Aspect Mining
An Introduction

Andy Kellens

Thanks to Kim Mens & Kris Gybels
Overview

• What is aspect mining and why do we need it?

• Different kinds of approaches:
  • Advanced Browsers
  • Mining approaches

• Overview of automated approaches

• Research directions of aspect mining

• Conclusion: limitations of Aspect Mining
Aspects in existing systems?

• Aspect-Oriented Programming
  • Better modularization
  • Improved -ilities
• Also useful for already existing systems
  • How migrate a system to AOSD?
Migrating a System

1. Identify crosscutting concerns
2. Refactor into AO program
3. Evolution of AO program

A legacy program (with crosscutting concerns) → An aspect-oriented program
Migrating a System

1. Identify crosscutting concerns

A legacy program (with crosscutting concerns)

2. Refactor into AO program

An aspect-oriented program

3. Evolution of AO program

Aspect Mining
Migrating a System

1. Identify crosscutting concerns

A legacy program (with crosscutting concerns)

2. Refactor into AO program

2. Refactor into AO program

3. Evolution of AO program

An aspect-oriented program

Aspect Mining

Aspect Refactoring
Why do we need aspect mining?

- Legacy systems
  - large, complex systems
  - not always clearly documented
  - need to find the crosscutting concerns
  - need to find the extent of the crosscutting concerns
- program understanding/documentation
Why do we need aspect mining?

- Legacy systems
  - large, complex systems
  - not always clearly documented
  - need to find the crosscutting concerns
  - need to find the extent of the crosscutting concerns
  - program understanding/documentation
Kinds of techniques

• “Early aspect” discovery techniques
• Advanced special-purpose browsers
• (Semi-)automated code analysis techniques
Kinds of techniques

- “Early aspect” discovery techniques
- Advanced special-purpose browsers
- (Semi-)automated code analysis techniques
Early Aspects

• Try to find the aspects in artifacts of the early phases of development:
  • requirements
  • design
• Not really useful in the context of finding aspects in legacy code
Code Browsers

- Idea: help a developer in exploring a concern
- User provides an “interesting” point in the code (seed); tool suggests other related points
- Iteratively, build up a model of the crosscutting concerns
- Examples: FEAT, JQuery, Soul, Prism
FEAT

• Create a “Concern Graph”, a representation of the concern mapping back to the source code.

• Consists out of a set of program entities

• Start out with a number of program entities (e.g. “all callers of the method print()”)

• Iteratively explore the relations (fan-in, callers, ...) and refine the concern graph
FEAT

FEAT 2.2.1 - Eclipse Platform

**Managing Figure Attributes**
- Setting Attributes
- Change Command

**Participants - Setting Attributes**
- AttributeFigure
  - fAttributes
  - getAttribute(String)
  - read(StorableInput)
  - setAttribute(String, Object)
  - write(StorableOutput)
- FigureAttributes

**Projection**
- AttributeFigure
  - setAttribute(String, Object)
    - accessing
  - AttributeFigure
  - fAttributes

**AttributeFigure.setAttribute(String, Object) accessing AttributeFigure.fAttributes - 1/3**

```java
/**
 * Sets the named attribute to the new value
 */
public void setAttribute(String name, Object value) {
    if (fAttributes == null)
        fAttributes = new FigureAttributes();
    fAttributes.set(name, value);
    changed();
}

/**
 * Stores the Figure to a StorableOutput.
 */
public void write(StorableOutput dw) {
    super.write(dw);
}
```
Browsers

• Require a lot of manual effort (do not scale well)
  • => Try to automate this process
• Require bit of knowledge about system (need for seeds)
Automated Approaches

• Semi-automatically find possible aspect candidates in source code of a system

• No magic:
  • Manual filtering of the results
  • False positives/negatives
  • May require preliminary knowledge

• Possibility to use results as seed for browsers
Automated Approaches

Hidden assumption behind techniques:

- symptoms of scattering and tangling
- either static (code) or dynamic (run-time)

Discover redundancy using a variety of data analysis techniques:

- inspired by data mining, code analysis, ...
- but specifically (re)designed for aspect mining
Automated Approaches

- Pattern matching
  - Analysing recurring patterns of execution traces
  - Clone detection
    - PDF-based / token-based / AST-based
- Formal concept analysis
  - Execution traces / identifiers
- Natural language processing on code
- AOP idioms
  - Fan-in analysis / Detecting unique methods
- Cluster analysis
  - Method names / method invocations
Recurring Patterns (Breu and Krinke)

• Assumption: if certain method calls always happen before/after the same call, they might be part of an aspect

• “Reverse engineering” of manual implementation of before/after advice

• Technique: find pairs of methods which happen right before/after each other and check whether this assumption always holds
Recurring Patterns (Breu and Krinke)

- Inherently dynamic (analysis of execution trace)
- CCC needs to be rigorously implemented (calls always in same order)
Clone Detection (Bruntink, Shepherd)

- Assumption: crosscutting concerns result in code duplication
- Certain CCCs are implemented using a recurring pattern, idiom
- Technique: find classes of related code
  - exact same
  - same structure; different literals
**Clone Detection (Bruntink, Shepherd)**

```c
MC if (r != OK)
MC {
  MC ERXA_LOG(r, 0, ("PLXAmem_malloc failure.");
  MC
  MC ERXA_LOG(VSXA_MEMORY_ERR, r,
    MC    ("%s: failed to allocated %d bytes.",
            MC    func_name, toread));
  M
  M  r = VSXA_MEMORY_ERR;
  M }
```

- source code analysis
- applicable if the CCCs are implemented as a pattern
Formal Concept Analysis

- Starts from
  - a set of **elements**
  - a set of **properties** of those elements

- Determines **concepts**
  - Maximal groups of elements and properties
  - Group:
    - Every element of the concept has those properties
    - Every property of the concept holds for those elements
  - Maximal
    - No other element (outside the concept) has those same properties
    - No other property (outside the concept) is shared by all elements
# Formal Concept Analysis

<table>
<thead>
<tr>
<th></th>
<th>object-oriented</th>
<th>functional</th>
<th>logic</th>
<th>static typing</th>
<th>dynamic typing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>C++</strong></td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td><strong>Java</strong></td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td><strong>Smalltalk</strong></td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td><strong>Scheme</strong></td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td><strong>Prolog</strong></td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
</tbody>
</table>
### Formal Concept Analysis

<table>
<thead>
<tr>
<th></th>
<th>object-oriented</th>
<th>functional</th>
<th>logic</th>
<th>static typing</th>
<th>dynamic typing</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Java</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Smalltalk</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Scheme</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Prolog</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
</tbody>
</table>
## Formal Concept Analysis

<table>
<thead>
<tr>
<th></th>
<th>object-oriented</th>
<th>functional</th>
<th>logic</th>
<th>static typing</th>
<th>dynamic typing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>C++</strong></td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td><strong>Java</strong></td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td><strong>Smalltalk</strong></td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td><strong>Scheme</strong></td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td><strong>Prolog</strong></td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
</tbody>
</table>
## Formal Concept Analysis

<table>
<thead>
<tr>
<th></th>
<th>object-oriented</th>
<th>functional</th>
<th>logic</th>
<th>static typing</th>
<th>dynamic typing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>C++</strong></td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td><strong>Java</strong></td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td><strong>Smalltalk</strong></td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td><strong>Scheme</strong></td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td><strong>Prolog</strong></td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
</tbody>
</table>
Formal Concept Analysis

Diagram:

- Java Smalltalk C++
- Scheme Prolog
- {}
- Java C++ Smalltalk
  - {object oriented}
- Scheme Prolog Smalltalk
  - {dynamic typing}
- Java C++
  - {static typing, object oriented}
- Smalltalk
  - {dynamic typing, object oriented}
- Scheme
  - {dynamic typing, functional}
- Prolog
  - {dynamic typing, logic}
- {}
  - {object oriented, functional, logic, static typing, dynamic typing}
Identifier Analysis (Mens & Tourwé)

• Assumption: methods belonging to the same concern are implemented using similar names

• Eg. methods implementing Undo concern will have “undo” in their signature

• Technique: Use Formal Concept Analysis with the methods as objects and the different substrings as properties
## Identifier Analysis

<table>
<thead>
<tr>
<th>Method</th>
<th>figure</th>
<th>drawing</th>
<th>request</th>
<th>remove</th>
<th>update</th>
<th>change</th>
<th>event</th>
</tr>
</thead>
<tbody>
<tr>
<td>drawingRequestUpdate (DrawingChangeEvent e)</td>
<td></td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>figureRequestRemove (FigureChangeEvent e)</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>figureRequestUpdate (FigureChangeEvent e)</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>figureRequestRemove (FigureChangeEvent e)</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>figureRequestUpdate (FigureChangeEvent e)</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

- Lots of concepts: filter irrelevant ones
- Pre-classify “similar” concepts
Execution traces (ceccato & Tonella)

- Assumption: use case scenario’s are a good indicator for crosscutting concerns

- Technique: Analyse execution trace using FCA with use cases as objects; methods called from within use case as attributes

- Consider concept local to one case study. Possible aspect if:
  - Methods belong to more than one class (scattering)
  - Methods of same class occur in multiple use cases
Fan-in Analysis (Marin)

• Assumption: The fan-in metric is a good indicator of scattering

• “Methods which get called often from different contexts are possible crosscutting concerns”

• Technique: calculate the fan-in of each method, filter auxiliary methods/accessors and sort the resulting methods
Fan-in Analysis (Marin)

- Fan-in metric [Henderson-Sellers]
  - counts the number of locations from which
  - control is passed into a module

- Fan-in metric for OOP
  - applied to method M
  - number of distinct method bodies that can invoke M

- What about polymorphism?
  - one call-site can affect the fan-in of several methods
  - a call to M contributes to the fan-in of M but also of all its
    overriding methods as well as all methods it overrides
### Fan-in Analysis (Marin)

```java
interface A{
    public void m();
}
class B implements A {
    public void m() {
    }
}
class C1 extends B {
    public void m() {
    }
}
class C2 extends B {
    public void m() {
        super.m();
    }
}
class D {
    void f1(A a) { a.m(); }
    void f2(B b) { b.m(); }
    void f3(C1 c) { c.m(); }
}
```

<table>
<thead>
<tr>
<th>Method</th>
<th>Caller set</th>
<th>Fan-in</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.m</td>
<td>{D.f1, D.f2, D.f3}</td>
<td>3</td>
</tr>
<tr>
<td>B.m</td>
<td>{D.f1, D.f2, D.f3, C2.m}</td>
<td>4</td>
</tr>
<tr>
<td>C1.m</td>
<td>{D.f1, D.f2, D.f3}</td>
<td>3</td>
</tr>
<tr>
<td>C2.m</td>
<td>{D.f1, D.f2}</td>
<td>2</td>
</tr>
</tbody>
</table>
Unique Methods (Gybels & Kellens)

• Assumption: Some CCCs are implemented by calling a single entity from all over the code

• E.g. Logging

• Technique: devise a metric (unique method) which finds such crosscutting concerns
Unique Method
A method without a return value which implements a message implemented by no other method.

Q: Why no return type?
A: Use of value in base computation
# Unique Methods (Gybels & Kellens)

<table>
<thead>
<tr>
<th>Class</th>
<th>Selector</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parcel</td>
<td><code>#markAsDirty</code></td>
<td>23</td>
</tr>
<tr>
<td>ParagraphEditor</td>
<td><code>#resetTypeIn</code></td>
<td>19</td>
</tr>
<tr>
<td>UIPainter</td>
<td><code>#broadcastPendingSelectionChange</code></td>
<td>18</td>
</tr>
<tr>
<td>ControllerCodeRegenerator</td>
<td><code>#pushPC</code></td>
<td>15</td>
</tr>
<tr>
<td>AbstractChangeList</td>
<td><code>#updateSelection:</code></td>
<td>15</td>
</tr>
<tr>
<td>PundleModel</td>
<td><code>#updateAfterDo:</code></td>
<td>10</td>
</tr>
<tr>
<td>????????????????????????????????</td>
<td><code>#beGarbageCollectable</code></td>
<td>??</td>
</tr>
</tbody>
</table>
Cluster Analysis

• Input:
  • a set of objects
  • a distance measure between those objects

• Output:
  • groups of objects which are close (according to the distance function)
Cluster Analysis

Zoals reeds eerder vermeld is het doel van dit onderzoeksvoorstel om... bekommer-
nissen
4.2.1 Gebruik makend van cluster analyse
Figuur 10: Illustratie van hiërarchische clustering
18
Aspect Mining Using Cluster Analysis (He, Shepherd)

• Distance function between two methods

• Call Clustering:
  • distance in function of times they occur together in same method (cfr. recurring execution traces)

• Method Clustering:
  • distance in function of commonalities in name of methods (cfr. identifier analysis)
## Comparison (1)

<table>
<thead>
<tr>
<th></th>
<th>Kind of input data</th>
<th>Semantic level</th>
<th>Granularity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>static</td>
<td>dynamic</td>
<td>lexical</td>
</tr>
<tr>
<td>Execution patterns</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Dynamic analysis</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Identifier analysis</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Language clues</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Unique methods</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Method clustering</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Call clustering</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Fan-in analysis</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Clone detection (PDG)</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Clone detection (Token)</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Clone detection (AST)</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
## Comparison (1)

<table>
<thead>
<tr>
<th></th>
<th>Kind of input data</th>
<th>Semantic level</th>
<th>Granularity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>static</td>
<td>lexical</td>
<td>method</td>
</tr>
<tr>
<td></td>
<td>dynamic</td>
<td>semantic</td>
<td>code fragment</td>
</tr>
<tr>
<td>Execution patterns</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Dynamic analysis</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Identifier analysis</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Language clues</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Unique methods</td>
<td></td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Method clustering</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Call clustering</td>
<td></td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Fan-in analysis</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Clone detection (PDG)</td>
<td></td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Clone detection (Token)</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Clone detection (AST)</td>
<td></td>
<td>-</td>
<td>X</td>
</tr>
</tbody>
</table>

### Few dynamic

- Execution patterns
- Dynamic analysis
- Identifier analysis
- Language clues
- Unique methods
- Method clustering
- Call clustering
- Fan-in analysis
- Clone detection (PDG)
- Clone detection (Token)
- Clone detection (AST)
## Comparison (1)

<table>
<thead>
<tr>
<th></th>
<th>Kind of input data</th>
<th>Semantic level</th>
<th>Granularity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>static</td>
<td>dynamic</td>
<td>lexical</td>
</tr>
<tr>
<td>Execution patterns</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic analysis</td>
<td></td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Identifier analysis</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Language clues</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Unique methods</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Method clustering</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Call clustering</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fan-in analysis</td>
<td>X</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Clone detection (PDG)</td>
<td>X</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Clone detection (Token)</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Clone detection (AST)</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

*Few dynamic*

*Few below method-level*
## Comparison (2)

<table>
<thead>
<tr>
<th></th>
<th>Symptoms scattering</th>
<th>Symptoms tangling</th>
<th>Incremental</th>
<th>Iterative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution patterns</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Dynamic analysis</td>
<td>X</td>
<td>X</td>
<td>(X)</td>
<td>X</td>
</tr>
<tr>
<td>Identifier analysis</td>
<td>X</td>
<td>-</td>
<td>(X)</td>
<td>X</td>
</tr>
<tr>
<td>Language clues</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Unique methods</td>
<td>X</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Method clustering</td>
<td>X</td>
<td>-</td>
<td>(X)</td>
<td>X</td>
</tr>
<tr>
<td>Call clustering</td>
<td>X</td>
<td>-</td>
<td>(X)</td>
<td>X</td>
</tr>
<tr>
<td>Fan-in analysis</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Clone detection</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
</tbody>
</table>
## Comparison (2)

<table>
<thead>
<tr>
<th>Symptoms</th>
<th>Incremental</th>
<th>Iterative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scattering</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Tangling</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execution patterns</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Dynamic analysis</td>
<td>X</td>
<td>(X)</td>
</tr>
<tr>
<td>Identifier analysis</td>
<td>X</td>
<td>(X)</td>
</tr>
<tr>
<td>Language clues</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Unique methods</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Method clustering</td>
<td></td>
<td>(X)</td>
</tr>
<tr>
<td>Call clustering</td>
<td></td>
<td>(X)</td>
</tr>
<tr>
<td>Fan-in analysis</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Clone detection</td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>
### Comparison (2)

<table>
<thead>
<tr>
<th>Method Clustering</th>
<th>Symptoms</th>
<th>Incremental</th>
<th>Iterative</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>scattering</td>
<td>tangling</td>
<td></td>
</tr>
<tr>
<td>Execution patterns</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Dynamic analysis</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Identifier analysis</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Language clues</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Unique methods</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Method clustering</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Call clustering</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Fan-in analysis</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Clone detection</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

**Note:**
- Few tangling can be incremental.
- Many can be incremental.
## Comparison (2)

<table>
<thead>
<tr>
<th>Symptom</th>
<th>Scattering</th>
<th>Tangling</th>
<th>Incremental</th>
<th>Iterative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution patterns</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Dynamic analysis</td>
<td>X</td>
<td>X</td>
<td>(X)</td>
<td>X</td>
</tr>
<tr>
<td>Identifier analysis</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Language clues</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Unique methods</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Method clustering</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Call clustering</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fan-in analysis</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clone detection</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Few tangling*  
*Many can be incremental*  
*Most are iterative*
## Comparison (3)

<table>
<thead>
<tr>
<th></th>
<th>largest case</th>
<th>size case</th>
<th>empirically validated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution patterns</td>
<td>Graffiti</td>
<td>3100 methods/82KLOC</td>
<td>-</td>
</tr>
<tr>
<td>Dynamic analysis</td>
<td>JHotDraw</td>
<td>2800 methods/18KLOC</td>
<td>-</td>
</tr>
<tr>
<td>Identifier analysis</td>
<td>JHotDraw</td>
<td>2800 methods/18KLOC</td>
<td>-</td>
</tr>
<tr>
<td>Language clues</td>
<td>PetStore</td>
<td>10KLOC</td>
<td>-</td>
</tr>
<tr>
<td>Unique methods</td>
<td>Smalltalk image</td>
<td>3400 classes/66000 methods</td>
<td>-</td>
</tr>
<tr>
<td>Method clustering</td>
<td>JHotDraw</td>
<td>2800 methods/18KLOC</td>
<td>-</td>
</tr>
<tr>
<td>Call clustering</td>
<td>Banking example</td>
<td>12 methods</td>
<td>-</td>
</tr>
<tr>
<td>Fan-in analysis</td>
<td>JHotDraw</td>
<td>2800 methods/18KLOC</td>
<td>-</td>
</tr>
<tr>
<td>Clone detection (PDG)</td>
<td>TomCat 5.5 API</td>
<td>172KLOC</td>
<td>-</td>
</tr>
<tr>
<td>Clone detection (AST/Token)</td>
<td>ASML C-Code</td>
<td>20KLOC</td>
<td>X</td>
</tr>
</tbody>
</table>
## Comparison (3)

<table>
<thead>
<tr>
<th>Feature</th>
<th>largest case</th>
<th>size case</th>
<th>empirically validated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution patterns</td>
<td>Graffiti</td>
<td>3100 methods/82KLOC</td>
<td>-</td>
</tr>
<tr>
<td>Dynamic analysis</td>
<td>JHotDraw</td>
<td>2100 methods/18KLOC</td>
<td>-</td>
</tr>
<tr>
<td>Identifier analysis</td>
<td>JHotDraw</td>
<td>2000 methods/18KLOC</td>
<td>-</td>
</tr>
<tr>
<td>Language clues</td>
<td>PetStore</td>
<td>11KLOC</td>
<td>-</td>
</tr>
<tr>
<td>Unique methods</td>
<td>Smalltalk</td>
<td>66000 methods</td>
<td>-</td>
</tr>
<tr>
<td>Method clustering</td>
<td>JHotDraw</td>
<td>1000 methods/18KLOC</td>
<td>-</td>
</tr>
<tr>
<td>Call clustering</td>
<td>Banking examples</td>
<td>2 methods</td>
<td>-</td>
</tr>
<tr>
<td>Fan-in analysis</td>
<td>JHotDraw</td>
<td>850 methods/18KLOC</td>
<td>-</td>
</tr>
<tr>
<td>Clone detection (PDG)</td>
<td>TomCat 5.5 API</td>
<td>172KLOC</td>
<td>-</td>
</tr>
<tr>
<td>Clone detection (AST/Token)</td>
<td>TomCat</td>
<td>38KLOC</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>ASML C-Code</td>
<td>20KLOC</td>
<td>X</td>
</tr>
</tbody>
</table>

*no toy examples*
## Comparison (3)

<table>
<thead>
<tr>
<th>Execution patterns</th>
<th>largest case</th>
<th>size case</th>
<th>empirically validated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic analysis</td>
<td>Graffiti (3100 methods/82KLOC)</td>
<td>JHotDraw (30 methods/18KLOC)</td>
<td>-</td>
</tr>
<tr>
<td>Identifier analysis</td>
<td>JHotDraw (2 methods/18KLOC)</td>
<td>PetStore (56 methods/18KLOC)</td>
<td>-</td>
</tr>
<tr>
<td>Language clues</td>
<td>Smalltalk (16KLOC)</td>
<td>Banking (60 methods/18KLOC)</td>
<td>-</td>
</tr>
<tr>
<td>Unique methods</td>
<td>JHotDraw (172KLOC)</td>
<td>JHotDraw (38KLOC)</td>
<td>-</td>
</tr>
<tr>
<td>Method clustering</td>
<td>TomCat 5.5 (20KLOC)</td>
<td>TomCat (20KLOC)</td>
<td>X</td>
</tr>
<tr>
<td>Call clustering</td>
<td>ASML C-Code (empirical)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fan-in analysis</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clone detection (PDG)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clone detection (AST/Token)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Comparison (4)

<table>
<thead>
<tr>
<th>Technique</th>
<th>User Involvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution patterns</td>
<td>Inspection of the resulting “recurring patterns”.</td>
</tr>
<tr>
<td>Dynamic analysis</td>
<td>Selection of use cases and manual interpretation of results.</td>
</tr>
<tr>
<td>Identifier analysis</td>
<td>Browsing of mined aspects using IDE integration.</td>
</tr>
<tr>
<td>Language clues</td>
<td>Manual interpretation of resulting lexical chains.</td>
</tr>
<tr>
<td>Unique methods</td>
<td>Inspection of the unique methods; eased by sorting on importance.</td>
</tr>
<tr>
<td>Method clustering</td>
<td>Browsing of mined aspects using IDE integration.</td>
</tr>
<tr>
<td>Call clustering</td>
<td>Manual inspection of resulting clusters.</td>
</tr>
<tr>
<td>Fan-in analysis</td>
<td>Selection of candidates from list of methods, sorted on highest fan-in.</td>
</tr>
<tr>
<td>Clone detection</td>
<td>Browsing and manual interpretation of the discovered clones.</td>
</tr>
</tbody>
</table>
## Comparison (4)

<table>
<thead>
<tr>
<th>Technique</th>
<th>User Involvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution patterns</td>
<td>Inspection of the results, followed by interpretation of results</td>
</tr>
<tr>
<td>Dynamic analysis</td>
<td>Selection of user-defined rules and interpretation of results</td>
</tr>
<tr>
<td>Identifier analysis</td>
<td>Browsing of results obtained by analysis</td>
</tr>
<tr>
<td>Language clues</td>
<td>Manual inspection of suspicious patterns</td>
</tr>
<tr>
<td>Unique methods</td>
<td>Inspection of results obtained by sorting on importance</td>
</tr>
<tr>
<td>Method clustering</td>
<td>Browsing of results obtained by method integration</td>
</tr>
<tr>
<td>Call clustering</td>
<td>Manual inspection of suspicious clusters</td>
</tr>
<tr>
<td>Fan-in analysis</td>
<td>Selection of suspicious clusters from a list of methods, sorted on highest fan-in.</td>
</tr>
<tr>
<td>Clone detection</td>
<td>Browsing and interpretation of the discovered clones</td>
</tr>
</tbody>
</table>

*Manual effort still required*
Summary (1)

- Aspect Mining
  - Identification of crosscutting concerns in legacy systems
  - Aid developers when migrating an existing application to aspects
  - Browsing approaches and automated techniques
Summary(2)

- Different techniques are complementary
  - Different input (dynamic vs. static); granularity
  - Rely on different assumptions/symptoms
  - Make use of different analysis techniques
    - cluster analysis, FCA, heuristics, ...
- When mining a system, apply different techniques
Possible research directions (1)

• Combinations of techniques
  • static + dynamic
  • lexical + semantic
  • different assumptions
• Fill in the gaps in existing research
  • Only one technique looks for tangling
  • Look at sub-method level
Possible research directions (2)

- Quantitative comparison of techniques:
  - Which aspects can be detected by which technique?
  - Common benchmark/analysis framework
- Cross-fertilization with other research domains
  - slicing, metrics, ...
Limitations

• Joinpoint mining?
  • how obtain the pointcut?
  • how obtain the advice?

• Scalability

• Benchmarks

• Coverage
  • false positives/negatives
  • granularity