



Fraglight: Shedding Light on Broken Pointcuts in Aspect-Oriented Software

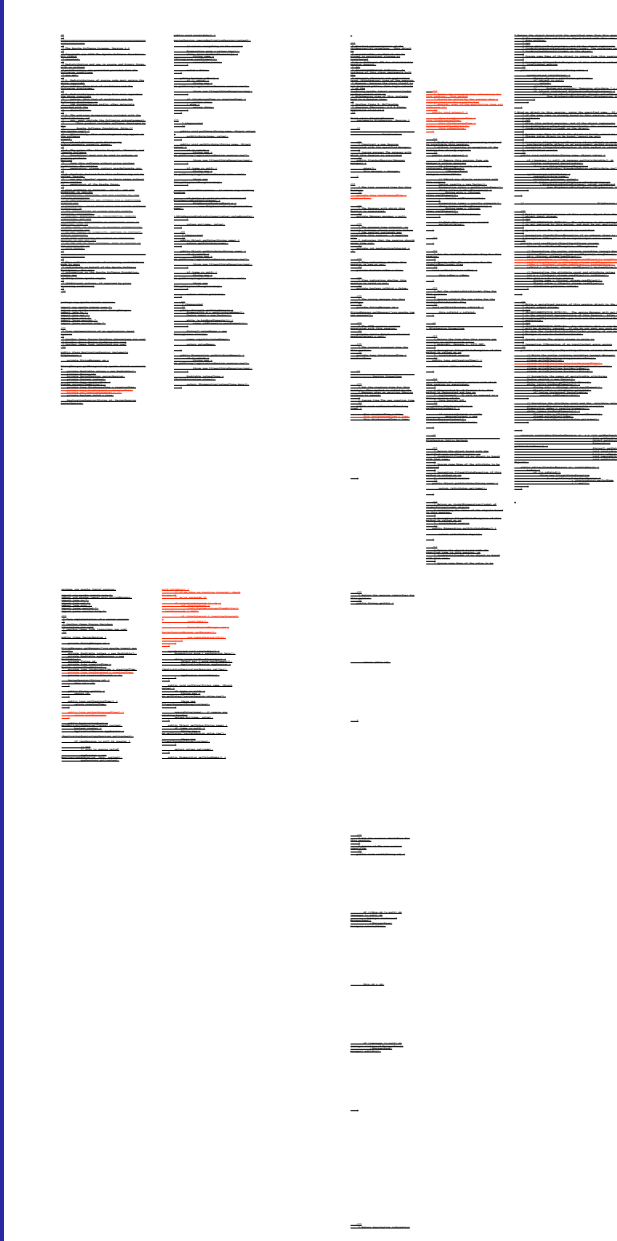
Raffi Khatchadourian¹, **Awais Rashid**², **Takuya Watanabe**³, and **Hidehiko Masuhara**⁴
 (1) Computer Systems Technology, New York City College of Technology, City University of New York
 (2) Computing, Lancaster University, Lancaster UK
 (3) Edirium K.K., Ichikawa, Japan
 (4) Mathematical and Computing Science, Tokyo Institute of Technology, Tokyo, Japan

Supported in part by the National Science Foundation (Grant No. OISE-1015773) and the Japan Society for the Promotion of Science (Grant No. Sp10024).

Message encryption is an example since many parts of a program involve security


Background

Bad Modularity



- Some program modules tend to affect many *other* modules.
- Such modules implement *crosscutting concerns* (CCCs).
- Code is **scattered** and **tangled**.

Good Modularity



- Aspect-Oriented Programming enables *localized* implementations of CCCs.
- **Pointcuts** select (*join*) **points** in the program where a CCC applies.
- Code (**advice**) is executed at those points.


Aspects allow code to be localized into a single module

Message Encryption

A pointcut (**pc**) may designate that:
all modules whose name begins with "send" must have their messages encrypted.

Problem

Fragile Pointcut Problem



What if a **new** module is made that **sends** messages but whose name begins with **"transfer"**?

- **pc** is **fragile** since it **fails** to capture the new module.
- Fragile pointcuts can cause **software** to **malfunction**.

Fixing Broken Pointcuts

- Requires **manually** identifying **all broken pointcuts**.
- Manually identifying all broken pointcuts is:
 - **tedious**,
 - **time-consuming**,
 - **error-prone**, and
 - **omission-prone**
 when there are **many** pointcuts!

Structural Commonality

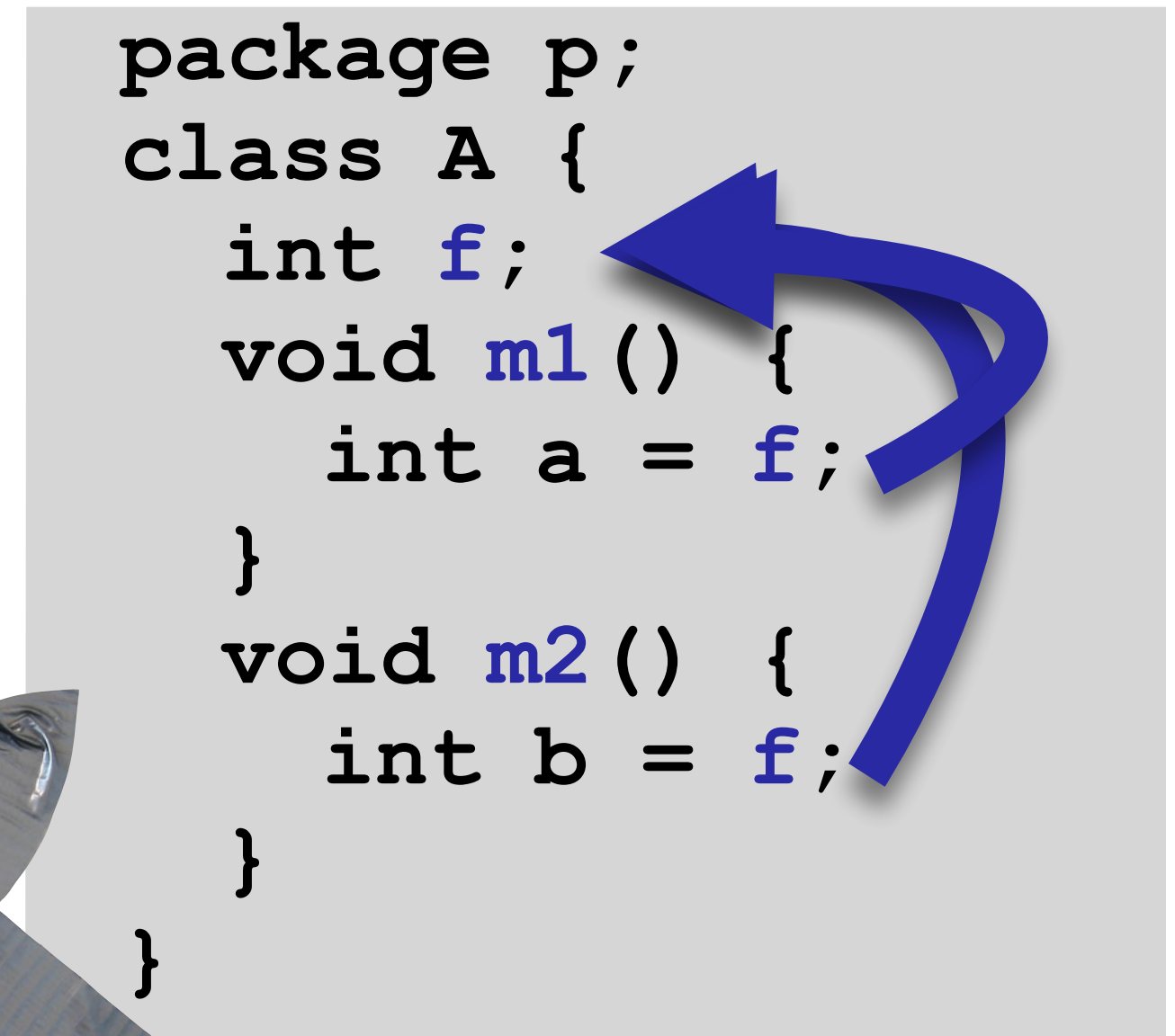
Hypothesis: Program elements corresponding to join points selected by a pointcut in a particular version typically share **structural commonality** that **persists** throughout subsequent **versions**.

Insight

Phase I: Structural Analysis

Extract commonality between currently selected join points.

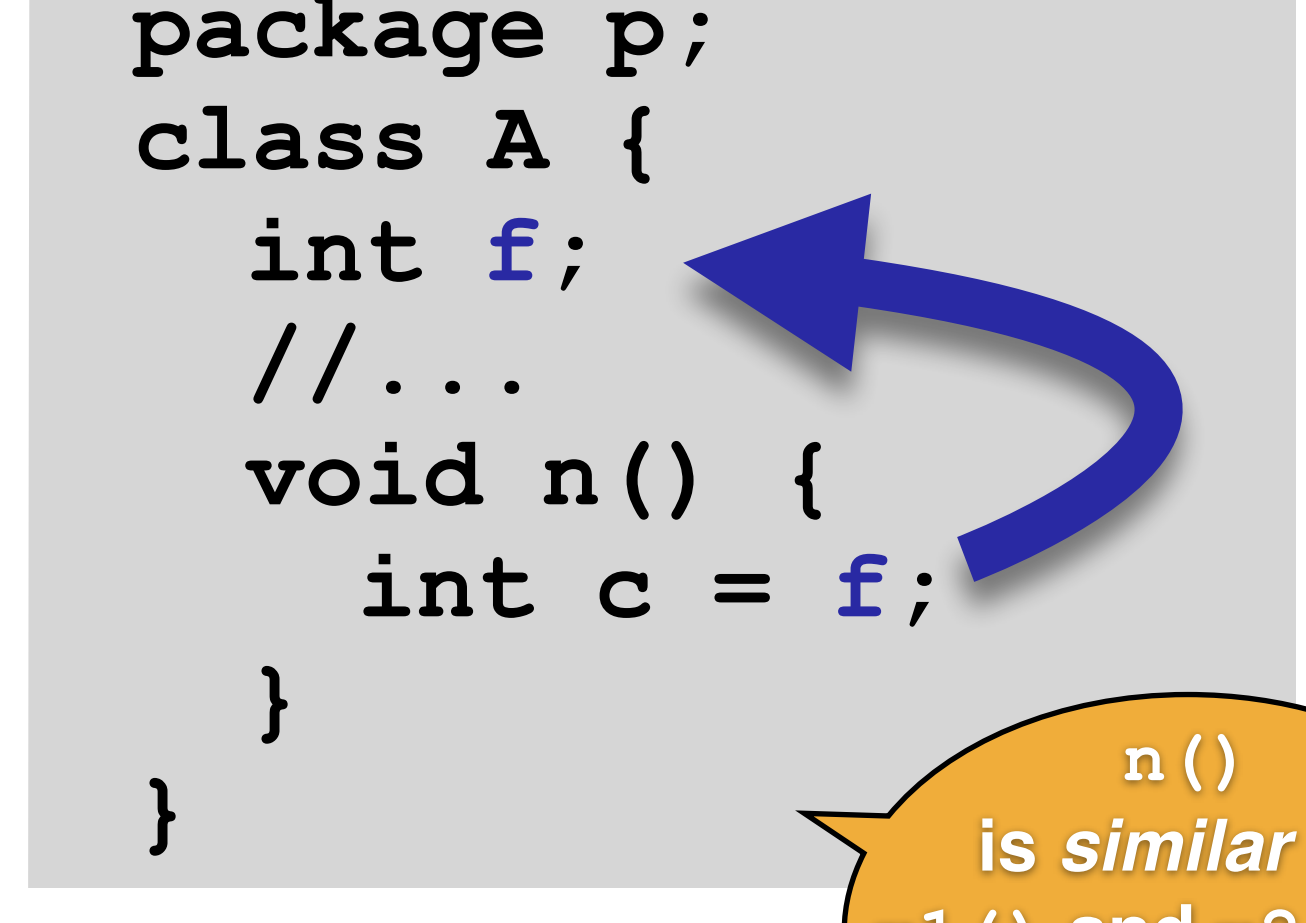
```
package p;
class A {
  int f;
  void m1() {
    int a = f;
  }
  void m2() {
    int b = f;
  }
}
```



Phase II: Break Detection

Apply extracted patterns to new evolved version.

```
package p;
class A {
  int f;
  //...
  void n() {
    int c = f;
  }
}
```



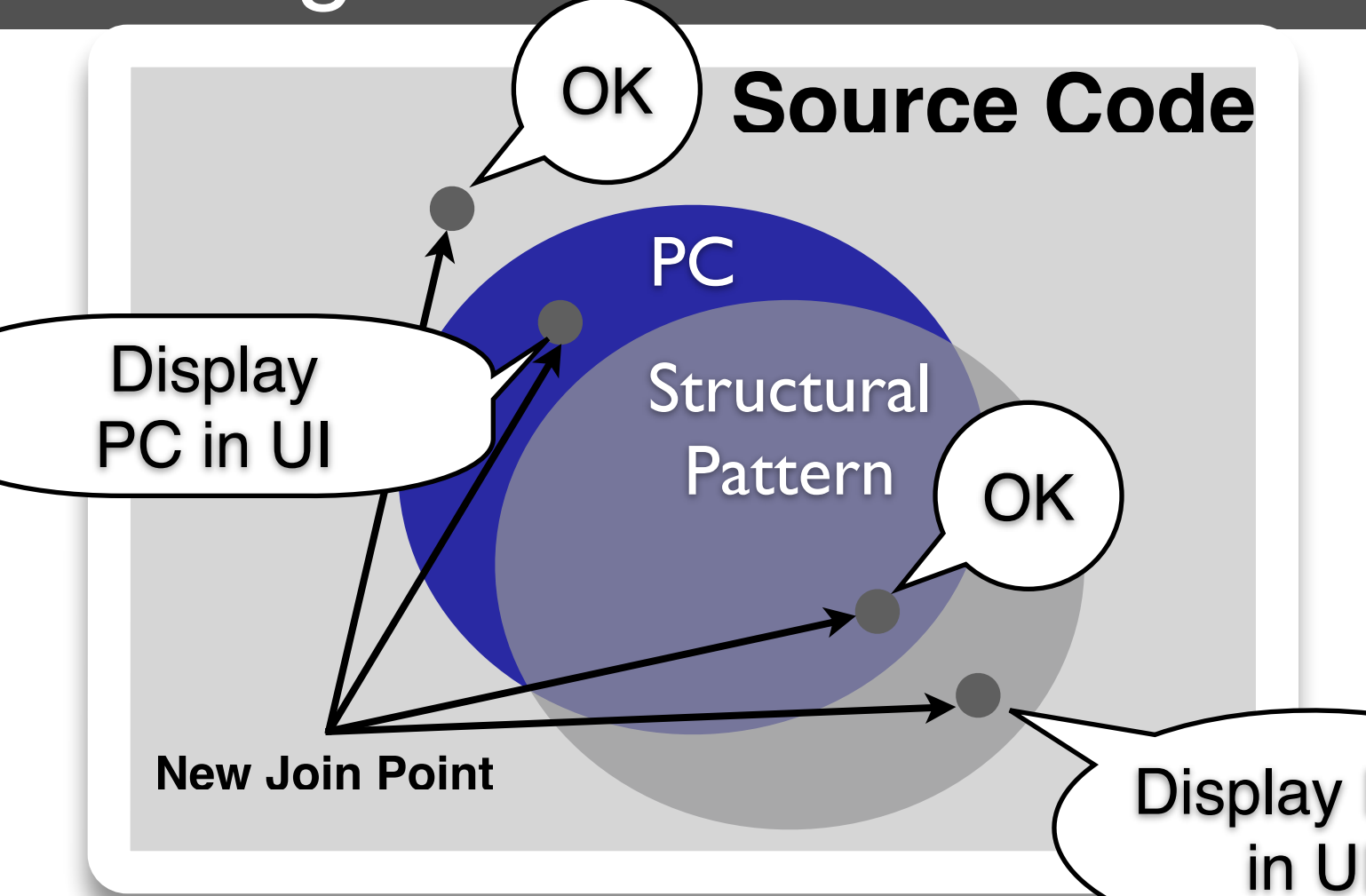
n() is similar to m1() and m2() but not captured by pc

On-the-fly Enlightenment

pc is **highlighted** in the user-interface (UI) workbench as the developer is typing!

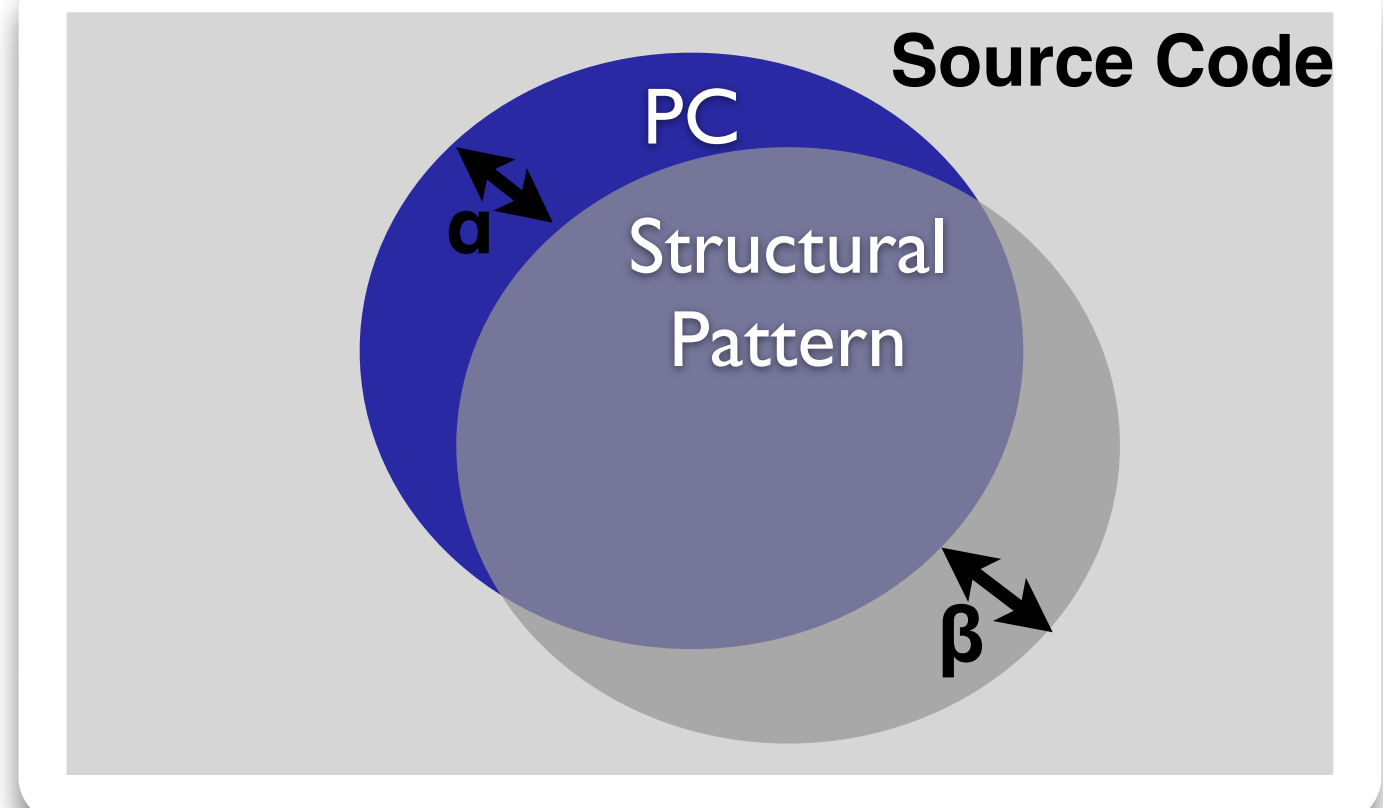
Approach

Adding a New Join Point



OK Source Code
 Display PC in UI
 New Join Point
 OK
 Display PC in UI

Is There Commonality?



- Compared pointcuts to patterns in 23 AspectJ programs.
- Avg. Type I (α) error rate of **0.18**.
- Avg. Type II (β) error rate of **0.16**.

Ensuing Research Questions

- Can *Fraglight* detect broken pointcuts accurately?
- Can *Fraglight* prevent bugs?
- Are there performance trade-offs?
- How can possibly broken pointcuts be brought to the developer's attention effectively without interrupting workflow?