Aspect-Oriented Software Development
Course 2009

Identifying Concerns on Source Code
Techniques, Tools & Methods

Eng. Esteban S. Abait
ISISTAN – UNCPBA

Universidad Nacional del Centro de la Provincia de Buenos Aires

Agenda

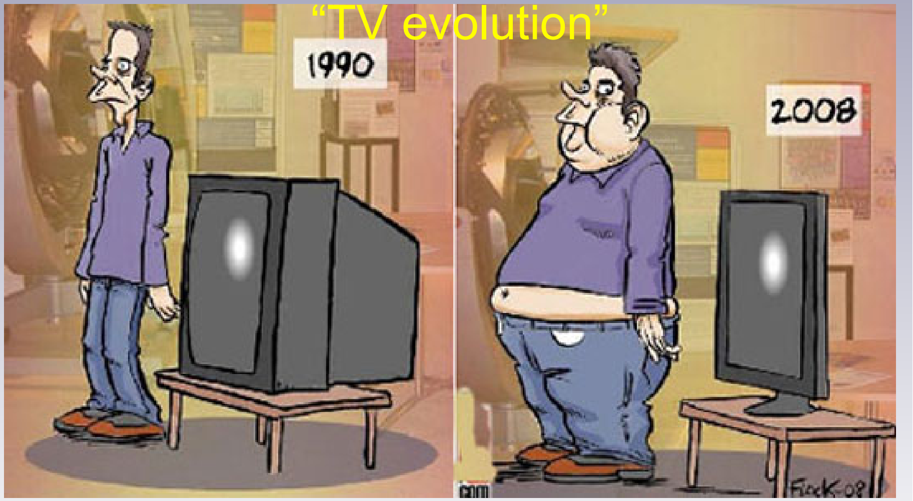
- Introduction
- Migration Process
- Aspect Mining Techniques
  - Static-based approaches
  - Dynamic-based approaches
- Concern-Sorts
- Pitfalls on Aspect Mining

Agenda

- Introduction
- Migration Process
- Aspect Mining Techniques
  - Static-based approaches
  - Dynamic-based approaches
- Concern-Sorts
- Pitfalls on Aspect Mining

Introduction: Software Evolution

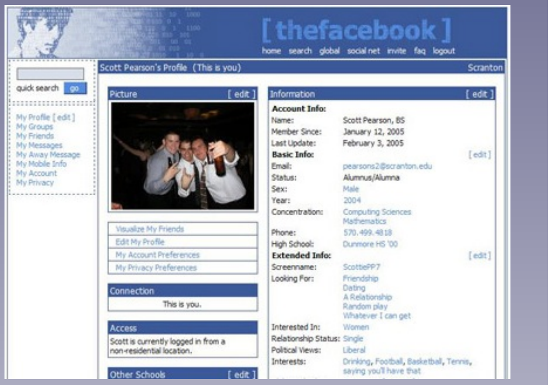
- The term evolution describes a phenomenon encountered in many different domains [1]
  - Classes of entities such as natural species, societies, concepts, ideas, for example, are said to evolve in time, each in its own context
- Real world software is essentially evolutionary in nature [1]



"TV evolution"

Introduction: Facebook Evolution

- Evolution of a real-world software: Facebook



[thefacebook]

home search global social net invite faq logout

quick search go

My Profile [edit]
My Groups
My Friends
My Messages
My Away Message
My Mobile Info
My Account
My Privacy

Scott Pearson's Profile (This is you) Scranton

Picture [edit]

Visualize My Friends
Edit My Profile
My Account Preferences
My Privacy Preferences

Connection
This is you.

Access
Scott is currently logged in from a non-residential location.

Other Schools [edit]

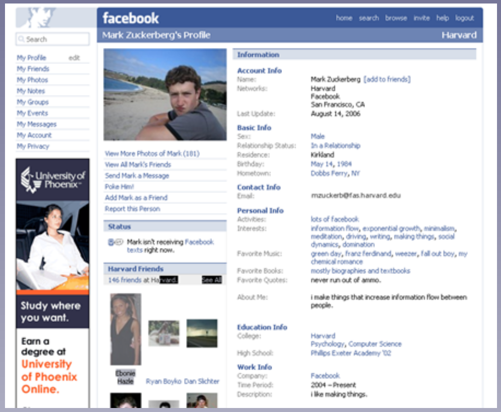
Information [edit]

Account Info:
Name: Scott Pearson, BS
Member Since: January 12, 2005
Last Update: February 3, 2005

Basic Info: [edit]
Email: pearsons2@scranton.edu
Status: Alumnus/Alumna
Sex: Male
Year: 2004
Concentration: Computing Sciences
Mathematics
Phone: 570.499.4818
High School: Dunmore HS '00

Extended Info: [edit]
Screenname: ScottiePP7
Looking For: Friendship
Dating
A Relationship
Random play
Whatever I can get
Interested In: Women
Relationship Status: Single
Political Views: Liberal
Interests: Drinking, Football, Basketball, Tennis, saying you'll have that

2004



facebook

home search browse invite help logout

Mark Zuckerberg's Profile Harvard

Search

My Profile edit
My Friends
My Photos
My Notes
My Groups
My Events
My Messages
My Account
My Privacy

University of Phoenix

Study where you want.

Earn a degree at University of Phoenix Online.

View More Photos of Mark (181)
View All Mark's Friends
Send Mark a Message
Poke Him!
Add Mark as a Friend
Report this Person

Status
Mark isn't receiving Facebook texts right now.

Harvard Friends
146 friends at Harvard. See All

Ebonie Hazle
Ryan Boyko
Dan Slichter

Information

Account Info
Name: Mark Zuckerberg [add to friends]
Networks: Harvard
Facebook
San Francisco, CA
Last Update: August 14, 2006

Basic Info
Sex: Male
Relationship Status: In a Relationship
Residence: Kirkland
Birthday: May 14, 1984
Hometown: Dobbs Ferry, NY

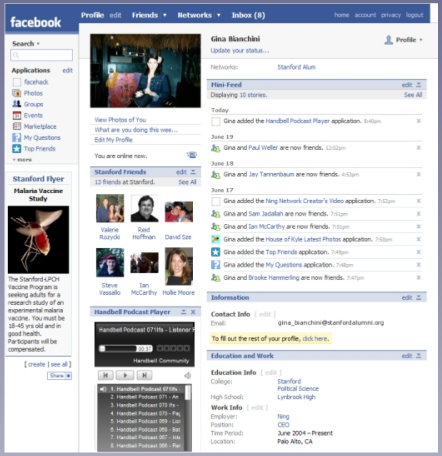
Contact Info
Email: mzuckerb@fas.harvard.edu

Personal Info
Activities: lots of facebook
Interests: information flow, exponential growth, minimalism, meditation, driving, writing, making things, social dynamics, domination
Favorite Music: green day, franz ferdinand, weezer, fall out boy, my chemical romance
Favorite Books: mostly biographies and textbooks
Favorite Quotes: never run out of ammo.
About Me: i make things that increase information flow between people.

Education Info
College: Harvard
Psychology, Computer Science
High School: Phillips Exeter Academy '02

Work Info
Company: Facebook
Time Period: 2004 – Present
Description: i like making things.

2006



facebook

Profile edit Friends ▾ Networks ▾ Inbox (8) home account privacy logout

Search ▾

Applications edit
facehack
Photos
Groups
Events
Marketplace
My Questions
Top Friends
▾ more

Stanford Flyer
Malaria Vaccine Study
The Stanford-LPCH Vaccine Program is seeking adults for a research study of an experimental malaria vaccine. You must be 18-45 yrs old and in good health. Participants will be compensated.
[create | see all]
Share +

View Photos of You
What are you doing this wee...
Edit My Profile
You are online now.

Stanford Friends edit
13 friends at Stanford. See All
Valerie Rozycki
Reid Hoffman
David Sze
Steve Vassallo
Ian McCarthy
Hollie Moore

Handbell Podcast Player
Handbell Podcast 071lfs - Listener
00:37
Handbell Community
1. Handbell Podcast 071lfs -
2. Handbell Podcast 071 - An
3. Handbell Podcast 070 lfs -
4. Handbell Podcast 070 - Pag
5. Handbell Podcast 069 - List
6. Handbell Podcast 068 - Bat
7. Handbell Podcast 067 - Inte
8. Handbell Podcast 066 - Ral

Gina Bianchini
Update your status...
Profile ▾

Networks: Stanford Alum

Mini-Feed edit
Displaying 10 stories. See All

Today
Gina added the Handbell Podcast Player application. 8:40pm

June 19
Gina and Paul Weller are now friends. 12:02pm

June 18
Gina and Jay Tannenbaum are now friends. 4:53pm

June 17
Gina added the Ning Network Creator's Video application. 7:52pm
Gina and Sam Jadallah are now friends. 7:51pm
Gina and Ian McCarthy are now friends. 7:51pm
Gina added the House of Kyle Latest Photos application. 7:50pm
Gina added the Top Friends application. 7:49pm
Gina added the My Questions application. 7:48pm
Gina and Brooke Hammerling are now friends. 7:47pm

Information edit

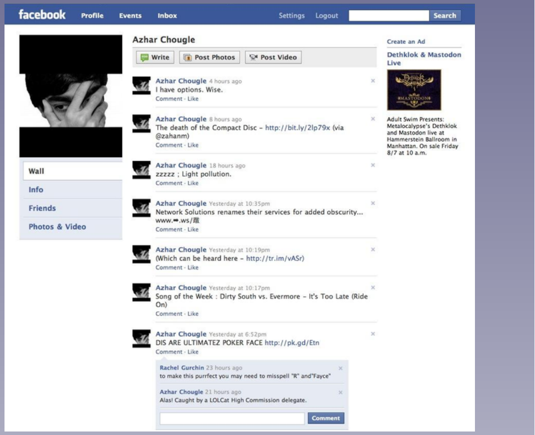
Contact Info [edit]
Email: gina_bianchini@stanfordalumni.org
To fill out the rest of your profile, click here.

Education and Work edit

Education Info [edit]
College: Stanford
Political Science
High School: Lynbrook High

Work Info [edit]
Employer: Ning
Position: CEO
Time Period: June 2004 – Present
Location: Palo Alto, CA

2007



facebook Profile Events Inbox Settings Logout Search

Wall
Info
Friends
Photos & Video

Azhar Chougle

Write Post Photos Post Video

Azhar Chougle 4 hours ago
I have options. Wise.
Comment · Like

Azhar Chougle 8 hours ago
The death of the Compact Disc – http://bit.ly/2lp79x (via @zahanm)
Comment · Like

Azhar Chougle 18 hours ago
zzzzz ; Light pollution.
Comment · Like

Azhar Chougle Yesterday at 10:35pm
Network Solutions renames their services for added obscurity... www.➡.ws/䨻
Comment · Like

Azhar Chougle Yesterday at 10:19pm
(Which can be heard here – http://tr.im/vASr)
Comment · Like

Azhar Chougle Yesterday at 10:17pm
Song of the Week : Dirty South vs. Evermore – It's Too Late (Ride On)
Comment · Like

Azhar Chougle Yesterday at 6:52pm
DIS ARE ULTIMATEZ POKER FACE http://pk.gd/Etn
Comment · Like

Rachel Gurchin 23 hours ago
to make this purrfect you may need to misspell "R" and"Fayce"

Azhar Chougle 21 hours ago
Alas! Caught by a LOLCat High Commission delegate.

Comment

Create an Ad

Dethklok & Mastodon Live

Adult Swim Presents: Metalocalypse's Dethklok and Mastodon live at Hammerstein Ballroom in Manhattan. On sale Friday 8/7 at 10 a.m.

2009

February: Mark Zuckerberg and co-founders launch Facebook from their Harvard dorm room

December: Facebook reaches 1 millon users

Open its registration, anyone can join the social network

Launches a mobile feature

Reaches 12 millions users

Add a gift shop feature

Reaches 50 million users

2008: translation to 21 languages

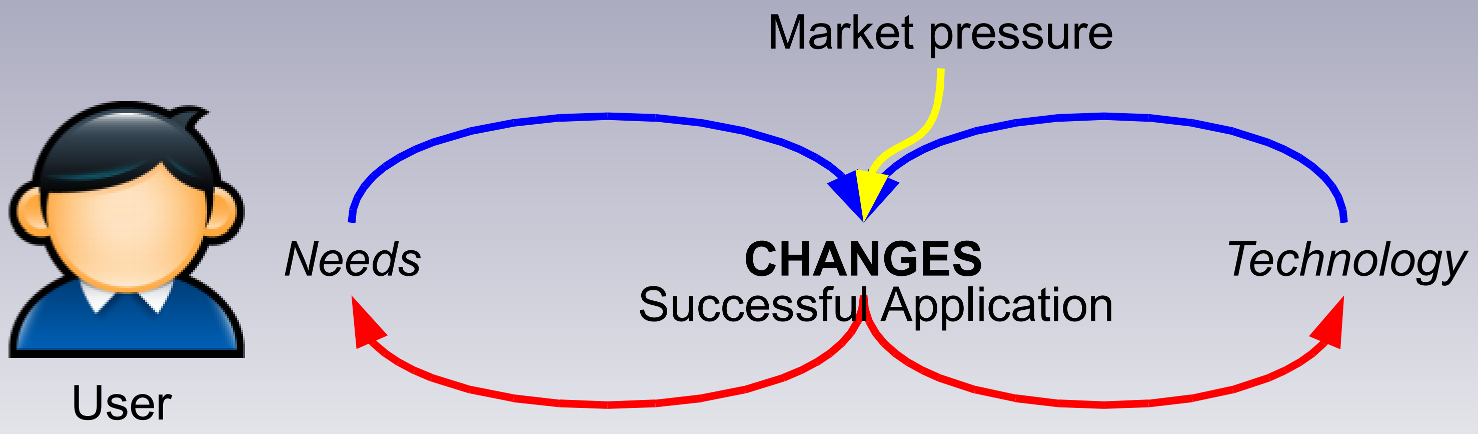
Reaches 300 million active users

http://www.facebook.com/press/info.php?timeline

Introduction: Laws of Software Evolution

(1) **Continuing Change:** Real-world systems must be continually adapted else they become progressively less satisfactory

(2) **Increasing Complexity:** As an Real-world system evolves its complexity increases unless work is done to maintain or reduce it



Introduction: Crosscutting Concerns (1)

- Crosscutting concerns are believed to negatively affect *evolvability*, maintainability and understandability [2]
  - A change to a crosscutting concern is likely to affect many different places in the source code
- AOP propose a solution to this problem by introducing the notion of aspects
  - An aspect is a language construct that allows us to localize a concern's implementation

Introduction: Crosscutting Concerns (2)

- Why crosscutting concerns negatively affect software properties like *evolvability*, maintainability or understandability?
  - A deadlock on the locking mechanism of ***eclipse*** had as a result the modification of 2573 methods
  - Developers inserted in 1 284 methods a call to lock, as well as a call to unlock

Agenda

- Introduction
- Migration Process
- Aspect Mining Techniques
  - Static-based approaches
  - Dynamic-based approaches
- Concern-Sorts
- Pitfalls on Aspect Mining

Migration Process: Overview

- In order to use aspects on existing legacy software systems we need techniques to [2, 3]:

  1. Identify where the crosscutting concerns are in the source code – Aspect Mining
  2. Discover the full extent for each of the discovered concerns – Aspect Extraction
  3. Encapsulate each crosscutting concern in an aspect of the new system – Aspect Refactoring

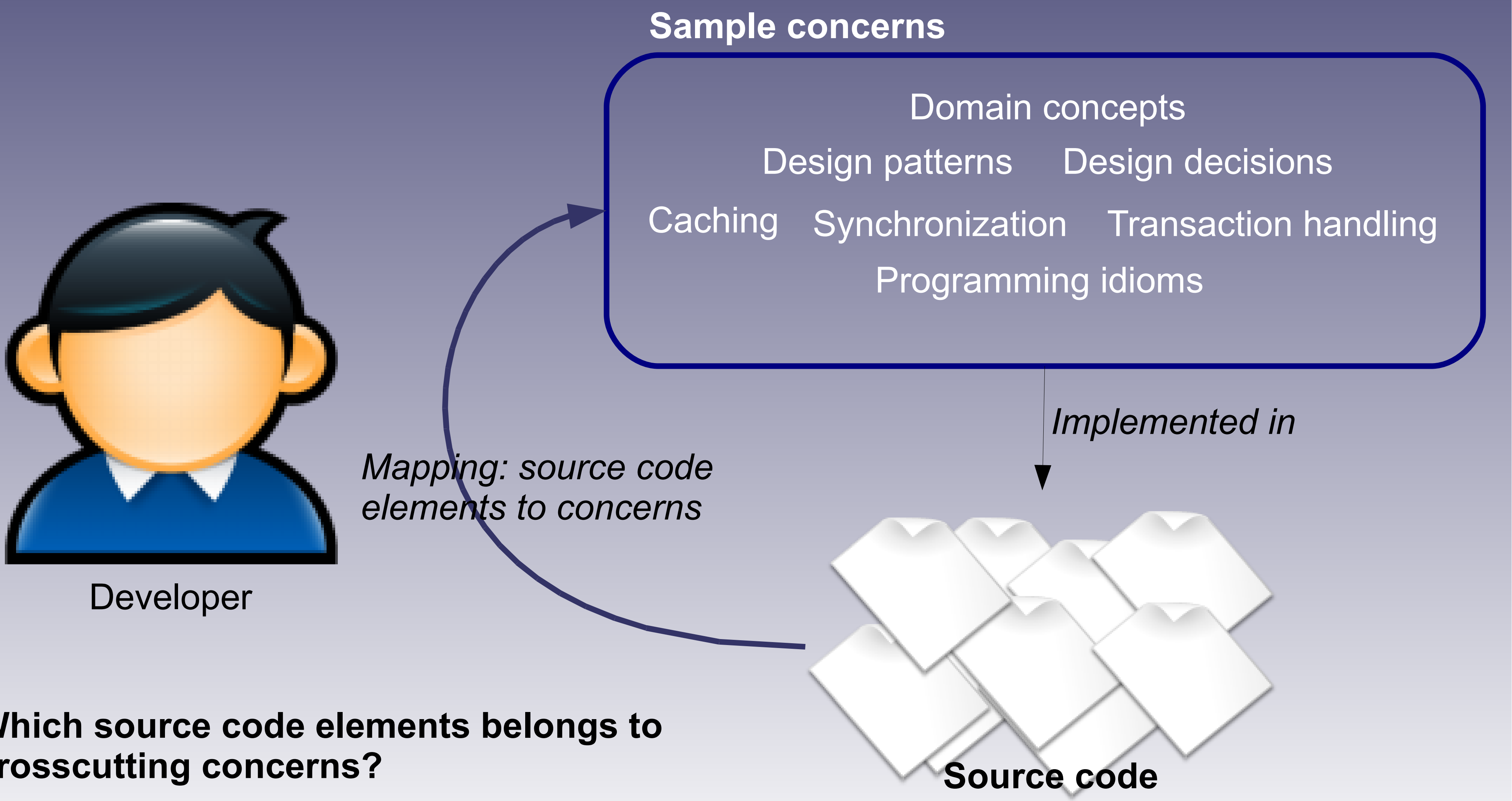
Agenda

- Introduction
- Migration Process
- Aspect Mining Techniques
  - Static-based approaches
  - Dynamic-based approaches
- Concern-Sorts
- Pitfalls on Aspect Mining

Aspect Mining: Introduction (1)

- Aspect mining aims to identify crosscutting concerns in existing systems, thereby improving the system’s comprehensibility and enabling migration of existing (object-oriented) programs to aspect-oriented ones
- Why (semi-)automatic techniques are needed?
  - The sheer *size and complexity* of many existing systems, combined with the *lack of documentation and knowledge* of such systems render it practically infeasible to manually transform their crosscutting concerns into aspects [2]

Aspect Mining: Mapping Concerns to Code



Sample concerns

Domain concepts

Design patterns Design decisions

Caching Synchronization Transaction handling

Programming idioms

Implemented in

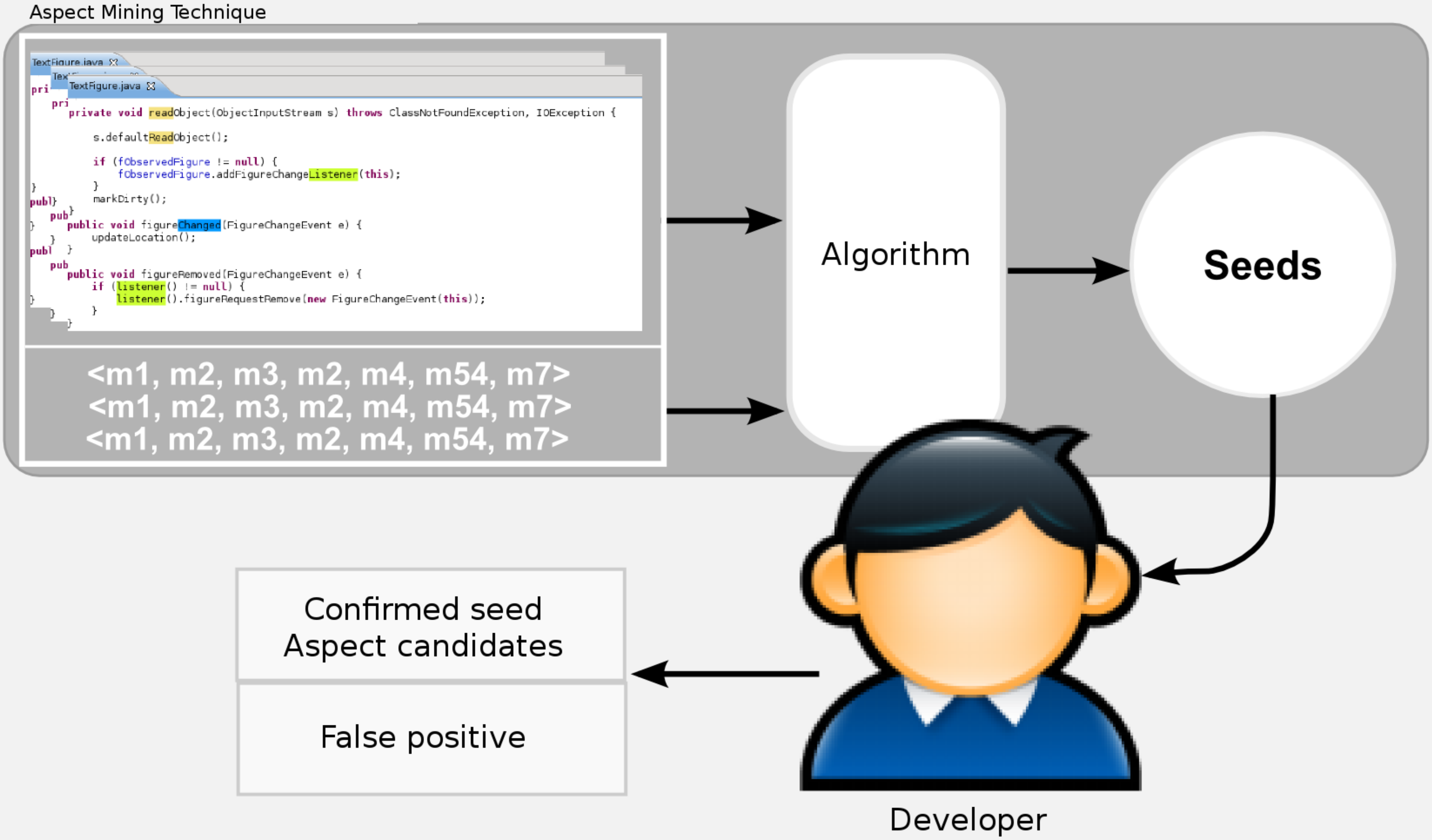
Mapping: source code elements to concerns

Developer

Source code

Which source code elements belongs to crosscutting concerns?

Aspect Mining: Process



```
private void readObject(ObjectInputStream s) throws ClassNotFoundException, IOException {

    s.defaultReadObject();

    if (fObservedFigure != null) {
        fObservedFigure.addFigureChangeListener(this);
    }
    markDirty();
}
public void figureChanged(FigureChangeEvent e) {
    updateLocation();
}

public void figureRemoved(FigureChangeEvent e) {
    if (listener() != null) {
        listener().figureRequestRemove(new FigureChangeEvent(this));
    }
}
```

<m1, m2, m3, m2, m4, m54, m7>
<m1, m2, m3, m2, m4, m54, m7>
<m1, m2, m3, m2, m4, m54, m7>

Confirmed seed
Aspect candidates

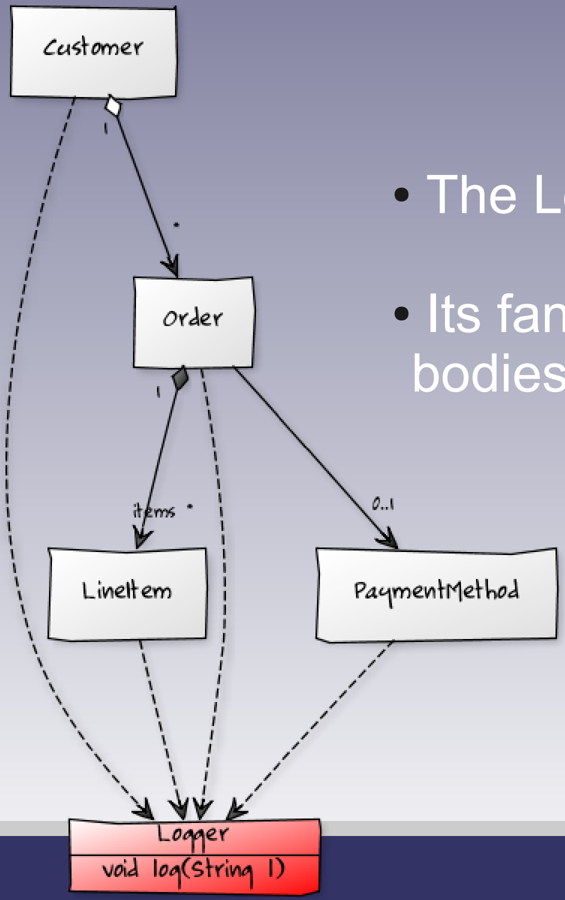
False positive

Agenda

- Introduction
- Migration Process
- Aspect Mining Techniques
  - Static-based approaches
  - Dynamic-based approaches
- Concern-Sorts
- Pitfalls on Aspect Mining

Static-based techniques: Fan-In Analysis (1)

- Determining methods that are called from many different places (and hence have a high fan-in) to identify candidate aspects



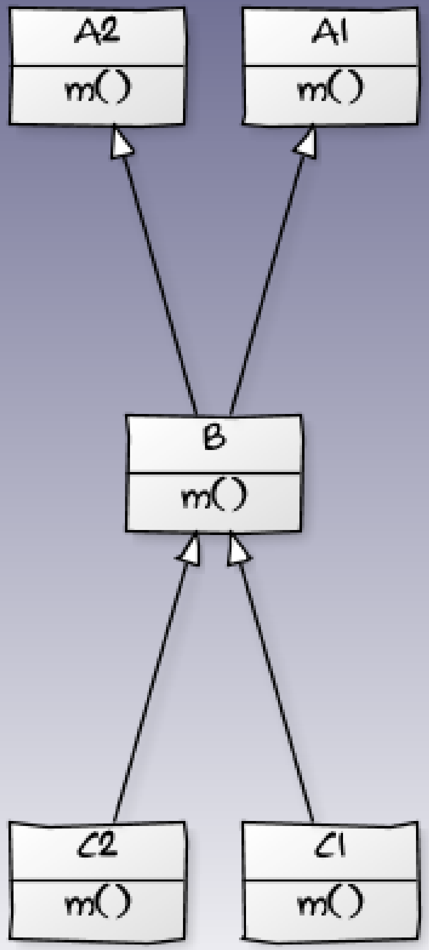
- The Logger's log method is called from different places
- Its fan-in value depends on the number of different methods bodies that use it

Static-based techniques: Fan-In Analysis (2)

- Marin et al. [4] proposed the use of this metric for finding methods that belongs to the implementation of a crosscutting concern
- Fan-in definition:
  - the fan-in of method *m* is the number of distinct method bodies that invoke *m*
  - due to polymorphism, a call to a method *m* contributes to all methods refining *m* as well as all method refined by *m*

Static-based techniques: Fan-In Analysis (3)

- Example of fan-in calculation

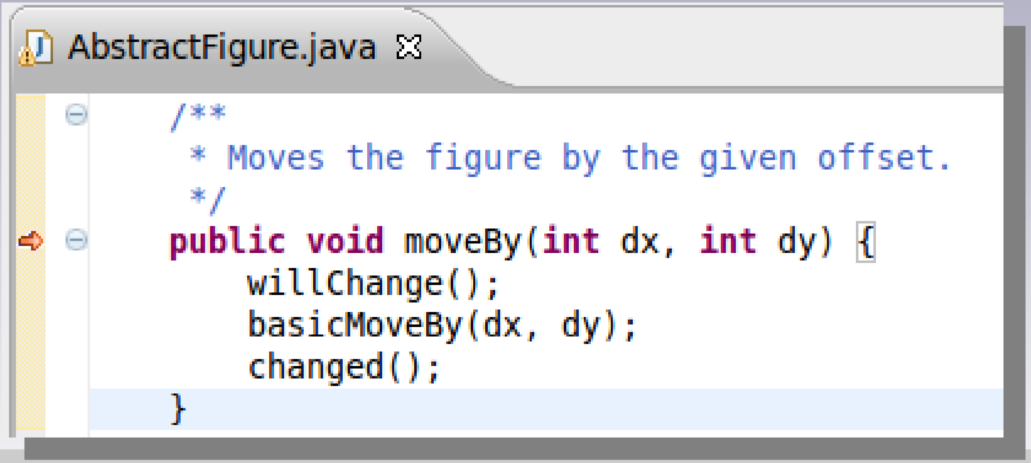


	Fan-in contribution				
Call site	A1.m	A2.m	B.m	C1.m	C2.m
f1(A1 a1) {a1.m();}	1	0	1	1	1
f2(A2 a2) {a2.m();}	0	1	1	1	1
f3(B b) {b.m();}	1	1	1	1	1
f4(C1 c1) {c1.m();}	1	1	1	1	0
f5(C2 c2) {c2.m();}	1	1	1	0	1
Total fan-in	4	4	5	4	4

Static-based techniques: Fan-In Analysis (3)

- Fan-in application on JHotDraw 5.4b1

Method	Fan-in	Concern
framework.Figure.willChange()	25	Observer
standard.AbstractFigure.changed()	37	Observer
util.StorableInput.readInt()	22	Persistence
util.StorableOutput.writeInt(int)	21	Persistence
util.UndoableAdapter.undo()	24	Undo
util.Undoable.isRedoable()	24	Undo



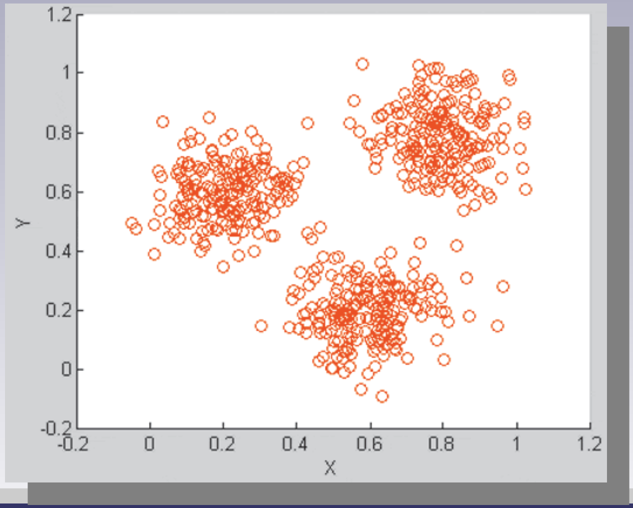
```
/**
 * Moves the figure by the given offset.
 */
public void moveBy(int dx, int dy) {
    willChange();
    basicMoveBy(dx, dy);
    changed();
}
```

Demo time

- Lets see how to work with an aspect mining tool

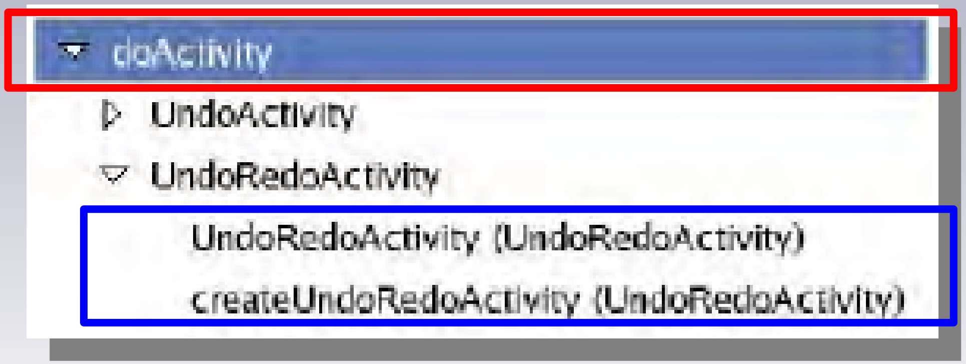
Static-based techniques: Clustering Based Mining

- Several works on aspect mining relies on the use of clustering algorithms
- A clustering algorithm groups elements into clusters based on a distance function
  - Each cluster maximizes the similarity between elements of the same cluster and minimizes the similarity among elements of different clusters



Static-based techniques: Clustering Based Mining Grouping Similar Method Names

- Shepherd and Pollock [5] proposed the use of clustering for grouping methods that share similar names
- *Hierarchical Clustering*: leave nodes are labeled with the name of the method they represent, non leaf nodes are labeled with the common substring of their children



doActivity

UndoActivity

UndoRedoActivity

UndoRedoActivity (UndoRedoActivity)

createUndoRedoActivity (UndoRedoActivity)

Non-leaf nodes: labeled with the common substring

Leaf nodes: method names from the application

Static-based techniques: Clustering Based Mining Grouping Methods by Fan-In Value (1)

- Moldovan and Serban [6] reported the application of three different clustering algorithms for aspect mining
- Steps
  - Computation. Computation of the set of methods in the selected source code and, for each method in the set, computation of the attributes set values.
  - Filtering.
  - Grouping (K-means; HAC; Fuzzy C-means)
  - Analysis by the developer

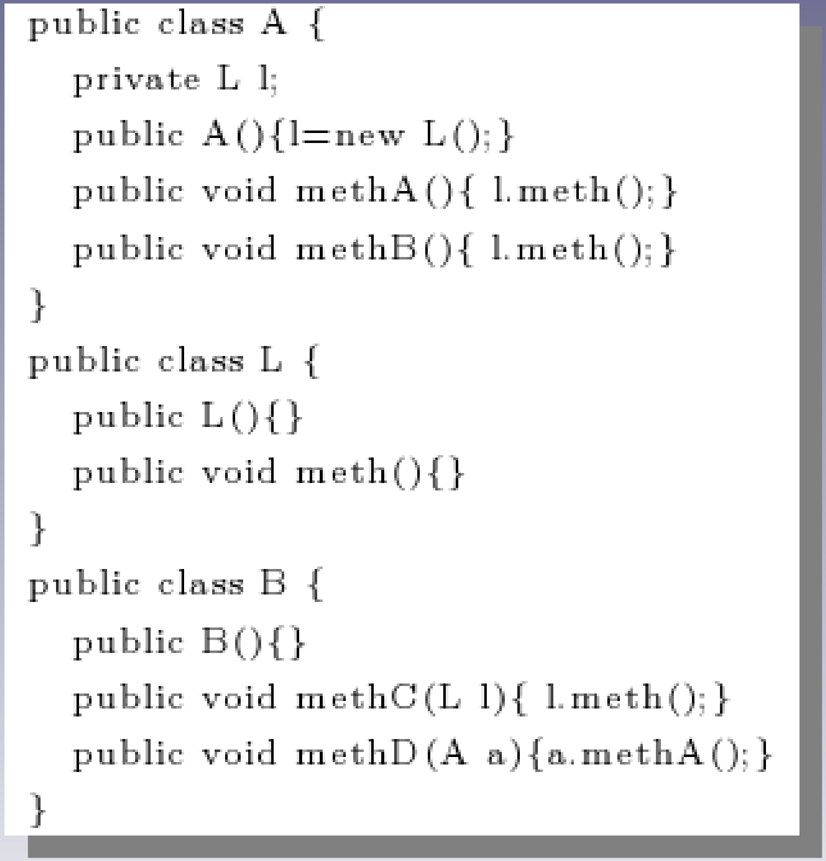
Static-based techniques: Clustering Based Mining Grouping Methods by Fan-In Value (2)

- In this approach, the methods to be clustered are represented as a l-dimensional vector: $m_i = \{m_{i1}, ..., m_{il}\}$
- The author considered two vector-space models
  - M1: {FIV, CC}, where FIV is the fan-in value and CC is the number of calling classes
  - M2: $\{FIV, B_1, B_2, ..., B_{l-1}\}$, where FIV is the fan-in value, and the value of B_i is 1, if the method M is called from a method belonging to C_i, 0 otherwise

Authors considered that while the fan-in value of a method is important also is the number of calling classes

Static-based techniques: Clustering Based Mining Grouping Methods by Fan-In Value (3)

- Example with the vector-space model M1



```
public class A {
   private L l;
   public A(){l=new L();}
   public void methA(){ l.meth();}
   public void methB(){ l.meth();}
}
public class L {
   public L(){}
   public void meth(){}
}
public class B {
   public B(){}
   public void methC(L l){ l.meth();}
   public void methD(A a){a.methA();}
}
```

Method	FIV	CC
A.A	0	0
A.methA	1	1
A.methB	0	0
B.B	0	0
B.methC	0	0
B.methD	0	0
L.L	1	1
L.meth	3	2

Cluster	Methods
C1	{L.meth}
C2	{A.methA, L.L}
C3	{A.A, A.methB, B.B, B.methC, B.methD }

Static-based techniques: Clone Detection Techniques (1)

- Code cloning as defined by Ryssel and Demeyer [7] is the act of copying code fragments and making minor non-functionals modifications
- The presence of duplicated code can be linked to the presence of
  - Bad (code) smells [8]
  - Crosscutting concerns

While a crosscutting concern represents something that cannot be modularized given the used language, a bad smell is any symptom in the code that possibly indicates a deeper problem

Static-based techniques: Clone Detection Techniques (2)

- Code duplication as a bad smell
  - Fowler and Beck [8] considered code duplication as the number one in the *stink parade*
- In this case, the duplicated code is fixed by applying one of the followings object-oriented refactorings
  - Extract method
  - Extract class

More on bad smells:

- http://www.soberit.hut.fi/mmantyla/BadCodeSmellsTaxonomy.htm
- http://c2.com/cgi/wiki?CodeSmell
- Fowler's book [8]
- Marinescu's book [9]

Static-based techniques: Clone Detection Techniques (3)

- The presence of crosscutting concerns may result in duplicated code
  - Developers may be unable to reuse concern implementations through the language module mechanism
  - Developers may use particular coding conventions and idioms to implements superimposed functionality
    - E.g.: tracing, logging, transactions, etc

As a consequence, clone detection techniques might be suitable for identifying some kinds of crosscutting concern code [10]

Static-based techniques: Clone Detection Techniques (4)

- Bruntink et al. [10] evaluated the suitability of clone detection techniques for automatically identifying crosscutting concern code.
- The authors considered a single component of a large-scale, industrial software system, consisting of 16,406 non-blank lines of C code

	Precision			
Concern	**Line count (%)**	**AST-based**	**Token-based**	**PDG-based**
Memory handling	750 (4.6%)	0.65	0.63	0.81
Null pointer checking	617 (3.8%)	0.99	0.97	0.80
Range checking	387 (2.4%)	0.71	0.59	0.42
Exception handling	927 (5.7%)	0.38	0.36	0.35
Tracing	1501 (9.1%)	0.62	0.57	0.68

Static-based techniques: Clone Detection Techniques (5)

- Clone detectors achieved higher precision and recall for concerns that exhibited relatively low tangling with other concerns or with the base code, than for concerns that exhibited high tangling

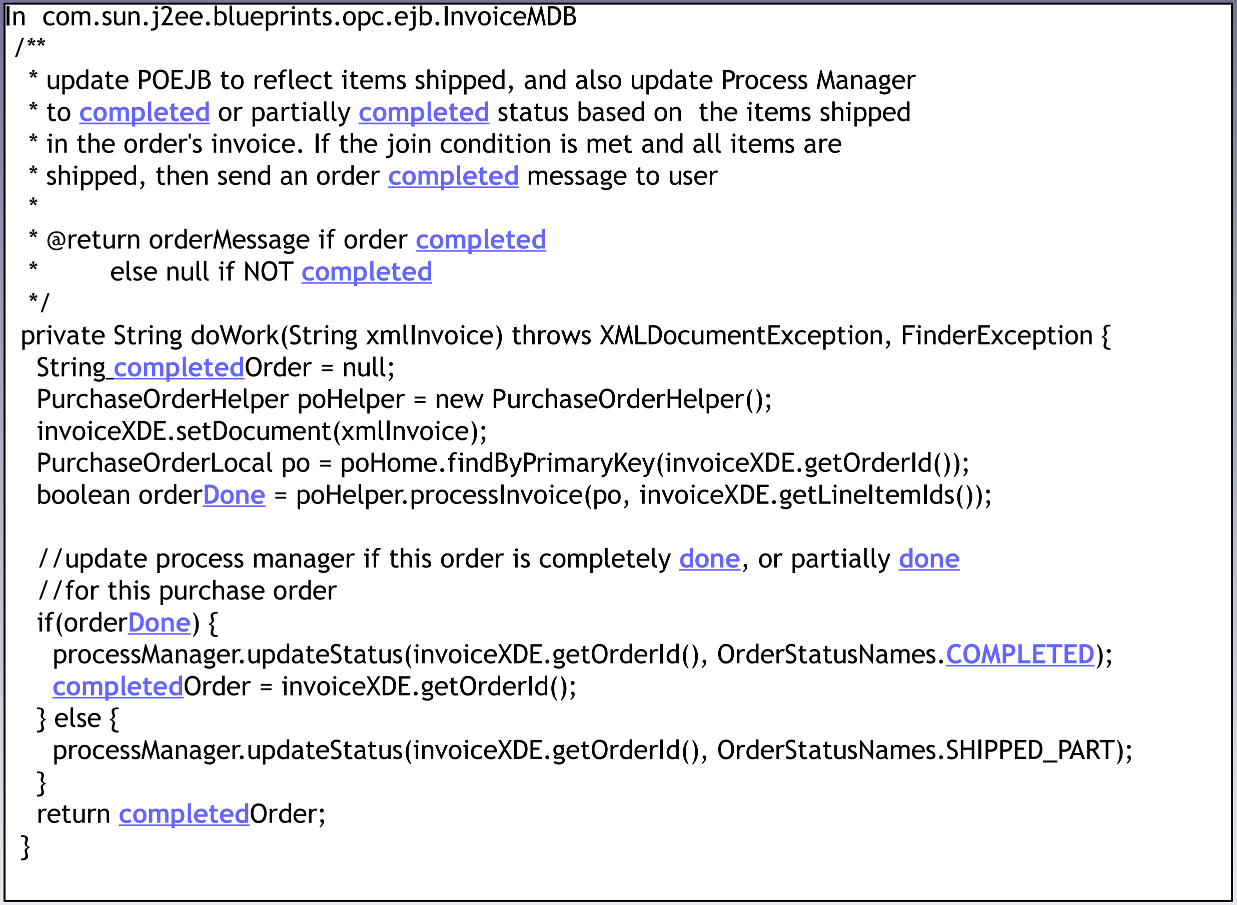
	Recall		
Concern	**AST-based**	**Token-based**	**PDG-based**
Memory handling	0.65	0.63	0.81
Null pointer checking	0.99	0.97	0.80
Range checking	0.71	0.59	0.42
Exception handling	0.38	0.36	0.35
Tracing	0.62	0.57	0.68

Static-based techniques: NPL Analysis (1)

- Shepher et al. [11] tried to identify crosscutting concerns in existing source code by exploiting the natural language clues that the developers left behind
- Use of *lexical chaining* to identify groups of semantically related source code entities, and evaluate whether those groups represent crosscutting concerns
- The assumption behind this technique is that crosscutting concerns are reflected in source code through naming conventions

Static-based techniques: NPL Analysis (1)

- Example of lexical chain: finished



```
In  com.sun.j2ee.blueprints.opc.ejb.InvoiceMDB
 /**
   * update POEJB to reflect items shipped, and also update Process Manager
   * to completed or partially completed status based on  the items shipped
   * in the order's invoice. If the join condition is met and all items are
   * shipped, then send an order completed message to user
   *
   * @return orderMessage if order completed
   *         else null if NOT completed
   */
  private String doWork(String xmlInvoice) throws XMLDocumentException, FinderException {
    String completedOrder = null;
    PurchaseOrderHelper poHelper = new PurchaseOrderHelper();
    invoiceXDE.setDocument(xmlInvoice);
    PurchaseOrderLocal po = poHome.findByPrimaryKey(invoiceXDE.getOrderId());
    boolean orderDone = poHelper.processInvoice(po, invoiceXDE.getLineItemIds());

    //update process manager if this order is completely done, or partially done
    //for this purchase order
    if(orderDone) {
      processManager.updateStatus(invoiceXDE.getOrderId(), OrderStatusNames.COMPLETED);
      completedOrder = invoiceXDE.getOrderId();
    } else {
      processManager.updateStatus(invoiceXDE.getOrderId(), OrderStatusNames.SHIPPED_PART);
    }
    return completedOrder;
  }
```

Static-based techniques: Formal Concept Analysis (1)

- Formal Concept Analysis (FCA) provides a way to identify maximal groupings of elements that share common attributes
- Input: A “context” which defines the relations between the elements and the attributes

Programming language	OO	Functional	Logic	Static typing	Dynamic typing
Java	X	-	-	X	-
Smalltalk	X	-	-	-	X
C++	X	-	-	X	-
Scheme	-	X	-	-	X
Prolog	-	-	X	-	X

Static-based techniques: Formal Concept Analysis (2)

- Based on a given *context*, the FCA algorithm finds maximal groups of objects and attributes (*concepts*) such that:
  - each object of the concept shares the attributes of the concept
  - every attribute of the concept holds for all of the concept's objects
  - no other object outside the concept has those same attributes, nor does any attribute outside the concept hold for all objects in the concept

Static-based techniques: Formal Concept Analysis (3)

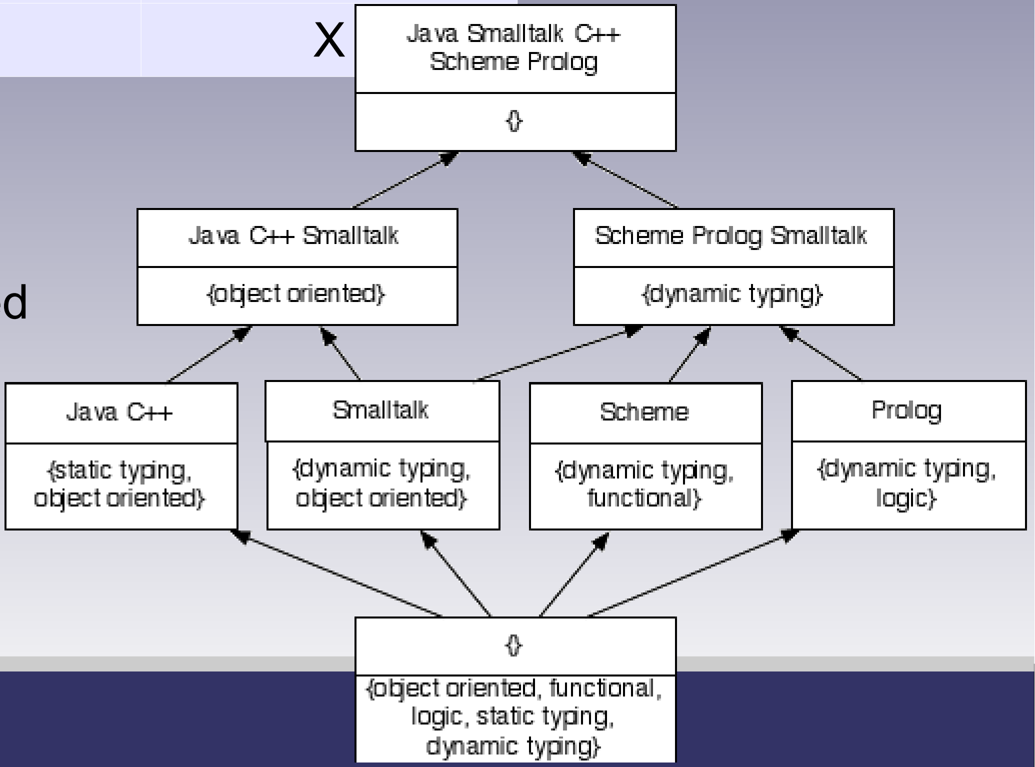
- Example: Lattice construction

PL	OO	Functional	Logic	Static typing	Dynamic typing
Java	X	-	-	X	-
Smalltalk	X	-	-	-	X
C++	X	-	-	X	-
Scheme	-	X	-	-	X
Prolog	-	-	X	-	X

- The bottom concept contains those elements that share all properties

- The top concept contains the properties shared by all elements

- The concept ({ Java, C++ }, { static typing, object oriented}), for example, groups all statically-typed object-oriented languages


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}

Static-based techniques: FCA + Identifiers (1)

- Towré and Mens [12] proposed to apply FCA algorithms in order to group elements of the source code that shares common substrings in their identifiers
- Examples of how to decompose several entities of the source code into substrings:
  - For a class: QuotedCodeConstant => 'quoted' 'code' 'constant'
  - For a method, the authors also considered the names of the parameters

```
 unifyWithDelayedVariable:inEnv:myIndex:hisIndex:inSource:
=> 'unify' 'delayed' 'variable' 'env' 'index' 'source'
```

Static-based techniques: FCA + Identifiers (2)

- Which is the condition to classify a concept as an aspect candidate?
  - When a concept contains only methods, defined in different classes without a common superclass (except for Object)
- Seeds reported by the authors
  - “delayed variable”
    - Class: DelayedVariable
    - Method: delayedVariableVisit → Visitor pattern
    - Method: buildDelayedVariable → Builder pattern
    - Method: makeDelayedVariable → Factory pattern
    - Method: unifyWithDelayedVariable → unify CC

Agenda

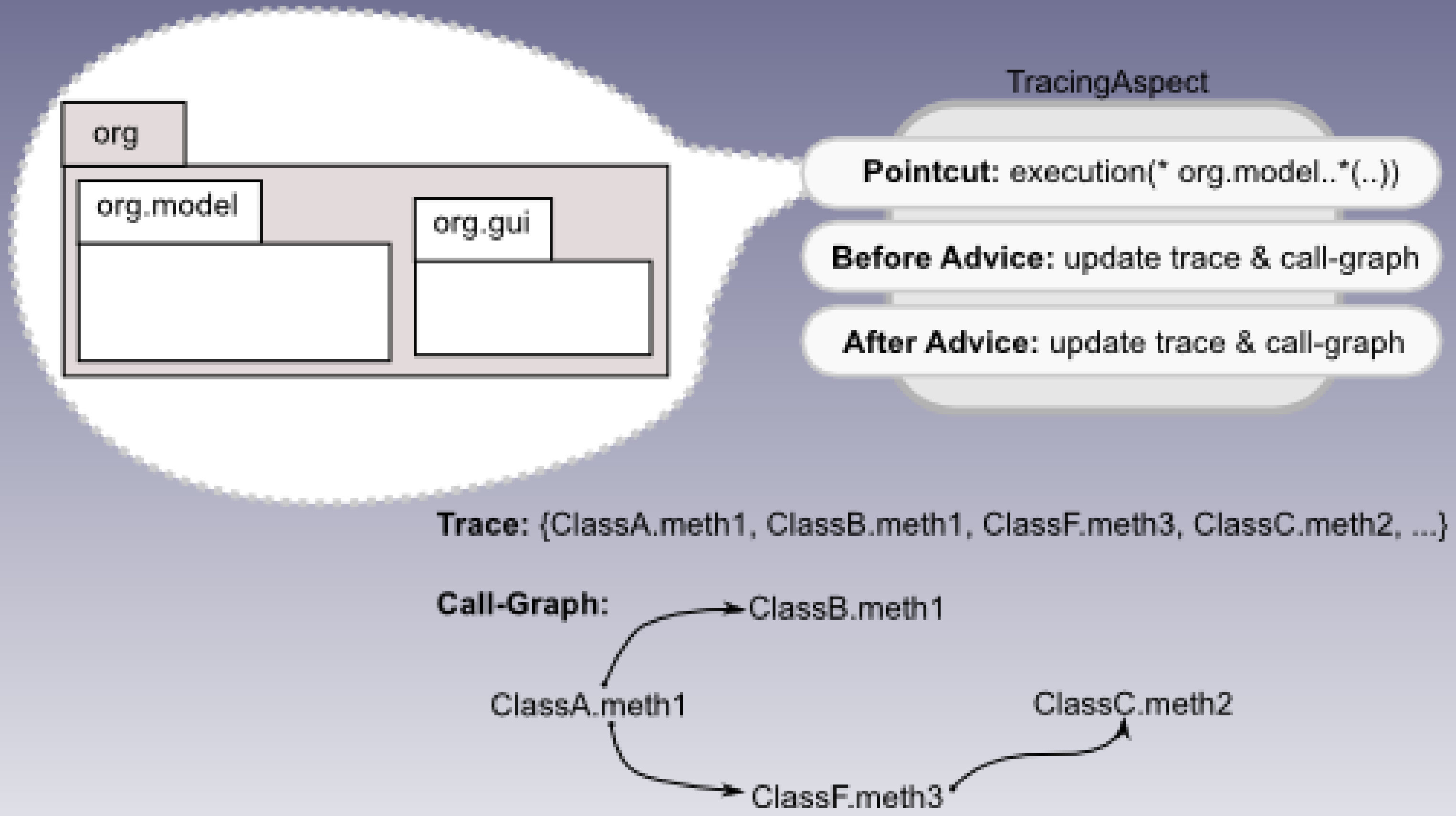
- Introduction
- Migration Process
- Aspect Mining Techniques
  - Static-based approaches
  - Dynamic-based approaches
- Concern-Sorts
- Pitfalls on Aspect Mining

Dynamic-based techniques: Dynamic Analysis Introduction (1)

- As Ball [13] put forth, dynamic analysis is the analysis of the properties of a running program
  - For instance, software testing and profiling are techniques based on dynamic analysis
- To perform dynamic analysis the following issues must be addressed:
  - System instrumentation
  - Definition of a set of execution scenarios
  - System executions

Dynamic-based techniques: Dynamic Analysis Introduction (2)

- Instrumentation example from [14]



TracingAspect

Pointcut: execution(* org.model..*(..))

Before Advice: update trace & call-graph

After Advice: update trace & call-graph

Trace: {ClassA.meth1, ClassB.meth1, ClassF.meth3, ClassC.meth2, ...}

Call-Graph:

A trace is, at least, a list of the methods executed for one execution scenario

Dynamic-based techniques: Execution Patterns (1)

- Breu and Krinke [15] proposed to analyze the execution traces to find *recurring execution patterns*
- Recurring execution patterns describe certain behavioral aspects of the software system
  - The authors consider this patterns as potential crosscutting concerns

Scenario: debti money from an account
Trace:

```
safeDebit(Subject, Money)
    authenticateUser(Subject)
    debitMoney(this, Money)
    commitTransaction()
```

Scenario: query the account's balance
Trace:

```
safeGetBalance(Subject)
    authenticateUser(Subject)
    getBalance(this)
```

Scenario: credit money to a an account
Trace:

```
safeCreditBalance(Subject, Money)
    authenticateUser(Subject)
    getBalance(this)
    commitTransaction()
```

Dynamic-based techniques: Execution Patterns (2)

- In order to detect these recurring patterns in the program traces, a classification of possible pattern forms is required
- The execution relations describe in which relation two method executions are in the program trace

Four different execution relations were defined by the authors:

- **outside-before** (B is called before A)
- **outside-after** (A is called after B)
- **inside-first** (G is the first call in C)
- **inside-last** (H is the last call in C)

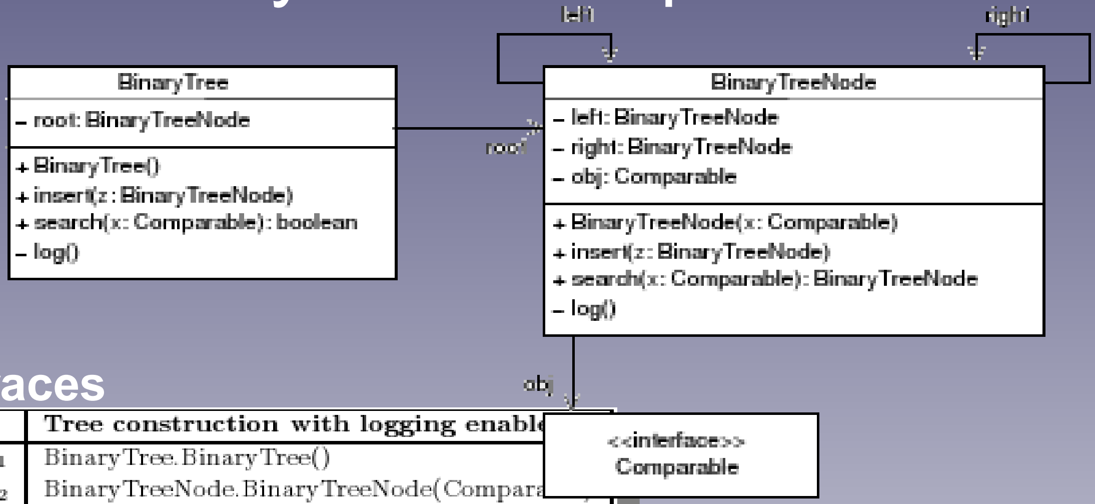
```
B() {
      C() {
          G()
          H()
      }
}
A() {}
```

Dynamic-based techniques: FCA Analysis Over Execution Traces (1)

- In Ceccato and Tonella [16] the relationship between execution traces and executed computational units is subjected to concept analysis
- In the resulting concept lattice, the concepts specific of each use case are located, yet if some of the following conditions holds then the concept can be classified as a seed
  - 1. Concept *c* is labeled by computational units (methods) that belong to more than one module (class)
  - 2. Different computational units (methods) from the same module (class) label more than one concept in *C*

Dynamic-based techniques: FCA Analysis Over Execution Traces (2)

- Binary tree example



Context

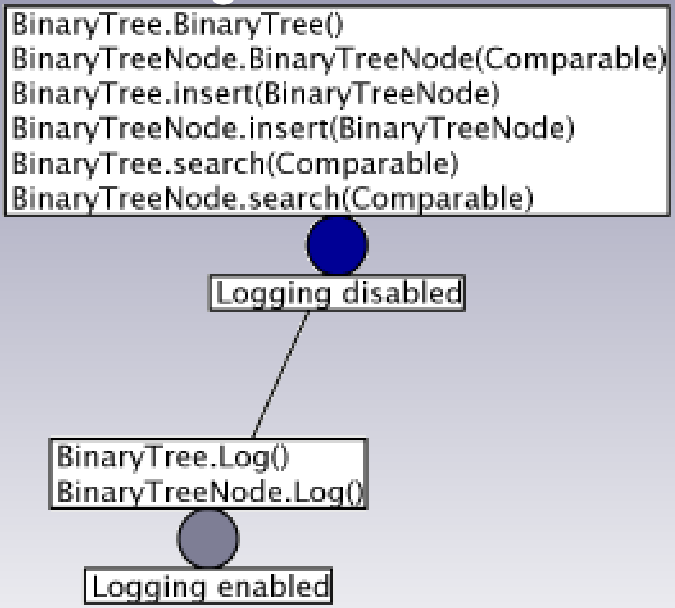
	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8
Logging enabled	×	×	×	×	×	×	×	×
Logging disabled	×	×	×		×		×	×

Traces

	Tree construction with logging enable
m_1	BinaryTree.BinaryTree()
m_2	BinaryTreeNode.BinaryTreeNode(Compara
m_3	BinaryTree.insert(BinaryTreeNode)
m_4	BinaryTree.log()
m_5	BinaryTreeNode.insert(BinaryTreeNode)
m_6	BinaryTreeNode.log()
m_7	BinaryTree.search(Comparable)
m_4	BinaryTree.log()
m_8	BinaryTreeNode.search(Comparable)
m_6	BinaryTreeNode.log()

	Tree construction with logging disabled
m_1	BinaryTree.BinaryTree()
m_2	BinaryTreeNode.BinaryTreeNode(Comparable)
m_3	BinaryTree.insert(BinaryTreeNode)
m_5	BinaryTreeNode.insert(BinaryTreeNode)
m_7	BinaryTree.search(Comparable)
m_8	BinaryTreeNode.search(Comparable)

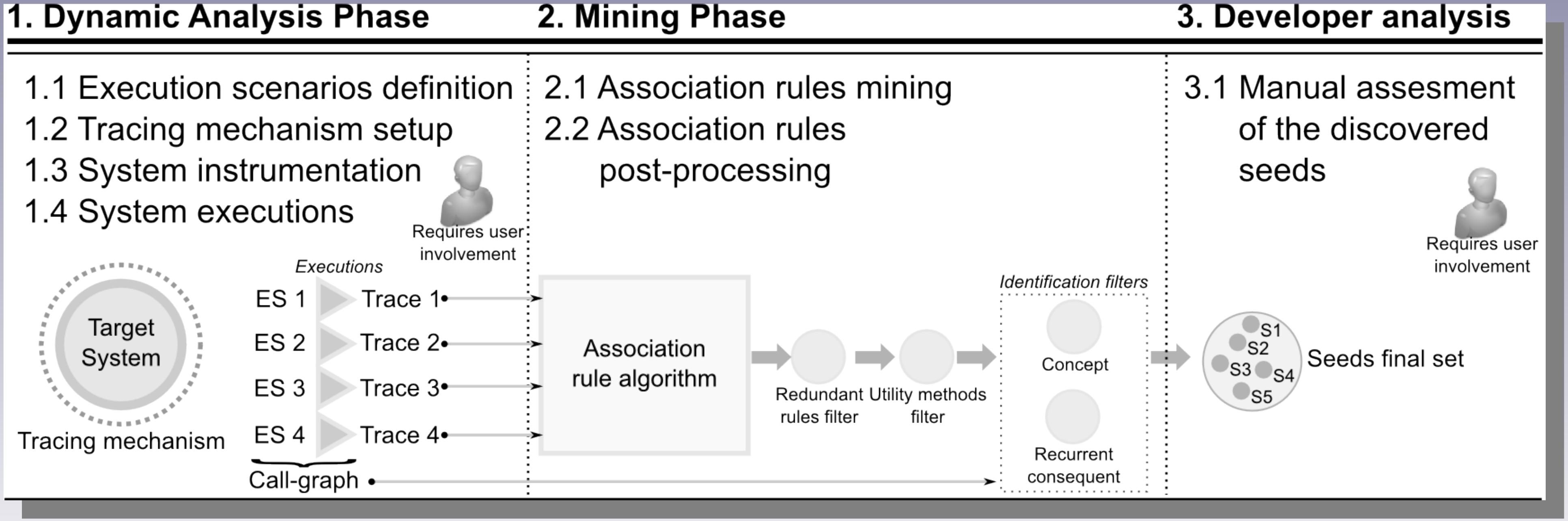
Resulting Lattice



BinaryTree.BinaryTree()
BinaryTreeNode.BinaryTreeNode(Comparable)
BinaryTree.insert(BinaryTreeNode)
BinaryTreeNode.insert(BinaryTreeNode)
BinaryTree.search(Comparable)
BinaryTreeNode.search(Comparable)
Logging disabled
BinaryTree.Log()
BinaryTreeNode.Log()
Logging enabled

Dynamic-based techniques: Association Rules (1)

- Abait and Marcos [14, 17] proposed an aspect mining technique that consists in the generation of execution traces and its analysis with association rules algorithms



1. Dynamic Analysis Phase

1.1 Execution scenarios definition
1.2 Tracing mechanism setup
1.3 System instrumentation
1.4 System executions

2. Mining Phase

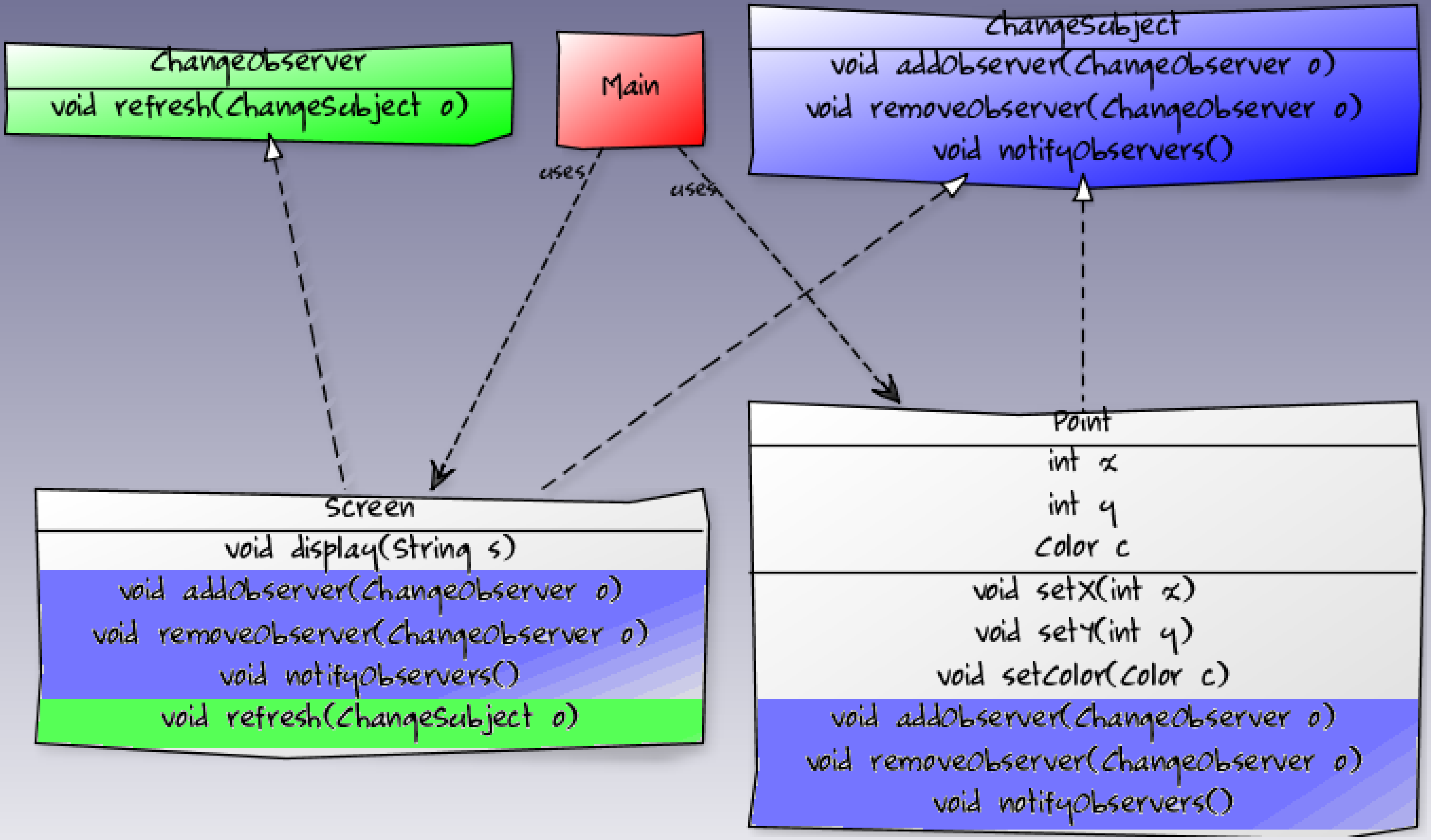
2.1 Association rules mining
2.2 Association rules post-processing

3. Developer analysis

3.1 Manual assesment of the discovered seeds

Dynamic-based techniques: Association Rules (2)

- Observer pattern example


ChangeObserver
void refresh(ChangeSubject o)
Main
ChangeSubject
void addObserver(ChangeObserver o)
void removeObserver(ChangeObserver o)
void notifyObservers()
uses
uses
Screen
void display(String s)
void addObserver(ChangeObserver o)
void removeObserver(ChangeObserver o)
void notifyObservers()
void refresh(ChangeSubject o)
Point
int x
int y
Color c
void setX(int x)
void setY(int y)
void setColor(Color c)
void addObserver(ChangeObserver o)
void removeObserver(ChangeObserver o)
void notifyObservers()

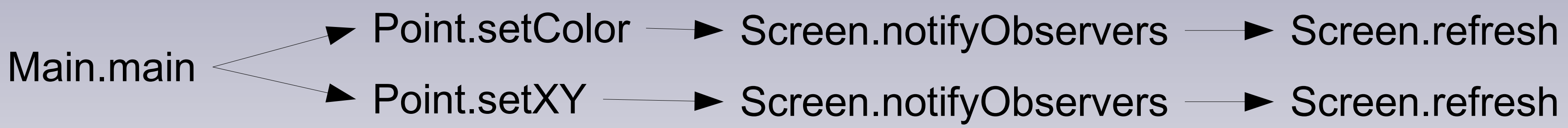
Dynamic-based techniques: Association Rules (3)

- Observer pattern example continued
- Execution traces

Trace "a point changes its color"
Screen.addObserver; Point.addObserver; Point.setColor ; **Screen.notifyObservers; Screen.refresh**

Trace "a point changes its position"
Screen.addObserver; Point.addObserver; Point.setXY ; **Screen.notifyObservers; Screen.refresh**

- Dynamic call-graph



Main.main

Point.setColor → Screen.notifyObservers → Screen.refresh

Point.setXY → Screen.notifyObservers → Screen.refresh

Dynamic-based techniques: Association Rules (3)

- By applying an association rule algorithm to the execution traces the authors were able to discover indicators of crosscutting behavior

Seeds

(1) [Concept] Screen.addObserver ⇒ Point.addObserver [s: 1.0, c: 1.0]
(2) [Concept] Screen.notifyObservers ⇒ Point.notifyObservers [s: 1.0, c: 1.0]

(3) [Cons. Req.] Point.setColor ⇒ Point.notifyObservers [s: 0.5, c: 1.0]
(4) [Cons. Req.] Point.setX ⇒ Point.notifyObservers [s: 0.5, c: 1.0]
(5) [Cons. Req.] Point.notifyObservers ⇒ Screen.refresh [s: 1.0, c: 1.0]
(6) [Cons. Req.] Screen.notifyObservers ⇒ Screen.refresh [s: 1.0, c: 1.0]

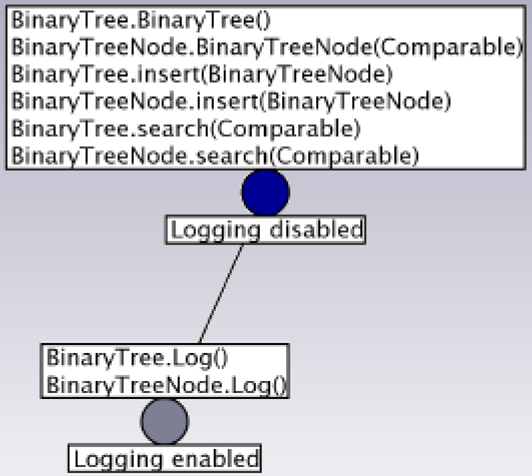
Agenda

- Introduction
- Migration Process
- Aspect Mining Techniques
  - Static-based approaches
  - Dynamic-based approaches
- Concern-Sorts
- Pitfalls on Aspect Mining

Concern-Sorts: Introduction

- **Problem:** there is no consistency and compatibility between aspect mining techniques and their results
  - Most mining techniques rely on heterogeneous descriptions of the crosscutting concerns they aim to identify and the steps to be taken to map their results onto potentially associated concerns

DA + FCA



BinaryTree.BinaryTree()
BinaryTreeNode.BinaryTreeNode(Comparable)
BinaryTree.insert(BinaryTreeNode)
BinaryTreeNode.insert(BinaryTreeNode)
BinaryTree.search(Comparable)
BinaryTreeNode.search(Comparable)

BinaryTree.Log()
BinaryTreeNode.Log()

Fan-in Analysis

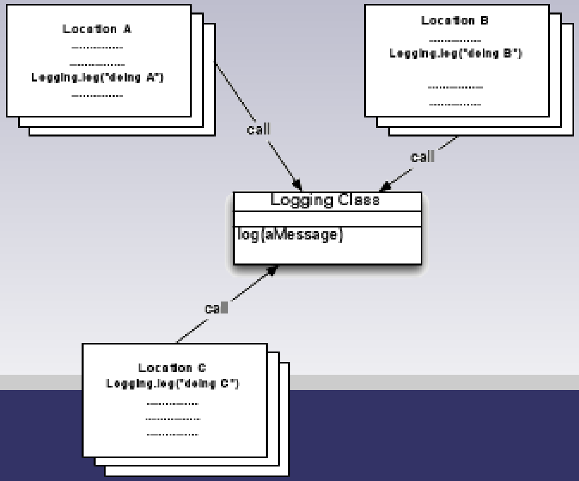
Method	Fan-in	Concern
framework.Figure.willChange()	25	Observer
standard.AbstractFigure.changed()	37	Observer
util.StorableInput.readInt()	22	Persistence

Concern-Sorts: Definition

- In order to overcome this problem Marin et al. [18] have introduced the notion of crosscutting concern sorts
    - Crosscutting concern sorts are atomic descriptions of crosscutting functionality. Each sort has the following characteristics
        1. a generic description of the sort (i.e., the sort's intent)
        2. a specific implementation idiom of the sort's instances in a non-aspect-oriented language (i.e., sort's specific symptom)
        3. an atomic aspect language mechanism to modularize concrete instances of the sort

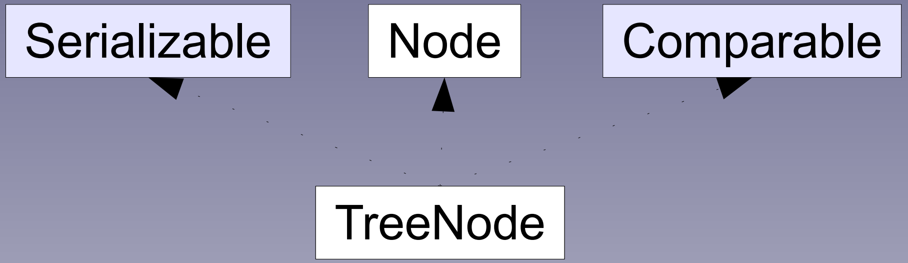
Concern-Sorts: Sort Examples (1)

- **Consistent Behavior sort:** The purpose of this sort is to enforce and ensure that specific functionality is consistently executed by a number of methods
  - *Intent*: The enforced behavior is a precise step in the execution of several methods
  - *OO idiom*: Method invocations
  - *Aspect mechanism*: pointcut + advice
  - *Examples*: Authentication; Notify listeners (Observer pattern); logging

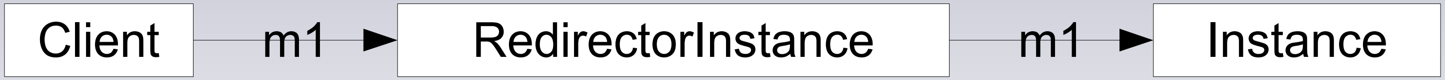


Concern-Sorts: Sort Examples (2)

- **Role superimposition:** class that implements a secondary role or responsibility



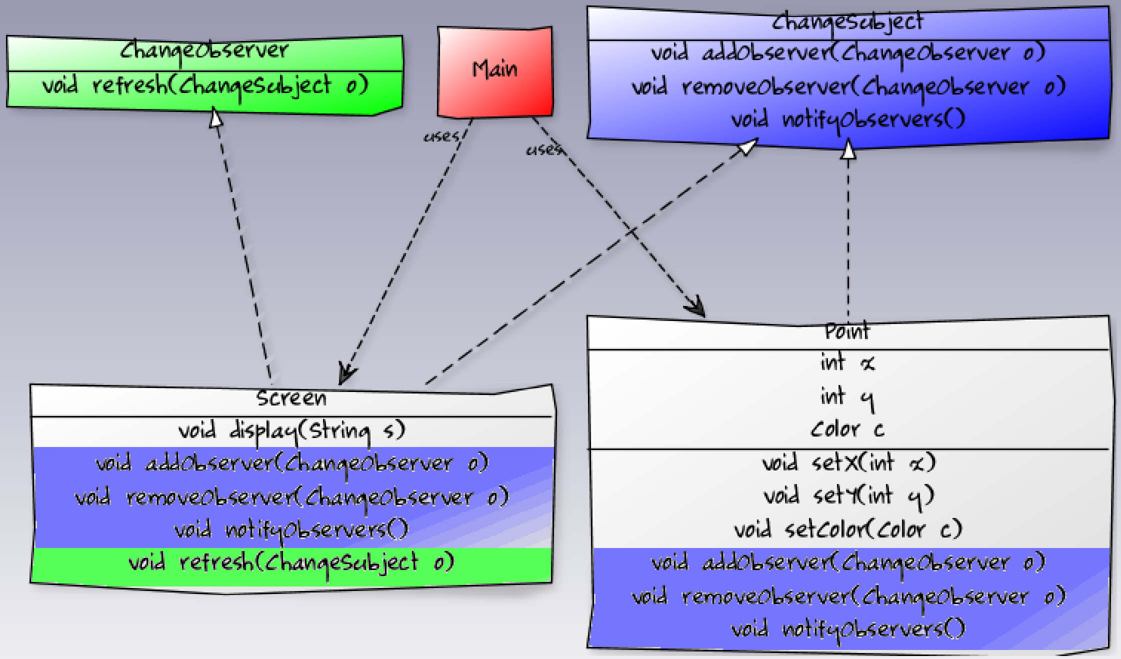
- **Redirection layer:** is an interfacing layer to an object which forward the calls to that object
  - Examples: Decorator and Adapter design patterns



Concern-Sorts: Composing Sorts

- Sorts that compose the Observer crosscutting concern

Observer
RSI(contextElem1, ChangeObserver) + RSI(contextElem2,ChangeSubject) + CB(contextElem3, notifyObservers) + CB(contextElem1, addObserver) + CB(contextElem1, removeObserver);


ChangeObserver
void refresh(ChangeSubject o)
Main
uses
uses
ChangeSubject
void addObserver(ChangeObserver o)
void removeObserver(ChangeObserver o)
void notifyObservers()
Screen
void display(String s)
void addObserver(ChangeObserver o)
void removeObserver(ChangeObserver o)
void notifyObservers()
void refresh(ChangeSubject o)
Point
int x
int y
Color c
void setX(int x)
void setY(int y)
void setColor(Color c)
void addObserver(ChangeObserver o)
void removeObserver(ChangeObserver o)
void notifyObservers()

Concern-Sorts: Adapting AM Techniques

- Retrofitting aspect mining techniques into the crosscutting concern sorts framework

Technique	Search-goal	Presentation	Mapping
DA + FCA	Role Superimposition	Set of methods in a type hierarchy defining the superimposed role, and their implementing crosscut classes	The methods map onto the members of the superimposed type and cut across their implementing classes
Clone detection	Consistent behavior	Set of relations (and statements) grouped by a code fragment that is duplicated in multiple method bodies (and that is refactorable by a method extraction)	The method to extract the cloned fragment maps onto the crosscutting element; the methods containing the cloned code fragment map onto the elements being crosscut
Execution patterns	Consistent behavior	Call relations between a set of methods (i.e. callers) and identical(ly positioned) sequence of other methods	The recurrent sequence of method invocations maps onto the elements crosscutting the callers in the relation
Fan-in Analysis	Consistent behavior	Results are call relations, described by a callee and a set of callers	The method with a high fan-in value (the callee) maps onto the method implementing the crosscutting functionality, and the callers of the method correspond to the crosscut elements

Agenda

- Introduction
- Migration Process
- Aspect Mining Techniques
  - Static-based approaches
  - Dynamic-based approaches
- Concern-Sorts
- Pitfalls on Aspect Mining

Pitfalls on Aspect Mining

- Mens et al. [19] have listed the main problems that they have encountered when using aspect mining techniques
  - Poor precision
  - Poor recall
  - Subjectivity
  - Scalability
  - Empirical validation

Questions & Answers

References (1)

1. Lehman
2. Mens book
3. Kellens survey
4. Marin fan-in
5. Shepherd clustering
6. Moldovan y Serban clustering
7. Ryssell y Demeyer clone code
8. Fowler book
9. Marinescu metrics book
10. Bruntink clone code
11. Shepherd lexixal chain
12. Towré y Mens FCA+identif

References (2)

13. Ball dynamic analysis
14. Chapter nuestro
15. Breu and Krinke
16. Tonella and Ceccato
17. Paper PLATE
18. Marin sort classif
19. Mens pitfalls

Version