Article ID: 1000-5641(2017)01-0038-14

Complex network analysis in Java application systems

SHEN Ping-ting, CHEN Liang-yu

(*Shanghai Key Lab of Trustworthy Computing, East China Normal University, Shanghai* 200062, *China*)

Abstract: A lot of empirical studies have demonstrated that Java software system is a kind of artificial complex network and its in-degree distribution obeys the power law, while out-degree distribution is lognormal. However, most experiment objects in these studies are Java development tools, such as JDK, log4j and Tomcat, and the type of data analyzed in dependence graph is limited, because they only consider some class dependence relationships and omit certain useful data types, like member variables and local variables. In this paper, we all useful dependence relationships between entities or modules on both class and function levels, and we further propose a novel method to transform a system network into a weighted directed graph. Comprehensive experiment results show that the in- and out-degree of 10 types of Java application systems mostly fit the power law distributions, and our proposed method to detect the scale-free feature of a weighted and directed network is effective in analyzing Java application systems.

Key words: complex network; Java application systems; power law distribution; class dependence graph; function dependence graph

CLC number: O231 **Document code:** A

DOI: 10.3969/j.issn.1000-5641.2017.01.006

Java 应用系统的复杂网络分析

沈娉婷, 陈良育

(华东师范大学 上海市高可信计算重点实验室, 上海 200062)

摘要: 大量研究表明, Java 软件系统是一种人工复杂网络, 它的入度分布符合幂律, 然而出度却是对数正态分布. 现有的这些研究都仅仅以 Java 开发工具包, 如 JDK、log4j 和 Tomcat 这一类软件系统为研究对象. 除此之外, 所分析的数据类型也很有限, 只考虑了程序包与类的依赖关系, 忽略了很多有用的数据类型, 比如函数成员变量和函数临时变量. 本文将这两类数据类型纳入了研究范畴, 拓展了类依赖关系. 不仅如此, 本文也将研究粒度细化至函数层面, 构建了函数依赖图. 针对这两类依赖关系, 本文提出了将系统转化为加权有向网络图的理论方法.

收稿日期: 2015-12-29

基金项目: 国家自然科学基金 (61321064); 上海市重点学科建设项目 (B412)

第一作者: 沈娉婷, 女, 硕士研究生, 研究方向为数据库与数据挖掘. E-mail: spt0726@hotmail.com.

通信作者: 陈良育, 男, 副教授, 研究方向为符号计算. E-mail: lychen@sei.ecnu.edu.cn.

全面的实验结果显示, 本文所选取的10类Java应用系统, 无论是出度还是入度绝大多数都符合幂律分布, 由此证明了所提出的理论方法, 验证了有向加权网络是否具有无标度特性在分析Java应用系统网络结构时是有效的.

关键词: 复杂网络; Java应用系统; 幂律分布; 类依赖图; 函数依赖图

0 Introduction

The study of software systems using complex network analysis has become an enthusiasm[1-6], because the theory of scale-free feature behind complex networks can be a useful approach to the analysis of concrete software systems. Modeling software systems as networks is a good way to investigate the internal structure, and it is effective to excavate the collaboration or dependence relationships between modules or entities, which are the building blocks of systems. Such method would help developers better understand the structure of large-scale software systems, especially make it easier for them to do thorough study and extension on open-source systems.

Recently, taking advantage of power-law distribution as one scale-free feature to study object-oriented (OO) software systems has been greatly conducted. Concretely, for an OO software system, we usually treat it as a directed graph, where the node represents an entity (e.g, package, class or function) and the directed edge represents the correlation between two entities. Previous studies[3-4,7-14] analyzed systems with the distributions of in-degree and out-degree on the class dependence graph, and detected the scale-free feature according to whether the degree distribution follows power law. However, these previous studies have two limits: First, they only apply analysis on professional systems. For example, the study[7] examines Linux open-source systems and Apache web servers, and researches[10,11] study Java development tools. But, some normal application systems, which are developed for specified uses, are still valuable and significant, because they may have more entities holding closely dependence relationships. Second, those previous studies only investigate the system at package or class level, which is coarse-grained and holds limit kinds of relationships (3 common relationships), as a result, it is not enough for us to understand a system thoroughly.

In this paper, we conduct comprehensive analysis on both class and function levels in 10 types of Java application systems in Tab. 1. The degree distribution is investigated on a class dependence graph that is built up by 5 relationships, and a function dependence graph. We propose a novel method to extract all kinds of relationships between entities of a system, and further transform the system into a weighted directed graph by a modeling method. We then analyze the distributions of in- and out-degree of each graph to see whether they obey power-law distributions. Experimental results show that the in- and out-degree of these Java application systems mostly fit the power-law distributions, which indicates normal Java application systems also belong to the class of scale-free networks and our proposed method is effective in analyzing software systems using complex network theory.

The main purpose of our study is to thoroughly analyze the structure of Java application systems and help developers to approve the system performance with complex network theory. Whether the networks transformed from these systems obey power law can tell us the pros and cons of them, which lack regulation due to open source management and development. If the system obey power law, it has robustness and fragility, and we can find the most influential node to control its bug propagation. If not, the system structure may need to be changed or adjusted according to our analysis results.

The rest of the paper is organized as follows. In section 1, we introduce some preliminary knowledge, including the definitions of dependence relationship, the class and function dependence graphs, and a method to do relationship extraction. We explain a complete analysis method on system network and propose a promising weighting mechanism to relationships in section 2. Section 3 shows the experimental results and analysis. At last we make a conclusion in section 4.

Tab. 1 CDG and FDG networks for 10 types of Java application systems

Network	Description	$\|V\|$	$\|E\|$	$\|V_f\|$	$\|E_f\|$
GISToolkit	GIS system	470	2 467	468	1 488
OpenGTS	GIS system	357	2 474	67 04	7 122
OpenJUMP	GIS system	1 369	8 429	6 062	8 815
dotCMS	CMS system	2 412	13 801	2 382	6 680
infoGlue	CMS system	1 371	6 891	1 359	3 375
OpenCms	CMS system	1 572	12 288	1 558	6 452
eConf	LMS system	61	169	134	103
a-LMS	LMS system	440	1 861	1 988	1 759
OLMS	LMS system	130	358	141	97
JSPWiki	Wiki system	489	2 056	1 843	1 927
JAMWiki	Wiki system	150	744	762	770
FitNesse	Wiki system	672	2 789	1 957	1 764
jforum	Forum system	355	1 587	1 290	1 423
jGossip	Forum system	263	919	507	611
Yazd	Forum system	218	885	891	793
jPortlet	Portal system	133	555	318	273
OpenPortal	Portal system	84	151	87	92
Pluto	Portal system	351	1 189	821	706
GatorMail	WebMail system	110	324	269	197
OlivaMail	WebMail system	60	247	191	215
yawebmail	WebMail system	76	196	157	139
FocusSNS	Blog and SNS system	299	913	283	491
Pebble	Blog and SNS system	655	3 003	2 295	2 875
Roller	Blog and SNS system	513	2 843	2 536	3 314
ITracker	Bug tracking system	386	1 835	382	736
BugRat	Bug tracking system	74	383	364	513
Scarab	Bug tracking system	575	2 449	5 947	5 332
Hipergate	ERP and CRM system	643	2 517	3 181	3 366
SourceTap	ERP and CRM system	169	737	652	636
OpenCustomer	ERP and CRM system	398	2 019	1 151	1 392

1 Preliminaries

In this section, we mainly introduce the definitions of class dependence graph and function dependence graph, then we propose a method to extract kinds of relationships between entities in Java application systems.

1.1 Class Dependence Graph and Function Dependence Graph

Java is a kind of OO development language, and *class* is a basic unit consisting of member variables and member functions. In a program, there are 5 reference and invocation relationships between classes, which are inheritance and implementation, aggregation, parameter, signature and invocation. These dependence relationships between two classes can be used to build up a class dependence graph (abbreviated to CDG), which is defined as a directed network $G=(V,E)$, where V is the set of nodes and E is the set of edges[10]. A node $v \in V$ is a class or an interface. A directed edge $e_{ij}=(v_i,v_j)$ $(e_{ij} \in E)$ connected by node v_i and v_j represents one of the following five relationships[15]:

- R_1-Inheritance and implementation: v_i extends or implements v_j;
- R_2-Aggregation: v_j is the data type of fields (member variables) or attributes in v_i;
- R_3-Parameter: v_j is the parameter type, return type or exception type of member functions in v_i;
- R_4-Signature: v_j is the type of local member variables in v_i;
- R_5-Invocation: v_j is the type of invoked methods inside the member functions in v_i;

Furthermore, function is a smaller granularity unit in application systems. According to the basis of UML theory, the function calls along with their orders play an important role in the sequence diagram[16], since we can know the information and data flow of a system from them. Therefore, we further build up the function dependence graph (abbreviated to FDG) that describe the dependence relationships between two functions in a system.

FDG network is also defined as a directed network $G_f=(V_f,E_f)$, where V_f is the set of function nodes and E_f is the set of edges between functions. There is only one dependence relationship R_6 between two functions, and the directed edge is denoted by $e_{f_{ij}}=(v_{f_i},v_{f_j})$ $(e_{f_{ij}} \in E_f$, $v_{f_i} \in V_f$, $v_{f_j} \in V_f)$. The function dependence relationship is defined as[15]:

- R_6-Invoke: v_{f_j} is the type of invoked method in v_{f_i};

For example, based on the definitions of CDG and FDG, we can get all the relationships from the following Java code as:

```
public class A extends B implements C{
 HashSet <B> va=new HashSet <B>( );
 public static ArrayList <B> fa(C vb, D vc, ArrayList <E> vd) throws F{
        F ve;
        F vf=new F( );
        vf.fb( );
        vf.fc( );}}
```

R_1: $e_{AB}=(v_A,v_B), e_{AC}=(v_A,v_C)$;

R_2: $e_{AB}=(v_A,v_B)$;

R_3: $e_{AB}=(v_A,v_B), e_{AC}=(v_A,v_C), e_{AD}=(v_A,v_D), e_{AE}=(v_A,v_E), e_{AF}=(v_A,v_F)$;

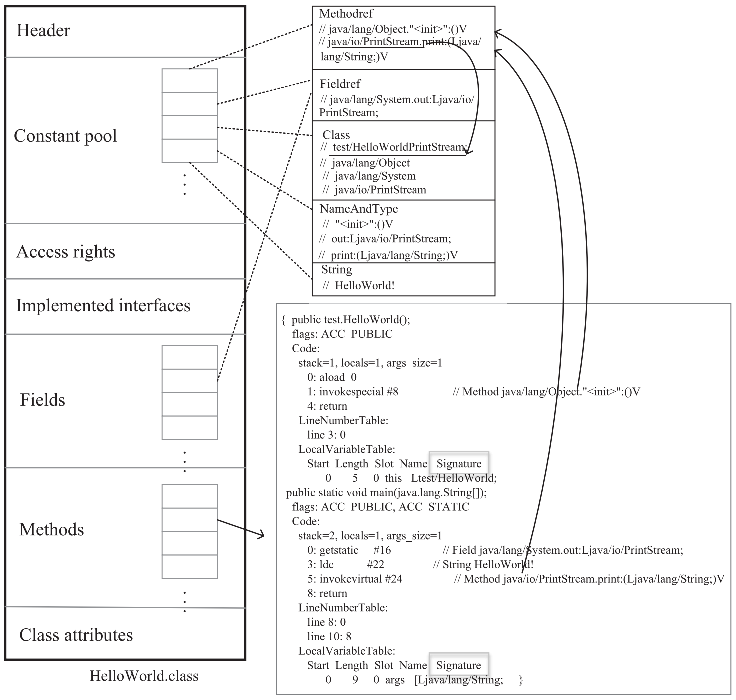
R_4: $e_{AF}=(v_A,v_F)$;

R_5: $e_{AF}=(v_A,v_F), e_{AF}=(v_A,v_F), e_{AF}=(v_A,v_F)$;

R_6: $e_{fab}=(v_{fa},v_{fb}), e_{fac}=(v_{fa},v_{fc})$;

1.2 Extraction of Dependence Relationships from Systems

To build up complete CDG and FDG networks, we need to extract all the six dependence relationships from a Java application system. Based on the source code of a system, only the former three dependence relationships (i.e. R_1, R_2, R_3) could be extracted[10]. To get the left relationships, which are hidden within method bodies, we propose a byte code based method, because Java application systems are typically compiled into byte codes (stored in a .class file shown in Fig. 1.) and the codes own all the information we want to know[17].



```
{  public test.HelloWorld();
   flags: ACC_PUBLIC
   Code:
     stack=1, locals=1, args_size=1
        0: aload_0
        1: invokespecial #8                // Method java/lang/Object."<init>":()V
        4: return
     LineNumberTable:
        line 3: 0
     LocalVariableTable:
        Start  Length  Slot  Name   Signature
            0       5     0  this   Ltest/HelloWorld;
 public static void main(java.lang.String[]);
   flags: ACC_PUBLIC, ACC_STATIC
   Code:
     stack=2, locals=1, args_size=1
        0: getstatic     #16              // Field java/lang/System.out:Ljava/io/PrintStream;
        3: ldc           #22              // String HelloWorld!
        5: invokevirtual #24              // Method java/io/PrintStream.print:(Ljava/lang/String;)V
        8: return
     LineNumberTable:
        line 8: 0
        line 10: 8
     LocalVariableTable:
        Start  Length  Slot  Name   Signature
            0       9     0  args   [Ljava/lang/String;    }
```

Fig. 1 A sample of Java class binary file

Concretely, we first decode the Java .class file (a binary file) into byte codes, where we can get a structured description of a class in detail, including the header, a constant pool, access rights, implemented interfaces, fields, methods and attributes. Therefore, the relationships of R_1, R_2 and R_3 can be directly extracted from the constant pool, and the relationships of R_4 and R_5 are now available in the method bodies which are stored in the Local Variable Table and with keyword Signature. Similarly, the function dependence relationship R_6 can be also identified from the method body in each binary file.

To validate the dependence relationships extracted from byte codes of a system, we use reflection technique[18] to extract the relationships of R_1, R_2 and R_3, and compare the results obtained by the two methods. Reflection is a powerful and trustworthy technique to examine the runtime behavior of applications[8], and thus it provides a way to retrieve methods in a class[19]. For example, when a program is running, we will know that class v_2 is invoked by class v_1 through reflection mechanism. However, signature and invocation relationships (R_4, R_5) cannot be identified in this way. Finally, we get the same results of R_1, R_2 and R_3 through these two methods, and we can draw the conclusion that using our proposed extraction approach based on byte codes is practicable.

2 Network Analysis on Application Systems

Next, we describe a complete analysis method on the system networks. We first propose a weighting mechanism to transform a network graph into a weighted directed graph, then we evaluate the in- and out- degree to see whether they obey power-law distributions .

For every network graph, we remove the isolated nodes and merge multiple edges between the same source node and target node into one weighted edge, which stands for the class dependence relationship or the function dependence relationship. We do the in- and out-degree analysis separately to illustrate the distribution characteristics of CDG and FDG network. The node and edge scale of those systems varies from hundreds to thousands in both CDG and FDG networks shown in Tab. 1. For each type of Java application system, we use three concrete softwares to do the evaluation.

2.1 A Modeling Method for Class Dependence Graph

Previous studies[7,10,20] only extract three kinds of class dependence relationships (i.e. R_1, R_2, R_3) when they build up the CDG. However, according to the software engineering theory[16], the member variables and local variables inside the method body have important positions in a Java system. As the statistical results in Tab. 2, the invocation number, which represents how many times the maximum node is called, is the in-degree for each class dependence relationship. And the values of R_4 and R_5 are far greater than those of R_1 and R_2. So we can conclude that building a weighted and directed CDG network based on 5 relationships model (abbreviated to 5R) instead of 3 relationships model (abbreviated to 3R) is significant and accurate.

Tab. 2　The invocation number of the maximum node in each system

Network	R_1	R_2	R_3	R_4	R_5
GISToolkit	1	20	155	86	54
OpenGTS	3	0	391	335	60
OpenJUMP	1	79	394	64	179
dotCMS	3	0	3900	349	243
infoGlue	4	0	1 850	58	199
OpenCms	0	107	1 894	292	1 162
eConf	0	1	2	5	14
a-LMS	1	0	309	52	2
OLMS	0	0	0	155	9
JSPWiki	1	57	126	52	73
JAMWiki	0	74	2	0	4
FitNesse	2	48	279	103	25
jforum	0	56	55	89	89
jGossip	1	0	105	18	2
Yazd	1	19	39	27	14
jPortlet	4	0	101	1	10
OpenPortal	0	0	8	1	6
Pluto	1	7	56	33	14
GatorMail	0	0	1	0	47
OlivaMail	0	1	1	26	27
yawebmail	1	1	4	0	6
FocusSNS	1	9	31	24	8
Pebble	0	32	86	132	147
Roller	13	0	98	104	0
ITracker	1	2	10	3	94
BugRat	0	0	250	51	101
Scarab	1	27	266	187	45
Hipergate	0	3	151	46	39
SourceTap	0	1	50	45	25
OpenCustomer	0	17	59	100	38

Formally, we use $A^{(i)}(i = 1, 2, 3, 4, 5)$ to denote a matrix representation for the relationship R_i, and the element $A_{jk}^{(i)}$ is the number of edges that node k points to node j. We suppose that all these relationships have the same importance in a system, so in this paper we assign a relationship R_i the weight c_i as $c_1 = 1, c_2 = 1, c_3 = 1, c_4 = 1, c_5 = 1$. So the weighted CDG can be represented by a matrix M as follows:

$$M = \sum_{i=1}^{5} c_i \times A^{(i)}. \tag{1.1}$$

And for a node j, the in-degree and out-degree can be respectively computed by:

$$M_j^{in} = \sum_{k}^{N} M_{jk}, \quad M_j^{out} = \sum_{k}^{N} M_{kj}, \tag{1.2}$$

where N is the number of nodes in graph CDG, and M_{jk} is the matrix representing the relationships from node j to k. In contrast, for the out-degree matrix of node j, M_{kj} represents the relationships from node k to j.

2.2 Construction of Function Dependence Graph

Function is a smaller entity inside the system than class, we should treat it differently and build a desperate graph for it to study its structure. Based on the modeling method of CDG, the FDG network can be constructed by R_6, which represents the invocation relationship between functions. Then we can extract the in-degree and out-degree from the FDG network.

As we can see, Fig. 2 (networks depicted by Gephi) visualizes the CDG and FDG of the focusns system, the dependence relationships correlated with classes are more complex and denser than those in function network, which is reasonable in real systems. This phenomenon may be caused by the reason that the connections between class entities contains the relationships of functions, which means there are fewer dependencies at function level and the degrees at function level are smaller than those at class level.

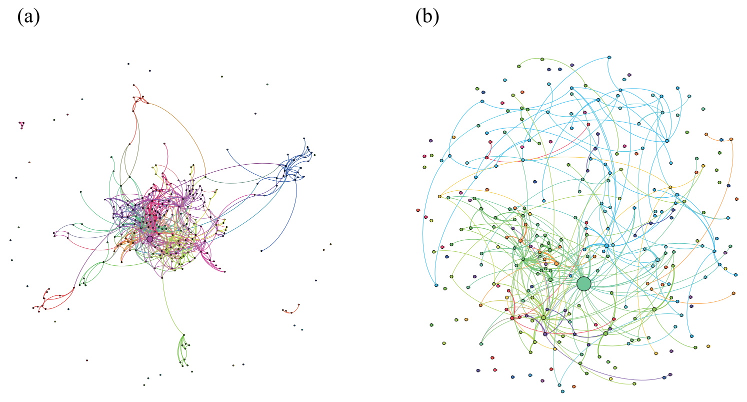


Fig. 2 (a) CDG network of focusns; (b) FDG network of focusns

2.3 Evaluation based on Degree Distributions

Studies[1-2] indicate that software systems have different distributions for in- and out-degree, and the work[7,10] move forward to prove that: the in-degree distribution follows power law, while the out-degree distribution is lognormal. But, all these studies are concerned about

Java development tools or other OO language systems. Here we do further study on Java application systems which have much tighter connections between entities or modules at both class level and function level.

The distributions of in- and out-degree can be calculated by either cumulative distribution function (CDF)[21] or probability density function (PDF)[5]. While CLAUSET[21] proves that in discrete cases, CDF gives more accurate results than what PDF does. In this work, we also use CDF to compute the probability distribution $P(x) = \Pr(X \geqslant x)$, where x is a degree that satisfies $x \geqslant x_{\min}$.

For both CDG and FDG networks, we treat the data sets of in- and out-degree separately and take them as discrete cases. Moreover, all the data are integer values and follows a probability distribution as Eq. (1.3)[21], where x stands for a node degree and α is a constant parameter.

$$p(x) = Cx^{-\alpha}. \tag{1.3}$$

we then use Eq. (1.4) and Eq. (1.5) to calculate the normalizing constant C[21], where Eq. (1.5) is the Hurwitz zeta function:

$$C = \frac{x^{-\alpha}}{\zeta(\alpha, x_{\min})}, \tag{1.4}$$

$$\zeta(\alpha, x_{\min}) = \sum_{n=0}^{\infty}(n + x_{\min})^{-\alpha}. \tag{1.5}$$

At last, we use CDF to compute the distribution $P(x)$ by

$$P(x) = \Pr(X \geqslant x_{\min}) = \frac{\zeta(\alpha, x)}{\zeta(\alpha, x_{\min})}. \tag{1.6}$$

More specifically, the scaling parameter α is estimated by the discrete maximum likelihood estimator(MLE) using:

$$\hat{\alpha} \simeq 1 + n\Big[\sum_{i=1}^{n} \ln \frac{x_i}{x_{\min} - \frac{1}{2}}\Big]^{-1}. \tag{1.7}$$

And the lower bound $x_{\min}$ is computed as:

$$D = \max_{x \geqslant x_{\min}} |S(x) - P(x)|. \tag{1.8}$$

Here we make use of Kolmogorov-Smirnov(KS) statistic[21] to get the maximum distance D between $S(x)$ and $P(x)$.

Now we can analyze $P(x)$ of the in- and out- degree to see whether they follow power-law distributions. However, just using $P(x)$ values is hard to distinguish in-degree or out-degree follows the power-law distribution from other distributions, like lognormal and exponential. CLAUSET[21] also proposed a goodness of fit method, which uses p-value, to validate whether a data set fits the power-law distribution better than others. Concretely, if and only if $p > 0.1$, we can get the conclusion that the power-law distribution is more reasonable on a data set.

Tab. 3 shows all the values of α and x_{min} for each CDG and FDG generated from 10 Java application systems.

Tab. 3 Value of α and $x_{\min}$ in all ten types of Java application networks

Network	CDG networks[5R]				CDG networks[3R]				FDG networks			
	α^{in}	$x^{in}_{\min}$	α^{out}	$x^{out}_{\min}$	α^{in}	$x^{in}_{\min}$	α^{out}	$x^{out}_{\min}$	α^{in}	$x^{in}_{\min}$	α^{out}	$x^{out}_{\min}$
GISToolkit	2.01	6	4.23	39	1.96	3	2.72	8	1.80	1	3.38	14
OpenGTS	2.01	27	2.08	20	2.02	8	2.26	8	2.50	2	3.84	3
OpenJUMP	2.09	12	4.33	42	2.19	8	3.41	16	2.22	2	4.25	6
dotCMS	1.81	6	2.19	35	2.02	24	1.96	5	3.36	6	2.72	6
infoGlue	1.81	5	2.03	11	1.88	1	2.25	6	1.79	3	2.02	4
OpenCms	1.90	11	2.34	36	1.94	6	2.35	17	1.84	4	2.29	16
eConf	3.03	9	3.56	10	2.70	5	3.38	5	3.10	1	2.82	1
a-LMS	1.96	8	2.17	9	1.65	1	2.19	5	2.80	2	4.75	2
OLMS	1.85	2	2.25	6	2.51	1	2.38	1	2.59	1	4.89	1
JSPWiki	1.61	1	3.31	33	1.91	3	2.30	4	2.80	4	2.70	1
JAMWiki	1.90	10	2.57	23	1.63	1	2.47	13	2.26	2	4.62	2
FitNesse	1.99	6	3.20	18	1.78	1	4.03	16	2.66	2	2.87	1
jforum	2.15	32	3.92	34	1.78	1	2.92	6	2.80	5	5.00	4
jGossip	1.95	4	3.05	6	2.13	1	2.82	4	2.61	1	5.00	4
Yazd	1.87	7	1.96	6	1.86	1	2.26	5	2.30	1	5.00	3
jPortlet	2.35	12	2.32	8	2.12	4	2.63	5	2.40	1	2.61	1
OpenPortal	2.21	2	2.28	2	2.52	2	3.10	3	2.26	1	2.83	2
Pluto	2.47	8	2.66	11	2.41	5	2.48	4	2.92	2	2.67	1
GatorMail	2.00	3	3.00	5	2.10	1	2.45	2	2.68	1	3.49	1
OlivaMail	2.34	10	2.67	7	3.51	4	2.05	1	2.38	1	2.37	1
yawebmail	3.35	9	2.19	3	1.81	1	2.78	4	3.03	2	3.32	2
FocusSNS	1.71	1	2.57	7	1.90	1	5.00	11	2.75	15	4.58	8
Pebble	1.73	1	2.79	14	1.89	3	2.71	3	2.49	5	4.03	3
Roller	1.84	4	2.61	20	1.77	1	2.41	5	2.36	2	4.54	5
ITracker	2.94	83	2.57	15	2.37	18	2.10	2	2.31	1	2.36	3
BugRat	1.42	1	2.56	21	2.05	5	2.77	8	2.13	2	2.10	1
Scarab	2.32	37	2.62	41	2.56	29	2.44	21	2.30	1	4.25	1
Hipergate	2.19	17	2.59	24	2.07	5	2.72	15	2.39	2	3.43	3
SourceTap	1.64	1	2.11	8	1.62	2	2.37	5	2.45	1	2.68	1
OpenCustomer	1.94	4	5.00	27	2.12	3	4.13	10	2.19	1	2.31	1

CDG networks[5R] are modeling networks constructed by our 5 relationship model, and CDG networks[3R] are relatively built up with 3 relationships (i.e. R_1, R_2 and R_3). FDG networks are the graphs for function dependencies. As we can see, in CDG networks[5R], α^{in} varies from 1.42 to 3.35 and α^{out} varies from 1.96 to 5.00. While in CDG networks[3R], α^{in} ranges in [1.62, 3.51] and α^{out} ranges in [1.96, 5.00]. In FDG networks, the range of α^{in} is from 1.79 to 3.36 and the range of α^{out} is from 2.02 to 5.00. It is obviously that α^{out} usually has broader and bigger values than α^{in}, which indicates some α^{out} may not follow the power-law distribution[21]. General speaking, if the network obeys power law, the value of α should locate in (2,3)[1,10,21]. So we should further detect the distribution of in-degree and out-degree by p-value of a goodness of fit method.

3 Experimental Result

We implement comprehensive experiments on 10 popular Java application systems††, which are listed in Tab. 1. Due to the characteristics of Java application systems, the ratio of edges to nodes in them is much higher than that in Java development tools[10,22].

3.1 Results and Discussion

Fig. 3 and Fig. 4 exhibits the in- and out-degree distributions computed on CDG networks[5R] and CDG networks[3R]. 5RID and 5ROD stand for the in- and out-degree distributions computed on the weighted and directed network with the 5 relation model. While 3RID and 3ROD represent the in- and out-degree distributions with 3 relationships. Fig. 5 shows the

†† http://www.open-open.com/code/.

function in-degree (abbreviated to FID) and out-degree (abbreviated to FOD) distribution calculated by our evaluation method.

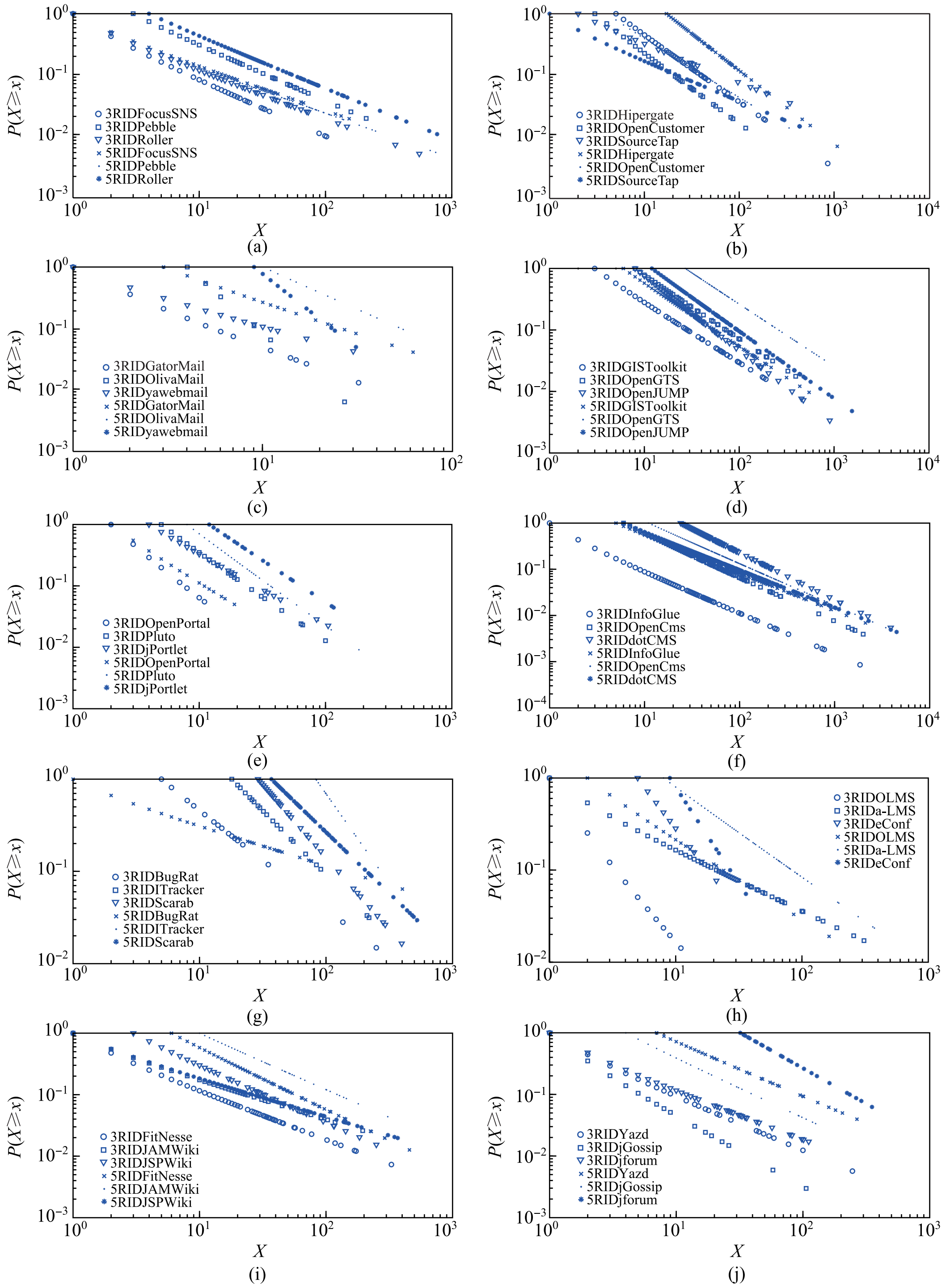


Fig. 3　In-degree distributions of CDG[3R] and CDG[5R] networks in all 10 types of systems
(a) Blog and SNS systems; (b) ERP and CRM systems; (c) WebMail systems; (d) GIS systems; (e) Portal systems; (f) CMS systems; (g) Bug tracking systems; (h) LMS systems; (i) Wiki systems; (j) Forum systems

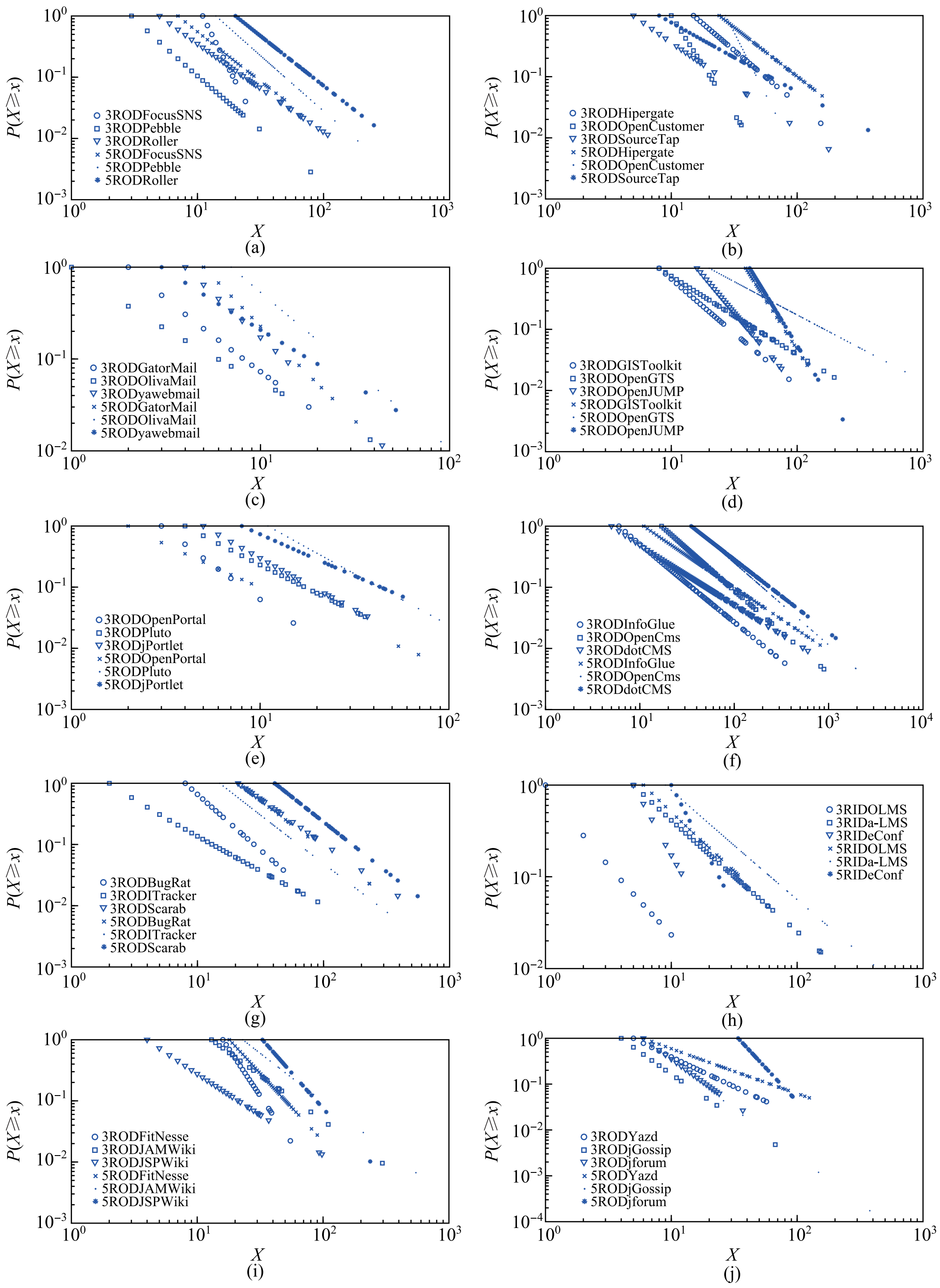


Fig. 4　Out-degree distributions of CDG[3R] and CDG[5R] networks in all 10 types of systems
(a) Blog and SNS systems; (b) ERP and CRM systems; (c) WebMail systems; (d) GIS systems; (e) Portal systems; (f) CMS systems; (g) Bug tracking systems; (h) LMS systems; (i) Wiki systems; (j) Forum systems

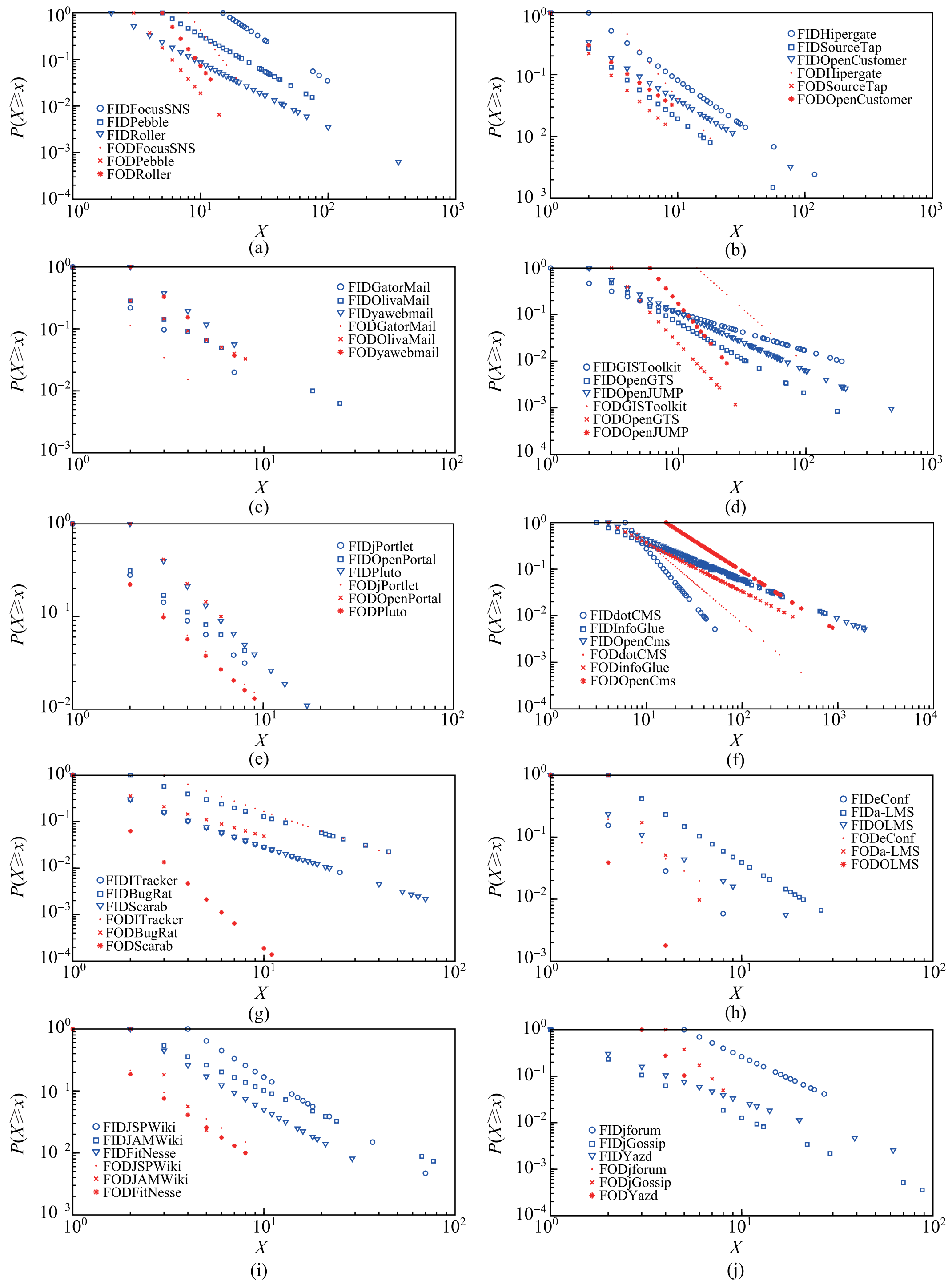


Fig. 5　In- and out-degree distributions of FDG networks in all 10 types of systems
(a) Blog and SNS systems; (b) ERP and CRM systems; (c) WebMail systems; (d) GIS systems; (e) Portal systems; (f) CMS systems; (g) Bug tracking systems; (h) LMS systems; (i) Wiki systems; (j) Forum systems

From Fig. 3 and Fig. 4, we can see that all the distribution curves are straight lines, which indicates the in- and out-degree probably follow power-law distributions[23]. However, as we talked about before, it is not enough to get this conclusion just by observing the figures, where the 3RID-line and 5RID-line of the same system substantially parallel to each other. Correspondingly, the 3ROD-line and 5ROD-line is also like that.

So we further compute the p-values to quantify the plausibility of power-law distribution, and Tab. 4 shows the p-values of all networks. We can see that the p-values generated on weighted and directed CDG network[5R] are different from the values generated on CDG networks[3R]. Now let's see these in- and out-degree whose p-values are greater than 0.1, which means they indeed follow power-law distributions. By contrast, there are more p-value^{in} values greater than 0.1 and less p-value^{out} smaller than 0.1 based on our modeling method (CDG networks[5R]). For FDG networks, there are only 1/6 of the p-value^{in} less than 0.1, and 7/15 of p-value^{out} less than 0.1. Apparently, most of the Java application systems have scale-free feature not only at class level but also at function level.

Tab. 4 p-values in all ten types of Java application networks

Network	CDG networks[5R]		CDG networks[3R]		FDG networks	
	p-value^{in}	p-value^{out}	p-value^{in}	p-value^{out}	p-value^{in}	p-value^{out}
GISToolkit	**0.070 8**	0.208 4	0.637 6	**0.001 2**	0.344 4	0.176 4
OpenGTS	0.641 4	**0.011 2**	0.352 0	0.813 6	0.543 6	**0.027 6**
OpenJUMP	0.407 2	0.766 4	0.928 4	**0.088 8**	0.741 2	0.560 8
dotCMS	**0.034 4**	**0.001 2**	0.664 0	**0.008 0**	0.234 0	0.168 0
infoGlue	0.218 8	0.175 6	0.794 0	**0.041 6**	0.326 0	**0.039 2**
OpenCms	0.232 8	0.144 4	0.289 2	0.745 2	0.127 6	0.566 8
eConf	0.476 4	0.756 8	0.477 2	0.199 2	0.339 6	0.832 4
a-LMS	**0.004 8**	0.205 6	**0.004 0**	0.258 0	0.558 8	**0.035 2**
OLMS	**0.020 8**	**0.055 6**	0.342 0	0.666 4	0.382 8	0.497 2
JSPWiki	**0.000 0**	0.593 6	0.595 6	**0.027 2**	0.342 4	**0.000 0**
JAMWiki	0.297 6	0.902 8	**0.000 4**	0.195 2	0.548 8	0.365 6
FitNesse	**0.027 6**	0.255 2	**0.019 2**	0.917 6	0.461 6	**0.000 0**
jforum	0.772 0	0.620 0	**0.000 0**	**0.084 4**	0.193 2	0.241 2
jGossip	0.305 2	**0.059 6**	**0.001 6**	0.860 8	**0.079 6**	0.692 8
Yazd	0.495 6	**0.060 8**	**0.053 2**	0.302 0	**0.014 4**	0.208 8
jPortlet	0.458 8	**0.017 2**	0.714 0	0.529 6	**0.003 2**	0.171 6
OpenPortal	0.188 0	**0.020 0**	**0.001 6**	0.710 8	0.360 8	0.252 4
Pluto	0.437 2	0.872 0	0.632 4	0.368 0	0.326 8	**0.000 0**
GatorMail	0.467 2	0.818 4	**0.016 4**	0.620 0	0.442 4	0.144 0
OlivaMail	0.503 2	0.800 4	0.110 8	**0.047 6**	0.512 0	0.142 0
yawebmail	0.658 0	0.665 6	**0.004 8**	0.562 0	0.549 2	**0.075 2**
FocusSNS	**0.001 6**	**0.042 8**	**0.080 4**	0.939 6	0.695 6	0.740 8
Pebble	**0.000 0**	0.613 6	0.643 2	**0.001 6**	0.264 0	**0.012 4**
Roller	**0.045 6**	0.353 2	**0.000 4**	0.666 8	0.916 0	**0.026 8**
ITracker	0.817 2	**0.010 4**	0.824 4	0.363 6	0.258 4	**0.041 6**
BugRat	**0.005 6**	0.463 6	0.228 8	0.772 8	0.436 4	**0.000 0**
Scarab	0.307 6	0.229 2	**0.001 6**	0.980 8	**0.000 0**	**0.000 0**
Hipergate	0.577 2	0.226 8	0.741 6	0.281 2	**0.068 0**	0.722 8
SourceTap	0.528 4	**0.001 2**	0.708 4	**0.044 8**	0.606 0	**0.000 0**
OpenCustomer	0.260 8	0.786 8	0.560 8	0.730 0	0.314 4	**0.000 0**

The comprehensive experiments show that our proposed method and model is appropriate to analyze the degree distributions and further detect whether a system has the scale-free feature of the complex network. If we get a scale-free network from a Java application system, we can conclude that it has robustness and fragility, which means the system is robust to random breakdown or fault[24-25] and fragile to malicious attacks. Through the analysis of the in-degree and out-degree for CDG and FDG networks, we can extract the most important nodes of each system and the relationships between the entities, which helps us to maintain our system and debug the errors.

4 Conclusion

In this paper, we conduct comprehensive analysis on both class and function dependence relationships in 10 types of Java application systems. First, we make use of our proposed method to transform each system network into weighted class dependence graph (CDG) and function dependence graph (FDG). We then further analyze the distributions of in- and out-degree of each graph to see whether they follow power-law distributions. Experimental results show that the in- and out-degree of these Java application systems mostly fit power-law distributions, and it indicates that our proposed modeling method is effective and accurate in analyzing degree distributions and suggests a possible way to assign dependence relationships different weights to do network analysis in the future work. From the perspective of industrial production, if the open source system has the power-law nature, which indicates it is robust yet fragile, our method has great significance to help developers to control the bug propagation and optimize performance of open source systems like Java application systems.

[References]

[1] CONCAS G, MARCHESI M, PINNA S, et al. Power-laws in a large object-oriented software system [J]. IEEE Trans Softw Eng, 2007, 33: 687-708.

[2] DE MOURA A P, LAI Y C, MOTTER A E. Signatures of small-world and scale-free properties in large computer programs [J]. Phys Rev E, 2003, 68: 017102. DOI: 1103/PhysRevE.68.017102.

[3] KOHRING G A. Complex dependencies in large software systems [J]. Adv Complex Syst, 2009, 12: 565-581.

[4] LABELLE N, WALLINGFORD E. Inter-package dependency networks in open-source software [J]. Computer Science, arXiv: cs/0411096v1.

[5] MAILLART T, SORNETTE D, SPAETH S, et al. Empirical tests of Zipf's law mechanism in open source linux distribution [J]. Phys Rev Lett, 2008, 101: 218-701.

[6] ZHENG X, ZENG D, LI H, et al. Analyzing open-source software systems as complex networks [J]. Physica A, 2008, 387: 6190-6200.

[7] HYLAND-WOOD D, CARRINGTON D, KAPLAN S. Scale-free nature of java software package, class and method collaboration graphs [C]//Submitted to the 5th International Symposium on Empirical Software Engineering. 2005.

[8] FORMAN I R, FORMAN N. Java Reflection in Action [M]. [S.l.]: Manning Publications, 2004: 121-142.

[9] GIULIO C, MICHELE M, SANDRO P, NICOLA S. On the suitability of Yule process to stochastically model some properties of object-oriented systems [J]. Physica A, 2006, 370: 817-831.

[10] GU Q, XIONG S J, CHEN D X. Correlations between characteristics of maximum influence and degree distributions in software networks [J]. Sci China Inf Sci, 2014, 57(7). DOI: 10.1007/s11432-013-5047-7.

[11] LI C F, LIU L Z, LI X Y. Software networks of Java class and application in fault localization [C]//Proceedings of the International Conference on Intelligent Systems Design and Engineering Application. 2012: 1117-1120.

[12] TETSUO T, TAKAKO N. Analysis of software evolution processes using statistical distribution models [C]//Proc International Workshop Principles of Software Evolution (IWPSE). 2002: 120-123.

[13] LI D Y, HAN Y N, HU J. Complex network thinking in software engineering [C]//Proceedings of the International Conference on Computer Science and Software Engineering, 2008, 1: 264-268.

[14] WHEELDON R, COURNSELL S. Power law distributions in class relationships [C]//Proceedings of the 3rd IEEE International Workshop Source Code Analysis and Manipulation. 2003: 45-54.

(下转第 70 页)