Unweaving the Impact of Aspect Changes in AspectJ

Luca Cavallaro – Mattia Monga
Problem Outline

• Small changes can have major and nonlocal effects in programs

• For Aspect Oriented software the problem is even more relevant, for the obliviousness of Aspect oriented programs

• Local changes are not really local
  – Changes in the base system Influence Aspects and vice versa!
Problem solution: Change Impact Analysis

• We suppose to have two versions of the same program and a test suite

• We run tests on two versions of the program

• We compare source of two versions to find “atomic changes”
  – “Small” changes in program source
  – There are interdependencies between atomic changes

• We compare graph representation of the two program versions
Change impact analysis overview

• We find dangerous paths and map them on atomic changes

• An atomic change
  – in dangerous paths is responsible for test result change
  – not mapped on dangerous edges do not to affect test result
  – not mapped on any test in the suite is not tested

• Deleting a set of AC in dangerous paths produces a version of the program giving previous test result
Running example

---

**Point**

- `x : int = 0`
- `y : int = 0`

- `<getter>`+getX() : int
- `<getter>`+getY() : int
- `<setter>`+setRectangular( new X : int, new Y : int ) : void
- `<setter>`+setX( new X : int ) : void
- `<setter>`+setY( new Y : int ) : void
- +toString() : String

---

**PointExt**
Running example

```
Point

~x : int = 0
~y : int = 0

<<getter>>+getX() : int
<<getter>>+getY() : int
<<setter>>+setRectangular( new X : int, new Y : int ) : void
<<setter>>+setX( new X : int ) : void
<<setter>>+setY( new Y : int ) : void
+toString() : String

PointExt

<<setter>>+setRectangular( new X : int, new Y : int ) : void
<<setter>>+setX( new X : int ) : void
```
Running example

• Bound point aspect:
  – A pointcut to capture setX and methods that calls it
  – A pointcut to capture setX calls only
  – We add a field in modified version

```java
// ====== advices ======

before (Point p, int x) throws InvalidException:
  setterX(p) && args(x) { // before }
void around (Point p): setterX(p) {
  //around1 }
void around (Point p): setterXonly(p)
  // around2 }
before (Point p): setterX(p){ // before2
  //modified to use added field
}

after (Point p) throwing (Exception ex):
  setterX(p) { // afterThrowing1 }
after (Point p): setterX(p){ // after1 }
```
Test Case

```java
public static void main(String[] a) throws Exception {
    Point p1 = new Point();
    p1.setRectangular(5,2);
    System.out.println("p1 = " + p1);
    if(p1.x> 5){
        p1.setX(6);
        p1.setY(3);
        System.out.println("p1 = " + p1);
    }
    else{
        System.out.println("p1 = " + p1);
    }
    Point p2 = new PointExt();
    p2.setRectangular(5,2);
    System.out.println("p2 = " + p2);
    p2.setX(5);
}
```
Atomic changes example

CM
PointExt.setX

AM
PointExt.setX

LC
PointExt, PointExt.setX

LC
PointExt, PointExt.setX

AM
PointExt.setRectangular

LC
PointExt, PointExt.setRectangular

CM
PointExt.setRectangular

AF
BoundPoint.previousXValue

CAB
before (Point p): setterX(p)

C1 structurally depends on C2
C2 ➔ C1

C1 has declaration dependence on C2
C2 ➔ C1

C1 has mapping dependence on C2
C2 ➔ C1
AspectJ interaction Graph

• We use the AspectJ Interaction Graph (AJIG) to represent program semantics

• Control flow representation of an AspectJ program

• Three main kinds of interactions:
  – Non-advice method calls
  – Interactions between advices and methods
  – Introductions and intertype declarations
Example

- Dangerous edge 1 is due to CAB of Before 2
  - It is mapped on CBM and AF
- Dangerous edge 2 is due to the LC PointExtm Point.setX()
  - It is mapped on two AC: LC, AM
Implementation

• We implemented change impact analysis for AspectJ on top of abc and Ajana
• abc is an extensible AspectJ compiler
  – Built on top of Soot and Polyglot
  – Allows to access program AST and to implement analysis
  – Due to two phases weaving we could analyze AspectJ programs without considering instructions added by the compiler
• Ajana is a framework for AspectJ analysis
  – Provides AJIG representation
Future work

- We produced and implemented an approach that helps the programmer maintaining code
  - Source code changes are decomposed into atomic changes and are related
  - Change in tests results are mapped on source code changes
- For future work we plan to rise abstraction level
  - Build changes classifiers
  - Classify possible changes following anti-patterns classification
  - Several work try to build metrics for changes in AO programs