Software systems through complex networks science

Lovro Šubelj & Marko Bajec

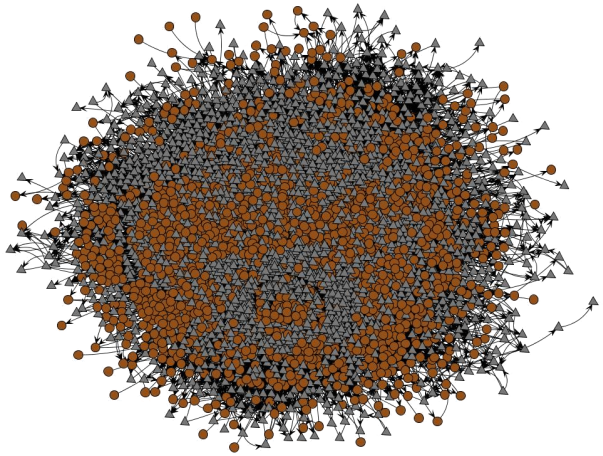
University of Ljubljana
Faculty of Computer and Information Science
Slovenia

August 12, 2012

Outline

Introduction

- Software is among most sophisticated human-made systems.
- Little is known about the structure of 'good' software.
- The above dilemma was denoted *software law* problem.
- Networks provide a possible framework for software analysis.



We review different network analysis techniques → software engineering!

Outline

Software networks

Class dependency networks:

- software project classes → nodes,
- software (inter-)class dependencies → links.

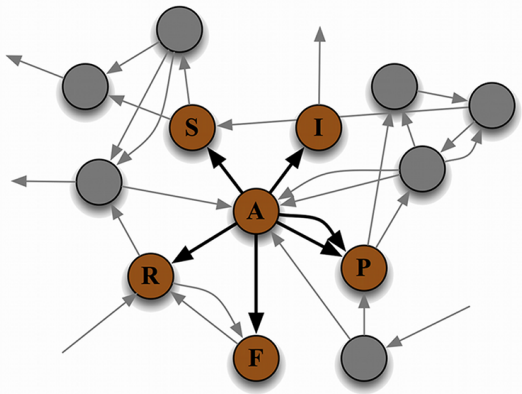
```
class A extends S implements I {

  F field;

  public A (P parameter) {
    ...
  }

  public R function(P parameter) {
    ...
    return R;
  }

}
```



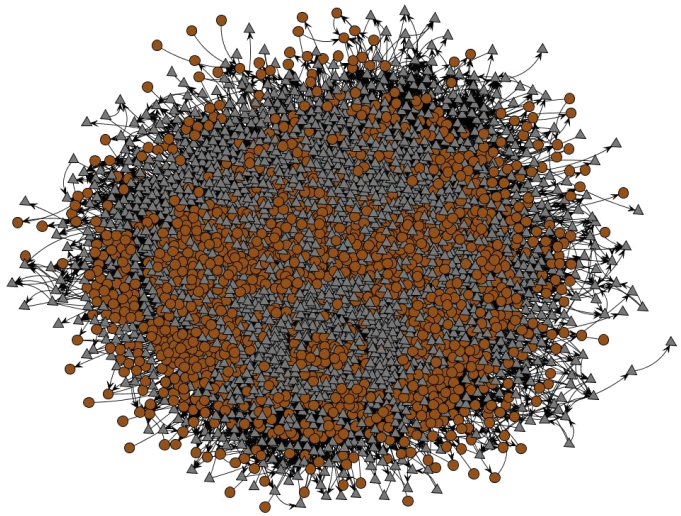


Figure: (left) Java class and corresponding class dependency network. (right) Class dependency network of `java` and `javax` namespaces of Java.

Software networks II

Class dependency networks:

- constructed merely from signatures,
- related to information flow within the project,
- mesoscopic structures coincide with project packages.

Network	Project	n	m	k	LCC	$\|A\|$	$\|P\|$
flmng	Flamingo 4.1	141	269	3.82	0.88	153	18
colt	Colt 1.2.0	243	720	5.93	0.94	267	21
jung	JUNG 2.0.1	317	719	4.54	0.96	357	41
org	Java 1.6.0.7	709	3571	10.07	0.69	778	50
weka	Weka 3.6.6	953	4097	8.60	0.98	1054	84
javax	Java 1.6.0.7	1595	5287	6.63	0.44	1889	118
java	Java 1.6.0.7	1516	10049	13.26	1.00	1518	56

Table: Class dependency networks used in the study.

Outline

Scale-freeness – complexity and reusability

Scale-free networks:

- degree distribution follows a power-law $p_k \sim k^{-\gamma}$, $\gamma > 1$,
- γ related to spreading processes (e.g., bug propagation),
- an artifact of Yule's process (*rich-get-richer* phenomena).

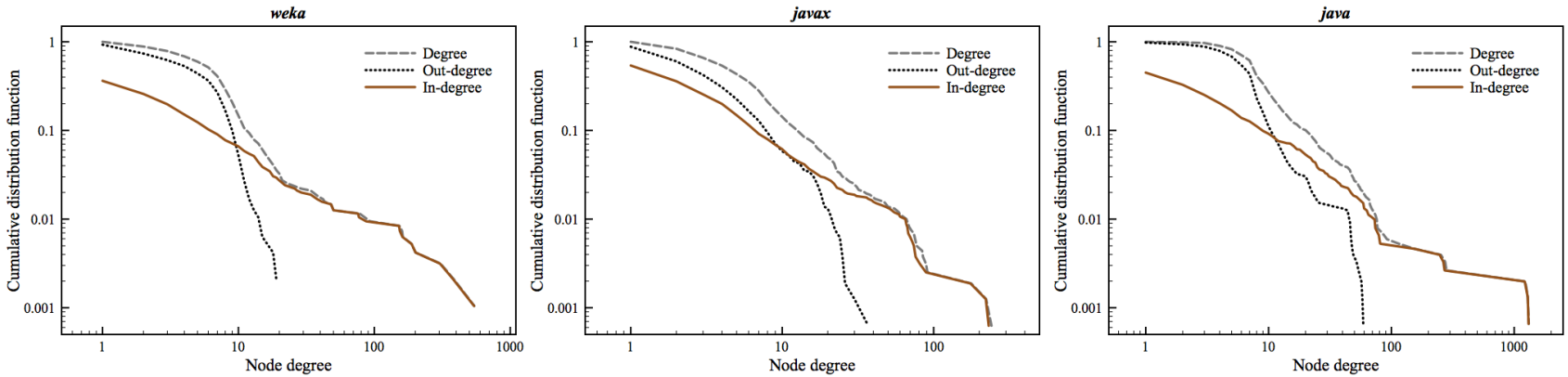


Figure: Degree distributions of *weka*, *javax* and *java* networks.

Distributions p_k^{in} and p_k^{out} are related to code reusability and complexity!

Scale-freeness – complexity and reusability II

weka			*javax*			*java*		
Node	k_i^{in}	k_i^{out}	Node	k_i^{in}	k_i^{out}	Node	k_i^{in}	k_i^{out}
`Instances`	541	5	`JComponent`	235	11	`String`	1308	7
`Instance`	381	4	`Accessible`	222	1	`Class`	1288	4
`ClassAssigner`	0	19	`JTable`	6	37	`FileDialog`	0	59
`Filter`	0	19	`JTextPane`	0	30	`Frame`	4	58

Table: Hubs (i.e., high degree nodes) within *weka*, *javax* and *java* networks.

Software networks:

- scale-free nature of p_k^{in} and highly truncated p_k^{out},
- lower γ implies higher code reuse and decreases fault propagation,
- classes with high k_i^{out} (and k_i^{in}) should be implemented with care.

Small-worldness – structure and design

Small-world networks:

- large clustering or transitivity $C \gg C_{ER}$,
- short distances between the nodes $l \approx l_{ER}$.

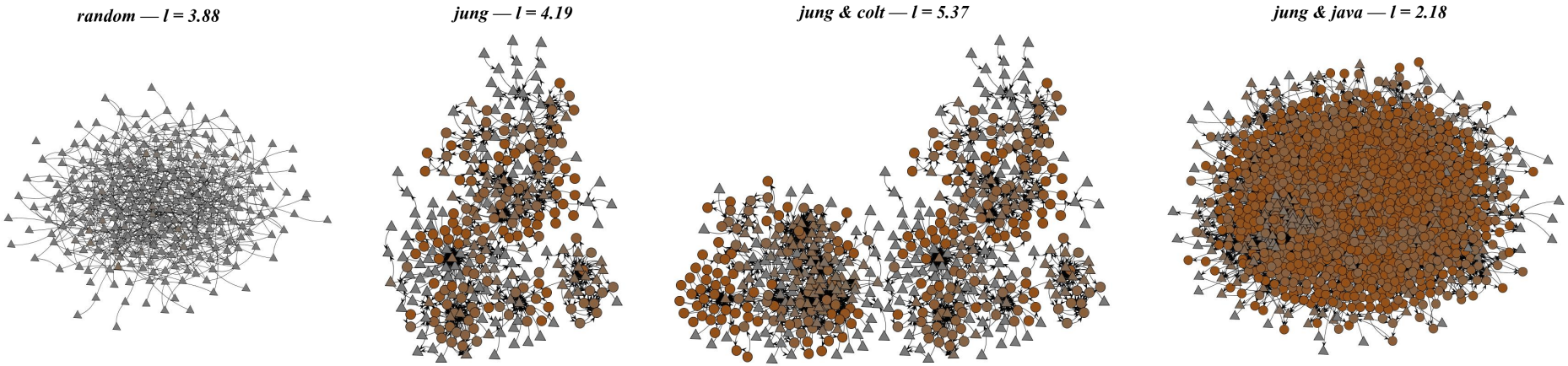


Figure: A random graph, *jung*, *jung & colt* and *jung & java* networks. l equals 3.88, 4.19, 5.37 and 2.18, while node symbols correspond to clustering C.

C and l are related to characteristics and structural design of the project!

Small-worldness – structure and design II

Network	γ	C	D	C_{ER}	l	E	l_{ER}	n_{d}/n
flmng	3.0	0.25	0.31	*0.03*	4.05	0.03	*3.47*	0.38
colt	2.7	0.41	0.47	*0.02*	3.44	0.03	*3.16*	0.30
jung	2.5	0.37	0.42	*0.01*	4.19	0.02	*3.88*	0.48
org	2.2	0.57	0.62	*0.01*	2.68	0.03	*2.81*	0.39
weka	3.0	0.39	0.43	*0.01*	2.91	0.01	*3.39*	0.12
javax	2.6	0.38	0.44	*0.00*	3.88	0.02	*3.16*	0.30
java	2.4	0.69	0.73	*0.01*	2.18	0.02	*3.09*	0.17

Table: Statistics for class dependency networks used in the study.

Software networks:

- well designed project should have $C \gg C_{ER}$ and $l \approx l_{ER}$,
- one should be wary of $l \gg l_{ER}$ throughout the project evolution,
- projects should not be combined with the core of the language.

Nodes – vulnerability and robustness

Network vulnerability and robustness:

- seed nodes can propagate faults throughout the project,
- centrality metrics DC_i, CC_i, BC_i are an indicator of seed nodes,
- classes with high BC_i (and DC_i) can influence the entire project,
- classes with high CC_i are prone to arbitrary fault within the project.

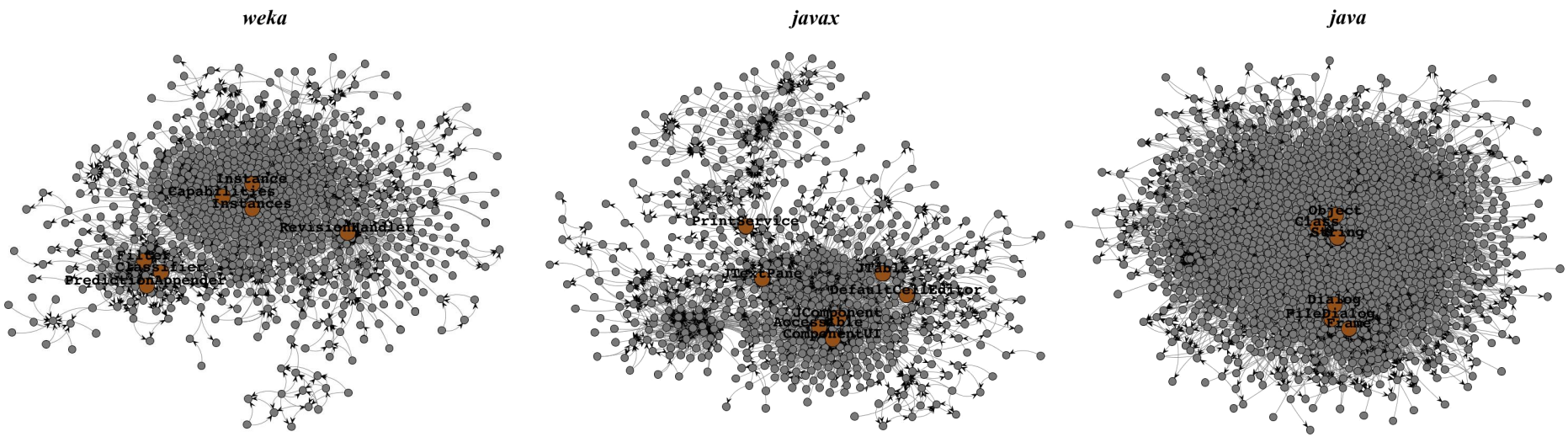


Figure: *weka*, *javax* and *java* networks with highlighted seed nodes.

Nodes – vulnerability and robustness II

weka			*javax*			*java*		
Node	CC_i	BC_i	Node	CC_i	BC_i	Node	CC_i	BC_i
`Prediction...`	0.03	0.00	`DefaultCell...`	0.10	0.00	`FileDialog`	0.09	0.00
`Classifier`	0.03	0.01	`JTable`	0.10	0.12	`Dialog`	0.09	0.00
`Instances`	0.01	0.51	`JComponent`	0.04	0.23	`String`	0.02	0.36
`RevisionHandler`	0.00	0.26	`Accessible`	0.01	0.18	`Object`	0.02	0.32

Table: Seed nodes (i.e., influential nodes) within *weka*, *javax* and *java* networks.

Software networks:

- classes with high BC_i (and DC_i) should be implemented with care,
- classes with high CC_i can be adopted for effective, efficient testing.

Nodes – controllability

Network controllability:

- driver nodes n_d can control the output of the entire project,
- contrary to seed nodes, driver nodes tend to avoid hubs,
- most software network are not highly controllable.

Network	γ	C	D	C_{ER}	l	E	l_{ER}	n_{d}/n
flmng	3.0	0.25	0.31	*0.03*	4.05	0.03	*3.47*	0.38
colt	2.7	0.41	0.47	*0.02*	3.44	0.03	*3.16*	0.30
jung	2.5	0.37	0.42	*0.01*	4.19	0.02	*3.88*	0.48
org	2.2	0.57	0.62	*0.01*	2.68	0.03	*2.81*	0.39
weka	3.0	0.39	0.43	*0.01*	2.91	0.01	*3.39*	0.12
javax	2.6	0.38	0.44	*0.00*	3.88	0.02	*3.16*	0.30
java	2.4	0.69	0.73	*0.01*	2.18	0.02	*3.09*	0.17

Table: Statistics for class dependency networks used in the study.

Software networks:

- controllability can be limited by decreasing k or γ.

Modules – aggregation and modularity

Network aggregation and modularity:

- software packages reflect in different structural modules,
- visualization classes aggregate into densely connected communities,
- parsers arrange into functional modules with common linkage pattern.

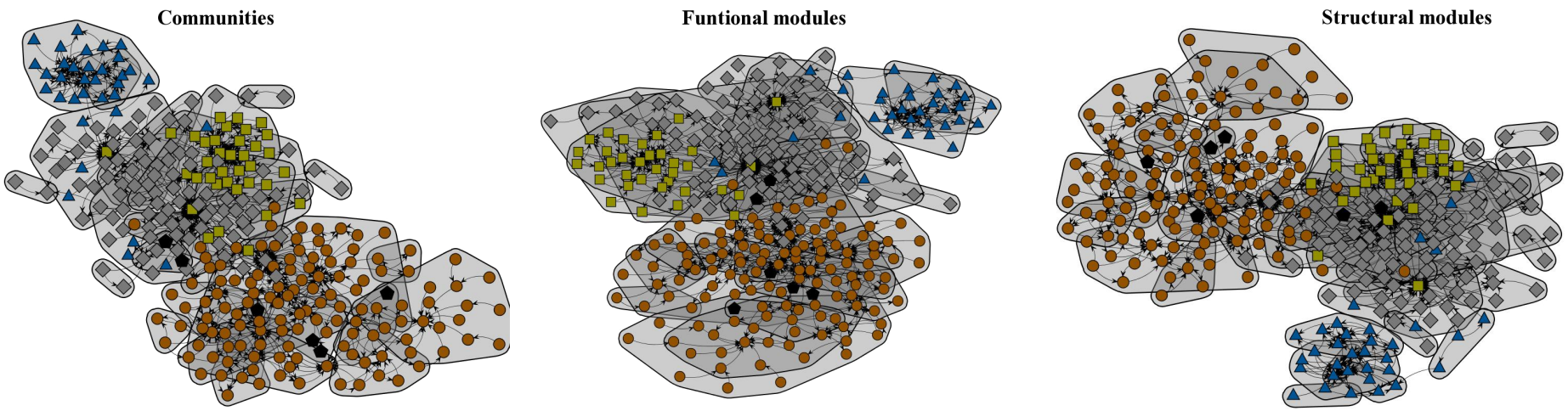


Figure: (left) Communities representing modular structure. (middle) Functional modules representing functional partitioning. (right) General structural modules.

Modules – aggregation and modularity II

General structural modules most accurately model the package structure!

Network		MO		CP		MM		GP	
flmng	16	0.580	14	**0.609**	27	0.521	16	**0.610**	26
colt	19	0.519	10	0.473	20	**0.533**	19	**0.530**	26
jung	39	0.614	13	0.650	30	0.661	39	**0.680**	41
org	47	0.503	11	**0.537**	30	0.378	39	**0.536**	33
weka	81	**0.558**	26	0.410	49	0.430	63	0.314	28
javax	107	0.704	59	**0.761**	155	0.392	89	0.747	192

Table: Normalized mutual information of packages and network modules.

Software networks:

- community structure signifies highly modular structure of the project,
- functional modules are related to functional roles within the project.

Outline

Applications – software project abstraction

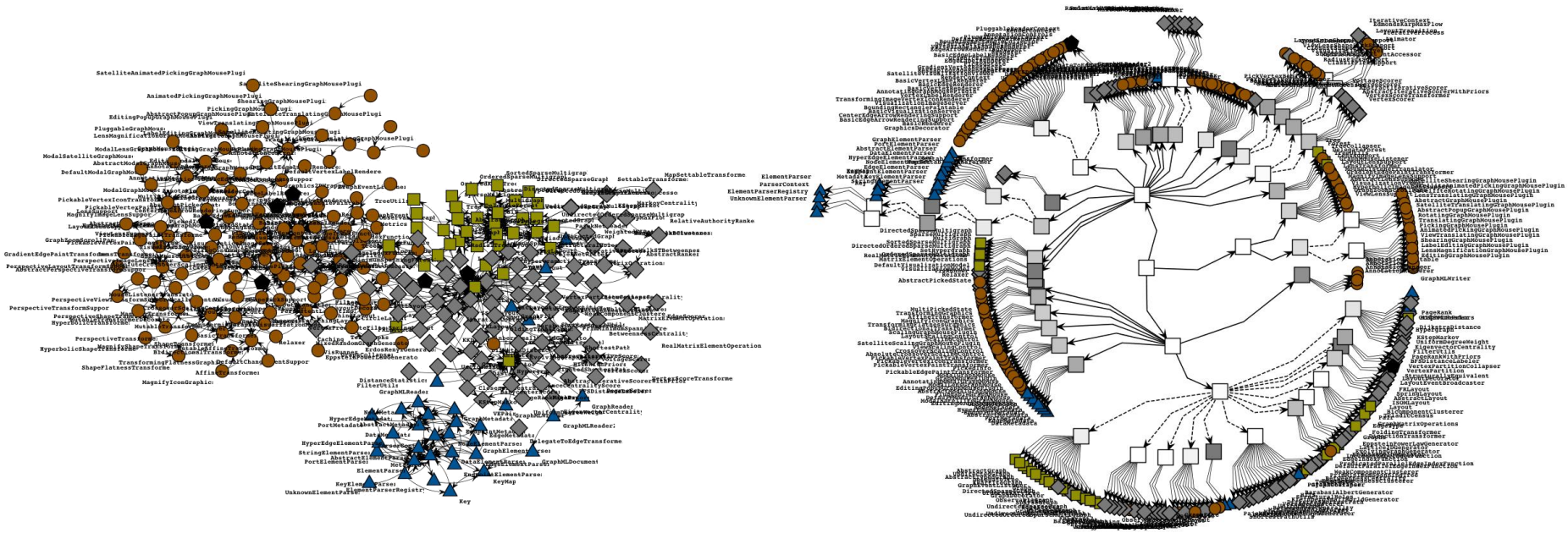


Figure: (left) *jung* network where node symbols represent high-level packages. (right) Revealed hierarchy of structural modules that is consistent with packages.

Applications – software package prediction

Software package prediction:

- package of a class is the most likely package within its module,
- nodes are weighted according to Jaccard similarity.

Network	l	l_∞	P	P_4	P_3	P_2	P_1
flmng	2.65	4	**0.566**	←	0.572	*0.793*	1.000
colt	3.35	4	**0.654**	←	*0.756*	0.942	1.000
jung	2.97	4	**0.617**	←	0.663	*0.857*	1.000
org	3.50	7	**0.616**	0.616	*0.714*	0.989	1.000
weka	3.02	6	**0.684**	0.692	*0.736*	0.871	1.000
javax	3.11	5	**0.626**	0.631	*0.816*	0.982	1.000

Table: Classification accuracy for software package prediction.

Packages can be predicted with $\approx 80\%$ probability for most classes, package hierarchy can be precisely identified for over 60% of the classes!

Outline

Conclusions

Conclusions:

- a study of software networks constructed from Java source code,
- macroscopic, mesoscopic and microscopic network properties,
- different network-based software project quality indicators,
- prominent set of techniques for software engineering.

Future work:

- comparison with other software quality metrics,
- framework that could be easily applied in practice,
- extension to also intra-class dependencies.

Thank you.

✉ lovro.subelj@fri.uni-lj.si

www http://lovro.lpt.fri.uni-lj.si/