



Testing AOP code

Mariano Ceccato

ITC-Irst

Centro per la ricerca
Scientifica e Tecnologica

ceccato@itc.it

New Fault Model

- ◆ Faults generated while developing AOP code are quite new with respect to those for traditional code.
- ◆ Some examples of them are:
 - Incorrect strength in pointcut patterns.
 - Incorrect aspect precedence.
 - Failure to establish expected post-conditions and invariants.
 - Failure due to inter-type declarations.

Incorrect strength in pointcut patterns.

- ◆ The pattern in the pointcut designator gives the set of the intercepted join points.
 - A too weak designator will intercept more join points than required
 - A too strong designator will miss some join points

Incorrect strength in pointcut patterns.

Point

```
void setColor ( Color )  
Color getColor ( )  
void setX ( int )  
int getX ( )  
void setY ( int )  
int getY ( )  
Object[ ] settings ( )
```

Point.set*(..)

- Point.setColor(Color)
- Point.setX (int)
- Point.setY (int)
- Point.settings ()

Point.set*(int)

- Point.setX (int)
- Point.setY (int)

Failure on post-conditions and invariants.

- ◆ **Post-conditions** are condition that are considered valid after a method in the base code has run.
- ◆ They represent the contracts between the caller and the called class.
- ◆ **Invariants** are conditions on the system that must be valid always.
- ◆ These conditions should keep valid also when aspects are weaved.
- ◆ When advices produces ripple effects that violate them, the overall system behavior might be incorrect

Conservative Solution

- ◆ Any existing coverage criterion (e.g. branch coverage) seems to be adequate to reveal the AOP specific faults.
- ◆ The AOP specific constructs introduces new branches that require to be covered by at least a test case.
- ◆ Minor adaptation to existing techniques are required, in order to make them applicable on AOP code

Incorrect strength in pointcut patterns.

- ◆ Problems in the patterns should be detected by a complete coverage:
 - In case of a too large intercepted join-point set, (AOP specific) test cases traversing incorrectly added branches should reveal defects.
 - In case of a too small intercepted join-point set, (OOP specific) test cases traversing the missed branches should reveal the missed advice executions.

Failure on post-conditions and invariants.

- ◆ These faults can be detected by those test cases whose observable output depend on such conditions.
- ◆ Running the test cases defined for the base code should reveal such kind of defects.

Adaptation of existing coverage criteria: dynamic crosscutting

- ◆ Dynamic crosscutting modifies the class behavior by defining:
 - Pointcuts: they declare the join points to intercept
 - Advices: they contain the code to execute on join points
- ◆ Advice execution can be treated as a method invocation: there is an implicit branch between the intercepted base code to the advice body.
- ◆ Around advices introduce more than one branch because they can selective run the intercepted join points.

Adaptation of existing coverage criteria: static crosscutting

- ◆ Static crosscutting affect the class structure using:
 - Intertype declarations: they modify inheritance relations
 - Introductions: insert new methods and fields into classes
- ◆ Static crosscutting can alter existing branches that evaluate conditions on the class structure (**instanceof**) or in case of polymorphic calls.

Drawbacks

- ◆ Adoption of existing test case will produce complex test cases.
- ◆ They are based on the control flow of the woven application.
- ◆ The resulting control flow corresponds to the base system behavior tangled with all the crosscutting concerns.
- ◆ Resulting test cases exercises both the base code and the crosscutting concerns at the same time.
- ◆ Definition of test input and expected output requires to reason about all the tangled concerns as a whole.
- ◆ Advantages of the separation of the concerns achieved in the development phase are lost in the testing phase.

A possible solution

- ◆ To achieve the separation of the concerns in also in the testing phase an incremental approach is required.
 - Write test cases for the base code and reach the full coverage.
 - For each concern:
 1. Weave the concern to the base code.
 2. Evaluate the coverage.
 3. Write test cases to complete the coverage for the waved code.

Advantages

- ◆ Separation of the concerns in the testing phase.
- ◆ More effective debugging: a fault in the code should be easy to locate when a test case fails, because it is required to reason about only one concern.
- ◆ Test should be easy to understand because they are divided, according to the concern identified in the development phase.

The project

- ◆ We use incremental testing:
 - The test suite for the base code is available.
 - A new concern has been added using AOP.
 - Test cases for the new code must be produced in order to achieve the full coverage for the resulting code.