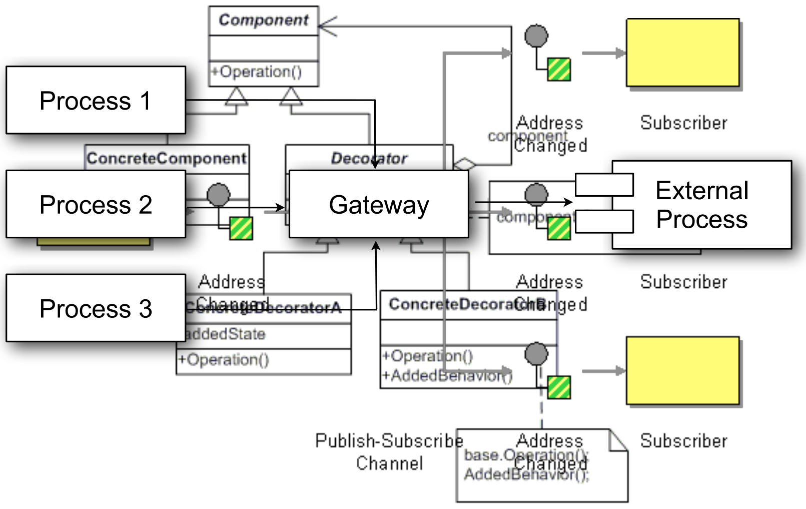
Common Anti-Patterns
And How To Avoid Them

Mark Richards
Director and Sr. Architect, Collaborative Consulting, LLC
Author of Java Message Service 2nd Edition (O'Reilly)

patterns

repeatable processes that produce positive results



anti-patterns

things we repeatedly do that produce negative consequences

there are lots of anti-patterns...

Obligatory subcontracting
Funding me-too research
Repackaging as original
Analysis paralysis
Cash cow
Cost migration
Crisis mode
Design by committee
Escalation of commitment
Management by neglect
Management by numbers
Management by perkele
Management by wondering
Milk Monitor Promotion
Moral hazard
Mushroom management
Stovepipe
Vendor lock-in
Violin string organization
Puppet programming
Copy and paste programming
De-factoring
Golden hammer
Improbability factor
Low hanging fruit
Not built here
Premature optimization
Programming by permutation

Reinventing the square wheel
Reinventing the wheel
Silver bullet
Copper bullet
Tester Driven Development
Hostile testing
Meta-testing
Moving target
Re-coupling
Nurses-auditing-doctors
Turkish hat reform
Classpath hell
Dependency hell
DLL hell
Extension conflict
JAR hell
Magic Bullet
Chain Reaction
Ivory Tower
Buzzword-Driven Architecture
Death march
Groupthink
Smoke and mirrors
Software bloat
Bystander apathy
Napkin specification
Phony requirements
Retro-specification

Abstraction inversion
Ambiguous viewpoint
Big ball of mud
Blob
Gas factory
Input kludge
Interface bloat
Magic pushbutton
Race hazard
Railroaded solution
Re-coupling
Stovepipe system
Staralised schema
Anemic Domain Model
BaseBean
Call super
Circle-ellipse problem
Empty subclass failure
God object
Object cesspool
Object orgy
Poltergeists
Sequential coupling
Singletonitis
Yet Another Useless Layer
Yo-yo problem
Accidental complexity
Accumulate and fire

Action at a distance
Blind faith
Boat anchor
Bug magnet
Busy spin
Caching failure
Cargo cult programming
Checking type
Code momentum
Coding by exception
Error hiding
Expection handling
Hard code
Lava flow
Loop-switch sequence
Magic numbers
Magic strings
Monkey work
Packratting
Parallel protectionism
Ravioli code
Soft code
Spaghetti code
Wrapping wool in cotton
Many others...

and lots of categories as well...

Economical
Organizational
Project Management
Analysis
Software Architecture
Software Development

Methodological
Testing
Requirements Management
Quality Assurance
Configuration Management
Enterprise Architecture

and lots of categories as well...

Economical
Organizational
Project Management
Analysis
Software Architecture
Software Development

Methodological
Testing
Requirements Management
Quality Assurance
Configuration Management
Enterprise Architecture

common anti-patterns

let’s take a closer look at the following anti-patterns...

cargo cult programming	lava flow
object orgy	accidental complexity
golden hammer	the blob

cargo cult programming anti-pattern

using patterns, methods, and techniques without understanding why

cargo cult programming anti-pattern

using patterns, methods, and techniques without understanding why

```
if (year == 2009 || month.startsWith("M")) {
   System.out.println("true");
} else {
   System.out.println("false");
}
```

cargo cult programming anti-pattern

using patterns, methods, and techniques without understanding why

```
if (year == 2009 | month.startsWith("M")) {
   System.out.println("true");
} else {
   System.out.println("false");
}
```

cargo cult programming anti-pattern

using patterns, methods, and techniques without understanding why

```
@Transactional
public void placeOrder(Order order) {
    insertOrder(order);
    updateAccount(order);
    updateInventory(order);
}
```

Will this work? What exactly are the default values for the Spring @Transactional annotation?

avoidance techniques

don't use a framework, product, or technology without a reason for doing so

when you see some code you aren't sure of, take the time right then and there to understand it

take the time to read and understand about the technology or framework you are using

most importantly, **RTFM!!!** (**R**ead **T**he **F**____ **M**anual)

lava flow anti-pattern

obsolete technologies and forgotten extensions leave hardened globules of dead code in its wake

lava flow anti-pattern

obsolete technologies and forgotten extensions leave hardened globules of dead code in its wake

```
public void placeOrder(Order order) {
    insertOrder(order);
    updateInventory(order);

    //check for overstock discount and tax
    checkOverstockDiscount(order);
    //checkOverstockDiscount(order);
    calculateTax(order);
    //calculateTax(order);

    processPayment(order);
}
```

lava flow anti-pattern

obsolete technologies and forgotten extensions leave hardened globules of dead code in its wake

old reports

converted reports

avoidance techniques

leverage version control to safely remove old code, knowing it can easily be recovered if needed

test-driven development and meaningful regression tests (with code coverage tools) helps to avoid this anti-pattern

utilize open source and commercial tools to detect dead code (Eclipse, Aivosto, etc.)

the use of CDLs or interfaces can help avoid this anti-pattern

INDEX

object orgy anti-pattern

objects are insufficiently encapsulated, resulting in unrestricted access to their private parts

William Hogarth (1697-1764), *The Orgy*

object orgy anti-pattern

```
public class Account {
   public BigDecimal balance;
   public String name;
   ...
}
```

object orgy anti-pattern

```
public class Account {
   public BigDecimal balance;
   public String name;
   ...

   public BigDecimal getBalance() {
      if (balance == null) {
         return new BigDecimal(0);
      } else {
         return balance;
      }
   }

   public void setBalance(BigDecimal bal) {
      ...
   }
}
```

object orgy anti-pattern

```
public class Account {
   private BigDecimal balance;
   private String name;
   ...

   public BigDecimal getBalance() {
      if (balance == null) {
         return new BigDecimal(0);
      } else {
         return balance;
      }
   }

   public void setBalance(BigDecimal bal) {
      ...
   }
}
```

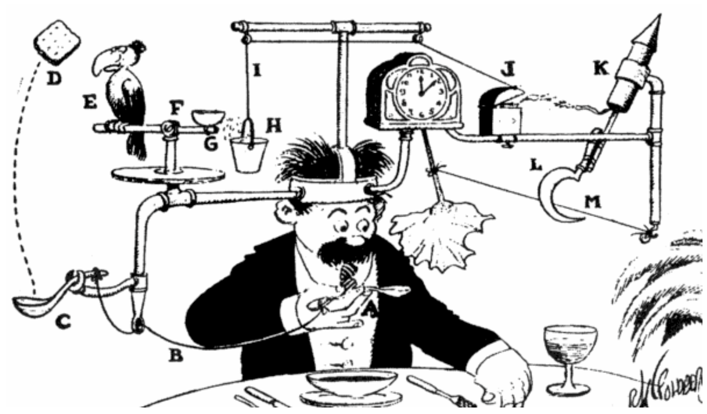
avoidance techniques

haste and apathy usually contribute to this anti-pattern - avoid the shortcuts and *always* use encapsulation

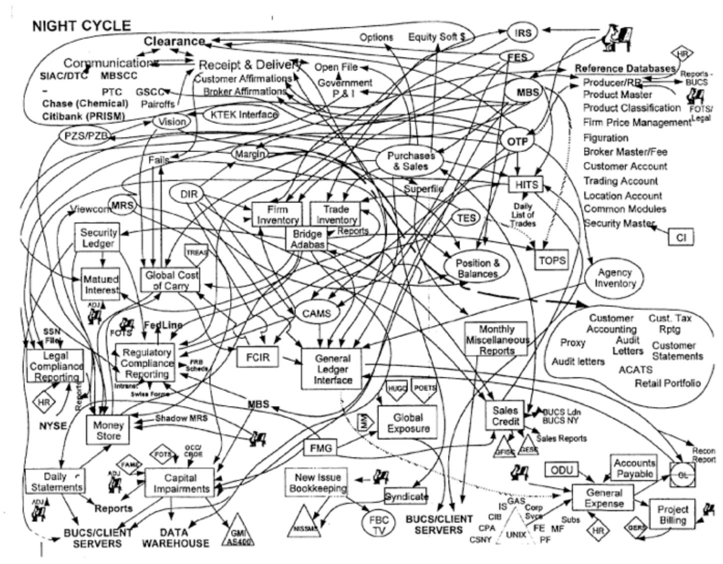
INDEX

accidental complexity anti-pattern

introducing non-essential complexity into a solution



accidental complexity anti-pattern



accidental complexity anti-pattern

```
public int adjustNumber(int x) {
   int y=x-(x-1) <= 0 ? 1 : x-(x-1);
   return x % ++y == 0
          ? x*++y/3*2 : ++x*--y-1;
}
```

accidental complexity anti-pattern

```
public int doubleIfEven(int x) {
   if (x % 2 == 0)
      return x*2;
   else
      return x;
}
```

accidental complexity anti-pattern

```
public int adjustNumber(int x) {
   int y=x-(x-1) <= 0 ? 1 : x-(x-1);
   return x % ++y == 0
          ? x*++y/3*2 : ++x*--y-1;
}

public int doubleIfEven(int x) {
   if (x % 2 == 0)
      return x*2;
   else
      return x;
}
```

accidental complexity anti-pattern

essential complexity: we have a hard problem

accidental complexity: we have made a problem hard

“developers are drawn to complexity like moths to a flame - frequently with the same result”

avoidance techniques

focus on the essential complexity and avoid “tricky code”

look for this anti-pattern in architecture, design, and coding - it exists in all three!

frequent code reviews! Make sure you can read the code your team members write

INDEX

golden hammer anti-pattern

using the same tool, product, or technique to solve every problem

golden hammer anti-pattern

using the same tool, product, or technique to solve every problem

Groovy

Bistro

Correlate

Clojure

Scala

JRuby

CAL

http://www.is-research.de/info/vmlanguages/

avoidance techniques

focus on *Java the Platform,* not *Java the Language*

embrace *polyglot programming*

avoid the "tower of babel" anti-pattern as a result!

END

the blob anti-pattern

an all encompassing class or component that knows too much and does too much

the blob anti-pattern

an all encompassing class or component that knows too much and does too much

a single class with a large number of attributes and/or methods (60 or more is a good sign of a "blob")

unrelated methods and attributes contained in a single class

the presence of a large "controller" class indicates a blob

the blob anti-pattern

factors that can lead to this anti-pattern...

lack of object-oriented skills on the team

lack of a solid software design and/or architecture

use of agile methodology techniques can sometimes lead to this anti-pattern!

avoidance techniques

use a roles and responsibility model

make sure your team members have the proper level of skill in object-oriented concepts

frequent code reviews can stop a “Blob” before it gets too big

INDEX

summary and q&a

references

- AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis by William J. Brown et.al. (Wiley)
- http://en.wikipedia.org/wiki/Anti-pattern
- http://www.antipatterns.com/EdJs_Paper/Antipatterns.html
- http://c2.com/cgi/wiki?AntiPatternsCatalog
- http://sourcemaking.com/antipatterns
- Complete Slides - http://www.wmrichards.com/slides