Hindawi
Mathematical Problems in Engineering
Volume 2020, Article ID 5346498, 7 pages
https://doi.org/10.1155/2020/5346498

Review Article

Modeling Software Systems as Complex Networks: Analysis and Their Applications

Hao Li,[1] **Tian Wang**,[1] **Xinxin Xu**,[1] **Bo Jiang**,[1] **Jianliang Wei**,[2] **and Jiale Wang**[1]

[1]*School of Computer Science and Information Engineering, Zhejiang Gongshang University, Hangzhou 310018, China*
[2]*School of Management Engineering and E-commerce, and Contemporary Business and Trade Research Center, Zhejiang Gongshang University, Hangzhou 310018, China*

Correspondence should be addressed to Jianliang Wei; 2356862895@qq.com

Received 10 January 2020; Accepted 25 February 2020; Published 27 April 2020

Guest Editor: Hua Ming

Software systems are of great importance, whose quality will influence every walk of our life. However, with increase in their scale and complexity, we are unable to control their quality since only little is known about their actual internal structure. "We cannot control what we cannot measure." Thus, to control these complex software systems, the first task that we should do is to measure their internal structure. In recent years, people applied the theories and techniques in the field of complex networks to systematically investigate the structure of software systems by representing software systems as networks (i.e., software networks), and many interesting and useful results have been revealed. In this work, we aim to briefly review some recent research advances in the interdisciplinary research between complex networks and software engineering, including modeling, analysis, and applications. Specifically, we first describe some novel techniques to model the structural details of a specific software system. Then, based on these modeling techniques, we introduce some research work on characterizing the static and dynamic structural properties of software systems. Third, we describe some promising applications of software networks in real-world scenarios. Finally, we suggest some future research topics.

1. Introduction

Nowadays, software systems have almost been used in every walk of life. Thus, how to provide a piece of software with high quality has been a problem attracting a lot of attention. However, with increase in the scale and complexity of software systems, it is a hard task to control the quality of a specific piece of software, especially when we know very little about the internal complexity of a specific software system [1, 2]. It is well known that we cannot control what we cannot measure. Thus, to control the quality of software systems, the first task that we should do is to measure their internal complexity [3]. Software structure which is defined as the software elements (e.g., attributes, methods, classes, interfaces, and packages) and their couplings (e.g., "method-call" couplings between methods and "inheritance" between classes) have been one of the most important factors that may influence the software complexity and further influence the quality of the software. Thus, how to measure and even to control the complexity of a software system has been a challenge faced by many researchers [4]. There is an urgent need to develop a systematic approach to deeply explore the internal structure of software systems.

Networks (or graphs) provide a natural and most adequate representation of the software structure; i.e., software elements are nodes and the couplings between software elements are edges (or links). Though network representation is not novel in software engineering, its form is simple and intelligible, which makes it feasible to perform the network analysis of software structure by using theories and techniques in the field of complex networks, and many significant discoveries and research results have been provided in the last decade [1, 2]. Note that such a network representation of the internal software structure of a specific software system is usually termed "software networks," a notion similar to "complex networks" [5].

In this work, we aim to briefly review some recent research advances in the field of software networks, highlighting different techniques in modeling, analysis, and applications. Specifically, we first describe some novel techniques to model the structural details of a specific software system. Then, based on these modeling techniques, we introduce some research work on characterizing the static and dynamic structural properties of software systems. Third, we describe some promising applications of software networks in real-world scenarios. Finally, we provide some future research topics. Note that, in this work, we only review the most recent research work which is published in the last seven years (2013 to 2019). For research work published before 2013, please refer to the reviews [1, 2]. In this work, we only focus on the brief review of the most recent research work in the field of software networks, rather than their detailed comparison.

The rest of this paper is organized as follows: Section 2 introduces the related reviews on software networks. Section 3 describes the data set we used. Section 4 introduces the related advances in software networks from three perspectives, i.e., modeling, analysis, and applications. Section 5 outlines some future research topics. Finally, in Section 6, we conclude the paper.

2. Related Work

To the best of our knowledge, there are a total of three reviews related to software networks.

Li et al. [6] reviewed 36 research papers related to software networks in 2008. They organized the existing work into three groups, i.e., related work on discoveries of software structural properties, related work on models of software growth, and related work on software metrics based on software networks. In the related work on discoveries of software structural properties, they discussed some research work on revealing shared structural properties in software networks. In the related work on models of software growth, they discussed the proposed evolution models to characterize the software growth. In the related work on software metrics, they discussed the metrics which are based on software networks and are used to characterize software quality.

Pan's review examined 32 research papers on software networks published before 2011 [2]. He discussed the existing research from four perspectives, i.e., characterization of software networks, modeling of software networks growth, measurement of software networks, and application of software networks in software engineering. In the "characterization of software networks," he reviewed the work that aims to characterize the properties of software structures such as scale-free and small world at different levels of granularity. In the "modeling of software networks growth," he reviewed the work that aims to propose an evolution model to explain the growth of software structures. In the "application of software networks in software engineering," he reviewed the work that applied software networks in software engineering practices such as software refactoring and software selection.

Sbelj and Bajec [1] also reviewed the related work on software networks. First, they reviewed the work on discovering the shared structural properties such as scale-free and small world phenomena. Second, they reviewed the work on characterizing the dynamical properties of software networks such as bug propagation. Third, they reviewed some work on the application of software networks such as refactoring and software abstraction.

Our current review is different from those of Li, Pan, and Sbelj. We cover a different time period from 2013 to 2019. Thus, our focus is to review the very recent research work in the field of software networks and shed some lights on the future research topic.

3. Data Set

We searched the 7 most popular digital libraries, i.e., ACM, IEEE, Springer, Scopus, ISI, ScienceDirect, and Compendex and Inspec, to obtain a relatively complete list of the primary research work. When searching the digital libraries, we used the following search string:

(Java OR OO OR object-oriented OR object oriented OR package OR packages OR class OR classes OR OR interface OR interfaces OR method OR methods OR attribute OR attributes) AND (software network OR software networks OR complex networks OR complex network OR graph OR graphs).

The search string contains the major research terms and their alternative spellings from the titles and keywords of related work on software networks. We use Boolean expressions "AND" and "OR" to connect the major research terms and their alternative spellings, respectively. Note that, in a specific digital library, the search string should be adapted slightly according to the grammar that library uses. For example, in the Springer library, the above string should be written as follows:

(Java OR OO OR "object-oriented" OR "object oriented" OR package OR packages OR class OR classes OR interface OR interfaces OR method OR methods OR attribute OR attributes) AND ("software network" OR "software networks" OR "complex networks" OR "complex network" OR graph OR graphs).

Obviously, the results returned by each digital library may overlap. Thus, we should identify and remove the redundant results. Furthermore, we also exclude research papers based on their titles, abstracts, and full texts. Finally, our data set contains 30 research papers (see the References section).

4. Analysis and Discussion

The research papers in our data set can be roughly categorized into three groups, i.e., modeling, analysis, and applications. Papers in the "modeling" category focus on the novel techniques to model the structural details of a specific software system. Papers in the "analysis" category focus on characterizing the static and dynamic structural properties of software systems. Papers in the "application" category try to apply the software networks to solve some real-world

problems in software engineering. The three categories will be detailed in the following sections.

4.1. Modeling of Software Structure. Different types of software networks have been proposed to represent the structural details of a specific software system at different levels of granularity, such as associated software graphs [7], class diagram [8], and cyclic dependency graphs [9] (see Table 1). These software networks can be differentiated from the levels of granularity (i.e., package level, class level, and method (or attribute) level). Furthermore, these software networks can also be differentiated from the nature of couplings, i.e., whether their couplings are directed or weighted.

As is shown in Table 1, we can observe that, in the method level software networks, nodes represent methods and edges (or links) represent the method call relations between methods. We can use the frequency of method calls to weigh the edge (or link) with the aim to signify the coupling intensity that might exist between the two methods. Edge can also be directed to denote the coupling direction. The existing two software networks at the method level (i.e., weighted networks [10] and FCN [12]) are all not very accurate to describe the software structure. Weighted networks ignored the reference relations between methods and attributes, while FCN ignored the coupling direction. Thus, we can combine the two software networks to build a much more accurate software network, i.e., weighted directed feature coupling network (WDFCN). We use this feature to denote methods and attributes. In the WDFCN, nodes denote features, edges denote method class relations and method-attribute reference relations, weights on the edges denote the coupling frequencies, and the direction of edges denotes coupling directions.

In the class level software networks, nodes represent classes (or interfaces) and edges (or links) represent the couplings between classes (i.e., inheritance and implement) and couplings between the methods and attributes the classes contain (i.e., parameter, global variable, local variable, return type, and method call). Edges can also be assigned weights to signify the coupling intensity between classes (or interfaces) and can also be directed to denote the coupling direction. In the existing class-level software networks, CCN and MCN are much more accurate than others. However, they ignored two important coupling types between classes, i.e., the reference relations between methods and attributes the classes contain and the instantiate relations between classes. Thus, CCN and MCN can be improved by considering the two coupling types.

In the package-level software networks, nodes represent packages and edges (or links) represent the couplings between packages, which are derived from the couplings between classes. Edges can also be assigned weights to signify the coupling intensity and can also be directed to denote the coupling direction. In the existing two types of package-level software networks, PDN is much more accurate than MPN. However, as that in CCN and MCN, PDN and MPN also ignored two important coupling types between classes, i.e., the reference relations between methods and attributes the classes contain and the instantiate relations between classes. Thus, PDN and MPN can also be improved by considering the two coupling types.

Note that, compared with the research work on software networks published before 2013, the main difference of the software recently built is that they took into consideration much more information in the software systems such as different coupling types, coupling frequencies, and the nature of couplings. The software networks recently built are much more accurate. But they still ignored some information, e.g., the reference relations between methods and attributes the classes contain and the instantiate relations between classes. If the software networks we built are not very accurate, the results or findings that we obtained from experimental studies may contain errors. Thus, there is still much more work that we can do.

In fact, how accurately the software networks can describe the software systems depends on the tools that are used to extract the information enclosed in the software. To the best of our knowledge, many research papers only provide their software network models and show their results or findings. They usually do not mention the tools that they used to build software networks. Pan et al. [4, 5, 12] developed a software network analysis platform (SNAP) to build many types of software networks at different levels of granularity. Their tools can be obtained via the URLs provided in their work. By their tools, we can build all the abovementioned software networks.

4.2. Analysis of Software Networks. Chaikalis and Chatzigeorgiou [18] proposed a network-based prediction model to characterize the growth of software systems. Their model took into consideration both of the information from past data and domain-related rules.

Wang and Xiao [10] represented the runtime structure of the Linux operating system as a weighted network, where nodes represent functions and edges represent function calls. Based on the weighted network, they explored the execution process of Linux by using theories and techniques in complex networks. They found that the weight distribution follows a power-law distribution, the process management component of Linux plays the most important role, and the reliability of Linux declines with the versions from 3.15 to 4.4.

Yang et al. [11] modeled software systems as Function Call Networks (FCNs), where nodes represent functions and edges represent function calls. Based on the FCNs, they characterized the software structure using a set of measurements from the perspective of modularity, hierarchy, complexity, and fault propagation. They also proposed a model to quantify software structural quality, which gives a better understanding of the evolution of software systems.

Trindade et al. [19] represented software at the class level as Little House, where nodes represent classes and edges represent dependencies among classes. Based on the Little House, they analyzed 81 versions of 6 software systems and found some software evolution patterns. These patterns are

Table 1: Summary of the existing software networks.

Name	Level	Nodes	Couplings	Directed?	Weighted?
Weighted network [10], FCNs [11]	Method	Methods	Method call	Yes	Yes
FCN [12]	Method	Methods and attributes	Method call, method-attribute reference	No	Yes
Associated software graphs [7]	Class	Classes	Inheritance, composition, dependence	No	No
Class diagram [8]	Class	Classes	Dependency, common association, qualified association, association class, aggregation association, composition association, generalization, binding, generalization, realize	Yes	Yes
Cyclic dependency graphs [9]	Class	Classes and interfaces	Inheritance	No	No
DTMC [13]	Class	Classes	Method call, method-attribute reference	Yes	Yes
WCCN [3, 4, 12, 14]	Class	Classes and interfaces	Inheritance, implement, parameter, global variable, local variable, return type	No	Yes
CCN [15]	Class	Classes and interfaces	Inheritance, implement, parameter, global variable, local variable, method call, return type	Yes	Yes
MCN [5, 16]	Class	Classes and interfaces	Inheritance, implement, parameter, global variable, local variable, method call, return type	Yes	Yes
MPN [17]	Package	Packages	Inheritance, implement, global variable, method call	Yes	Yes
PDN [5]	Package	Packages	Inheritance, implement, parameter, global variable, local variable, method call, return type	Yes	Yes

further applied to define a software evolution model to characterize software evolution and growth.

Pan et al. [16] use a multilayer network at the class level to model software systems, where nodes are classes and interfaces and edges are different coupling types between classes (or interfaces). In their model, a specific type of coupling forms a layer. They used an aggregation approach to analyze the multilayer structure of a specific software system by using a set of 10 topological measures from complex networks. It is the first work to represent software systems as multilayer networks, providing a novel perspective to analyze software systems.

Note that the abovementioned papers on analysis of software networks used a traditional way to explore the growth of software systems from a structural perspective and also the structural properties enclosed in the software systems, but some of them took a new perspective. Specifically, Chaikalis and Chatzigeorgiou characterize software evolution from a network perspective, Wang and Xiao built the software network from execution process of the software, Yang et al. characterized the software structure by using the dynamic process of faults, Trindade and Orfano tried to use a model to characterize software evolution and growth, and Pan et al. used a multilayer networks, which is a much more accurate software network model.

4.3. Applications of Software Networks

4.3.1. Software Metrics. Gu et al. [20] proposed metrics to quantify the class cohesion from a complex network perspective.

Pan and Chai [3] modeled software systems at the class level as a weighted directed software network, and based on the network, they proposed a metric, NIN, to quantify the coupling intensity of two classes. They further proposed a metric to quantify the class stability. In [14], they further proposed a simulation way to calculate the software stability, which is based on the analysis of change propagation dynamics in the software structure.

In [12], Pan et al. modeled software systems as feature coupling networks (FCNs), where nodes are methods and attributes and edges are method-call relations and method-attribute reference relations. Based on the FCNs, they borrowed some idea from community detection techniques in complex networks and used the metric "modularity" to quantify the modularity of a specific software system.

Obviously, the recent research work on software metrics followed the traditional line of thoughts of the related work published before 2013. The only difference is they used a much more accurate software network model and characterized the software structure from a different perspective.

4.3.2. Bug Prediction. Concas et al. [7] used an Associated Software Graph (ASG) to represent software systems at the class level, where nodes represent classes and edges represent the "inheritance," "composition," and "dependence" relations between classes. Based on the ASG, they computed the number of communities, modularity of the software network, and other network metrics such as clustering coefficient, average path length, and mean degree. Then, they analyzed the correlation between these metrics and the number of bugs in the software. They found that medium-size systems with community structures tend to be buggy.

Yang et al. [11] proposed a software class network to represent software systems at the class level. In the software network, classes are nodes, and the calling relations between the methods that every pair of classes contain constitute the edges. Based on the software network, they proposed a set of metrics to characterize the software network structure and used some machine-learning algorithms to construct defect prediction models. Their results showed promising results.

Zakari et al. [21] proposed a software network at the statement level, where statements are nodes and the execution traces between statements are edges. Based on the software network, they computed two centrality metrics (i.e., degree centrality and closeness centrality) for fault diagnosis. Experimental results showed their approach is promising and better than existing fault localization techniques.

Obviously, in the existing research work on fault prediction, software networks are usually used to calculate some structural metrics and the structural metrics can be used to correlate with bugs or be used in traditional prediction models to improve fault prediction performance. However, the software networks the existing approaches used are not very accurate, which makes the metrics obtained inaccurate. Thus, in the future, we can use a much more accurate software network to compute structural metrics.

4.3.3. Software Refactoring. Pan et al. [22] modeled the software structure at the method level as SFN, where nodes represent methods and attributes and edges represent method-call relations and method-attribute reference. Then, they applied an evolutionary algorithm to optimize software structure and detect the methods to be moved. In their algorithm, they optimized a function which is based on software modularity. In [23], Pan et al. proposed a similar approach to identify the classes to be moved.

In [24], Wang et al. represented software at the class level as a Class-Level Multirelation Directed Network (CMDN), where nodes are classes and edges are the coupling between classes, i.e., inheritance, association, and aggregation. Based on the CMDN, they used the community detection algorithm to identify many refactoring opportunities simultaneously. Experimental results showed that their approach is better than some existing approaches.

There are many other refactorings in object-oriented software systems. However, the existing work only considered three refactorings, i.e., move method refactoring, move field refactoring, and extract class refactoring. Many other refactorings such as extract method, pull up method, and inline class need further exploration.

4.3.4. Key Class Identification. Meyer et al. [25] modeled software systems at the class level as a software network and applied the coreness in the k-core decomposition to measure class importance in the software network. The coreness is further used as a criterion to rank classes.

Şora and Chirila [26] recently modeled software systems as graph, where nodes represent classes and edges represent the couplings between classes. Weights are assigned to the edges to measure the coupling intensity. Then, they applied PR-U2-W, CONN-TOTAL, and CONN-TOTAL-W to measure class importance, respectively.

In [27], Luo et al. proposed an extend call graph to represent methods and their calling relations and utilized a VertexRank algorithm to quantify the importance of methods.

In [28], He et al. modeled software systems as a weighted software network, where nodes are methods and edges are their calling relations. Then, they applied a PageRank-like algorithm to quantify the importance of methods.

In [15], Pan et al. modeled software systems at the class level as a weighted directed software network, where nodes are classes and interfaces, edges are the 7 types of couplings between classes (or interfaces), and the weights on the edges are the coupling frequencies. Then, they proposed a generalized k-core decomposition to quantify the importance of classes. In [5], they further proposed a multilayer software network at the class level. Based on the software network, they compute the importance of classes at each layer and further combine the class importance at each layer to obtain the final importance.

The software network proposed in [5] is the best accurate one in the existing research work. But the authors ignored two important types of couplings, i.e., the reference relations between methods and attributes the classes contain and the instantiate relations between classes. Thus, there is still much room for improving the existing work on key classes identification. We can also use improved ranking algorithm to improve the performance of the existing approaches. To the best of our knowledge, there is no work on identifying important software elements at other levels of granularity.

5. Future Research Topics

Based on the brief review of the related work on software network, we proposed the following research topics that we can carry out in the future:

(i) Much more accurate software networks at different levels of granularity: for example, in the existing software networks, no one considered all the coupling types that might exist between classes. Thus, much work should be performed to consider much more information in the software.

(ii) Runtime software networks: the majority of the software networks is constructed statically from the source code or bytecode of a specific software system. Only one research paper [10] reported constructing a runtime software network. Thus, much work can be carried out on runtime software network modeling, analysis, and applications.

(iii) Software evolution model: software evolution models are used to characterize the software evolution and growth. They should reflect the properties enclosed in software systems. Thus, if we find more properties of software, the evolution model can be updated.

(iv) Bug prediction: we can propose many software metrics to characterize software structure and further use them to improve any bug prediction models.

(v) Software refactoring: much more work can be proposed to identify other refactoring opportunities such as extract methods, pull up methods, and inline classes.

(vi) Software comprehension: identifying important software elements can be used to aid people

understand a specific software system. Much more work can be carried out on identifying important software elements at other levels of granularity (i.e., package level, method level, or even statement level). Furthermore, much more work can also be performed to guide the specific comprehension process of a software system.

(vii) Service-oriented system analysis: software networks have also been used in service-oriented software systems. Pan et al. [29, 30] used software networks to represent API and their couplings in service-oriented systems and applied community detection algorithm to organize APIs into clusters. Thus, in the future, we can also perform service-oriented software modeling, analysis, and applications.

6. Conclusions

This paper briefly reviewed the recent advances in the research field of software networks from 2013 to 2019. First, we described the data set we used, i.e., the research work published in the time period of 2013 to 2019. Then, we briefly described the existing work from three perspectives, i.e., modeling, analysis, and applications. Specifically, we reviewed the software networks that used to model the structural details of specific software systems and highlighted the problems in the existing models. We briefly introduced some research work on characterizing the static and dynamic structural properties of software systems. We also described some promising applications of software networks in real-world scenarios such as software metrics, bug prediction, refactoring, and key element identification. Finally, we outlined some future research topics.

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 Key R&D Program of Zhejiang Province (Grant nos. 2019C01004 and 2019C03123) and the Humanities and Social Science Foundation of Chinese Ministry of Education (Grant nos. 17YJA870020 and 18YJA870010).

References

[1] L. Šubelj and M. Bajec, "Software systems through complex networks science: review, analysis and applications," in *Proceedings of the First International KDD Workshop on Software Mining (SoftwareMining 2012)*, pp. 9–16, Beijing, China, August 2012.

[2] W. F. Pan, "Applying complex network theory to software structure analysis," *World Academy of Science, Engineering and Technology*, vol. 60, pp. 1636–1642, 2011.

[3] W. Pan and C. Chai, "Measuring software stability based on complex networks in software," *Cluster Computing*, vol. 22, no. s2, pp. 2589–2598, 2019.

[4] 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.

[5] W. F. Pan, H. Ming, C. K. Chang, Z. J. Yang, and D.-K. Kim, "ElementRank: ranking Java software classes and packages using multilayer complex network-based approach," *IEEE Transactions on Software Engineering*, p. 1, 2019.

[6] B. Li, Y. T. Ma, J. Liu et al., "Advances in the studies on complex networks of software systems," *Advances in Mechanics*, vol. 38, no. 6, pp. 805–814, 2009.

[7] G. Concas, C. Monni, M. Orru et al., "Software systems through complex networks science: review, analysis and applications," in *Proceedings of the 4th International Workshop on Emerging Trends in Software Metrics (WeTSOM 2013)*, pp. 14–20, San Francisco, CA, USA, May 2012.

[8] 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.

[9] T. D. Oyetoyan, J. R. Falleri, J. Dietrich, and K. Jezek, "Circular dependencies and change-proneness: an empirical study," in *Proceedings of the 22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER 2015)*, pp. 241–250, Montreal, Canada, March 2015.

[10] H. Wang and G. Xiao, "Analysis of the runtime Linux operating system as a complex weighted network," in *Proceedings of the 2016 International Conference on Software Analysis, Testing and Evolution (SATE 2016)*, pp. 7–11, Kunming, China, November 2016.

[11] Y. Yang, J. Ai, X. Li, and W. E. Wong, "MHCP model for quality evaluation for software structure based on software complex network," in *Proceedings of the IEEE 27th International Symposium on Software Reliability Engineering (ISSRE 2016)*, pp. 298–308, Ottawa, Canada, October 2016.

[12] 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.

[13] S. M. Srinivasan, R. S. Sangwan, and C. J. Neill, "On the measures for ranking software components," *Innovations in Systems and Software Engineering*, vol. 13, no. 2-3, pp. 161–175, 2017.

[14] W. F. Pan, H. B. Jiang, H. Ming, C. Chai, B. Chen, and H. Li, "Characterizing software stability via change propagation simulation," *Complexity*, vol. 2019, Article ID 9414162, 17 pages, 2019.

[15] 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.

[16] W. Pan, B. Hu, J. Dong, K. Liu, and B. Jiang, "Structural properties of multilayer software networks: a case study in Tomcat," *Advances in Complex Systems*, vol. 21, no. 2, Article ID 1850004, 2018.

[17] W. F. Pan, B. Li, Y. T. Ma et al., "Identifying the key packages using weighted PageRank algorithm," *Acta Electronica Sinica*, vol. 42, no. 11, pp. 2174–2183, 2014, in Chinese.

[18] 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.

[19] R. P. F. Trindade, T. S. Orfano, K. A. M. Ferreira, and E. F. Wanner, "The dance of classes - a stochastic model for software structure evolution," in *Proceedings of the 4th*

International Workshop on Emerging Trends in Software Metrics (WeTSOM 2017), pp. 22–28, Buenos Aires, Argentina, May 2017.

[20] 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.

[21] A. Zakari, S. P. Lee, and C. Y. Chong, "Simultaneous localization of software faults based on complex network theory," *IEEE Access*, vol. 6, pp. 23990–24002, 2018.

[22] W. F. Pan, J. Wang, and M. C. Wang, "Identifying the move method refactoring opportunities based on evolutionary algorithm," *International Journal of Modelling, Identification and Control*, vol. 18, no. 2, pp. 182–189, 2013.

[23] W.-F. Pan, B. Jiang, and B. Li, "Refactoring software packages via community detection in complex software networks," *International Journal of Automation and Computing*, vol. 10, no. 2, pp. 157–166, 2013.

[24] Y. Wang, H. Yu, Z. Zhu, W. Zhang, and Y. Zhao, "Automatic software refactoring via weighted clustering in method-level networks," *IEEE Transactions on Software Engineering*, vol. 44, no. 3, pp. 202–236, 2018.

[25] 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. 07n08, Article ID 1550004, 2014.

[26] I. Şora and C.-B. Chirila, "Finding key classes in object-oriented software systems by techniques based on static analysis," *Information and Software Technology*, vol. 116, Article ID 106176, 2019.

[27] H. Luo, Y. Dong, Y. Y. Ng, and S. Wang, "VertexRank: importance rank for software network vertices," in *Proceedings of the 2014 IEEE 38th Annual Computer Software and Applications Conference (COMPSAC 2014)*, pp. 251–260, Vasteras, Sweden, 2014.

[28] H. He, C. Shan, X. Tian, Y. Wei, and G. Huang, "Analysis on influential functions in the weighted software network," *Security and Communication Networks*, vol. 2018, Article ID 1525186, 10 pages, 2018.

[29] 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.

[30] 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.