Recognizing And Understanding AntiPatterns In Java Application Development

ExxonMobil Research & Engineering Co.
Clinton, New Jersey

Michael P. Redlich
(908) 730-3416
michael.p.redlich@exxonmobil.com

About Myself

- **Degree**
  - **B.S. In Computer Science**
  - **Rutgers University (Go Scarlet Knights!)**
- **ExxonMobil Research & Engineering**
  - **Senior Research Technician (1988-1998, 2004-Present)**
  - **Systems Analyst (1998-2002)**
- **Ai-Logix, Inc.**
  - **Technical Support Engineer (2003-2004)**
- **ACGNJ**
  - **Java Users Group Leader**
- **Publications**
  - **James: The Java Apache Mail Enterprise Server**
    - **Co-Authored With Barry Burd**
    - **Java Boutique**

What are Design Patterns?

- **Recurring solutions to software design problems that are repeatedly found in real-world application development**
- **Are about the design and interaction of objects**

Gang of Four (GoF)

- **Erich Gamma**
- **Richard Helm**
- **Ralph Johnson**
- **John Vlissides**
- **Design Patterns – Elements of Reusable Object-Oriented Software**
  - **ISBN 0-201-63361-2**
  - **1995**

What are AntiPatterns?

- **Opposite of a design pattern**
- **A classified bad design**
- **Designed for the developer to understand problems with bad solutions**
- **Serve two important purposes:**
  - **To Help Identify Problems**
  - **To Help Implement Solutions**
- **Next generation of design patterns research**
- **A design pattern becomes an antipattern when it causes more problems than it solves**

Anti Gang of Four (AGoF)???

- **William H. Brown**
- **Raphael C. Malveau**
- **Hays W. "Skip" McCormick III**
- **Thomas J. Mowbray**
- **AntiPatterns – Refactoring Software, Architectures, and Projects in Crisis**
  - **ISBN 0-471-19713-0**
  - **1998**

Evolution of AntiPatterns

- **The intended range and scope of using design patterns was never fully expressed**
- **1977**
  - **Christopher Alexander documented a pattern language for the planning of towns and buildings**
- **1987**
  - **Ward Cunningham and Kent Beck developed a design pattern language for SmallTalk**
- **1994**
  - **Hillside Group hosted the first industry conference on design patterns, Pattern Languages of Program Design (PLoP)**
- **1995**
  - **The Gang of Four released "Design Patterns – Elements of Reusable Object-Oriented Software**

Evolution of AntiPatterns (continued)

- **The tremendous growth of design patterns also had a "dark side"**
- **1996**
  - **Michael Akroyd presented "AntiPatterns: Vaccinations against Object Misuse" at the 1996 Object World West conference**
- **The usefulness of antipatterns began almost in parallel with design patterns**

How AntiPatterns Happen

- **A manager or developer:**
  - **Not knowing any better**
  - **Not having sufficient knowledge or experience**
  - **Applying a perfectly good design pattern in the wrong context**

Using AntiPatterns

- **Don’t use them in a destructive manner**
- **The absence of antipatterns does not guarantee success**
- **Don’t need to address every antipattern to be successful**
- **“If it’s not broken, don’t fix it”**
- **Attempting to correct several antipatterns simultaneously is risky**
- **The purpose is to develop strategies that fix problems as they arise**
- **Implement an antipattern solution only if the technical staff has the required skills**

AntiPattern Types

- **Software Development**
- **Software Architecture**
- **Software Project Management**

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***

- *** denotes Mini-AntiPattern**

Spaghetti Code

- **Background**
  - **Classic and most famous antipattern**
  - **Has existed in some form since the early days of programming languages**
  - **Structured programming languages are most susceptible**
- **General Form**
  - **A program that lacks real structure**
  - **The software structure is compromised to the extent that the structure lacks clarity**
  - **Having a small number of objects with very large implementations**
  - **Software that is very difficult to maintain and extend**
  - **No opportunity for code reuse**

Spaghetti Code (continued)

- **Symptoms and Consequences**
  - **Methods are process-oriented**
  - **There are minimal relationships among objects**
  - **The software quickly reaches a point of diminishing returns**
- **Typical Causes**
  - **Inexperience**
  - **No mentoring**
  - **Ineffective code reviews**
  - **No design**

Spaghetti Code (continued)

- **Refactored Solution**
  - **Refactoring, refactoring, refactoring!**
  - **Prevention is the best way to resolve this antipattern**
  - **Refactor spaghetti code to a more maintainable form**
    - **Use getter/setter methods**
    - **Resist the cut-and-paste antipattern**
    - **Reorder method/function arguments for better consistency**
    - **Remove portions of code that may be (or already are) inaccessible**
    - **Rename classes, methods/functions, and data types to conform to an industry standard**

Cut-and-Paste Programming

- **Background**
  - **A very common, yet degenerate form of software reuse**
  - **Has good software instincts, but this technique can be over used**
- **General Form**
  - **Identified by the presence of many similar code snippets interspersed throughout a software project**
  - **Programmers who are learning how to develop software from the more experienced programmers**
  - **Code duplication**

Cut-and-Paste Programming (continued)

- **Symptoms and Consequences**
  - **The same bug(s) reoccur throughout the application**
  - **It becomes difficult to locate and fix all instances of a mistake**
  - **Leads to excessive software maintenance costs**
- **Typical Causes**
  - **Excessive effort to create quality reusable code**
  - **The intent behind a software module is not preserved with the code**
  - **Reusable components are not sufficiently documented**
  - **Unfamiliarity with new technology or tools**

Cut-and-Paste Programming (continued)

- **Refactored Solution**
  - **Modify code to emphasize black-box reuse**
  - **Effective code refactoring requires three stages:**
    - **Code mining**
    - **Refactoring**
    - **Configuration management**

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***

- *** denotes Mini-AntiPattern**

Vendor Lock-In

- **Background**
  - **WYSISLWYG syndrome**
  - **Difficult to avoid due to an organizational dependence of a vendor's products**
- **General Form**
  - **A software project adopts a vendor's product technology and becomes completely dependent on it**
  - **Problems may arise due to vendor product upgrades**

Vendor Lock-In (continued)

- **Symptoms and Consequences**
  - **Commercial vendor product upgrades dictate the software maintenance cycle**
  - **Features promised by the vendor are delayed or never delivered**
  - **A vendor's product upgrade varies significantly from the advertised open systems standard**
- **Typical Causes**
  - **The vendor's product is selected based solely on marketing and sales information**

Vendor Lock-In (continued)

- **Refactored Solution**
  - **Isolation layer**
    - **Separates application software from vendor product-dependent interfaces**
  - **The isolation layer solution can be used under the following conditions:**
    - **Isolation of application software from lower-level infrastructure**
    - **Anticipated changes to the infrastructure**
    - **A more convenient programming interface is useful or necessary**
    - **There is a need for consistent infrastructure handling across many systems**

Reinvent The Wheel

- **Background**
  - **Software reuse and design reuse are significantly different paradigms**
  - **Greenfield System (alias of Reinvent the Wheel)**
- **General Form**
  - **Custom software is built from the ground up**
  - **Software reuse is limited and interoperability is accommodated after the fact**
  - **Greenfield System Assumptions**
    - + **Eventually become stovepipes**
    - + **Mismatched to most real-world software development challenges**

Reinvent The Wheel (continued)

- **Symptoms and Consequences**
  - **Closed system architectures**
  - **Replication of commercial software functions**
  - **Inadequate support for change management**
- **Typical Causes**
  - **No communication and technology transfer between software projects**
  - **Assumption that the software project will be built from scratch**
  - **Absence of an explicit architecture process**

Reinvent The Wheel (continued)

- **Refactored Solution**
  - **Architecture mining**
    - + **Valuable information can be found in precursor designs:**
      - Legacy systems, commercial products, standards, prototypes, design patterns
    - + **It is typical to find about six precursor designs**
    - + **Bottom-up design approach**

Software Project 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***

- *** denotes Mini-AntiPattern**

Analysis Paralysis

- **Background**
  - **A misconception that designs never fail, only implementations**
  - **Prolonging the analysis and design phases avoids risking accountability for results**
- **General Form**
  - **Occurs when the goal is to achieve perfection and completeness of the analysis phase**
  - **Characterized by turnover and revision of models**
  - **Usually involves waterfall assumptions**
  - **The analysis documents no longer make sense to the domain experts**

Analysis Paralysis (continued)

- **Symptoms and Consequences**
  - **Multiple project restarts**
  - **Cost of analysis exceeds expectation**
  - **The analysis no longer involves user interaction**
  - **The complexity of the analysis results in intricate implementations**
- **Typical Causes**
  - **Managers over specify and over supervise assignments**
  - **Management has more confidence in their ability to analyze and decompose as opposed to design and implement**
  - **Goals in the analysis phase are not well-defined**

Analysis Paralysis (continued)

- **Refactored Solution**
  - **Incremental development is key to success of object-oriented development**
    - **All phases of the process occur with each iteration**
      - Analysis, design, coding, test, validation
    - **Internal increment**
    - **External increment**

Project Mismanagement

- **Background**
  - **Concerns the monitoring and controlling of software projects**
  - **Occurs after planning and during the analysis, design, construction, and testing of a project**
- **General Form**
  - **Key activities are often overlooked or minimized**
    - **Technical planning**
    - **Quality control**
    - **Inadequate architecture definition and test coverage**
    - **Insufficient code reviews**

Project Mismanagement (continued)

- **Symptoms and Consequences**
  - **The design is difficult to implement due to a lack of an architectural strategy**
  - **Code reviews happen infrequently**
  - **The test design requires extra effort due to an inadequately defined behavioral guideline**
- **Typical Causes**
  - **The technical criteria for code inspection, testing, integration, and interoperability due to an inadequate architecture**

Project Mismanagement (continued)

- **Refactored Solution**
  - **Three categories of proper risk management**
    - **Managerial**
      - Processes
      - Roles
    - **Common Project Failure Points**
      - Cost overruns
      - Premature project termination
      - Development of the wrong project
      - Technical failure

Project Mismanagement (continued)

- **Refactored Solution (continued)**
  - **Three categories of proper risk management (continued)**
    - **Quality**
      - Program and project management
      - Product identification
      - Architecture definition
      - Solution design
      - Solution implementation
      - Solution validation
      - Product support

Resources

- **Design Patterns – Elements of Resusable Object-Oriented Software**
  - **Erich Gamma, et. al**
  - **ISBN 0-201-63361-2**
- **AntiPatterns – Refactoring Software, Architectures, and Projects in Crisis**
  - **William H. Brown, et. al**
  - **ISBN 0-471-19713-0**
- **An Introduction to AntiPatterns in Java Applications**
  - **Puneet Sangal**
  - **http://www.devx.com/Java/Article/29162/1954/**