provide a template for exception handling
  - Contract interfaces

Autogenerated Stovepipe

- Problem
  - Migrating an existing system to a distributed system
  - Converting existing software interfaces to distributed interfaces
  - Existing interfaces use fine- grain data
  - Implementation- specific subsystem interdependencies
- Solution
  - Reengineer interfaces
  - Define a separate, larger- grain object model
  - The interoperability among subsystems constitutes the core of the new design
  - Aim for stable interfaces; even more important for distributed systems than for standalone systems

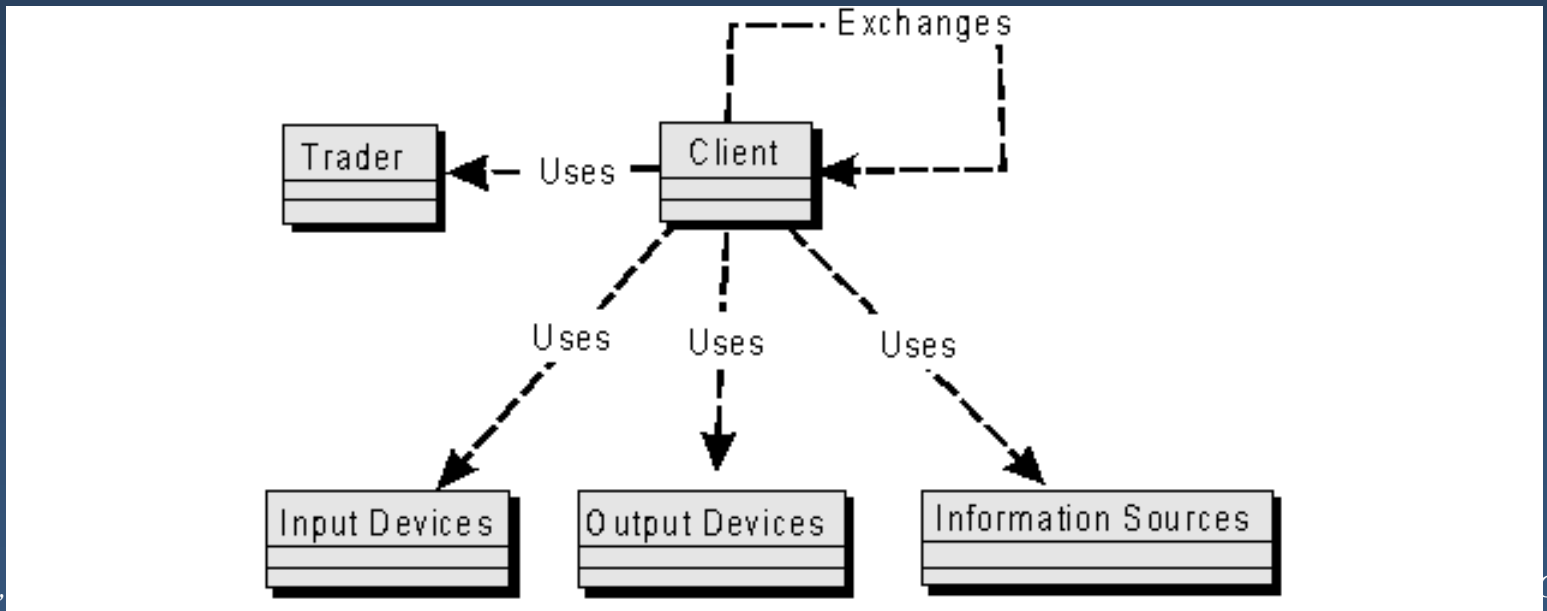
Stovepipe Enterprise

- Problem
  - Islands of automation
  - Stovepipe Enterprise is characterized by a software structure that inhibits change
  - Must be constantly repaired
  - Changes are done one island at a time
  - Brittle, monolithic system architectures (usually undocumented)
  - Inability of systems to interoperate



Stovepipe Enterprise

- Solution
  - Product lines
  - Identify requirements for the enterprise
  - Identify specification documents for the enterprise
  - Coordination of technologies at several levels
  - Identify common standards and migration direction with a standard reference model
  - Usage conventions across systems
  - Detailed interoperability conventions across systems



Design by Committee

- Problem
  - Gold Plating, Standards Disease, Make Everybody Happy, Political Party
  - project team are egalitarian
  - everyone has equal say
  - decisions are democratic
- The majority rule leads to diffusion of abstraction and excess complexity
- “A camel is a horse designed by a committee.”

Design by Committee

- Symptoms
  - Design documentation is voluminous
  - The requirements do not converge and are unstable
  - Design meetings are slow, concentrate on details, and avoid big picture discussions
  - Decisions are only made in meetings
  - No prioritization of design features

Design by Committee …

- Causes

  - No designated project architect
  - Ineffective meeting facilitation
  - The suggestions of all committee members are incorporated to keep everybody happy
  - No separation of concerns

Design by Committee …

- Refactored solution
  - Reform the meeting process
  - Why are we here?
  - What outcomes do we want?
  - Assign explicit roles
  - Owner, facilitator, architect, developer, tester, domain expert

"My specialty is being right when other people are being wrong."
(*George Bernard Shaw)*

Reinvent the Wheel

- Problem
  - *"Our problem is unique"*
  - Developers have minimal knowledge of each other's code
  - Building systems from the ground up even though related legacy systems exist
  - The existence of legacy systems is the norm rather than the exception
  - Lack of program families or product lines

Reinvent the Wheel

- Symptoms

  - Closed system architectures: no provision of reuse, interoperability, or change management
  - Replication of COTS components
  - Inability to deliver desired features on time and within budget
  - Corporate knowledge is not leveraged

Reinvent the Wheel …

- Causes

  - No communication and technology transfer among software development projects
  - Corporate knowledge is not leverage
  - No explicit architecture process
  - Lack of enterprise management

Vendor Lock-in

- **Symptoms**
  - "Our architecture is CORBA, Microsoft, and Oracle."
  - "We don't have an architecture."
  - "We completely depend on vendor X."

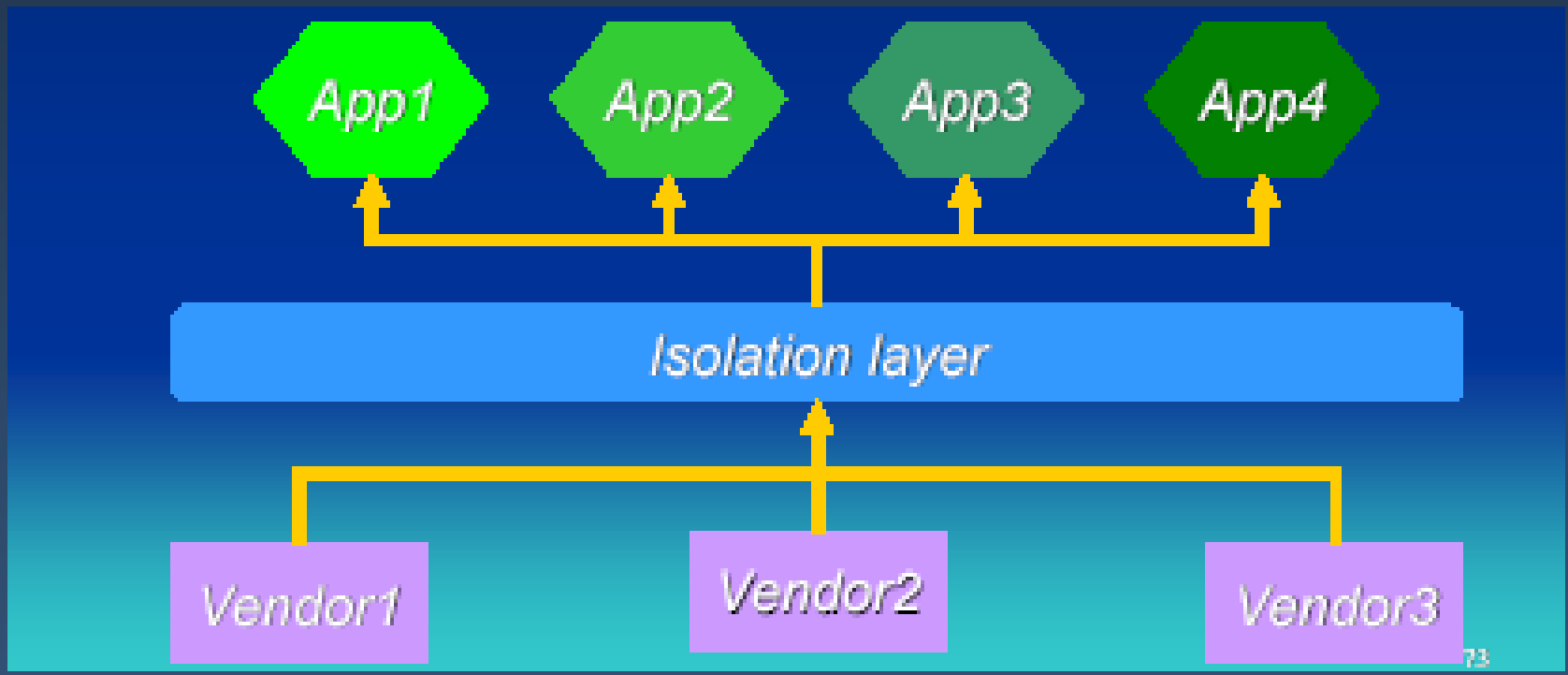
Vendor Lock-in

Problem

- Loss of control
  - The product does not live up to expectations
  - The features you need are always six months away
  - The vendor changed the product and broke your software
- Connector conspiracy
  - Vendors products hardly interoperate
  - Product versions proliferate
  - Only certain versions work together, but not the ones you bought

Vendor Lock-in

- Solution
  - Isolation layer or firewall



Software Management AntiPatterns

- Blowhard Jamboree
- Analysis Paralysis
- Viewgraph Engineering
- Death By Planning
- Fear of Success
- Corncob
- Intellectual Violence
- Irrational Management
- Smoke and Mirrors
- Project MisManagement
- Throw it over the Wall
- Fire Drill
- The Feud
- E-mail is dangerous

Analysis Paralysis

Classic AntiPattern

- "We need to redo this analysis to make it more object-oriented."
- "We need to complete OOA&D before we can start coding."
- "The design is not sufficiently detailed."
- "I have to know a lot more about the system before I can change anything."

Analysis Paralysis

- Symptoms
  - "Above all, strive for consistency and completeness"
  - Multiple project restarts—now we know enough/more and can do it right this time
  - Source code holds the truth and the design
  - Overly complex analysis models
  - Identify design patterns at all cost
  - Waterfall process model is followed

Analysis Paralysis

- Causes
  - Waterfall process model is followed
  - Management has more confidence in analysts than implementers
  - The goals of the analysis phase are not well defined
  - Micromanagement

Analysis Paralysis

- Refactored solution
  - Incremental development
  - Use spiral process model instead of waterfall model: design a little, build a little
  - Risk management
  - Internal (middleware) and external (user-visible functionality) increments

Corncob

- Symptoms
  - Frequently “difficult” people obstruct and divert the software development process
- Causes
  - Stress
  - Personality
  - Hidden agendas
    - On Wall Street 75% of programmer compensation is incentive bonus
    - In most organizations, project managers are competing with each other
    - Negative training or background
  - Defensiveness: fear of the unknown
  - Intellectual arsenic: obsession with a pet idea

Corncob

- Refactored solution
  - Responsibility: you raised the issue, fix it
  - Corrective interview
  - Pizza party
  - Stress reduction
  - Reform policies and procedures
  - Reorganization
  - Termination

Summary

- During maintenance and evolution one should be particularly aware of the potential presence of AntiPatterns
- Awareness of AntiPatterns is critical for reengineering projects
- Consider AntiPatterns next time you sign on to a new project
- Invest in reading the AntiPatterns book