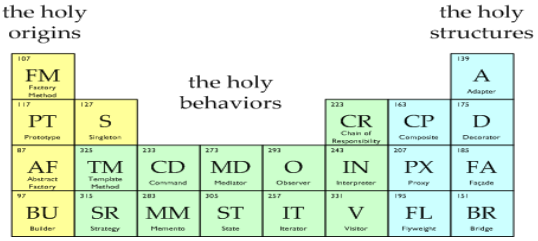
Antipatterns

Commonly reinvented bad solutions to problems

(textbook p.606)

1. On Design Patterns

The Sacred Elements of the Faith



The Pattern Zoo (pp.604-5)

- 23 Gamma Patterns
- 17 Buschmann Architectural Patterns
- 72 Analysis Patterns (Fowler)
- 38 CORBA Design Patterns
- 42 Antipatterns
- And more (e.g., for testing, UI, RT, process, etc.): e.g., top 5 Anti for Continuous Delivery!
- And now: elemental patterns (see Appendix 1)

Statistics from: http://www.serve.com/hibc/briefing/sld043.htm

2. What Are Antipatterns?

- "Negative Solutions," or solutions that present more problems than they address.
- Natural extensions to design patterns
- Provide knowledge to prevent and recover from common mistakes.
- But is this really worth it?

Check out: http://antipatterns.com/

Software Development Statistics

- Five out of six software projects are considered unsuccessful
- Nearly 1/3 of software projects are outright canceled
- 2/3 of delivered software was typically twice the expected budget and took twice as long to developed as originally planned
  - Such projects are known as Death March Projects (Yourdon, 2003)

Why Study Antipatterns

- Antipatterns provide easily identifiable templates for common problems, as well as a path of action to rectify these problems.
- Antipatterns provide real world experience in recognizing recurring problems in the software industry providing a detailed remedy for the most common ones.
- Antipatterns provide a common vocabulary for identifying problems and discussing solutions.
- Antipatterns provide stress release in the form of shared misery ☺
- Antipatterns ensure common problems are not continually repeated within an organization

Reference Model

The reference Model helps us gain an understanding of what causes particular Antipatterns, and the scope at which they occur. This information helps us better understand the problem, and why it continues to happen.

The Reference Model is broken down in three main areas:

- Root causes
  - provide fundamental context for the AntiPattern
- Primal forces
  - are the key motivators for decision making
- Software design-level model
  - defines at what level the problem applies
  - See Appendix 2 for SLDM levels, as well as a template for defining an antipatterns

Root Causes: Seven Deadly Sins

- Haste:
  - Tight deadlines often lead to neglecting important activities
- Apathy:
  - The attitude of not caring about solving known problems.
- Narrow-Mindedness:
  - Refusal of developers to learn proven solutions.
- Sloth:
  - Adaptation of the most simple "solution" (regardless of reqs).
- Avarice:
  - Greed in creating a system can result in very complex, and difficult to maintain software.
- Ignorance:
  - The lack of motivation to understand things.
- Pride:
  - The Failure to reuse existing software packages because they were not invented by a specific company

Primal Forces: Key Motivators of decision making

- 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 software
- Management of IT Resources
  - Managing people and IT artifacts
- Management of Technology Transfer
  - Controlling technology evolution

Viewpoints in s/w development

- *The Manager Viewpoint*:
  - The viewpoint of those who manage teams, projects, and programs. This viewpoint is related to the responsibility for planning and scheduling across the software development lifecycle. Management is often responsible for the software development lifecycle used, including the software configuration management.
- *The architect viewpoint*:
  - focuses on identifying the technology, and specifying system configurations and architecture. This covers both logical and physical representations of the architecture and establishing the scope of the use of the selected technologies. During implementation the role of the architect involves ensuring designs conform to the architecture and updating the architecture based on coding discoveries made during implementation.
- *The developer viewpoint*:
  - focuses on the implementation of the software development process. A developer can be anyone who has a role in mainstream development or implementation of a software system. Such roles include gathering requirements, designing, coding, and testing.

Types of Antipatterns

- Development Antipatterns
  - Problems encountered by programmers
- Architecture Antipatterns
  - Common problems in system structure
- Management Antipatterns
  - Problems in communication

3. What are Development Antipatterns?

- Common pitfalls in the development process
- The goal of identifying Development Antipatterns is to help to identify what to refactor in your system.

Development Antipatterns (1)

- Ambiguous Viewpoint
- Boat Anchor
- Continuous Obsolescence
- Cut and Paste Programming
- Dead End
- Fire Drill
- Functional Decomposition

List from: http://www.serve.com/hibc/dev_cat.htm

Development Antipatterns (2)

- Golden Hammer (see page 606)
- Input Kludge
- Lava Flow
- Mushroom Management
- Poltergeists
- The Blob (see separate presentation)
- Walking Through a Mine Field

List from: http://www.serve.com/hibc/dev_cat.htm

A Few Examples: The Blob

- Synopsis - Procedural-style design leads to one object with numerous responsibilities and most other objects only holding data.
- Refactored Solution- Refactor the design to distribute responsibilities more uniformly and isolate the effect of changes.

Example from: http://www.serve.com/hibc/dev_cat.htm

A Few Examples: Poltergeists

- Synopsis- Small Classes with very limited responsibilities and short life cycles.
- Refactored Solution- Allocate the Responsibility to Larger Objects and eliminate the Poltergeists.
- Blob (too few) and Poltergeists (too many) are at the opposite ends of a spectrum

Example from: http://www.serve.com/hibc/dev_cat.htm

A Few Examples: Cut and Paste Programming

- Synopsis- Code reused by copying source statements leads to significant maintenance problems.
- Refactored Solution- Black Box reuse (i.e., reuse of interface, typically through composition) reduces maintenance issues.

Example from: http://www.serve.com/hibc/dev_cat.htm

4. What are Architecture Antipatterns?

- Architectural AntiPatterns focus on some common problems and mistakes in the creation, implementation, and management of architecture.

Architectural Antipatterns (1)

- Architecture by Implication
- Auto generated Stovepipe
- Cover Your Assets
- Design by Committee
- Intellectual Violence
- Jumble
- Reinvent the Wheel
- Spaghetti Code

List from: http://antipatterns.com/dev_cat.htm

Architectural Antipatterns (2)

- Stovepipe Enterprise
- Stovepipe System
- Swiss Army Knife (see separate page)
- The Grand Old Duke of York
- Vendor Lock-In
- Warm Bodies
- Wolf Ticket

List from: http://antipatterns.com/dev_cat.htm or http://sourcemaking.com/antipatterns/

A Few Examples: Reinvent the Wheel

- Synopsis- Legacy systems with overlapping functionality. Every system built in isolation.
- Refactored Solution- Take advantage of existing, tested, and available systems

Example from: http://antipatterns.com/dev_cat.htm

A Few Examples: Vendor Lock in

- Synopsis- Proprietary, product-dependent architectures do not manage complexity and lead to a loss of control of the architecture and maintenance costs.
- Refactored Solution- Providing an isolation layer between product-dependent interfaces and the majority of application software enables management of complexity and architecture.

Example from: http://antipatterns.com/dev_cat.htm

A Few Examples: Stovepipe System

- Synopsis- Ad hoc integration and lack of abstraction lead to brittle, un-maintainable architectures
- Refactored Solution- Proper use of abstraction, subsystem facades, and metadata leads to adaptable systems.

Example from: http://antipatterns.com/mgmt_cat.htm

5. What are Management Antipatterns?

- Areas where human communication can be destructive to the software process
- The purpose of management AntiPatterns is to develop awareness that enables you to increase your success.

Management Antipatterns (1)

- Analysis Paralysis
- Blowhard Jamboree
- Corncob
- Death By Planning
- Email is dangerous
- Fear of Success
- Irrational management

List from: http://antipatterns.com/mgmt_cat.htm

Management Antipatterns (2)

- Project Mis-Management
- Smoke and Mirrors
- The Feud
- Throw it over the Wall
- Viewgraph Engineering

List from: http://antipatterns.com/mgmt_cat.htm

A Few Examples: Analysis Paralysis

- Synopsis- Striving for perfection and completeness in the analysis phase leads to project gridlock.
- Refactored Solution- Use an Incremental, iterative development processes. Defer the detailed analysis until the knowledge is available.

Example from: http://antipatterns.com/mgmt_cat.htm

A Few Examples: Corncob

- Synopsis- Frequently, difficult people obstruct and divert the software development process.
- Refactored Solution- Address agendas of the individual through various tactical, operational, and strategic organizational actions. That is, the team must be more important!

Example from: http://antipatterns.com/mgmt_cat.htm

A Few Examples: Fear of Success

- Synopsis- People (software developers included) do crazy things when a project is near successful completion.
- Refactored Solution- When project completion is close-at-hand, a clear declaration of success is important for the project environment.

Example from: http://antipatterns.com/mgmt_cat.htm

A Few Examples: Smoke and Mirrors

- Synopsis- End-users mistakenly assume that a brittle demonstration is a capability that is ready for operational use.
- Refactored Solution- Practice of proper ethics is important to manage expectations, risk, liabilities, and consequences in computing sales and marketing situations.

Example from: http://antipatterns.com/mgmt_cat.htm

Architectural or Management? Cover Your Assets

- Synopsis- Document driven software processes often employ authors who list alternatives instead of making decisions.
- Refactored Solution-Establish clear purposes and guidelines for documentation tasks; inspect the results for the value of documented decisions.

Example from: http://antipatterns.com/dev_cat.htm

Appendix 1
Elemental Patterns?

Elemental Patterns (1)

IMMUTABLE

- ***Context***:
  - An immutable object is an object that has a state that never changes after creation
- ***Problem***:
  - How do you create a class whose instances are immutable?
- ***Forces***:
  - There must be no loopholes that would allow 'illegal' modification of an immutable object
- ***Solution***:
  - Ensure that the constructor of the immutable class is the *only* place where the values of instance variables are set or modified.
  - Instance methods that access properties must not have side effects.
  - If a method that would otherwise modify an instance variable is required, then it has to return a *new* instance of the class.

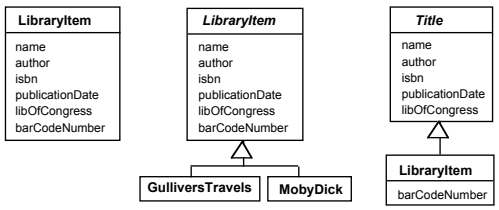
Elemental Patterns (2)

Delegation

- ***Context***:
  - You are designing a method in a class
  - You realize that another class has a method that provides the required service
  - Inheritance is not appropriate (e.g., not is-a relationship)
- ***Problem***:
  - How can you most effectively make use of a method that already exists in the other class?
- ***Forces***:
  - You want to minimize development cost by reusing methods

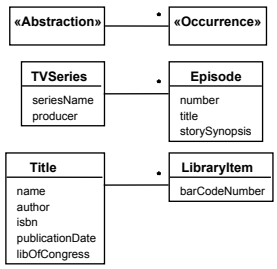
Elemental 3a: Abstraction-Occurrence

- Problem: A book can have several copies
- Antipatterns?



Elemental 3b: Abstraction-Occurrence

Solution:



Elemental 4a: The Player-Role Pattern

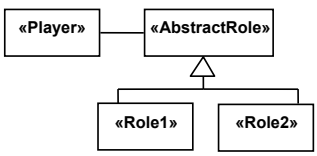
- ***Context***:
  - A *role* is a particular set of properties associated with an object in a particular context.
  - An object may *play* different roles in different contexts.
- ***Problem***:
  - How do you best model players and roles so that a player can change roles or possess multiple roles?

Elemental 4b: Player-Role

- Antipatterns:
  - Merge all the properties and behaviours into a single «Player» class and not have «Role» classes at all.
  - Create roles as subclasses of the «Player» class.

Elemental 4c: Player-Role

- ***Forces***:
  - It is desirable to improve encapsulation by capturing the information associated with each separate role in a class.
  - You want to avoid multiple inheritance.
  - You cannot allow an instance to change class
- ***Solution:***



Appendix 2
SLDM levels
&
Antipattern Template

Software Design Level Model (SDLM):

- *Global scale:*
  - involves design issues that are globally applicable across all systems. This level is concerned with coordination across all organizations that participate in information sharing.
- *Enterprise scale:*
  - focuses on coordination and communication across a single organization. The organization can be distributed across many locations.
- *System scale:*
  - deals with communications and coordination across applications and sets of applications.

SDLM (cont.)

- *Application scale:*
  - focuses upon the organization of applications developed to meet a set of user requirements.
- *Framework scale:*
  - focuses on the organization and development of application frameworks.
- *Micro-architecture scale:*
  - centered on the development of software components that solve recurring software problems.
- *Object scale:*
  - concerned with the development of reusable objects and classes. The object level is more concerned with code reuse than design reuse.

AntiPattern Template (1)

- *Name:*
  - The formal name of the AntiPattern
- *Also Known As:*
  - Other popular, descriptive, or humorous names for the AntiPattern.
- Most Frequent Scale:
  - Where the AntiPattern fits into the SDLM Model
- *Refactored Solution Name:*
  - The name of the pattern that acts as the proper Refactored solution.
- *Refactored Solution Type:*
  - This will identify the type of improvement that results from applying the AntiPattern solution. Such improvements include: Software, Technology, Process, and Role improvements.

AntiPattern Template (2)

- *Root Causes*:
  - One or more key root causes that result in the AntiPattern.
- *Unbalanced Forces:*
  - Identifies the Primal Forces that are ignored, misused, or overused in the AntiPattern.
- *Anecdotal Evidence:*
  - Common phrases and humorous anecdotes that describe the problem.
- *Background:*
  - Sets the Scene for the AntiPattern and introduces the problem under discussion.
- *General Form:*
  - General characteristics of the AntiPattern are identified, and an overview of the nature of the problem is presented.

AntiPattern Template (3)

- *Symptoms and Consequences:*
  - A list of symptoms and related consequences resulting from this AntiPattern.
- *Typical Causes:*
  - A list of the unique causes of an AntiPattern, which should relate to corresponding symptoms and consequences where possible.
- *Known Exceptions:*
  - Specific occasions when AntiPattern behavior and processes may not always be wrong.
- *Refactored Solutions:*
  - Resolves the unbalanced forces, causes, symptoms, and consequences of the AntiPattern.

AntiPattern Template (4)

- *Variations:*
  - Lists known variations of the AntiPattern,
- *Example:*
  - An example of the AntiPattern based on real-world experience
- *Related Solutions:*
  - Identifies and lists any cross-references to other Antipatterns which are closely related.
- *Applicability to Other Viewpoints and Scales:*
  - Describes the impact of the AntiPattern to other applicable SDLM scales.