

Aspect – Oriented Programming

Paolo Falcarin



SoftEng
<http://softeng.polito.it>

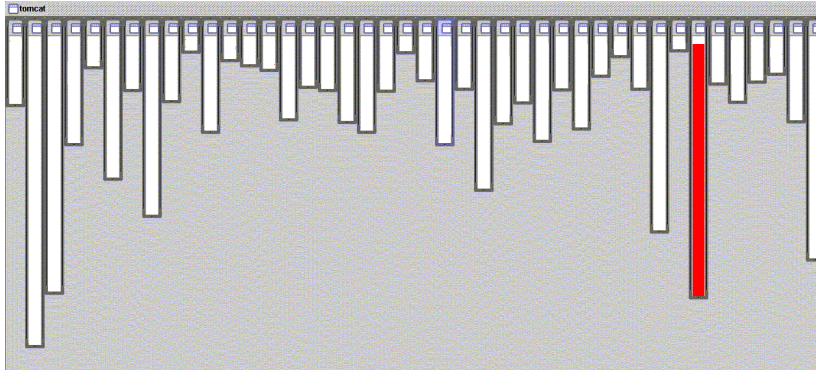
Corso di 3° livello

2007

Outline

1. **Aspect-Oriented Programming**
2. AspectJ language
3. AOP Examples
4. AOSD Tools & Research

Motivations for AOP

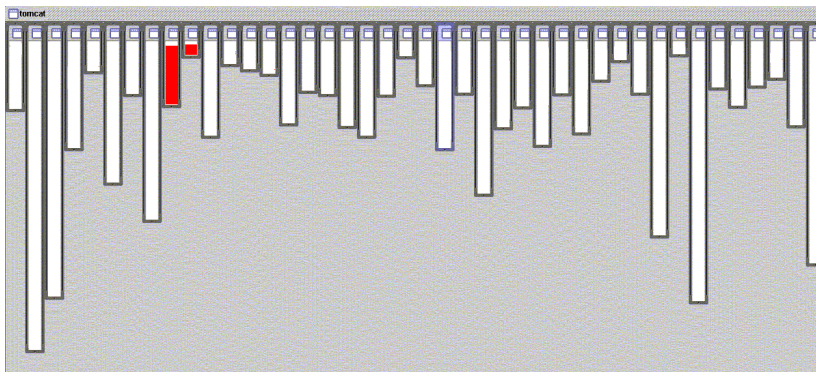


- XML parsing in org.apache.tomcat
 - ♦ Red shows relevant lines of code
 - ♦ It fits in one box

SoftEng
http://softeng.polito.it

Paolo Falcarin

Good modularity

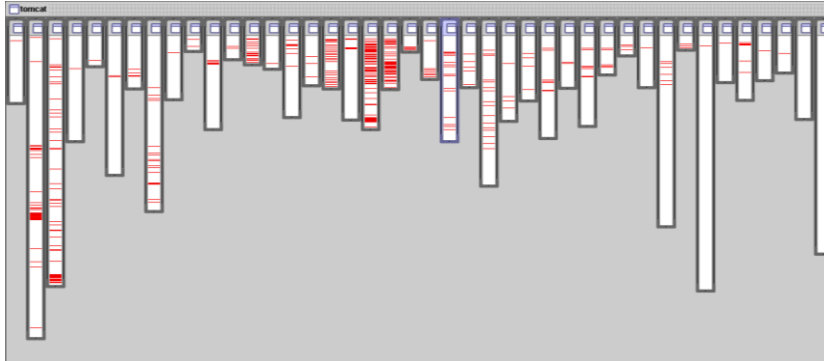


- URL pattern matching in org.apache.tomcat
 - ♦ Red shows relevant lines of code
 - ♦ It fits in two boxes (using inheritance)

SoftEng
http://softeng.polito.it

Paolo Falcarin

Logging is not modularized



- Logging in org.apache.tomcat
 - ◆ Red shows lines of code that handles logging
 - ◆ Not just in one place
 - ◆ Not in a small number of places

SoftEng
http://softeng.polito.it

Paolo Falcarin

What is AOP ?

- A new addition to the world of programming
 - ◆ New concepts & language constructs
 - ◆ New tools (aspect compiler, browser, debugger)
 - ◆ Many examples & patterns of practical use
 - ◆ A lot of hype
- Key advantage: separation of concerns
 - ◆ *Cross-cutting concerns* : those that are not well encapsulated in the usual OOD way - classes
 - ◆ Often appear in the form of 'code guidelines'

SoftEng
http://softeng.polito.it

Paolo Falcarin

For example...

```
public class SomeClass extends OtherClass {
    // Core data members
    // Other data members: Log stream, consistency flag

    public void DoSomething( OperationInformation info) {
        // Ensure authentication
        // Ensure info satisfies contracts
        // Lock the object in case other threads access it
        // Ensure the cache is up to date
        // Log the start of operation
        // ===== Perform the core operation =====
        // Log the completion of operation
        // Unlock the object
        // Do Standard Exception Handling
    }
    // More operations similar to above
}
```



Paolo Falcarin

Some definitions...

- *Scattered code* = related code not local
 - ◆ ...the Tomcat logging example
- *Tangled code* = local code not related
 - ◆ ..the previous example
- *Redundant code* = same fragment of code in many places
 - ◆ ...everyone has seen it... too often



Paolo Falcarin

The cost of tangled code



- Difficult to reason about
 - ◆ non-explicit structure
 - ◆ the big picture of the tangling isn't clear
- Difficult to change
 - ◆ have to find all the code involved
 - ◆ ...and be sure to change it consistently
 - ◆ ...and be sure not to break it by accident
- Problems:
 - ◆ Reduced reuse, speed, quality, ability to change

SoftEng
http://softeng.polito.it

Paolo Falcarin

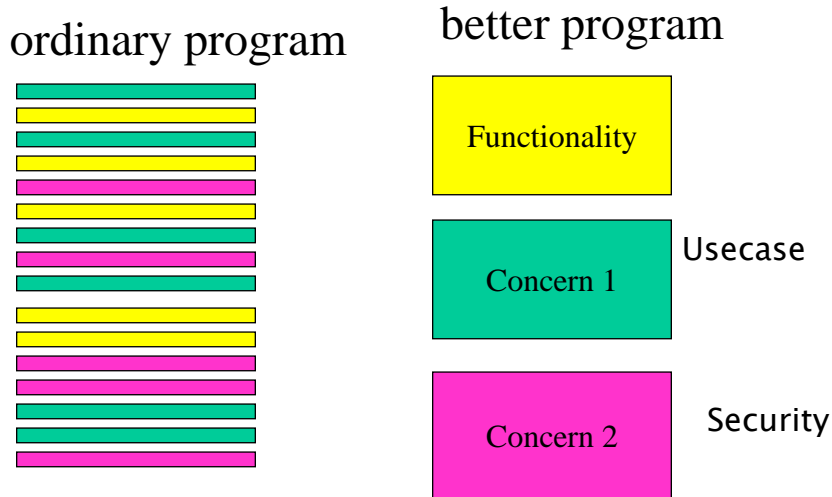
Concern

- Concern = system property
 - ◆ Functional property
 - ◆ Constraint of system behavior, i.e. conditions that need to be satisfied when the system runs
- An issue that the programmer has to deal with
 - ◆ How do I implement use-case "New-Item"
 - ◆ How do I implement the security policy?

SoftEng
http://softeng.polito.it

Paolo Falcarin

Cross-cutting of concerns



SoftEng
http://softeng.polito.it

Paolo Falcarin

Cross-cutting Concerns

- Are those system concerns which take effect over multiple artifacts within a design or program.
- Artifact = any tangible object at any level of the software development process.
- At implementation level, an artifact is any part of the application code.

SoftEng
http://softeng.polito.it

Paolo Falcarin

Example of Cross-cutting concerns

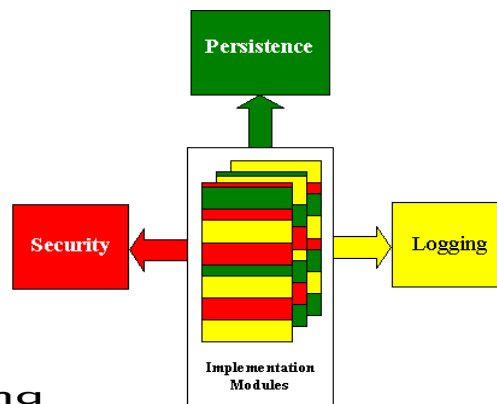
- Logging
- Debugging
- Profiling (Performance)
- Security & Authentication
- Exception Handling
- Event Handling
- Synchronization
- Session tracking and expiration
- Database connection & Persistence
- Standards issues & middleware
- Others...

SoftEng
http://softeng.polito.it

Paolo Falcarin

Separation of Concerns

- Separate logical concerns should be in separate modules of code – called aspects



SoftEng
http://softeng.polito.it

Paolo Falcarin

The AOP idea

- **Crosscutting concerns** have a clear purpose and structure
 - ◆ defined set of methods,
 - ◆ module boundary crossings
- It captures the structure of crosscutting concerns explicitly...
 - ◆ in a modular way
 - ◆ with language and tool support
- **Aspects** are
 - ◆ well-modularized crosscutting concerns
- **Aspect-Oriented Software Development**
 - ◆ AO support throughout lifecycle

SoftEng
<http://softeng.polito.it>

Paolo Falcarin

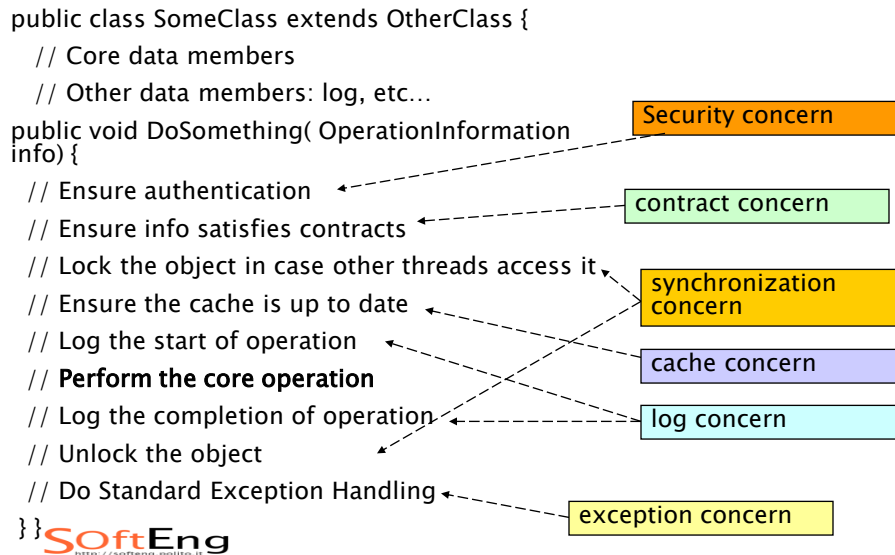
Aspect Oriented Programming

- New programming concept based on OOP
- 3 steps to programming in AOP
 - ◆ Separate concerns of a system
 - ◆ Specify description of their relationships
 - ◆ Use AO language mechanisms to weave them together into a program

SoftEng
<http://softeng.polito.it>

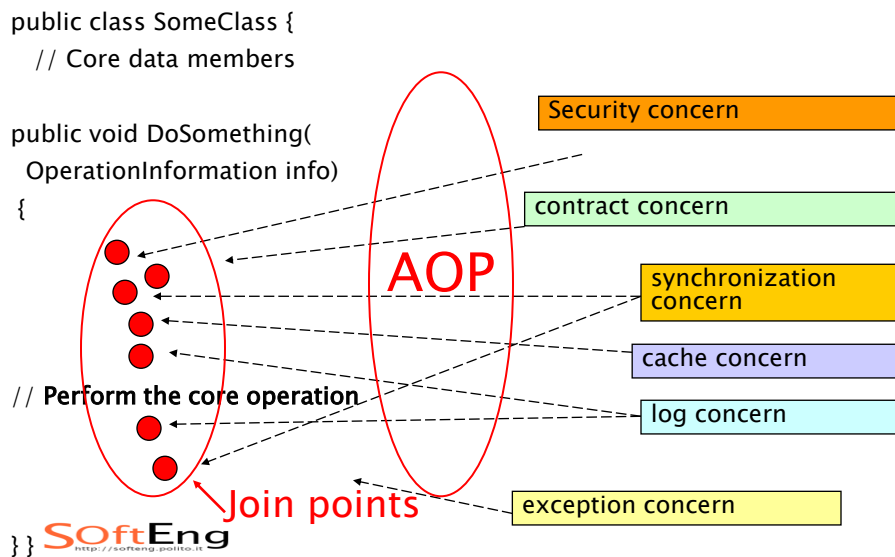
Paolo Falcarin

Example: identify concerns



Paolo Falcarin

Aspects implement concerns



Paolo Falcarin

Join-Point Model

- We want to specify conditions under which methods should execute.
- Those methods that execute conditionally when an event happens are called advice.
- Need a mechanism to specify sets of join points using
 - ◆ some primitives
 - ◆ wildcards
 - ◆ set operations

SoftEng
http://softeng.polito.it

Paolo Falcarin

Hello, World

- Let's start with a simple example

```
// HelloWorld.java
public class HelloWorld {
    public static void say(String message) {
        System.out.println(message);
    }

    public static void sayToPerson(
        String message, String name) {
        System.out.println(name + ", " + message);
    }
}
```

SoftEng
http://softeng.polito.it

Paolo Falcarin

Polite Hello, World

- Guess what the following aspect does

```
public aspect MannersAspect {
    pointcut callSayMessage() :
        call(public static void HelloWorld.say*(..));

    before() : callSayMessage() {
        System.out.println("Good day!");
    }

    after() : callSayMessage() {
        System.out.println("Thank you!");
    }
}
```

What's in the Example

- A **Pointcut** defines at which points in the dynamic execution of the program – at what **Join Points** – extra code should be inserted
- An **Advice** defines when, relative to the join point, the new code runs, and that actual code
- An **Aspect** encapsulates pointcuts and advices

OOP & AOP

- Object–Oriented Programming
 - ♦ Basic concept of modularity : the class
 - ♦ Good for common concerns (inheritance)
 - ♦ A program is a set of classes
- Aspect–Oriented Programming
 - ♦ Basic concept of modularity: the aspect
 - ♦ Good for unrelated concerns (pointcuts)
 - ♦ A program is a set of aspects and classes
- AOP complements OOP

Quantification

- Aspects are mechanisms for localizing the expression of a crosscutting concern
 - ♦ Aspect actually separates the concern from the rest of the code into its own structure
 - ♦ One can write unitary and separate statements (aspects) that have effects in many non–local places in a programming system.

Obliviousness

- Invocation of an aspect is implicit
- Writer may not even have to be aware of particular concern
- Every time get to a point in code where concern needs to be addressed, aspect is implicitly called
- This property is also known as *obliviousness* (implicit invocation)
- The places where the defined quantifications applied (join-points) did not have to be prepared to receive these enhancements (aspect's advice)

SoftEng
<http://softeng.polito.it>

Paolo Falcarin

Benefits of Aspects

- Better modularity.
 - ♦ Crosscutting concerns are now localized in an aspect.
- Easier understanding.
 - ♦ The information about a concern is localized. It is less scattered and tangled with other information.
- Easier modifiability.
 - ♦ Look at one aspect and its binding.

SoftEng
<http://softeng.polito.it>

Paolo Falcarin

How can you benefit immediately?

- Use Aspect-Oriented Programming (AOP) for developing your Java programs. What you turn in is all Java.
 - ♦ Debugging Aspects
 - ♦ Testing pre- and post-conditions
 - ♦ Profiling
- Your programs will be better and you can produce them more easily.

Brief history & background

- Karl Lieberherr
 - *Demeter & Adaptive Programming*, 1991
- Gregor Kiczales
 - *The Art of Metaobject Protocol*, 1991
- Kiczales, Lopes, Lieberherr
 - AOP, Xerox Parc AOP Team, 1996
- First release of AspectJ (AOP for Java), 1998
 - Now at <http://eclipse.org/aspectj>
- Aspect Oriented Software Development community: <http://aosd.net>

Meta-programming and AOP

- **Meta-programming** = “*The art of programming programs that read, transform or write other programs*” (F.R. Rideau et al., 1999)
- It naturally appears in software development:
 - Compiler, debugger, interpreter, builder
- Meta-programming includes existing programs that have code as input and/or output instead of data.
- We can define **meta-programming categories** $N \rightarrow P$:
 - N is the number of types of code in input
 - P is the number of types of code in output

Meta-programming categories $N \rightarrow P$

- $0 \rightarrow 0$: normal program;
- $1 \rightarrow 0$: interpreter, code inspector, code analyzer, automatic tester
- $0 \rightarrow 1$: data pre-compiler, self extractor
- $1 \rightarrow 1$: compiler, translator, optimizer, debugger, obfuscator, beautifier, refactoring
- $2 \rightarrow 0$: model-checker, CVS diff
- $2 \rightarrow 1$: meta-compiler, aspect weaver, CVS update
- $1 \rightarrow 2$: code splitter