AOP Framed!

Henrique Rebêlo

Informatics Center
Federal University of Pernambuco
Contacting Me…

Ph.D. student Henrique Rebêlo
Specialist on AOSD, DbC, Static Metrics, JML
Software Architecture, Product Lines, Empirical Studies
email: hemr@cin.ufpe.br

Informatics Center
Federal University of Pernambuco, UFPE
Brazil
Talk Material - Contributions

• Prof. Paulo Borba (expert on aspect-oriented languages and refactoring)
• Prof. Sérgio Soares (expert on aspect-oriented software development)
• Prof. Alessandro Garcia (expert on aspect-oriented software assessment metrics)
• Prof. Fernando Castor Filho (expert on exception handling)
talk timeline

- OO, concerns, tangle, scatter, crosscut
- AOP, AOSD, and AspectJ overview
- research with AOP
- AOP in industry
- conclusion

UCF 2009
exploring some of the terms we use

OO
scatter
tangle
crosscut
and
all
that
Object-Oriented Paradigm (OOP)

- object-oriented decomposition

- modularization
  - objects
  - classes
  - encapsulated data and procedures

Diagram showing the evolution of object-oriented programming languages:

- Simula 67
- Smalltalk
- Eiffel
- CLOS
- C++
- Java

Timeline:

- 1960
- 1965
- 1970
- 1975
- 1980
- 1985
- 1990
- 1995
- 2000
- 2005

Problem: ?
Bad OO Design

Problem: tangling of code with different purposes
Good OO Design

- Graphical user interface (GUI)
- Communication
- Business
- Data
Layered Architecture

- The various kinds of code should be written separately
  - Separation of
    - concepts
    - concerns
    - kinds of code

separation of concerns
Need for Separation of Concerns

Large complex distributed software systems

Development requires focusing on one concern at a time
Separation of Concerns

• A concern is a “specific requirement or consideration that must be addressed in order to satisfy the overall system goal”

• The principle was coined by Edsger Dijkstra
  – 1974 paper "On the Role of Scientific Thought"
  – he received the 1972 A. M. Turing Award for fundamental contributions in the area of programming languages
Effective Separation of Concerns improve...

Quality attributes

- stability
- evolvability
- composability
- reliability

- reusability
- traceability
- maintainability
- testability
- ...

Good modularization without persistence, distribution, ...
We have problems with persistence using JDBC

JDBC code in red...
What is the problem?

- Bad modularization of system-wide, peripheral concerns at the implementation level

- Symptoms
  - Code tangling
  - Code scattering
Problems with OO implementation

Code tangling

Code scattering
Problem with OO

- Code related to such concerns are crosscutting
- Scattered across several modules of implementation (classes)

Each color is one different concern
More OO Problems

- **Redundance**
  - several similar code fragments in several places of the source code

- **Weak cohesion**
  - methods of the affected classes contain instructions that are not directed related to the core concern in which they implement

- **Strong coupling**
  - methods of the affected classes need to be aware of methods from classes which implement the scattered functionality
The Problem of Crosscutting Concerns

- Certain properties naturally have global, pervasive effects over the system decompositions
  - examples: exception handling, distribution, persistence…

OO Decomposition

- Hamper software reusability, maintainability, and the like
- Domain-specific programming languages started to emerge…
Persistence is a crosscutting concern
Crosscutting concerns...

cannot be properly modularized by OO constructs because their realization affects the behavior of several methods, possibly in several classes, cutting across class structures
Tangling and scattering of...
Generalizing the Problem…

Data Classes

- Account
- Loan
- Customer

User Interface

- ATM
- Web
- PC Terminal

Primary Functionality, Persistence, distribution
The concern is crosscutting to the dimension of the implementation
AOP
AOSD
and
AspectJ overview

exploring another way to promote modularization

© Copyright 2009, Henrique Rebêlo.
Aspect-oriented languages are quite popular... due to the promise of modularizing crosscutting concerns
Revisiting the Example

Data Classes

Account

Loan

Customer

User Interface

ATM

Web

PC Terminal

Primary Functionality
Persistence
Distribution
Wouldn’t it be Nice if …
From your experience…

• Which other system concerns are “typically” crosscutting?
Quantification is the basic concept of AOP
The success of AOP

• Persistence code is **localized**, can be understood in isolation, part of it can be reused
• Business code **can be reused** to work with other persistence APIs
• Business code is **not invaded** by changes to persistence APIs
• Less code, more code units
Towards to

Aspect Oriented

Software Development

(AOSD)
Step 1: Concern identification (decomposition)
Examples of Identified Concerns

- Graphical user interface
- Distribution
- Concurrency control
- Data management
- Business rules
Step 2: Concern Implementation
Step 3: System Recomposition

classes and interfaces

W E A V E R

aspects

Executable System
Aspect Oriented Software Development

Concern identification
Concerns

OOP
Classes

AOP
Aspects

Program

Weaver

- Business rules
- Data management
- Distribution
- Concurrency control
- Graphical user interface
OOP vs. AOP

- Business rules
- Data management
- Distribution
- Concurrency control
- Graphical user interface
Context: Objects vs. Aspects

- Aspect-oriented programming (AOP)
  - modularizing crosscutting concerns (CCCs)

- enhancing reusability, maintainability, and the like
- AspectJ: most popular aspect-oriented language
Aspect-Oriented Programming (AOP): The AspectJ Approach

Legend
- Class
- Aspect

Join Points: Reference points in classes used by aspects for specifying relationships with classes
Pointcuts define a set of join points

- join points can be:
  - calls and executions of methods (and also constructors)
  - field access
  - exception handling
  - static initialization

- joint point composition
  - &&, || e!
Advice in AspectJ provides extra behavior at join points

• Define additional code that should be executed...
  – before
  – after
    • after returning
    • after throwing
  – or around
join points
public class Banco {
    private CadastroContas contas;...
    public void cadastrar(Conta conta) {
        try {
            getPM().startTransaction();
            contas.cadastrar(conta); ...
            getPM().commitTransaction();
        } catch (Exception ex){
            getPM().rollBackTransaction();...
        }
    }
}...
Pointcut (with a simple alternative)

```java
pointcut transMethods():
    execution (public * Banco.cadastrar(..)) ||
    execution (public * Banco.creditar(..)) ||
    execution (public * Banco.deditar(..)) ||
...

pointcut transMethods():
    execution (public * Banco.*(..));
```
public aspect PersistenceAspect {
    before(): transMethods() {
        getPM().StartTransaction();
    }

    after() returning(): transMethods() {
        getPM().commitTransaction();
    }
}
Advice for exceptional flow

```java
after() throwing(): transMethods() {
    getPM().rollBackTransaction();
}
...
```

Exception is not captured
Besides dynamic crosscutting with advice...

- AspectJ also supports static crosscutting
  - change the relation of subtypes
  - add members into classes
Static crosscutting

```java
pointcut TransMethods():
  execution (public * Trans.*(..));
private interface Trans {
  public void cadastrar(Conta conta);...
}
declare parents: Banco implements Trans;
```

interface local to the aspect

changing the hierarchy
Research with AOP discussing about some researches that use AOP

© Copyright 2009, Henrique Rebêlo.
Research with AOP

• There are several other
  – Exception handling improvement
  – Exception handling refactoring
  – AO refactoring
  – ...

industry

AOP

discussing about the adoption of AOP in industry

© Copyright 2009, Henrique Rebêlo.
AOP in Industry

• Used tools
  – Spring AOP
  – JBoss AOP
  – AspectJ
  – PostSharp (.NET)
  – Compose*
  – EOS (.NET)
  – Glassbox
Aspects in some Brazilian companies

- In most cases this means Spring AOP, but AspectJ is used too
- Popular concerns
  - Transactional control
  - Access control
  - Logging
  - Security
- Business code with reduced maintenance and more reuse
Aspects in *some* known companies

- IBM
  - supports AspectJ
- Motorola
  - WEAVR
- Microsoft
  - Policy Injection Application Block
Conclusions

• OO appeared with Simula 67
  – became popular in the industry after ...
  25 years?
  • design patterns

• AOP has 15 years
  – some use in big companies
AOP Framed!

Henrique Rebêlo
Informatics Center
Federal University of Pernambuco

© Henrique Rebêlo 2009