Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2016, Article ID 3858637, 9 pages
http://dx.doi.org/10.1155/2016/3858637

Research Article

An Improved Approach to Identifying Key Classes in Weighted Software Network

Yi Ding,[1,2] Bing Li,[1,3,4] and Peng He[5]

[1]*State Key Lab of Software Engineering and School of Computer, Wuhan University, Wuhan 430072, China*
[2]*School of Computer, Wuhan Vocational College of Software and Engineering, Wuhan 430205, China*
[3]*International School of Software, Wuhan University, Wuhan 430072, China*
[4]*Research Center of Complex Network, Wuhan University, Wuhan 430072, China*
[5]*Faculty of Computer Science and Information Engineering, Hubei University, Wuhan 430062, China*

Correspondence should be addressed to Bing Li; bingli@whu.edu.cn

Received 3 June 2016; Accepted 7 August 2016

Academic Editor: Emilio Insfran

To help the newcomers understand a software system better during its development, the key classes are in general given priority to be focused on as soon as possible. There are numerous measures that have been proposed to identify key nodes in a network. As a metric successfully applied to evaluate the productivity of a scholar, little is known about whether h-index is suitable to identify the key classes in weighted software network. In this paper, we introduced four h-index variants to identify key classes on three open-source software projects (i.e., Tomcat, Ant, and JUNG) and validated the feasibility of proposed measures by comparing them with existing centrality measures. The results show that the measures proposed not only are able to identify the key classes but also perform better than some commonly used centrality measures (the improvement is at least 0.215). In addition, the finding suggests that mE-Weight defined by the weight of a node's top k edges performs best as a whole.

1. Introduction

With the increment development of open-source software (OSS), the overall scale and complexity of software system become more and more great. For a newcomer in this context, if he or she wants to obtain a general understanding of the software system as soon as possible, the key classes are in general recommended to the newcomer to master some basic concepts better [1]. Software can be characterized as a dependency network in terms of relationship between various elements (class, package, feature, and so on), labeled as software dependency network (SDN). The nodes in SDN present classes or interfaces while the links present different dependencies between the nodes. The frequency of dependency between two nodes is viewed as edge weight. Therefore, we can understand software in terms of weighted software network. In other words, the problems of identifying key classes of the software system converted into measuring key nodes in SDN.

The key nodes refer to a number of nodes which are more likely to affect the structure and function in a network. Although the proportion of such node usually is not high, they can rapidly influence most of the remaining nodes. In complex networks, node importance represents the node's influence, transmission capacity, and robustness. At present, numerous measures have been proposed to identify the important node in a network; for example, both Jian-Guo et al. [2] and Ren and Lü [3] made a comparison between several commonly used methods to analyze their differences and the specific application scenarios. Sun and Luo [4] also reviewed the approaches to measuring important nodes from the perspective of the global network topology structure and node attributes. According to node degree and clustering coefficient, Ren et al. [5] proposed a new method for measuring node importance. Kitsak et al. [6] proposed a k-shell based method to resolve the issue.

In social network analysis, except for centrality, h-index has been widely applied to evaluate the location or influence

of actors [7]. For coauthorship network, h-index is useful for reflecting the scholars' performance based on their position and influence within their collaboration network. The main advantage of h-index is the consideration of the quantity and quality of the papers published by a scholar.

As mentioned above, there are lots of methods for measuring node importance in a network, but few people attempt to use h-index or its variants to identify key classes in context of software engineering. To compensate the lack and verify the feasibility of h-index, this paper proposes new measures based on h-index to identify key classes in SDN.

Our contributions are summarized as follows:

(1) We proposed four new measures based on h-index to study class importance in SDN, respectively, according to the degree of neighbors and the edge weights.

(2) Compared with several existing centrality measures, we validated the feasibility of proposed measures to identify important nodes.

The rest of this paper is organized as follows. Section 2 is a review of related work. In Section 3, the preliminary theories and approaches are introduced. Section 4 shows the results of our experiments in detail. After that, a conclusion and future work are made in Section 5.

2. Related Work

There are many indicators measuring a node importance defined in complex networks and social network analysis, such as node centrality [8]; one of the simplest is the node degree centrality, and we could understand it as the number of other nodes connected to the target node, representing the potential impact that a node made on the surrounding nodes, which is the local properties of nodes in a network. To reflect the node's global properties, this paper puts forward betweenness centrality, closeness centrality, and eigenvector centrality and so forth. Both betweenness centrality and closeness centrality are involved in topological distance between nodes in network.

Wang and Pan [9] used the betweenness centrality, closeness centrality, and eigenvector centrality to measure the importance of classes in the software network and analyzed differences of these indexes in identifying key classes. In the process of prediction, we could focus attention on different contents according to the importance of entities in software systems. Zimmermann and Nagappan [10] take Windows Server 2003 as the experimental object, built software network according to the dependence relationship between classes, and contrastively analyzed the effect that network indexes such as the node degrees, betweenness centrality, closeness centrality, and source code metrics have on defect prediction. The results showed that the prediction results of network indexes were verified to be more effective than source code metrics.

Meneely et al. [11] built the developer cooperation network according to the code change information for the file level, and then network centricity indexes were used to measure contribution of developers and predicted the system fault condition after the release. The authors found that centricity index and the fault occurring after release have obvious correlation in the developer cooperation network; it confirmed that the network indexes have advantages to earlier found faults during the development phase. Under the use of centricity indexes, the difference of the role of developers in the community was measured by Crowston et al. [12], and then the core-edge hierarchy between developers was identified. According to the contribution relationship between developers and the modules, Pinzger et al. [13] used network centricity indexed metric to measure the contribution of developers and, combined with the relationship between the developer's contribution and the number of defects after release, found module in the center more likely to break down than the edge. Shin and others [14] used centricity indexes and fragile code snippet forecasted system and found that the use of these indicators could well distinguish fragile and neutral file of the system and build file vulnerability forecasting model.

Zimmerman et al. [15] identified the core program unit by using centricity indexes to analyze software network. Bhattacharya et al. [16], respectively, built code level and module level network and used network indicators to assess the seriousness of bug and optimization of refactoring and predict defects. Steidl and others [17] used centricity, PageRank, and node degrees to determine the core classes in software system and confirmed that the results found agree well with the results of practical experience developers. Zanetti and others [18] also used centricity indexes to measure the ability level of defect reporters and then direct the distribution flaw based on the index values.

Besides using centricity indexes, Perin et al. [19] order classes according to the dependency between them by using PageRank algorithm. HITS algorithm was used by Zaidman and Demeyer to calculate importance of classes to determine the system's key classes [20]. Pan et al. [21] put forward a kind of weighted PageRank algorithm to identify the key package of software system. Zhou and Xu [22] compared the differences in identifying important classes under the use of PageRank and HITS and betweenness centrality indexes at the class level in software network. Meyer et al. [23] model software as a network and apply k-core decomposition to identify a core subset of potentially important classes. Jiang et al. [24] proposed a technique to measure the importance of each class based on unique input/output sequence to identify the key class. Kamran et al. [25] propose an efficient technique that pinpoints the core architecture classes of the system. Sora [26] models the static dependencies structure of the system as a graph and applies a graph ranking algorithm to identify key classes in software systems. Pan et al. [27] put forward a kind of weighted k-core decomposition to identify important packages of object-oriented software. Şora [28] proposed a tool to automatically extract some summary to identify the most important classes of a system.

As one of the important indexes for evaluating research level of scholars, few people applied h-index to measure the importance of classes in software system. Wang et al. [29] used the h-index and its variant identified the key classes in the class-level software network, and it was also compared

Table 1: Centrality measures for node 1 mentioned in Figure 1.

k	Node (degree)	aN-Degree	mN-Degree	Edge (weight)	aE-Weight	mE-Weight
1	2 (6)	$1 < 6$	$1^2 < 6$	1–4 (8)	$1 < 8$	$1^2 < 8$
2	3 (5)	$2 < 5$	$2^2 < 11$	1–3 (6)	$2 < 6$	$2^2 < 14$
3	4 (3)	$3 = 3$	$3^2 < 14$	1–5 (5)	$3 < 5$	$3^2 < 19$
4	5 (3)	$4 > 3$	$4^2 < 17$	1-2 (3)	$4 > 3$	$4^2 < 22$
5	6 (2)		$5^2 > 19$	1–7 (3)		$5^2 = 25$
6	7 (1)			1–6 (2)		$6^2 > 27$
	Value	14/3	4		19/3	5

with the code index in the identification accuracy. They found that if they take the top 15% of the class based on the indicators the recall rate could achieve 70%. In their work, however, the calculation for h-index only considered the node degrees and did not consider the weight of edge. Compared to Wang et al., although this article did not consider other variants of the h-index, when calculating the h-index of nodes, we, in turn, considered the number of node edges (the node degrees) and the influence of edge weight.

3. Research Approach

3.1. Software Dependency Network (SDN). Considering that the direction and weight of the dependencies between classes have practical significance, so, in this paper, reverse engineering method was used to construct SDN model. We adopt $G = (V, E, W)$ to represent a weighted network, $V = \{v_i\}$ is all nodes in this network, that is, the set of all classes in a system, and $E = \{e_{ij}\}$ is all edge sets. Note that if there is a dependency between node i and node j, then $e_{ij} = 1$, otherwise it is 0; $W = \{w_{ij}\}$ is a weight set corresponding to the edge set E, where w_{ij} is the number of dependencies between node i and node j, and is the weight of e_{ij}.

During building of SDN, a directed edge e_{ij} mainly considers the following five kinds of dependencies:

(1) Class v_i inherited or realized the class or interface v_j (inheritance dependence).

(2) Class v_i contained the type field of class v_j (field dependence).

(3) Class v_i called the method of class v_j (method dependency).

(4) Class v_i returned an object of class v_j (return dependency).

(5) Method of class v_i took the object of the class v_j as a parameter (parameter dependency).

Figure 1 gives a simple undirected weighted dependency graph between 21 classes of the Tomcat project; classes 2, 3, 4, 5, 6, and 7 have a direct dependency with class 1, and the corresponding edge weights were 3.0, 6.0, 8.0, 5.0, 2.0, and 3.0.

3.2. Centrality Measures. The definition of h-index given by Hirsh is as follows: "Suppose that the number of papers published by a scholar is N_p (descending order according to the number of references), if the references of top h articles

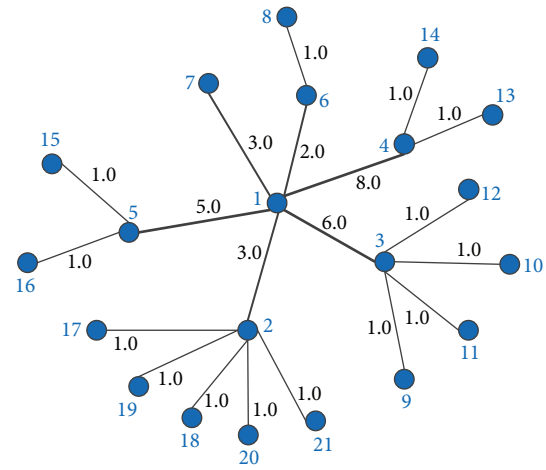

Figure 1: A simple weighted software dependency network.

are not less than h, the scholar is considered have index h." Using this concept proposed by Hirsh, we apply h-index to SDN and define four new centrality measures.

aN-Degree: aN-Degree of a node is the average degrees of its top k neighbors such that the degree of each is not less than k.

mN-Degree: mN-Degree of a node is the max k such that the degree of its top k neighbors together is at least k^2.

aE-Weight: aE-Weight of a node is the average weights of its top k edges that have at least a weight of k.

mE-Weight: mE-Weight of a node is the max k such that its top k edges together have minimum weight of k^2.

The aN-Degree of node 1 is 14/3 as its top 3 neighbors have degree of at least 3. The mN-Degree is 4 as sum of the top 4 neighbors' degrees is greater than 4^2. The aE-Weight is the average of the top 3 edges' weights (8 + 6 + 5 divided by 3). The mE-Weight is 5 as the sum of the weights of the top 5 edges is not less than 5^2 (see Table 1).

4. Experiments

4.1. Data. Three open-source projects are chosen as the research objects in this paper. Table 2 lists the statistical information. Tomcat (http://tomcat.apache.org/) is a free Web

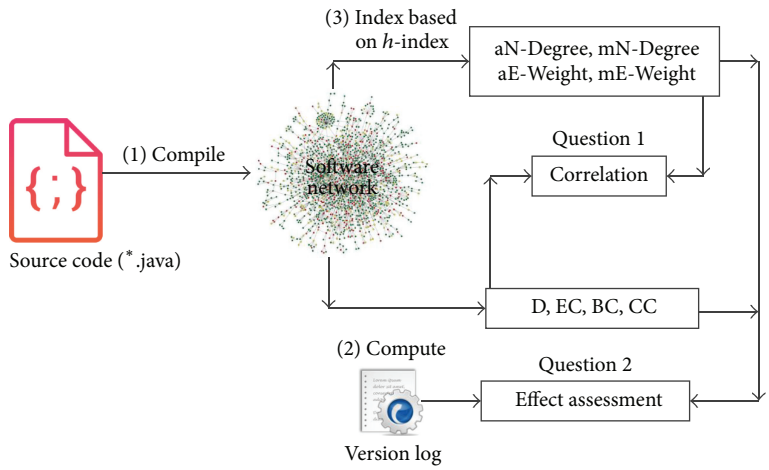

Figure 2: The framework of our experiment.

Table 2: The information of projects used in our experiment.

Projects	Version	Line of code	Number of classes	Number of edges
Tomcat	6.0.14	159,749	2152	10,765
Ant	1.9.0	129,240	1228	4848
JUNG	1.5.0	28,465	404	1737

application developed by Java, from Apache. Ant (http://ant.apache.org/) is also from Apache, a known well open-source project which serves to compiling, testing, and deployment. JUNG (http://sourceforge.net/projects/jung/, Java Universal Network/Graph Framework) is a common framework for modeling, analyzing, and development.

4.2. Research Problems. In this paper, the research mainly focuses on the following two questions:

(1) Do the Proposed Centrality Measures Work Well? As mentioned above, in complex network and social network analysis, various measures used to identify the important nodes have been proposed. Thus, it is worth analyzing the correlation between the proposed centrality measures and existing centrality measures.

(2) Do the Proposed Centrality Measures Identify More Key Classes in SDN? To further investigate the usefulness of the proposed centrality measures, we compared them with several existing centrality measures by analyzing the ability of identifying key classes in the actual maintenance, in the context of weighted software networks.

4.3. Experiment Design. In this paper, we first present the framework of our work (see Figure 2). It mainly consists of three parts. First, we used Dependency Finder and SNAT—a kind of network analysis tool which is developed by us to parse source codes and to extract classes and their dependencies. Then, we built SDN at the class level. Second, we calculated the centrality of all nodes in SDN and analyzed the correlation between four existing centrality measures and the proposed ones. Finally, according to the version control log derived by TortoiseSVN, we further validated the feasibility of the proposed centrality measures.

4.4. Results. We organize our results according to the two research questions proposed in Section 4.2.

(1) Do the Proposed Centrality Measures Work Well? The purpose of various centrality measures is to sort the importance of classes in software system. The greater the measure values, the more important the class is. For the proposed measures, in order to test their feasibility, we first need to quantify their relationship with other commonly used centrality measures. In other words, we should ensure the results are significantly related as a whole. Note that, in this paper, we introduce four typical centrality measures: node degree, betweenness centrality (BC), closeness centrality (CC), and eigenvector centrality (EC).

Considering our purpose is to compute the correlation of two groups of results, Kendall rank correlation coefficient can be used for analysis (using tau-b (k) in SPSS). $\tau_b = 1$ represents that the results are completely positively correlated; $\tau_b = -1$ represents the results are not relevant. Table 3 shows that there is a strong correlation between two groups of centrality measures, except for EC. For example, for JUNG the correlation coefficient is up to 0.9 between aN-Degree and node degree and up to 0.6 between aN-Degree and CC. Furthermore, although the correlation coefficients between the proposed centrality and EC are small, the statistical results show that they are significant. Figure 3 gives the relationship among aN-Degree and four typical centrality measures in three software projects, respectively.

Therefore, on the one hand, there is a significant correlation between the proposed centrality measure and four benchmark measures. On the other hand, the consistency within mE-Weight and other centrality measures is more

Table 3: The correlative coefficient τ_b.

	Project	Degree	EC	BC	CC
aN-Degree	Tomcat	0.861	0.125**	0.629	0.626
	Ant	0.889	0.230**	0.439	0.615
	JUNG	0.943	0.223**	0.578	0.628
mN-Degree	Tomcat	0.878	0.262**	0.628	0.710
	Ant	0.862	0.384**	0.720	0.912
	JUNG	0.933	0.244**	0.665	0.647
aE-Weight	Tomcat	0.921	0.151**	0.682	0.451
	Ant	0.994	0.144**	0.791	0.568
	JUNG	0.938	0.407**	0.661	0.618
mE-Weight	Tomcat	0.981	0.311**	0.564	0.675
	Ant	0.980	0.342**	0.570	0.650
	JUNG	0.987	0.305**	0.587	0.676

Note: ** represents the significant correlation.

obvious. In a word, the results show that the proposed centrality measures work as well as four benchmark centrality measures.

(2) Do the Proposed Centrality Measures Identify More Key Classes in SDN? In the previous question, we learnt that the proposed centrality measures show a remarkable consistency in measuring the key classes with existing network centrality measures. However, whether the important nodes identified by these proposed measures are key classes in the actual system is still unknown, especially the important nodes identified by mN-Degree and mE-Weight. A key class might be more complex because it is associated with other classes and might be highly reused because it relies on more classes. In the process of software maintenance, the chance to change this kind of class often is greater.

Therefore, we export the corresponding revision information from version control log for each software system and have access to the number of revisions of these classes during the period of change. Figure 4 shows that the changed times of a class are positive to its centrality measure value as a whole. For example, the value of R^2 (determinate coefficient) is up to 0.864 between the average values on mE-Weight measure and the changed times of classes. That is, the classes with greater centrality are changed more frequently. On the other hand, the results validated that centrality measures are useful to identify the key classes in software systems.

Given that the results in the other two projects have similar tendency to that of the Tomcat project, only the case obtained in Tomcat was given. Figure 4 also shows that the trends from our proposed centrality measures are more obvious than CC because of the greater values of R^2. Note that, except for mE-Weight, BC measure performs better than the proposed measures. The results further validated the advantage of BC measure as mentioned in [6, 30]. Meanwhile, it is clear that mE-Weight measure performs best, which indicated that computing the centrality on the weighted edge of nodes has an advantage compared to that based on the node degree and even prior to other frequently used centrality measures.

Table 4: The proportion of actual modified classes in top k key classes returned by different centrality measures (Tomcat).

Top k	5	10	30	50	100	150	200
aN-Degree	0.00	0.200	0.300	0.500	0.450	0.660	0.630
mN-Degree	0.20	0.200	0.367	0.520	0.550	0.700	0.740
aE-Weight	0.000	0.100	0.333	0.520	0.600	0.733	0.685
mE-Weight	0.000	0.300	0.600	0.580	0.660	0.793	0.875
Degree	0.000	0.100	0.300	0.400	0.500	0.673	0.610
EC	0.000	0.100	0.267	0.400	0.440	0.527	0.560
BC	0.200	0.200	0.367	0.480	0.590	0.727	0.660
CC	0.000	0.100	0.333	0.480	0.470	0.653	0.625

Besides, the proportion of the classes that have actually been modified in the top k key classes returned by different centrality measures were reported. Table 4 shows that the proportion becomes larger with the increase of k value. When k is 5, the most important five classes have not been modified, except for the case of mN-Degree and BC. A possible explanation is that the core framework should keep stable and in general is less likely to modify during the process of maintenance of software system. When k is set to 200, the proportion (7 out of 8) is more than 60%, especially for the mN-Degree and mE-Weight. For instance, when using mE-Weight measure, 87.5% of the top 200 classes were successfully recognized that they needed to be changed. In addition, in Table 4 the proportion from mE-Weight is larger than other measures as a whole. Compared to BC, the improvement is up to 0.215. It further verified the advantage of mE-Weight to identify key classes in software engineering.

In a word, compared to existing network centrality measures, the proposed measures do identify more key classes in software network, especially for the mE-Weight measure.

5. Conclusion

This paper puts forward four centrality measures based on h-index to compute the importance of a class in software system from two aspects: the degree of its neighbors and the weight of the edges that connected the current class and its neighbors. Taking three open-source software projects as the research objects, the results indicate that the proposed measures not only are able to identify the key classes as some commonly used centrality measures (correlative coefficient 0.987) but also perform better than some commonly used centrality measures (the improvement is at least 0.215). In addition, the finding suggests that mE-Weight defined by the weight of a node's top k edges performs best as a whole.

The work will help new developers understand the core parts of a software system faster and provide a guideline for the priority of class modification in software maintenance. In future, we will further validate the measures proposed in this paper with more open-source software applications and apply these measures to software network from the other

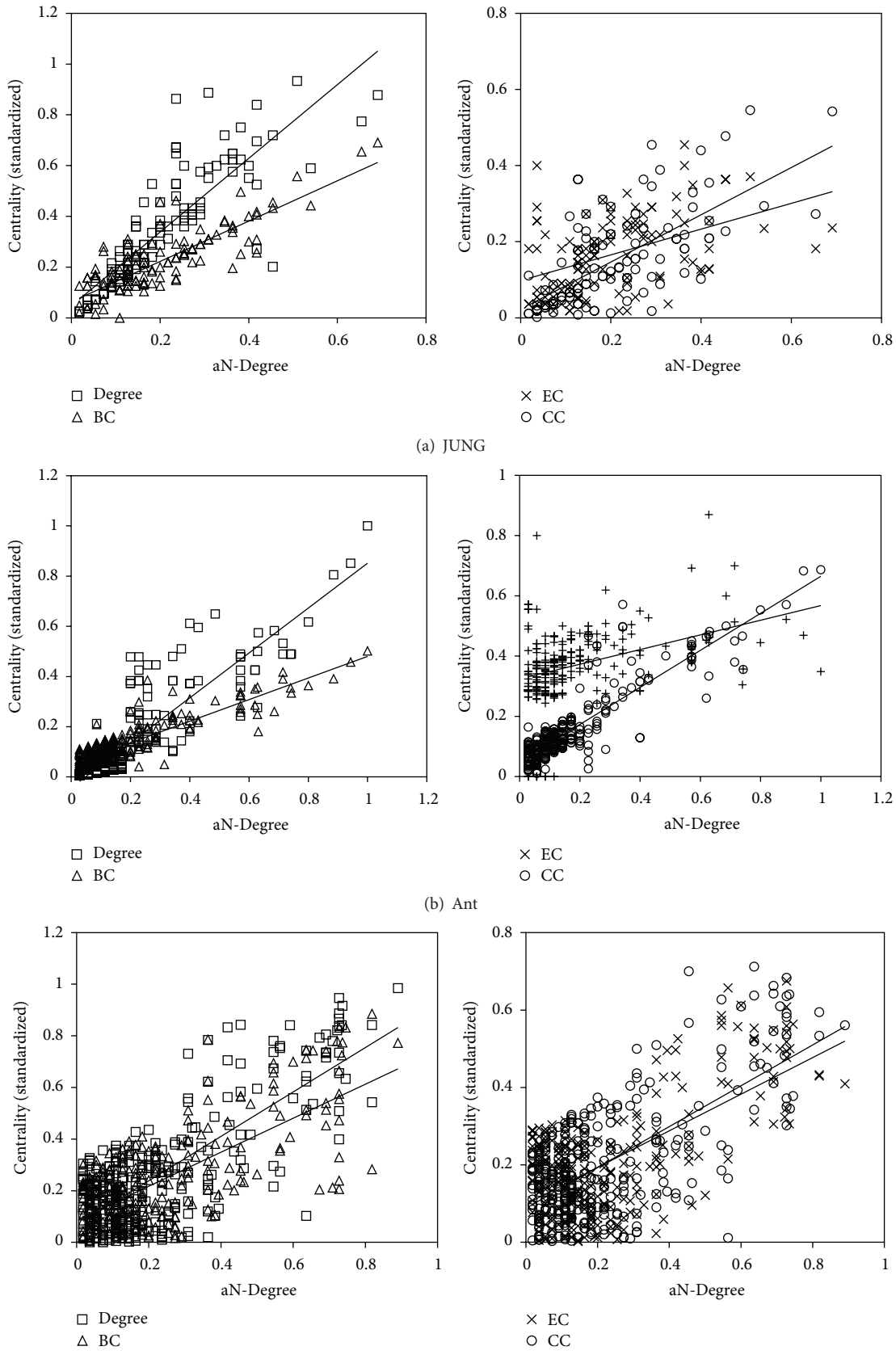

Figure 3: An example: the relationship among *aN-Degree* and other centrality measures in three software projects.

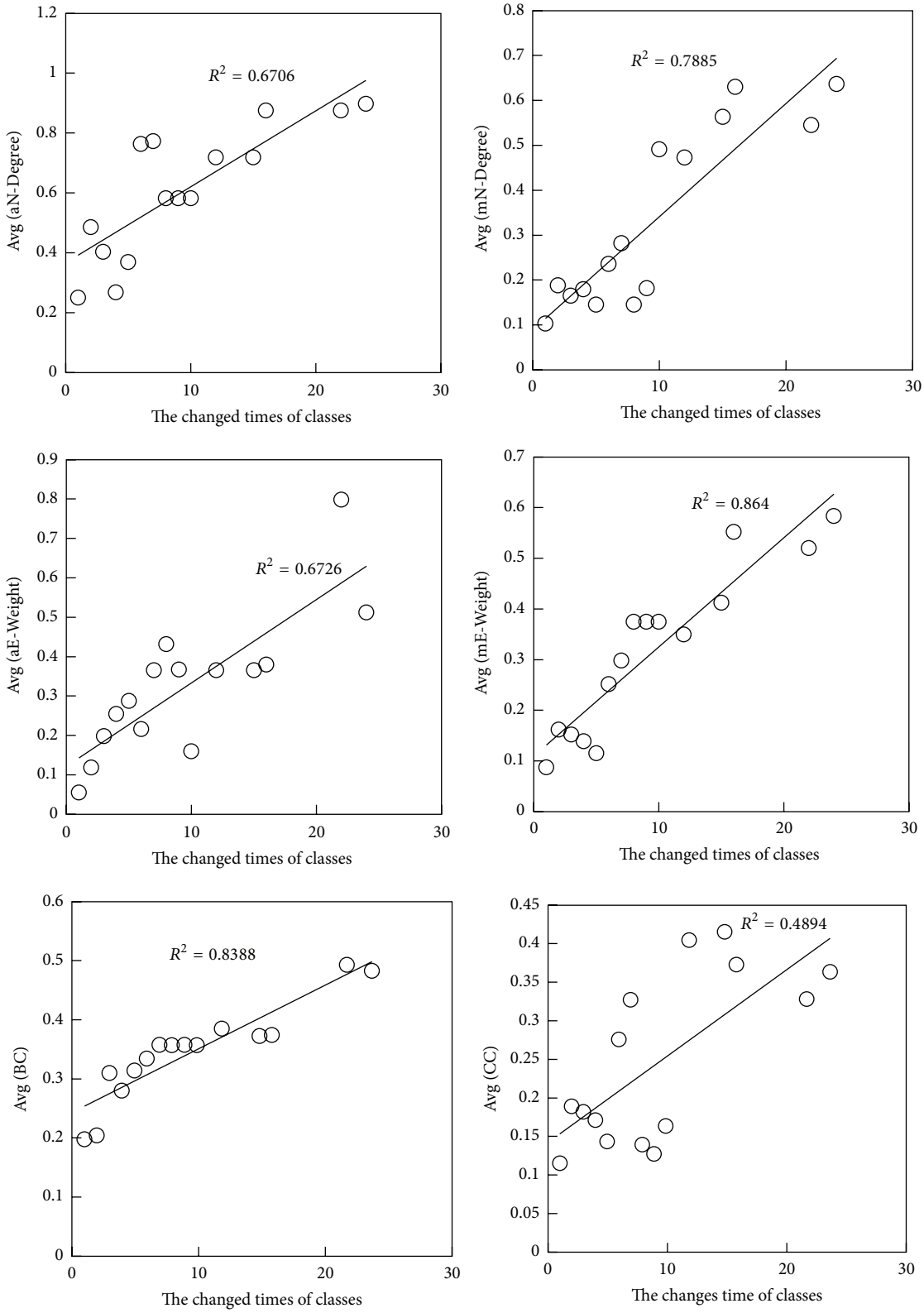

Figure 4: The relationship between the changed times of classes and four proposed centrality measures (Tomcat project).

granularity levels, such as package and feature, to verify the practicability of proposed centrality measures.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This work is supported by the National Key Research and Development Program of China under Grant no. 2016YFB0800400, the 973 Program of China under Grant no. 2014CB340401, the National Natural Science Foundation of China under Grants nos. 61572371, 61273216, and 61272111, and the China Postdoctoral Science Foundation under Grant no. 2015M582272.

References

[1] D. Steidl, B. Hummel, and E. Juergens, "Using network analysis for recommendation of central software classes," in *Proceedings of the 19th Working Conference on Reverse Engineering*, pp. 93–102, IEEE, Kingston, Canada, October 2012.

[2] L. Jian-Guo, R. Zhuo-Ming, G. Qiang, and W. Bing-Hong, "Node importance ranking of complex networks," *Acta Physica Sinica*, vol. 62, no. 17, Article ID 178901, pp. 1–9, 2013.

[3] X.-L. Ren and L.-Y. Lü, "Review of ranking nodes in complex networks," *Chinese Science Bulletin*, vol. 59, no. 13, pp. 1175–1197, 2014.

[4] R. Sun and W. Luo, "Review on evaluation of node importance in public opinion," *Application Research of Computers*, vol. 29, no. 10, pp. 3606–3608, 2012.

[5] Z.-M. Ren, F. Shao, J.-G. Liu, Q. Guo, and B.-H. Wang, "Node importance measurement based on the degree and clustering coefficient information," *Acta Physica Sinica*, vol. 62, no. 12, Article ID 128901, 2013.

[6] M. Kitsak, L. K. Gallos, S. Havlin et al., "Identification of influential spreaders in complex networks," *Nature Physics*, vol. 6, no. 11, pp. 888–893, 2010.

[7] J. E. Hirsch, "An index to quantify an individual's scientific research output," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 102, no. 46, pp. 16569–16572, 2005.

[8] T. Opsahl, F. Agneessens, and J. Skvoretz, "Node centrality in weighted networks: generalizing degree and shortest paths," *Social Networks*, vol. 32, no. 3, pp. 245–251, 2010.

[9] M. C. Wang and W. F. Pan, "A comparative study of network centrality metrics in identifying key classes in software," *Journal of Computational Information Systems*, vol. 8, no. 24, pp. 10205–10212, 2012.

[10] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," in *Proceedings of the 30th International Conference on Software Engineering (ICSE '08)*, pp. 531–540, ACM, Leipzig, Germany, May 2008.

[11] A. Meneely, L. Williams, W. Snipes, and J. Osborne, "Predicting failures with developer networks and social network analysis," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT '08)*, pp. 13–23, ACM, November 2008.

[12] K. Crowston, K. Wei, Q. Li, and J. Howison, "Core and periphery in free/libre and open source software team communications," in *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS '06)*, vol. 6, p. 118a, IEEE, Kauia, Hawaii, USA, January 2006.

[13] M. Pinzger, N. Nagappan, and B. Murphy, "Can developer-module networks predict failures?" in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT '08)*, pp. 2–12, ACM, November 2008.

[14] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, "Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities," *IEEE Transactions on Software Engineering*, vol. 37, no. 6, pp. 772–787, 2011.

[15] T. Zimmerman, N. Nagappan, K. Herzig, R. Premraj, and L. Williams, "An empirical study on the relation between dependency neighborhoods and failures," in *Proceedings of the 4th IEEE International Conference on Software Testing, Verification and Validation (ICST '11)*, pp. 347–356, IEEE, Berlin, Germany, March 2011.

[16] P. Bhattacharya, M. Iliofotou, I. Neamtiu, and M. Faloutsos, "Graph-based analysis and prediction for software evolution," in *Proceedings of the 34th International Conference on Software Engineering (ICSE '12)*, pp. 419–429, IEEE, Zürich, Switzerland, June 2012.

[17] D. Steidl, B. Hummel, and E. Juergens, "Using network analysis for recommendation of central software classes," in *Proceedings of the 19th Working Conference on Reverse Engineering (WCRE '12)*, pp. 93–102, IEEE, Kingston, Canada, October 2012.

[18] M. S. Zanetti, I. Scholtes, C. J. Tessone, and F. Schweitzer, "Categorizing bugs with social networks: a case study on four open source software communities," in *Proceedings of the 35th International Conference on Software Engineering (ICSE '13)*, pp. 1032–1041, IEEE, San Francisco, Calif, USA, May 2013.

[19] F. Perin, L. Renggli, and J. Ressia, "Ranking software artifacts," in *Proceedings of the 4th Workshop on FAMIX and Moose in Reengineering (FAMOOSr '10)*, p. 120, 2010.

[20] A. Zaidman and S. Demeyer, "Automatic identification of key classes in a software system using webmining techniques," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 20, no. 6, pp. 387–417, 2008.

[21] W.-F. Pan, B. Li, Y.-T. Ma, and B. Jiang, "Identifying the key packages using weighted PageRank algorithm," *Acta Electronica Sinica*, vol. 42, no. 11, pp. 2174–2183, 2014.

[22] Y.-M. Zhou and B.-W. Xu, "Dependence structure analysis-based approach for measuring importance of classes," *Journal of Southeast University: Natural Science Edition*, vol. 38, no. 3, pp. 380–384, 2008.

[23] P. Meyer, H. Siy, and S. Bhowmick, "Identifying important classes of large software systems through K-core decomposition," *Advances in Complex Systems*, vol. 17, no. 7-8, Article ID 1550004, 2014.

[24] S.-J. Jiang, X.-L. Ju, X.-Y. Wang, H.-Y. Li, Y.-M. Zhang, and Y.-Q. Liu, "Measuring the importance of classes using UIO sequence," *Acta Electronica Sinica*, vol. 43, no. 10, pp. 2062–2068, 2015.

[25] M. Kamran, F. Azam, and A. Khanum, "Discovering core architecture classes to assist initial program comprehension," in *Proceedings of the 2012 International Conference on Information Technology and Software Engineering: Information Technology & Computing Intelligence*, vol. 211 of *Lecture Notes in Electrical Engineering*, pp. 3–10, Springer, Berlin, Germany, 2013.

[26] I. Sora, "A PageRank based recommender system for identifying key classes in software systems," in *Proceedings of the IEEE 10th Jubilee International Symposium on Applied Computational Intelligence and Informatics (SACI '15)*, pp. 495–500, Timisoara, Romania, May 2015.

[27] W. Pan, B. Hu, B. Jiang, and B. Xie, "Identifying important packages of object-oriented software using weighted k-core decomposition," *Journal of Intelligent Systems*, vol. 23, no. 4, pp. 461–476, 2014.

[28] I. Şora, "Finding the right needles in hay: helping program comprehension of large software systems," in *Proceedings of the 10th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE '15)*, pp. 129–140, April 2015.

[29] M. Wang, H. Lu, Y. Zhou, and B. Xu, "Identifying key classes using h-index and its variants," *Journal of Frontiers of Computer Science and Technology*, vol. 5, no. 10, pp. 891–903, 2011.

[30] P. He, B. Li, Y. Ma, and L. He, "Using software dependency to bug prediction," *Mathematical Problems in Engineering*, vol. 2013, Article ID 869356, 12 pages, 2013.