

---

# Using FCA to Suggest Refactorings to Correct Design Defects

---

**Naouel Moha, Jihene Rezgui, Yann-Gaël Guéhéneuc,  
Petko Valtchev, and Ghizlane El Boussaidi**

DIRO, University of Montreal, Canada

**Présenté par Ghada Gasmi**

**CLA 2006**

Oct 30 – Sept 1<sup>st</sup>, 2006, Tunis, Tunisie

# Context

---

- Design Patterns are “good” solutions to recurring design problems
- Design Defects (DDs)
  - are “bad” solutions to recurring problems
  - transgress good OO (Object-Oriented) principles
    - Cohesion
    - Coupling
- Why it is important to detect and correct DDs ?
  - DDs lessen the quality of OO architectures and impede their evolution and their maintenance
  - Maintenance is expensive because of DDs [PER 92] : adding, debugging, and evolving of features are difficult
  - A good software architecture without DDs: easier to understand, change, and thus maintain

# Context

## ■ An example of high-level DDs

```
18     if (fNamespacesEnabled) {
19         fNamespacesScope.increaseDepth();
20         if (attrIndex != -1) {
21             int index = List.getFirstAttr(attrIndex);
22             ...
23             ...
24             ...
25             ...
26             ...
27             ...
28             ...
29             ...
30         }
31         int pre ...
32         int elem ... RI;
33         if (pre ... -1) {
```

### ■ Blob

- Large controller class
- Many fields and methods with a low cohesion\*
- Highly coupled\*\* with data stored in associated data classes

\* Cohesion: How closely the methods are related to the instance variables in the class. Measure: LCOM (Lack of Cohesion Metric)

\*\* Coupling: the degree of reliance on services provided by other classes.

→ **Bad impact** : low cohesion and high coupling!

# Overview

---

- Related Work
- Problem Description
- Proposed Approach
- Implementation
- Conclusion

# Related Work

---

- **Huchard and LeBlanc** (ASE, 2000)
    - use FCA to suggest restructurations of class hierarchies
    - to maximise the sharing of data structure and code through fields and methods
  
  - **Arévalo *et al.*** (ICFCA, 2005)
    - applied FCA to identify implicit dependencies among classes in program models
- None of these approaches attempts to suggest refactorings to correct DDs!

# Overview

---

- Related Work
- Problem Description
- Proposed Approach
- Methodology
- Validation
- Conclusion

# Problem Description

---

- What are the problems ?
  - Lack of (semi-)automated techniques and tools
  - Lack of work on high-level DDs
  - Semi-manual detection and correction
  - The detection and correction are time-consuming and error-prone activities

# Overview

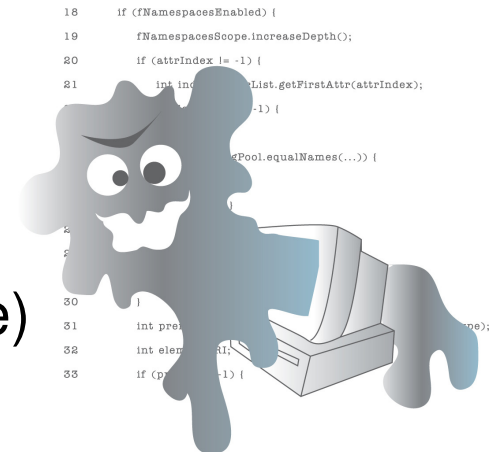
---

- Related Work
- Problem Description
- Proposed Approach
- Implementation
- Conclusion

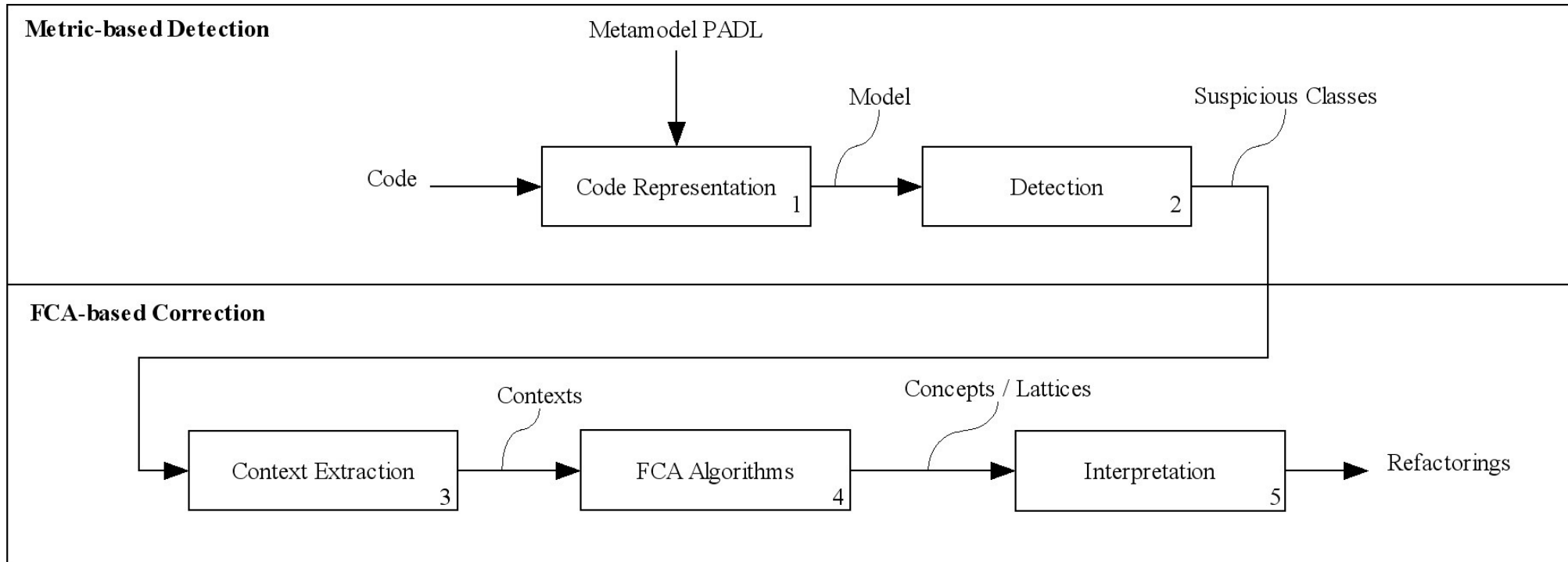
# Proposed Approach

We propose to apply **FCA** on a suitable representation of a program to **suggest appropriate refactorings** for certain **DDs**.

- **Refactorings ?**
    - Change performed on the source code of a program to improve its internal structure without changing its external behaviour [Fowler].
  - We illustrate our approach with the Blob
  - Azureus version 2.3.0.6
    - A peer-to-peer program
    - 143 Blobs for 1,449 classes (191,963 lines of code)
- We show that **FCA** can suggest relevant refactorings to improve the program!



# Proposed Approach



# Encoding Blobs into Formal Contexts 1/2

- To Identify cohesive sets of methods and fields, we define three formal contexts:
  - $K_1$ , individuals are methods of a suspect large class and properties are fields of that class.
    - allows to assess the cohesion of a class because methods sharing the same fields are, by definition, cohesive.
  - $K_2$ , both individuals and properties are methods of the suspect large class.
    - highlights subsets of cohesive methods, because methods invoking the same set of other methods are highly cohesive.
  - $K_3$ , individuals are methods and fields of the large class and properties are the surrounding data classes.
    - allows to assess the coupling between the large class and its data classes.

# Encoding Blobs into Formal Contexts 2/2

	(a0) alien_average	(a1) alien_fv_average	(a2) bad_ip_bloom_filter	(a3) bootstrap_node	(a4) external_address	(a5) last_address_change	(a6) last_alien_count	(a7) last_alien_fv_count	(a8) listeners	(a9) local_contact	(a10) logger	(a11) other_non_routable_total	(a12) other_routable_total	(a13) packet_handler	(a14) reachable	(a15) reachable_accurate	(a16) recent_reports	(a17) request_handler	(a18) request_timeout	(a19) stats	(a20) stats_start_time	(a21) store_timeout	(a22) STATS_INIT_PERIOD	(a23) STATS_PERIOD
(m0) checkAddress()			×																					
(m1) externalAddressChange()						×																		
(m2) getAddressChange()					×																			
(m3) process()				×																				
(m4) sendFindNode()									×	×	×													
(m5) sendFindValue()									×	×	×										×	×		
(m6) sendStore()									×	×	×			×							×	×		
(m7) setRequestHandler()																		×						
(m8) testInstanceIDChange()									×															
(m9) testTransportIDChange()					×				×															
(m10) updateContactStatus()												×	×											
(m11) updateStats()	×	×																						

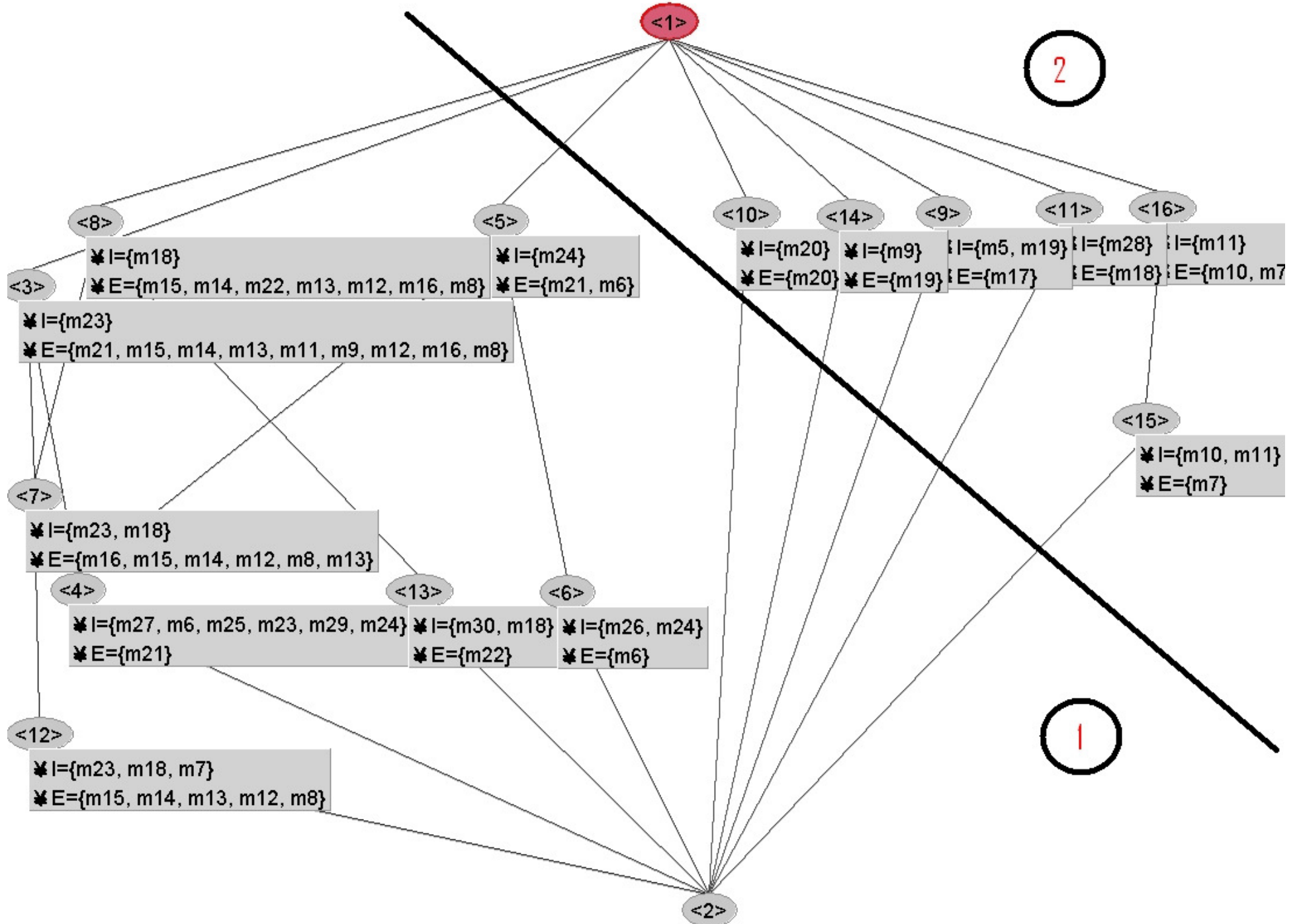
Context *K1* linking methods of the large class to fields of the class.

# Interpretation of Lattice Structure 1/3

---

- We build Lattices from the 3 contexts  $K_1$ ,  $K_2$ ,  $K_3$  to
  - Interpret the inner structure of the large class
  - Suggest refactorings
- We define two interpretations rules:
  - Rule1: collection of cohesive and independent subsets (figure, zone2)
  - Rule2: Detect large cohesive subset (figure, zone1)

# Interpretation of Lattice Structure 2/3



# Interpretation of Lattice Structure 3/3

- Refactorings:
  - Split large class into 2 ways:
    1. Move disjoint and cohesive subsets of methods and/or fields that are related to a data class.
    2. Organise cohesive subsets that are not related to data class in separate classes.
  
  - Azureus, class DHTTransportUDPImpl
    - Before the refactorings: cohesion: 0.542, coupling: 41
    - After the refactorings: cohesion: 0.278, coupling: 39

# Overview

---

- Related Work
- Problem Description
- Proposed Approach
- Implementation
- Conclusion

# Implementation

---

- **PADL** (*Pattern and Abstract-level Description Language*)
  - Metamodel of the tool suite Ptidej
  - To model source code
- **GALICIA**
  - Multi-tool open-source platform for creating, visualizing, and storing lattices
- **Both tools communicate by means of XML files**
  - Ptidej generates contexts in the XML format of Galicia
  - Galicia transforms the contexts into lattices

# Overview

---

- Related Work
- Problem Description
- Proposed Approach
- Implementation
- Conclusion

# Conclusion

---

- DDs transgress good OO principles
  - Low coupling and high cohesion
- Our approach
  - Joint use of metrics and FCA to suggest refactorings to correct DDs
  - Can be generalised to other DDs characterised by a high coupling and a low cohesion
- Future Work
  - Fully automatic correction mechanisms
  - An integrated tool platform to support FCA-based refactorings
  - Refinement of the proposed rules for lattice structure interpretation

# Questions

---



**For any question,  
please contact directly the authors at**  
[mohanaou@iro.umontreal.ca](mailto:mohanaou@iro.umontreal.ca)  
[rezguiji@iro.umontreal.ca](mailto:rezguiji@iro.umontreal.ca)