

# An Exploration of Power-law in Use-relation of Java Software Systems

Makoto Ichii<sup>†</sup>

Makoto Matsushita<sup>†</sup>

Katsuro Inoue<sup>†</sup>

<sup>†</sup>Graduate School of Information Science and Technology, Osaka University  
1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan

E-mail: {m-itii, matusita, inoue}@ist.osaka-u.ac.jp

## Abstract

*A software component graph, where a node represents a component and an edge represents a use-relation between components, is widely used for analysis methods of software engineering. It is said that a graph is characterized by its degree distribution. In this paper, we investigate software component graphs composed of Java classes, to seek whether the degree distribution follows so-called the power-law, which is a fundamental characteristic of various kinds of graphs in different fields. We found that the in-degree distribution follows the power-law and the out-degree distribution does not follow the power-law. In a software component graph with about 180 thousand components, just a few of the components have more than ten thousand in-degrees while most of the components have only one or zero in-degree.*

**Keywords:** Software Component Graph, Degree Distribution, Power-law, Scale-free Network

## 1 Introduction

Modern software systems are rarely developed from scratch, that is, developed reusing various software components. For example, thousands of standard library components are included in the Java development kit [4]; hundreds of open source software systems including libraries and application systems using the libraries on Apache software foundation [2], SourceForge [8] and so on. The structure of the software systems built from components is focused by some authors [14, 18, 22, 26, 29] because of its complex nature.

A software component graph (a component graph), which is a graph where a node represents a software component (a component) and an edge represents a use-relation between components, is widely used for the methods which support the various phases of software development. Using various types of component graphs, we can represent inherent characteristics of the static structure and dynamic

behavior of software. We may obtain the static characteristics of the software such as design structure by analyzing the impact relation of the software statically. We would obtain the dynamic characteristics of the software such as collaborations among objects by analyzing the object invocation dynamically.

In this paper, we investigate component graphs constructed by static analysis, which are widely used in various software engineering methods such as software component retrieval [12, 19], software measurement [16, 27], design recovery [11, 25, 28], and software modularization [20]. It is important to know the characteristic of component graphs for effective and efficient analysis; however, there are little researches on their characteristic.

We focus on whether the degree distribution of a component graph follows so-called power-law, intuitively meaning that very few nodes have extremely high degrees and most nodes have very low degrees. The degree distribution characterizes structure of a graph. The graphs whose degree distributions follow the power-law are known in various domains such as physics, social science, biology, information science [10, 13, 23]. Such graphs are also called as scale-free networks and attract researchers' interest because of its complex nature. If the degree distribution of a component graph followed the power-law, we could say not only that the fact will help to make software analysis methods more effective and efficient, but also that the analysis from new viewpoints based on these graph characteristics can be expected.

In this paper, we explore various component graphs based on the following questions:

**Question 1** *Do the in- and out-degree distributions of a component graph of a software system follow the power-law?*

There are some related works on the power-law of the distribution of a component graph [14, 18, 22, 29, 31], but the results may be different because we use different definition of a component graph from those related works.

**Question 2** *Do the in- and out-degree distributions of a component graph for multiple software systems follow the power-law?*

We will target on not only a single software system but also but also collection of software systems where a system uses other systems each other. In the related works, component graphs constructed from multiple software systems are not explored. The details about the difference between a single software system and multiple software systems are described in Section 4.2

**Question 3** *Do the in- and out-degree distributions of sub-graph of a component graph for software systems follow the power-law?*

A subset of components may be used for dependency analysis, but there are no related works explored on such subset.

**Question 4** *What aspects of components contribute to the power-law (or non-power-law) at the in- and/or out-degree distribution of a component graph?*

It is believed that the degree reflects some sort of aspects of a component structure. Therefore, exploring relationships between components and its degree helps to discuss on findings the above questions.

This paper is organized as follows: In Section 2, we describe related works where the power-law of component graph is focused on. In Section 3, we describe our definitions of component graph and a brief overview of power-law. In Section 4, we describe our experiments against above questions. In Section 6, we conclude this paper with our future works.

## 2 Related works

Valverde et al. [29] show that the power-law is observed in the degree distribution of component graphs, where nodes are classes and interfaces in Java class diagrams (such as JDK [4]) and edges are relations such as inheritance or dependency. They also show that small-worldness [30], a characteristic of graphs where the minimal length between edges is very small in spite of sparsity of the edges, appears in the component graphs. Additionally, they argue that such characteristics of component graphs emerge if the software systems are well designed considering reusability, understandability, and so on.

Myers [22] presents the power-law of the degree distribution, small-worldness and hierarchical structure in component graphs based on class diagrams. In his experiments, the in- and out-degrees are examined individually and it is shown that the in- and out-degrees both follow the power-laws with different parameters. He also proposes the gener-

ative model of graphs which have the same characteristics with component graphs.

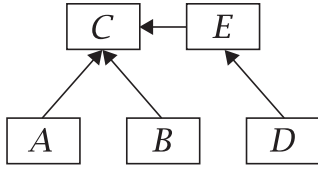
Concas et al. [18] measured the degrees of component graphs and CK metrics [17] of Smalltalk programs and Java programs. They observed the power-law or the lognormal distributions as the results.

Wheeldon et al. [31] show that the power-law is observed in several types of component graphs for Java programs in which edges represent different use-relation respectively.

Baxter et al. [14] found the power-law or the lognormal distributions in the class relationships in Java programs.

The works described above have similar approaches with ours, where the degree distribution of the component graphs which are constructed based on statical analysis of source files are explored. However, our work is different from them in the following points:

- We explore the component graphs based on multiple software systems and their subgraphs, which have not been explored by the other researchers. The component graphs of this type are also used by software engineering researches such as software component retrieval system [19]. By exploring the component graphs of this type, we can obtain the global nature of a component graph. Details of the component graphs which we investigate are described in Section 4.2.
  - Related to foregoing, we focus on acquiring general characteristic of the component graph. In addition to investigation of the power-lawness of component graphs, we examined correlation between degrees and software metrics (in Section 4.1) and discuss on the results from the viewpoint of the software practice (in Section 5) In contrast, in the related works, insights towards the factor behind the characteristics such as the power-law of the component graph.
  - The results in our experiments may differ from ones of the related works since our definition of component graph is different from any of the related works. The difference is twofold. First, in the related works, the use-relations as edges of a component graph are only “significant” types of use-relation, such as inheritance or field declaration, and ignores some types of use-relation such as method call or local variable declaration, which sometimes dominates use-relation among components. Second, some works acquires use-relations only based on lexical analysis of source code and text matching of component name, by which precise relationships are hard to be acquired.
- We construct component graphs using all types of use-relations which are analyzable statically for the component graph based on light semantic analysis. Our



**Figure 1. An example of component graph**

definition of component graph is described in Section 3.1 and the analysis method is described in Section 4.4.

Our exploration and the related works described above are based on static analysis. In contrast, Potanin et al. [26] shows that the degree distribution of component graph based on dynamic analysis of a software system in operation follows the power-law.

### 3 Component graph and Power-law

#### 3.1 Component graph

A **software component** generally represents a building unit of a software system such as a module, a function or a class in a broad sense. It also represents a software entity designed for reuse purpose in a restricted sense. We use the term software component in the broad sense and call software component as **component**.

A **use-relation** is an interaction between components in a software system, such as a method call or a field access.

A **software component graph** is a graph where a node represents a component and an edge represents a use-relation between components. We call a software component graph as a **component graph**. Figure 1 is an example of a component graph, where components *A*, *B*, and *E* use component *C*, and component *D* uses component *E*.

**In-degree** and **out-degree**, which we investigate in the experiments, mean the number of incoming edges (relations) to a node (component) and the number outgoing edges from a node respectively.

In this paper, a component graph is constructed with components and use-relations defined as follows:

**Component** A Java class or a Java interface extracted from the source files. Neither a class nor an interface contained in a binary library (“jar” file) is not treated as a component in this paper.

**Use-relation** Any of the following six relation types acquired by static analysis of the component source files. If a pair of components has any of the following relations, a directed edge is created between the corresponding nodes.

- A class or an interface *extends* another class or interface respectively.
- A class *implements* an interface.
- A class or an interface *declares a variable* (a local variable, a field, a parameter, or the return type of a method) of a class or an interface.
- A class *instantiates* a class object.
- A class *calls a method* of a class or an interface. If the callee method is an inherited method from a parent class (or interface) of the callee class (or interface), we interpret that the method of the parent class (or interface) is called.
- A class or an interface *refers to a field variable* of a class or an interface. If the referenced field is an inherited field from a parent class (or interface) of the referenced class (or interface), we interpret that the field of the parent class (or interface) is referred to.

#### 3.2 Power-law

We focus on whether the in- and out-degree distributions of a component graph follow the power-law or not. A graph with the power-law distribution are also called “scale-free network” and it can be seen on various domains such as coauthoring relationship among scientists [23], hyperlink relationship among WWW pages [10]. Power-law is also called as Pareto’s law or Zipf’s law [24].

The power-law, where the probability where a value occurs is proportional to the power of the value, is the distribution of the following form:

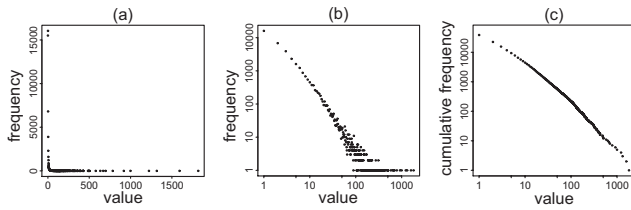
$$p(x) \sim x^{-\alpha} \quad (1)$$

The distribution which follows the power-law is plotted with double logarithmic axes because we cannot identify well the characteristics of the distribution with linear axes as shown in Figure 2 (a). Equation (1) can be transformed into Equation (2) and we can see that the plotted values form a straight line whose gradient is  $-\alpha$ .

$$\log p(x) \sim -\alpha \log x \quad (2)$$

When plotting real data, the cumulative frequency is plotted instead of the frequency since a plot of real data has jaggy as seen in Figure 2 (b). Equation (1) is transformed into the cumulative distribution function shown in Equation (3)<sup>1</sup>; we can see that the plotted values form a straight line which gradient is  $-(\alpha - 1)$  as shown in Figure 2 (c).

$$P(x) \sim x^{-(\alpha-1)} \quad (3)$$



**Figure 2. Plottings of the power-law**

In the experiments, we calculate the value of  $\alpha$  of Equation (1) as the characteristics value of the distribution by estimating the regression line of the cumulative frequency plot using double logarithmic axes. We also find the determination coefficient which was adjusted for the degree of freedom  $R^{*2}$ , which represents fitness of a regression model for data and runs from 0 to 1.

## 4 Experiments

### 4.1 Scheme of experiments

In order to answer to the questions in Section 1, the following experiments have been performed:

**Experiment 1** (Question 1) We examined component graphs for single software systems whether the in- and out-degree distributions of them follow power-law. The cumulative distribution is plotted on double logarithmic axes, and the gradient and the determination coefficient of the regression line are calculated to determine whether they are the power-law or not.

**Experiment 2** (Question 2) The same experiment as Experiment 1 was performed using the component graphs for multiple software systems.

**Experiment 3** (Question 3) The same experiments performed as Experiment 1 with component graphs for a subset of components of software systems.

**Experiment 4** (Question 4) We calculated spearman's rank correlation coefficient between the degree and metrics of the components. In addition, we list top ten components on the in- and out-degree to find characteristics which are not represented by the metrics used in this experiment.

### 4.2 Component sets

We prepare two types of component sets for experiments. For Experiment 1, we prepare component sets where com-

ponents are extracted from a single software system. Meanwhile we prepare component sets where components are extracted from multiple software systems for Experiment 2.

#### 4.2.1 Single software system

We prepared following four component sets based on a single software system respectively. Each of them includes the components which are acquired by analyzing the source files in the distribution package. The binary libraries are not included in the component set even if the distribution package contains binary libraries and the source files depend on them. The component sets varies in lines of code and application domain. Table 1 describes the sizes of the component sets.

**ANT** A component set of the distribution package of Apache Ant 1.6.2 [1], a Java build tool. This component set is the smallest in our experiments.

**JBOSS** A component set of the distribution package of JBoss Application Server 3.2.5 [5], a Java enterprise application server.

**JDK** A component set of JDK1.4, a standard library of Java. Use-relations based on foundational components are likely formed in this component set. JDK was also examined by the related works, and we can compare the results.

**ECLIPSE** A component set of the distribution package of Eclipse SDK 3.0.1 [3], a Java software development environment.

#### 4.2.2 Multiple software systems

The component sets based on multiple software systems are composed of the components acquired by analyzing the source files retrieved from the distribution packages and/or software repositories. Each of the component sets is constructed so that the component graph includes the use-relations that cross the border of the software systems.

**ASF** A component set contained in the software systems checked out from the CVS repository of the Apache Software Foundation [2] at June 2005. This component set does not include JDK. Each of the software systems is developed individually, and some software systems are developed as common libraries. Therefore, the use-relations based on such libraries are likely formed.

**SPARS\_DB** A component set contained in the database of SPARS-J [9] at June 2005. This includes JDK and many software systems checked out from open source repositories: the Apache Software Foundation (that

<sup>1</sup>See [24] for details of the transformation.

is, SPARS\_DB includes ASF), SourceForge.net [8], Eclipse, and NetBeans [6]. There are various components including libraries such as JDK, application software such as Eclipse, and small sample components. Because the similarity of the components in this component set is nothing or only using JDK if any, use-relations based on JDK are likely formed.

### 4.2.3 Subset

For Experiment 3, we use variations of subset of SPARS\_DB. The strategies to pick out subsets are as follows:

**Random** Pick out components randomly.

**Use-relation** Pick out the components which use a pivot component. The pivot components are randomly selected so that the number of components is about 10000, 1000 and 100.

**Keyword** Pick out the components which include a keyword in the source code. The keywords are randomly selected so that the number of components is about 10000, 1000, and 100.

Using above strategies, we prepare following component sets.

**RND10K, RND1K** Component sets picked out with the Random strategy. RND10K and RND1K contain 10,000 and 1,000 components respectively.

**REL10K, REL1K, REL1H** Component sets picked out with the Use-relation strategy. The pivot components are `java.util.HashMap`, `java.io.OutputStreamWriter`, and `java.awt.GraphicsEnvironment` respectively.

**KWD10K, KWD1K, KWD1H** Component set picked out with the Keyword strategy. The keywords are `getString`, `labels`, and `getsummary` respectively.

### 4.3 Metrics

For Experiment 4, we use following metrics whose values are measured using only the source code of a given component and which represents some sort of the quality.

**LOC** Non-comment source lines of code. LOC represents the size of the component.

**WMC1, WMC2** Variants of weighted methods per class [17]. WMC represents the complexity of the component. We use two kinds of definitions as the weight for the methods.

WMC1 uses a constant value, 1 as its weight. WMC1 is identical to the number of methods defined in the component and represents the size of the implemented functions in the component. On the other hand, WMC2 uses cyclomatic complexity as the weight. WMC2 represents the complexity of the implementation of the component because the cyclomatic complexity is measured based on branches and loops in the methods.

**LCOM** Lack of cohesion of methods. There are some different definitions of LCOM. We use LCOM5 [15], which is measured based on use of the fields by methods. LCOM5 runs from 0 to 1 and smaller value means the higher cohesion of the implementation.

We calculate correlation between LCOM and degree or another metrics using components where LCOM can measure; LCOM is defined only classes which have 1 or more implemented methods.

### 4.4 Analysis method

In order to analyze Java source code, construct its component graphs and measure metric values from them, we use SPARS-J [19], a software component retrieval system. Meanwhile, we use R [7], an environment for statistical computing and graphics, for the analysis of the degree distributions.

First, SPARS-J analyses Java source files and stores the classes and interfaces as components into its database. The use-relations between the components are also analyzed and stored into the database. SPARS-J can analyze all types of the use-relations described at Section 3.1. The metric values of the components are measured after finishing the relation analysis. Then, the component graph of a component set is constructed using the analysis result of SPARS-J. Finally, the in- and out-degree distributions of the component graph are plotted, and regression analysis is applied to them.

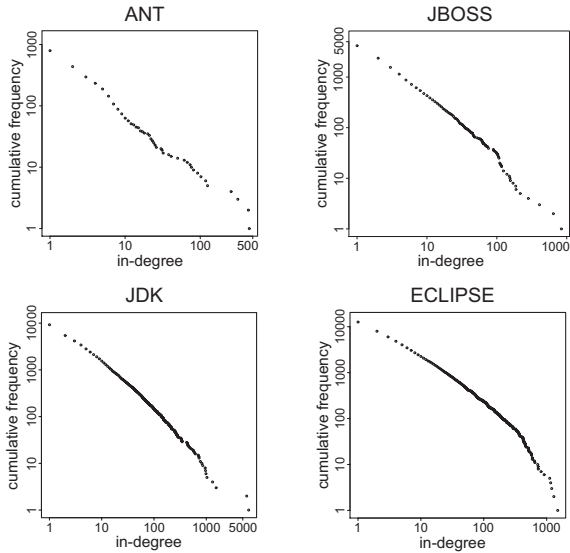
### 4.5 Results

The number of nodes (components) and the number of edges (use-relations) of the component graphs are shown in Table 1.

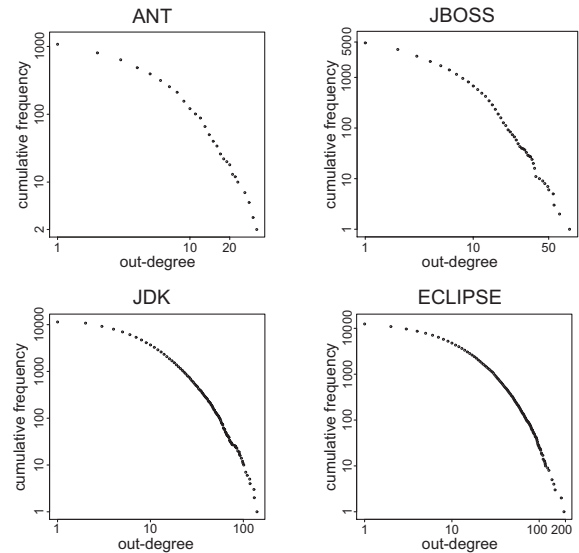
#### 4.5.1 Experiment 1

Figure 3 and 4 illustrate the cumulative frequency of the in- and out-degrees on the each of the single software systems respectively. Table 2 presents the characteristic values: the values of  $\alpha$  in Equation (1) and  $R^{*2}$ .

It is clear that the in-degree follows the power-law because the all plotted values form almost a straight line and each of the values of  $R^{*2}$  is much closed to 1.



**Figure 3. In-degree distribution (single software system)**



**Figure 4. Out-degree distribution (single software system)**

In contrast, the out-degree seems to incompletely follow the power-law. The plotted values form a straight line in the range of large values; however, we can see it forms a peak near the value of 1. Furthermore, the values of  $R^2$  are relatively small.

#### 4.5.2 Experiment 2

Figure 5 and 6 depict the cumulative frequency of the in- and out-degree respectively. The characteristic values are shown in Table 2.

We can see that both of ASF and SPARS\_DB have similar tendency with the results of Experiment 1. The in-degree distribution follows the power-law. The value of  $\alpha$  is about 2, which is near to the value of JDK and ECLIPSE. The out-degree distribution follows the power-law only in the range of large values.

#### 4.5.3 Experiment 3

Figure 7 and 8 show the cumulative frequency of the in- and out-degree respectively. Table 3 shows the characteristic values.

**Random** The in- and out-degree distribution of RND1K does not follow the power-law. The in-degree distribution of RND10K follows the power-law.

The in-degree distributions of the all component sets based on Keyword strategy follow the power-law and the values of  $\alpha$  are similar to SPARS\_DB, their original set. In

contrast, as regards the component sets based on Random strategy and Use-relation strategy, the in-degree distributions of the component sets which number of components are less than or equal to 1,000 do not follow the power-law clearly.

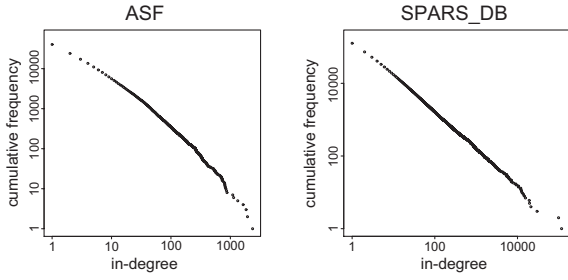
The out-degree distributions seem to have the similar characteristics as SPARS\_DB, except for RND1K.

#### 4.5.4 Experiment 4

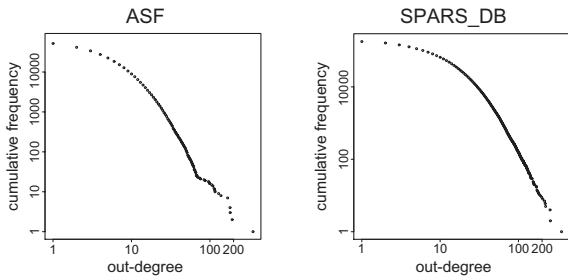
Table 4 describes correlation between the degree and the metrics. Table 5 and 6 show the top ten components on the in- and out-degree respectively. They are based on component set SPARS\_DB.

The in-degree does not have high correlation with any of the metrics in Table 4. Table 5 is filled with the foundational classes in JDK. For example, the `java.lang.String` class represents a character string and the `java.lang.Object` class is the parent class of all other classes. They are the roles which are given at design-phase. It is believed that components with a foundational or general role tend to have high in-degrees, while components with a specialized role tend to have small in-degrees.

The out-degree has a high correlation with LOC and WMC1, WMC2. Table 6 is filled with classes which are large/complex. None of them seems to be generated by code-generators. It seems to be reasonable that the out-degree has a correlation with the size and the complexity since the outgoing edges of a node in a component graph come from statements using other components in the source



**Figure 5. In-degree distribution (multiple software systems)**



**Figure 6. Out-degree distribution (multiple software systems)**

code corresponding to the node.

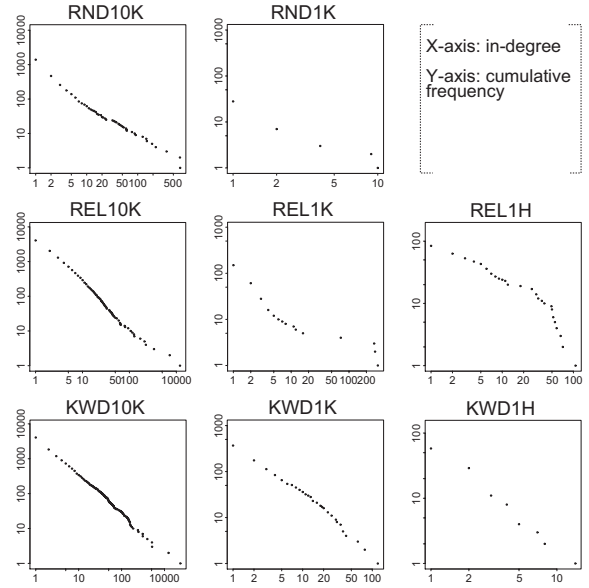
#### 4.6 Summary of the experiments

We summarize findings from these experiments by the questions mentioned at Section 1.

**Question 1** As the results of Experiment 1, with the component graphs with our definition, we found that the in-degree distribution follows the power-law almost ideally. We also found that the out-degree distribution does not follow the power-law, where the distribution has a peak in the range of small values while a straight line is observed in the range of larger values.

**Question 2** As the results of Experiment 2, we found the same results as Experiment 1 with the component graphs for multiple software systems: the in-degree distribution follows the power-law and the out-degree distribution does not follow the power-law.

**Question 3** As the results of Experiment 3, we found that the subsets whose components are picked out based on a keyword has similar characteristic with the superset: The



**Figure 7. In-degree distribution (subset)**

in-degree distribution follows the power-law with the similar parameters and the out-degree distribution partially follows the power-law.

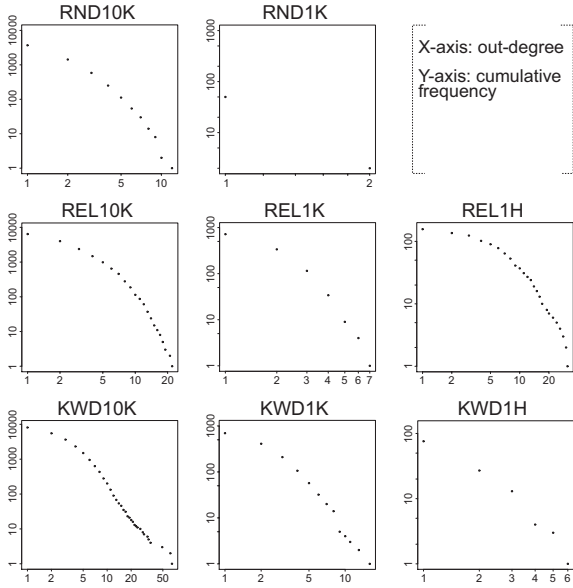
**Question 4** As the results of Experiment 4, we found that the in-degree relates to the roles of the components but it has low correlation with the size, the complexity, and the cohesion. We also found that the out-degree has higher correlation with the size and the complexity of the components.

## 5 Discussion

We found interesting results by these experiments. There is asymmetry between the in- and out-degree of the component graphs constructed with various use-relations, where the in-degree follows the power-law meanwhile the out-degree does not follow the power-law.

### 5.1 Power-law of in-degree

We found that the in-degree distribution of the component graph follows the power-law, and it does not simply mean that a few components are extremely used. There are some generative models of a graph whose degree distribution follows the power-law, and they are common in a point that a node with large degree tends to get the edge when a new edge is added to the graph. This means that the components (nodes) with large in-degree such as the components shown in Table 5 tend to get more in-degree when new components are added to the component graph. Therefore, it



**Figure 8. Out-degree distribution (subset)**

is likely that the set of components which have extremely large in-degree tends not to change even if new components are added to the graph.

Using this tendency, we believe that it is possible to observe the changes of software architecture along with software development. The set of components which have extremely large in-degree hardly changes even when the software development proceeds, that is, components are added to or removed from the component graph. But such set of components will change when large changes occur in the software such as the modification of the software architecture. An example of such changes is that a new commonly-used component is added to the software and many of the existing components in the software are modified to use the new component. Hence, we believe that it is possible to detect the changes of the software architecture, or to measure its stability by watching the changes of the set of the components which have the large in-degree.

## 5.2 Non power-law of out-degree

The out-degree distribution shown in the experiments have a “peak” in the small values, and “heavy-tail” in the large values. These characteristics are known as the ones of the lognormal distribution. However, the tails seems to be too “heavy” to resolve the distribution of the out-degree as lognormal distribution. These characteristics likely match with the double-pareto distribution [21], which is known as the distribution of the file sizes. Considering that the out-degree is shown to have high correlation with the LOC, it

is reasonable that the out-degree follows the same distribution with the file sizes. It is above the subject of this paper to determine which distribution the out-degree distribution follows; however, it is interesting topic and further exploration is needed.

The definition out-degree is identical with the CBO (Coupling between objects), one of the CK metrics suite [17]. It is said that classes with high values of CBO are complex and hard to maintenance and/or understand. Therefore, the experimental results on out-degree, where a few components have the higher out-degree, can be interpreted as that components with problems with maintenance exist in almost all software projects. It seems to be that the components which are to be refactored are neglected. This means that many software developers and managers lack the awareness for the design quality of software. However, it also seems that the components which have unavoidable complexity (i.e. can not be refactored) appears in the almost all software systems. This means that the techniques are needed to address the software complexity in addition to the ordinary object-oriented development approaches, on which the software systems we explored are based.

## 5.3 Asymmetry between in- and out-degree

Myers [22] found that the in- and out-degrees of the component graph follow the power-law and that their parameters are different each other. In our experiments, the different results were found: the in-degree follows the power-law but the out-degree incompletely follows the power-law. This difference likely comes from the difference of the definition of a component graph, in particular, the difference of definition of the use-relation as an edge: the use-relations include method calls and local variable declarations in our definition, which are ignored in definition in [22].

## 5.4 Power-law of subset collection

Some sort of scale-free networks have a property such that networks where some of those nodes are deleted still hold the scale-freeness. We found that the subsets of the component sets based on keywords have the same characteristic with the original component set, although the keywords have no explicit relation to the use-relations. The reason is that the related components such as the ones in the same module tend to have common words and have use-relation each other. Another reason is believed as that when a component includes a keyword as the name of itself, a method or etc., the users of the components also includes the keyword to reference the component.

## 6 Conclusions

In this paper, we have investigated the component graphs composed of Java components to seek whether the degree distribution follows the power-law.

As the results, we found that the in-degree distribution of the component graph follows the power-law almost ideally and that the out-degree distribution does not follow the power-law; out-degree distribution seems to follow power-law-like distribution such as lognormal distribution. We also found similar result in component graphs for multiple software systems and some sort of its subsets. Additionally, we found that the in-degree is related with the role of each component and the out-degree is related with the size and the complexity of each component. We believe that these results help improvement of the software analysis methods.

Future works are to explore other sort of component graphs such as ones based on other types of use-relation and to discuss about applying to these results for analysis methods of software engineering.

**Acknowledgements** This work has been conducted as parts of EASE Project, Comprehensive Development of e-Society Foundation Software Program, Grant-in-Aid for Exploratory Research(186500006), and The 21st Century Center of Excellence Program, all supported by Ministry of Education, Culture, Sports, Science and Technology of Japan.

## References

- [1] Apache Ant. <http://ant.apache.org/>.
- [2] The Apache Software Foundation. <http://www.apache.org/>.
- [3] Eclipse. <http://www.eclipse.org/>.
- [4] Java technology. <http://java.sun.com/>.
- [5] JBoss. <http://www.jboss.org/>.
- [6] NetBeans. <http://www.netbeans.org/>.
- [7] The R project. <http://www.r-project.org/>.
- [8] SourceForge. <http://sourceforge.net>.
- [9] SPARS-J. <http://demo.spars.info/>.
- [10] R. Albert, H. Jeong, and A.-L. Barabási. Diameter of the world-wide web. *Nature*, 401(6749):130–131, Sept. 1999.
- [11] G. Antoniol, R. Fiutem, and L. Cristoforetti. Design pattern recovery in object-oriented software. In *Proc. 6th Int'l Conf. Programming Comprehension (IWPC'98)*, pages 153–160, June 1998.
- [12] S. Bajracharya, T. Ngo, E. Linstead, Y. Dou, P. Rigor, P. Baldi, and C. Lopes. Sourcerer: A search engine for open source code supporting structure-based search. In *Proc. Int'l Conf. Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'06)*, pages 25–26, Oct. 2006.
- [13] A.-L. Barabási. *Linked: The New Science of Networks*. Perseus Books Group, 2002.
- [14] G. Baxter, M. Frean, J. Noble, M. Rickerby, H. Smith, M. Visser, H. Melton, and E. Tempero. Understanding the shape of java software. In *Proc. Int'l Conf. Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'06)*, pages 397–412, Oct. 2006.
- [15] L. C. Briand, J. W. Daly, and J. Wust. A unified framework for cohesion measurement in object-oriented systems. *Empirical Software Eng.*, 3(1):650–117, 1998.
- [16] A. Chatzigeorgiou, S. Xanthos, and G. Stephanides. Evaluating object-oriented designs with link analysis. In *Proc. 26th Int'l Conf. Software Eng. (ICSE'04)*, pages 533–542, 2004.
- [17] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Software Eng.*, 20(6):476–493, June 1994.
- [18] G. Concas, M. Marchesi, S. Pinna, and N. Serra. Power-laws in a large object-oriented software system. *IEEE Trans. Software Eng.*, 33(10):687–708, Oct. 2007.
- [19] K. Inoue, R. Yokomori, T. Yamamoto, M. Matsushita, and S. Kusumoto. Ranking significance of software components based on use relations. *IEEE Trans. Software Eng.*, 31(3):213–225, Mar. 2005.
- [20] B. S. Mitchell and S. Mancoridis. On the automatic modularization of software systems using the Bunch tool. *IEEE Trans. Software Eng.*, 32(3):193–208, Mar. 2006.
- [21] M. Mitzenmacher. Dynamic models for file sizes and double pareto distributions. *Internet Math.*, 1(3):305–333, 2003.
- [22] C. R. Myers. Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs. *Physical Rev. E*, 68(4):046116, 2003.
- [23] M. E. J. Newman. The structure of scientific collaboration networks. *Proc. Nat'l Academy of Sciences of the USA*, 98(2):409–415, 2001.
- [24] M. E. J. Newman. Power laws, pareto distributions and zipf's law. *Contemporary Physics*, 46:323–351, 2005.
- [25] J. Niere, W. Schäfer, J. P. Wadsack, L. Wendehals, and J. Welsh. Towards pattern-based design recovery. In *Proc. 24th Int'l Conf. Software Eng. (ICSE'02)*, pages 338–348, May 2002.
- [26] A. Potanin, J. Noble, and M. Frean. Scale-free geometry in oo programs. *Comm. ACM*, 48(5):99–103, May 2005.
- [27] S. Sarkar, G. M. Rama, and A. C. Kak. API-based and information-theoretic metrics for measuring the quality of software modularization. *IEEE Trans. Software Eng.*, 33(1):14–32, Jan. 2007.
- [28] K. Sartipi and K. Kontogiannis. On modeling software architecture recovery as graph matching. In *Proc. IEEE Int'l Conf. Software Maintenance (ICSM'03)*, pages 224–234, Sept. 2003.
- [29] S. Valverde, R. Ferrer-Cancho, and R. V. Solé. Scale-free networks from optimal design. *Europhysics Letters*, 60(4):512–517, 2002.
- [30] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, June 1998.
- [31] R. Wheeldon and S. Council. Power law distributions in class relationships. In *Proc. 3rd IEEE Int'l Workshop on Source Code Analysis and Manipulation (SCAM'03)*, pages 45–57, Sept. 2003.

**Table 1. Component sets for the experiments**

	# of Nodes	# of Edges	LOC
ANT	1,260	4,995	95K
JBOSS	5,752	23,636	424K
JDK	11,556	107,198	1.1M
ECLIPSE	13,941	140,678	1.3M
ASF	59,486	303,755	4.5M
SPARS_DB	180,637	1,808,982	14M
RND10K	10,000	6,184	780K
RND1K	1,000	52	80K
REL10K	9,286	17,201	2.1M
REL1K	972	1,218	250K
REL1H	163	1,086	76M
KWD10K	8,938	24,317	1.6M
KWD1K	1,002	1,564	290K
KWD1H	129	124	28K

**Table 2. Characteristic values of the degree distributions**

		$\alpha$		$R^{*2}$
In	ANT	2.01	$\pm$ 0.0195	0.984
	JBOSS	2.27	$\pm$ 0.020	0.978
	JDK	2.11	$\pm$ 0.00816	0.987
	ECLIPSE	2.19	$\pm$ 0.0163	0.955
	ASF	2.37	$\pm$ 0.0109	0.981
	SPARS_DB	2.02	$\pm$ 0.00145	0.999
Out	ANT	2.92	$\pm$ 0.143	0.872
	JBOSS	3.17	$\pm$ 0.104	0.905
	JDK	3.10	$\pm$ 0.0818	0.876
	ECLIPSE	3.03	$\pm$ 0.0770	0.856
	ASF	3.41	$\pm$ 0.0637	0.942
	SPARS_DB	3.66	$\pm$ 0.0693	0.903

**Table 3. Characteristic values of the in-degree distributions - Subset**

	$\alpha$		$R^{*2}$
RND10K	1.94	$\pm$ 0.0210	0.979
RND1K	2.27	$\pm$ 0.177	0.926
REL10K	2.30	$\pm$ 0.0203	0.986
REL1K	1.63	$\pm$ 0.0815	0.806
REL1H	1.83	$\pm$ 0.0646	0.867
KWD10K	2.12	$\pm$ 0.00927	0.993
KWD1K	2.21	$\pm$ 0.0332	0.980
KWD1H	2.62	$\pm$ 0.0805	0.983

**Table 4. Correlation between the degree and the metrics**

	Out	LOC	WMC1	WMC2	LCOM
In	0.002	0.070	0.239	0.075	0.118
Out	-	0.824	0.641	0.751	0.399
LOC	-	-	0.793	0.816	0.462
WMC1	-	-	-	0.567	0.494
WMC2	-	-	-	-	0.332

**Table 5. Top ten components (in-degree)**

Name	LOC	In	Out
java.lang.String	675	116,239	21
java.lang.Object	35	98,261	4
java.lang.Class	605	29,682	41
java.lang.Exception	15	21,046	2
java.lang.Throwable	136	19,519	12
java.lang.System	170	19,175	27
java.util.Iterator	5	15,522	1
java.util.List	27	14,462	4
java.util.ArrayList	200	13,656	19
java.lang.Integer	285	12,736	9

**Table 6. Top ten components (out-degree)**

Name	LOC	In	Out
org.apache...FunctionEval	364	1	354
org.jgraph.GPGraphpad	2,196	130	255
com.jgraph.GPGraphpad	2,200	131	253
org.jgraph.GPGraphpad	542	209	252
org.eclipse...ASTConverter	4,520	3	223
org.eclipse...JavaEditor	1,368	115	220
net.sourceforge...GanttProject	3,055	98	216
it.businesslogic...MainFrame	7,177	46	204
org...InstConstraintVisitor	1,626	3	197
org...ASTInstructionCompiler	2,449	1	189