Densification of cyclic dependencies among classes in OO software systems

Miloš Savić and Mirjana Ivanović

Department of Mathematics and Informatics
Faculty of Sciences
University of Novi Sad

Outline

- Introduction
- Related work and motivation
- Methodology
- Experiments and results
- Conclusions and future work

Outline

- **Introduction**
- Related work and motivation
- Methodology
- Experiments and results
- Conclusions and future work

Introduction

- ***A*** and ***B*** are cyclic (mutually) dependent iff ***A*** directly or indirectly depends on ***B***, and ***B*** directly or indirectly depends on ***A***
- Parnas, 1978: two mutually dependent modules cannot be tested until both modules are finished and working
  - Long cycles: **nothing works until everything works**
- Booch, 1995: well structured OO systems have clearly defined, hierarchical layers
  - Long cycles can cause mutually dependend (non-hierarchical) layers
- Fowler, 2001: structural cycles can cause **endless cycles of change propagation**
- Cyclic dependencies are caused by **internal reuse**, **but prevent efficient external reuse and program comprehension**

Cyclic dependencies in OO software systems

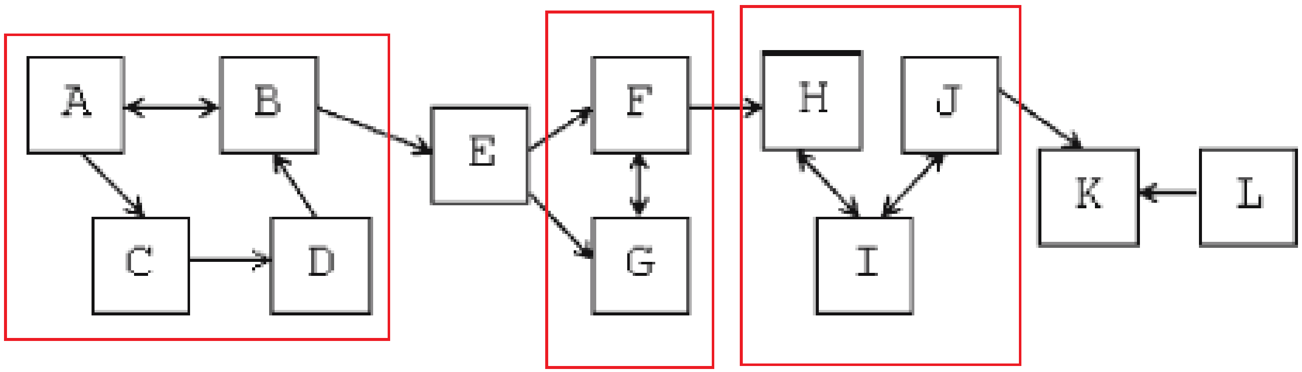
- **Cyclic dependencies among methods**
  - A calls B and B calls A
  - A calls B, B calls C, ..., Y calls Z, Z calls A
  - Not bad if the code is generated (e.g. generated parsers)

- **Cyclic dependencies among classes**
  - Caused by cyclic dependencies among methods or mutual internal class agregation/reuse
  - Not bad if they are short (understandable and maintainable) and natural (e.g. classes representing nodes and links)

- **Cyclic dependencies among packages**
  - Caused by cyclic dependencies among classes

Cyclic dependencies in OO software systems

- **Cyclic dependencies are easy to detect, but hard to remove**
  - Two classes are mutually dependent if they belong to the same **strongly connected component** in the **class collaboration network**



  - Computation of **minimum edge feedback set** is NP-complete
  - Intrinsic interdependency between the real world objects the classes model

Outline

- Introduction
- **Related work and motivation**
- Methodology
- Experiments and results
- Conclusions and future work

Related work

- Research related to cyclic dependencies in software systems is mostly focused on breaking cycles during integration testing.
  - Integration testing order heavily depends on topological sorting of class collaboration networks

- Just a few empirical studies investigating whether the principle *"avoid cycle dependencies"* is being followed and to what extent.
  - Melton and Tempero, 2007, An empirical study of cycles among classes in Java
  - Laval et al, 2012, Efficient retrieval and ranking of undesired package cycles in large software systems
  - Oyetoyan et al., 2013, A study of cyclic dependencies on defect profiles of software components

Related work

- **Melton and Tempero, 2007 (78 software systems)**
  - 45% systems have a cycle involving at least 100 classes, 10% systems have a cycle involving at least 1000 classes

- **Laval et al., 2012 (4 software systems)**
  - Large strongly connected components in package collaboration networks
  - Metrics of cycle desirability

- **Oyetoyan et al., 2013 (6 software systems)**
  - Classes belonging to strongly connected components tend to be more defective than classes not involved in cyclic dependencies

Motivation

- Previously mentioned empirical studies are focused on size of SCCs, not on their structural characteristics

- Our empirical study is focused on:
  I. Complexity of strongly connected components
  II. Mining characteristics of classes involved in strongly connected components

Outline

- Introduction
- Related work and motivation
- **Methodology**
- Experiments and results
- Conclusions and future work

Methodology

- **Complexity of strongly connected components**
  - Two SCCs of the same size can have different complexity
  - The less dense (cohesive) one is less complex to understand, refactor or maintain
  - Average intra-SCC degree as a measure of cohesiveness of SCC

- **Mining structural characteristics of classes involved in cyclic dependencies**
  - Classes are characterized by a rich metric vector
    - Metrics of internal complexity (LOC, cyclomatic complexity)
    - Metrics of design complexity (coupling and inheritance metrics), domain-independent metrics of centrality
  - Comparison of set of nodes based on the Mann-Whitney U test and probabilities of superiority

Outline

- Introduction
- Related work and motivation
- Methodology
- **Experiments and results**
- Conclusions

Experimental dataset

- 5 open-source widely used Java software systems
- Class collaboration networks extracted using SNEIPL (SSQSA back-end)

Experimental dataset of class collaboration networks. N is the number of nodes, L is the number of links.

Software system	**Version**	LOC	N	L
Tomcat	7.0.29	329924	1494	6841
Lucene	3.6.0	111763	789	3544
Ant	1.9.2	219094	1175	5521
Xerces	2.11.0	216902	876	4775
JFreeChart	1.0.17	226623	624	3218

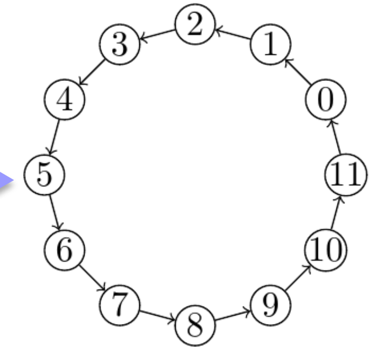
Basic characteristics of SCCs

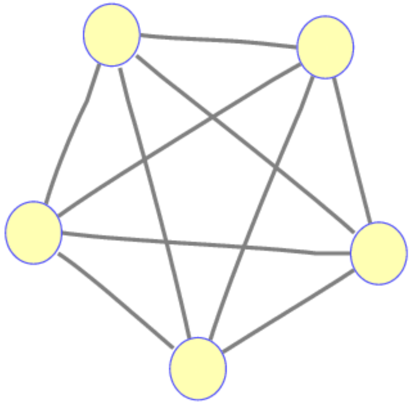
- We used Tarjan's algorithm to identify SCCs
- Existence of large SCCs
- Link reciprocity is small, but higher than expected by random chance
  - The Erdos-Renyi model of random graphs as the null model
- Path reciprocity is significantly higher than link reciprocity for each examined system → **cyclic dependencies are mostly indirect**

Software system	$\#SCC$	$LSCC$ [%]	$N(SCC)$ [%]	R	R_n	R^p
Tomcat	56	12.72	35.74	0.078	0.075	0.179
Lucene	40	17.87	35.23	0.080	0.075	0.162
Ant	27	24.34	35.06	0.046	0.042	0.237
Xerces	32	13.81	32.76	0.078	0.072	0.118
JFreeChart	19	7.05	17.63	0.032	0.024	0.048

Densification of SCCs (I)

- Average intra-SCC degree, A(S) = L(S) / N(S)
  - 1 ≤ A(S) ≤ N(S) - 1
  - A(S) = 1 → S is a pure circle
  - A(S) = N(S) – 1 → S is a clique





- **SCCs densify with size**

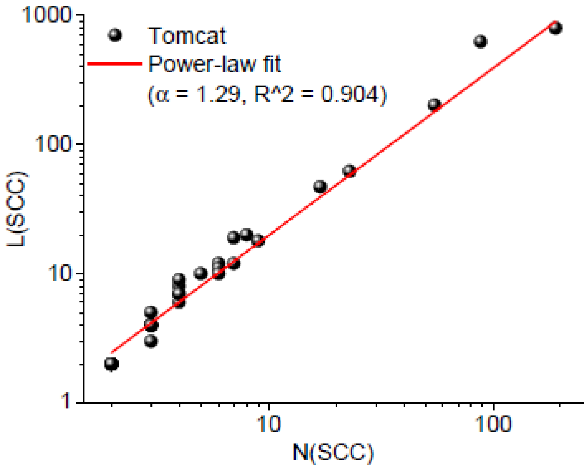
Software system	$\rho(N(S), \frac{L(S)}{N(S)})$
Tomcat	0.975
Lucene	0.965
Ant	0.982
Xerces	0.977
JFreeChart	0.825

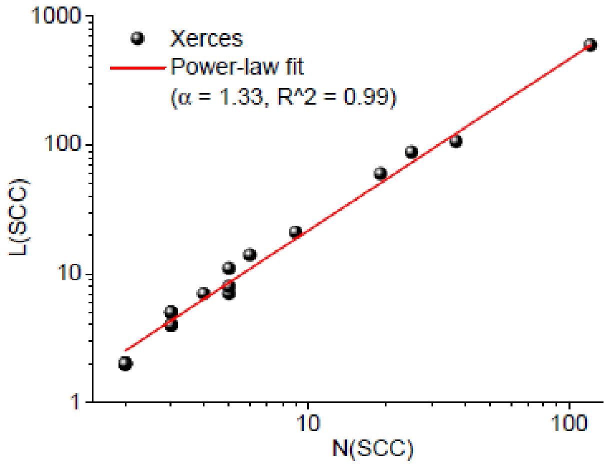
- Densification of SCCs indicates that the number of links in a SCC grows **super-linearly** with the number of nodes.

Densification of SCCs (II)

- Power law is a simple model of super- or sub-linear growth frequently observed in nature, society and engineered systems.
  - $Y \sim X^{\alpha}$ (straight lines on log-log plots)
  - In our case smaller α implies smaller growth rate

Software system	α
Tomcat	1.29
Lucene	1.27
Ant	1.27
Xerces	1.33
JFreeChart	1.32





α can be used as an indicator of software quality

Ideal α = 1 → all SCCs are (nearly) pure circles

Worst α = 2 → all SCCs are (nearly) cliques

Smaller α means better quality

Characteristics of nodes involved in cyclic dependencies

- For a system, we divide classes into two categories
  - C – classes involved in cyclic dependencies (belong to SCCs)
  - N – classes not involved in cyclic dependencies

- Does classes in C tend to have higher values of metric M compared to classes in N?
  - Mann-Whitney U test – to test stochastic superiority of C over N with respect to metric M
  - Probability of superiority – probability that randomly selected metric value from C exceeds randomly selected metric value from N.

Strongly connected core of classes

- **Ant and JFreeChart have strongly connected core: the most central and important classes tend to be in SCCs.**

Software system	Metric	$\overline{C_1}$	$\overline{C_2}$	U	p	NullHyp	PS_1	PS_2
Ant	LOC	194.4	162.2	176965	0.00036	rejected	0.56	0.43
	CC	15.69	14.29	174507	0.0018	rejected	0.51	0.39
	NUMA	4.36	5.01	160886	0.504	**accepted**	0.42	0.44
	NUMM	9.86	8.23	172325	0.006	rejected	0.51	0.42
	IN	9.78	1.95	228723	$< 10^{-4}$	rejected	0.64	0.19
	OUT	5.93	4.02	200409	$< 10^{-4}$	rejected	0.59	0.31
	CBO	15.11	5.98	215327	$< 10^{-4}$	rejected	0.65	0.28
	NOC	1.17	0.21	168921	0.0343	rejected	0.16	0.08
	DIT	1.11	1.14	157439	0.9624	**accepted**	0.35	0.35
	BET	3255.16	103.95	253004	$< 10^{-4}$	rejected	**0.76**	0.14
	PR	0.001744	0.000369	242063	$< 10^{-4}$	rejected	**0.77**	0.22

- Hard refactorings: to remove cyclic dependencies in Ant and JFreeChart we have to reorganize dependencies between core classes

Outline

- Introduction
- Related work and motivation
- Methodology
- Experiments and results
- **Conclusions and future work**

Conclusions and future work

- Real-world OO software systems contain large cyclic dependencies
- Cyclic dependencies densify with size
- The densification phenomena can be modeled by a power-law whose scaling exponent can be used as an indicator of software quality
- Presence of strongly connected cores that negatively impact software maintainability
- Future work:
  - structure of strongly connected components in package and method collaboration networks
  - evolution of strongly connected components in software networks

Densification of cyclic dependencies among classes in OO software systems

Miloš Savić and Mirjana Ivanović

Department of Mathematics and Informatics
Faculty of Sciences
University of Novi Sad