Fraglight: Shedding Light on Broken Pointcuts Using Structural Commonality

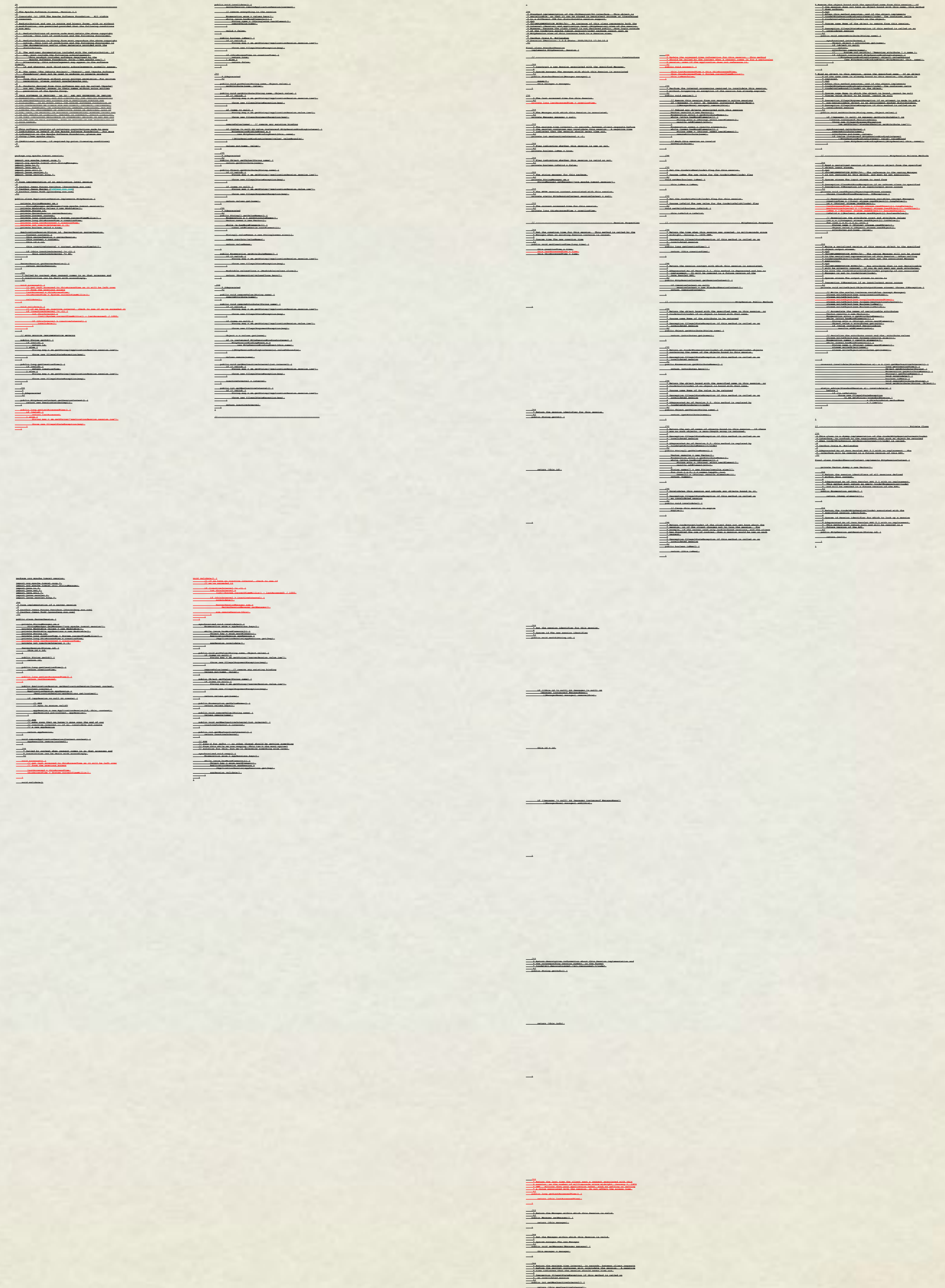
Work in progress

Raffi Khatchadourian[1], Phil Greenwood[2], Awais Rashid[2], Hidehiko Masuhara[3]
[1]New York City College of Technology, [2]Lancaster University, [3]Tokyo Institute of Technology

Crosscutting Concerns

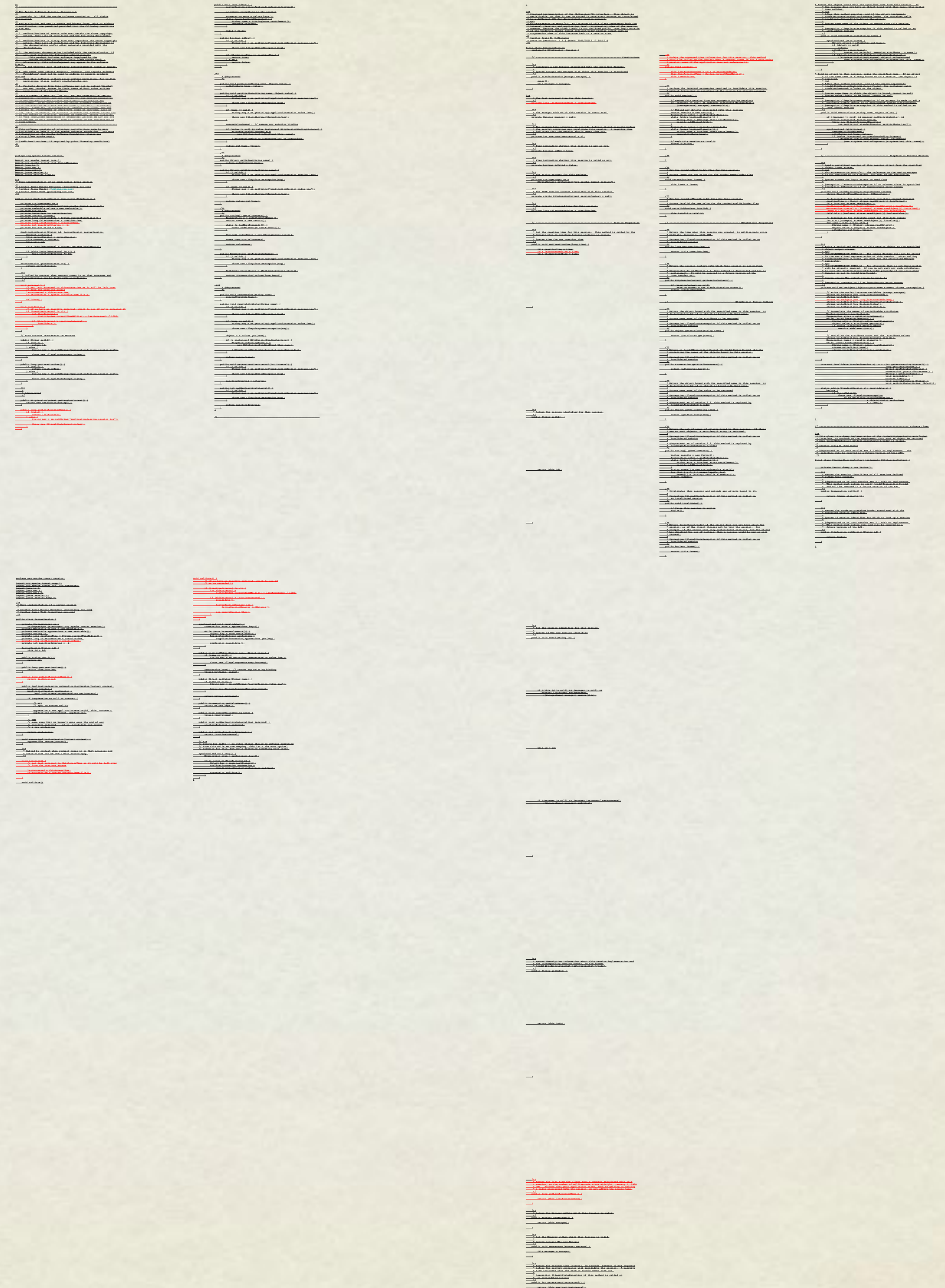
Crosscutting Concerns

- Crosscutting concerns (CCCs) affect many heterogenous software modules.



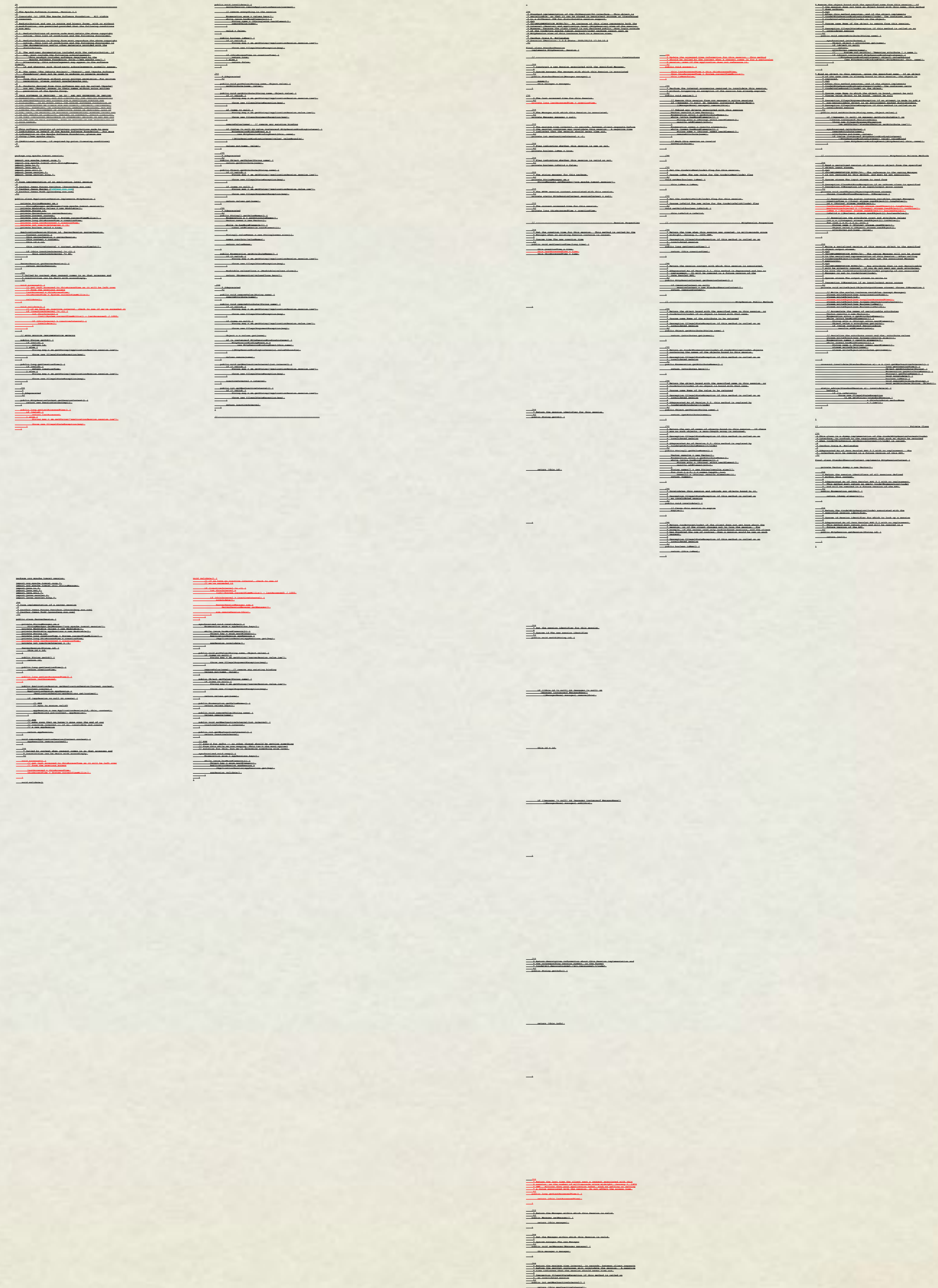
Crosscutting Concerns

- Crosscutting concerns (CCCs) affect many heterogenous software modules.
- Code is:



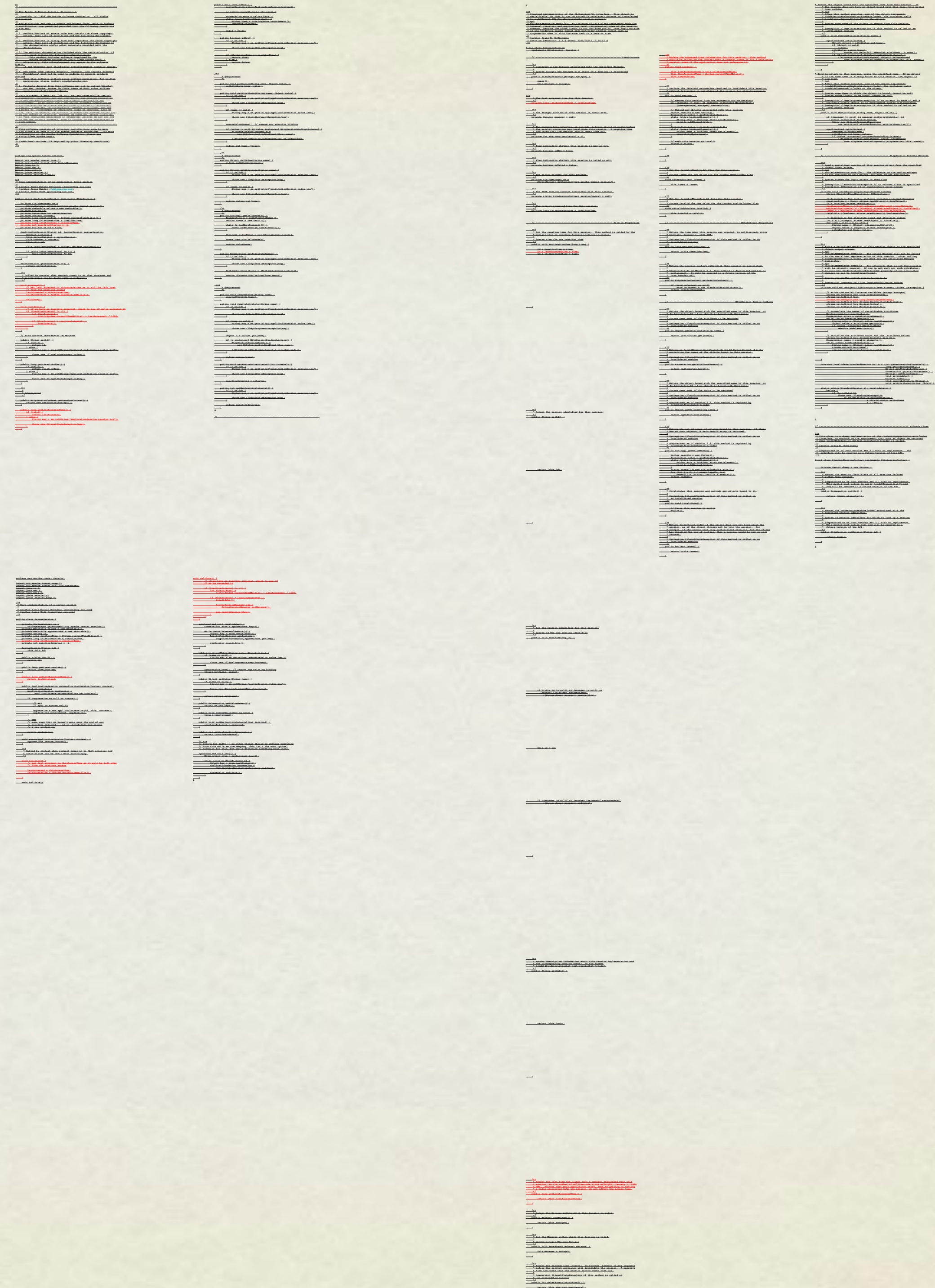
Crosscutting Concerns

- Crosscutting concerns (CCCs) affect many heterogenous software modules.
- Code is:
  - Scattered throughout many modules.



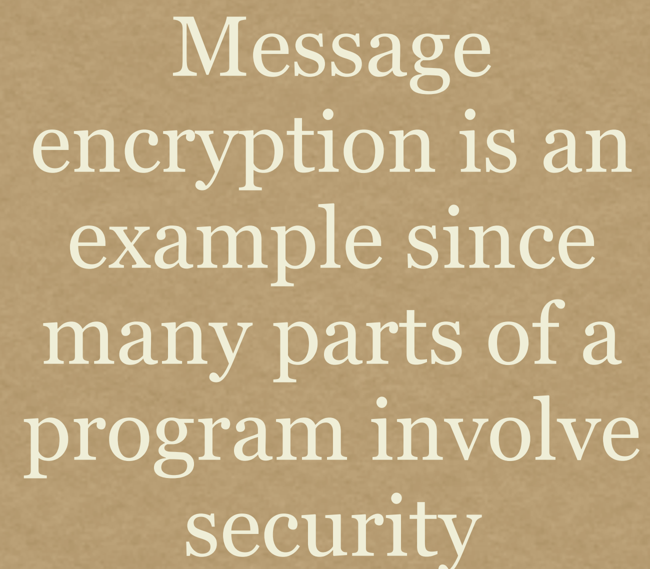
Crosscutting Concerns

- Crosscutting concerns (CCCs) affect many heterogenous software modules.
- Code is:
  - Scattered throughout many modules.
  - Tangled with unrelated modules.



Crosscutting Concerns

- Crosscutting concerns (CCCs) affect many heterogenous software modules.
- Code is:
  - Scattered throughout many modules.
  - Tangled with unrelated modules.



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

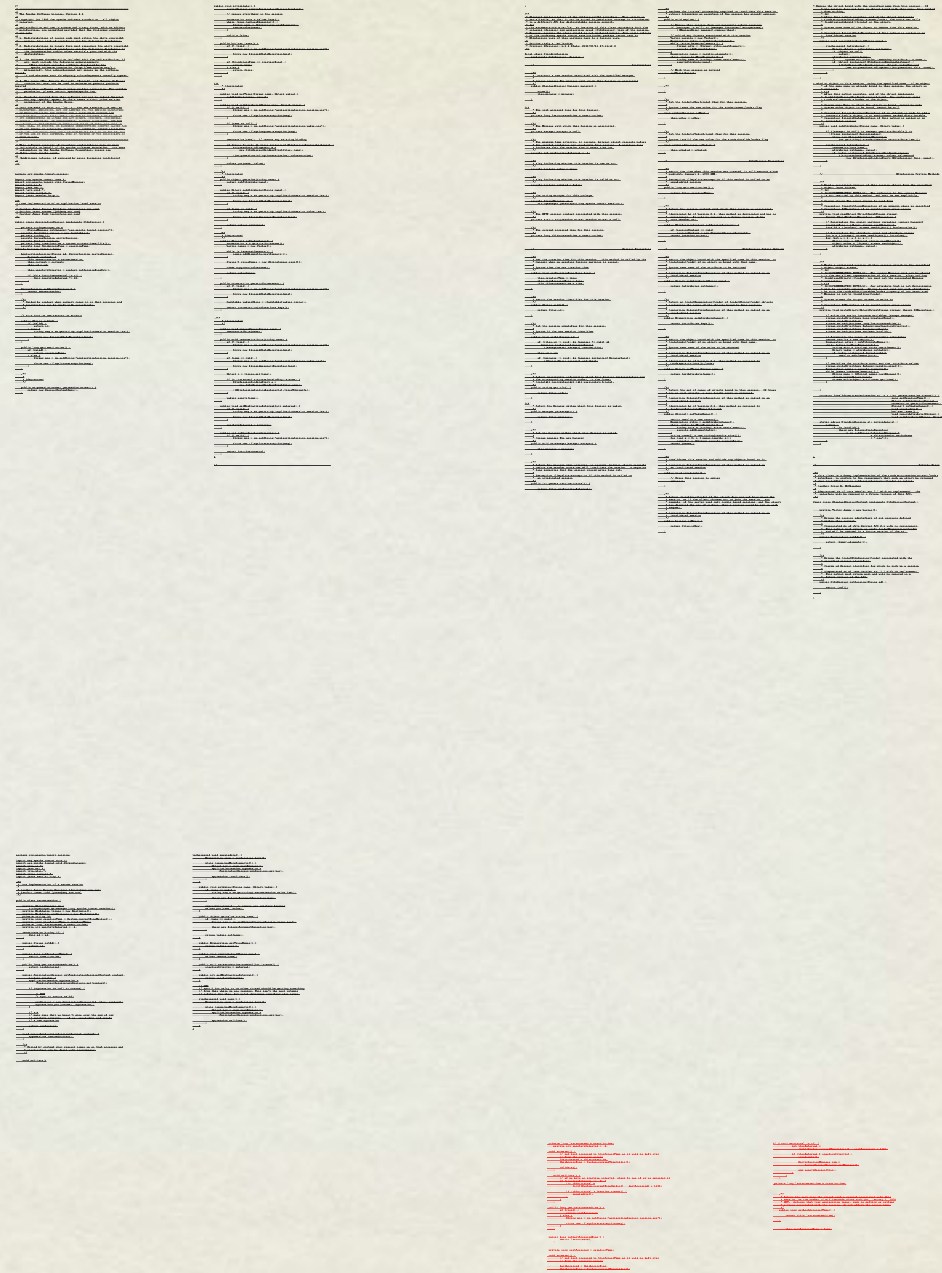
Aspect-Oriented Programming

Aspect-Oriented Programming

- Aspect-Oriented Programming enables localized implementations of CCCs.

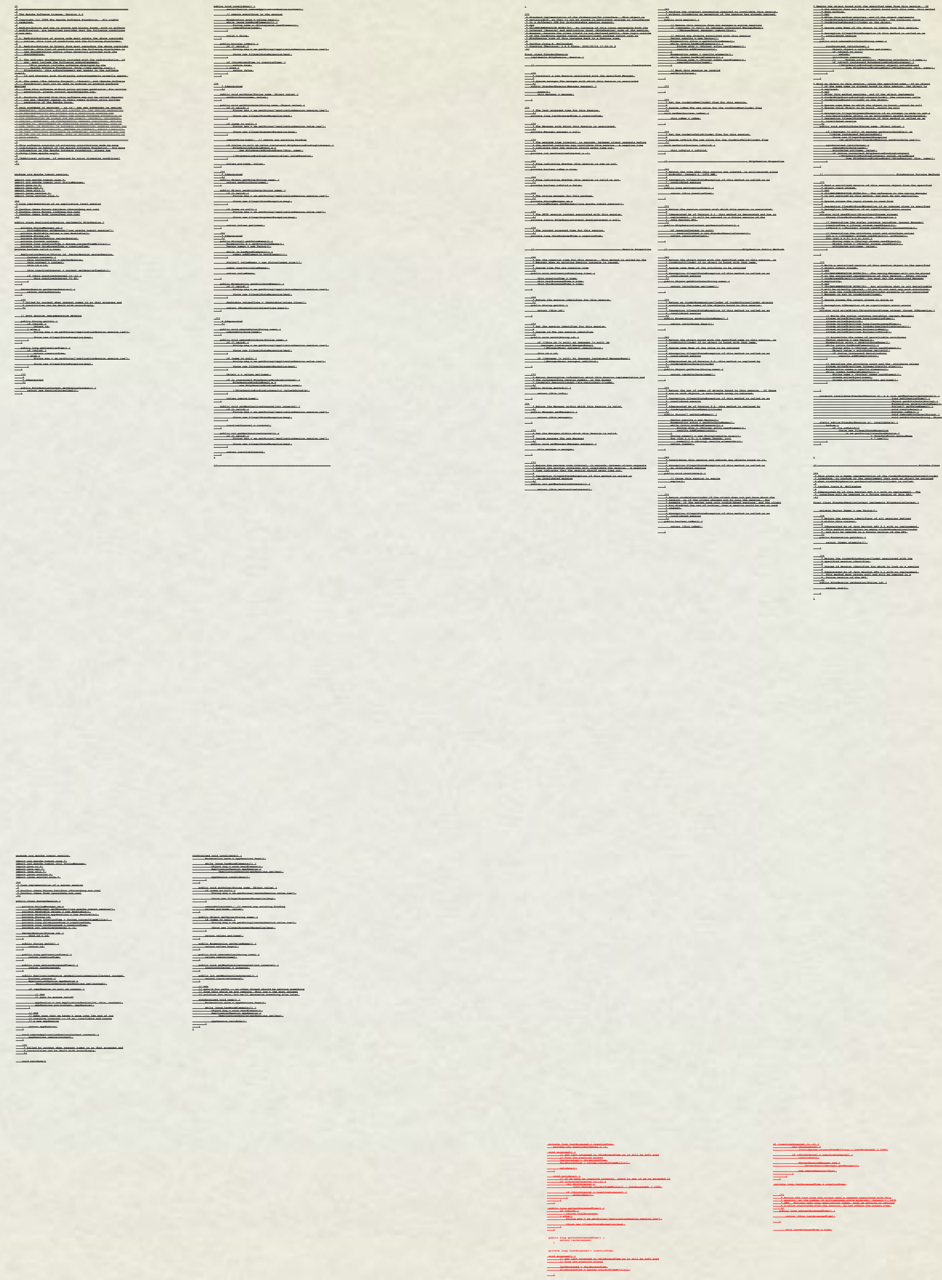
Aspect-Oriented Programming

- Aspect-Oriented Programming enables localized implementations of CCCs.
- Pointcuts select (join) points in the program where a CCC applies.



Aspect-Oriented Programming

- 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.

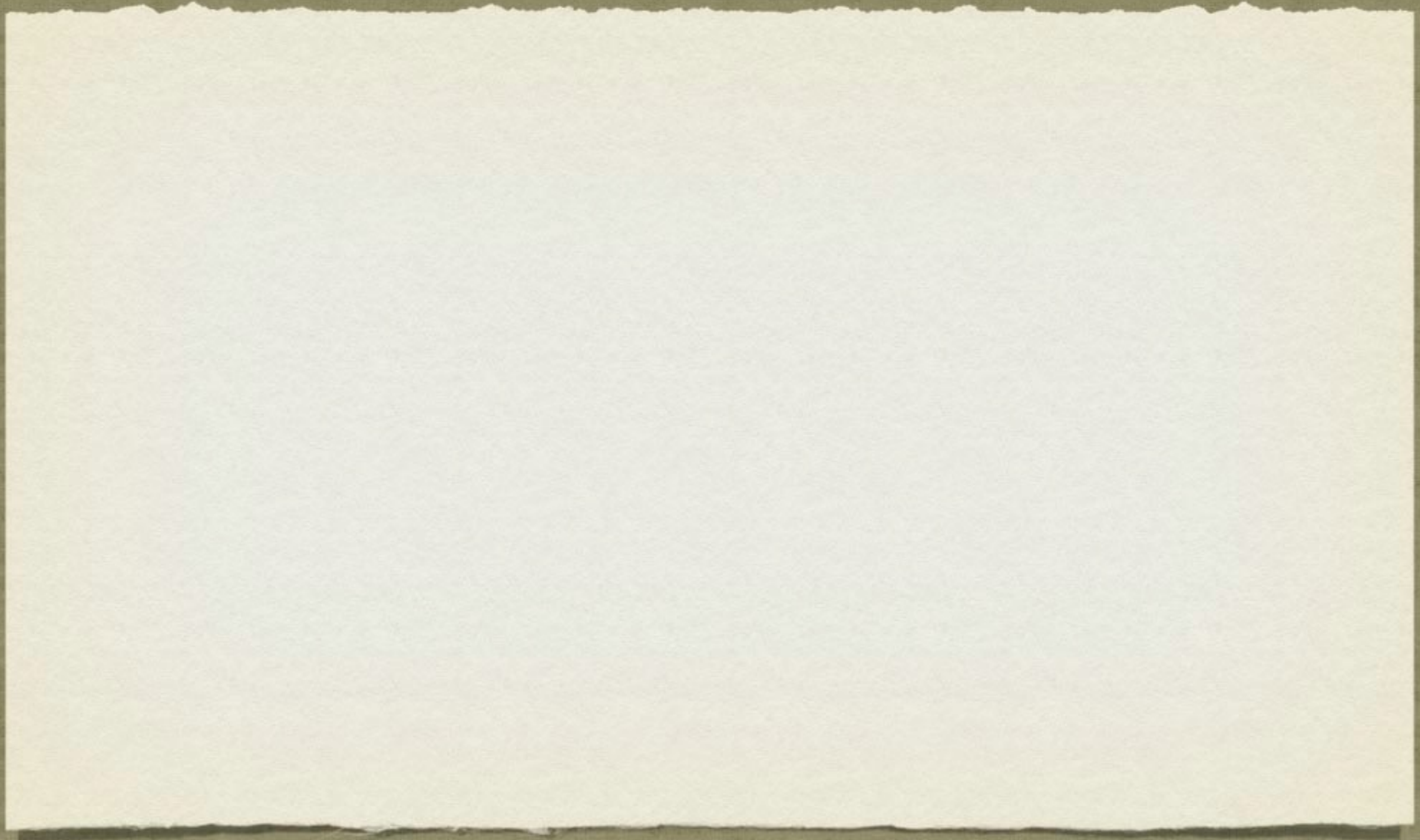


Aspect-Oriented Programming

- 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 Example

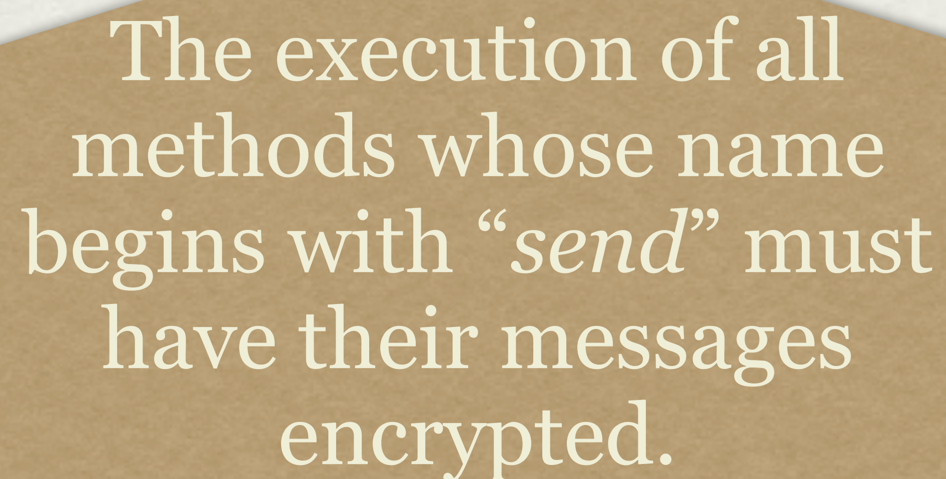


Message Encryption Example

```
messageSent():

execution(* send*(..))
```

Message Encryption Example

```
messageSent():

execution(* send*(..))
```



Fragile Pointcut Problem

Fragile Pointcut Problem

- A pointcut is ***robust*** iff it continues to select points where the CCC applies in subsequent base-code versions *without* textual modification.

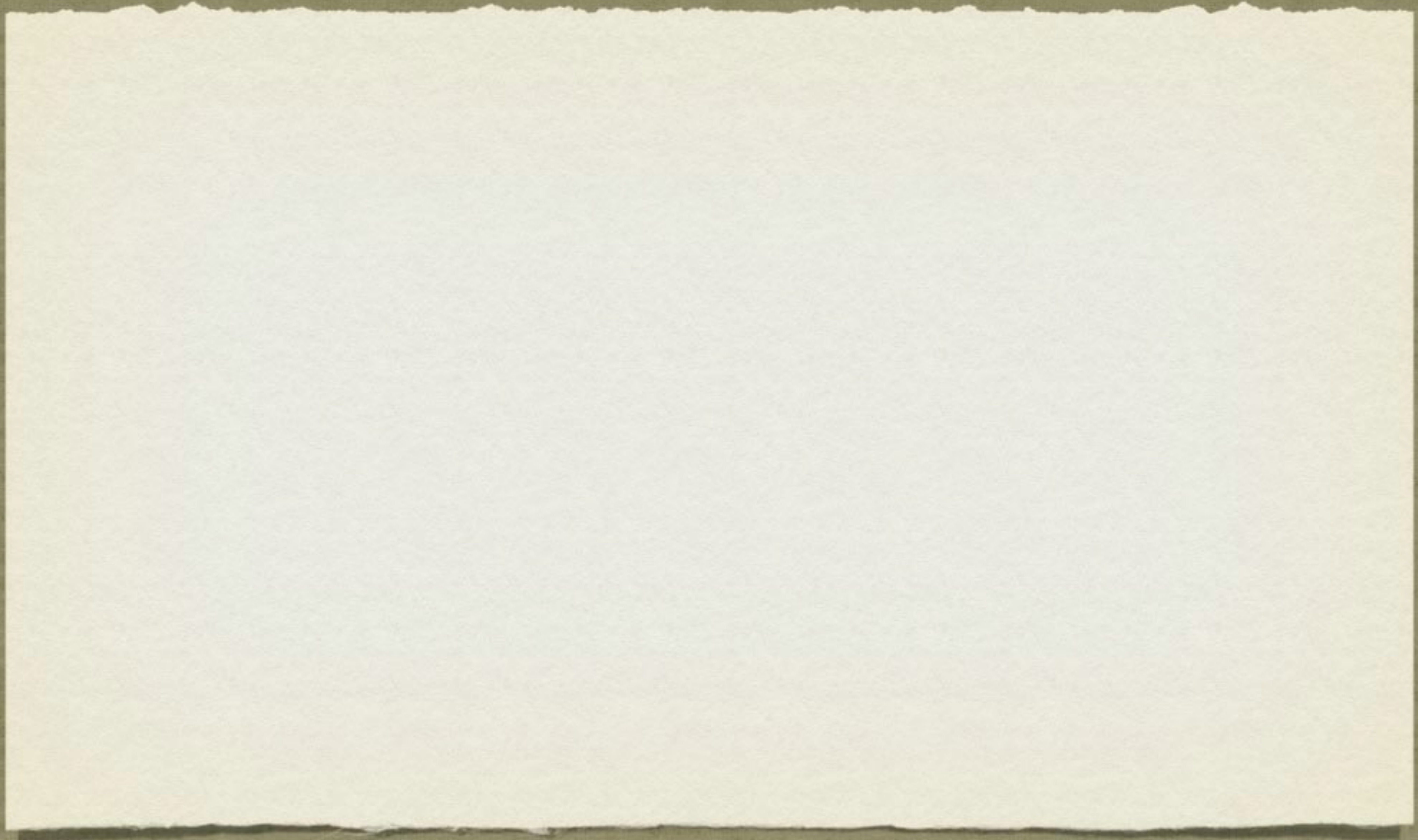
Fragile Pointcut Problem

- A pointcut is ***robust*** iff it continues to select points where the CCC applies in subsequent base-code versions *without* textual modification.
- Otherwise, a pointcut is ***fragile***.

Fragile Pointcut Problem

- A pointcut is ***robust*** iff it continues to select points where the CCC applies in subsequent base-code versions *without* textual modification.
- Otherwise, a pointcut is ***fragile***.
- Fragile pointcuts can cause software to ***silently*** **malfunction**!

Pointcut Fragility Example



Pointcut Fragility Example

- Suppose a subsequent version, a ***new*** method is added that ***sends*** messages but whose name begins with “***transmit***?”

Pointcut Fragility Example

- Suppose a subsequent version, a ***new*** method is added that ***sends*** messages but whose name begins with “***transmit***?”
- Our pointcut ***breaks*** as a result.

Pointcut Fragility Example

- Suppose a subsequent version, a ***new*** method is added that ***sends*** messages but whose name begins with "***transmit***?"
- Our pointcut ***breaks*** as a result.
- Thus, `messageSent()` is a fragile pointcut.

Fixing Broken Pointcuts

Fixing Broken Pointcuts

- Developer must ***manually*** identify ***all*** broken pointcuts following changes to the base-code to ensure the ***software functions*** as ***intended***.

Fixing Broken Pointcuts

- Developer must ***manually*** identify ***all*** broken pointcuts following changes to the base-code to ensure the ***software functions*** as ***intended***.

Can be ***tedious***, ***time-consuming***, ***error-prone***, and ***omission-prone*** when there are ***many pointcuts***!

Fixing Broken Pointc[...]

- Developer must ***manually*** identify ***all*** broken pointcuts following changes to the base-code to ensure the ***software functions*** as ***intended***.

Appox. 12.95 pointcuts per project

Can be ***tedious***, ***time-consuming***, ***error-prone***, and ***omission-prone*** when there are ***many pointcuts***!

Broken Pointc

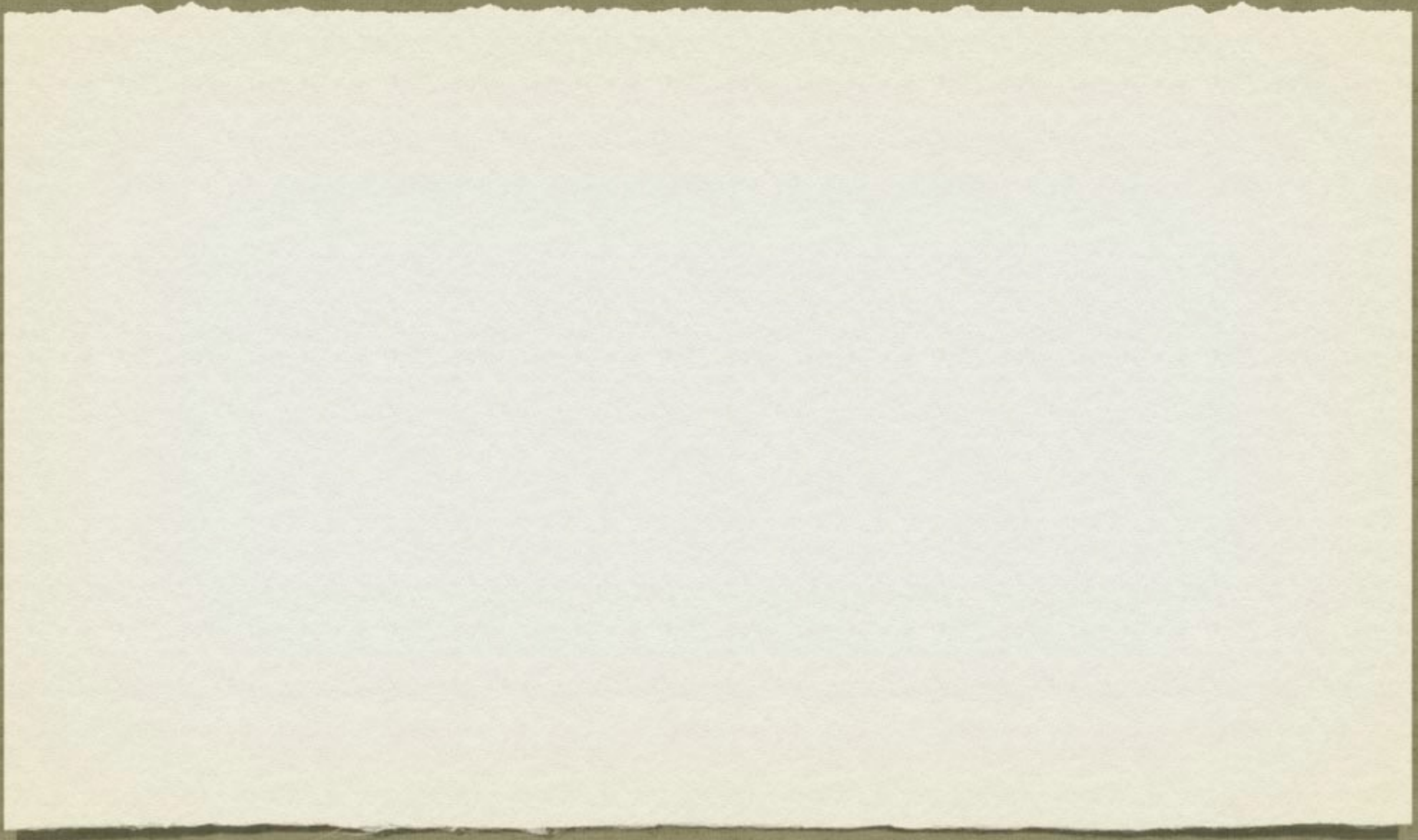
On average, ≈12.95 pointcuts must be checked after ***every*** Δ base-code change!

Appox. 12.95 pointcuts per project

- Developer must ***manually*** identify ***all*** broken pointcuts following changes to the base-code to ensure the ***software functions*** as ***intended***.

Can be ***tedious***, ***time-consuming***, ***error-prone***, and ***omission-prone*** when there are ***many pointcuts***!

Goals



Goals

- ***Automation***: Mechanically alleviate the burden of detecting pointcuts that have broken due to base-code change(s).

Goals

- ***Automation****:* Mechanically alleviate the burden of detecting pointcuts that have broken due to base-code change(s).
- ***On-the-fly prediction:*** Inform the developer when base-code changes may break pointcuts *as he/she is typing*.

Goals

- ***Automation***: Mechanically alleviate the burden of detecting pointcuts that have broken due to base-code change(s).
- ***On-the-fly prediction:*** Inform the developer when base-code changes may break pointcuts *as he/she is typing*.
- ***Conditional Obliviousness:*** Only inform developer if it is *highly likely* that new base-code will break a pointcut.

Goals

How can broken pointcuts be mechanically identified?

- ***Automation****:* Mechanically alleviate the burden of detecting pointcuts that have broken due to base-code change(s).

- ***On-the-fly prediction:*** Inform the developer when base-code changes may break pointcuts *as he/she is typing*.

- ***Conditional Obliviousness:*** Only inform developer if it is *highly likely* that new base-code will break a pointcut.

Goals

How can broken pointcuts be mechanically identified?

May involve ***frequently*** analyzing ***large*** portions of the system!

- ***Automation***: Mechanically alleviate th… detecting pointcuts that have broken… change(s).
- ***On-the-fly prediction:*** Inform the developer when base-code changes may break pointcuts *as he/she is typing*.
- ***Conditional Obliviousness:*** Only inform developer if it is *highly likely* that new base-code will break a pointcut.

Goals

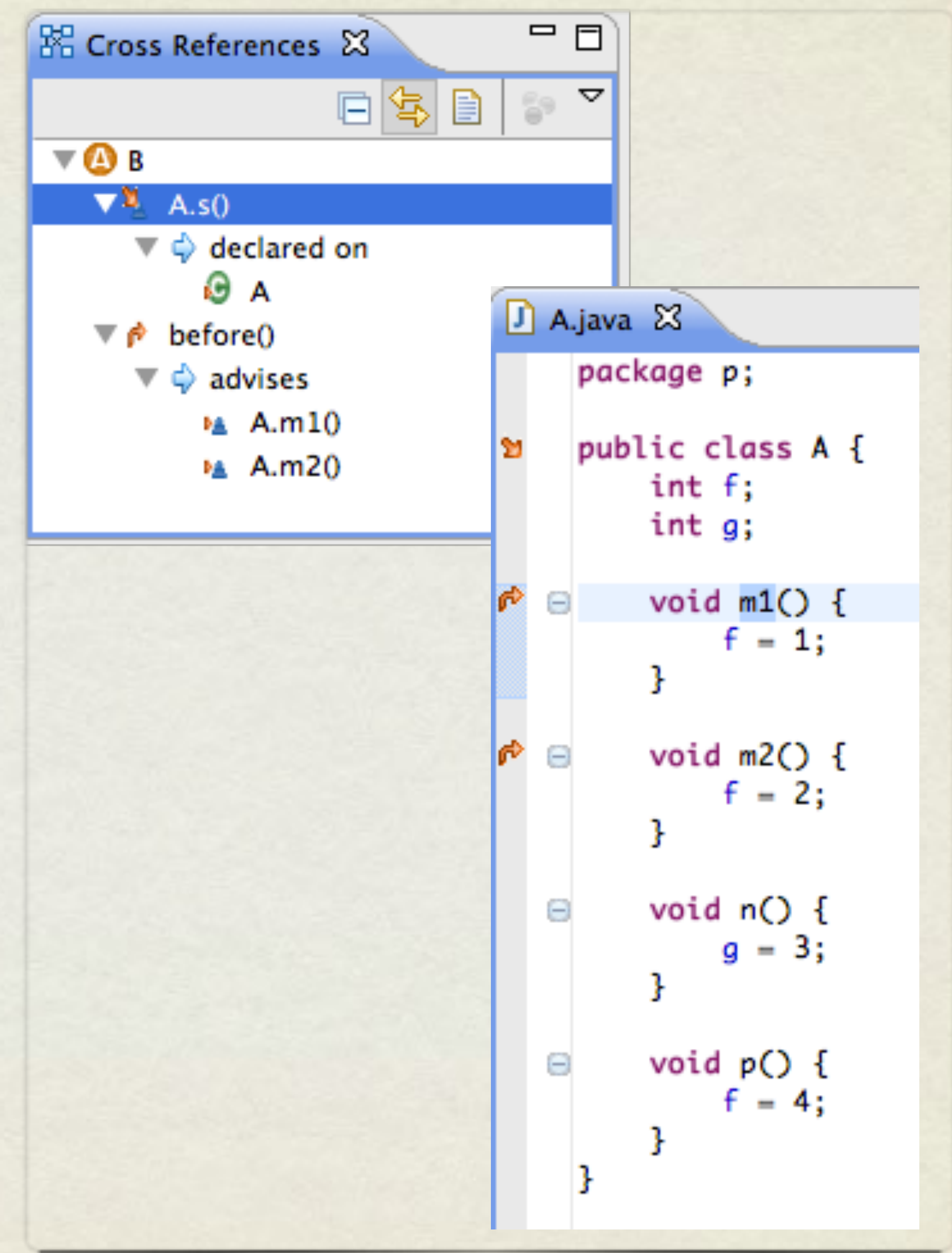
- ***Automation***: Mechanically alleviate t[…] detecting pointcuts [...] broken [...] change(s).
- ***On-the-fly pr***[...] [th]e developer when base-code changes ma[...] *as he/she is typing*.
- ***Conditional Obliviousness:*** Only inform developer if it is *highly likely* that new base-code will break a pointcut.

How can broken pointcuts be mechanically identified?

May involve ***frequently*** analyzing ***large*** portions of the system!

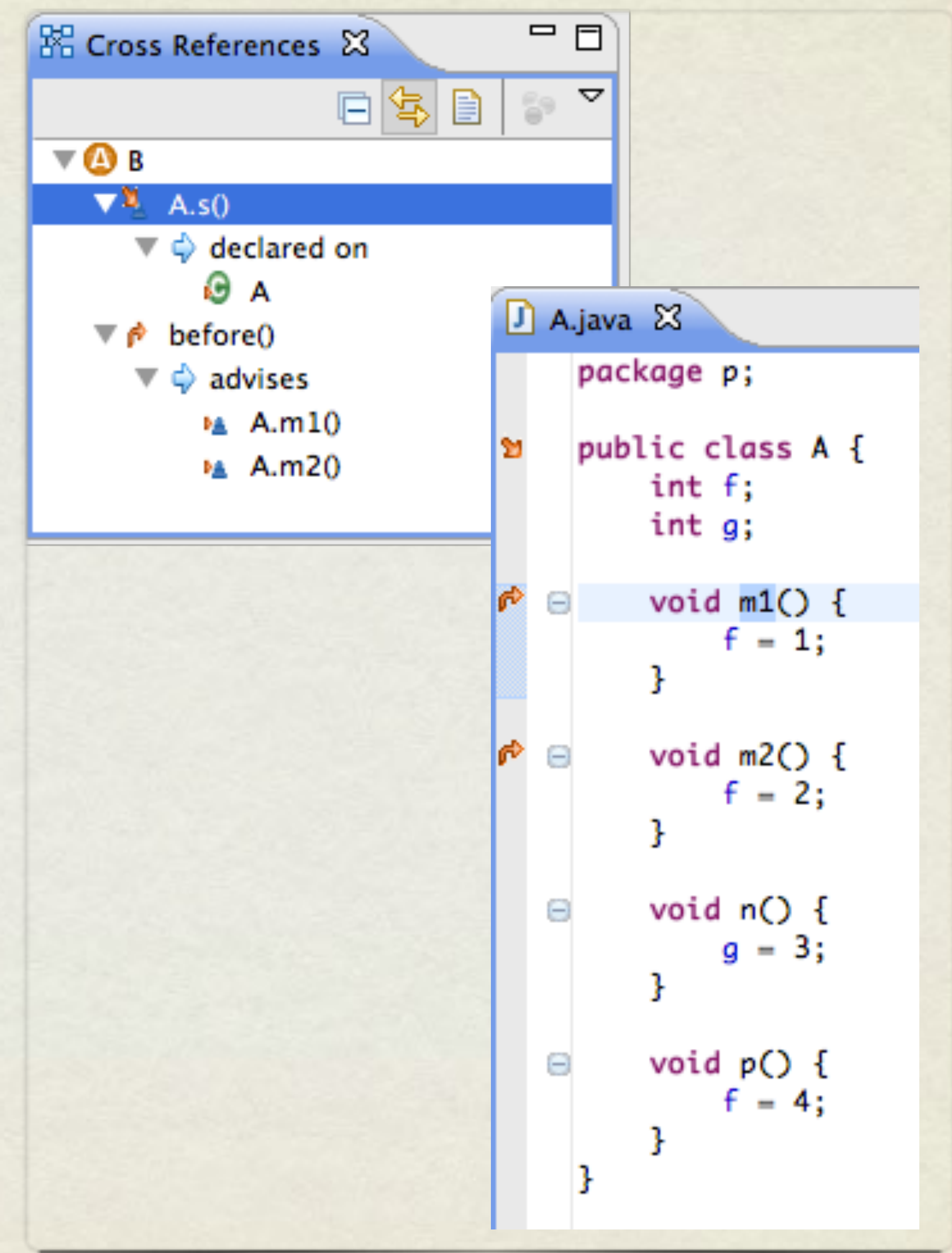
Use *Mylyn* context to promote *interesting* pointcuts in UI

AspectJ Developer Tools



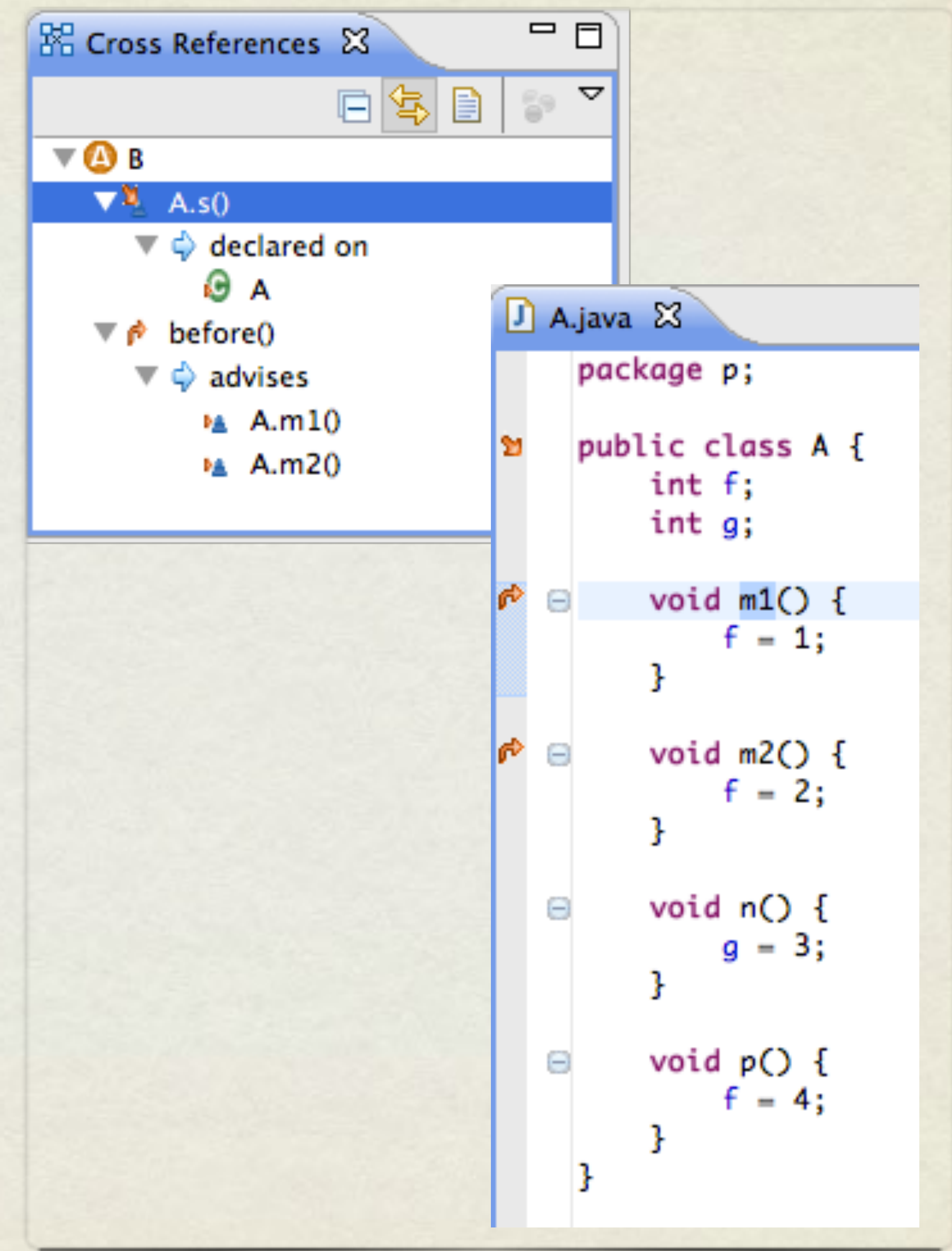
AspectJ Developer Tools

- AJDT can present developers with broken pointcuts to an extent.



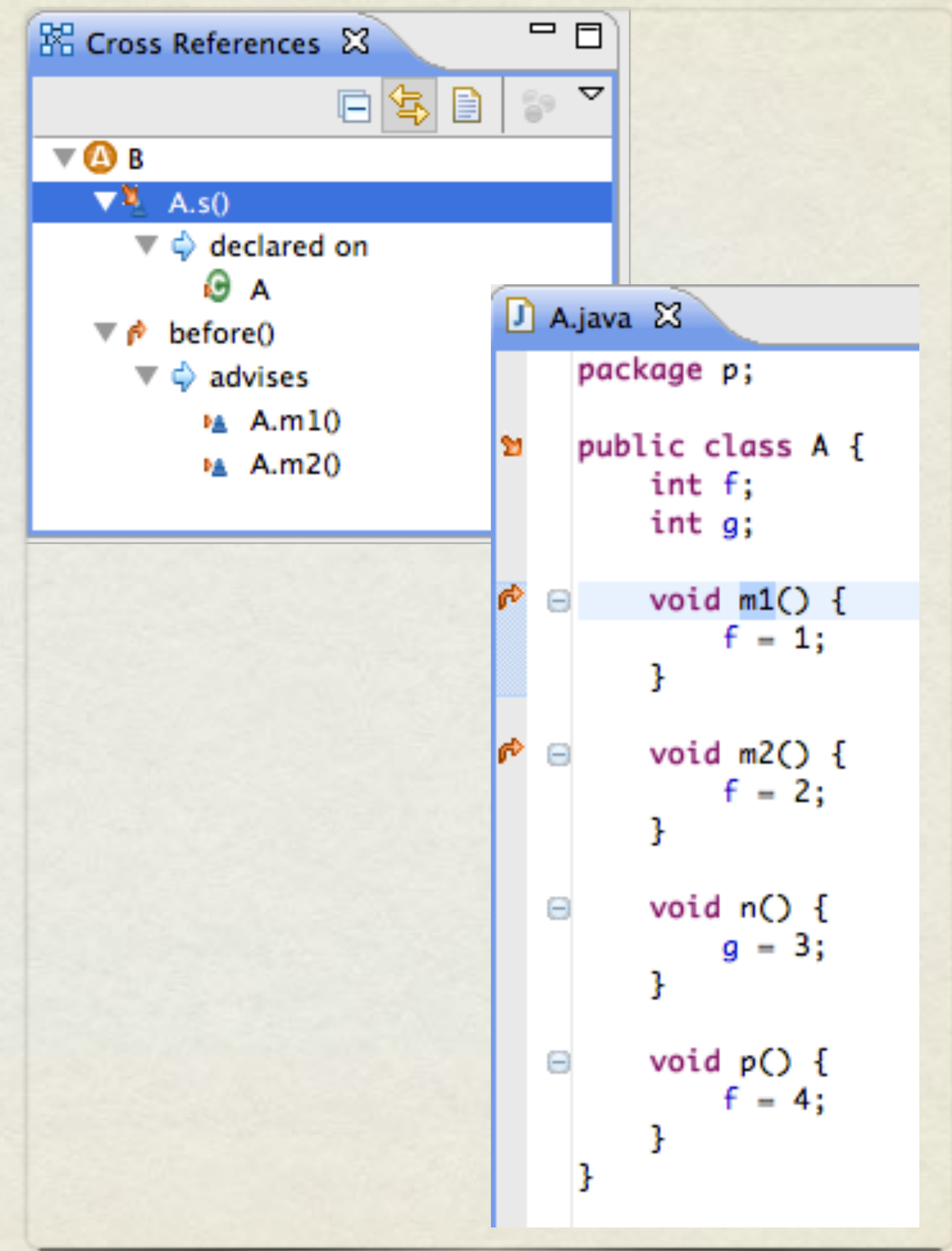
AspectJ Developer Tools

- AJDT can present developers with broken pointcuts to an extent.
- However, AJDT is more general purpose:



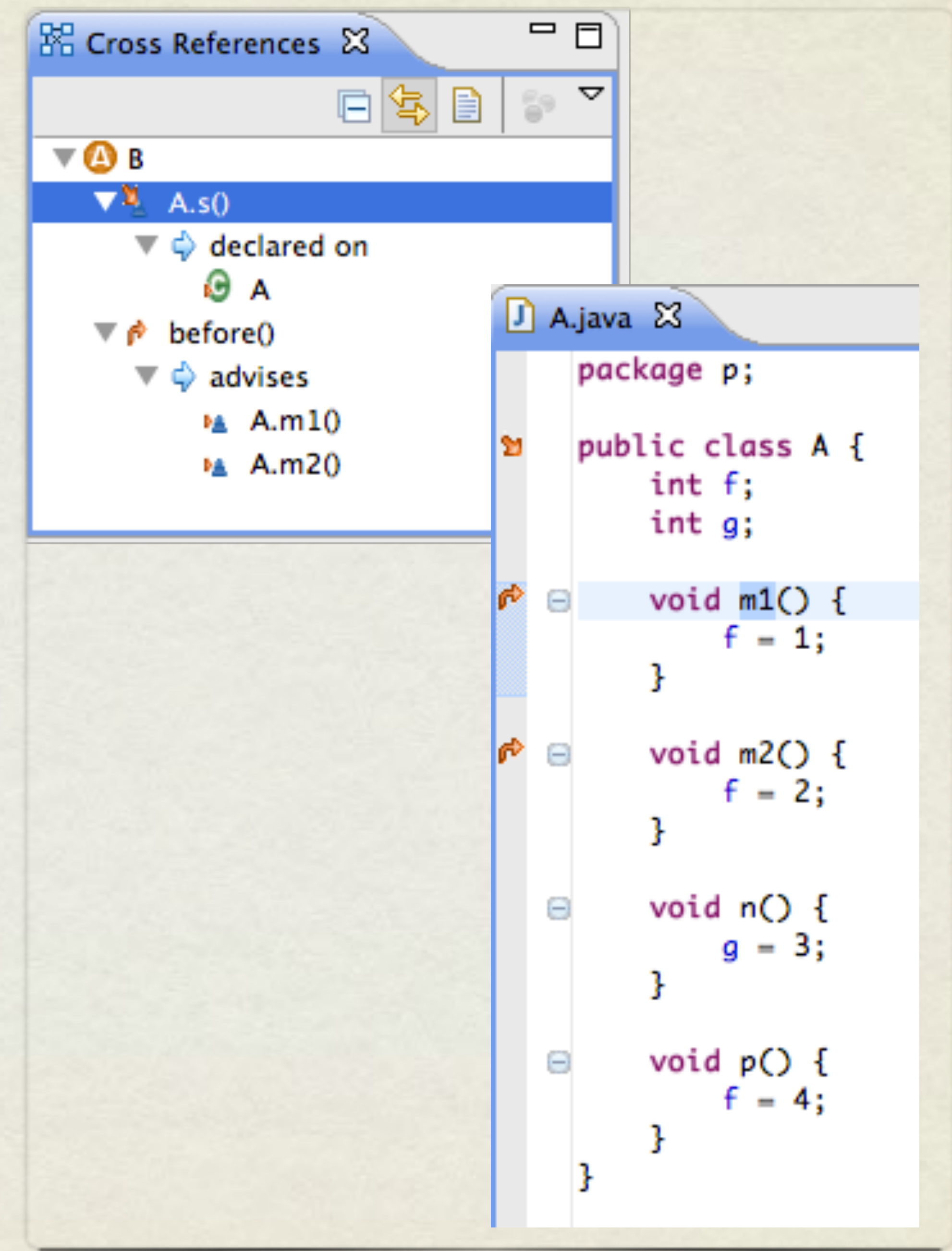
AspectJ Developer Tools

- AJDT can present developers with broken pointcuts to an extent.
- However, AJDT is more general purpose:
  - All matching pointcuts are displayed in the UI, regardless if they are broken.



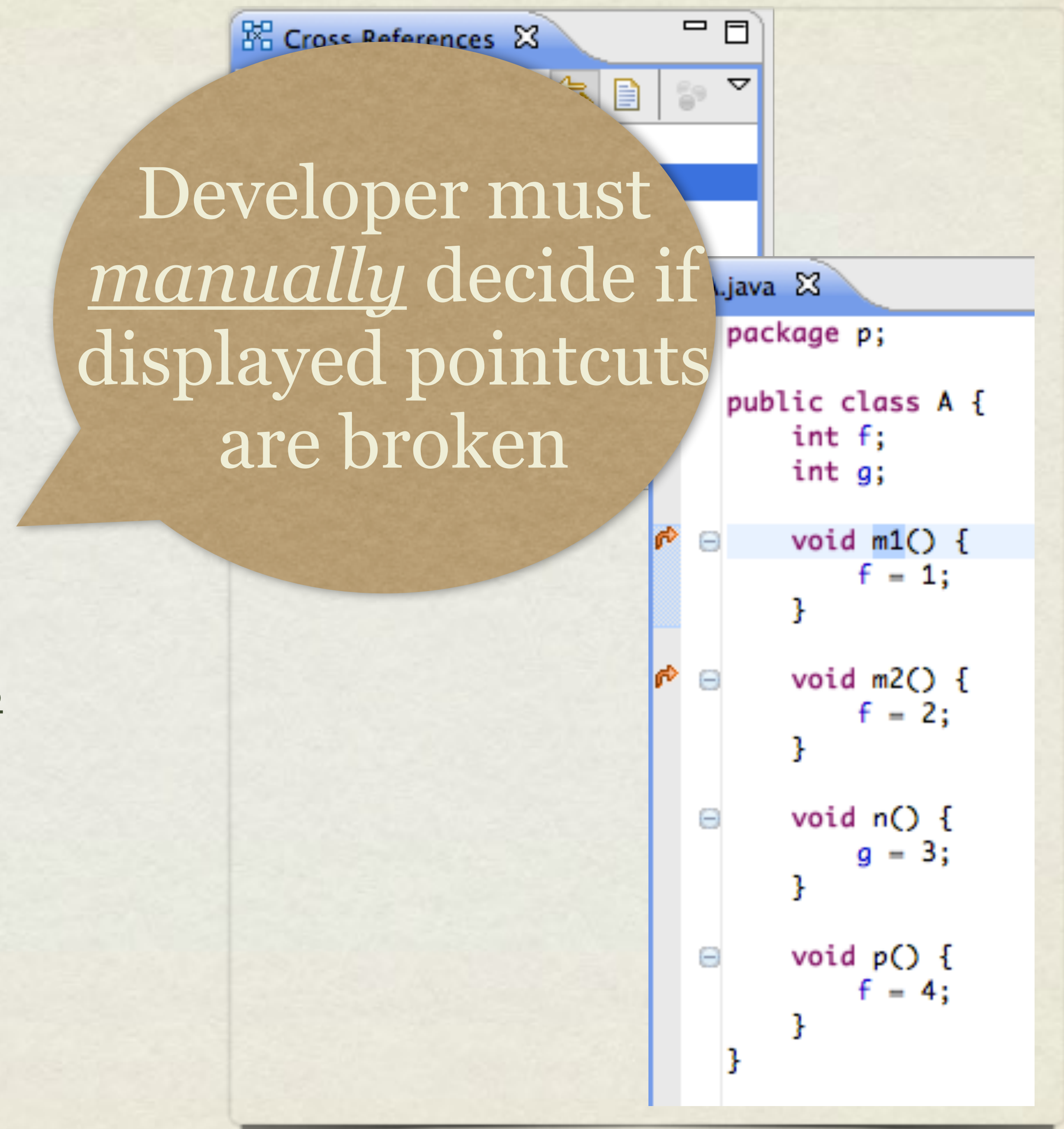
AspectJ Developer Tools

- AJDT can present developers with broken pointcuts to an extent.
- However, AJDT is more general purpose:
  - All matching pointcuts are displayed in the UI, regardless if they are broken.
  - Broken pointcuts that do not match focused base code are not displayed.



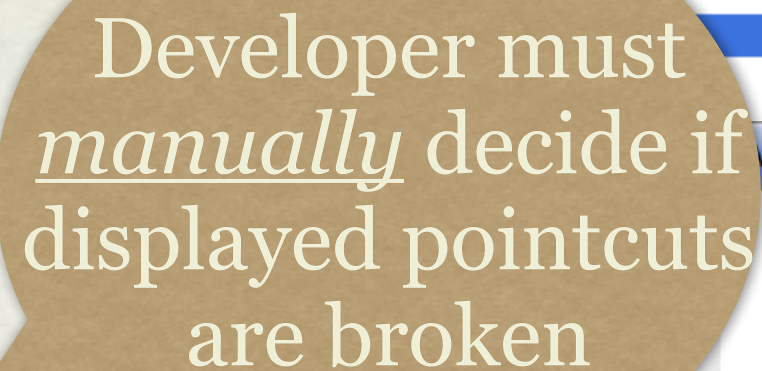
AspectJ Developer Tools

- AJDT can present developers with broken pointcuts to an extent.
- However, AJDT is more general purpose:
  - All matching pointcuts are displayed in the UI, regardless if they are broken.
  - Broken pointcuts that do not match focused base code are not displayed.



AspectJ Developer Tools

- AJDT can present developers with broken pointcuts to an extent.
- However, AJDT is more general purpose:
  - All matching pointcuts are displayed in the UI, regardless if they are broken.
  - Broken pointcuts that do not match focused base code are not displayed.



Developer must *manually* find all pointcuts and decide if each should apply to the given join point

Our Hypothesis

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

[Khatchadourian, Greenwood, Rashid, Xu ’09]

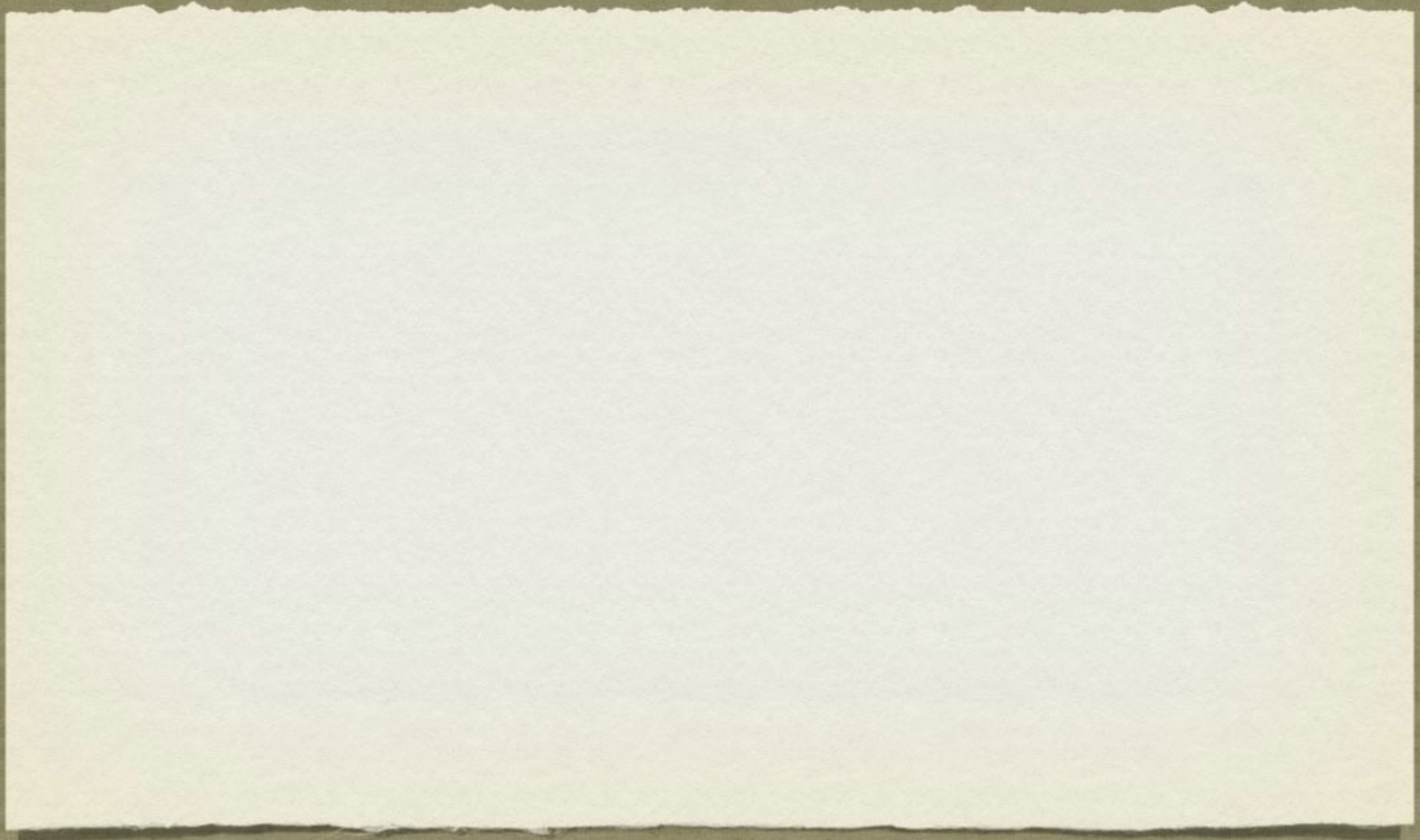
Our Hypothesis

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

[Khatchadourian, Greenwood, Rashid, Xu '09]

Can be used to *mechanically identify* broken pointcuts!

Aspect Example



Aspect Example

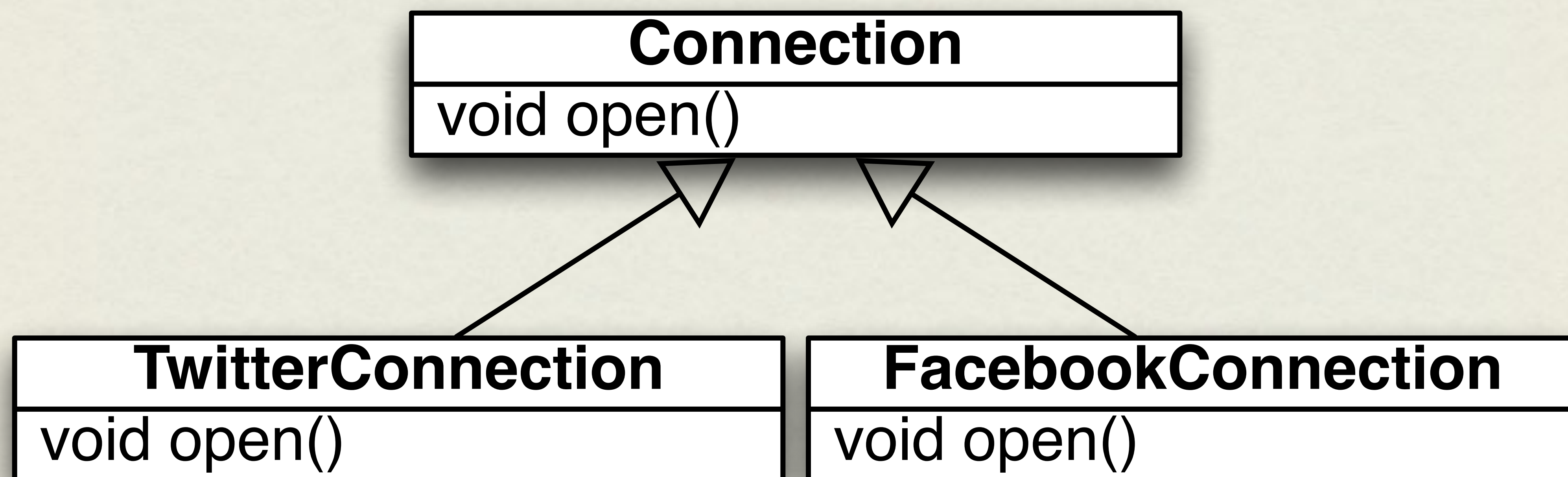
```
aspect Encryption {









}
```

Aspect Example

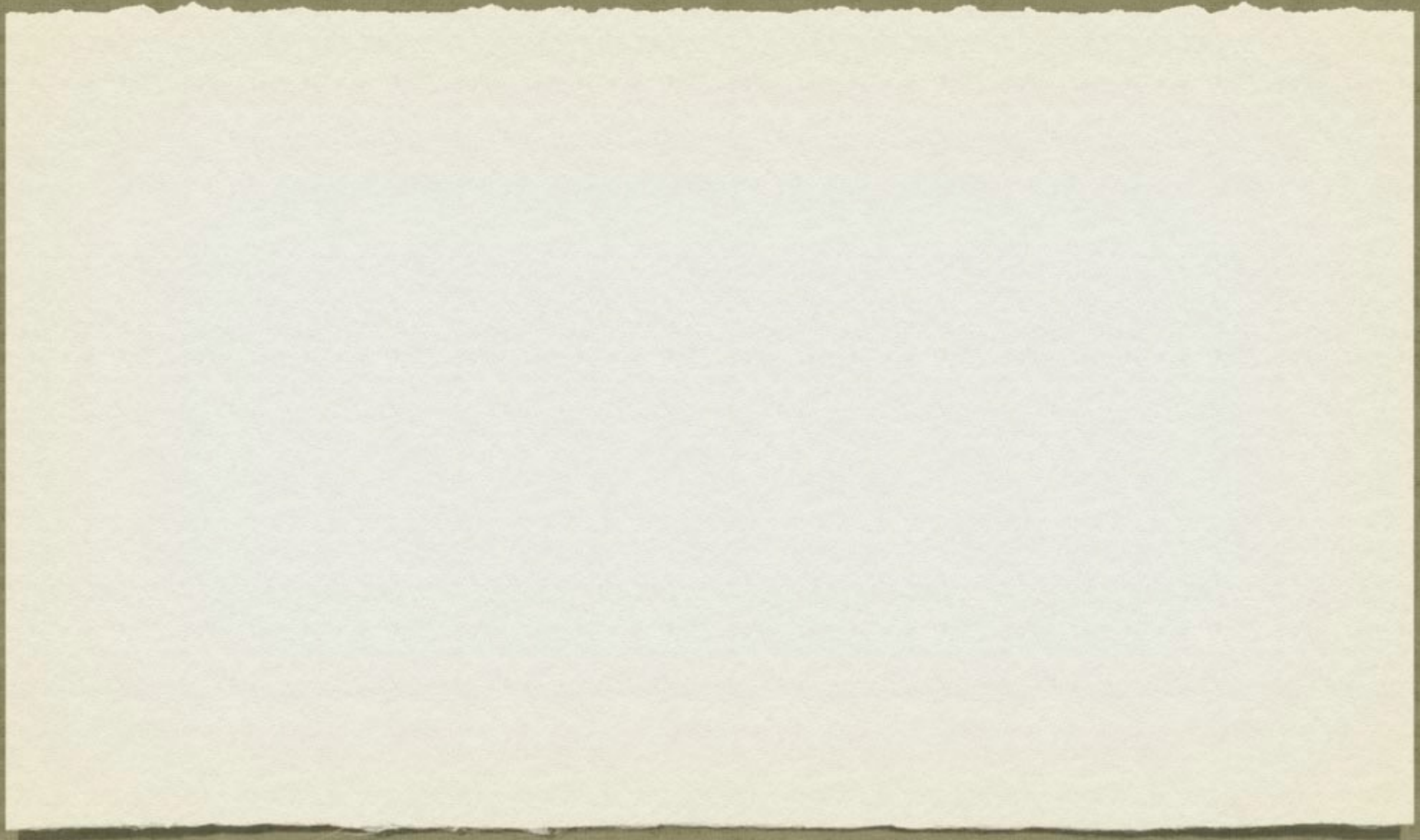
```
aspect Encryption {

    pointcut messageSent() : execution(* send*(String));









}
```

Aspect Example

```
aspect Encryption {

    pointcut messageSent() : execution(* send*(String));



    void around(String message) :
    messageSent() && args(message) {
         proceed(encrypt(message));}


}
```


Connection
void open()
TwitterConnection
void open()
FacebookConnection
void open()

Base Code Example



Base Code Example

```
class SocialMessanger {

}
```

Base Code Example

```
class SocialMessanger {


    Connection tconn = new TwitterConnection();
    Connection fconn = new FacebookConnection();





}
```

Base Code Example

```
class SocialMessanger {

    Connection tconn = new TwitterConnection();
    Connection fconn = new FacebookConnection();

    void sendTweet(String tweet) {tconn.open();…}

}
```

Base Code Example

```
class SocialMessanger {


    Connection tconn = new TwitterConnection();
    Connection fconn = new FacebookConnection();

    void sendTweet(String tweet) {tconn.open();…}
    void sendFacebookUpdate(String update) {fconn.open();…}



}
```

Base Code Example

```
class SocialMessanger {

    Connection tconn = new TwitterConnection();
    Connection fconn = new FacebookConnection();

    void sendTweet(String tweet) {tconn.open();…}
    void sendFacebookUpdate(String update) {fconn.open();…}
    void updateLocation() {tconn.open();fconn.open();…}

}
```

Structural Analysis

Phase I: Extract structural commonality between currently selected join points.

Base Code Example

```
class SocialMessanger {

    Connection tconn = new TwitterConnection();
    Connection fconn = new FacebookConnection();

    void sendTweet(String tweet) {tconn.open();…}
    void sendFacebookUpdate(String update) {fconn.open();…}
    void updateLocation() {tconn.open();fconn.open();…}

}
```

Base Code Example

```
class SocialMessanger {

    Connection tconn = new TwitterConnection();
    Connection fconn = new FacebookConnection();

    void sendTweet(String tweet) {tconn.open();…}
    void sendFacebookUpdate(String update) {fconn.open();…}
    void updateLocation() {tconn.open();fconn.open();…}

}
```

Base Code Example

Both methods call
Connection.open()

```
class SocialMessanger {

    Connection tconn = new TwitterConnection();
    Connection fconn = new FacebookConnection();

    void sendTweet(String tweet) {tconn.open();…}
    void sendFacebookUpdate(String update) {fconn.open();…}
    void updateLocation() {tconn.open();fconn.open();…}

}
```

- A ***structural pattern*** for `messageSent()` is "all executions of method whose bodies (textually) include a call to `Connection.open()`"

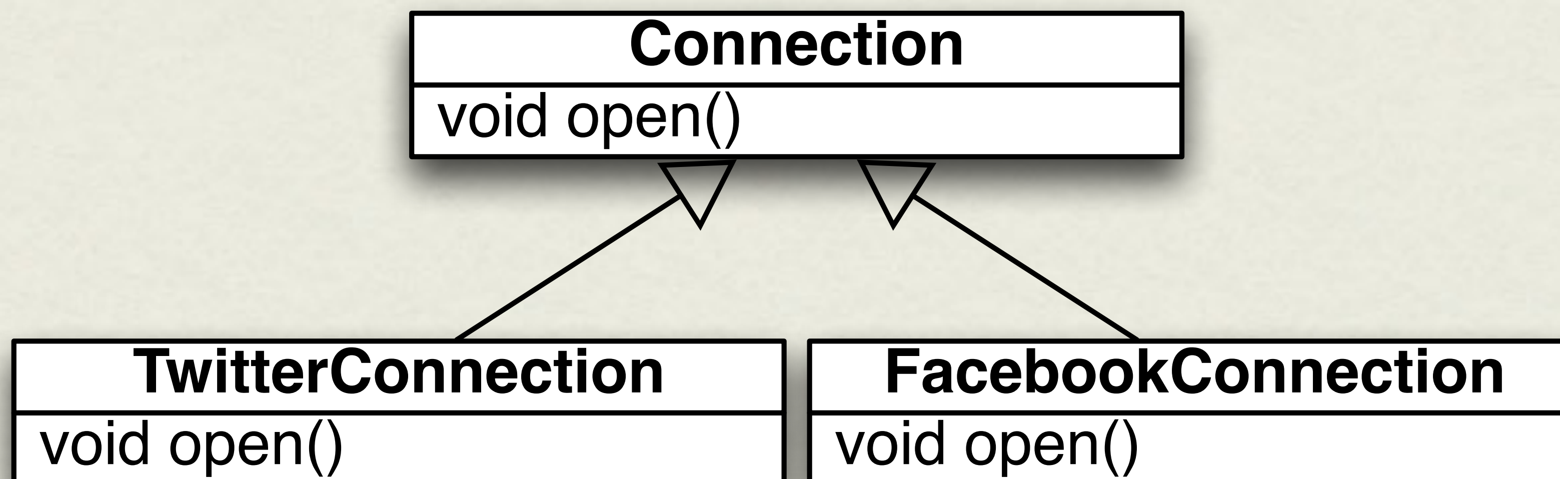
- `messageSent()` selects *two* of these methods.

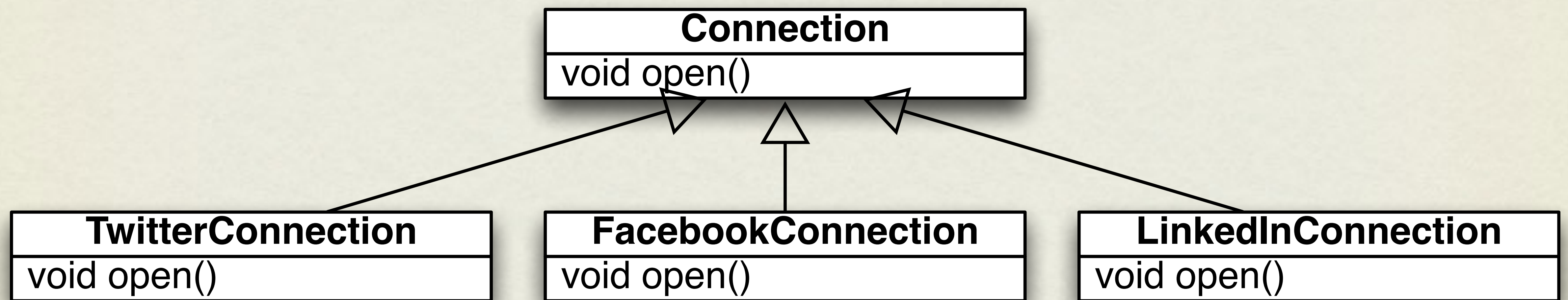
- The pattern describes *three* methods (includes `updateLocation()`).

- Thus, the ***confidence*** [Dagenais, Breu, Warr, Robillard '07] of this pattern is 2/3.

Pointcut Break Detection

Phase II: Apply extracted structural patterns to revised base code.


Connection
void open()
TwitterConnection
void open()
FacebookConnection
void open()


Connection
void open()
TwitterConnection
void open()
FacebookConnection
void open()
LinkedInConnection
void open()

Base Code Evolution

```
class SocialMessanger {

    //...



    //...




}
```

Base Code Evolution

```
class SocialMessanger {

    //...
    Connection lconn = new LinkedInConnection();



    //...




}
```

Base Code Evolution

```
class SocialMessanger {

    //...
    Connection lconn = new LinkedInConnection();



    //...

    void transmitLinkedInStatus(String update) {lconn.open();…}



}
```

Base Code Evolution

```
class SocialMessanger {

    //...
    Connection lconn = new LinkedInConnection();



    //...

    void transmitLinkedInStatus(String update) {lconn.open();…}


}
```

Should be encrypted but not selected by `messageSent()`!

Base Code Evolution

```
class SocialMessanger {

    //...
    Connection lconn = new LinkedInConnection(

    //...

    void transmitLinkedInStatus(String update) {lconn.open();…}

}
```

Also calls `Connection.open()`

Should be encrypted but not selected by `messageSent()`!

Base Code Evo

```
class SocialMessanger {

    //...
    Connection lconn = new LinkedInC

    //...
    void transmitLinkedInStatus(String update) {lconn.open();…}

}
```

2/3 *confident* that `messageSent()` has broken as a result since `transmitLinkedInStatus()` is not selected by `messageSent()` but shares ***structural commonality*** with elements selected by `messageSent()`

Also calls `Connection.open()`

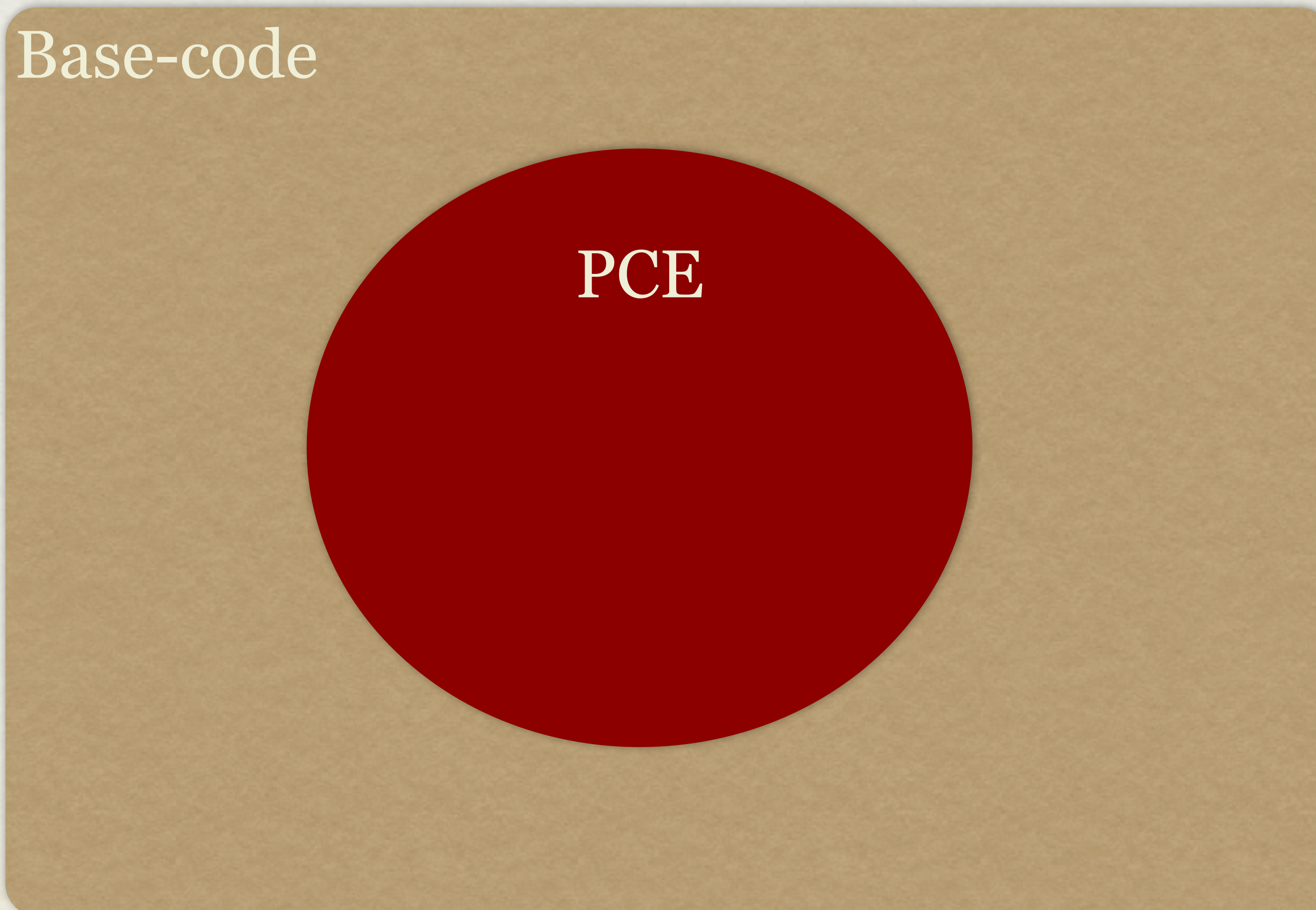
Should be encrypted but not selected by `messageSent()`!

If a *new join point* is added to the base code that *shares* a *high degree* of structural commonality with elements selected by a pointcut and is *not selected* by the pointcut, the pointcut becomes *more interesting* as it may need to be altered to *include* the new join point.

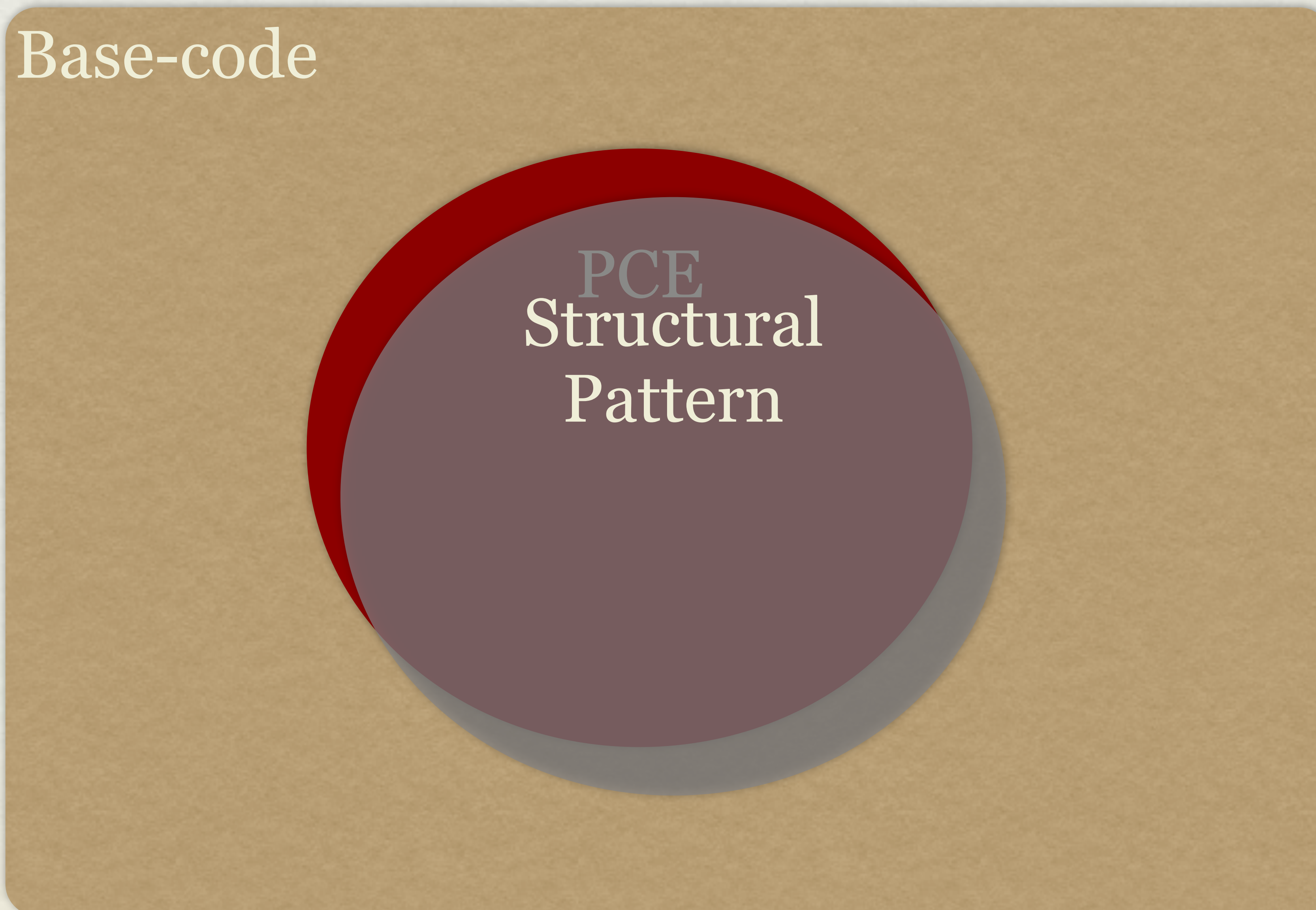
If a *new join point* is added to the base code that *shares* a *high degree* of structural commonality with elements selected by a pointcut and is *not selected* by the pointcut, the pointcut becomes *more interesting* as it may need to be altered to *include* the new join point.

Base-code

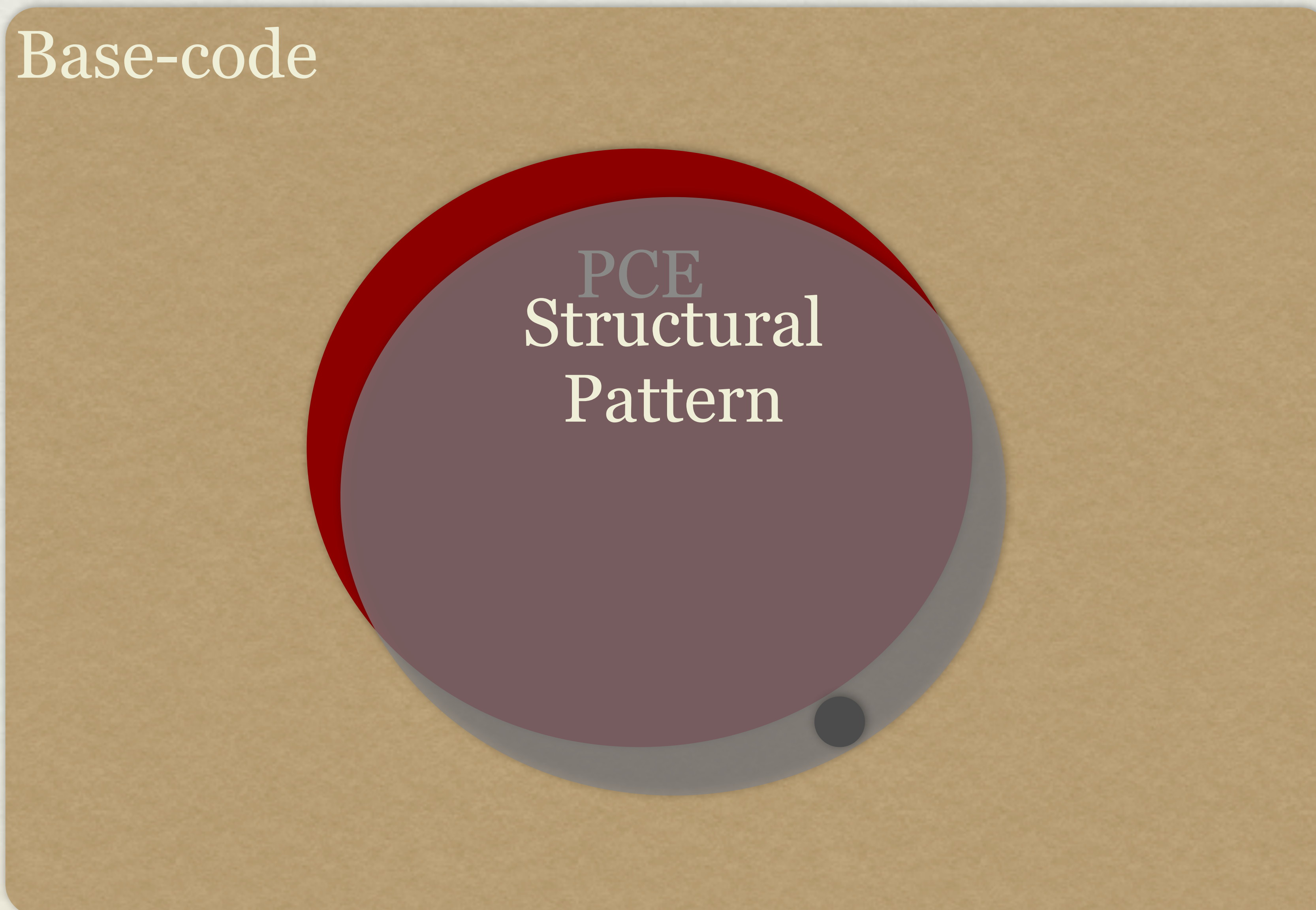
If a *new join point* is added to the base code that *shares* a *high degree* of structural commonality with elements selected by a pointcut and is *not selected* by the pointcut, the pointcut becomes *more interesting* as it may need to be altered to *include* the new join point.



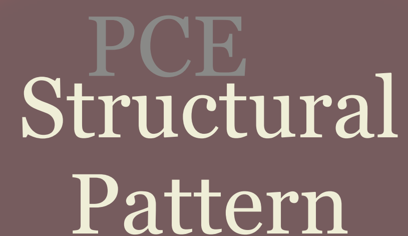
If a *new join point* is added to the base code that *shares* a *high degree* of structural commonality with elements selected by a pointcut and is *not selected* by the pointcut, the pointcut becomes *more interesting* as it may need to be altered to *include* the new join point.



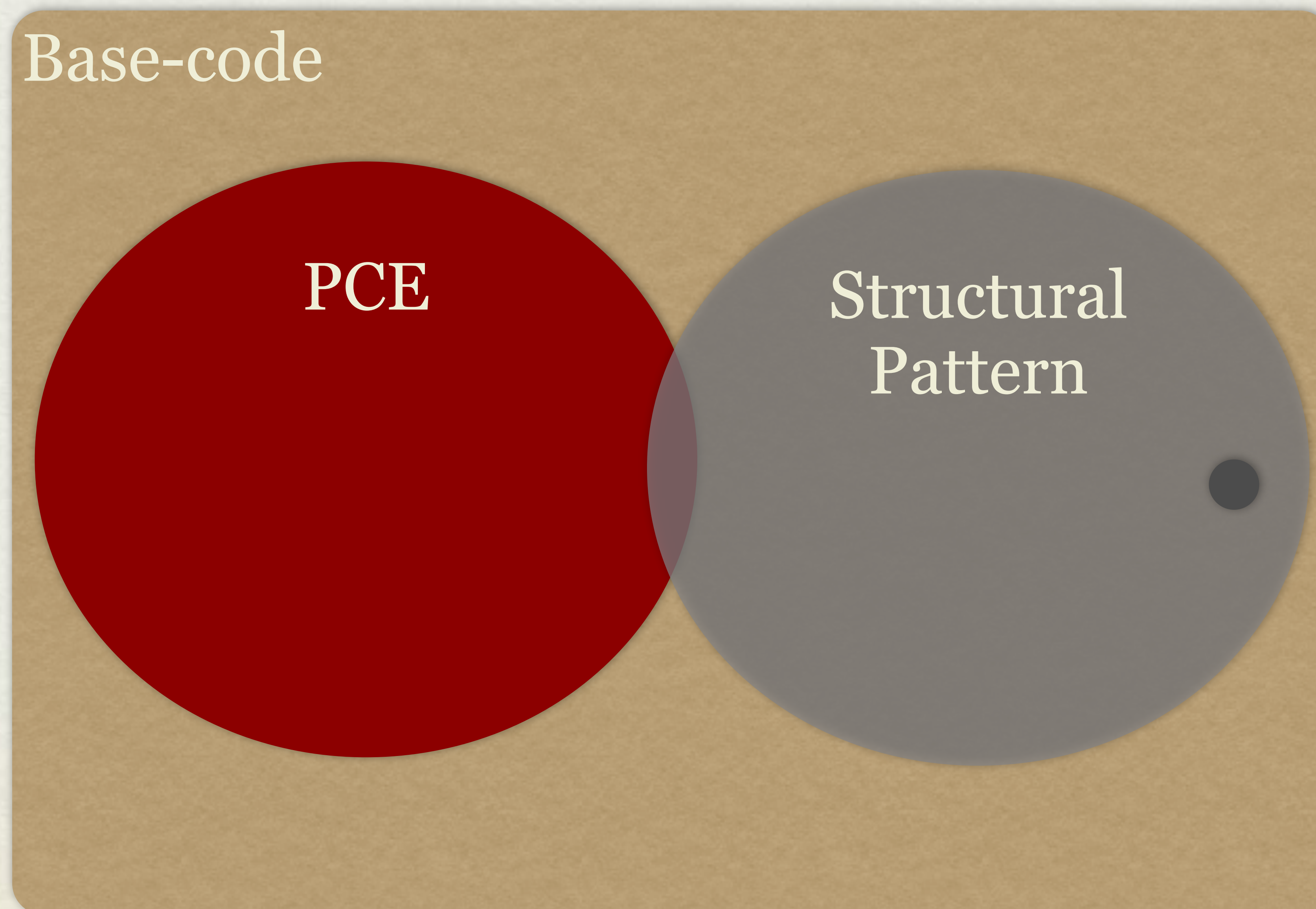
If a *new join point* is added to the base code that *shares* a *high degree* of structural commonality with elements selected by a pointcut and is *not selected* by the pointcut, the pointcut becomes *more interesting* as it may need to be altered to *include* the new join point.



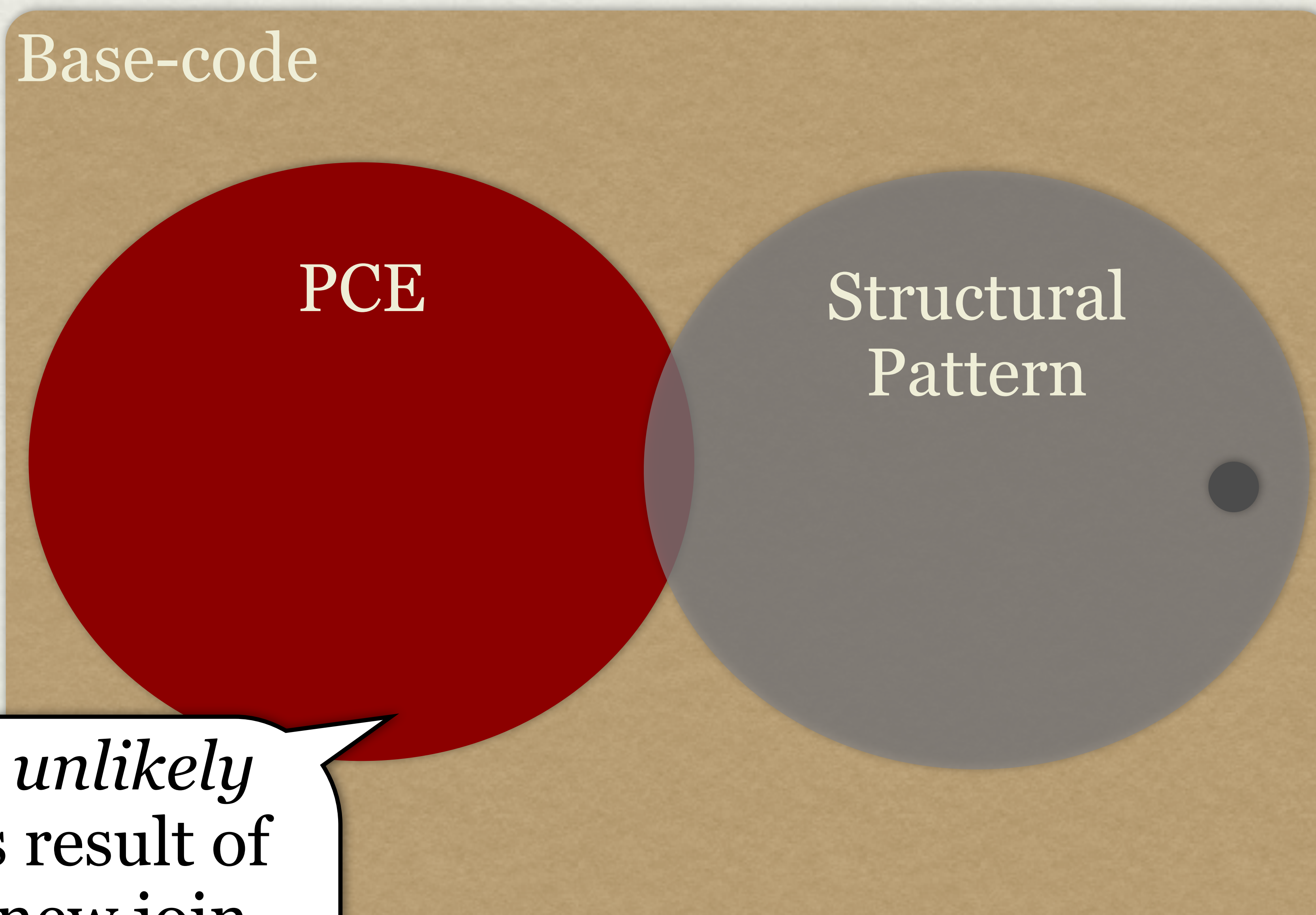
If a *new join point* is added to the base code that *shares* a *high degree* of structural commonality with elements selected by a pointcut and is *not selected* by the pointcut, the pointcut becomes *more interesting* as it may need to be altered to *include* the new join point.



If a *new join point* is added to the base code that does not *share a high degree* of structural commonality with elements selected by a pointcut and *is not selected* by the pointcut, the pointcut becomes *less interesting* to preserve obliviousness.



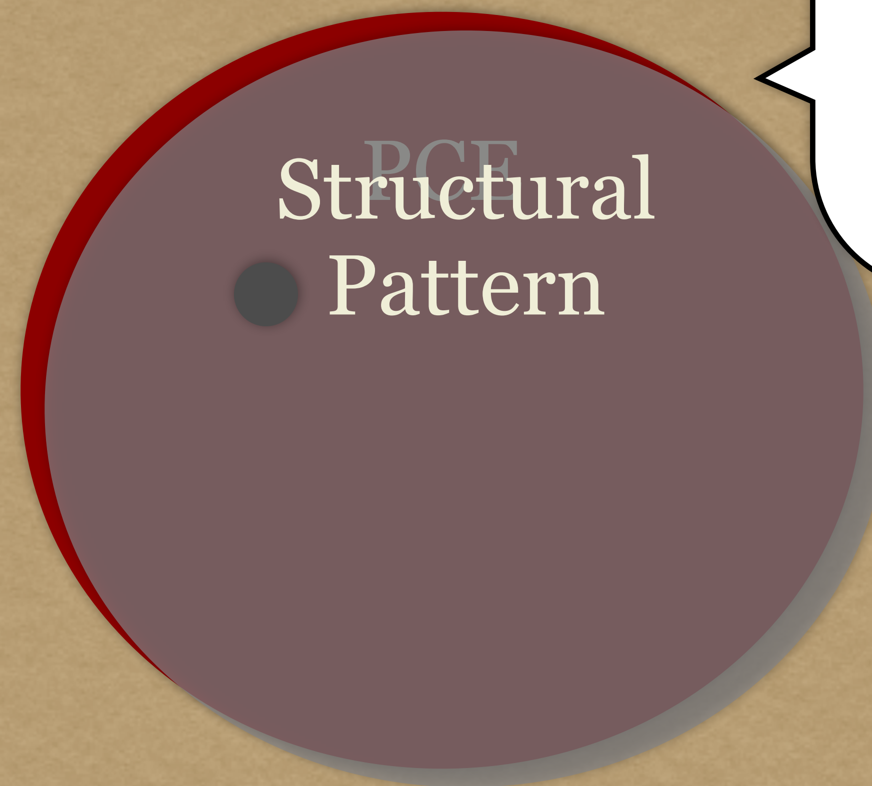
If a *new join point* is added to the base code that does not *share a high degree* of structural commonality with elements selected by a pointcut and *is not selected* by the pointcut, the pointcut becomes *less interesting* to preserve obliviousness.



If a *new join point* is added to the base code that *shares a high degree* of structural commonality with elements selected by a pointcut and *is selected* by the pointcut, the pointcut becomes *less interesting* to preserve obliviousness.

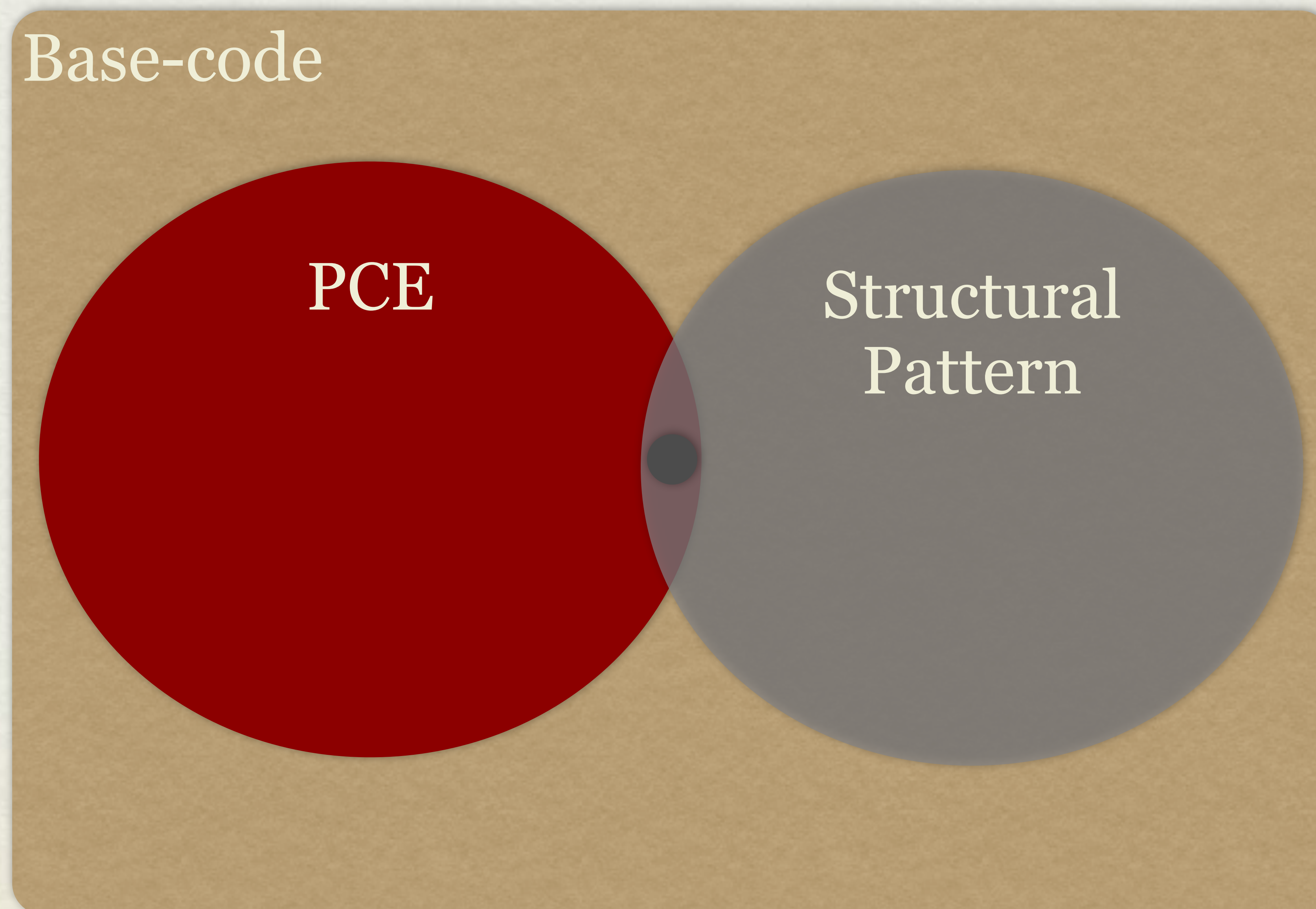
If a *new join point* is added to the base code that *shares a high degree* of structural commonality with elements selected by a pointcut and *is selected* by the pointcut, the pointcut becomes *less interesting* to preserve obliviousness.

Base-code

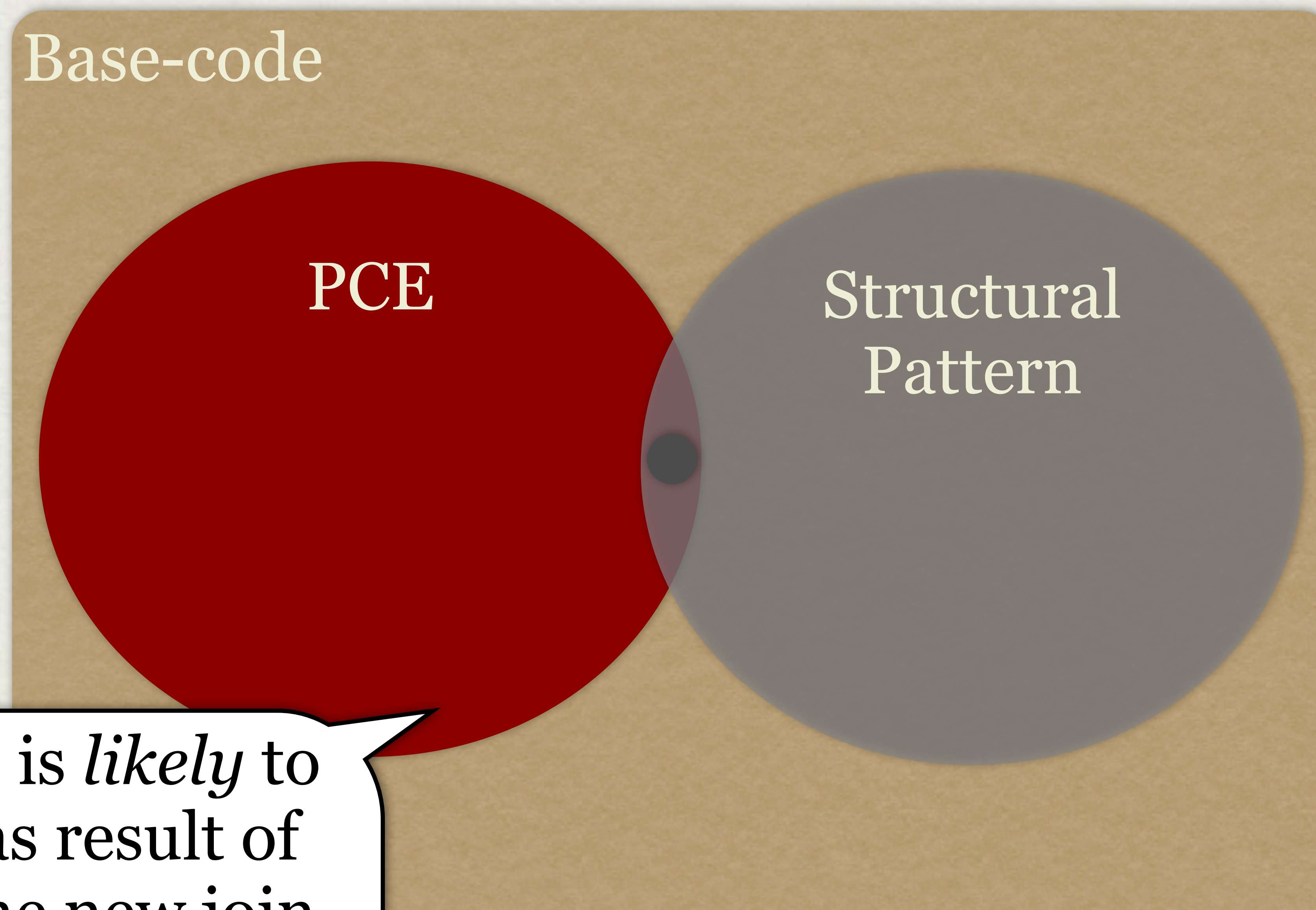


This PCE is *unlikely* to change as result of adding the new join point

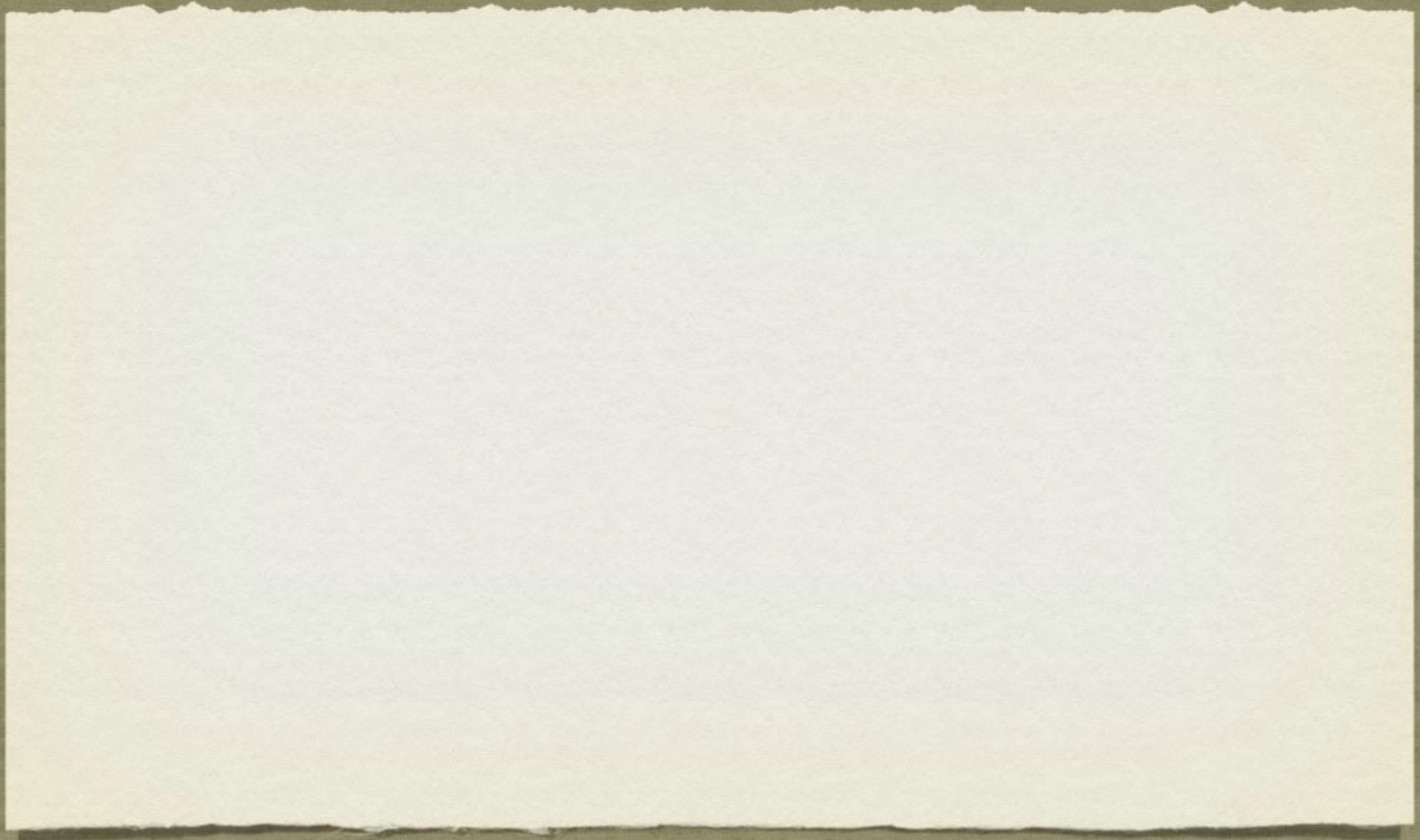
If a new *join point* is added to the base code that does *not* share a *high degree* of commonality with other elements selected by the pointcut and *is selected* by a pointcut, the pointcut becomes *more interesting* as it may need to be altered to *exclude* the new join point.



If a n*ew join point* is added to the base code that does *not* share a *high degree* of commonality with other elements selected by the pointcut and *is selected* by a pointcut, the pointcut becomes *more interesting* as it may need to be altered to *exclude* the new join point.



A Little About Pattern Construction



A Little About Pattern Construction

- Patterns are built from structural relations in source code

A Little About Pattern Construction

- Patterns are built from structural relations in source code
    - Method calls (CHA), field reads/writes, package containment etc.

A Little About Pattern Construction

- Patterns are built from structural relations in source code
  - Method calls (CHA), field reads/writes, package containment etc.
- Arbitrary length (parameter into analysis)

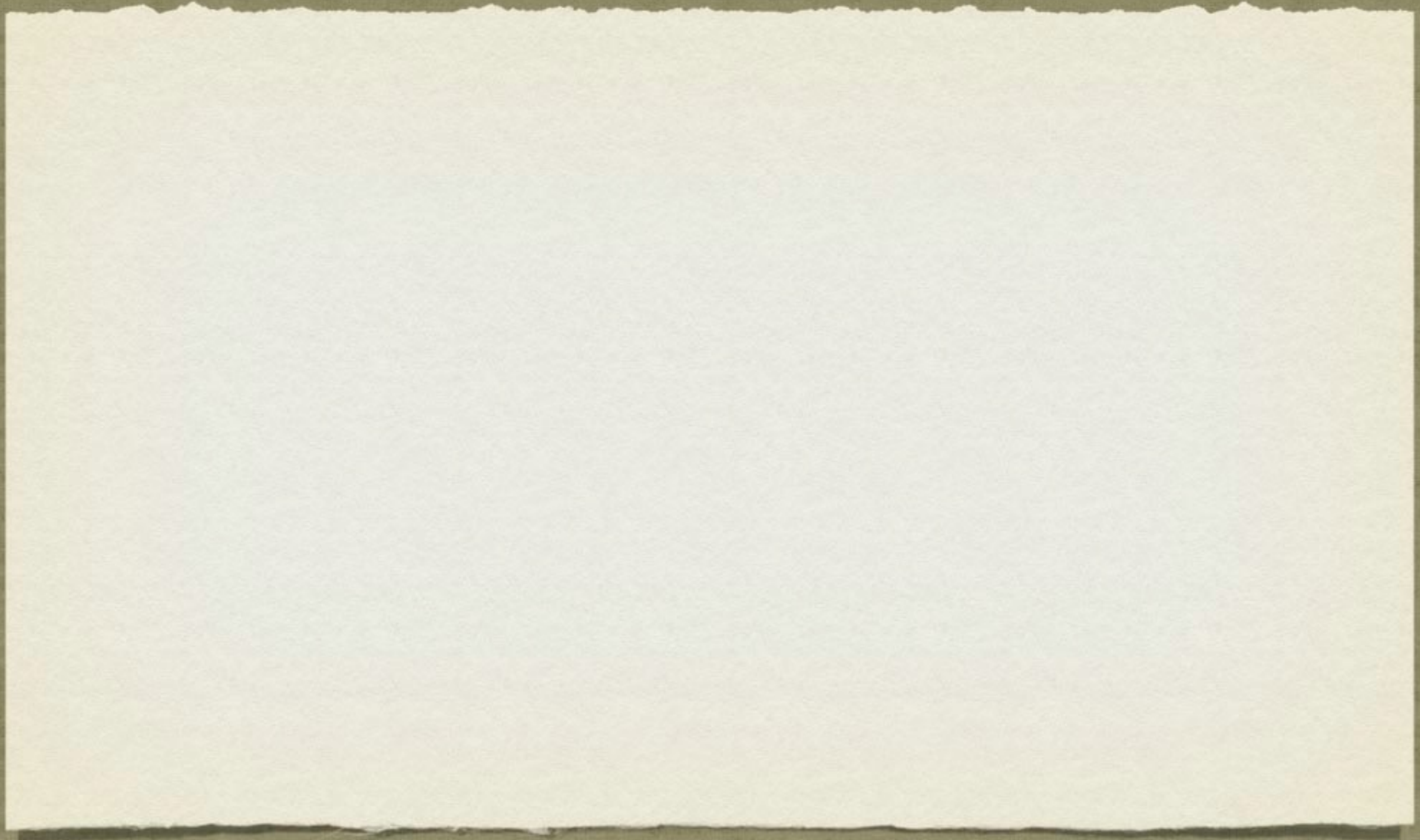
A Little About Pattern Construction

- Patterns are built from structural relations in source code
  - Method calls (CHA), field reads/writes, package containment etc.
- Arbitrary length (parameter into analysis)
  - Patterns derived from finite, acyclic paths in graph

A Little About Pattern Construction

- Patterns are built from structural relations in source code
  - Method calls (CHA), field reads/writes, package containment etc.
- Arbitrary length (parameter into analysis)
  - Patterns derived from finite, acyclic paths in graph
- Confidence is evaluated on three levels (see [Khatchadourian, Greenwood, Rashid, Xu ASE ’09] for details.

Ensuing Research Questions



Ensuing Research Questions

- Can Fraglight detect broken pointcuts accurately?

Ensuing Research Questions

- Can Fraglight detect broken pointcuts accurately?
- Can Fraglight prevent bugs?

Ensuing Research Questions

- Can Fraglight detect broken pointcuts accurately?
- Can Fraglight prevent bugs?
- Are there performance trade-offs?

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?