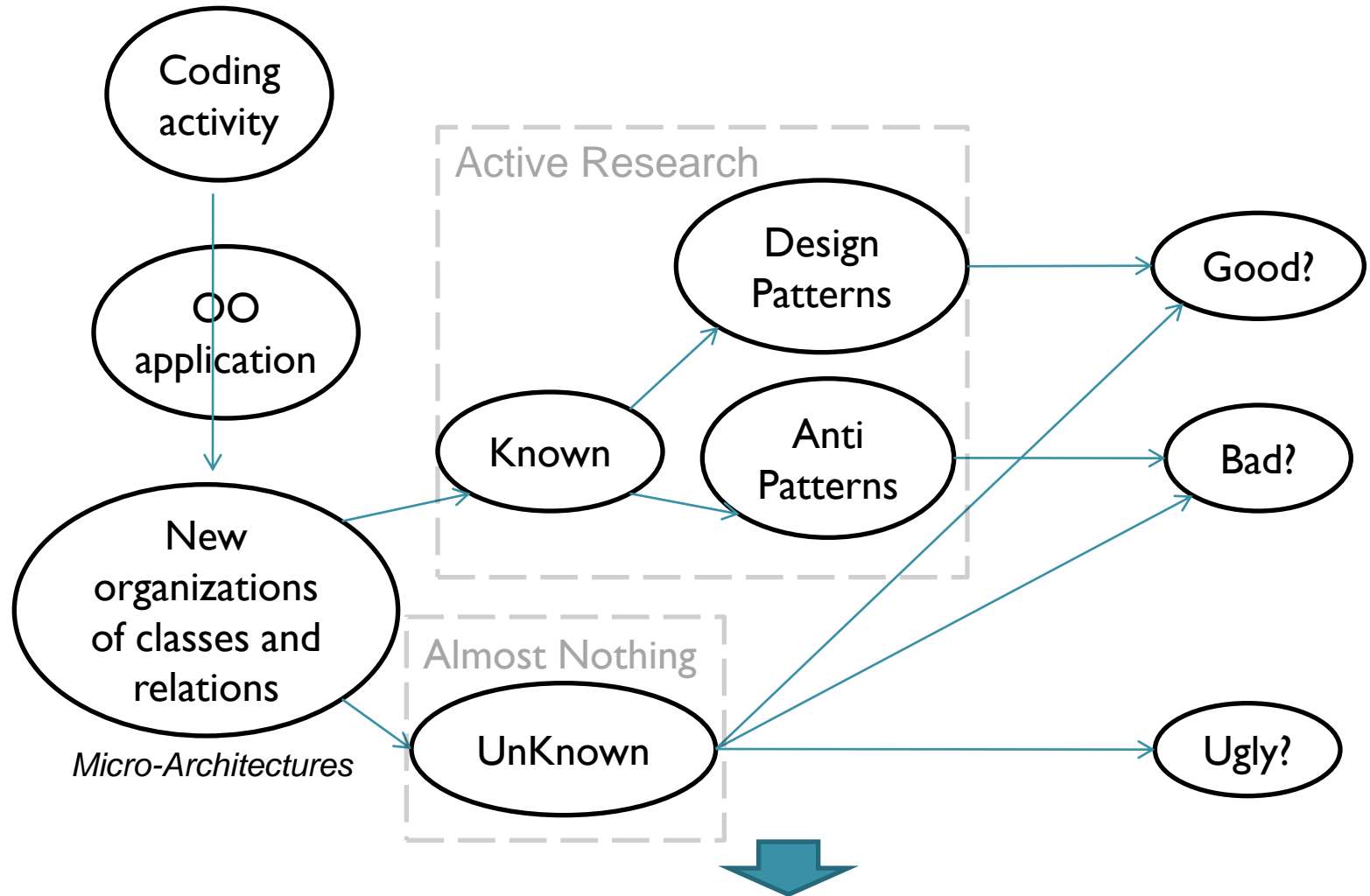


Sub-graph Mining: Identifying Micro-architectures in Evolving Object-oriented Software

Ahmed Belderrar, Segla Kpodjedo, Yann-Gaël Guéhéneuc, Giuliano Antoniol, Philippe Galinier



Context & Motivation



Preliminary study exploring the feasibility and prospects of an approach aimed at building empirical knowledge about organizations of classes.

Related work (I)

In general, the proposed approaches rely on a library of known motifs and use architectural recovery techniques based on sub-graph matching or on motifs properties.

- Constraint Programming to recognize plans [Rich and Waters-90]
- Explanation-based constraint programming [Guéhéneuc and Antoniol-2008]
- Use of queries [Kullbach and Winter-99]
- Generic fuzzy reasoning nets [Niere et al.-2002].
- Graph-transformation techniques [Tsantalis et al.-2006]

Related work (II)

- Similar to our approach: [Tonella and Antoniol-01] in which concept analysis was used to infer domain-specific design patterns.

Main Difference :

“maximal collections of class sets having the same relations between them” VS isomorphic induced subgraphs of fixed size

→ improves on scalability via an efficient sub-graph enumeration

→ eliminates the need of manual inspection of “concept lattice”.

- Inspiration from sub-graph mining algorithms
- Sub-graph discovery problem : discover all recurring sub-graphs, or only the most frequent ones.

Limitations of previous works

(restriction to undirected graphs, or to sub-graphs with limited number of labels on their arcs [Wernicke and Rasche-2006])

→ Our own tool SGFinder

Our approach – Introduction

- Considering a class diagram as a labeled graph, we define a micro-architecture as the connected subgraph induced by a given subset of classes.

Pre-requisite

- We model a class diagram as a labeled graph, with nodes being classes (and interfaces) and arcs representing the relations among classes.
- Arc label \rightarrow type of relation between two classes (e.g., *association*, *aggregation*, or *inheritance*).

Example

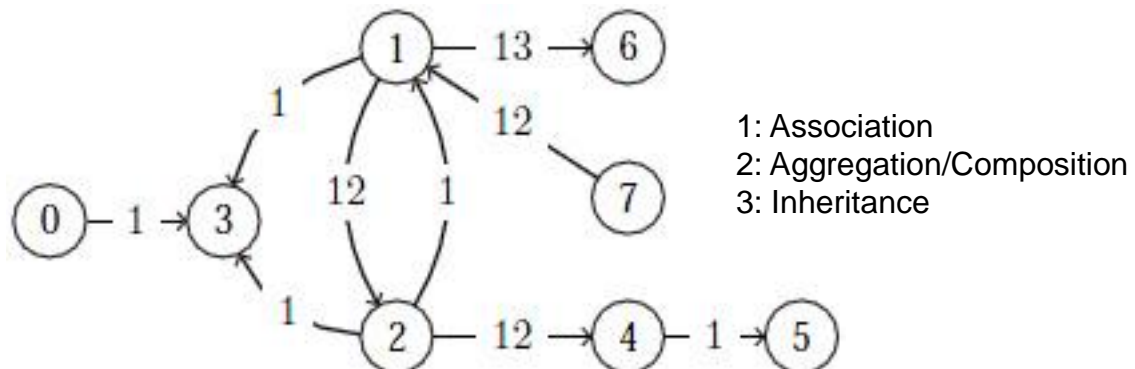


Figure 1 A sub-graph $G(V,E)$ extracted from Rhino (release 1.7R1).

Our approach – Definitions

- **Labeled graph:** given a set of labels L , it is defined by a triplet $G = (V, A, l)$ where V is the vertex set of G , $A \subseteq V \times V$ the arc set, and $l : A \rightarrow L$ the labeling function.
- **Connected Graph:** There is a chain between any two vertices.
- **Isomorphic Graphs:** There exists a perfect one-to-one correspondence between the vertices and edges of those graphs.
- **Induced Subgraph:** G_X induced by X ($X \subseteq V$) is a graph which vertex set is X and which arc set contains all the arcs of G that link two vertices of X .
- **Embedding of a sub-graph H of G :** set X of vertices such that G_X is isomorphic to H .

Example

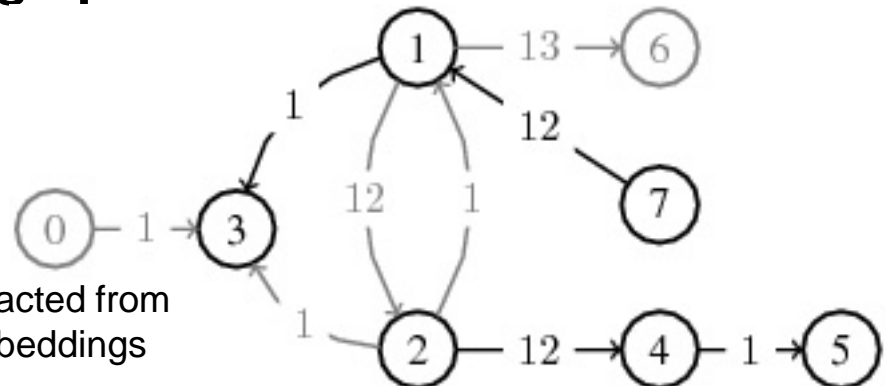


Figure 2 A sub-graph $G=(V,E)$ extracted from Rhino (release 1.7R1) with two embeddings of size three $\{7, 1, 3\}$ and $\{2, 4, 5\}$.

Our approach – Algorithm (I)

- **Inputs**

A graph G and the size k of the sought sub-graphs

- **Output**

The set of induced subgraphs with their embeddings.

- **Principle**

Generate all k -subsets X of V such that G_X is connected.

Generates only a limited number of k -subsets of V

- **Underlying idea**

if G_X is a connected induced sub-graph of G , then X can not contain two vertices x and y such that $\text{dist}(x, y) \geq k$.

Our approach – Algorithm (II)

SGFinder

Algorithm 1 SGFinder() procedure.

```

procedure SGFinder(graph  $G = (V, A, l)$ , integer  $k$ )
2: Choose a vertex  $x \in V$ 
   for  $i:=0..k-1$  do
4:   Compute  $L_i = \{y \in V : dist(x, y) = i\}$ 
   end for
6: CompleteSG( $\{\}, 0, k, (L_0..L_{k-1}), G$ )
   SGFinder( $G_{V-\{x\}}$ ,  $k$ )
8: end procedure

```

Build k disjoint subsets of vertices, denoted by $L_0..L_{k-1}$ (line 3-5), where each L_i contains the vertices which distance to x equals i ($L_0 = \{x\}$)

Algorithm 2 CompleteSG() procedure.

```

procedure CompleteSG( $X, j, k, (L_0..L_{k-1}), G$ )
if  $|X| = k$  then
3:   Build graph  $H := G_X$ 
   if Connected( $H$ ) then
   if  $c \notin M$  then
6:     Insert ( $c, X$ ) into  $S$ 
   end if
   Insert  $c$  into  $M$ 
9:   end if
else
   for all  $Y \subseteq L_i$  such that  $|Y| \geq 1$  and  $|X| + |Y| \leq k$  do
12:    CompleteSG( $X \cup Y, j+1, k, (L_0..L_{k-1}), G$ )
   end for
end if
15: end procedure

```

To build X (line 6, developed in the CompleteSG() procedure), we choose the unique vertex present in L_0 (namely x), plus one or more vertices chosen in L_1 , plus one or more vertices chosen in L_2 , and so on until we reach a total number k of vertices.

Our approach – Algorithm (III)

- Illustration

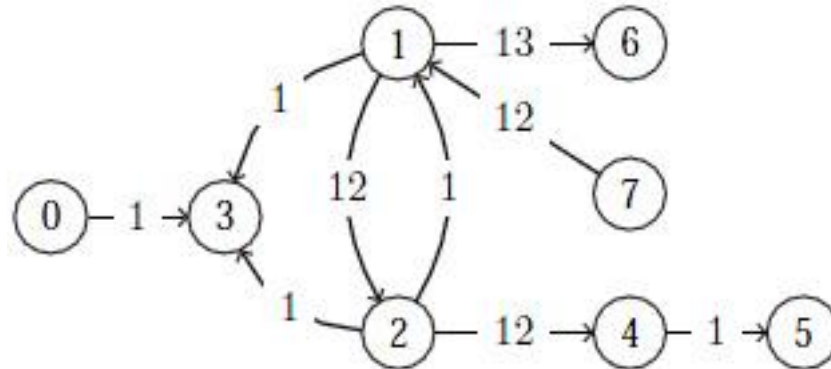


Figure 1 A sub-graph $G(V,E)$ extracted from Rhino (release 1.7R1).

- Let us choose $k = 4$,
 - And let us assume that the first chosen vertex is $x = 0$.
 1. We have $L_0 = \{0\}$, $L_1 = \{3\}$, $L_2 = \{1, 2\}$, and $L_3 = \{4, 6, 7\}$.
 2. There are now seven possibilities for X :
 - $\{0, 3, 1, 2\}$, $\{0, 3, 1, 4\}$, $\{0, 3, 1, 6\}$, $\{0, 3, 1, 7\}$,
 - $\{0, 3, 2, 4\}$, $\{0, 3, 2, 6\}$, $\{0, 3, 2, 7\}$.
 3. Some of the induced sub-graphs are not connected, e.g., $G\{0,3,1,4\}$ and are discarded.

Case Study

- **Goal**

Identify in OO systems, micro-architectures with desirable or harmful properties.

- **Quality focus**

Verify applicability of SGFinder to small and medium size OO programs in order to detect micro-architectures that are stable or fault-prone.

- **Perspective**

Researchers, developers, and managers, who want to get information about undocumented micro-architectures found in OO systems and possible related drawbacks.

- **Context**

Two open-source systems: the Rhino JavaScript/ECMAScript interpreter, and the ArgoUML CASE tool.

Case Study - Objects

- Rhino: JavaScript/ECMAScript interpreter and compiler
 - Defect data extracted from [Eaddy et al-2008] study
 - Change data mined from the Rhino CVS logs.
- ArgoUML: UML CASE tool written in Java.
 - Defect data from the ArgoUML customized Bugzilla repository
 - Change data from the SVN logs.

Systems	Releases (Number Thereof)	Number of			
		Classes	LOCs	Defects	Changes
Rhino	1.4R3-1.6R1 (8)	99-194	21K-75K	12-114	217-1476
ArgoUML	0.10-0.17.5 (8)	876-1243	86K-123K	102-664	541-5048

Table I

SUMMARY OF THE OBJECT SYSTEMS

Class diagrams recovered using the Ptidej tool suite and its PADL meta-model.

Size of micro-architectures of interest: 3, 4 and 5 nodes.

Case Study – Research Questions

- RQ1 – Does SGFinder scale up to medium size programs and what kind of microarchitectures are found in OO systems?
- RQ2 – Are there micro-architectures particularly fault-prone or fault-free?
- RQ3 – Are there micro-architectures particularly stable or change-prone?

Case Study – Analysis Method (I)

- Characterizing a micro-architecture mAi

Connectivity

- *Total number of relations ($nbRel$),*
- *Number of associations ($nbAssoc$),*
- *Number of aggregations or compositions ($nbAggr$),*
- *Number of inheritances ($nbInher$),*
- *Number of "cyclic relations" ($nbCycl$).*

Presence and repartition

- *Number of releases in which mAi can be found ($nbReleases$)*
- *Number of embeddings of mAi in a given release ($nbEmbeddings$)*
- *Number of zones ($nbZones$).*
 - an embedding is counted only when it does not share any common edge with a previously found embedding
- *Number of classes appearing at least once in mAi ($nbClasses$)*

Case Study – Analysis Method (II)

- Answering the Research Questions

RQ1

Execution times and Characterization of micro-architectures

RQ2 and RQ3

1. For each mA, we compute the percentage of its classes that are fault-prone (i.e., with one or more documented faults) or changed between two subsequent releases.
2. Rank micro-architectures from the fault(change)-prone to the fault(change)-free using the precision measure of step 1.
3. Inspection of the top 10% and bottom 10% of ranked microarchitectures → hints of what makes those micro-architectures outstanding.

Use of descriptive statistics : the five-number summary statistic (Min, Quartile I, Median, Quartile3, Max).

Results – RQ1.1 *SGFinder Applicability*

Computation times:

From a few seconds to ...two days and half.

3-nodes micro-architectures → 20 seconds at most

4-nodes micro-architectures → less than 30 minutes

5-nodes ...another story

Rhino max <8 min

Argo max > 60 h to retrieve the

82,877 different 5-nodes mAs in ArgoUML 0.17.5
and their 13,741,073,588 embeddings.

Depend mostly on the edge density of the considered class diagrams.
More specifically, the single most time-costly factor is the presence of highly connected nodes which lead to a near-combinatorial explosion of the number of embeddings.

Results – RQ1.2 *Micro-architectures' Description*

Number of different micro-architectures.

- 3 basic relations (association 1, aggregation 2 and inheritance 3)
- 4 derived mixed cases (12,13,23,123)
- No relation

 8 possible connections between two nodes

For $k \times k$ pairs of nodes (including loops), if we do not take into account symmetry

At most $8^{(k \times k - (k-1))} \times 7^{k-1}$ connected subgraphs of k nodes.

For $k=5$, in theory, possibly 22×10^{21} different micro-architectures.

In practice, the union of five-nodes micro-architectures from Rhino and Argo contains only about 32×10^4 different micro-architectures.



Suggests, unsurprisingly, that the set of micro-architectures is restricted to specific subgraphs

Results – RQ1.2 Micro-architectures' Description

Characterization

	three-nodes (373)	four-nodes (9203)	five-nodes (190061)
nbRel	2,4,5,6,11	3,6,7,9,20	4,8,10,11,27
nbAssoc	0,2,3,4,6	0,4,5,6,12	0,6,7,9,18
nbAggr	0,0,1,2,4	0,1,1,2,7	0,1,1,2,9
nbInher	0,0,1,1,3	0,0,1,2,5	0,0,1,2,6
nbCycl	0,0,1,1,3	0,0,1,2,6	0,0,1,2,8
nbZones	1,1,1,2,68	1,1,1,1,33	1,1,1,1,20
nbClasses	3,3,5,10,140	4,4,5,9,147	5,5,6,10,156
nbReleases	1,2,5,8,8	1,1,3,6,8	1,1,2,4,8

Table II
MICRO-ARCHITECTURES IN RHINO; EACH LINE REPORTS THE FIVE-NUMBER SUMMARY: MIN, Q1, MEDIAN, Q3, MAX

	three-nodes (349)	four-nodes (8224)	five-nodes (180295)
nbRel	2,4,5,6,10	3,6,7,8,17	4,8,9,10,23
nbAssoc	0,2,3,4,6	0,3,4,6,11	0,5,6,8,15
nbAggr	0,0,1,2,4	0,1,1,2,6	0,1,2,2,8
nbInher	0,0,1,1,3	0,0,1,1,5	0,0,1,2,7
nbCycl	0,0,1,1,3	0,0,1,1,5	0,0,1,2,7
nbZones	1,1,1,3,583	1,1,1,2,315	1,1,1,1,171
nbClasses	3,3,6,19,944	4,4,7,16,940	5,5,8,19,944
nbReleases	1,4,7,8,8	1,2,4,7,8	1,1,3,5,8

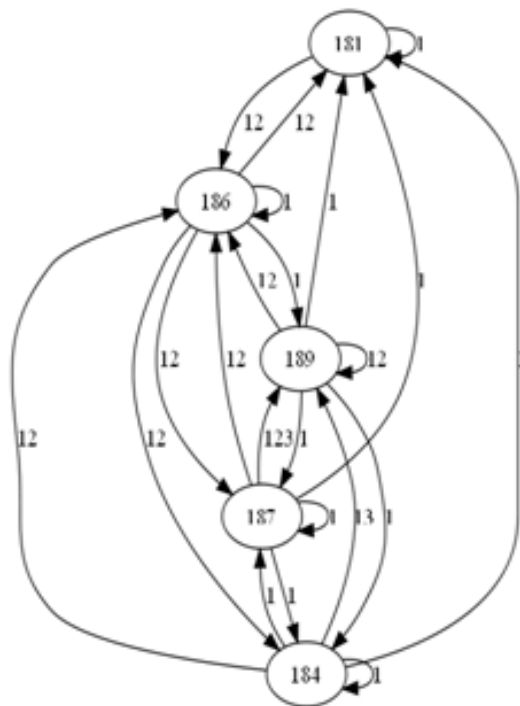
Table III
MICRO-ARCHITECTURES IN ARGO; EACH LINE REPORTS THE FIVE-NUMBER SUMMARY: MIN, Q1, MEDIAN, Q3, MAX

	3-nodes (250)	4-nodes (3993)	5-nodes (52862)
nbRel	2,4,4,5,8	3,5,6,7,11	4,7,8,9,15
nbAssoc	0,2,3,4,6	0,3,4,5,10	0,5,6,7,14
nbAggr	0,0,1,2,4	0,0,1,2,4	0,0,1,2,6
nbInher	0,0,1,1,3	0,0,1,1,4	0,0,1,1,6
nbCycl	0,0,0,1,3	0,0,1,1,4	0,0,1,1,5
nbZones	1,1,2,5,326	1,1,1,2,174	1,1,1,2,95
nbClasses	3,5,8,26,542	4,6,11,24,544	5,8,14,31,540
nbReleases	1,5,7,8,8	1,3,5,7,8	1,3,4,6,8

Table IV
MICRO-ARCHITECTURES IN BOTH RHINO AND ARGO; EACH LINE REPORTS THE FIVE-NUMBER SUMMARY: MIN, Q1, MEDIAN, Q3 AND MAX

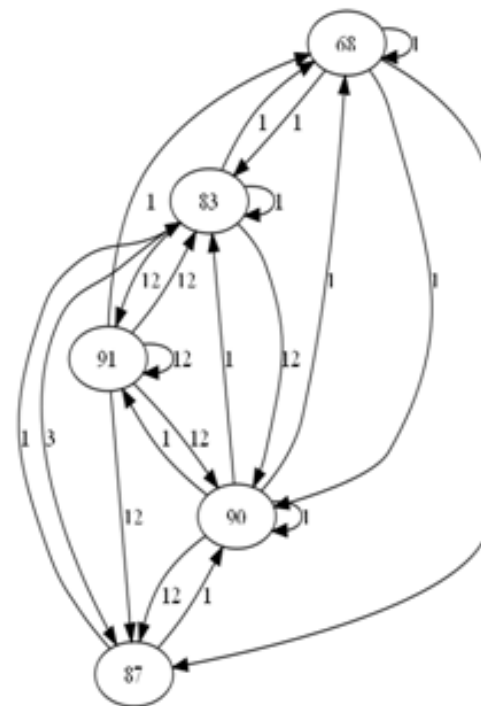
Averages over the two systems

Results – RQ1.2 *Micro-architectures' Description*



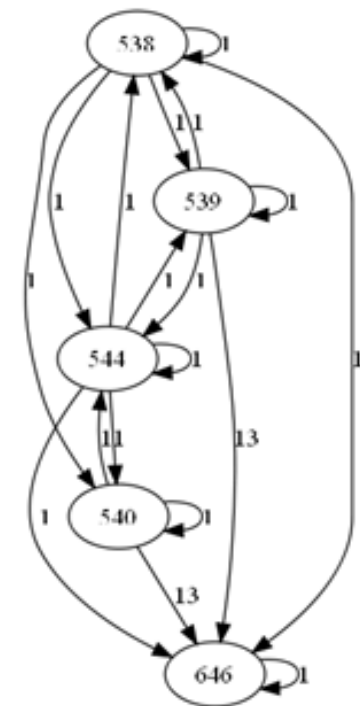
181,org.mozilla.javascript.xmlimpl.Namespace
 184,org.mozilla.javascript.xmlimpl.XML
 186,org.mozilla.javascript.xmlimpl.XMLLibImpl
 187,org.mozilla.javascript.xmlimpl.XMLList
 189,org.mozilla.javascript.xmlimpl.XMLObjectImpl

(a) Rhino1.6R1



68,org.argouml.cognitive.critics.Critic
 83,org.argouml.cognitive.Designer
 87,org.argouml.cognitive.Poster
 90,org.argouml.cognitive.ToDoItem
 91,org.argouml.cognitive.ToDoList

(b) Argo0.14-0.17.5



..sequence.ui.Fig* (Argo)
 ..javascript.Script*(Rhino)

(c) Rhino1.4R3
Argo0.10-0.14

Illustration - Most Connected 5-nodes micro-architectures

Results – RQ2 Fault proneness

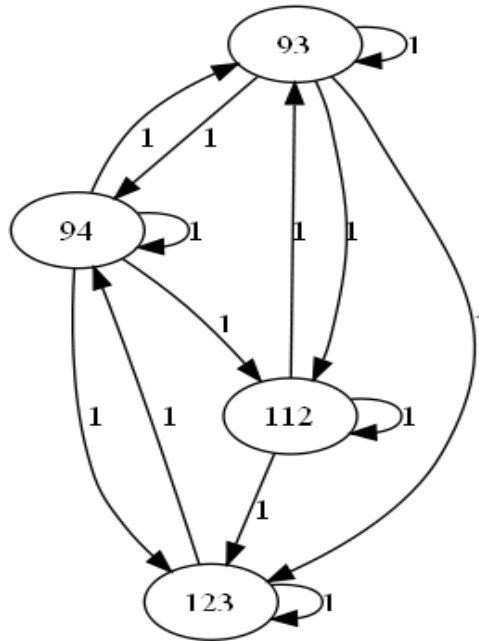


Figure 4. Example of a particularly faulty (precision=81) 4-nodes micro-architecture

(almost) every class “talks” to every other class. In fact, when we retrieve the set of micro-architectures with three cyclic relations and where every class is connected to every other class, we obtain a 5-number summary (28,44,56,67,81) with values mostly higher to those from the top 10% most faulty microarchitectures (42,45,48,54,83).

	three-nodes (250)	four-nodes (3994)	five-nodes (52862)
Precision			
—top10%	41,44,47,52,67	42,45,48,54,83	41,43,46,52,84
—bot10%	0, 5, 9,11,13	0, 8,10,13,14	0, 2, 5, 7,22
F1			
—top10%	5,8,13,17,30	2,10,12,15,40	4,11,14,18,54
—bot10%	0,1, 1, 2, 5	0, 1, 3, 5,16	0, 2, 5, 7,22
nbRel			
—top10%	3,4,6,6,8	4,7,8,8,10	4,8,9,10,14
—bot10%	2,4,4,5,6	3,6,6,7,09	4,7,8,09,13
nbAssoc			
—top10%	2,3,4,5,6	2,5,6,7,10	2,7,8,9,14
—bot10%	0,2,2,3,4	0,3,4,5,08	0,4,5,6,11
nbAggr			
—top10%	0,1,1,2,4	4,7,8,8,10	4,8,9,10,14
—bot10%	0,1,1,2,2	3,6,6,7,09	4,7,8,09,13
nbInher			
—top10%	0,0,0,1,2	0,0,0,1,3	0,0,0,1,4
—bot10%	0,1,1,2,2	0,0,1,1,3	0,0,1,2,5
nbCycl			
—top10%	0,1,1,2,3	0,1,1,2,4	0,1,2,2,5
—bot10%	0,0,0,1,1	0,0,0,1,4	0,0,0,1,5
nbZones			
—top10%	1,1,2,2,9	1,1,1,1,6	1,1,1,1, 7
—bot10%	1,1,1,2,5	1,1,1,1,7	1,1,1,1,14
nbClasses			
—top10%	3,4,5,7,33	4,5,6, 9, 59	1,1,1,1, 7
—bot10%	3,4,5,7,18	4,5,7,12,133	1,1,1,1,14
nbReleases			
—top10%	1,4,5,6,8	1,3,4,5,8	1,2,3,5,8
—bot10%	1,3,4,5,8	1,2,3,5,8	1,2,3,4,8

Table V
MOST AND LEAST FAULTY MICRO-ARCHITECTURES PRESENT
IN BOTH RHINO AND ARGO; EACH LINE REPORTS THE
FIVE-NUMBER SUMMARY: MIN, Q1, MEDIAN, Q3 AND MAX

Results – RQ3 Change proneness

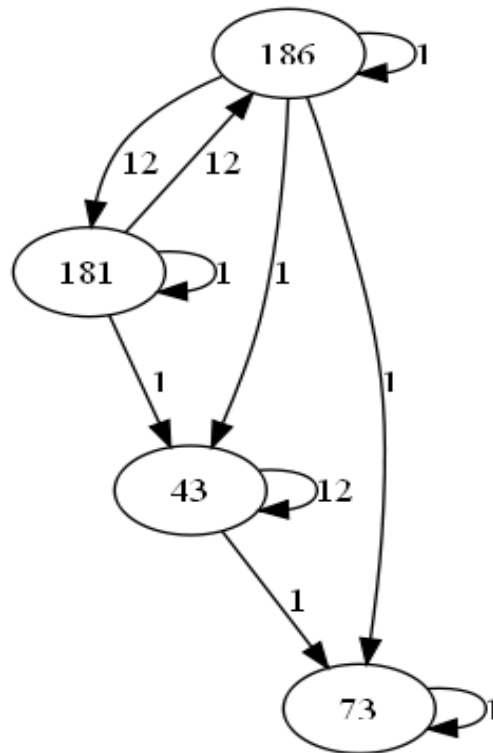


Figure 5. Example of a particularly stable (precision=30) 4-nodes micro-architecture

presents a "cascade"- like pattern and we were able to retrieve many other similar organizations with mostly low changeability.

	three-nodes (250)	four-nodes (3994)	five-nodes (52862)
Precision			
—top10%	83,86,88,91,97	85,88,90,93,100	84,86,89,92,100
—bot10%	20,35,38,39,44	17,38,43,48, 50	10,42,47,50, 53
F1			
—top10%	3,5,6,9,27	4,5,6,8,22	5,8,9,12,37
—bot10%	0,1,1,2,13	0,2,3,5,33	0,3,6,11,55
nbRel			
—top10%	3,5,6,6,7	4,6,7,8,10	4,8,9,10,15
—bot10%	2,4,4,6,7	3,5,6,7,10	4,7,8, 9,13
nbAssoc			
—top10%	1,3,4,5,6	2,5,6,7,10	1,6,8,9,13
—bot10%	0,1,2,3,4	0,3,4,5,08	1,4,5,7,11
nbAggr			
—top10%	0,0,0,1,3	0,0,1,2,4	0,0,1,1,5
—bot10%	0,1,2,2,3	0,1,2,2,4	0,1,2,2,6
nbInher			
—top10%	0,0,1,1,2	0,0,0,1,4	0,0,0,1,5
—bot10%	0,1,1,2,2	0,0,1,1,3	0,0,1,1,4
nbCycl			
—top10%	0,1,1,2,3	0,1,1,2,4	0,0,1,2,5
—bot10%	0,0,1,1,2	0,1,1,2,3	0,0,1,1,4
nbZones			
—top10%	1,1,2,2,5	1,1,1,1, 5	1,1,1,1,10
—bot10%	1,1,1,2,5	1,1,1,1,11	1,1,1,1,10
nbClasses			
—top10%	3,4,5,7,26	4,5,6, 8, 90	5,6, 8,11,158
—bot10%	3,4,5,7,69	4,5,8,14,321	5,8,13,29,404
nbReleases			
—top10%	1,4,5,7,8	1,2,4,5,8	1,2,3,4,8
—bot10%	2,4,5,7,7	1,3,4,5,8	1,2,3,5,8

Table VI

MOST AND LEAST CHANGED MICRO-ARCHITECTURES PRESENT IN BOTH RHINO AND ARGO; EACH LINE REPORTS THE FIVE-NUMBER SUMMARY: MIN, Q1, MEDIAN, Q3 AND MAX

Threats to Validity

Threats to *conclusion validity* concern the *relationship* between the treatment and the outcome.

We do not claim any causal relation between the microarchitectures and unwanted or desired features.

Threats to *external validity* concern the *possibility* of generalizing our results.

The study is limited to 2 systems: Rhino and ArgoUML.

But threats are mitigated given that the two systems
correspond to different domains and applications,
have different sizes,
and are developed by different teams.

Conclusion

- I presented SGFinder, an algorithm and a tool to support micro-architecture discovery based on a reformulation as a sub-graph mining problem.
- SGFinder uses an effective enumeration technique that allows us to infer instances of micro-architectures and to study micro-architecture properties such as stability or fault proneness.
- We used SGFinder on 8 releases of 2 well known Java applications: Rhino and ArgoUML.
- We provided insight about the kind of micro-architectures (of 3, 4 or 5 nodes) found in OO systems.
- We characterize the most and least faulty/changed micro-architectures and report some of the most interesting micro-architectures with respect to their connectivity and frequency.

Future Ongoing Work

Despite the encouraging results, more work is needed to

- (i) further optimize the proposed algorithm and gain in scalability, and/or investigate the partitioning of large OO diagrams into subsystems
- (ii) define heuristics and rules able to classify micro-architectures discovered in a system under development,
- (iii) go beyond the micro-architectures and analyze, similarly to design patterns, the roles played by participant classes, and
- (iv) provide qualitative analysis and validation of the findings. For generalization purposes, our plan for future work also include replication of the study on a larger system.

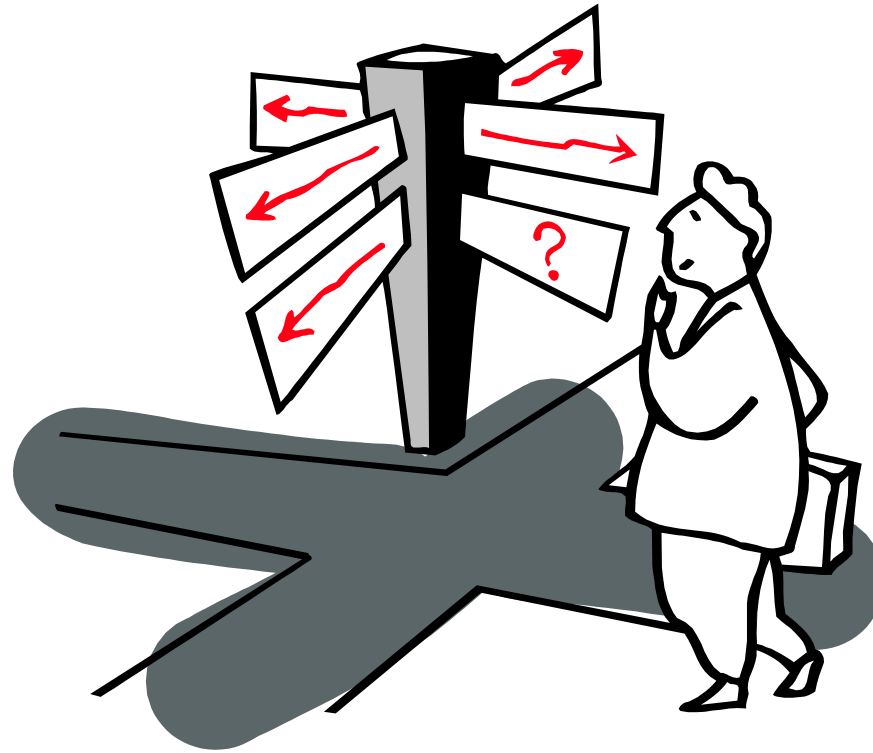
References

- [Guéhéneuc and Antoniol-2008] Y.-G. Guéhéneuc and G. Antoniol, “DeMIMA: A multi-layered framework for design pattern identification,” *Transactions on Software Engineering (TSE)*, vol. 34, no. 5, pp. 667–684, September 2008.
- [Tonella and Antoniol-01] P. Tonella and G. Antoniol, “Inference of object oriented design patterns,” *Journal of Software Maintenance - Research and Practice*, vol. 13, no. 5, pp. 309–330, September-October 2001.
- [Rich and Waters-90] C. Rich and R. C. Waters, *The Programmer’s Apprentice*, 1st ed. ACM Press Frontier Series and Addison-Wesley, January 1990.
- [Kullbach and Winter-99] B. Kullbach and A. Winter, “Querying as an enabling technology in software reengineering,” in *Proceedings of the 3rd Conference on Software Maintenance and Reengineering*, P. Nesi and C. Verhoef, Eds. IEEE Computer Society Press, March 1999, pp. 42–50.
- [Niere et al.-2002] J. Niere, W. Schäfer, J. P. Wadsack, L. Wendehals, and J. Welsh, “Towards pattern-based design recovery,” in *Proceedings of the 24th International Conference on Software Engineering*, M. Young and J. Magee, Eds. ACM Press, May 2002, pp. 338–348.

References

- [Tsantalis et al.-2006] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, and S. Halkidis, “Design pattern detection using similarity scoring,” *Transactions on Software Engineering*, vol. 32, no. 11, November 2006.
- [Wernicke and Rasche-2006] S. Wernicke and F. Rasche, “A tool for fast network motif detection,” *Bioinformatics*, vol. 22, pp. 1152–1153, 2006.
- [Eaddy et al.-2008] M. Eaddy, T. Zimmermann, K. D. Sherwood, V. Garg, G. C. Murphy, N. Nagappan, and A. V. Aho, “Do crosscutting concerns cause defects?” *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 497–515, 2008.

Thanks for the attention



Questions?