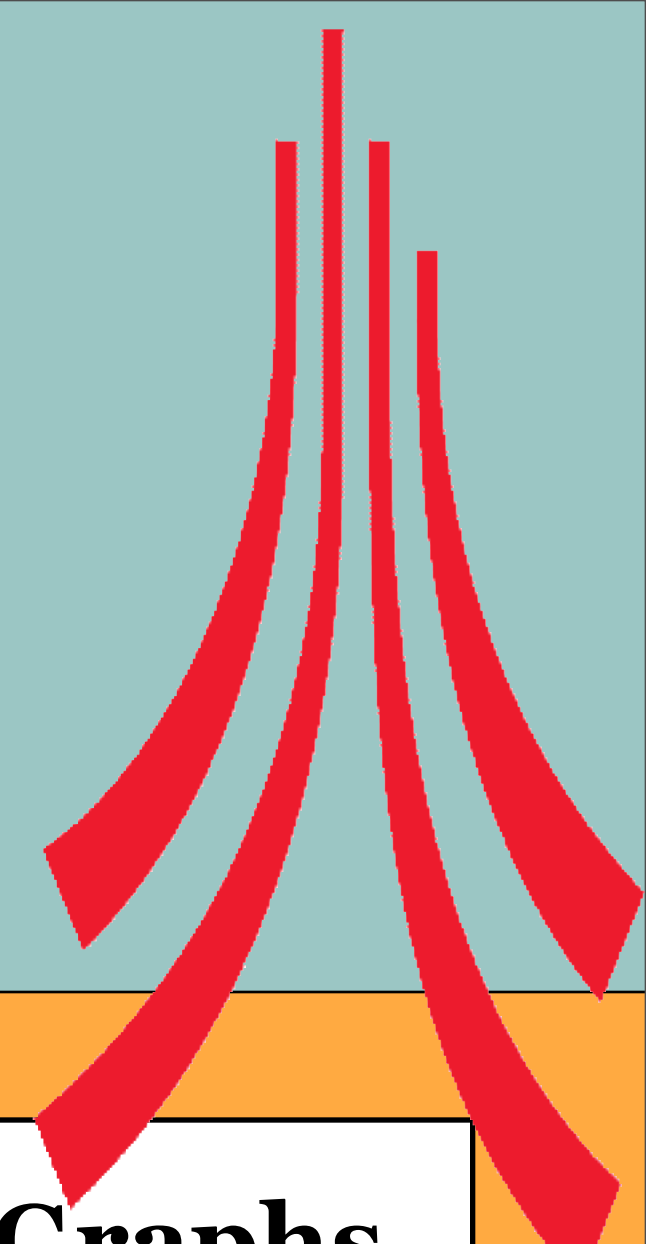


Pointcut Rejuvenation: Recovering Pointcut Expressions in Evolving Aspect-Oriented Software

Raffi Khatchadourian, Ohio State University
Awais Rashid, Lancaster University



Issues in Evolving Aspect-Oriented Programs

- Constructing *optimal* pointcut expressions that capture *true* intentions of where a CCC applies can be difficult.
- Ideally, pointcut expressions should remain valid even as the software evolves.
- As the *base-code* evolves, however, pointcut expressions may no longer be valid and thus require *reconstruction*.
- But which *join points* should be included in the new expression?

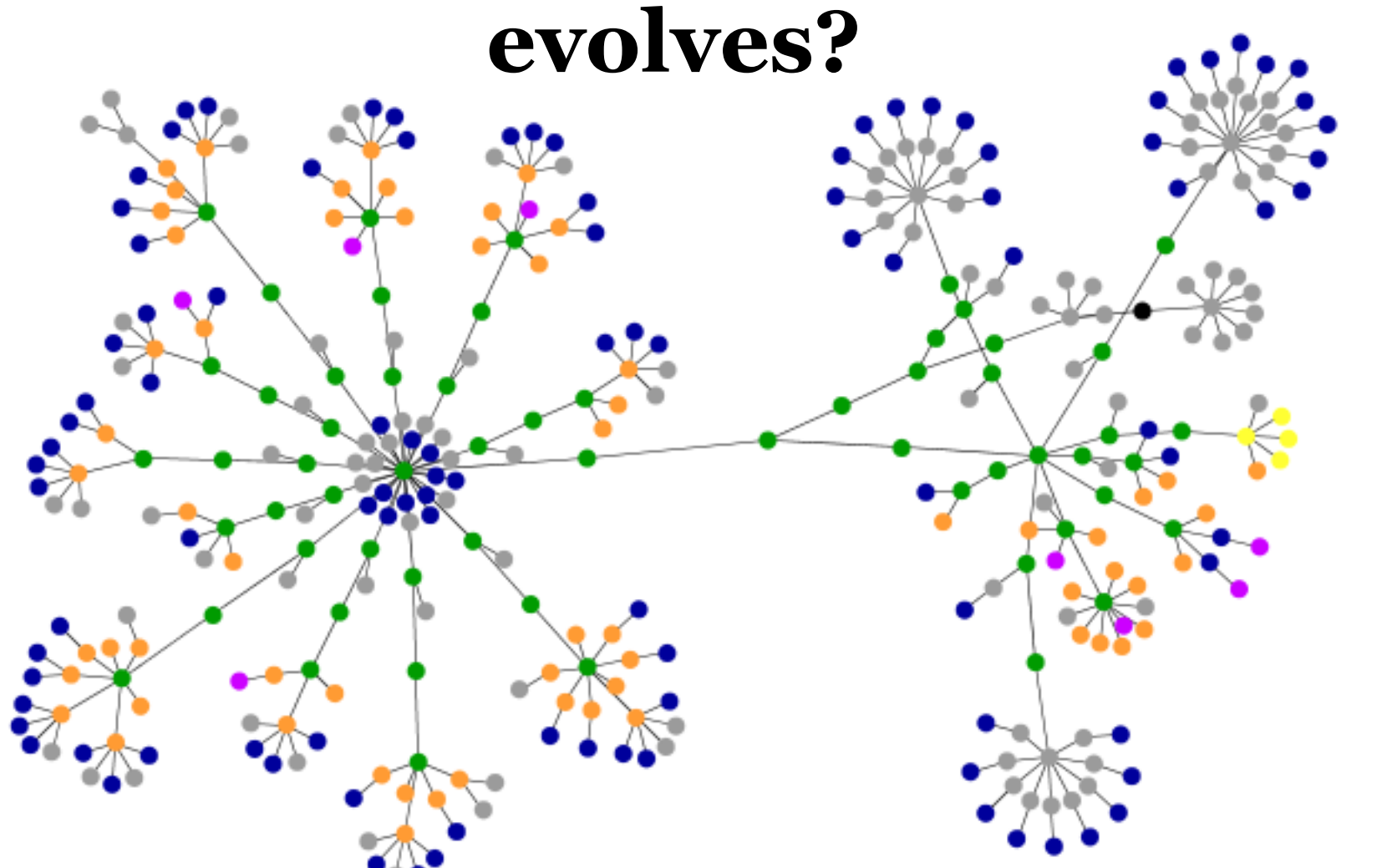
“Can we automatically infer join points that should be included a pointcut upon evolution?”

Managing Requirements Traceability

- Some resemblance to the *requirements traceability problem*.
- Software is *requirement traceable* if we can trace a requirement to its corresponding implementation.
- Like pointcut expressions, traceability information may be invalidated over time.

“Can we adapt requirements traceability techniques to manage pointcut expressions as the underlying base-code evolves?”

In other words, can we automatically “complete the crosscut” as the software evolves?



Courtesy <http://www.allthingsdistributed.com>

Intention Graph Mining: An Automated Inferencing Proposal

- Intention graphs, commonly used in HCI, are adapted to capture *programmer* intentions.
 - Analyze** patterns exhibited by join points contained within an existing pointcut.
 - Identify** similarities in the structure of *paths* in the intention graph where join points occur.
 - Derive** intentional patterns between enabled paths.
 - Evolve the base-code**
 - Mine** for other paths that *match* the pattern.
 - Incorporate** successfully selected join points into the existing pointcut.

Executions of `x()` are currently being advised by aspect B

Motivating Example

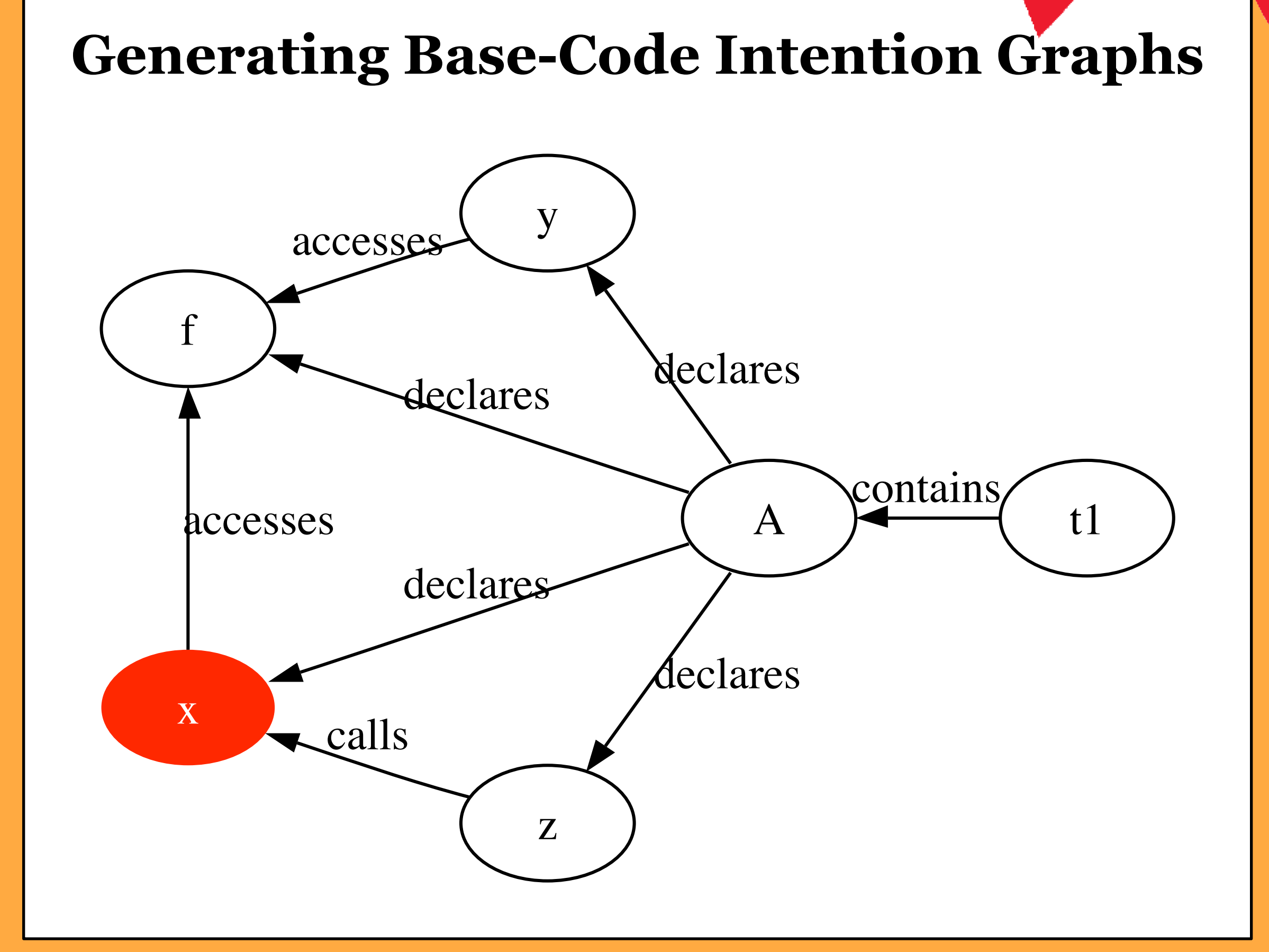
```

1 package t1;
2
3 public class A {
4     int f;
5
6     void x() {
7         f = 1;
8     }
9
10    void y() {
11        f = 2;
12    }
13
14    void z() {
15        x();
16    }
17 }
    
```

```

1 package t1;
2
3 public aspect B {
4     pointcut p() :
5         execution(void x());
6
7     before() : p() {
8         //...
9     }
10 }
    
```

`x()` is one of two methods accessing field `f`



“Should new methods accessing field `f` also be included in pointcut `p`?”

- Ideally, but highly unlikely, there would be a perfect match.
 - The join points contained in the original pointcut would match those produced by the pattern mining.
 - We compute a **confidence factor** which is a function of existing matched elements and wild-cards in the pattern.
 - Resulting join points are ranked by their confidence.

$$precision(\hat{\delta}, \Delta') = \begin{cases} 1 & \text{if } |apply(\hat{\delta})| = 0 \\ \frac{|\Delta' \cap apply(\hat{\delta})|}{|apply(\hat{\delta})|} & \text{otherwise} \end{cases}$$

$$accuracy(\hat{\delta}) = \frac{|\hat{\delta}| - |\pi(\hat{\delta})|}{|\hat{\delta}|}$$

$$confidence(\hat{\delta}, \Delta', w_{precision}, w_{accuracy}) = precision(\hat{\delta}, \Delta')w_{precision} + accuracy(\hat{\delta})w_{accuracy}$$
