

On the Assessment of Pointcut Design in Evolving Aspect-Oriented Software¹

Raffi Khatchadourian^{1,2} Phil Greenwood² Awais Rashid²

¹Department of Computer Science & Engineering
Ohio State University
Columbus, OH USA

²Computing Department
Lancaster University
Lancaster UK

Workshop on Assessment of Contemporary Modularization
Techniques, 2008

¹This material is based upon work supported in part by European Commission grants IST-33710 (AMPLE) and IST-2-004349 (AOSD-Europe).

²This work was partially administered during this author's visit to the Computing Department, Lancaster University, United Kingdom.

Outline

- 1 Brief Introduction to AOP
- 2 Problem Statement
 - Definitions
- 3 Proposal
 - Metrics
 - Formalism
- 4 Example
 - Scenario
 - Metrics
- 5 Conclusion and Future Work



InfoLab21

Aspect-Oriented Programming

- AOP reduces scattering and tangling of crosscutting concern (CCCs) implementations.
- Developer specifies behavior (advice).
- Advice is composed at specific execution points (join points).
- Join points specified via pointcut expressions (PCEs).

Example

```
execution(* Foo.*(..))  
execution(* Foo.methodA())
```



Aspect-Oriented Programming

- AOP reduces scattering and tangling of crosscutting concern (CCCs) implementations.
- Developer specifies behavior (advice).
- Advice is composed at specific execution points (join points).
- Join points specified via pointcut expressions (PCEs).

Example

```
execution(* Foo.*(..))  
execution(* Foo.methodA())
```



Aspect-Oriented Programming

- AOP reduces scattering and tangling of crosscutting concern (CCCs) implementations.
- Developer specifies behavior (advice).
- Advice is composed at specific execution points (join points).
- Join points specified via pointcut expressions (PCEs).

Example

```
execution(* Foo.*(..))  
execution(* Foo.methodA())
```



Aspect-Oriented Programming

- AOP reduces scattering and tangling of crosscutting concern (CCCs) implementations.
- Developer specifies behavior (advice).
- Advice is composed at specific execution points (join points).
- Join points specified via pointcut expressions (PCEs).

Example

```
execution(* Foo.*(..))  
execution(* Foo.methodA())
```



Aspect-Oriented Programming

- AOP reduces scattering and tangling of crosscutting concern (CCCs) implementations.
- Developer specifies behavior (advice).
- Advice is composed at specific execution points (join points).
- Join points specified via pointcut expressions (PCEs).

Example

```
execution(* Foo.*(..))  
execution(* Foo.methodA())
```



Aspect-Oriented Programming

- AOP reduces scattering and tangling of crosscutting concern (CCCs) implementations.
- Developer specifies behavior (advice).
- Advice is composed at specific execution points (join points).
- Join points specified via pointcut expressions (PCEs).

Example

```
execution(* Foo.*(..))  
execution(* Foo.methodA())
```



The Problem

- Constructing optimal PCEs can be difficult.

Fragile Pointcuts

PCEs should capture correct join points not only in the current version of the base-code but future versions as well.

- Capturing **true** intentions of where a CCC applies may alleviate fragility.
 - PCEs often in terms of low-level programming elements.
 - High-level specifications must be translated.

Can we measure how well a PCE captures a developer's *intentions*?



INTOLAB21

The Problem

- Constructing optimal PCEs can be difficult.

Fragile Pointcuts

PCEs should capture correct join points not only in the current version of the base-code but future versions as well.

- Capturing **true** intentions of where a CCC applies may alleviate fragility.
 - PCEs often in terms of low-level programming elements.
 - High-level specifications must be translated.

Can we measure how well a PCE captures a developer's *intentions*?



INTOLAD 21

The Problem

- Constructing optimal PCEs can be difficult.

Fragile Pointcuts

PCEs should capture correct join points not only in the current version of the base-code but future versions as well.

- Capturing **true** intentions of where a CCC applies may alleviate fragility.
 - PCEs often in terms of low-level programming elements.
 - High-level specifications must be translated.

Question

Can we **measure** how well a PCE **captures** a developer's *intentions*?



The Problem

- Constructing optimal PCEs can be difficult.

Fragile Pointcuts

PCEs should capture correct join points not only in the current version of the base-code but future versions as well.

- Capturing **true** intentions of where a CCC applies may alleviate fragility.
 - PCEs often in terms of low-level programming elements.
 - High-level specifications must be translated.

Question

Can we **measure** how well a PCE **captures** a developer's *intentions*?



The Problem

- Constructing optimal PCEs can be difficult.

Fragile Pointcuts

PCEs should capture correct join points not only in the current version of the base-code but future versions as well.

- Capturing **true** intentions of where a CCC applies may alleviate fragility.
 - PCEs often in terms of low-level programming elements.
 - High-level specifications must be translated.

Question

Can we **measure** how well a PCE **captures** a developer's *intentions*?



The Problem

- Constructing optimal PCEs can be difficult.

Fragile Pointcuts

PCEs should capture correct join points not only in the current version of the base-code but future versions as well.

- Capturing **true** intentions of where a CCC applies may alleviate fragility.
 - PCEs often in terms of low-level programming elements.
 - High-level specifications must be translated.

Question

Can we **measure** how well a PCE **captures** a developer's *intentions*?



Intentions

“High”-level intention: A natural language description of where CCC applies.

Example

Advise whenever data is sent over a network connection

Low-level intention: A description of where a CCC applies in terms of *structural* program element relationships.

Example

Intentions

“High”-level intention: A natural language description of where CCC applies.

Example

Advise whenever data is sent over a network connection

Low-level intention: A description of where a CCC applies in terms of *structural* program element relationships.

Example

Intentions

“High”-level intention: A natural language description of where CCC applies.

Example

Advise whenever data is sent over a network connection

Low-level intention: A description of where a CCC applies in terms of *structural* program element relationships.

Example

Intentions

“High”-level intention: A natural language description of where CCC applies.

Example

Advise whenever data is sent over a network connection

Low-level intention: A description of where a CCC applies in terms of *structural* program element relationships.

Example

Advise calls to methods that write to field networkConn

Intentions

“High”-level intention: A natural language description of where CCC applies.

Example

Advise whenever data is sent over a network connection

Low-level intention: A description of where a CCC applies in terms of *structural* program element relationships.

Example

Advise calls to methods that write to field `networkConn`

Intentions

“High”-level intention: A natural language description of where CCC applies.

Example

Advise whenever data is sent over a network connection

Low-level intention: A description of where a CCC applies in terms of *structural* program element relationships.

Example

Advise calls to methods that write to field `networkConn`

Pointcut Expressions

Join point shadow: Base-code corresponding to a join point.

Pointcut: A set of join point shadows.

Assume any dynamic condition always evaluates to *true*.

Example

```
execution(* x()) ≡ {C.x() {...}, D.x() {...}}
```



InfoLab21

Pointcut Expressions

Join point shadow: Base-code corresponding to a join point.

Pointcut: A set of join point shadows.

Conservative Approach

Assume any dynamic condition always evaluates to *true*.

Example

```
execution(* x())  $\equiv$  {C.x() {...}, D.x() {...}}
```



InfoLab21

Pointcut Expressions

Join point shadow: Base-code corresponding to a join point.

Pointcut: A set of join point shadows.

Conservative Approach

Assume any dynamic condition always evaluates to *true*.

Example

```
execution(* x())  $\equiv$  {C.x() {...}, D.x() {...}}
```



Pointcut Expressions

Join point shadow: Base-code corresponding to a join point.

Pointcut: A set of join point shadows.

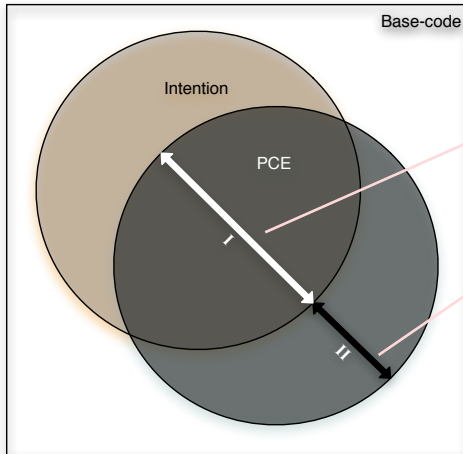
Conservative Approach

Assume any dynamic condition always evaluates to *true*.

Example

$$\text{execution}(* x()) \equiv \{C.x() \{..\}, D.x() \{..\}\}$$


Metrics: Two Dimensions



Metric I: Coverage

How well does PCE cover entire intention?

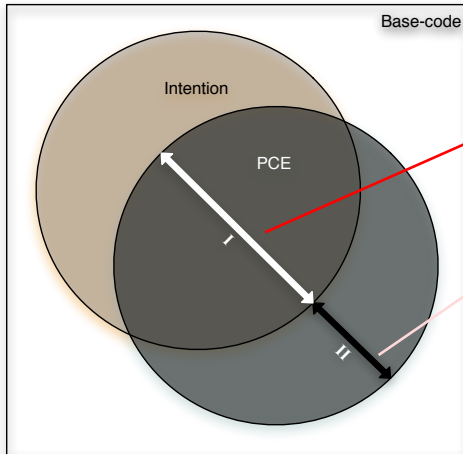
Metric II: Precision

How well does PCE capture solely the intention?



InfoLab21

Metrics: Two Dimensions



Metric I: Coverage

How well does PCE cover entire intention?

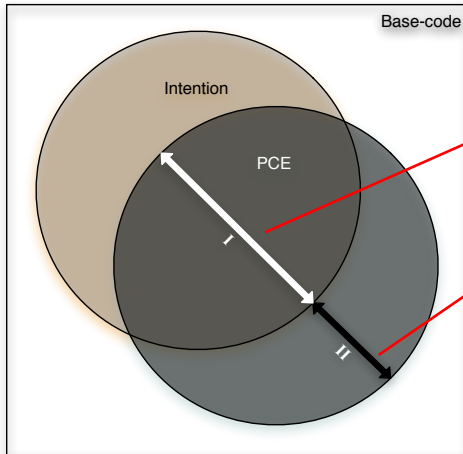
Metric II: Precision

How well does PCE capture solely the intention?



InfoLab21

Metrics: Two Dimensions



Metric I: Coverage

How well does PCE cover entire intention?

Metric II: Precision

How well does PCE capture solely the intention?



InfoLab21

Intention Matching Approach

- Capture rich program element relationships in a *concern graph*[1] adapted for AOP.
- Express intentions as *patterns* over finite, acyclic paths in the graph.
- Apply the patterns to the extended concern graph.



Intention Matching Approach

- Capture rich program element relationships in a *concern graph*[1] adapted for AOP.
- Express intentions as *patterns* over finite, acyclic paths in the graph.
- Apply the patterns to the extended concern graph.



Intention Matching Approach

- Capture rich program element relationships in a *concern graph*[1] adapted for AOP.
- Express intentions as *patterns* over finite, acyclic paths in the graph.
- Apply the patterns to the extended concern graph.



Metric Formalisms

Formalism

\mathcal{A}_{pce} A pointcut expression

$\hat{\pi}$ A pattern

$CG_{\mathcal{P}}^+$ An extended concern graph

$Paths(CG_{\mathcal{P}}^+)$ Finite acyclic paths in $CG_{\mathcal{P}}^+$

$$M_1(\hat{\pi}, \mathcal{A}_{pce}) = \frac{|\mathcal{A}_{pce} \cap Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}{|\mathcal{A}_{pce}|}$$

$$M_2(\hat{\pi}, \mathcal{A}_{pce}) = \frac{|\mathcal{A}_{pce} \cap Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}{|Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}$$



Metric Formalisms

Formalism

\mathcal{A}_{pce} A pointcut expression

$\hat{\pi}$ A pattern

$CG_{\mathcal{P}}^+$ An extended concern graph

$Paths(CG_{\mathcal{P}}^+)$ Finite acyclic paths in $CG_{\mathcal{P}}^+$

$$M_1(\hat{\pi}, \mathcal{A}_{pce}) = \frac{|\mathcal{A}_{pce} \cap Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}{|\mathcal{A}_{pce}|}$$

$$M_2(\hat{\pi}, \mathcal{A}_{pce}) = \frac{|\mathcal{A}_{pce} \cap Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}{|Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}$$



Metric Formalisms

Formalism

\mathcal{A}_{pce} A pointcut expression

$\hat{\pi}$ A pattern

$CG_{\mathcal{P}}^+$ An extended concern graph

$Paths(CG_{\mathcal{P}}^+)$ Finite acyclic paths in $CG_{\mathcal{P}}^+$

$$M_1(\hat{\pi}, \mathcal{A}_{pce}) = \frac{|\mathcal{A}_{pce} \cap Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}{|\mathcal{A}_{pce}|}$$

$$M_2(\hat{\pi}, \mathcal{A}_{pce}) = \frac{|\mathcal{A}_{pce} \cap Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}{|Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}$$



Metric Formalisms

Formalism

\mathcal{A}_{pce} A pointcut expression

$\hat{\pi}$ A pattern

$CG_{\mathcal{P}}^+$ An extended concern graph

$Paths(CG_{\mathcal{P}}^+)$ Finite acyclic paths in $CG_{\mathcal{P}}^+$

$$M_1(\hat{\pi}, \mathcal{A}_{pce}) = \frac{|\mathcal{A}_{pce} \cap Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}{|\mathcal{A}_{pce}|}$$

$$M_2(\hat{\pi}, \mathcal{A}_{pce}) = \frac{|\mathcal{A}_{pce} \cap Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}{|Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}$$



Metric Formalisms

Formalism

\mathcal{A}_{pce} A pointcut expression

$\hat{\pi}$ A pattern

$CG_{\mathcal{P}}^+$ An extended concern graph

$Paths(CG_{\mathcal{P}}^+)$ Finite acyclic paths in $CG_{\mathcal{P}}^+$

$$M_1(\hat{\pi}, \mathcal{A}_{pce}) = \frac{|\mathcal{A}_{pce} \cap Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}{|\mathcal{A}_{pce}|}$$

$$M_2(\hat{\pi}, \mathcal{A}_{pce}) = \frac{|\mathcal{A}_{pce} \cap Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}{|Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}$$



Metric Formalisms

Formalism

\mathcal{A}_{pce} A pointcut expression

$\hat{\pi}$ A pattern

$CG_{\mathcal{P}}^+$ An extended concern graph

$Paths(CG_{\mathcal{P}}^+)$ Finite acyclic paths in $CG_{\mathcal{P}}^+$

$$M_1(\hat{\pi}, \mathcal{A}_{pce}) = \frac{|\mathcal{A}_{pce} \cap Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}{|\mathcal{A}_{pce}|}$$

$$M_2(\hat{\pi}, \mathcal{A}_{pce}) = \frac{|\mathcal{A}_{pce} \cap Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}{|Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}$$



Metric Formalisms

Formalism

\mathcal{A}_{pce} A pointcut expression

$\hat{\pi}$ A pattern

$CG_{\mathcal{P}}^+$ An extended concern graph

$Paths(CG_{\mathcal{P}}^+)$ Finite acyclic paths in $CG_{\mathcal{P}}^+$

$$M_1(\hat{\pi}, \mathcal{A}_{pce}) = \frac{|\mathcal{A}_{pce} \cap Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}{|\mathcal{A}_{pce}|}$$

$$M_2(\hat{\pi}, \mathcal{A}_{pce}) = \frac{|\mathcal{A}_{pce} \cap Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}{|Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}$$



Metric Formalisms

Formalism

\mathcal{A}_{pce} A pointcut expression

$\hat{\pi}$ A pattern

$CG_{\mathcal{P}}^+$ An extended concern graph

$Paths(CG_{\mathcal{P}}^+)$ Finite acyclic paths in $CG_{\mathcal{P}}^+$

$$M_1(\hat{\pi}, \mathcal{A}_{pce}) = \frac{|\mathcal{A}_{pce} \cap Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}{|\mathcal{A}_{pce}|}$$

$$M_2(\hat{\pi}, \mathcal{A}_{pce}) = \frac{|\mathcal{A}_{pce} \cap Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}{|Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}$$



Metric Formalisms

Formalism

\mathcal{A}_{pce} A pointcut expression

$\hat{\pi}$ A pattern

$CG_{\mathcal{P}}^+$ An extended concern graph

$Paths(CG_{\mathcal{P}}^+)$ Finite acyclic paths in $CG_{\mathcal{P}}^+$

$$M_1(\hat{\pi}, \mathcal{A}_{pce}) = \frac{|\mathcal{A}_{pce} \cap Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}{|\mathcal{A}_{pce}|}$$

$$M_2(\hat{\pi}, \mathcal{A}_{pce}) = \frac{|\mathcal{A}_{pce} \cap Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}{|Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}$$



Metric Formalisms

Formalism

\mathcal{A}_{pce} A pointcut expression

$\hat{\pi}$ A pattern

$CG_{\mathcal{P}}^+$ An extended concern graph

$Paths(CG_{\mathcal{P}}^+)$ Finite acyclic paths in $CG_{\mathcal{P}}^+$

$$M_1(\hat{\pi}, \mathcal{A}_{pce}) = \frac{|\mathcal{A}_{pce} \cap Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}{|\mathcal{A}_{pce}|}$$

$$M_2(\hat{\pi}, \mathcal{A}_{pce}) = \frac{|\mathcal{A}_{pce} \cap Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}{|Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}$$



Metric Formalisms

Formalism

\mathcal{A}_{pce} A pointcut expression

$\hat{\pi}$ A pattern

$CG_{\mathcal{P}}^+$ An extended concern graph

$Paths(CG_{\mathcal{P}}^+)$ Finite acyclic paths in $CG_{\mathcal{P}}^+$

$$M_1(\hat{\pi}, \mathcal{A}_{pce}) = \frac{|\mathcal{A}_{pce} \cap Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}{|\mathcal{A}_{pce}|}$$

$$M_2(\hat{\pi}, \mathcal{A}_{pce}) = \frac{|\mathcal{A}_{pce} \cap Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}{|Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}$$



Metric Formalisms

Formalism

\mathcal{A}_{pce} A pointcut expression

$\hat{\pi}$ A pattern

$CG_{\mathcal{P}}^+$ An extended concern graph

$Paths(CG_{\mathcal{P}}^+)$ Finite acyclic paths in $CG_{\mathcal{P}}^+$

$$M_1(\hat{\pi}, \mathcal{A}_{pce}) = \frac{|\mathcal{A}_{pce} \cap Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}{|\mathcal{A}_{pce}|}$$

$$M_2(\hat{\pi}, \mathcal{A}_{pce}) = \frac{|\mathcal{A}_{pce} \cap Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}{|Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}$$



Metric Formalisms

Formalism

\mathcal{A}_{pce} A pointcut expression

$\hat{\pi}$ A pattern

$CG_{\mathcal{P}}^+$ An extended concern graph

$Paths(CG_{\mathcal{P}}^+)$ Finite acyclic paths in $CG_{\mathcal{P}}^+$

$$M_1(\hat{\pi}, \mathcal{A}_{pce}) = \frac{|\mathcal{A}_{pce} \cap Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}{|\mathcal{A}_{pce}|}$$

$$M_2(\hat{\pi}, \mathcal{A}_{pce}) = \frac{|\mathcal{A}_{pce} \cap Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}{|Match(\hat{\pi}, Paths(CG_{\mathcal{P}}^+))|}$$



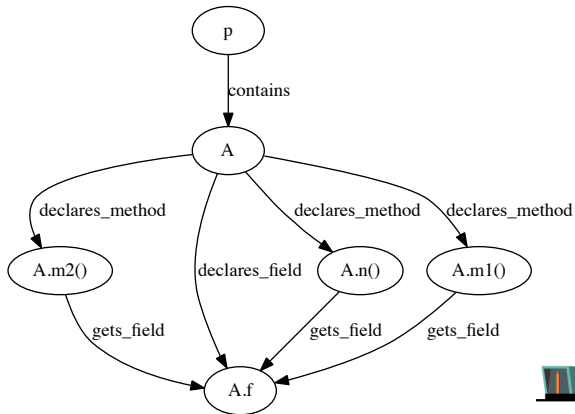
Base-code

```
package p;
public class A {
    int f;
    void m1() {
        int a = f + 1;
    }
    void m2() {
        int b = f + 2;
    }
    void n() {
        int c = f + 3;
    }
}
```



InfoLab21

Concern Graph



InfoLab21

An Intention

Low-level Intention of where CCC Applies

To advise all method executions that read from field A.f



InfoLab21

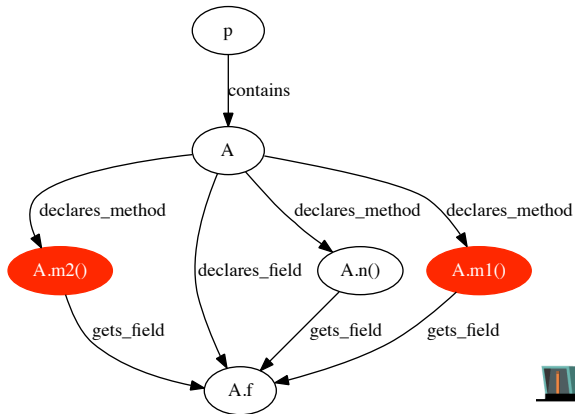
An Aspect

```
package p;
public aspect B {
    before() : execution(* m*(..)) {
    }
}
```



InfoLab21

Concern Graph



InfoLab21

Metric M_1

$$\begin{aligned}
 M_1(\hat{\pi}, \mathcal{A}_{pce}) &= \frac{|\mathcal{A}_{pce} \cap \text{Match}(\hat{\pi}, \text{Paths}(CG_P^+))|}{|\mathcal{A}_{pce}|} \\
 &= \frac{|\{A.m1(), A.m2()\} \cap \{A.m1(), A.m2(), A.n()\}|}{|\{A.m1(), A.m2()\}|} \\
 &= 1
 \end{aligned}$$



InfoLab21

Metric M_1

$$\begin{aligned}
 M_1(\hat{\pi}, \mathcal{A}_{pce}) &= \frac{|\mathcal{A}_{pce} \cap \text{Match}(\hat{\pi}, \text{Paths}(CG_P^+))|}{|\mathcal{A}_{pce}|} \\
 &= \frac{|\{\text{A.m1}(), \text{A.m2}()\} \cap \{\text{A.m1}(), \text{A.m2}(), \text{A.n}()\}|}{|\{\text{A.m1}(), \text{A.m2}()\}|} \\
 &= 1
 \end{aligned}$$



InfoLab21

Metric M_1

$$\begin{aligned}
 M_1(\hat{\pi}, \mathcal{A}_{pce}) &= \frac{|\mathcal{A}_{pce} \cap Match(\hat{\pi}, Paths(CG_P^+))|}{|\mathcal{A}_{pce}|} \\
 &= \frac{|\{A.m1(), A.m2()\} \cap \{A.m1(), A.m2(), A.n()\}|}{|\{A.m1(), A.m2()\}|} \\
 &= 1
 \end{aligned}$$



InfoLab21

Metric M_2

$$\begin{aligned}
 M_2(\hat{\pi}, \mathcal{A}_{pce}) &= \frac{|\mathcal{A}_{pce} \cap \text{Match}(\hat{\pi}, \text{Paths}(CG_P^+))|}{|\text{Match}(\hat{\pi}, \text{Paths}(CG_P^+))|} \\
 &= \frac{|\{A.m1(), A.m2()\} \cap \{A.m1(), A.m2(), A.n()\}|}{|\{A.m1(), A.m2(), A.n()\}|} \\
 &= \frac{2}{3}
 \end{aligned}$$



InfoLab21

Metric M_2

$$\begin{aligned}
 M_2(\hat{\pi}, \mathcal{A}_{pce}) &= \frac{|\mathcal{A}_{pce} \cap \text{Match}(\hat{\pi}, \text{Paths}(CG_{\mathcal{P}}^+))|}{|\text{Match}(\hat{\pi}, \text{Paths}(CG_{\mathcal{P}}^+))|} \\
 &= \frac{|\{\text{A.m1}(), \text{A.m2}()\} \cap \{\text{A.m1}(), \text{A.m2}(), \text{A.n}()\}|}{|\{\text{A.m1}(), \text{A.m2}(), \text{A.n}()\}|} \\
 &= \frac{2}{3}
 \end{aligned}$$



InfoLab21

Metric M_2

$$\begin{aligned}
 M_2(\hat{\pi}, \mathcal{A}_{pce}) &= \frac{|\mathcal{A}_{pce} \cap \text{Match}(\hat{\pi}, \text{Paths}(CG_{\mathcal{P}}^+))|}{|\text{Match}(\hat{\pi}, \text{Paths}(CG_{\mathcal{P}}^+))|} \\
 &= \frac{|\{\text{A.m1}(), \text{A.m2}()\} \cap \{\text{A.m1}(), \text{A.m2}(), \text{A.n}()\}|}{|\{\text{A.m1}(), \text{A.m2}(), \text{A.n}()\}|} \\
 &= \frac{2}{3}
 \end{aligned}$$



Conclusion

- Capturing **true** intentions of where a CCC applies may alleviate fragility.
- Would like to **measure** how well a PCE **captures** a developer's *intentions*.



InfoLab21

Conclusion

- Capturing **true** intentions of where a CCC applies may alleviate fragility.
- Would like to **measure** how well a PCE **captures** a developer's *intentions*.



InfoLab21

Future Work

- Automatic inference of low-level intentions from high-level ones?
- How to encode all high-level intentions?
- Do metrics help alleviate fragility?
- Implementation/evaluation.



Future Work

- Automatic inference of low-level intentions from high-level ones?
- How to encode all high-level intentions?
- Do metrics help alleviate fragility?
- Implementation/evaluation.



InfoLab21

Future Work

- Automatic inference of low-level intentions from high-level ones?
- How to encode all high-level intentions?
- Do metrics help alleviate fragility?
- Implementation/evaluation.



InfoLab21

Future Work

- Automatic inference of low-level intentions from high-level ones?
- How to encode all high-level intentions?
- Do metrics help alleviate fragility?
- Implementation/evaluation.



References



M. P. Robillard and G. C. Murphy.

Concern graphs: finding and describing concerns using structural program dependencies.

In International Conference on Software Engineering, 2002.



InfoLab21