Measuring Cohesion of Software Systems Using Weighted Directed Complex Networks

Jie Zhang*†, Jiajing Wu*†§, Yongxiang Xia‡ and Fanghua Ye*†
*School of Data and Computer Science
Sun Yat-sen University, Guangzhou 510006, China
†National Engineering Research Center of Digital Life
Sun Yat-sen University, Guangzhou 510006, China.
‡College of Information Science and Electronic Engineering
Zhejiang University, Hangzhou 310027, China.
§Email: wujiajing@mail.sysu.edu.cn

***Abstract*—Network theory has been demonstrated as an effective approach for better understanding and analysis of software systems from a systematic perspective. In this paper, we develop a directed and weighted software dependency network model to analyse software systems from a complex network perspective. In particular, to measure the "High Cohesion and Low Coupling" nature of object-oriented software systems, we propose to use a directed and weighted modularity index, which can better reflect the cohesion of software systems. Experiments are conducted on a series of open source object-oriented software systems with various scales, and the results show that the directed and weighted modularity can better characterize software systems with different cohesion.**

I. Introduction

Over the last decades, complex network theories have been widely studied and applied to various realistic complex systems, such as World Wide Web, communication networks, power grids, etc. [1], [2]. Software systems, as one of the most diverse and sophisticated human-made systems, are also intensively investigated from a complex network perspective in recent years. Comparing with existing approaches, the complex network approach can give a comprehensive understanding of the structure and internal relationship of complex software systems.

In the context of employing complex network theories to software analysis, software components at different granularity are often abstracted as nodes, while various kinds of relationships between them are taken as edges. Such modeling method provides a valuable insight into the underlying structure of software systems. Using this approach, several discoveries and analysis methods have been made in previous studies. For example, many kinds of software networked systems have been demonstrated to have *scale-free* and *small-world* properties in its topology [3], [4], [5]. And other researchers have applied complex network to software quality measurement [6], [7], [8], software evolution analysis [9], [10], [11] and defect prediction [12], [13], [14], [15].

Communities are subsets of nodes that are more densely linked mutually than to the rest of the network, and community structures are ubiquitous in many networked systems.A series of research works have preliminarily investigated community structure in software systems. On the one hand, several community detection algorithms have been applied to find structural or functional modules in software systems [16], [17]. On the other hand, packages of software systems can be modeled as a natural community partition of software networks[18], [19]. Concas *et al.*[20] and Lovro *et al.*[19] have discussed the relationship between software packages and network communities. Both of them concluded that a great disparity exists between software packages and the related software network's community structures. Moreover, some researchers measure the community structure of realistic software networks as a guidance of software architecture design or package refactoring.

Object-oriented design advocate to incorporate data and related functionality into modules, while reducing coupling between them. This principle is referred to as "high cohesion and low coupling". Software systems with such nature are usually assumed to be better understood, modified, and maintained. Thus, it is pivotal to find a reasonable and effective quantitative measure of this property. Prior work has proposed various metrics to characterize the cohesion of software systems. However, most of these metrics are limited to a microscopic point of view. By modeling the software systems as complex networks, software cohesion can be measured from a systematic perspective.

In [18] and [21], the concept of network modularity, which measures the quality of a particular partition of network has been used to measure cohesion of software systems. However, in these studies, the software systems are simply modeled as undirected or unweighted networks and the conclusions can be further verified. In this paper, we represent a software system as a weighted and directed network to better characterize the internal dependency relationships. We propose to use a weighted and directed version of modularity to quantitatively measure the "high cohesion and low coupling" nature of software systems.

The rest of this paper is organized as follow. First, in Section II, we describe the model of directed and weighted software dependency networks and propose a metric for software cohesion measurement. Next, in Section III, we conduct

978-1-5386-4881-0/18/$31.00 ©2018 IEEE

experiments on a series of object-oriented software systems and give a detailed performance evaluation. Finally, main conclusions and possible directions of further work are given in Section IV.

II. Methodology

A. Software Dependency Network

In the context of modeling software systems as complex networks, various types of network models have been developed [7]. In general, software components (methods/classes/packages) are abstracted as nodes, while edges represent the interaction or relationships between components. In this paper, we aim to measure cohesion of software systems from a network perspective, and thus adopt and refine the software dependency network model to characterize the structure and internal relationships of objected-oriented software systems.

In this paper, the dependency network of a software system is modeled as a directed and weighted network $G = (N, E, W)$, where N is the set of nodes, E is the set of directed edges, and W denotes the set of weights of the edges. A network G is constructed from source code in the following manner: each node $n_i \in N$ represent a class c_i, and a directed edge e_{ij} exists if and only if one of the four following types of dependency relationships exist between c_i and c_j:

1) c_i calls a method of c_j
2) c_i accesses a field of c_j
3) c_i contains a field of type c_j
4) c_i contains a local variable of type c_j

As for $w_{ij} \in W$, the weight of edge e_{ij} is calculated by counting the occurrence times of the above dependency relationships. Based on the operation rules of realistic software systems, weights of edges is extremely important for accurate characterization of the dependency strength between software components.

Besides, some special details of network modeling should be mentioned. First, we do not take an anonymous inner class as a node, because they can be regarded as a part of the outer classes from the perspective of source code organization. Next, we allow self-edges for the existence of dependency between methods of the same class. Last, we exclude Test classes from the source code as they are usually unrelated with the software quality.

B. Weighted and Directed Modularity Metric

In [22] and [23], Newman proposed a well-known metric called *modularity* to measure the quality of a particular community partition for an undirected and unweighted network. Let m be the number of edges, A_{ij} be the number of edges between n_i and n_j, k_i be the degree of n_i, and c_i be the community that node n_i belongs to. Then the definition of *modularity* is given as

$$Q = \frac{1}{2m} \sum_{ij} (A_{ij} - \frac{k_i k_j}{2m}) \delta(c_i, c_j) \tag{1}$$

TABLE I
Basic information of software systems.

Name	Description	Version	#Class
Kryo	Develop Framework	4.0.1	83
Maven-Wagon	Develop Tool	3.0.0	107
Metrics	Program Monitor	4.0.0	120
Jackcess	Java Library	2.1.0	137
Webmagic	Develop Framework	0.7.3	173
Gora	Develop Framework	0.7.0	200
Jajuk	Music Organizer	1.10.9	438
Helix	Develop Framework	0.6.10	439
Dubbo	Develop Framework	2.5.6	766
Jfree-chart	Java Library	1.0.19	830
Jasperreports	Java Library	6.4.4	2735
Eclipse	IDE	2.0	5999

A higher *modularity* value corresponds to a more reasonable community partition in which nodes inside a community are connected more densely than with the rest of the network. This idea coincides with "high cohesion and low coupling" principle of software engineering which also emphasis maximizing the degree to which elements of a module belong together and reducing the coupling between modules.

Intuitively, we can take the software package architecture as a community partition and use the concept of modularity to measure the cohesion of software. Pan *et al.* [18] used the change rate of modularity to detect the refactoring points of software systems. However, they only conduct experiments on one particular software and the results are not further analysed. Lovro *et al.* [21] compared the modularity of software packages with that of the communities divided by community detection algorithms. They found that the modularity of software packages is quite low and concluded that it can not be regarded as an indication of significant modular structure. However, both of the above work ignored the strength of dependency relationship which is extremely important in software cohesion measurement. This assumption may lead to an inappropriate representation of software structure and its internal dependency relationships, thus obtaining inaccurate conclusions.

To address the above-mentioned problems, we propose a directed and weighted software dependency network model and refined the *modularity* metric to adapt to this model.

In [24] and [25], Newman upgraded the *modularity* formula for directed network and weighted network respectively. Combining these two formulae, we can easily get the directed and weighted modularity as follow:

$$Q_{dw} = \frac{1}{m} \sum_{ij} (A_{ij} - \frac{k_i^{out} k_j^{in}}{m}) \delta(c_i, c_j) \tag{2}$$

where m denotes the sum of all the edges' weights, A_{ij} denotes the weight of the edge from n_i to n_j, k_i^{out} denotes the out-degree of n_i and k_j^{in} denotes the in-degree of n_j, c_i denotes the community that node *i* belong to, and $\delta(c_i, c_j)$ takes 1 when c_i equals c_j, 0 otherwise.

In this paper, we build a directed and weighted software dependency network by extracting dependency relationships

978-1-5386-4881-0/18/$31.00 ©2018 IEEE

TABLE II

PROPERTIES OF SOFTWARE DEPENDENCY NETWORK. *#Node* AND *#Edge* DENOTE NUMBER OF NODES AND NUMBER OF EDGES RESPECTIVELY; *AverIODegree* DENOTES THE AVERAGE IN-DEGREE OR OUT-DEGREE WHILE *WAverIODegree* IS THE WEIGHTED ONE; *MaxInDegree* AND *MaxOutDegree* DENOTE THE LARGEST NODE IN-DEGREE AND OUT-DEGREE WHILE *WMaxInDegree* AND *WMaxOutDegree* DENOTES THE WEIGHTED ONES.

Name	#Node	#Edge	AverIODegree	WAverIODegree	MaxInDegree	WMaxInDegree	MaxOutDegree	WMaxOutDegree
Kryo	83	390	4.7	153.0	46	1558	35	1326
Maven-Wagon	107	378	3.5	54.0	20	460	25	738
Metrics	120	338	2.8	62.3	28	586	14	1381
Jackcess	137	853	6.2	184.6	40	2289	43	3600
Webmagic	173	827	4.8	42.8	58	694	21	754
Gora	200	13305	4.4	66.5	76	668	22	541
Jajuk	438	4383	10.0	114.4	241	1875	92	2509
Helix	439	2741	6.2	103.0	144	3210	54	2790
Dubbo	766	4034	5.3	56.3	305	4028	31	769
Jfree-chart	830	5265	6.3	101.8	218	2220	100	4790
Jasperreports	2735	21088	7.7	81.2	553	4171	212	7832
Eclipse	5999	78617	13.1	147.0	1220	14390	162	10878

from the source code. Then we take the top-level package architecture as a natural community partition. Nodes in one community correspond to the classes belong to the same package. In particular, the classes directly belong to the top-level compose a community named 'none'. Furthermore, using the directed and weighted *modularity* given above, the degree of the "high cohesion and low coupling" nature of software systems can be measured.

III. EXPERIMENTS

Next, we aim to illustrate that the directed and weighted software dependency network can better characterize the structure and internal dependency relationship of software systems, and the proposed modularity index is a reasonable and accurate metric to characterize software cohesion.

In this work, we conduct experiments on 12 object-oriented Java software systems whose source codes are collected from *GitHub* or other websites [26]. The basic information of these software systems is given in Table I. As we can see, due to the property of Java language, most of these software systems are develop frameworks or libraries. Without loss of generality, we choose software systems with a variety of scales, among which the smallest one only contains 83 classes and the largest one contains 5999 classes.

We then extracted the dependency relationships from the source code using the methodology described in Section II-A and build a directed and weighted network model for each software system. For an intuitive understanding of the network structure, we extract some topology properties of each network in Table II. In this model, the edges denote the dependency relationships, and thus the metrics related to degree are quite important. From the difference between degree metrics and the weighted ones, we can see that the weights of edges can capture more information of software systems. For example, the weighted metrics is not in proportion to the unweighted ones and the *MaxInDegree* commonly larger than *MaxOutDegree* while the weighted ones show reverse property.

Furthermore, we take the top-level package architecture as the community partition of software dependency network and calculate the directed and weighted modularity proposed in (2). For comparison, we also calculate the *modularity* metrics used in previous work, namely the *modularity*, *weighted modularity* and *directed modularity*. The results are shown in Table III. Similar to the results of [21], the undirected or unweighted modularity is extremely low and sometimes even negative. More importantly, different software systems perform quite similar in terms of the previous modularity indexes. In other words, these metrics can hardly distinguish the software systems with different degree of the "high cohesion and low coupling" nature.

TABLE III

NUMBER OF PACKAGES AND DIFFERENT MODULARITY VALUES. '#COMMUNITY' DENOTES THE NUMBER OF PACKAGE, Q DENOTES THE UNDIRECTED AND UNWEIGHTED MODULARITY, Q_w DENOTES THE WEIGHTED MODULARITY, Q_d DENOTES THE DIRECTED MODULARITY, AND Q_{dw} DENOTES THE DIRECTED AND WEIGHTED MODULARITY.

Name	#Community	Q	Q_w	Q_d	Q_{dw}
Kryo	6	-0.035	0.074	0.217	0.423
Maven-Wagon	11	0.001	0.073	0.280	0.382
Metrics	19	0.084	0.099	0.469	0.516
Jackcess	5	-0.112	-0.237	0.204	0.187
Webmagic	17	0.089	0.163	0.323	0.476
Gora	24	0.191	0.330	0.460	0.724
Jajuk	6	-0.097	-0.004	0.170	0.345
Helix	22	0.113	0.207	0.345	0.530
Dubbo	12	0.120	0.186	0.458	0.568
Jfree-chart	14	-0.085	-0.036	0.274	0.390
Jasperreports	25	-0.096	-0.088	0.269	0.330
Eclipse	16	0.114	0.186	0.418	0.558

By taking the direction and weights of the edges into consideration, the modularity values increase significantly, especially for the directed and weighted modularity Q_{dw}. Meanwhile, we can observe an obvious diversity between different software systems in terms of Q_{dw}. Thus, it can be concluded that the direct and weighted modeling imports some critical information of software systems and also illustrate the reasonableness and effectiveness of the proposed software dependency network model.

To better understand the *modularity* values, we also divide

978-1-5386-4881-0/18/$31.00 ©2018 IEEE

TABLE IV
DIRECTED AND WEIGHTED MODULARITY FOR SOFTWARE PACKAGES AND COMMUNITIES DIVIDED BY VARIOUS ALGORITHMS

Name	Package	IM	ML	LP	LE	WT
Kryo	0.423	0.308	**0.369**	0.171	0.310	0.309
Maven-Wagon	0.382	0.457	**0.587**	0.496	0.548	0.283
Metrics	0.516	0.364	**0.482**	0.307	0.452	0.375
Jackcess	0.187	0.352	**0.688**	0.605	0.656	0.186
Webmagic	0.476	0.295	**0.466**	0.259	0.380	0.402
Gora	0.724	0.730	**0.770**	0.638	0.638	0.580
Jajuk	0.345	0.566	**0.606**	0.48	0.522	0.474
Helix	0.530	0.409	**0.493**	0.249	0.379	0.463
Dubbo	0.568	0.667	**0.714**	0.486	0.544	0.663
Jfree-chart	0.390	0.653	**0.727**	0.571	0.620	0.492
Jasperreports	0.330	0.594	**0.691**	0.464	0.517	0.586
Eclipse	0.558	0.582	**0.693**	0.378	0.528	0.496

the software networks using several classic community detection algorithms, including *Info Map(IM)*, *Muti-Level(ML)*, *Label Propagation(LP)*, *Leading Eigenvector(LE)* and *Walk Trap(WT)*. Most of these community detection algorithms support directed and weighted network partition except the IM and the ML. For these two algorithms, we translate the software networks to weighted but undirected ones in advance. To assure fair comparisons, we also use the directed and weighted modularity in (2) for calculation.

From the results given in Table IV, we can see that the ML algorithm performs steadily and can always achieve higher modularity than other algorithms. Therefore, we can take the modularity of the communities divided by the ML algorithm as a benchmark for comparison. On one hand, the values of the benchmark reflect the goodness of the software structure. On the other hand, the ratio of the modularity of package to the benchmark shows the reasonableness of the software package architecture. The closer the modularity of package to the benchmark, the higher the cohesion of the corresponding software system.

As we can see, most of these software systems have a relatively high value of Q_{dw} and the relative value. Some of them perform extremely good (e.g., Metrics, Gora, Eclipse), while there also exist some systems with a very low Q_{dw}, such as the Jackcess. It is worth noting that the package modularity of some software systems are even higher than their community modularity. This may be caused by the small size of the software networks or the unsteadiness of the community detection algorithms.

To further evaluate whether the proposed modularity metric can accurately measure network cohesion, we choose a network with a low modularity, namely, the Jackcess, to analysis whether it poorly obey the "high cohesion and low coupling" principle. The package dependency matrix is given in Table V, in which the element in i-th row and j-th column denotes the sum of dependency strength from the classes in package i to classes in package j. Because of the small size of this software, we directly look at the bottom-level packages. The results indicate the existence of a great data skew which refers to the unbalanced distribution of dependency. As we can see, the package "impl.none" contains a majority of internal dependency links. This is obviously not a good software structure design from a macroscopic view. This result further verified the reasonableness of the proposed directed and weighted modularity in a sense.

IV. Conclusions and Future Work

In this paper, we represent software systems as directed and weighted software dependency networks to capture their structures and internal dependency relationships. Further, we discuss the relationship between modularity of network community partition and the 'high cohesion and low coupling' property of software architecture. To address the problems existing in previous related work, we propose to extract the directed and weighted modularity from the developed network model to measure software cohesion. We conduct experiments on 12 Java open source software systems, and the results show that the import of direct and strength of dependency relationship is valuable, and the proposed metric can better distinguish software systems in terms of their "high cohesion and low coupling" properties. Finally, we further illustrate the reasonableness and effectiveness of the proposed modularity metric by analysing the dependency structure of the software with the lowest modularity.

For future work, we will try to improve the software dependency network model and modify the modularity metrics. First, more types dependency relationships can be considered, especially some call relationships completed by developing frameworks. Next, the dependency strength can be defined in a more reasonable way. For example, different weights can be given to different types of dependency relationships. Finally, the modularity metric can be further modified by taking the hierarchical structure of source code organization into consideration.

Acknowledgment

This work was supported by the National Natural Science Foundation of China under Grant Nos. 61503420 and 61573310, the Fundamental Research Funds for the Central Universities under Grant No. 17lgpy120, and the Zhejiang Provincial Natural Science Foundation of China under Grant No. LY15F030006.

References

[1] D. J. Watts and S. H. Strogatz, "Collective dynamics of small-worldnetworks," *nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[2] J. Wu, C. K. Tse, and F. C. Lau, "Optimizing performance of communication networks: An application of network science," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62, no. 1, pp. 95–99, 2015.

[3] C. R. Myers, "Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs," *Physical Review E*, vol. 68, no. 4, p. 046116, 2003.

[4] A. P. De Moura, Y.-C. Lai, and A. E. Motter, "Signatures of small-world and scale-free properties in large computer programs," *Physical review E*, vol. 68, no. 1, p. 017102, 2003.

[5] G. Concas, M. Marchesi, S. Pinna, and N. Serra, "Power-laws in a large object-oriented software system," *IEEE Transactions on Software Engineering*, vol. 33, no. 10, pp. 687–708, 2007.

978-1-5386-4881-0/18/$31.00 ©2018 IEEE

TABLE V
PACKAGE DEPENDENCY MATRIX

	none	complex	impl.none	impl.complex	impl.query	query	util
none	**1357**	4	150	0	0	0	42
complex	10	**128**	0	0	0	0	0
impl.none	1755	34	**16599**	24	8	0	534
impl.complex	372	585	62	**731**	0	0	0
impl.query	55	0	4	0	**960**	86	0
query	0	0	0	0	0	**56**	0
util	442	0	64	0	0	0	**1224**

[6] T. H. Nguyen, B. Adams, and A. E. Hassan, "Studying the impact of dependency network measures on software quality," in *Proc. IEEE International Conference on Software Maintenance*, Timisoara, Romania, 2010, pp. 1–10.

[7] C. Y. Chong and S. P. Lee, "Analyzing maintainability and reliability of object-oriented software using weighted complex network," *Journal of Systems and Software*, vol. 110, pp. 28–53, 2015.

[8] Y.-T. Ma, K.-Q. He, B. Li, J. Liu, and X.-Y. Zhou, "A hybrid set of complexity metrics for large-scale object-oriented software systems," *Journal of Computer Science and Technology*, vol. 25, no. 6, pp. 1184–1201, 2010.

[9] K. A. Ferreira, M. Bigonha, R. S. Bigonha, and B. M. Gomes, "Software evolution characterization-a complex network approach," *X Brazilian Simposium on Software Quality-SBQS*, pp. 41–55, 2011.

[10] P. Bhattacharya, M. Iliofotou, I. Neamtiu, and M. Faloutsos, "Graph-based analysis and prediction for software evolution," in *Proc. IEEE International Conference on Software Engineering*, Zurich, Switzerland, 2012, pp. 419–429.

[11] T. Chaikalis and A. Chatzigeorgiou, "Forecasting java software evolution trends employing network models," *IEEE Transactions on Software Engineering*, vol. 41, no. 6, pp. 582–602, 2015.

[12] T. Zimmermann and N. Nagappan, "Predicting subsystem failures using dependency graph complexities," in *Proc. IEEE International Symposium on Software Reliability*, Trollhattan, Sweden, 2007, pp. 227–236.

[13] M. Pinzger, N. Nagappan, and B. Murphy, "Can developer-module networks predict failures?" in *Proc. ACM International Symposium on Foundations of software engineering*, Atlanta, Georgia, 2008, pp. 2–12.

[14] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," in *Proc. ACM/IEEE 30th International Conference on Software Engineering*, Leipzig, Germany, 2008, pp. 531–540.

[22] M. E. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical review E*, vol. 69, no. 2, p. 026113, 2004.

[15] C. Bird, N. Nagappan, H. Gall, B. Murphy, and P. Devanbu, "Putting it all together: Using socio-technical networks to predict failures," in *Proc. IEEE International Symposium on Software Reliability Engineering*, Atlanta, Georgia, 2009, pp. 109–119.

[16] D. Li, Y. Han, and J. Hu, "Complex network thinking in software engineering," in *Proc. IEEE International Conference on Computer Science and Software Engineering*, vol. 1. Hubei, China: IEEE, 2008, pp. 264–268.

[17] Y. Qu, X. Guan, Q. Zheng, T. Liu, L. Wang, Y. Hou, and Z. Yang, "Exploring community structure of software call graph and its applications in class cohesion measurement," *Journal of Systems and Software*, vol. 108, pp. 193–210, 2015.

[18] W. Pan, B. Li, Y. Ma, and J. Liu, "Multi-granularity evolution analysis of software using complex network theory," *Journal of Systems Science and Complexity*, vol. 24, no. 6, pp. 1068–1082, 2011.

[19] L. Šubelj and M. Bajec, "Software systems through complex networks science: Review, analysis and applications," in *Proc. ACM International Workshop on Software Mining*. Beijing, China: ACM, 2012, pp. 9–16.

[20] G. Concas, C. Monni, M. Orrù, and R. Tonelli, "A study of the community structure of a complex software network," in *Proc. IEEE International Workshop on Emerging Trends in Software Metrics*, San Francisco, CA, USA, 2013, pp. 14–20.

[21] L. Šubelj and M. Bajec, "Community structure of complex software systems: Analysis and applications," *Physica A: Statistical Mechanics and its Applications*, vol. 390, no. 16, pp. 2968–2975, 2011.

[23] M. E. Newman, "Modularity and community structure in networks," *Proceedings of the national academy of sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.

[24] E. A. Leicht and M. E. Newman, "Community structure in directed networks," *Physical review letters*, vol. 100, no. 11, p. 118703, 2008.

[25] M. E. Newman, "Analysis of weighted networks," *Physical review E*, vol. 70, no. 5, p. 056131, 2004.

[26] "Github," https://github.com/.

978-1-5386-4881-0/18/$31.00 ©2018 IEEE