Hindawi
Mathematical Problems in Engineering
Volume 2020, Article ID 3428604, 20 pages
https://doi.org/10.1155/2020/3428604

Research Article

Method of Coupling Metrics for Object-Oriented Software System Based on CSBG Approach

Aihua Gu,[1] **Lu Li**,[1] **Shujun Li**,[1] **Qifeng Xun**,[1] **Jian Dong**,[1] **and Jianhong Lin**[2]

[1]*School of Information Engineering, Yancheng Teachers University, Yancheng 224002, China*
[2]*Zhejiang Ponshine Information Technology Co., Ltd., Hangzhou 310012, China*

Correspondence should be addressed to Aihua Gu; guaihua1978@163.com

Received 2 November 2019; Accepted 5 February 2020; Published 19 March 2020

Guest Editor: Weifeng Pan

Context. Coupling between classes is an important metric for software complexity in software systems. *Objective*. In order to overcome the shortcomings of the existing coupling methods and fully investigate the weighted coupling of classes in different cases in large-scale software systems, this study analyzed the relationship between classes at package level, class level, and method level. *Method*. The software system is considered as a set of special bipartite graphs in complex networks, and an effective method for coupling measurement is proposed as well. Furthermore, this method is theoretically proved to satisfy the mathematical properties of coupling measurement, leading to overcome the disadvantages of the majority of existing methods. In addition, it was revealed that the proposed method was efficient according to the analyses of existing methods for coupling measurement. Eventually, an algorithm was designed and a program was developed to calculate coupling between classes in three open-source software systems. *Results*. The results indicated the scale-free characteristic of complex networks in the statistical data. Additionally, the calculated power-law value was used as a metric for coupling measurement, so as to calculate coupling of the three open-source software. It indicated that coupling degrees of the open-source software systems contained a certain impact on evaluation of software complexity. *Conclusions*. It indicated that coupling degrees of the open-source software systems contained a certain impact on evaluation of software complexity. Moreover, statistical characteristics of some complex networks provided a reliable reference for further in-depth study of coupling. The empirical evidence showed that within a certain range, reducing the coupling was helpful to attenuate the complexity of the software, while excessively blindly pursuit of low coupling increases the complexity of software systems.

1. Introduction

Coupling refers to the degree of interdependence between software modules; a measure of how closely connected two routines or modules are [1]; and the strength of the relationships between modules. Structured design, including cohesion and coupling, was published in an article by Stevens et al. and a book by Stevens et al. [2, 3], and the latter subsequently became standard terms. Coupling is considered as a double-edged sword in object-oriented programming. On the one hand, object-oriented software development (OOSD) includes object-oriented requirement analysis, as well as object-oriented design. OOSD is a practical method of developing a software system which focuses on the objects of a problem throughout development. Interactions between objects reflect the interdependence between objects. If objects are isolated, then the software system can only achieve simple functions. However, objects are equivalent to cells in human body. If cells are completely isolated from human body, they basically do not play any significant role, reflecting that functions of a software system require a tight coupling between objects. On the other hand, tight coupling between objects would lead to a water-wave effect, meaning that changes in one object may result in further changes in other objects. The most terrible case is that there is a possibility of "avalanche" effect, which may affect the whole system, leading to a sharp decline in the testability, understandability, reliability, and maintainability

of the system. Therefore, it is expected that classes are loosely coupled in terms of software design. A system can be tightly coupled in one aspect while being loosely coupled in another. However, software developers mainly prefer to develop those systems that are as loosely coupled as possible; thus, design, testing, and maintenance of the system would be relatively independent and more reasonable. Moreover, a decrease may be observed in the possibility that errors propagate between modules if there are few connections between modules [4]. Coupling has been widely used in evaluation of the degree of failure in classification [5–7], effective analysis [8, 9], and design pattern [10] of software systems.

The present article has the following organization: Section 2 summarizes the materials and methods. Section 3 shows the results. Section 4 provides a conclusion and suggests perspectives.

2. Materials and Methods

2.1. Methods. Currently, the methods for coupling measurement of object-oriented software systems are mainly structure-oriented measurement methods (Tables 1 and 2) [8, 11–15].

Comparative analysis of typical methods is shown in Tables 1–3, indicating that

(1) Methods for coupling measurement are mainly based on method invocation between classes.

(2) Calculation of coupling strength is defined as the degree of method invocation, which is weighted coupling.

(3) A small number of methods use fan-in and fan-out as metrics.

(4) Inheritance is dominant.

(5) Few methods use static method invocation, system measurement, and package-level metrics.

In addition to the abovementioned structural information methods, some scholars have recently used dynamic information methods [17, 18], semantic information methods [19–21], and logical information methods [22, 23]. Based on the results of previous studies, methods of coupling measurement cover the following cases:

(1) The DCMs are more accurate than that of structural information methods, while it seems to be difficult in the measurement of coupling metrics. However, structural information methods are more intuitive and easier to be perceived compared with semantic information and logical information methods.

(2) At present, the majority of the traditional structural information methods analyze coupling based on the degree of connecting edges between classes and mainly focus on complexity between classes and emphasize more on measurement from a local fine-grained aspect. Moreover, these degrees of connecting edges only consider a certain or a limited aspects of software engineering. Therefore, these methods contain some limitations, which cannot properly satisfy the requirement of an effective coupling measurement in complex software systems.

(3) Although a number of coupling measurement methods analyze network relationship from overall and macro perspectives based on graph theory, the majority of measurement metrics mainly use classes, packages, or methods as nodes to construct some undirected, directed, unweighted, or weighted network models. Moreover, they ignore a complex relationship of object-oriented characteristics between different classes. Some methods have not been theoretically verified for developing the mathematical characteristics of the measurement metrics.

The process of establishment of an effective method for coupling measurement between classes in a software system is determined by the following two aspects: reasonable measurement metrics and theoretical support of measurement metrics [24, 25]. Briand et al. mathematically analyzed measurement metrics of the software system and presented a robust theoretical support for the measurement metrics [4, 26, 27]. Many of complex networks have been shown to share the features such as "scale-free" and "small world" [28, 29]. Pan et al. revealed many physics-like laws in software systems from a complex network perspective recently [30, 31]. Studies on complex networks and software engineering revealed that class-level, method-level, and package-level diagrams of a software system could show the characteristics of "scale-free" and "small world," which provided a novel perspective for finding more reasonable measurement metrics [32–34]. Complex network theories were applied to measure software [35, 36], identify key software elements [37], and cluster Web services [38–39]. Researchers have found that many real networks have the bipartite graph characteristics of complex networks [40–45]. With combination of package level, class level, and method level, this study analyzed a complex relationship between classes in the same layer and all layers of a package and proposed a method for coupling metrics for object-oriented systems based on bipartite graph of complex networks, named here CSBG, and object-oriented software systems were expressed as a set of special bipartite graphs.

2.2. Problem Description. In this study, a statistical method for complex networks was used to analyze the degree of fan-out and the heterogeneity of classes at the same layer and all layers of a package, in addition to the calculation of coupling of software systems.

2.2.1. Relationship between Classes

Definition 1. ASS relationship (association and aggregation).

Association means which/how classes interact with each other, and association can be represented by a line between these classes with an arrow indicating the navigation

Table 1: The first part of existing methods for coupling measurement.

Method	Description
CBO [11]	$\mathrm{CBO}(c) = \lvert\{d \in C - \{c\} \mid \mathrm{uses}(c,d) \vee \mathrm{uses}(c,d)\}\rvert$; the metric is 1, if method in one class invokes other classes or is attributed to another class, otherwise it is 0
CBO′ [12]	$\mathrm{CBO}'(c) = \lvert\{d \in C - (\{c \cup \mathrm{Ancestors}(C)\}) \mid \mathrm{uses}(c,d) \vee \mathrm{uses}(c,d)\}\rvert$; this is similar to CBO method; however, that does not consider inheritance
RFC [13]	$\mathrm{RFC}(c) = \mathrm{RFC}_1(c)$, which is used for calculating the number of methods responding to an object's message
RFCα [13]	$R_0(c) = M(c)$, $R_{i+1}(c) = \cup_{m \in R_i(c)} \mathrm{PIM}(m)$, that is, a set of polymorphic methods invoked by functions in set $R_i(c)$; then $\mathrm{RFC}_\alpha(c) = \lvert\cup_{i=0}^{\alpha} R_i(c)\rvert$ with $\alpha = 1, 2, 3, \ldots$,
RFC′ [13]	$\mathrm{RFC}'(c) = \mathrm{RFC}_\infty(c)$
MPC [13]	$\mathrm{MPC}(c) = \sum_{m \in M_I(c)} \sum_{m' \in \mathrm{SIM}(m) - M_I(c)} \mathrm{NSI}(m, m')$; this calculates the number of static method invocation of classes
DAC [14]	$\mathrm{DAC}(c) = \lvert\{a \mid a \in A_I(c) \vee T(a) \in C\}\rvert$
DAC′ [14]	$\mathrm{DAC}'(c) = \lvert\{T(a) \mid a \in A_I(c) \vee T(a) \in C\}\rvert$; this formula is similar to DAC; however, if there is a relationship between classes, the metric is 1, otherwise the metric is 0

Table 2: The second part of existing methods for coupling measurement.

Method	Description
COF [14]	$\mathrm{COF}(C) = (\sum_{c \in C} \lvert\{d \mid d \in C - \{c\} \cup \mathrm{Ancestors}(c) \wedge \mathrm{uses}(c,d)\}\rvert)/\lvert C\rvert^2 - \lvert C\rvert - 2\sum_{c \in C} \mathrm{Descendent}(c)$
ICP [14]	This method calculates the parameters invoked by the method in a weighted class
IH-ICP [14]	This is similar to ICP, however, that only considers inheritance
NIH-ICP [14]	This is similar to ICP, however, that does not consider inheritance
SIMAS [8]	This method calculates the number of direct or indirect invocations between static methods of two different classes
PIM [8]	This method calculates the number of invocations in class C of methods in class D, and polymorphism is considered
PIMAS [8]	This method calculates the number of direct or indirect invocations between class methods, and polymorphism is taken into account
INAG [8]	The metric is 1 if there is an indirect aggregation between two classes; otherwise, the metric is 0
ACAIC [15]	$\mathrm{ACAIC}(c) = \sum_{d \in \mathrm{Ancestors}(c)} \mathrm{CA}(c,d)$; this calculates the number of out-degrees between one class and attributes of another classes in two classes with inheritance
OCAIC [15]	$\mathrm{OCAIC}(c) = \sum_{d \in \mathrm{Others}(c) \cup \mathrm{Friends}(c)} \mathrm{CA}(c,d)$; it calculates the number of out-degrees between one class and attributes of another class in two classes without inheritance
ACMIC [15]	$\mathrm{ACMIC}(c) = \sum_{d \in \mathrm{Ancestors}(c)} \mathrm{CA}(c,d)$; it calculates the number of out-degrees between one class and methods of another class in two classes with inheritance
OCMIC [15]	$\mathrm{OCMIC}(c) = \sum_{d \in \mathrm{Others}(c) \cup \mathrm{Friends}(c)} \mathrm{CA}(c,d)$; it calculates the number of out-degrees between one class and methods of another class in two classes without inheritance
AMMIC [15]	$\mathrm{AMMIC}(c) = \sum_{d \in \mathrm{Ancestors}(c)} \mathrm{MM}(c,d)$; it calculates the number of out-degrees for methods between two classes with inheritance
OMMIC [15]	$\mathrm{OMMIC}(c) = \sum_{d \in \mathrm{Others}(c) \cup \mathrm{Friends}(c)} \mathrm{MM}(c,d)$
ICF, FCF [16]	$\mathrm{ICF}_i = \sum_{k=1}^{n} I(k,i)\mathrm{ICF}_k$, $I(i,j) = e(i,j)/\sum_{k=1}^{n} e(i,k)$, $\mathrm{FCF}_i = \sum_{k=1}^{n} F(k,i)\mathrm{FCF}_k$, $I(i,j) = e(i,j)/\sum_{k=1}^{n} e(k,j)$

direction. Aggregation implies that one class exists in another class in the form of attribute.

Definition 2. DEP relationship (dependency)

DEP_*D*: dynamic dependency refers to an instance method in a class that invokes methods and attributes in another class.

DEP_*S*: static dependency refers to static methods in a class invokes methods and attributes in another class.

Definition 3. GEN (generalization)

One class inherits with another class, or one class implements interfaces with another class, or that of an abstract class.

2.2.2. Definition of Package Hierarchy. Packages of an object-oriented software system include classes and subpackages in the current hierarchy, and these subpackages contain classes in the current hierarchy and their subpackages. Software systems can actually be considered to be a tree hierarchical structure composed of packages.

t layer of a package is defined as $p^t \leq E^{t+1}, R^{t+1} >$. E^{t+1} represents a set of classes in the t layer, while this layer does not contain subpackages of this layer. R^{t+1} represents class relationship in the t layer, that is, $R^{t+1} \subseteq E^{t+1} \times E^{t+1}$.

$S\_\mathrm{layer}(i)$ is defined as a set of weighted fan-out of C_i at the t layer of the package, that is, $S\_\mathrm{layer}(i) \subseteq R^{t+1}$. $S\_\mathrm{all}(i)$ is the set of weighted cross-package fan-out of C_i, that is, $S\_\mathrm{all}(i) \subseteq R^1 \cup R^2 \cdots R^t \cdots$.

2.3. CSBG for Coupling Measurement. Software stability and modularity could be measured based on complex network theories. In this study, software systems can be expressed as a set of bipartite graphs that use nodes as classes, and ASS as well as DEP are the edges constituted by attributes of the class with those of another class based on complex network theories. However, GEN is a direct connection between

Table 3: Comparative analysis for the typical methods of coupling measurement.

Method	Type	Strength	Fan-out/ fan-in	Indirect coupling	Inheritance	Weighted	Static invocation	System metric	Package level
CBO	Method invocation, attribute reference	#coupled classed	Both	No	Both	No	No		
CBO′					Non-inh.-based	No	No		
RFC	Method invocation	#methods invoked	Import	No	Both	Yes	No	No	
RFCα				Yes		Yes			
RFC′				Yes		Yes			
MPC	Method invocation	#methods invocations	Import	No	Both	Yes	Yes		
DAC	Type of attribute	#attributes	Import	No	Both	Yes	No		
DAC′		#distinct types				No	No		
COF	Method invocation, attribute reference	#coupled classed	Both	No	Non-inh.-based	Yes	No	Yes	
ICP	Method invocation	#methods invocations, #parameters passed	Import	No	Both	Yes	No		No
IH-ICP					inh.-based	Yes	No		
NIH-ICP					Non-inh.-based	Yes	No		
SIMAS	Method invocation	#methods invocations	Import	Yes	Both	Yes	Yes		
PIM	Method invocation	#methods invocations		No	Both	Yes	No		
PIMAS	Method invocation	#methods invocations		Yes	Both	Yes	No		
INAG	Type of attribute	#attributes		Yes	Both	No	No		
ACAIC	Type of attribute	#attributes		No	inh.-based	Yes	No		
OCAIC					Non-inh.-based	Yes			
ACMIC	Type of parameter	#of parameters	Import	No	inh.-based	Yes	No		
OCMIC					Non-inh.-based	Yes			
AMMC	Method invocation	#method invocations	Import	No	inh.-based	Yes	No	No	No
OMMC					Non-inh.-based	Yes			
ICF	Method invocation, attribute reference	#method invocations	Import	No	Both	Yes	No	No	No
FCF			Export			Yes			

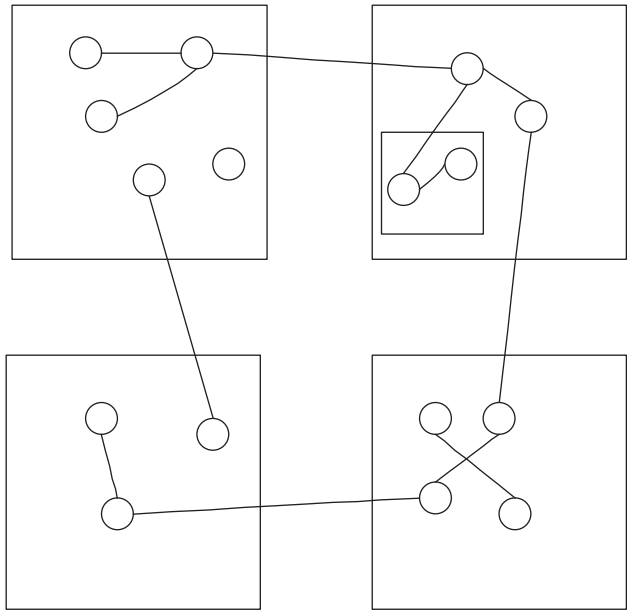

Figure 1: Illustration of a software system network composed of a set of special bipartite graphs (the large squares represent packages, and the 4 packages are at the same layer. The small squares show subpackages in the package. The circles denote classes in the package, and edges represent relationship between classes).

classes. Therefore, object-oriented software systems are taken into account as a set of special bipartite graphs constituted by classes in the package, as shown in Figure 1. There are defects in the coupling metrics containing the two metrics of fan-in and fan-out, because their total number is equal in a software system [46]. Therefore, this study only analyzed fan-out metric. The coupling strength between classes is correlated with the complexity of information exchange between modules. The more complex the information interaction (such as CBO), the tighter the coupling [47]. Coupling measurement metrics refer to the weighted fan-out of classes in special bipartite graphs. In these special bipartite graphs, classes associate with a class that may be at the same layer of the same package or at different layers of the package. Therefore, this study analyzed degrees of fan-out for classes in the same layer and all layers of the package. Moreover, heterogeneity of the abovementioned weighted out-degree was analyzed.

2.3.1. Demonstration of CSBG for Coupling Measurement. The detailed scheme proposed here is explained in the following, as illustrated in Figure 2:

(1) The object-orient software systems are constructed as directed weighted network graphs, and classes and relationship between classes are shown as nodes and edges, respectively.

(2) The package level, class level, and method level are combined to construct special weighted bipartite graphs between classes, aiming to make preparation for calculating the weighted out-degree of classes at the same layer of the package (see step 3, $(|\sum S\_\text{layer}(i)|)$), and the weighted out-degree $((|\sum S\_\text{all}(i)|))$ of all classes with classes across layers of the package (see step 3).

(3) ASS_layer, DEP_D_layer, DEP_S_layer, and GEN_layer at the same layer of the package were calculated. S_layer is determined by adding the weights of x_1, x_2, x_3, and x_4 to the four mentioned metrics, respectively, in order to calculate ASS_all, DEP_D_all, DEP_S_all, and GEN_all between classes across different layers of the package. Then, weights of x_5, x_6, x_7, and x_8 were added to these four metrics to determine S_all:

$$\begin{aligned}\left|\sum_{i=1}^{n} S\_\text{layer}(i)\right| &= \left[\left|\sum_{i=1}^{n} \text{ASS\_layer}(i)\right|, \left|\sum_{i=1}^{n} \text{DEP\_}D\text{\_layer}(i)\right|, \left|\sum_{i=1}^{n} \text{DEP\_}S\text{\_layer}(i)\right|, \left|\sum_{i=1}^{n} \text{GEN\_layer}(i)\right|\right] \times [x_1, x_2, x_3, x_4]^T, \\ \left|\sum_{i=1}^{n} S\_\text{all}(i)\right| &= \left[\left|\sum_{i=1}^{n} \text{ASS\_all}(i)\right|, \left|\sum_{i=1}^{n} \text{DEP\_}D\text{\_all}(i)\right|, \left|\sum_{i=1}^{n} \text{DEP\_}S\text{\_all}(i)\right|, \left|\sum_{i=1}^{n} \text{GEN\_all}(i)\right|\right] \times [x_5, x_6, x_7, x_8]^T.\end{aligned} \tag{1}$$

(4) S_layer and S_all are weighted to calculate the weighted out-degree S of the software system. The system coupling is calculated through dividing $\overline{S}$ by the number of classes:

$$S = \left|\sum_{i=1}^{n} S(i)\right| = \left[\left|\sum_{i=1}^{n} S\_\text{layer}(i)\right|, \left|\sum_{i=1}^{n} S\_\text{all}(i)\right|\right] \times [\alpha, \beta]^T, \quad \overline{S} = \frac{S}{n}. \tag{2}$$

2.3.2. Calculation of the Weighted Fan-Out of Classes in a Software System. A special bipartite graph is constructed between classes of a software system. Weighted fan-out of classes in the special bipartite graph is analyzed based on characteristics of the object-oriented software.

(1) Construction of the Special Bipartite Graph in Software Systems. In the graph $G(V, E)$, if we divide the set V of nodes into two complementary subsets S and T, $S \cup T = V$, and $S \cap T = \phi$, the graph $G(V, E)$ is the bipartite graph. In the graph $G_{ij}(C_{ij}, E_{ij})$ constructed by classes C_i and C_j for the software, if only coupling relationship between classes is considered, coupling of methods and attributes in the class wouldn't be taken into account; then, the

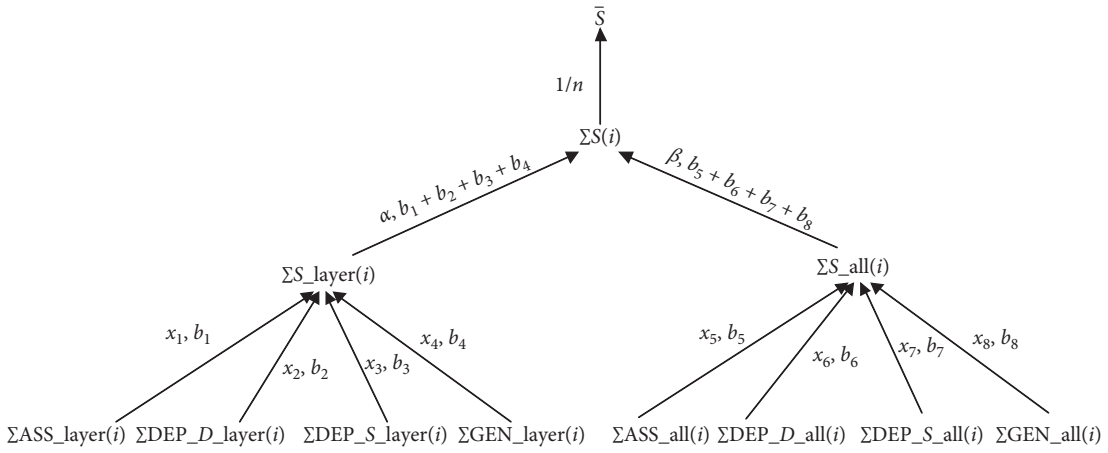

Figure 2: Illustration of the coupling measurement for a software system.

property of bipartite graph $C_i \cap C_j = \varnothing$ is satisfied. A network diagram constructed by classes C_i and C_j satisfies the following formula: $C_i \cup C_j = C_{ij}$. Moreover, the two points of a connecting edge between classes C_i and C_j are in classes C_i and C_j, respectively.

In summary, the complex coupling between classes C_i and C_j constructs a bipartite graph $G_{ij}(C_{ij}, E_{ij})$. However, $G_{ij}(C_{ij}, E_{ij})$ not only possesses the general properties of a bipartite graph, including method invocation and dependencies, but also possesses its own characteristics. In aggregation, reference, inheritance, and interface implementation between classes C_i and C_j, the two points of the connecting edge are in classes C_i and C_j, respectively. This bipartite graph $G_{ij}(C_{ij}, E_{ij})$ is defined as a special bipartite graph. However, the software system $G(C, E)$ can be considered as a set of special bipartite graphs $G_{ij}(C_{ij}, E_{ij})$ as well (Figure 3).

In the present study, the coupling of the complete bipartite graph $G_{ij}(C_{ij}, E_{ij})$ constructed by classes C_i and C_j was used to analyze the coupling of the software system $G(C, E)$.

(2) Modeling the Coupling of Special Bipartite and Calculating the Number of Weighted Fan-Out in Software Systems. In this study, a software system $G = (C_1, C_2, \ldots, C_N)$ was defined. Classes C_i and C_j were defined as two different classes in a software system: $C_i = (O_i, A_i, M_i)$. Among them, $O_i = \{O_{i1}, O_{i2}, \ldots, O_{ip}\}$ was the set of instantiated objects in class C_i and p is the number of instantiated objects. $A_i = \{A_{i1}, A_{i2}, \ldots, A_{iq}\}$ is the attribute set of class C_i, and q is the number of attribute. $M_i = \{M_{i1}, M_{i2}, \ldots, M_{ir}\}$ is the method set of class C_i, and r is the number of methods. The methods included class methods and instance methods, that is, $(C\_M \cup C\_O\_M) \subset M$.

The relationship of the special bipartite graph between classes C_i and C_j can be summarized as follows:

ASS

In class C_i, there was instantiation of class C_j (association), or one class existed in another class in the form of attribute (aggregation), which was defined as $C_j\_O_{jp'}$, where $1 \le p' \le p$.

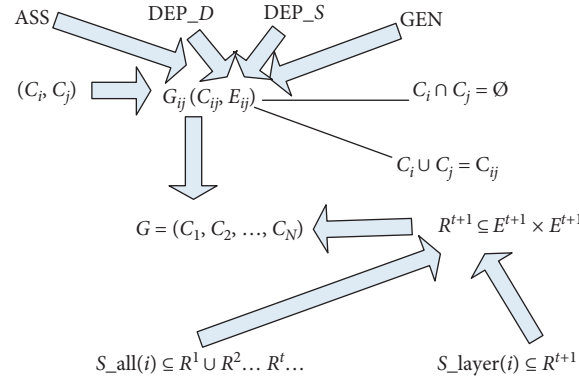

Figure 3: Abstract diagram of the software system network composed of a set of special bipartite graphs.

In the class C_i, instantiated object $O_{jp'}$ of class C_i was implemented $(1 \le p' \le p)$, or $O_{jp'}$ was a part of class C_i; then, there was an ASS edge between classes C_i and C_j, that is, $(C_i, C_j\_O_{jp'}) \in R_{\text{ASS}}$. The set of ASS weighted fan-out of class C_i was

$$\text{ASS}(i) = \left\{ \left(C_i, C_j\_O_{jp'}\right) \,\middle|\, 1 \le j \le N, 1 \le p' \le p, \left(C_i, C_j\_O_{jp'}\right) \in R_{\text{ASS}} \right\}. \tag{3}$$

DEP

Relationship between classes is implemented defined by instance methods and variables.
In class C_i, there are instance methods of class C_j: if $C_j\_O_{jp'}\_M_{jr'}$, $1 \le j \le N, 1 \le p' \le p$, and $1 \le r' \le r$,
then the relationship between classes C_i and C_j is defined as $(C_i, C_j\_O_{jp'}\_M_{jr'}) \in R_{\text{DEP\_D\_M}}$. In class C_i, there are instance variables of class C_j: if $C_j\_O_{jp'}\_A_{jq'}$, $1 \le j \le N, 1 \le p' \le p$, and $1 \le q' \le q$,
then relationship between classes C_i and C_j is defined as $(C_i, C_j\_O_{jp'}\_A_{jq'}) \in R_{\text{DEP\_D\_A}}$. Under the condition of instance methods and instance variables, the set of weighted fan-out for C_i was

$$\begin{aligned}\mathrm{DEP\_}D(i) = \{&\left(C_i, C_j\text{-}O_{jp'}\text{-}M_{jr'}\right), \left(C_i, C_j\text{-}O_{jp'}\text{-}A_{jq'}\right)\\ &\cdot \mid 1 \le j \le N, 1 \le p' \le p, 1 \le r' \le r,\\ &1 \le q' \le q, \left(C_i, C_j\text{-}O_{jp'}\text{-}M_{jr'}\right) \in R_{\mathrm{DEP\_}D},\\ &\left(C_i, C_j\text{-}O_{jp'}\text{-}A_{jq'}\right) \in R_{\mathrm{DEP\_}D\_A}\}.\end{aligned} \tag{4}$$

Implementing connecting edges between two classes through class methods and class variables.

If there are class methods of C_j (static methods) in class C_i: $C_j.M_{jr'}$, and $1 \le r' \le r$, then the relationship between classes C_i and C_j is defined as $(C_i, C_j\text{-}M_{jr'}) \in R_{\mathrm{DEP\_}S\_M}$, $1 \le r' \le r$. If there are class variables (static variables) of C_j in class C_i:$C_j.A_{jq'}$, $1 \le q' \le q$, then the relationship between classes C_i and C_j is defined as $(C_i, C_j\text{-}A_{jq'}) \in R_{\mathrm{DEP\_}S\_A}$, $1 \le q' \le q$. Thereafter, under the conditions of class methods and class variables, the set of weighted out-degree for class C_i was

$$\begin{aligned}\mathrm{DEP\_}S(i) = \{&\left(C_i, C_j\text{-}M_{jr'}\right), \left(C_i, C_j\text{-}A_{jq'}\right)\\ &\cdot \mid 1 \le j \le N, 1 \le r' \le r, 1 \le q' \le q,\\ &\left(C_i, C_j\text{-}M_{jr'}\right) \in R_{\mathrm{DEP\_}S\_M},\\ &\left(C_i, C_j\text{-}A_{jq'}\right) \in R_{\mathrm{DEP\_}S\_A}\}.\end{aligned} \tag{5}$$

GEN

As the inheritance is preferred in software engineering, if one class is a subclass of another class, the derived connecting edge was taken into account only once in this study. Because transfer of derived connecting edges would make the software system network more complex, this study did not consider transfer of derived connecting edge but only considered the conditions that class C_i was a direct subclass of class C_j (through extension), or class C_i was implemented through interface class C_j (through implementation), or class C_i was implemented by abstract class C_j (through extension). Thus, there was a GEN connecting edge between classes C_i and C_j, which was defined as $(C_i, C_j) \in R_{\mathrm{GEN}}$, and a set of GEN weighted out-degree for class C_i was

$$\mathrm{GEN}(i) = \left\{\left(C_i, C_j\right) \mid \left(C_i, C_j\right) \in R_{\mathrm{GEN}}\right\}. \tag{6}$$

(3) Determination of Weights. In software systems, one class can construct one or more special bipartite graphs with other classes. Supposing that the number of classes and the total number of weighted fan-out of all classes are definite in a software system, the first case is that the number of weighted fan-out in each class is the same or roughly the same. The second case is that there is no rule for the distribution of the number of weighted fan-out in a class. The third case is that the number of weighted fan-out of a class is heterogeneity, which approximately obeys the power-law distribution. For the second case, heterogeneity of the out-degree of the class is superior than that of the first case; however, this is impossible to be compared with the third case. For the third case, because the number of fan-out is limited for the majority of classes, only few classes have a large number of fan-out; therefore, maintenance staff can dedicate more effort on these few classes. Moreover, the maintenance workload of these classes is lower than that of the first case.

In this study, heterogeneity under the situation of fan-out was analyzed. If the distribution was the above-mentioned third case, then the larger the power-law value, the easier the maintenance, and the smaller the coupling degree. However, if the distribution was one of the other two cases, then it was considered in this study that the power-law value was equal to 1. b_1, b_2, b_3, and b_4 are the power-law values for the distribution of ASS_layer, DEP_D_layer, DEP_S_layer, and GEN_layer, respectively. b_5, b_6, b_7, and b_8 are the power-law values for the distribution of ASS_all, DEP_D_all, DEP_S_all, and GEN_all, respectively. The calculating formula for weights was as follows:

$$\begin{aligned}\left[x_1, x_2, x_3, x_4\right]^T &= \left[\sum_{i=1}^{4}\frac{b_i}{b_1}, \sum_{i=1}^{4}\frac{b_i}{b_2}, \sum_{i=1}^{4}\frac{b_i}{b_3}, \sum_{i=1}^{4}\frac{b_i}{b_4}\right]^T,\\ \left[x_5, x_6, x_7, x_8\right]^T &= \left[\sum_{i=5}^{8}\frac{b_i}{b_5}, \sum_{i=5}^{8}\frac{b_i}{b_6}, \sum_{i=5}^{8}\frac{b_i}{b_7}, \sum_{i=5}^{8}\frac{b_i}{b_8}\right]^T,\\ \alpha &= \frac{\sum_{i=4}^{8} b_i}{\sum_{i=1}^{8} b_i},\\ \beta &= 1 - \alpha.\end{aligned} \tag{7}$$

In the present study, statistical analyses were performed for the out-degree of the three open-source software systems, and the distributions were the first and the third cases as mentioned above, demonstrating that the proposed method had a certain practical value.

2.4. Theoretical Verification of Coupling Metrics. Whether the proposed CSBG method met the mathematical properties of the measurement metrics [4] was theoretically verified.

CSBG Property 1. CSBG satisfies nonnegativity.

Proof. In an object-oriented software system $G = (c_1, c_2, \ldots, c_N)$, there are two classes $c_1, c_2 \in G$. When ASS_layerDEP_D_layer, DEP_S_layer, and GEN_layer are all 0, the minimum value of the software system CSBG(G) is 0. However, there is a maximum value M $(M > 0)$, so that the CSBG(G) value is in the range of $[0, M]$. Thus, nonnegativity of CSBG is satisfied, and the proposition is proved. □

CSBG Property 2. CSBG satisfies zero value.

Proof. As described in CSBG property 1, if the minimum value is 0, then CSBG satisfies zero-value, and the proposition is proved as well. □

CSBG Property 3. CSBG satisfies monotonicity.

Proof. If one edge is arbitrarily added in the system, the weighted out-degree of classes would increase according to CSBG measurement metrics. Obviously, the coupling increases as well. Thus, CSBG meets monotonicity and the proposition is proved. □

CSBG Property 4. CSBG meets the property of class merging.

Proof. In an object-oriented software system $G = (c_1, c_2, \ldots, c_N)$, there are two classes $c_1, c_2 \in G$, and class c' is a merger of classes c_1 and c_2. The object-oriented system G' is a system in which classes c_1 and c_2 in G are replaced by class c'. CSBG mainly calculates the weighted out-degree of classes in software systems. Therefore,

$$\left[\mathrm{CSBG}(c_1) + \mathrm{CSBG}(c_2) \geq \mathrm{CSBG}(c') \,\middle|\, \mathrm{CSBG}(G) \geq \mathrm{CSBG}(G')\right]. \tag{8}$$

□

CSBG Property 5. CSBG satisfies the merge property of two irrelevant classes.

Proof. In an object-oriented software system $G = (c_1, c_2, \ldots, c_N)$, there are two classes $c_1, c_2 \in G$, and the two classes are not coupled. Moreover, class c' is the merger of classes c_1 and c_2. The object-oriented system G' is a system in which classes c_1 and c_2 in G are replaced by class c'. CSBG mainly calculates the weighted out-degree of classes in software systems. Therefore,

$$\left[\mathrm{CSBG}(c_1) + \mathrm{CSBG}(c_2) = \mathrm{CSBG}(c') \,\middle|\, \mathrm{CSBG}(G) = \mathrm{CSBG}(G')\right]. \tag{9}$$

□

2.5. Comparative Experiment. In the next sections, the CSBG method is herein proposed for coupling measurement and the existing measurement methods were compared and analyzed in order to verify the rationality of the results of CSBG measurement.

2.5.1. Calculating the Coupling of the Software System Using CSBG. In this section, CSBG for coupling measurement was compared with the existing measurement methods.

This experiment was conducted on a simple system as an example to analyze and compare the measurement values by the existing coupling measurements. This system was composed of 6 classes (Shape.java, Point.java, Line.java, Triangle.java, Quadrilateral.java, and Square.java), which described shapes, points, edges, triangles, quadrilaterals, and squares, respectively.

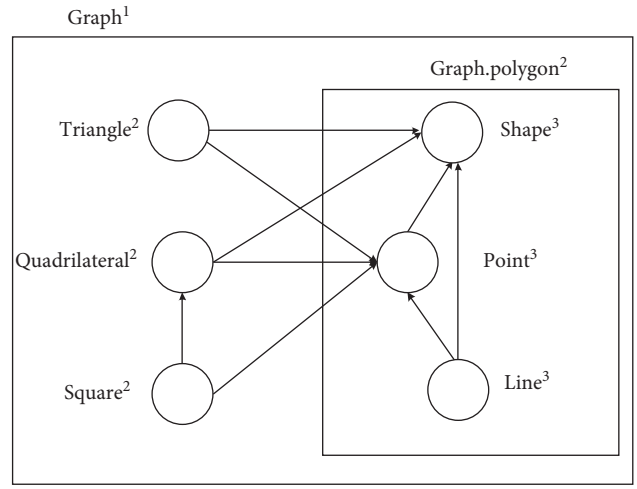

Figure 4: Diagram at package level.

```
1  package graph;
2
3  public class Shape {
4      String name;
5      int n;
6      int getn(){
7          return n;
8          }
9      public void speak(){
10
11     }
12 }
13
```

Figure 5: Demonstration of class shape.

```
1  package graph;
2
3  public class Point extends Shape{
4      private int x,y;
5      public Point(int x,int y){
6          this.x=x;
7          this.y=y;
8      }
9      public int GetX(){
10         return x;
11     }
12     public int GetY(){
13         return y;
14     }
15
16 }
17
```

Figure 6: Demonstration of class point.

Among them, the first three classes were in package graph, and the last three classes were in package graph.polygon (hierarchy of classes in package level is shown in Figure 4). There were inheritance, combination, variable declaration, and method invocation among these classes, which were appropriate for analyzing the coupling model. Codes of classes are shown in Figures 5–10.

```
package graph;
public class Line extends Shape{
	private Point L1,L2,L3;

    Line(Point a, Point b, Point c)
    {
        L1=new Point(a.GetX(), a.GetX());
        L2=new Point(b.GetX(), b.GetX());
        L3=new Point(c.GetX(), c.GetX());
    }

        public double Length()
        {
            return Math.sqrt(Math.pow(L2.GetX()-L1.GetX(), 2)+Math.pow(L2.GetY()-L1.GetY(), 2));
        }

}
```

Figure 7: Demonstration of class line.

```
package graph.polygon;
import graph.Point;
import graph.Shape;
public class Triangle extends Shape{
    private Point T1,T2,T3;
    double l1,l2,l3;
    double p;
    Triangle(Point a, Point b, Point c)
    {
        T1=new Point(a.GetX(), a.GetX());
        T2=new Point(b.GetX(), b.GetX());
        T3=new Point(c.GetX(), c.GetX());
    }

    public double circumference()
    {
        l1=Math.sqrt(Math.pow(T2.GetX()-T1.GetX(), 2));
        l2=Math.sqrt(Math.pow(T3.GetX()-T1.GetX(), 2));
        l3=Math.sqrt(Math.pow(T2.GetX()-T3.GetX(), 2));
         p=(l1+l2+l3)/2;
         return p;
    }
    public double area()
    {
        return Math.sqrt(p*(p-l1)*(p-l2)*(p-l3));
    }

}
```

Figure 8: Demonstration of class triangle.

```
package graph.polygon;
import graph.Point;
public class Square extends Quadrilateral  {
    Square(Point a, Point b, Point c, Point d)
    {
        super(a, b, c, d);
    }
    public double area(){
        return Math.pow(Q1, 2);
    }
}
```

Figure 9: Demonstration of class square.

There were three classes in the package graph, including class Shape, class Point, and class Line.

There were three classes in the package graph.polygon, which were classes of Triangle, Square, and Quadrilateral.

In this study, an algorithm was designed and the program was developed based on the aforementioned mathematical model, mainly calculating the four metrics for the out-degree of classes in the same layer and different layers of the package in software systems, including ASS, DEP_*D*, DEP_*S*, and GEN. Coupling metrics, including ASS, DEP_*D*, DEP_*S*, and GEN, were corresponded to the cases described in Section 2.3. Out-degrees of classes in various layers are shown in Table 4.

The mathematical model described in Section 2.3 was herein used. Because the number of classes was small, the heterogeneity of out-degree of classes could not be reflected. Moreover, heterogeneity had little impact on the coupling in this example. Therefore, it was considered that heterogeneity was approximately the same. Coupling of software systems was calculated as follows:

```
1  package graph.polygon;
2  import graph.Point;
3  import graph.Shape;
4  public class Quadrilateral extends Shape {
5      private Point T1,T2,T3,T4;
6      double Q1,Q2,Q3,Q4;
7      Quadrilateral(Point a, Point b, Point c, Point d)
8      {
9          T1=new Point(a.GetX(), a.GetX());
10         T2=new Point(b.GetX(), b.GetX());
11         T3=new Point(c.GetX(), c.GetX());
12         T4=new Point(c.GetX(), c.GetX());
13     }
14
15     public double circumference()
16     {
17
18         Q1=Math.sqrt(Math.pow(T2.GetX()-T1.GetX(), 2));
19         Q2=Math.sqrt(Math.pow(T2.GetX()-T3.GetX(), 2));
20         Q3=Math.sqrt(Math.pow(T3.GetX()-T4.GetX(), 2));
21         Q4=Math.sqrt(Math.pow(T4.GetX()-T1.GetX(), 2));
22          return Q1+Q2+Q3+Q4;
23     }
24 }
25
```

Figure 10: Demonstration of class quadrilateral.

$$
\begin{aligned}
\mathrm{CSBG}(G) &= \frac{\sum_i^6 \mathrm{CSBG}(C_i)}{6} = 6.55,\\
[x_1, x_2, x_3, x_4]^T &= [1,1,1,1]^T,\\
[x_5, x_6, x_7, x_8]^T &= [1,1,1,1]^T,\\
\alpha &= 0.5,\\
\beta &= 1-\alpha = 0.5,\\
\left|\sum_{i=1}^{n} S\_\mathrm{layer}(i)\right| &= \left[\left|\sum_{i=1}^{n} \mathrm{ASS}\_\mathrm{layer}(i)\right|, \left|\sum_{i=1}^{n} \mathrm{DEP}\_D\_\mathrm{layer}(i)\right|, \left|\sum_{i=1}^{n} \mathrm{DEP}\_S\_\mathrm{layer}(i)\right|, \left|\sum_{i=1}^{n} \mathrm{GEN}\_\mathrm{layer}(i)\right|\right] \times [x_1, x_2, x_3, x_4]^T\\
&= [6,10,0,3] \times [1,1,1,1]^T\\
&= 19,\\
\left|\sum_{i=1}^{n} S\_\mathrm{all}(i)\right| &= \left[\left|\sum_{i=1}^{n} \mathrm{ASS}\_\mathrm{all}(i)\right|, \left|\sum_{i=1}^{n} \mathrm{DEP}\_D\_\mathrm{all}(i)\right|, \left|\sum_{i=1}^{n} \mathrm{DEP}\_S\_\mathrm{all}(i)\right|, \left|\sum_{i=1}^{n} \mathrm{GEN}\_\mathrm{all}(i)\right|\right] \times [x_5, x_6, x_7, x_8]^T\\
&= [24,38,0,5] \times [1,1,1,1]^T\\
&= 67,\\
S = \left|\sum_{i=1}^{n} S(i)\right| &= \left[\left|\sum_{i=1}^{n} S\_\mathrm{layer}(i)\right|, \left|\sum_{i=1}^{n} S\_\mathrm{all}(i)\right|\right] \times [\alpha, \beta]^T\\
&= [19,67] \times [0.5, 0.5]^T\\
&= 43,\\
\overline{S} &= \frac{S}{n} = \frac{43}{6} = 7.17.
\end{aligned}
\tag{10}
$$

Table 4: Out-degrees of classes at the same layer and all layers of the package.

Class name	Out-degree at the same layer					Out-degree of all layers				
	ASS	DEP_*D*	DEP_*S*	GEN	Total	ASS	DEP_*D*	DEP_*S*	GEN	Total
Quadrilateral	0	0	0	0	0	8	16	0	1	25
Triangle	0	0	0	0	0	6	12	0	1	19
Line	6	10	0	1	17	6	10	0	1	17
Point	0	0	0	1	1	0	0	0	1	1
Shape	0	0	0	0	0	0	0	0	0	0
Square	0	0	0	1	1	4	0	0	1	5
Total	6	10	0	3	19	24	38	0	5	67

2.5.2. Analysis of the Results of Various Methods for Coupling Measurement. Coupling of software systems was calculated based on existing measurement methods, which is shown in Table 5. In addition to CSBG coupling measurement, other measurement methods mainly focus on measurement of a certain local fine-grained aspect. These methods were based on the theory of reductionism, which did not investigate the coupling of software systems from an overall and global perspective. Therefore, the measurement values were mostly either very large or very small, and several metrics were equal to 0. In addition, discrimination of these metrics was not significant compared with other methods for coupling measurement. The metrics calculated by CSBG had a better discrimination. Therefore, the existed methods have limitations, which cannot accordingly satisfy an effective coupling measurement for complex software systems. CSBG not only can consider a complex relationship between classes in object-oriented software systems but also analyze the complexity of classes and the special bipartite graph composed of classes from the prospective of overall package level. Therefore, the CSBG measurement method contained a certain rationality in theory.

3. Results

3.1. Application of CSBG Measurement Metrics in the Three Open-Source Software Systems. In order to further validate the effects of CSBG, this study used CSBG to measure and analyze coupling between classes in the three Java open-source software systems from different fields, including Art of Illusion [48], JabRef [49], and GanttProject [50]. Some studies have reported results of class cohesion metrics for the three open-source software systems [51–53]; it is feasible to further study the complexity of the three open-source software systems if there is a more reasonable method for coupling measurement. In order to verify the effects of the CSBG measurement method in actual open-source software systems, three Java open-source software systems from different aforementioned fields were used. Art of Illusion is a software system for 3D rendering, modeling, and animation. JabRef is a graphical application for managing bibliographic database. GanttProject is a software system for project scheduling characterized by resource calendar, management, and import or export (MS Project, PDF, HTML). The reasons to use the three open-source software systems in the measurement were because (1) these systems were based on object-oriented Java; (2) the classes in the systems had a certain scale; (3) the three systems were from different fields; and (4) the source codes were available as well. Scholars can freely download the source codes from an open-source website (http://sourceforge.net).

3.2. Association of Coupling with Statistical Characteristics of the Three Open-Source Software Systems. Firstly, the program was developed and out-degree of classes at the same layer and all layers of the package was eventually obtained, including ASS, DEP_*D*, DEP_*S*, and GEN.

In this section, DEP_*D* and DEP_*S* were analyzed, and the results are shown in Figures 11–18. In the experimental results, DEP(i) was a nonstandardized part of probability distribution $P(i)$. If, $P(i) \sim i^{-\gamma}$, then DEP $\sim (i)^{-\gamma}$. A linear function was fitted using the double logarithmic method that was fitted to estimate Gamma index γ (R is the Pearson's correlation coefficient and SD is standard deviation; γ is also expressed as B in the following table).

Although inheritance between classes increases coupling of the system, this is encouraged by the software system, which is conducive to reduce function definition and attribute definition in order to create a new class; thus, it is a poor coupling. It can be seen from linear distribution of GEN (Table 6) that neither all classes have an inheritance relationship, nor the GEN fan-out of all classes were very large or very small. However, classes with values equal to 0 or 1 were dominant.

Pearson's correlation coefficient (R) and SD value provided the quality of the linear fitting; the larger the R value, the better the quality of the linear fitting, and B is estimated Gamma index γ. Moreover, the smaller the SD value, the better the quality of the linear fitting. As shown in Table 7, if 0.95 is considered to be the minimum value, it can be approximated that the distribution obeyed the power-law distribution except that ASS value in JabRef was relatively small (0.91651 and 0.88148). Furthermore, the distributions of ASS layer, ASS_all layer, DEP_*D* layer, DEP_*S* layer, DEP_*D*_all layer, and DEP_*S*_all layer were assumed to obey power-law distribution. The results demonstrated that there was a certain rule for the number of fan-out of classes in the form of ASS and DEP, which was not the case that the values were mostly large or small. However, they had "scale-free" property for complex networks, which obeyed the power-law distribution. In actual software development process, if software developers excessively pursue low coupling between classes, a class may be divided into two or more subclasses; thus, system complexity may be accordingly increased. The process of determination of the range of coupling

Table 5: Results of various methods for coupling measurement.

	Quadrilateral	Triangle	Line	Point	Shape	Square	Software system
CSBG	17.5	13.3	5.1	0.3	0	3.1	7.17
CBO	3	2	2	5	4	2	
CBO′	1	1	1	4	0	1	
RFC	0	0	0	38	0	0	
RFCα	0	0	0	38	0	0	
RFC′	0	0	0	38	0	0	
MPC	0	0	0	0	0	0	
DAC	8	6	6	0	0	4	
DAC′	1	1	1	0	0	1	
COF							0.2
ICP	16	12	10	0	0	1	
IH-ICP	0	0	0	0	0	1	
NIH-ICP	16	12	10	0	0	0	
SIMAS	0	0	0	0	0	0	
PIM	16	12	10	0	0	1	
PIMAS	16	12	10	0	0	1	
INAG	1	1	1	0	0	1	
ACAIC	0	0	0	0	0	0	
OCAIC	4	3	3	0	0	0	
ACMIC	0	0	0	0	0	0	
OCMIC	0	0	0	0	0	0	
AMMC	0	0	0	0	0	1	
OMMC	16	12	10	0	0	1	
ICF	0	0	0	1	0	0	
FCF	1	1	1	0	0	1	

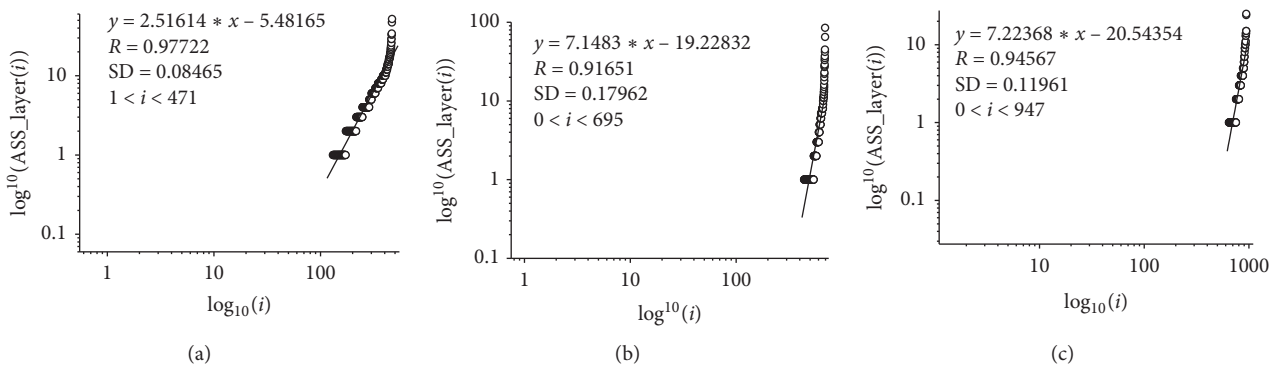

Figure 11: The double logarithmic diagrams of the fan-out of ASS invocation for classes at the same layer of a package.

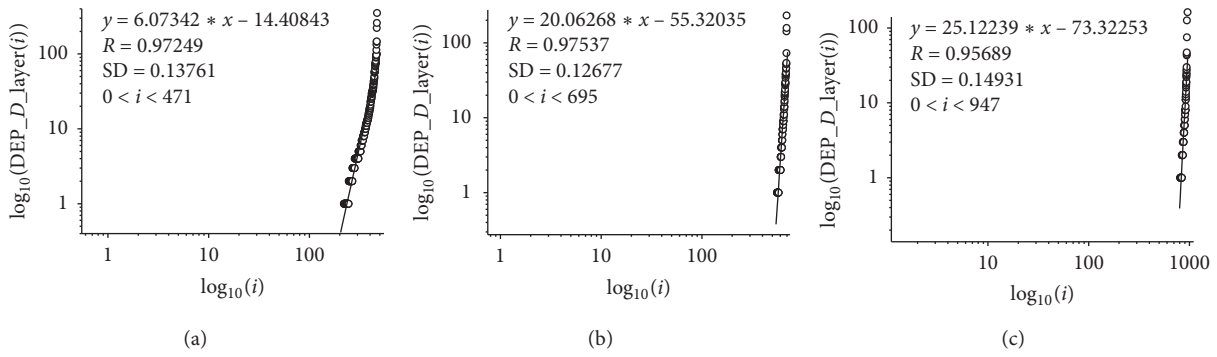

Figure 12: The double logarithmic diagrams of the fan-out of DEP_*D* invocation for classes at the same layer of a package.

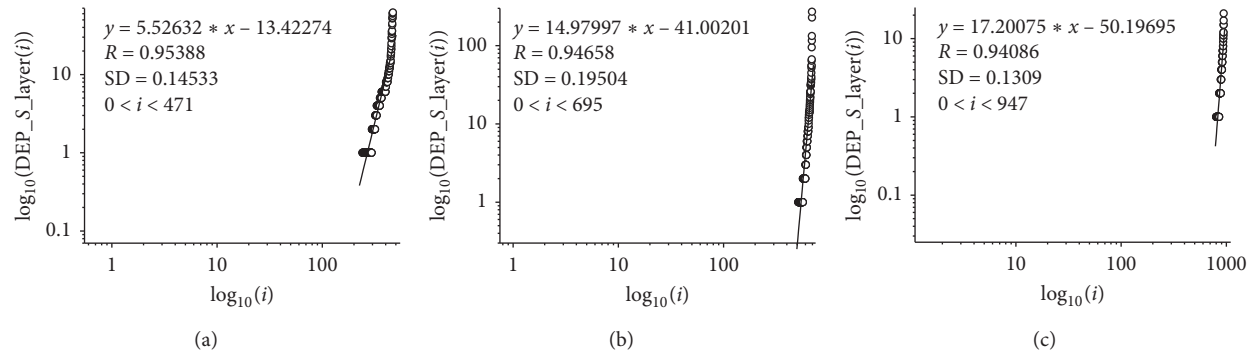

Figure 13: The double logarithmic diagrams of the fan-out of DEP_*S* invocation for classes at the same layer of a package.

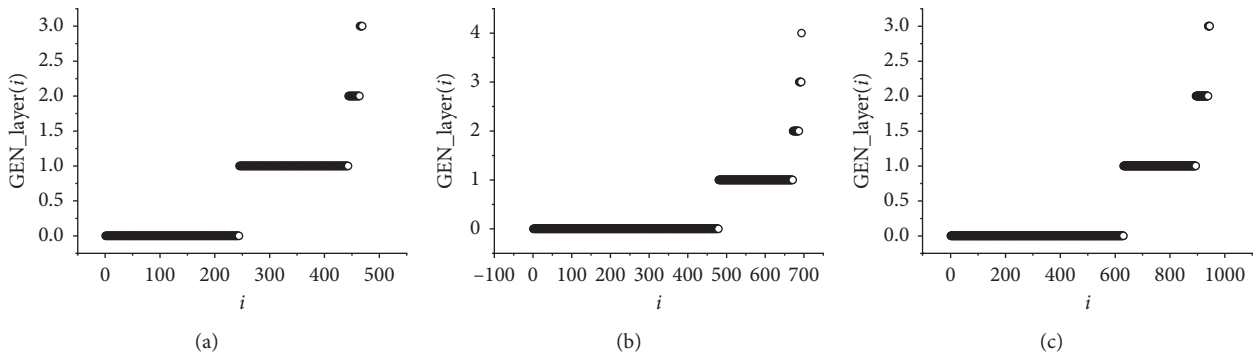

Figure 14: The double logarithmic diagrams of fan-out of GEN invocation for classes at the same layer of a package.

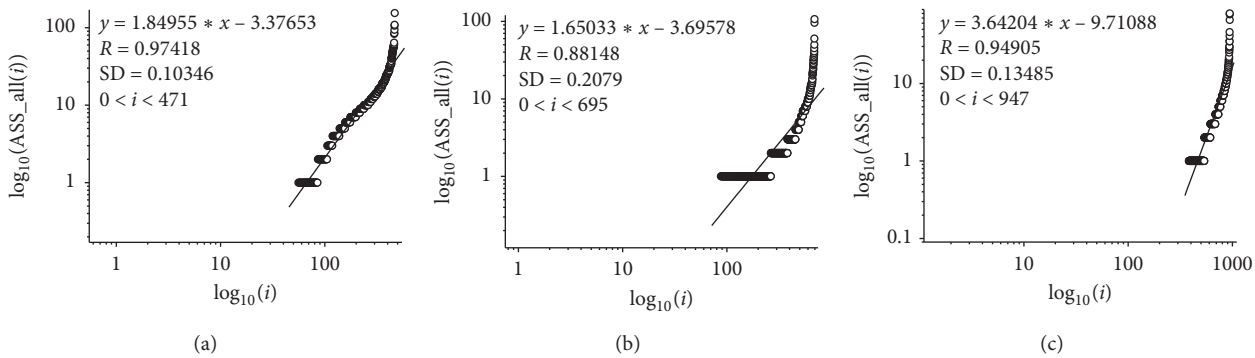

Figure 15: The double logarithmic diagrams of the fan-out of ASS invocation for classes at all layers of a package.

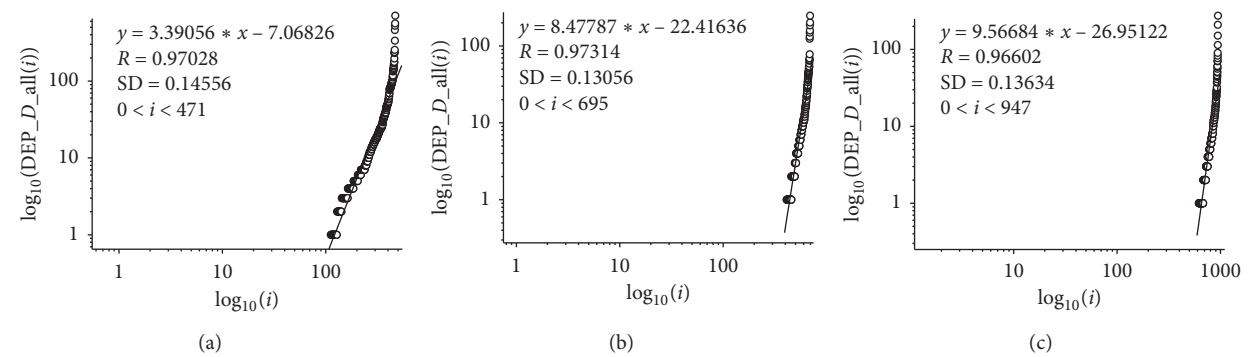

Figure 16: The double logarithmic graph of fan-out of DEP_*D* invocation for classes at all layers of a package.

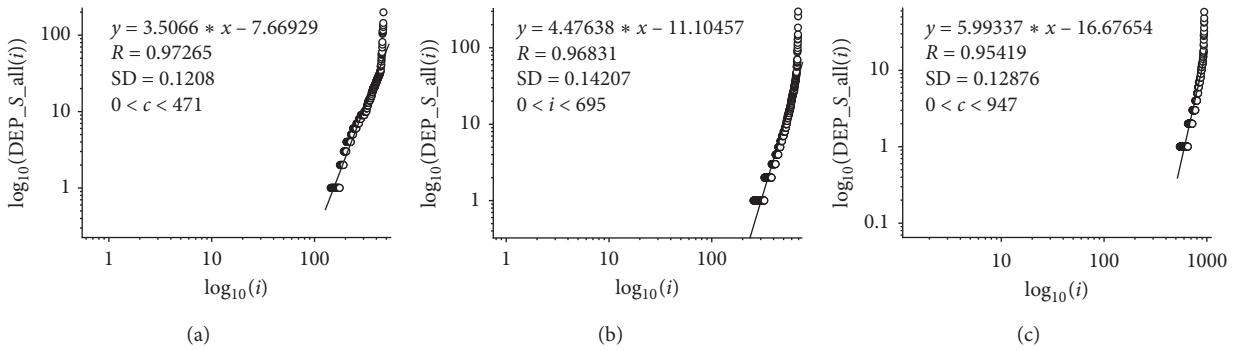

Figure 17: The double logarithmic diagrams of fan-out of DEP_*S* invocation for classes at all layers of a package.

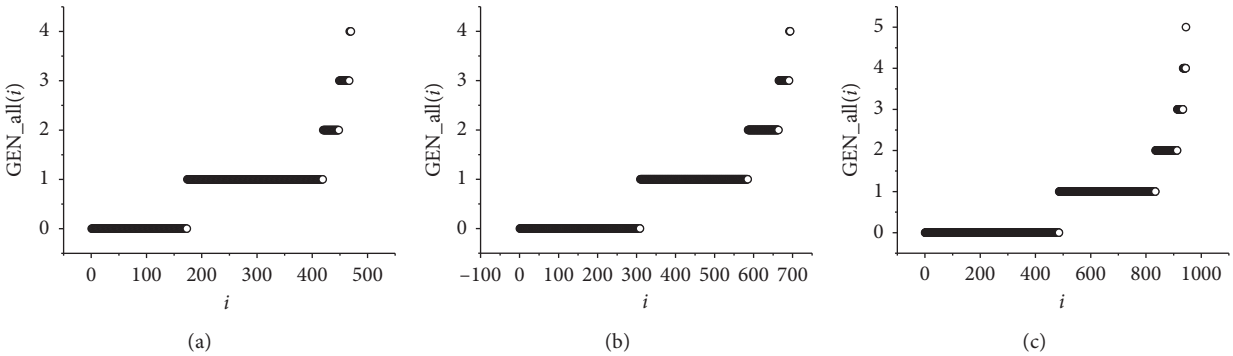

Figure 18: The double logarithmic diagrams of fan-out of GEN invocation for classes at all layers of a package.

Table 6: Values of fan-out for different classes of GEN.

	Value of fan-out	0	1	2	3	4	5	7
GEN_layer	Illusion	244	199	21	5	1	0	0
	JabRef	479	192	16	6	1	0	0
	GanttProject	630	264	45	6	0	1	0
GEN_all	Illusion	173	246	29	19	3	0	0
	JabRef	309	276	79	27	3	0	0
	GanttProject	485	349	79	21	10	1	1

Table 7: Values of *R*, SD, and *B* parameters.

	Software system	*R*	SD	*B*
ASS_layer	Illusion	0.9772	0.08465	2.51614
	JabRef	0.91651	0.17962	7.1483
	GanttProject	0.94567	0.11961	7.22368
DEP_*D*_layer	Illusion	0.97249	0.31761	6.07342
	JabRef	0.97537	0.12677	20.06268
	GanttProject	0.95689	0.14931	25.12239
DEP_*S*_layer	Illusion	0.95388	0.14533	5.52632
	JabRef	0.94658	0.19504	14.97997
	GanttProject	0.94086	0.1309	17.20075
ASS_all	Illusion	0.97414	0.10346	1.84955
	JabRef	0.88148	0.2079	1.65033
	GanttProject	0.94905	0.13485	3.64204
DEP_*D*_all	Illusion	0.97028	0.14556	3.39056
	JabRef	0.97314	0.13056	8.47787
	GanttProject	0.96602	0.13634	9.56684
DEP_*S*_all	Illusion	0.97265	0.1208	3.5066
	JabRef	0.96831	0.14207	4.47638
	GanttProject	0.95419	0.12876	5.99337

between classes in software systems is significant. Based on data analysis, it can be seen that scale-free" property of complex networks motivated software developers to pay more attention to the distribution range of the coupling in large-scale software systems, which could provide a reliable reference for developing more reasonable software systems.

3.3. Coupling Measurement for the Three Open-Source Software Systems. According to the results of the above-mentioned analysis, out-degrees of classes were often equal to 0, 1, and 2 for class inheritance in generalization, interface implementation, and implementation of abstract classes, which were approximately linearly distributed. Therefore, the power-law value of GEN was approximated to 1.

3.3.1. Calculation of Coupling Measurement for Art of Illusion. According to the CSBG method for coupling measurement, coupling of the software system for Art of Illusion was calculated as follows:

$$
\begin{aligned}
[x_1, x_2, x_3, x_4]^T &= \left[\sum_{i=1}^{4}\frac{b_i}{b_1}, \sum_{i=1}^{4}\frac{b_i}{b_2}, \sum_{i=1}^{4}\frac{b_i}{b_3}, \sum_{i=1}^{4}\frac{b_i}{b_4}\right]^T \\
&= \left[\sum_{i=1}^{4}\frac{b_i}{2.51614}, \sum_{i=1}^{4}\frac{b_i}{6.07342}, \sum_{i=1}^{4}\frac{b_i}{5.52632}, \sum_{i=1}^{4}\frac{b_i}{1}\right]^T \\
&= [6.007567, 2.48858, 2.735252, 15.11588]^T, \\
[x_5, x_6, x_7, x_8]^T &= \left[\sum_{i=5}^{8}\frac{b_i}{b_5}, \sum_{i=5}^{8}\frac{b_i}{b_6}, \sum_{i=5}^{8}\frac{b_i}{b_7}, \sum_{i=5}^{8}\frac{b_i}{b_8}\right]^T \\
&= \left[\sum_{i=5}^{8}\frac{b_i}{1.84955}, \sum_{i=5}^{8}\frac{b_i}{3.39056}, \sum_{i=5}^{8}\frac{b_i}{3.5066}, \sum_{i=5}^{8}\frac{b_i}{1}\right]^T \\
&= [5.269774, 2.874661, 2.779533, 9.74671]^T, \\
\alpha &= \frac{\sum_{i=4}^{8} b_i}{\sum_{i=1}^{8} b_i} = 0.392023, \\
\beta &= 1 - \alpha = 0.607977, \\
\left|\sum_{i=1}^{n} S\_\text{layer}(i)\right| &= \left[\left|\sum_{i=1}^{n} \text{ASS\_layer}(i)\right|, \left|\sum_{i=1}^{n} \text{DEP\_}D\text{\_layer}(i)\right|, \left|\sum_{i=1}^{n} \text{DEP\_}S\text{\_layer}(i)\right|, \left|\sum_{i=1}^{n} \text{GEN\_layer}(i)\right|\right] \times [x_1, x_2, x_3, x_4]^T \\
&= [2479, 5848, 2005, 260] \times [6.007567, 2.48858, 2.735252, 15.11588]^T \\
&= 38861.91, \\
\left|\sum_{i=1}^{n} S\_\text{all}(i)\right| &= \left[\left|\sum_{i=1}^{n} \text{ASS\_all}(i)\right|, \left|\sum_{i=1}^{n} \text{DEP\_}D\text{\_all}(i)(i)\right|, \left|\sum_{i=1}^{n} \text{DEP\_}S\text{\_all}(i)\right|, \left|\sum_{i=1}^{n} \text{GEN\_all}(i)\right|\right] \times [x_5, x_6, x_7, x_8]^T \\
&= [6705, 13883, 6255, 373] \times [5.269774, 2.874661, 2.779533, 9.74671]^T \\
&= 96264.25, \\
S &= \left|\sum_{i=1}^{n} S(i)\right| = \left[\left|\sum_{i=1}^{n} S\_\text{layer}(i)\right|, \left|\sum_{i=1}^{n} S\_\text{all}(i)\right|\right] \times [\alpha, \beta]^T \\
&= [38861.91, 96264.25] \times [0.392023, 0.607977]^T \\
&= 73761.21, \\
\overline{S} &= \frac{S}{n} = \frac{73761.21}{470} = 156.9387.
\end{aligned}
\tag{11}
$$

3.3.2. Calculation of Coupling Measurement for JabRef. According to CSBG for coupling measurement, coupling of the software system for JabRef was calculated as follows:

$$
\begin{aligned}
[x_1, x_2, x_3, x_4]^T &= \left[\sum_{i=1}^{4}\frac{b_i}{b_1}\frac{b_i}{b_1}, \sum_{i=1}^{4}\frac{b_i}{b_2}, \sum_{i=1}^{4}\frac{b_i}{b_3}, \sum_{i=1}^{4}\frac{b_i}{b_4}\right]^T \\
&= \left[\sum_{i=1}^{4}\frac{b_i}{7.1483}\frac{b_i}{7.1483}, \sum_{i=1}^{4}\frac{b_i}{20.06268}, \sum_{i=1}^{4}\frac{b_i}{14.97997}, \sum_{i=1}^{4}\frac{b_i}{1}\right]^T \\
&= [6.042129, 2.152801, 2.883247, 43.19095]^T, \\
[x_5, x_6, x_7, x_8]^T &= \left[\sum_{i=5}^{8}\frac{b_i}{b_5}, \sum_{i=5}^{8}\frac{b_i}{b_6}, \sum_{i=5}^{8}\frac{b_i}{b_7}, \sum_{i=5}^{8}\frac{b_i}{b_8}\right]^T \\
&= \left[\sum_{i=5}^{8}\frac{b_i}{1.65033}, \sum_{i=5}^{8}\frac{b_i}{8.47787}, \sum_{i=5}^{8}\frac{b_i}{4.47638}, \sum_{i=5}^{8}\frac{b_i}{1}\right]^T \\
&= [9.45543, 1.840625, 3.485982, 15.60458]^T, \\
\alpha &= \frac{\sum_{i=4}^{8} b_i}{\sum_{i=1}^{8} b_i} = 0.26540419, \\
\beta &= 1 - \alpha = 0.734596, \\
\left|\sum_{i=1}^{n} S\_\text{layer}(i)\right| &= \left[\left|\sum_{i=1}^{n} \text{ASS\_layer}(i)\right|, \left|\sum_{i=1}^{n} \text{DEP\_}D\text{\_layer}(i)\right|, \left|\sum_{i=1}^{n} \text{DEP\_}S\text{\_all}(i)\right|, \left|\sum_{i=1}^{n} \text{GEN\_layer}(i)\right|\right] \times [x_1, x_2, x_3, x_4]^T \\
&= [1326, 2032, 2396, 246] \times [6.042129, 2.152801, 2.883247, 43.19095]^T \\
&= 29375.8, \\
\left|\sum_{i=1}^{n} S\_\text{all}(i)\right| &= \left[\left|\sum_{i=1}^{n} \text{ASS\_all}(i)\right|, \left|\sum_{i=1}^{n} \text{DEP\_}D\text{\_all}(i)\right|, \left|\sum_{i=1}^{n} \text{DEP\_}S\text{\_all}(i)\right|, \left|\sum_{i=1}^{n} \text{GEN\_all}(i)\right|\right] \times [x_5, x_6, x_7, x_8]^T \\
&= [3430, 4403, 6733, 527] \times [9.45543, 1.840625, 3.485982, 15.60458]^T \\
&= 71947.46, \\
S = \left|\sum_{i=1}^{n} S(i)\right| &= \left[\left|\sum_{i=1}^{n} S\_\text{layer}(i)\right|, \left|\sum_{i=1}^{n} S\_\text{all}(i)\right|\right] \times [\alpha, \beta]^T \\
&= [29375.8, 71947.46] \times [0.26540419, 0.734596]^T \\
&= 60648.76, \\
\overline{S} &= \frac{S}{n} = \frac{60648.76}{694} = 87.39015.
\end{aligned}
\tag{12}
$$

3.3.3. Calculation of Coupling Measurement for GanttProject. According to CSBG for coupling measurement, coupling of the software system for GanttProject was calculated as follows:

$$
\begin{aligned}
[x_1, x_2, x_3, x_4]^T &= \left[\sum_{i=1}^{4}\frac{b_i}{b_1}, \sum_{i=1}^{4}\frac{b_i}{b_2}, \sum_{i=1}^{4}\frac{b_i}{b_3}, \sum_{i=1}^{4}\frac{b_i}{b_4}\right]^T \\
&= \left[\sum_{i=1}^{4}\frac{b_i}{7.22368}, \sum_{i=1}^{4}\frac{b_i}{25.12239}, \sum_{i=1}^{4}\frac{b_i}{17.20075}, \sum_{i=1}^{4}\frac{b_i}{1}\right]^T \\
&= [6.997378, 2.012023, 2.93864, 50.54682]^T, \\
[x_5, x_6, x_7, x_8]^T &= \left[\sum_{i=5}^{8}\frac{b_i}{b_5}, \sum_{i=5}^{8}\frac{b_i}{b_6}, \sum_{i=5}^{8}\frac{b_i}{b_7}, \sum_{i=5}^{8}\frac{b_i}{b_8}\right]^T \\
&= \left[\sum_{i=5}^{8}\frac{b_i}{3.64204}, \sum_{i=5}^{8}\frac{b_i}{9.56684}, \sum_{i=5}^{8}\frac{b_i}{5.99337}, \sum_{i=5}^{8}\frac{b_i}{1}\right]^T \\
&= [5.5496, 2.111695, 3.370766, 20.20225]^T, \\
\alpha &= \frac{\sum_{i=4}^{8} b_i}{\sum_{i=1}^{8} b_i} = 0.285548, \\
\beta &= 1 - \alpha = 0.714452, \\
\left|\sum_{i=1}^{n} S\_\text{layer}(i)\right| &= \left[\left|\sum_{i=1}^{n} \text{ASS\_layer}(i)\right|, \left|\sum_{i=1}^{n} \text{DEP\_}D\text{\_layer}(i)\right|, \left|\sum_{i=1}^{n} \text{DEP\_}S\text{\_layer}(i)\right|, \left|\sum_{i=1}^{n} \text{GEN\_layer}(i)\right|\right] \times [x_1, x_2, x_3, x_4]^T \\
&= [1102, 1331, 565, 379] \times [6.997378, 2.012023, 2.93864, 50.54682]^T \\
&= 31206.69, \\
\left|\sum_{i=1}^{n} S\_\text{all}(i)\right| &= \left[\left|\sum_{i=1}^{n} \text{ASS\_all}(i)\right|, \left|\sum_{i=1}^{n} \text{DEP\_}D\text{\_all}(i)\right|, \left|\sum_{i=1}^{n} \text{DEP\_}S\text{\_all}(i)\right|, \left|\sum_{i=1}^{n} \text{GEN\_all}(i)\right|\right] \times [x_5, x_6, x_7, x_8]^T \\
&= [3201, 3833, 2291, 622] \times [5.5491, 2.111695, 3.370766, 20.20225]^T \\
&= 46138.17, \\
S &= \left|\sum_{i=1}^{n} S(i)\right| = \left[\left|\sum_{i=1}^{n} S\_\text{layer}(i)\right|, \left|\sum_{i=1}^{n} S\_\text{all}(i)\right|\right] \times [\alpha, \beta]^T \\
&= [31206.69, 46138.17] \times [0.285548, 0.714452]^T \\
&= 41874.52, \\
\overline{S} &= \frac{S}{n} = \frac{41874.52}{946} = 44.26482.
\end{aligned}
\tag{13}
$$

The three aforementioned open-source software systems were analyzed from the points of view of package level, class level, and method level using CSBG for coupling measurement. A program was also developed to calculate various metrics; thus, the coupling of the three open-source software systems in descending order was the Art of Illusion, JabRef, and GanttProject, suggesting that it was feasible to use CSBG for coupling measurement of software systems that contained a certain practical value.

4. Conclusion

Based on bipartite graphs for complex networks, by comprehensive consideration of the weighted fan-out between classes from points of view of package level, class level, and method level, this study expressed that the interaction of classes is a special bipartite graph, while a software system is a set of these special bipartite graphs. For this purpose, first, this study analyzed the four relationships for a software system, including ASS, DEP_*D*, DEP_*S*, and GEN, and coupling relationship for a class with other classes in the same layer of package was considered as well. Moreover, coupling relationship for classes in a package with other classes in different layers of the package was taken into account. Therefore, the CSBG method for coupling measurement of software systems was proposed, which was completely in compliance with the mathematical characteristics of the widely accepted metrics. Second, for a software system, other typical methods and CSBG method were compared for the purpose of coupling measurement, and the results revealed that the measured value was either large or small due to the defects of other measurement methods that were analyzed from an overall and global perspective. Moreover, the corresponding values were mostly equal to 0. Therefore, there were some defects in other measurement methods, while CSBG had its rationality. Eventually, a program was developed based on the CSBG method to apply the three open-source software systems (Art of Illusion, JabRef, and GanttProject). The results demonstrated that coupling of the three open-source software systems in the descending order was the Art of Illusion, JabRef, and GanttProject. Although inheritance between classes increases coupling of the system, this is also followed by software engineering, which is conducive to reduce function definition and attribute definition in order to create a new class, and thus, this is weak coupling. It can be concluded from the linear distribution of GEN that all classes either had an inheritance relationship, or that the number of GEN fan-out of all classes was very large or very small. However, classes with values equal to 0 or 1 were accounted. Furthermore, it was revealed that in the same layer and total layers of the package, fan-out values of ASS, DEP_*D*, and DEP_*S* obeyed the scale-free property of complex networks. These findings provided empirical support for the CSBG method. The statistical power-law metrics were applied to the method for coupling measurement proposed in this study in order to calculate the coupling of the three open-source software systems, which provided a reliable reference for further investigation of coupling between classes in software systems using statistics of complex networks. In [54], it was mentioned that cohesion distribution of the majority classes of a software system contained a certain regularity. In other words, it was not the case that neither cohesion of all classes was very large nor very small. In the empirical analysis of coupling, the values of coupling metrics had a regularity similar to class cohesion. Although coupling represented the degree of interdependence between classes, the greater the coupling, the more complex the software from an intuitive aspect. However, excessive pursuit of "high cohesion and low coupling" of software systems increases the workload of software developers and the complexity of software systems as well. Therefore, the empirical evidence showed that within a certain range, reducing the coupling was helpful to attenuate the complexity of the software, while excessively blindly pursuit of low coupling increases the complexity of software systems.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (no. 61602400). This work was also supported in part by the Key Research and Development Program of Hangzhou under Grant 20182011A46.

References

[1] W. P. Stevens, G. J. Myers, and L. L. Constantine, "Structured design," *IBM Systems Journal*, vol. 13, no. 2, pp. 115–139, 1974.

[2] W. P. Stevens, G. J. Myers, and L. L. Constantine, "Structured design," *IBM Systems Journal*, vol. 38, no. 2, pp. 231–256, 1999.

[3] E. Yourdon and L. L. Constantine, *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*, Yourdon Press, Englewood Cliffs, NJ, USA, 1979.

[4] L. C. Briand, J. W. Daly, and J. K. Wust, "A unified framework for coupling measurement in object-oriented systems," *IEEE Transactions on Software Engineering*, vol. 25, no. 1, pp. 91–121, 1999.

[5] M. D'Ambros, M. Lanza, and R. Robbes, "On the relationship between change coupling and software defects," in *Proceedings of the 2009 16th Working Conference on Reverse Engineering*, Lille, France, October 2009.

[6] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE Transactions on Software Engineering*, vol. 31, no. 10, pp. 897–910, 2005.

[7] P. Yu, T. Systa, and H. Muller, "Predicting fault-proneness using OO metrics. An industrial case study," in *Proceedings of the Sixth European Conference on Software Maintenance and Reengineering*, Budapest, Hungary, March 2002.

[8] L. C. Briand, J. Wuest, and H. Lounis, "Using coupling measurement for impact analysis in object-oriented systems,"

in *Proceedings of the IEEE International Conference on Software Maintenance*, Oxford, UK, September 1999.

[9] F. G. Wilkie and B. A. Kitchenham, "Coupling measures and change ripples in C++ application software," *Journal of Systems & Software*, vol. 52, no. 2-3, pp. 157–164, 2000.

[10] G. Antoniol, R. Fiutem, and L. Cristoforetti, "Using metrics to identify design patterns in object-oriented software," in *Proceedings of the Fifth International Software Metrics Symposium*, Bethesda, MD, USA, November 1998.

[11] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[12] S. R. Chidamber and C. F. Kemerer, "Towards a metrics suite for object oriented design," in *Proceedings of the ACM Conference on Object Oriented Programming, Systems, Languages and Applications*, Orlando, FL, USA, 1991.

[13] W. Li and S. Henry, "Object-oriented metrics that predict maintainability," *Journal of Systems and Software*, vol. 23, no. 2, pp. 111–122, 1993.

[14] Y. Lee, "Measuring the coupling and cohesion of an object-oriented program based on information flow," in *Proceedings of the International Conference on Software Qualiy*, Maribor, Slovenia, 1995.

[15] L. Briand, P. Devanbu, and W. Melo, "An investigation into coupling measures for C++ software engineering," in *Proceedings of the 19th International Conference on Software engineering*, Boston, MA, USA, May 1997.

[16] H. Li and B. Li, "A pair of coupling metrics for software networks," *Journal of Systems Science and Complexity*, vol. 24, no. 1, pp. 51–60, 2011.

[17] E. Arisholm, L. C. Briand, and A. Foyen, *Dynamic Coupling Measurement for Object-Oriented Software*, IEEE Press, Piscataway, NJ, USA, 2004.

[18] J. K. Chhabra and V. Gupta, "A survey of dynamic software metrics," *Journal of Computer Science & Technology*, vol. 25, no. 5, pp. 1016–1029, 2010.

[19] D. Poshyvanyk and A. Marcus, "The conceptual coupling metrics for object-oriented systems," in *Proceedings of the 2006 22nd IEEE International Conference on Software Maintenance*, Philadelphia, PA; USA, September 2006.

[20] D. Poshyvanyk, A. Marcus, R. Ferenc, and T. Gyimóthy, "Using information retrieval based coupling measures for impact analysis," *Empirical Software Engineering*, vol. 14, no. 1, pp. 5–32, 2009.

[21] M. Gethers and D. Poshyvanyk, "Using relational topic models to capture coupling among classes in object-oriented software systems," in *Proceedings of the 2010 IEEE International Conference on Software Maintenance*, Timisoara, Romania, September 2010.

[22] H. Gall, K. Hajek, and M. Jazayeri, "Detection of logical coupling based on product release history," in *Proceedings of the International Conference on Software Maintenance*, Washington, DC, USA, April 1998.

[23] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl, "Mining version histories to guide software changes," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 429–445, 2005.

[24] E. J. Weyuker, "Evaluating software complexity measures," *IEEE Transactions on Software Engineering*, vol. 14, no. 9, pp. 1357–1365, 1988.

[25] I. Vessey and R. Weber, "Research on structured programming: an empiricist's evaluation," *IEEE Transactions on Software Engineering*, vol. SE-10, no. 4, pp. 397–407, 2009.

[26] L. C. Briand, S. Morasca, and V. R. Basili, "Property-based software engineering measurement," *IEEE Transactions on Software Engineering*, vol. 22, no. 1, pp. 68–86, 1996.

[27] L. C. Briand, J. W. Daly, and J. Wüst, "A unified framework for cohesion measurement in object-oriented systems," *Empirical Software Engineering*, vol. 3, no. 1, pp. 65–117, 1998.

[28] D. J. Watts and S. H. Strogatz, "Collective dynamics of "small-world" networks," *Nature*, vol. 393, 1998.

[29] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[30] W. Pan, H. Ming, C. K. Chang, Z. Yang, and D.-K. Kim, "ElementRank: ranking java software classes and packages using multilayer complex network-based approach," *IEEE Transactions on Software Engineering*, 2019.

[31] W. Pan, B. Li, J. Liu, Y. Ma, and B. Hu, "Analyzing the structure of java software systems by weighted K-core decomposition," *Future Generation Computer Systems*, vol. 83, pp. 431–444, 2018.

[32] C. R. Myers, "Software systems as complex networks: structure, function, and evolvability of software collaboration graphs," *Physical Review E*, vol. 68, no. 4, 15 pages, 2003.

[33] N. LaBelle and E. Wallingford, "Inter package dependency networks in open source software," 2004, http://arxiv.org/abs/0411096.

[34] D. Hyland-Wood, D. Carrington, and S. Kaplan, "Scale-free nature of java software package," class and method collaboration graphs," Techical report no. TR-MS1286, University of Maryland College, College Park, MD, USA, 2006.

[35] Y. Xiang, W. Pan, H. Jiang, Y. Zhu, and H. Li, "Measuring software modularity based on software networks," *Entropy*, vol. 21, no. 4, p. 344, 2019.

[36] W. Pan and C. Chai, "Measuring software stability based on complex networks in software," *Cluster Computing*, vol. 22, no. S2, pp. 2589–2598, 2019.

[37] W. Pan, B. Song, K. Li, and K. Zhang, "Identifying key classes in object-oriented software using generalizedk-core decomposition," *Future Generation Computer Systems*, vol. 81, pp. 188–202, 2018.

[38] W. Pan and C. Chai, "Structure-aware mashup service clustering for cloud-based internet of things using genetic algorithm based clustering algorithm," *Future Generation Computer Systems*, vol. 87, pp. 267–277, 2018.

[39] W. Pan, J. Dong, K. Liu, and J. Wang, "Topology and topic-aware service clustering," *International Journal of Web Services Research*, vol. 15, no. 3, pp. 18–37, 2018.

[40] F. Liljeros, C. R. Edling, L. A. N. Amaral, H. E. Stanley, and Y. Åberg, "The web of human sexual contacts," *Nature*, vol. 411, no. 6840, pp. 907-908, 2001.

[41] H. Jeong, B. Tombor, R. Albert, Z. N. Oltvai, and A.-L. Barabási, "The large-scale organization of metabolic networks," *Nature*, vol. 407, no. 6804, pp. 651–654, 2002.

[42] P.-P. Zhang, K. Kan Chen, Y. He et al., "Model and empirical study on some collaboration networks," *Physica A: Statistical Mechanics and Its Applications*, vol. 360, no. 2, pp. 599–616, 2006.

[43] Q. Xuan, F. Du, and T. J. Wu, "Empirical analysis of internet telephone network : from user ID to phone," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 19, no. 2, Article ID 023101, 2009.

[44] M.-S. Shang, L. Lü, Y.-C. Zhang, and T. Zhou, "Empirical analysis of web-based user-object bipartite networks," *EPL (Europhysics Letters)*, vol. 90, no. 4, p. 48006, 2010.

[45] K.-I. Goh, M. E. Cusick, D. Valle, B. Childs, M. Vidal, and A.-L. Barabasi, "The human disease network," *Proceedings of*

the National Academy of Sciences, vol. 104, no. 21, pp. 8685–8690, 2007.

[46] B. Kitchenham, "What's up with software metrics?-a preliminary mapping study," *Journal of Systems and Software*, vol. 83, no. 1, pp. 37–51, 2010.

[47] J. Eder and M. Schrefl, "Coupling and cohesion in object-oriented systems," in *Proceedings of the International Workshop on Object Orientation in Operating Systems*, pp. 264–272, Dordan, France, September 1992.

[48] Illusion, 2012, http://sourceforge.net/projects/aoi/.

[49] JabRef, 2012, http://sourceforge.net/projects/jabref/.

[50] GanttProject, 2012, http://sourceforge.net/projects/ganttproject/.

[51] J. Al Dallal and L. C. Briand, "An object-oriented high-level design-based class cohesion metric," *Information and Software Technology*, vol. 52, no. 12, pp. 1346–1361, 2010.

[52] J. Al Dallal, "Measuring the discriminative power of object-oriented class cohesion metrics," *IEEE Transactions on Software Engineering*, vol. 37, pp. 778–804, 2011.

[53] J. Al Dallal, "The impact of accounting for special methods in the measurement of object-oriented class cohesion on refactoring and fault prediction activities," *Journal of Systems and Software*, vol. 85, no. 5, pp. 1042–1057, 2012.

[54] A. Gu, X. Zhou, Z. Li, Q. Li, and L. Li, "Measuring object-oriented class cohesion based on complex networks," *Arabian Journal for Science and Engineering*, vol. 42, no. 8, pp. 3551–3561, 2017.