Software Development Anti-Patterns

Mark Richards
Hands-on Software Architect
Author of Enterprise Messaging Video Series (O'Reilly)
Author of Java Message Service 2nd Edition (O'Reilly)
Co-author of Software Architecture Fundamentals Video Series (O'Reilly)
patterns

repeatable processes that produce positive results
anti-patterns

things we repeatedly do that produce negative consequences

```java
public class AccountData {

    public long acctId;
    public String acctName;
    public double balance;

    public long getAcctId() {
        return this.acctId;
    }

    ...
}
```
anti-patterns

things we repeatedly do that produce negative consequences
what are the primary human traits that cause anti-patterns to occur?
so many categories...

Economical
Organizational
Project Management
Analysis
Software Architecture
Software Development

Methodological
Testing
Requirements Management
Quality Assurance
Configuration Management
Enterprise Architecture
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

and many others...
Software Development
AntiPatterns
software development anti-patterns

cargo cult programming

accidental complexity

parallel protectionism

lava flow

the blob
Cargo Cult Programming
AntiPattern
cargo cult programming anti-pattern

the practice of doing something without understanding the reasoning behind it
cargo cult programming anti-pattern

the practice of doing something without understanding the reasoning behind it
cargo cult programming anti-pattern

```java
if (year == 2012 || month.startsWith("M")) {
    System.out.println("true");
} else {
    System.out.println("false");
}
```
cargo cult programming anti-pattern

how would you parse this string into a String[] array?

String s = "BUY,AAPL,100,131.34";
cargo cult programming anti-pattern

```java
@.Transactional
public void placeOrder(Order order)
    throws AccountNotFoundException,
       NoInventoryException {
    insertOrder(order);
    updateAccount(order);
    updateInventory(order);
}

if one of the advertised exceptions is thrown, will the transaction be rolled back?
"the rate of technology change will be so high that existing skills will be as outdated as quill and parchment. We will need to constantly re-skill ourselves just to keep a job." (1999)
cargo cult programming anti-pattern

we simply don't have time to read and learn everything there is to know about every technology!
cargo cult programming anti-pattern
avoidance techniques
take responsibility to know your craft...
cargo cult programming anti-pattern
avoidance techniques
take time to learn the language...

workerThread.yield();
doc.setHeader(stdHeader.intern());

know these packages and classes extremely well...

java.lang.*
java.io.*
java.math.*
java.text.*
java.util.*
cargo cult programming anti-pattern avoidance techniques
cargo cult programming anti-pattern
avoidance techniques

RTFM!
(Read The F____ Manual)
Accidental Complexity
AntiPattern
accidental complexity anti-pattern

introducing non-essential complexity into a problem
accidental complexity anti-pattern
accidental complexity anti-pattern
accidental complexity anti-pattern

essential complexity
"we have a hard problem"

accidental complexity
"we have made a problem hard"
accidental complexity anti-pattern

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

- Neal Ford
accidental complexity anti-pattern
accidental complexity anti-pattern
avoidance techniques
ALL OF THESE PARENTHESES AND SEMICOLONS TOTALLY RUIN THE FENG SHUI OF THE CODE.

SALLY LEARNS AN IMPORTANT LESSON ABOUT INVITING THE RIGHT PEOPLE TO HER CODE REVIEWS.
accidental complexity anti-pattern

paired-programming
accidental complexity anti-pattern tool-based reviews
accidental complexity anti-pattern

over-the-shoulder code reviews

Joe O'Brien and Jim Weirich
Lava Flow AntiPattern
lava flow anti-pattern
dead code solidifies into hardened globules within fluid code, resulting in unnecessary maintenance
public void placeOrder(Order order) {
    insertOrder(order);
    updateInventory(order);
    //checkOverstockDiscount(order);
    //calculateStateTax(order);
    processPayment(order);
}

lava flow anti-pattern

dead code
lava flow anti-pattern

dead code

- dead class or interface
- dead function or method
- dead control statements
- dead parameter or return value
- dead variable
lava flow anti-pattern

dead artifacts

old reports

converted reports
lava flow anti-pattern
avoidance techniques

*leverage* version control
lava flow anti-pattern
avoidance techniques

*leverage* dead code tools

Eclipse IDE / IntelliJ IDE / NetBeans IDE
Google CodePro AnalytiX
UCDetector
PMD
DCD (Dead Code Detector)
lots more.....
Parallel Protectionism
AntiPattern
parallel protectionism anti-pattern

code becomes so complex and fragile that it is easier to clone it to make changes
public void nutation(double jd, double[] aNutLong, double[] aNutObliq) {
    double T = (jd - 2415020.0d) / 36525d;
    double L2 = 2d * radians(279.6967d + 0.000303d * sqr(T) + reduce((100.0021358d * T), 360d));
    double D2 = 2d * radians(270.4342 - 0.001133 * sqr(T) + reduce((1336.855231d * T), 360d));
    double M1 = radians(358.4758d - 0.00015d * sqr(T) + reduce((99.99736056d * T), 360d));
    double M2 = radians(296.1046d + 0.009192d * sqr(T) + reduce((1325.552359d * T), 360d));
    double N1 = radians(259.1833d + 0.002078d * sqr(T) - reduce((5.372616677d * T), 360d));
    double H = (-17.2327d - 0.01737d * T) * Math.sin(N1);
    H = H + (-1.2729d - 0.00013d * T) * Math.sin(L2) + 0.2088d;
    H = H - 0.2037d * Math.sin(D2) + (0.1261d - 0.00031d * T) * Math.sin(M1);
    H = H + 0.0675d * Math.sin(M2) - (0.0497d - 0.00012d * T) * Math.sin(L2 + M1);
    H = H - 0.0342d * Math.sin(D2 - N1) - 0.0261d * Math.sin(D2 + M2);
    H = H + 0.0124d * Math.sin(L2 - M1) + 0.0114d * Math.sin(D2 - M2);
    aNutLong[0] = H/3600d;
    H = (9.21d + 0.00091d * T) * Math.cos(N1);
    H = H + (0.5522d - 0.00029d * T) * Math.cos(L2) - 0.0904d;
    H = H + 0.0884d * Math.cos(D2) + 0.0216d * Math.cos(L2 + M1);
    H = H + 0.0183d * Math.cos(D2 - N1) + 0.0113d * Math.cos(D2);
    H = H - 0.0093d * Math.cos(L2 - M1) - 0.0066d * Math.cos(L2 - N1);
    aNutObliq[0] = H/3600d;
}
parallel protectionism anti-pattern
public void preciseNutation(double jd, double[] aNutLong, double[] aNutObliq) {
    double T = (jd - 2415020.0d) / 36525d;
    double L2 = 2d * radians(279.6967d + 0.000303d * sqr(T) + reduce((100.0021358d * T), 360d));
    double D2 = 2d * radians(270.4342 - 0.001133 * sqr(T) + reduce((1336.855231d * T), 360d));
    double M1 = radians(358.4758d - 0.00015d * sqr(T) + reduce((99.99736056d * T), 360d));
    double M2 = radians(296.1046d + 0.009192d * sqr(T) + reduce((1325.552359d * T), 360d));
    double N1 = radians(259.1833d + 0.002078d * sqr(T) - reduce((5.372616667d * T), 360d));
    double N2 = 2d * radians(N1);

    double H = (-17.2327d - 0.01737d * T) * Math.sin(N1);
    H = H + (-1.2729d - 0.00013d * T) * Math.sin(L2) + 0.2088d * Math.sin(N2);
    H = H - 0.2037d * Math.sin(D2) + 0.00013d * T) * Math.sin(M1);
    H = H + 0.0675d * Math.sin(M2) - (0.0497d - 0.00012d * T) * Math.sin(D2 - M2);
    H = H - 0.0342d * Math.sin(L2 - N1) - 0.0214d * Math.sin(L2 - D2 + M2);
    H = H + 0.0124d * Math.sin(L2 - M1) + 0.0114d * Math.sin(D2 - M2);
    aNutLong[0] = H/3600d;

    H = (9.21d + 0.00091d * T) * Math.cos(N1);
    H = H + (0.5522d - 0.00029d * T) * Math.cos(L2) - 0.0904d * Math.cos(N2);
    H = H + 0.0884d * Math.cos(D2) + 0.0216d * Math.cos(L2 + M1);
    H = H + 0.0183d * Math.cos(D2 - N1) + 0.0113d * Math.cos(D2 - N2);
    H = H - 0.0093d * Math.cos(L2 - M1) - 0.0066d * Math.cos(L2 - N1);
    aNutObliq[0] = H/3600d;
}

parallel protectionism anti-pattern
public void preciseNutation(double jd, double[] aNutLong, double[] aNutObliq) {
    double T = (jd - 2415020.0d) / 36525d;
    double L2 = 2d * radians(279.6967d + 0.000303d * sqr(T) + reduce((100.0021358d * T), 360d));
    double D2 = 2d * radians(270.4342 - 0.001133 * sqr(T) + reduce((1336.855231d * T), 360d));
    double M1 = radians(358.4758d - 0.00015d * sqr(T) + reduce((99.99736056d * T), 360d));
    double M2 = radians(296.1046d + 0.009192d * sqr(T) + reduce((1325.552359d * T), 360d));
    double N1 = radians(259.1833d + 0.002078d * sqr(T) - reduce((5.372616667d * T), 360d));
    double N2 = 2d * radians(N1);
    double H = (-17.2327d - 0.01737d * T) * Math.sin(N1);
    H = H + (-1.2729d - 0.00013d * T) * Math.sin(L2) + 0.2088d * Math.sin(N2);
    H = H - 0.2037d * Math.sin(D2) + (0.0497d - 0.00031d * T) * Math.sin(M1);
    H = H + 0.0675d * Math.sin(D2 - M1) + 0.0183d * Math.cos(D2 - N1) + 0.0113d * Math.cos(D2 - M2);
    H = H - 0.0093d * Math.cos(L2 - M1) - 0.0066d * Math.cos(L2 - N1);
    aNutLong[0] = H/3600d;

    H = (9.21d + 0.00091d * T) * Math.cos(N1);
    H = H + (0.5522d - 0.00029d * T) * Math.cos(L2 - 0.0904d * Math.cos(N2));
    H = H + 0.0884d * Math.cos(D2) + 0.216d * Math.cos(L2 + M1);
    H = H + 0.0183d * Math.cos(D2 - N1) + 0.0113d * Math.cos(D2 - N2);
    H = H - 0.0093d * Math.cos(L2 - M1) - 0.0066d * Math.cos(L2 - N1);
    aNutObliq[0] = H/3600d;
}
parallel protectionism anti-pattern

what is bad about this anti-pattern?

what is good about this anti-pattern?
parallel protectionism anti-pattern
parallel protectionism anti-pattern
The Blob AntiPattern
the blob anti-pattern
modules become so big they consume the entire application
the blob anti-pattern
modules become so big that they consume the entire application
the blob anti-pattern

ways to detect this anti-pattern

- a single class with a large number of attributes and/or methods
- unrelated methods and attributes contained in a single class
- lack of a solid software design, component design, and architecture
the blob anti-pattern

avoidance techniques

leverage the roles and responsibility model
the blob anti-pattern

stock trade order validation
the blob anti-pattern

stock trade order validation

responsible for receiving a trade order, dispatching it to the next available controller, and returning the formatted results to the caller.
the blob anti-pattern

stock trade order validation

responsible for orchestrating the trade order validation process and returning the results to the dispatcher.
the blob anti-pattern
stock trade order validation

responsible for making sure the trader isn't exceeding assigned trader limits with the order being placed.
the blob anti-pattern

stock trade order validation

responsible for making sure the trade order symbol isn't on the restricted stock list.
the blob anti-pattern

stock trade order validation

who should be responsible for retrieving and caching all of the common data needed by the compliance modules?
the blob anti-pattern

stock trade order validation

responsible for orchestrating the trade order validation process and returning the results to the dispatcher. also responsible for retrieving and caching all common data needed by the compliance modules
the blob anti-pattern
stock trade order validation

who should be responsible for persisting trade validation errors when they occur?
the blob anti-pattern

stock trade order validation

responsible for orchestrating the trade order validation process and returning the results to the dispatcher. also responsible for retrieving and caching all common data needed by the compliance modules and persisting all validation errors.
the blob anti-pattern
stock trade order validation
the blob anti-pattern

stock trade order validation
the blob anti-pattern

stock trade order validation

responsible for retrieving and caching all common data needed by the compliance modules and persisting all validation errors.
message dispatcher
compliance controller
data manager
trader limits
restriction

the blob anti-pattern
stock trade order validation

responsible for orchestrating the trade order validation process and returning the results to the dispatcher.
Software Development AntiPatterns

Mark Richards
Independent Consultant
Hands-on Software Architect
Published Author / Conference Speaker

http://www.wmrichards.com
http://www.linkedin.com/pub/mark-richards/0/121/5b9