5th International Workshop on Emerging Trends in Software Metrics (WETSoM 2014)

In-Depth Measurement and Analysis on *Densification Power Law* of Software Execution

Yu Qu

yqu@sei.xjtu.edu.cn

MOE Key Lab for Intelligent Networks and Network Security

Xi'an Jiaotong University

China

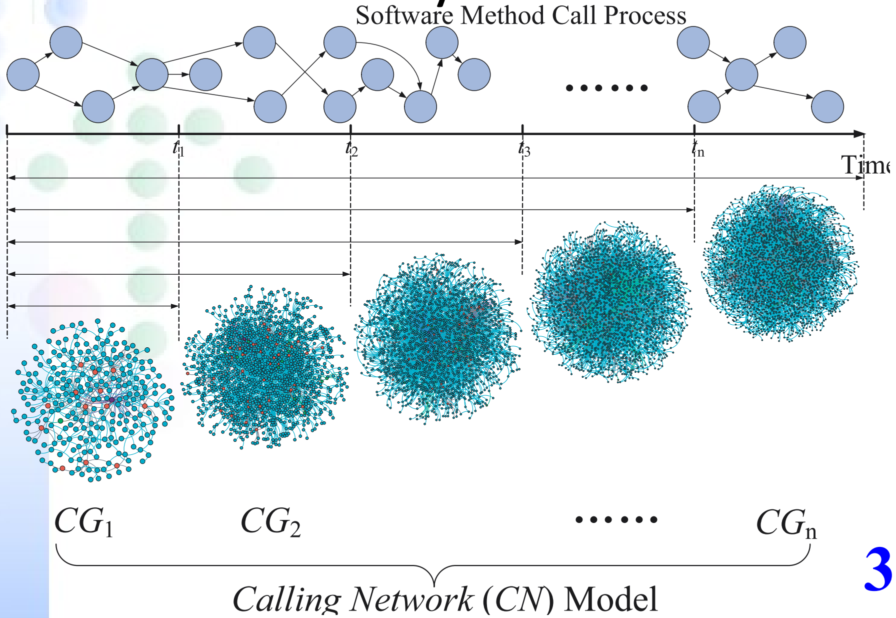
2014/6/11

Outline

1. Introduction
2. Densification Power Law & Calling Network Model
3. Measurements & Discussions
4. Conclusion & Future Work

Introduction

- ✓ *Complex Network* theory & graph algorithms: successfully applied to software measurement and modeling.
  - ➢ But: Only a few of them concentrate on dynamic execution.
- ✓ Many network growing models, e.g., *preferential attachment model*, have been proposed in *Complex Network* theory.
  - ➢ But: None of the existing research has investigated whether or not dynamic execution of software also obey such models.



Growing Process of Makagiga's (http://sourceforge.net/projects/makagiga) dynamic *Call Graph* during its execution.

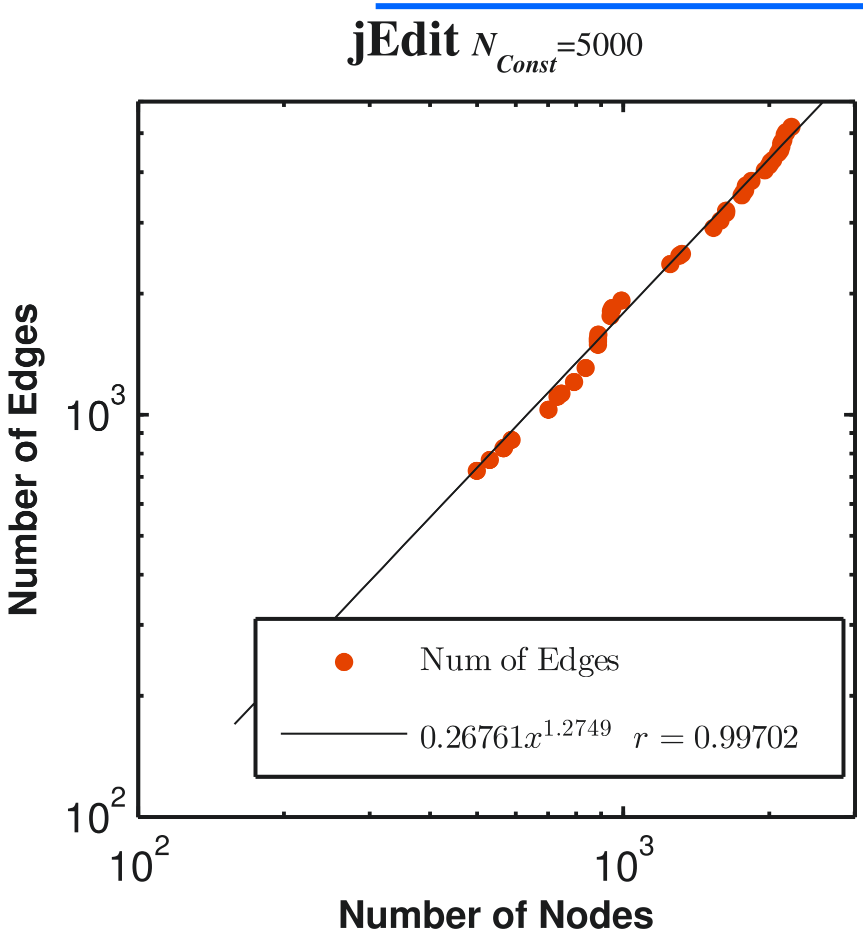
Introduction

- ✓ Research Questions:
  1. Is there any common law among different software systems' execution processes?
  2. Can we discover new metrics for software execution from a growing network point of view?
- ✓ Contributions:
  1. Based on 15 widely-used Java programs, we show the universality of an interesting feature – *Densification Power Law* (*DPL*) of software execution. Might be an appropriate metric for software execution process.
  2. A comparison between static *Call Graph* and *DPL* is presented.
  3. An *explanation* for DPL is given.

Outline

1. Introduction
2. Densification Power Law & Calling Network Model
3. Measurements & Discussions
4. Conclusion & Future Work

Densification Power Law



In recent years, it has been discovered that, real word networks' evolution often follows a pattern:

$$e(t) \propto n(t)^a, 1 \leq a \leq 2$$

Number of Edges **Number of Nodes**

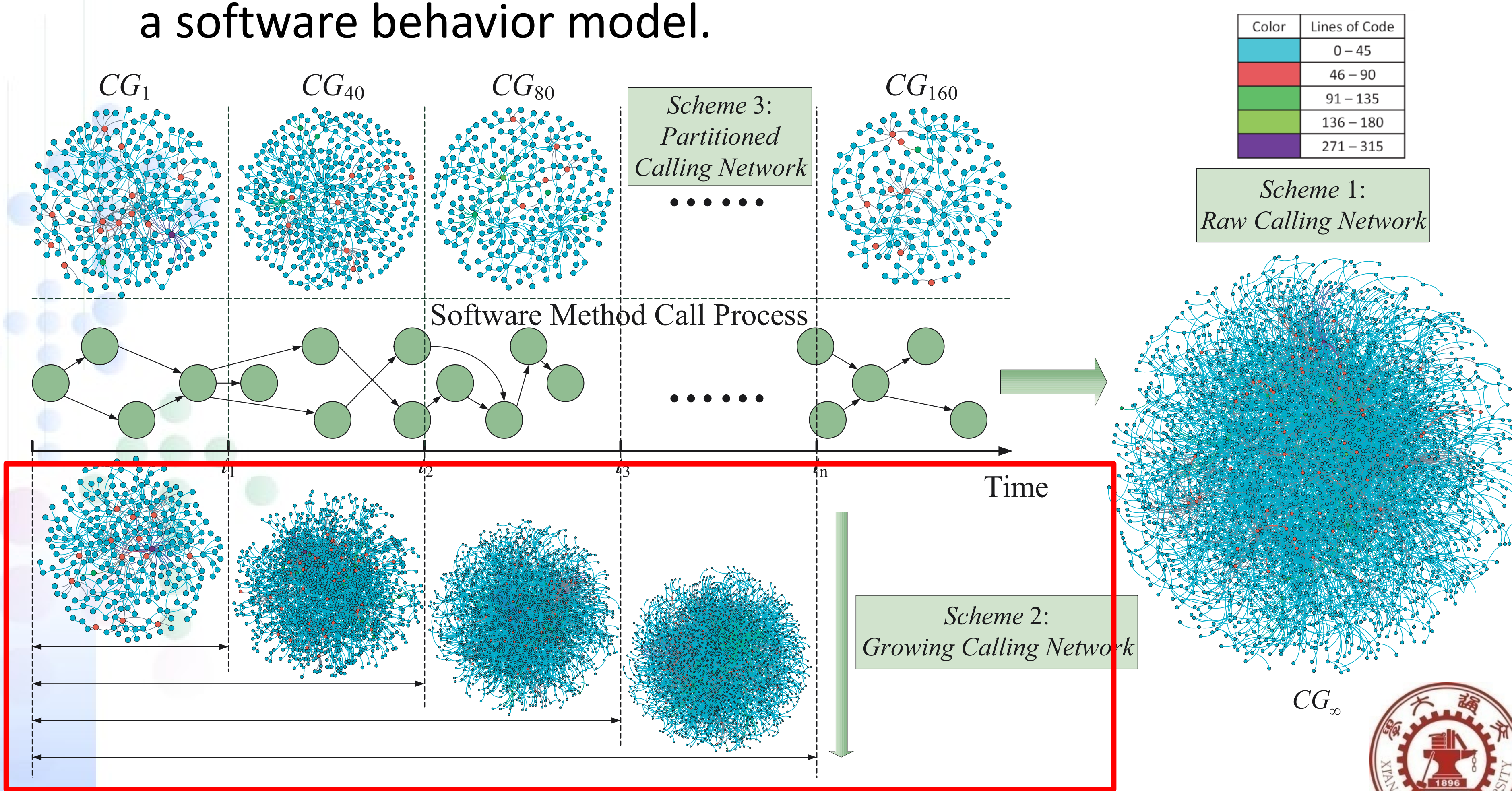
Densification Power Law (DPL)

[Leskovec et al., KDD '05, ACM ToKDD '07]

Such phenomenon is not in accordance with traditional models, e.g., *preferential attachment model*, *copying model* etc.

Calling Network Model

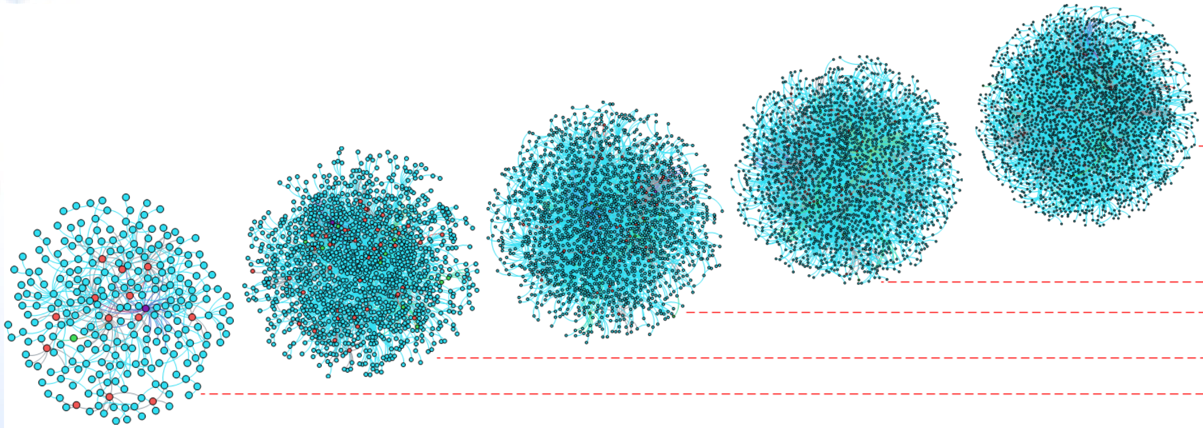
✓ In our previous research [Yu et al., SoftwareMining 13'], we have discovered ***DPL*** feature in software's *Calling Network* (*CN*), CN is a software behavior model.



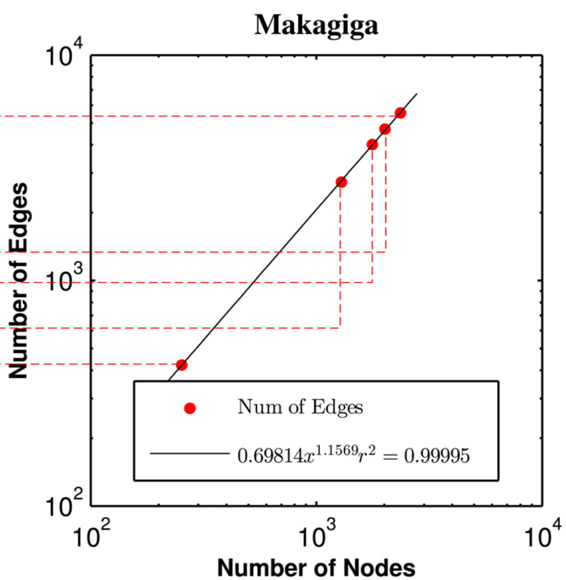
Densification Power Law of Growing CN

$$\begin{cases} CN = \{CG_i \mid i \in \mathbb{N}\}, \\ CG_i = f_{CG-Gen}(CB_i), CB_i \subseteq CB \text{ and} \\ CB_i = \{cb_k \mid (i-1)\cdot N_{Itv} \leq k \leq (i-1)\cdot N_{Itv} + N_{CG}\}, \\ CG = (V, E), w : E \to \mathbb{N}, \\ CB = \{cb_k \mid k \in \mathbb{N}\}, \\ cb_k = (t_k, Caller_k, Callee_k, Param_k). \end{cases}$$

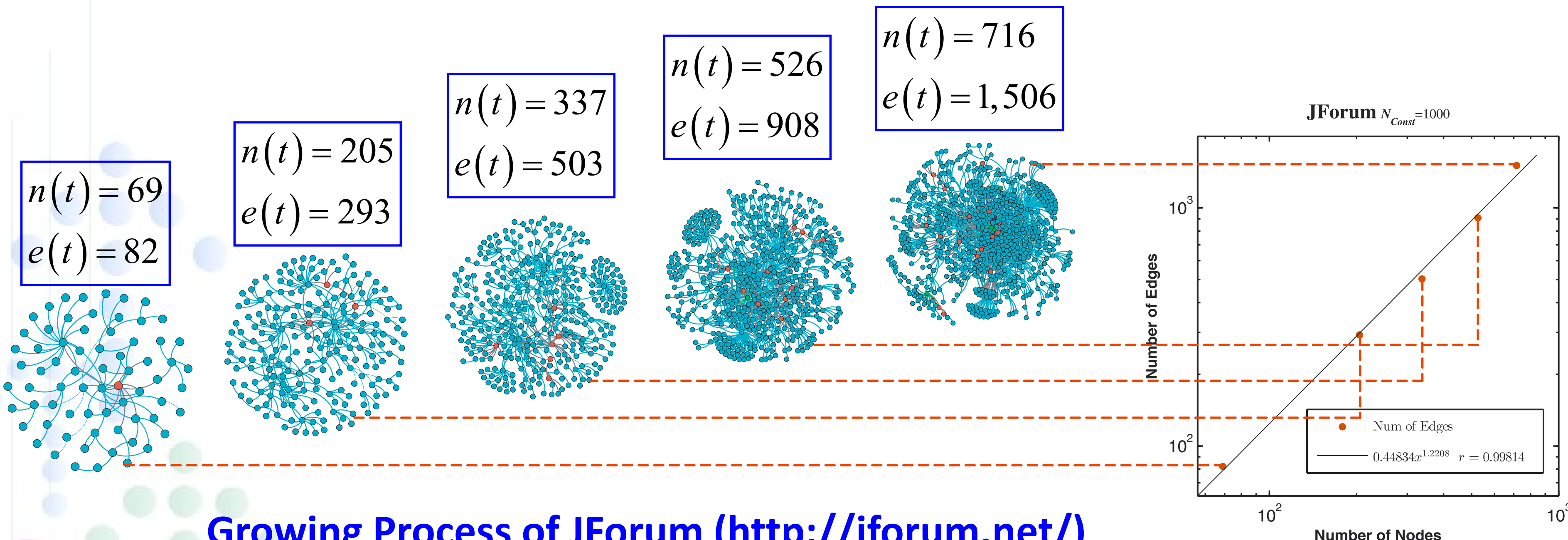
$$\boxed{N_{Itv} = 0 \text{ and } N_{CG} = i \cdot N_{Const}}$$



Growing Process of Makagiga



Densification Power Law of Growing CN



Growing Process of JForum (http://jforum.net/)

$$N_{Const} = 1,000$$

Outline

1. Introduction
2. Densification Power Law & Calling Network Model
3. Measurements & Discussions
4. Conclusion & Future Work

Data Set & Experiment Setup

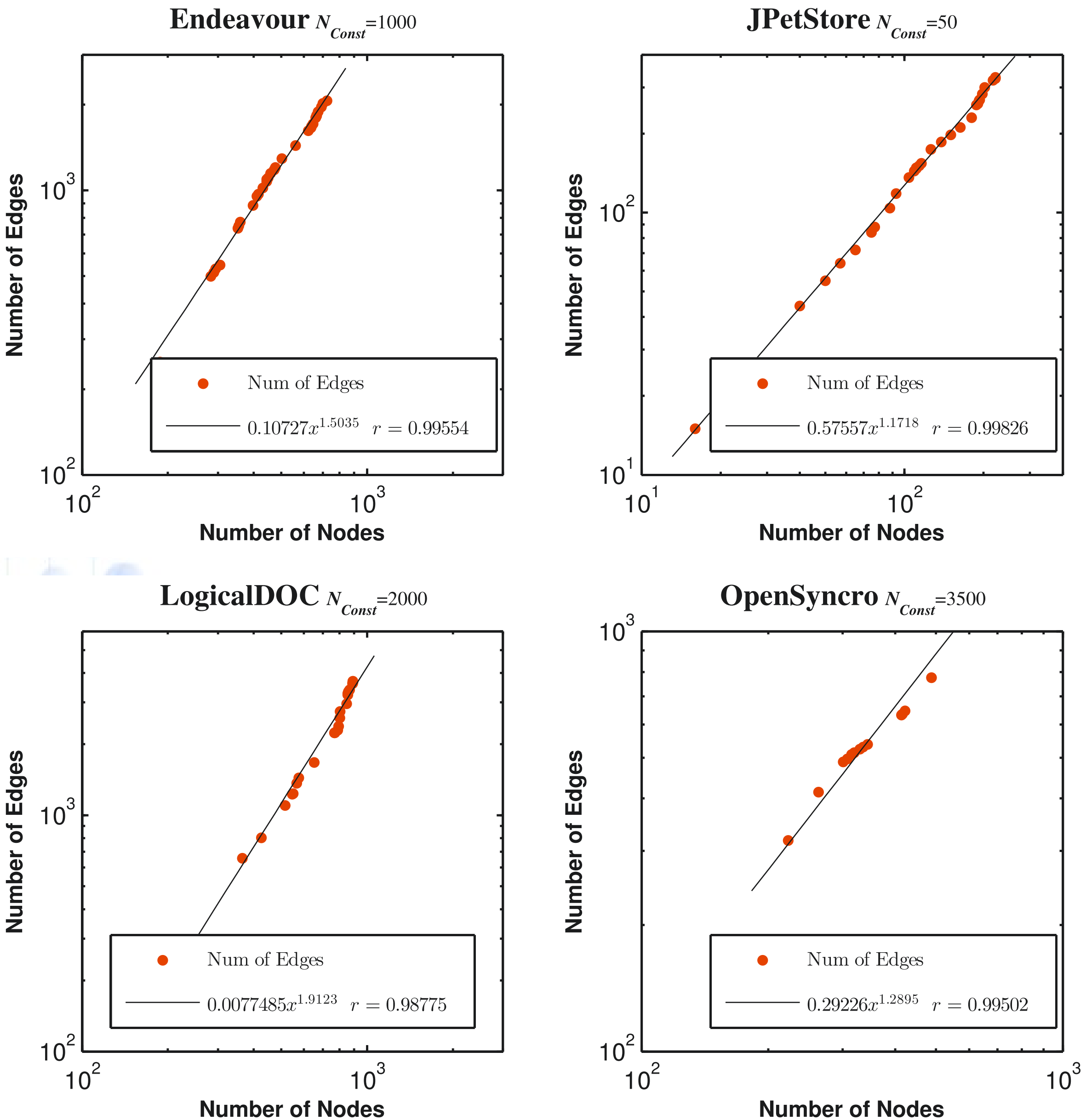
Table 1: Experiment subject programs

Programs	Arch	Version	SLoC	# Method	$\|CB\|$	Website
DrJava	Desktop	5.7.5	162,416	10,593	447,842	http://www.drjava.org/
Endeavour	Web	1.21	18,312	1,706	39,322	http://sourceforge.net/projects/endeavour-mgmt/
FreeMind	Desktop	0.9.0	53,669	5,974	192,694	http://sourceforge.net/projects/freemind/
JabRef	Desktop	2.9.2	144,406	6,449	185,134	http://jabref.sourceforge.net/
jEdit	Desktop	5.1.0	185,569	7844	438,321	http://www.jedit.org/
JForum	Web	2.1.9	65,040	2,991	42,516	http://jforum.net/
JPetStore	Web	6.0	1,893	289	2,099	http://code.google.com/p/mybatis/
Kunagi	Web	0.23	176,486	18,021	198,259	http://kunagi.org/
LogicalDOC	Web	6.7.0	131,888	8,692	160,685	http://www.logicaldoc.com/
Makagiga	Desktop	3.8.2	156,906	10,356	324,928	http://sourceforge.net/projects/makagiga/
OpenKM	Web	6.2.2	N/A	N/A	249,990	http://www.openkm.com/
OpenProj	Desktop	1.4	151,821	11,632	371,750	http://sourceforge.net/projects/openproj/
OpenSyncro	Web	2.2	54,163	3,276	137,433	http://www.opensyncro.org/
Sweet Home 3D	Desktop	4.2	109,090	6,346	381,586	http://www.sweethome3d.com/
Weka	Desktop	3.7.10	N/A	N/A	237,239	http://www.cs.waikato.ac.nz/ml/weka/

8 Desktop and 7 Web systems CB: quantity of method call records

- ✓ A data set containing 15 real-world open-source Java programs are collected.
- ✓ The Kieker framework (http://kieker-monitoring.net/), which is an open-source dynamic monitoring framework based on AspectJ, is used as the instrumentation tool.

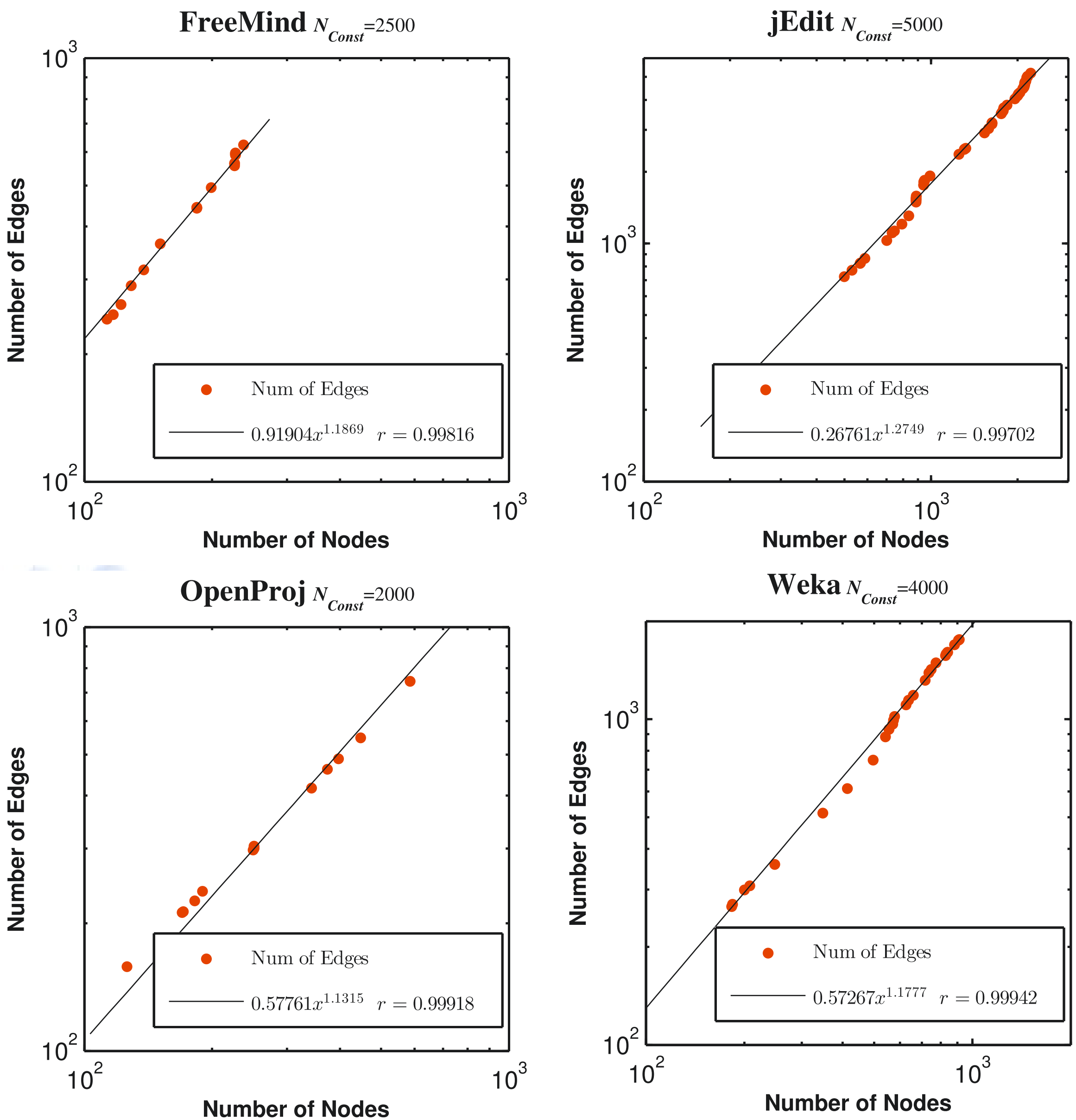
DPL results (1)



DPL results of 4 Web Programs.

Straight line is the linear regression fit result, *r* is correlation coefficient.

DPL results (2)



DPL results of 4 Desktop Programs.

DPL results (3)

Table 2: Densification Power Law results and comparison with *Static Call Graphs*

Programs	Equation	N_{Const}	r	$N_{highest}$	# Methods	N_{static}	E_{static}	$E_{equation}$	$\Delta E\%$
DrJava	$0.68431x^{1.1276}$	7,000	0.99956	1,701	10,593	6,204	13,346	12,938	-3.06%
Endeavour	$0.10727x^{1.5035}$	1,000	0.99544	723	1,706	1,468	3,331	6,189	85.8%
FreeMind	$0.91904x^{1.1869}$	2,500	0.99816	236	5,974	3,724	7,804	15,914	103.91%
JabRef	$0.67521x^{1.1477}$	4,000	0.99894	867	6,449	4,885	9,805	11,565	17.95%
jEdit	$0.26761x^{1.2749}$	5,000	0.99702	2,221	7,844	5,606	13,845	16,094	16.24%
JForum	$0.31046x^{1.2845}$	1,000	0.99341	715	2,991	2,051	5,749	5,579	-3.03%
JPetStore	$0.57557x^{1.1718}$	50	0.99826	221	289	97	124	122	-1.2%
Kunagi	$0.73925x^{1.1287}$	4,500	0.99858	780	18,021	12,583	24,699	31,349	26.92%
LogicalDOC	$0.0077485x^{1.9123}$	2,000	0.98775	891	8,692	5,932	14,112	127,270	801.82%
Makagiga	$0.47128x^{1.2075}$	1,500	0.99887	1,776	10,356	7,078	17,781	20,992	18.06%
OpenKM	$0.46685x^{1.1842}$	6,000	0.99963	1,389	N/A	N/A	N/A	N/A	N/A
OpenProj	$0.57761x^{1.1315}$	2,000	0.99918	2,823	11,632	7,258	13,752	13,494	-1.87%
OpenSyncro	$0.29226x^{1.2895}$	3,500	0.99502	657	3,276	1,865	2,937	4,823	64.21%
Sweet Home 3D	$0.7917x^{1.1404}$	5,500	0.9995	1,117	6,346	4,547	11,219	11,744	4.68%
Weka	$0.57267x^{1.1777}$	4,000	0.99942	910	N/A	N/A	N/A	N/A	N/A

DPL equations

DPL results (3)

Table 2: Densification Power Law results and comparison with *Static Call Graphs*

Programs	Equation	N_{Const}	r	$N_{highest}$	# Methods	N_{static}	E_{static}	$E_{equation}$	$\Delta E\%$
DrJava	$0.68431x^{1.1276}$	7,000	0.99956	1,701	10,593	6,204	13,346	12,938	-3.06%
Endeavour	$0.10727x^{1.5035}$	1,000	0.99544	723	1,706	1,468	3,331	6,189	85.8%
FreeMind	$0.91904x^{1.1869}$	2,500	0.99816	236	5,974	3,724	7,804	15,914	103.91%
JabRef	$0.67521x^{1.1477}$	4,000	0.99894	867	6,449	4,885	9,805	11,565	17.95%
jEdit	$0.26761x^{1.2749}$	5,000	0.99702	2,221	7,844	5,606	13,845	16,094	16.24%
JForum	$0.31046x^{1.2845}$	1,000	0.99341	715	2,991	2,051	5,749	5,579	-3.03%
JPetStore	$0.57557x^{1.1718}$	50	0.99826	221	289	97	124	122	-1.2%
Kunagi	$0.73925x^{1.1287}$	4,500	0.99858	780	18,021	12,583	24,699	31,349	26.92%
LogicalDOC	$0.0077485x^{1.9123}$	2,000	0.98775	891	8,692	5,932	14,112	127,270	801.82%
Makagiga	$0.47128x^{1.2075}$	1,500	0.99887	1,776	10,356	7,078	17,781	20,992	18.06%
OpenKM	$0.46685x^{1.1842}$	6,000	0.99963	1,389	N/A	N/A	N/A	N/A	N/A
OpenProj	$0.57761x^{1.1315}$	2,000	0.99918	2,823	11,632	7,258	13,752	13,494	-1.87%
OpenSyncro	$0.29226x^{1.2895}$	3,500	0.99502	657	3,276	1,865	2,937	4,823	64.21%
Sweet Home 3D	$0.7917x^{1.1404}$	5,500	0.9995	1,117	6,346	4,547	11,219	11,744	4.68%
Weka	$0.57267x^{1.1777}$	4,000	0.99942	910	N/A	N/A	N/A	N/A	N/A

Different values of
are used

DPL results (3)

Table 2: Densification Power Law results and comparison with *Static Call Graphs*

Programs	Equation	N_{Const}	r	$N_{highest}$	# Methods	N_{static}	E_{static}	$E_{equation}$	$\Delta E\%$
DrJava	$0.68431x^{1.1276}$	7,000	0.99956	1,701	10,593	6,204	13,346	12,938	-3.06%
Endeavour	$0.10727x^{1.5035}$	1,000	0.99544	723	1,706	1,468	3,331	6,189	85.8%
FreeMind	$0.91904x^{1.1869}$	2,500	0.99816	236	5,974	3,724	7,804	15,914	103.91%
JabRef	$0.67521x^{1.1477}$	4,000	0.99894	867	6,449	4,885	9,805	11,565	17.95%
jEdit	$0.26761x^{1.2749}$	5,000	0.99702	2,221	7,844	5,606	13,845	16,094	16.24%
JForum	$0.31046x^{1.2845}$	1,000	0.99341	715	2,991	2,051	5,749	5,579	-3.03%
JPetStore	$0.57557x^{1.1718}$	50	0.99826	221	289	97	124	122	-1.2%
Kunagi	$0.73925x^{1.1287}$	4,500	0.99858	780	18,021	12,583	24,699	31,349	26.92%
LogicalDOC	$0.0077485x^{1.9123}$	2,000	0.98775	891	8,692	5,932	14,112	127,270	801.82%
Makagiga	$0.47128x^{1.2075}$	1,500	0.99887	1,776	10,356	7,078	17,781	20,992	18.06%
OpenKM	$0.46685x^{1.1842}$	6,000	0.99963	1,389	N/A	N/A	N/A	N/A	N/A
OpenProj	$0.57761x^{1.1315}$	2,000	0.99918	2,823	11,632	7,258	13,752	13,494	-1.87%
OpenSyncro	$0.29226x^{1.2895}$	3,500	0.99502	657	3,276	1,865	2,937	4,823	64.21%
Sweet Home 3D	$0.7917x^{1.1404}$	5,500	0.9995	1,117	6,346	4,547	11,219	11,744	4.68%
Weka	$0.57267x^{1.1777}$	4,000	0.99942	910	N/A	N/A	N/A	N/A	N/A

All the programs' growing processes obey *DPL* with very close correlation.

DPL results (3)

Table 2: Densification Power Law results and comparison with *Static Call Graphs*

Programs	Equation	N_{Const}	r	$N_{highest}$	# Methods	N_{static}	E_{static}	$E_{equation}$	$\Delta E\%$
DrJava	$0.68431x^{1.1276}$	7,000	0.99956	1,701	10,593	6,204	13,346	12,938	-3.06%
Endeavour	$0.10727x^{1.5035}$	1,000	0.99544	723	1,706	1,468	3,331	6,189	85.8%
FreeMind	$0.91904x^{1.1869}$	2,500	0.99816	236	5,974	3,724	7,804	15,914	103.91%
JabRef	$0.67521x^{1.1477}$	4,000	0.99894	867	6,449	4,885	9,805	11,565	17.95%
jEdit	$0.26761x^{1.2749}$	5,000	0.99702	2,221	7,844	5,606	13,845	16,094	16.24%
JForum	$0.31046x^{1.2845}$	1,000	0.99341	715	2,991	2,051	5,749	5,579	-3.03%
JPetStore	$0.57557x^{1.1718}$	50	0.99826	221	289	97	124	122	-1.2%
Kunagi	$0.73925x^{1.1287}$	4,500	0.99858	780	18,021	12,583	24,699	31,349	26.92%
LogicalDOC	$0.0077485x^{1.9123}$	2,000	0.98775	891	8,692	5,932	14,112	127,270	801.82%
Makagiga	$0.47128x^{1.2075}$	1,500	0.99887	1,776	10,356	7,078	17,781	20,992	18.06%
OpenKM	$0.46685x^{1.1842}$	6,000	0.99963	1,389	N/A	N/A	N/A	N/A	N/A
OpenProj	$0.57761x^{1.1315}$	2,000	0.99918	2,823	11,632	7,258	13,752	13,494	-1.87%
OpenSyncro	$0.29226x^{1.2895}$	3,500	0.99502	657	3,276	1,865	2,937	4,823	64.21%
Sweet Home 3D	$0.7917x^{1.1404}$	5,500	0.9995	1,117	6,346	4,547	11,219	11,744	4.68%
Weka	$0.57267x^{1.1777}$	4,000	0.99942	910	N/A	N/A	N/A	N/A	N/A

The total number of methods in subject programs.

DPL results (3)

Table 2: Densification Power Law results and comparison with *Static Call Graphs*

Programs	Equation	N_{Const}	r	$N_{highest}$	# Methods	N_{static}	E_{static}	$E_{equation}$	$\Delta E\%$
DrJava	$0.68431x^{1.1276}$	7,000	0.99956	1,701	10,593	6,204	13,346	12,938	-3.06%
Endeavour	$0.10727x^{1.5035}$	1,000	0.99544	723	1,706	1,468	3,331	6,189	85.8%
FreeMind	$0.91904x^{1.1869}$	2,500	0.99816	236	5,974	3,724	7,804	15,914	103.91%
JabRef	$0.67521x^{1.1477}$	4,000	0.99894	867	6,449	4,885	9,805	11,565	17.95%
jEdit	$0.26761x^{1.2749}$	5,000	0.99702	2,221	7,844	5,606	13,845	16,094	16.24%
JForum	$0.31046x^{1.2845}$	1,000	0.99341	715	2,991	2,051	5,749	5,579	-3.03%
JPetStore	$0.57557x^{1.1718}$	50	0.99826	221	289	97	124	122	-1.2%
Kunagi	$0.73925x^{1.1287}$	4,500	0.99858	780	18,021	12,583	24,699	31,349	26.92%
LogicalDOC	$0.0077485x^{1.9123}$	2,000	0.98775	891	8,692	5,932	14,112	127,270	801.82%
Makagiga	$0.47128x^{1.2075}$	1,500	0.99887	1,776	10,356	7,078	17,781	20,992	18.06%
OpenKM	$0.46685x^{1.1842}$	6,000	0.99963	1,389	N/A	N/A	N/A	N/A	N/A
OpenProj	$0.57761x^{1.1315}$	2,000	0.99918	2,823	11,632	7,258	13,752	13,494	-1.87%
OpenSyncro	$0.29226x^{1.2895}$	3,500	0.99502	657	3,276	1,865	2,937	4,823	64.21%
Sweet Home 3D	$0.7917x^{1.1404}$	5,500	0.9995	1,117	6,346	4,547	11,219	11,744	4.68%
Weka	$0.57267x^{1.1777}$	4,000	0.99942	910	N/A	N/A	N/A	N/A	N/A

The number of nodes and the number of edges of the *static Call Graphs*.

DPL results (3)

Table 2: Densification Power Law results and comparison with *Static Call Graphs*

Programs	Equation	N_{Const}	r	$N_{highest}$	# Methods	N_{static}	E_{static}	$E_{equation}$	$\Delta E\%$
DrJava	$0.68431x^{1.1276}$	7,000	0.99956	1,701	10,593	6,204	13,346	12,938	-3.06%
Endeavour	$0.10727x^{1.5035}$	1,000	0.99544	723	1,706	1,468	3,331	6,189	85.8%
FreeMind	$0.91904x^{1.1869}$	2,500	0.99816	236	5,974	3,724	7,804	15,914	103.91%
JabRef	$0.67521x^{1.1477}$	4,000	0.99894	867	6,449	4,885	9,805	11,565	17.95%
jEdit	$0.26761x^{1.2749}$	5,000	0.99702	2,221	7,844	5,606	13,845	16,094	16.24%
JForum	$0.31046x^{1.2845}$	1,000	0.99341	715	2,991	2,051	5,749	5,579	-3.03%
JPetStore	$0.57557x^{1.1718}$	50	0.99826	221	289	97	124	122	-1.2%
Kunagi	$0.73925x^{1.1287}$	4,500	0.99858	780	18,021	12,583	24,699	31,349	26.92%
LogicalDOC	$0.0077485x^{1.9123}$	2,000	0.98775	891	8,692	5,932	14,112	127,270	801.82%
Makagiga	$0.47128x^{1.2075}$	1,500	0.99887	1,776	10,356	7,078	17,781	20,992	18.06%
OpenKM	$0.46685x^{1.1842}$	6,000	0.99963	1,389	N/A	N/A	N/A	N/A	N/A
OpenProj	$0.57761x^{1.1315}$	2,000	0.99918	2,823	11,632	7,258	13,752	13,494	-1.87%
OpenSyncro	$0.29226x^{1.2895}$	3,500	0.99502	657	3,276	1,865	2,937	4,823	64.21%
Sweet Home 3D	$0.7917x^{1.1404}$	5,500	0.9995	1,117	6,346	4,547	11,219	11,744	4.68%
Weka	$0.57267x^{1.1777}$	4,000	0.99942	910	N/A	N/A	N/A	N/A	N/A

The difference between total number of methods and N_{static} is significant. Complete and accurate static *Call Graph* is hard to construct. Frameworks like Spring make such task more difficult.

DPL results (3)

Table 2: Densification Power Law results and comparison with *Static Call Graphs*

Programs	Equation	N_{Const}	r	$N_{highest}$	# Methods	N_{static}	E_{static}	$E_{equation}$	$\Delta E\%$
DrJava	$0.68431x^{1.1276}$	7,000	0.99956	1,701	10,593	6,204	13,346	12,938	-3.06%
Endeavour	$0.10727x^{1.5035}$	1,000	0.99544	723	1,706	1,468	3,331	6,189	85.8%
FreeMind	$0.91904x^{1.1869}$	2,500	0.99816	236	5,974	3,724	7,804	15,914	103.91%
JabRef	$0.67521x^{1.1477}$	4,000	0.99894	867	6,449	4,885	9,805	11,565	17.95%
jEdit	$0.26761x^{1.2749}$	5,000	0.99702	2,221	7,844	5,606	13,845	16,094	16.24%
JForum	$0.31046x^{1.2845}$	1,000	0.99341	715	2,991	2,051	5,749	5,579	-3.03%
JPetStore	$0.57557x^{1.1718}$	50	0.99826	221	289	97	124	122	-1.2%
Kunagi	$0.73925x^{1.1287}$	4,500	0.99858	780	18,021	12,583	24,699	31,349	26.92%
LogicalDOC	$0.0077485x^{1.9123}$	2,000	0.98775	891	8,692	5,932	14,112	127,270	801.82%
Makagiga	$0.47128x^{1.2075}$	1,500	0.99887	1,776	10,356	7,078	17,781	20,992	18.06%
OpenKM	$0.46685x^{1.1842}$	6,000	0.99963	1,389	N/A	N/A	N/A	N/A	N/A
OpenProj	$0.57761x^{1.1315}$	2,000	0.99918	2,823	11,632	7,258	13,752	13,494	-1.87%
OpenSyncro	$0.29226x^{1.2895}$	3,500	0.99502	657	3,276	1,865	2,937	4,823	64.21%
Sweet Home 3D	$0.7917x^{1.1404}$	5,500	0.9995	1,117	6,346	4,547	11,219	11,744	4.68%
Weka	$0.57267x^{1.1777}$	4,000	0.99942	910	N/A	N/A	N/A	N/A	N/A

① ②

DPL results (3)

Table 2: Densification Power Law results and comparison with *Static Call Graphs*

Programs	Equation	N_{Const}	r	$N_{highest}$	# Methods	N_{static}	E_{static}	$E_{equation}$	$\Delta E\%$
DrJava	$0.68431x^{1.1276}$	7,000	0.99956	1,701	10,593	6,204	13,346	12,938	-3.06%
Endeavour	$0.10727x^{1.5035}$	1,000	0.99544	723	1,706	1,468	3,331	6,189	85.8%
FreeMind	$0.91904x^{1.1869}$	2,500	0.99816	236	5,974	3,724	7,804	15,914	103.91%
JabRef	$0.67521x^{1.1477}$	4,000	0.99894	867	6,449	4,885	9,805	11,565	17.95%
jEdit	$0.26761x^{1.2749}$	5,000	0.99702	2,221	7,844	5,606	13,845	16,094	16.24%
JForum	$0.31046x^{1.2845}$	1,000	0.99341	715	2,991	2,051	5,749	5,579	-3.03%
JPetStore	$0.57557x^{1.1718}$	50	0.99826	221	289	97	124	122	-1.2%
Kunagi	$0.73925x^{1.1287}$	4,500	0.99858	780	18,021	12,583	24,699	31,349	26.92%
LogicalDOC	$0.0077485x^{1.9123}$	2,000	0.98775	891	8,692	5,932	14,112	127,270	801.82%
Makagiga	$0.47128x^{1.2075}$	1,500	0.99887	1,776	10,356	7,078	17,781	20,992	18.06%
OpenKM	$0.46685x^{1.1842}$	6,000	0.99963	1,389	N/A	N/A	N/A	N/A	N/A
OpenProj	$0.57761x^{1.1315}$	2,000	0.99918	2,823	11,632	7,258	13,752	13,494	-1.87%
OpenSyncro	$0.29226x^{1.2895}$	3,500	0.99502	657	3,276	1,865	2,937	4,823	64.21%
Sweet Home 3D	$0.7917x^{1.1404}$	5,500	0.9995	1,117	6,346	4,547	11,219	11,744	4.68%
Weka	$0.57267x^{1.1777}$	4,000	0.99942	910	N/A	N/A	N/A	N/A	N/A

The difference between static *Call Graph* and *DPL* equation is significant. *DPL*'s properties can not be derived statically.

$$\Delta E = \frac{E_{equation} - E_{static}}{E_{static}}$$

What does the difference imply? Needs further research.

Microscopic Discussions

- ✓ What makes the difference between *DPL* of software systems and traditional *Complex Network* theory models?

- ✓ In traditional Complex Network models, when a new node arrives, it will connect to (or be connected by) old nodes following certain mechanisms:
  - ➢ ***Preferential attachment model*** [Barabási and Albert, 1999]:

    Growth: When a new node is added, it is connected to m existing nodes.

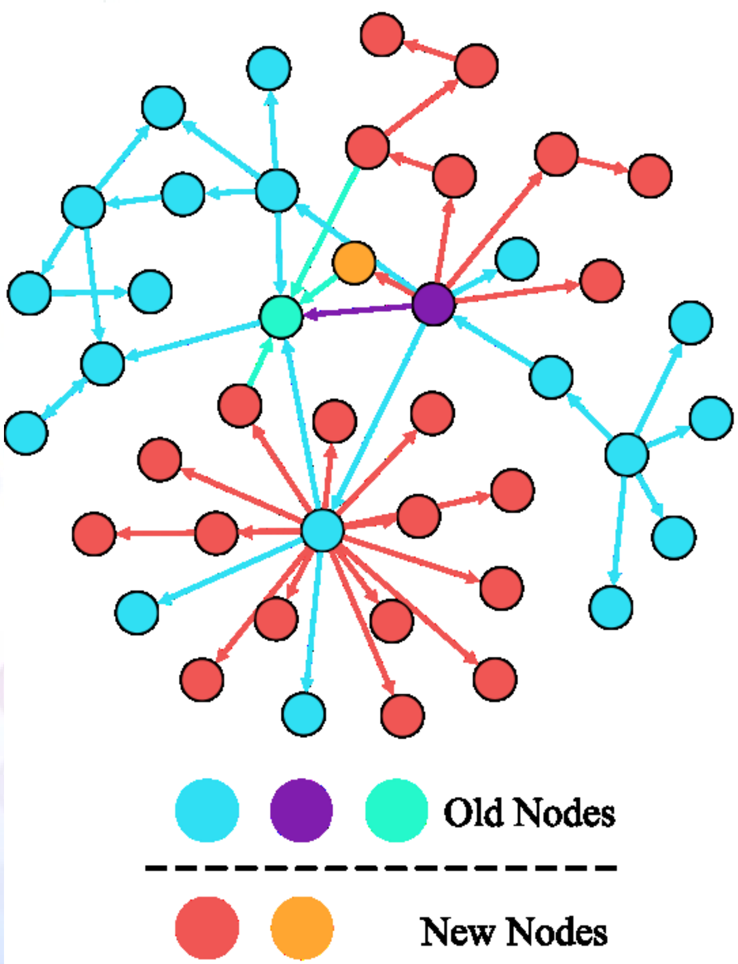
    Preferential attachment: Each new edge is connected to the old sth node with a probability proportional to its degree k_s.

- ✓ Leads to a constant average node degree.

Microscopic Discussions

✓ Growing Details of JForum's *CN*

$$N_{Const} = 50$$



SystemGlobals.getValue

SystemGlobals.getApplicationPath

```
public static String getApplicationPath ()
{
    return getValue(ConfigKeys.APPLICATION_PATH);
}
```

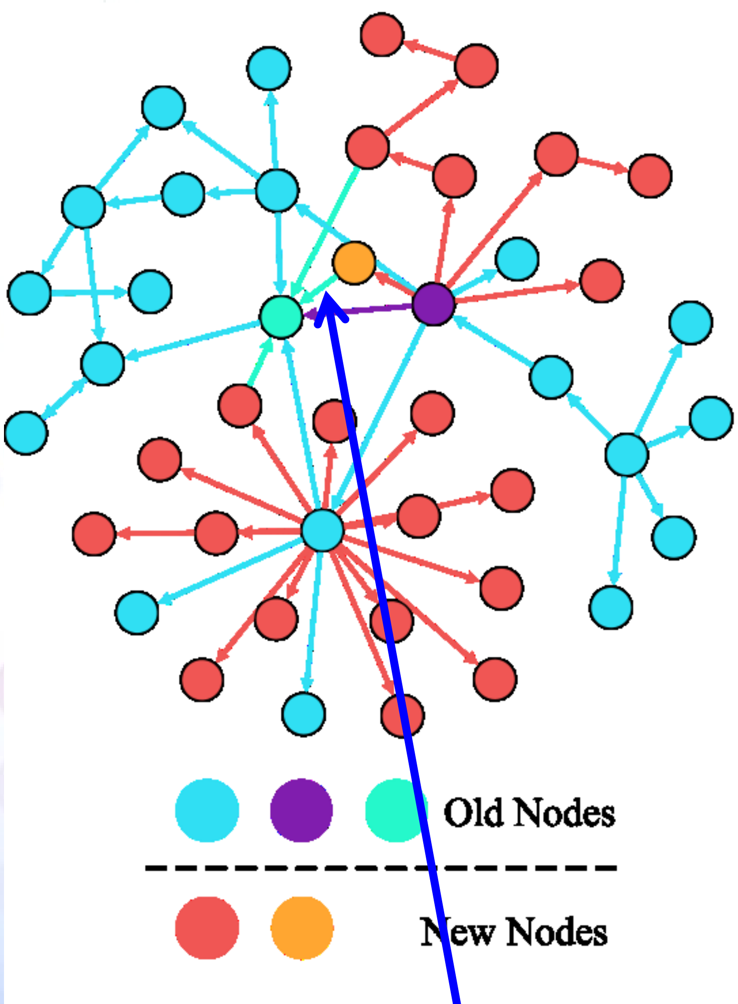
JForumBaseServlet.init

```
public void init(ServletConfig config) throws ServletException
{
    super.init(config);

    try {
        //......
        String defaultPath = SystemGlobals.getApplicationPath ();
        FileTemplateLoader defaultLoader = new FileTemplateLoader();

        String extraTemplatePath = SystemGlobals.getValue();
        //......
        }
    catch (Exception e) {
        //......
    }
}
```

Microscopic Discussions

✓ Growing Details of JForum's *CN*

$$N_{Const} = 50$$



A "Passive" method call

SystemGlobals.getValue

SystemGlobals.getApplicationPath

```
public static String getApplicationPath ()
{
    return getValue(ConfigKeys.APPLICATION_PATH);
}
```

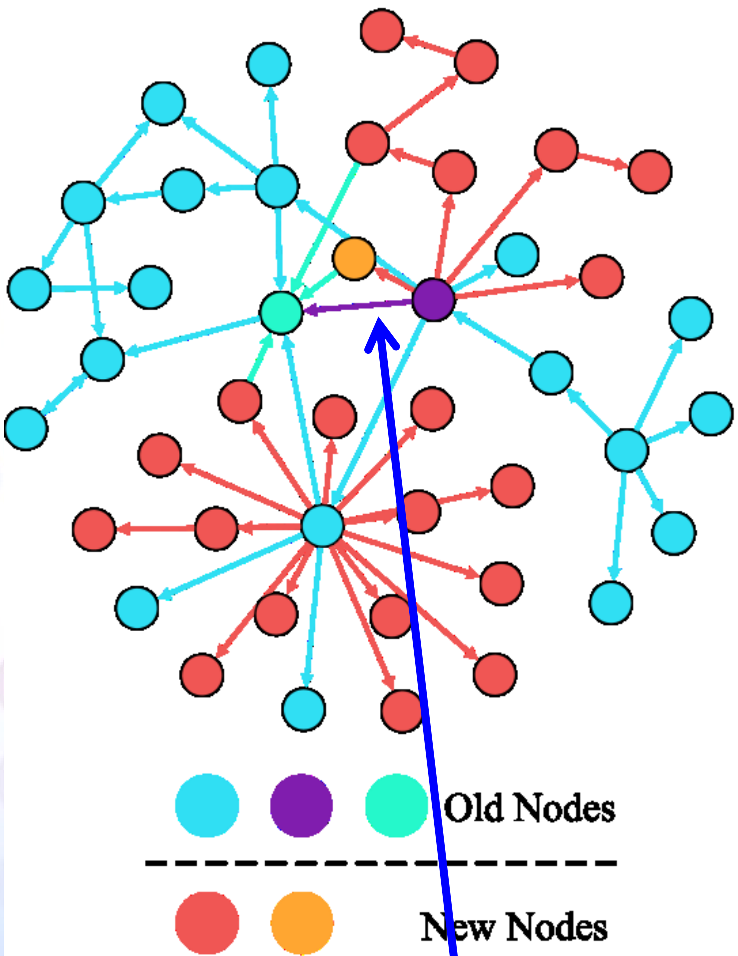
JForumBaseServlet.init

```
public void init(ServletConfig config) throws ServletException
{
    super.init(config);

    try {
        //......
        String defaultPath = SystemGlobals.getApplicationPath ();
        FileTemplateLoader defaultLoader = new FileTemplateLoader();

        String extraTemplatePath = SystemGlobals.getValue();
        //......
    }
    catch (Exception e) {
        //......
    }
}
```

Microscopic Discussions

✓ Growing Details of JForum's *CN*

$$N_{Const} = 50$$



SystemGlobals.getValue

SystemGlobals.getApplicationPath

```
public static String getApplicationPath ()
{
    return getValue(ConfigKeys.APPLICATION_PATH);
}
```

JForumBaseServlet.init

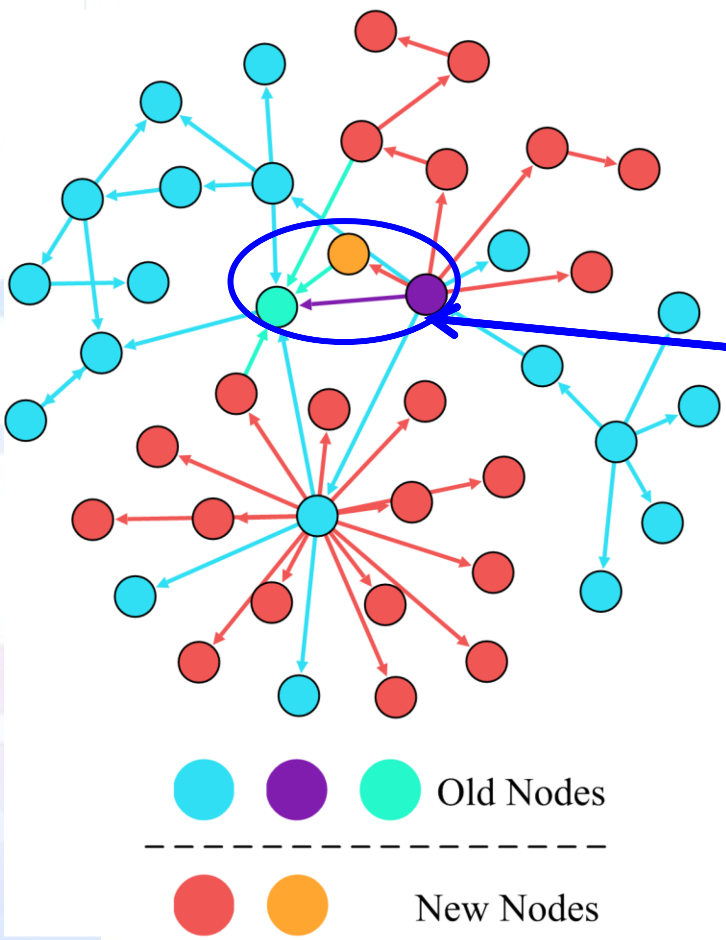
```
public void init(ServletConfig config) throws ServletException
{
    super.init(config);

    try {
        //......
        String defaultPath = SystemGlobals.getApplicationPath ();
        FileTemplateLoader defaultLoader = new FileTemplateLoader();

        String extraTemplatePath = SystemGlobals.getValue();
        //......
        }
    catch (Exception e) {
        //......
    }
}
```

A new edge between 2 old nodes

Microscopic Discussions

✓ Growing Details of JForum's *CN*

$$N_{Const} = 50$$

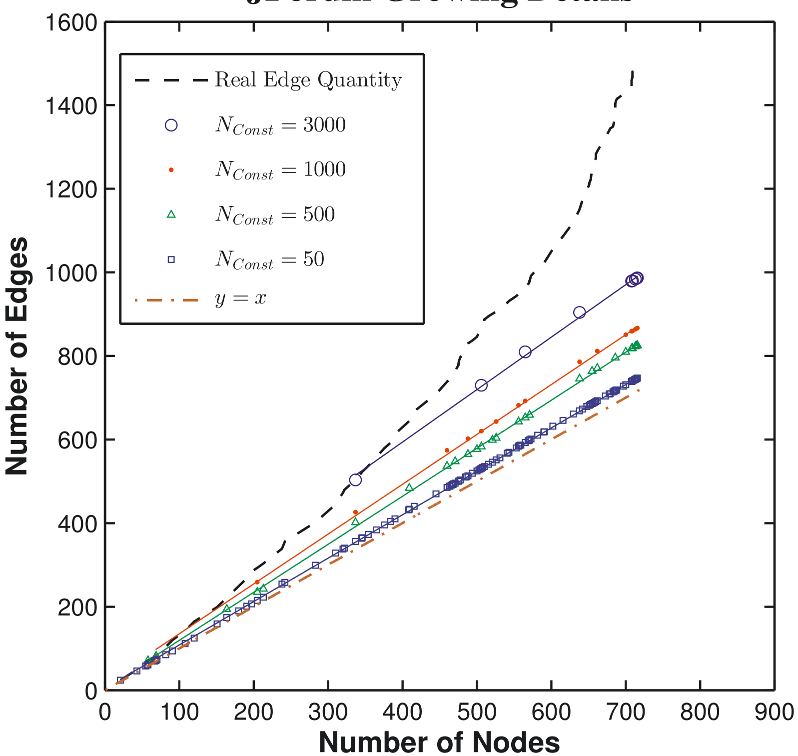


- The preceding Process added 3 new edges. The red one is an edge between old and new nodes.
- The green one is a *passive* edge.
- The purple one is a new edge between 2 old nodes.
- The latter 2 kinds are not considered in existing models.

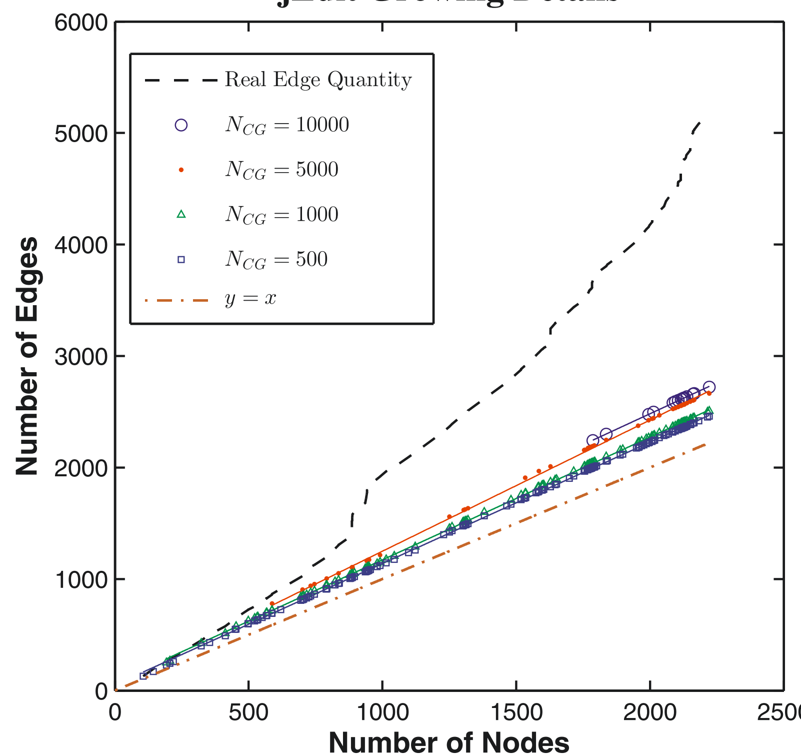
Microscopic Discussions

- ✓ What if these edges are excluded? Whether we could derive a similar result with traditional models?
- ✓ *Answer:* After removing these edges, the results are consistent with existing models.
- ✓ *Conclusion: DPL* is caused by intensive method-reusing.

JForum Growing Details



jEdit Growing Details



Outline

Conclusion

- ✓ Based on 15 widely-used software systems, we have shown the universality of *Densification Power Law* (*DPL*) of software execution.
- ✓ The difference between static *Call Graph* and *DPL* has been presented.
- ✓ An explanation for *DPL* has been given: the major cause of the finding is the reuse of software methods. After removing these reusing method calls, the growth of *CN* is in accordance with traditional *Complex Network* models.

➢ **These measurements and findings will pave new research directions for software metrics.**

Future work

- ✓ What does the difference between static *Call Graph* and *DPL* equation imply?

$$\Delta E = \frac{E_{equation} - E_{static}}{E_{static}}$$

- ✓ How can we take advantage of this interesting and universal feature in software engineering practice?
  - Fault Detection?
  - Structure Evaluation?
  - Redundant Code Size Estimation?
  - …

Thank you & Question?

Yu Qu

yqu@sei.xjtu.edu.cn

MOE Key Lab for Intelligent Networks and Network Security

Xi'an Jiaotong University

China