World Academy of Science, Engineering and Technology
International Journal of Computer and Systems Engineering
Vol:5, No:12, 2011

Applying Complex Network Theory to Software Structure Analysis

Weifeng Pan, *Member, ACM,*

Abstract—Complex networks have been intensively studied across many fields, especially in Internet technology, biological engineering, and nonlinear science. Software is built up out of many interacting components at various levels of granularity, such as functions, classes, and packages, representing another important class of complex networks. It can also be studied using complex network theory. Over the last decade, many papers on the interdisciplinary research between software engineering and complex networks have been published. It provides a different dimension to our understanding of software and also is very useful for the design and development of software systems. This paper will explore how to use the complex network theory to analyze software structure, and briefly review the main advances in corresponding aspects.

Keywords—Metrics, measurement, complex networks, software.

I. Introduction

SOFTWARE has been used in every walk of life, playing increasingly important role. The ever-increasing expansion of applications and users' requirements make a steep rise in the scale and complexity of software, which results in the decrease in the software quality. So it is a great challenge in software engineering to understand, measure, manage, control, and even to low the software complexity [1]. Software structure is one of the factors influencing software quality. So if we want to deeply investigate the software complexity, the information enclosed in the structure should be effectively measured [2].

Over the past few years, complex networks have been intensively studied across many fields of science. Examples include social networks, computer networks, the World Wide Web, etc [3-5]. Software is built up out of many interacting components at various levels of granularity, such as functions, classes, and packages, representing another important class of complex networks. It can also be studied using complex network theory [2, 6-7]. It provides a different dimension to our understanding of software evolutions and also are very useful for the design and development of software systems. Research on studying software from the perspective of complex networks is emerging.

This paper aims to explore how to use the complex network theory to analyze software structure, and briefly review the main research advances in the interdisciplinary research between software engineering (especially software structure analysis) and complex network theory. The rest of the paper is then organized as follows. Sections II brief the motivation of using complex network theory in software structure analysis, with focus on the traditional software structure metrics and their limitations. Section III gives a brief introduction to complex network research. Section IV briefly summarize the main progresses over the last decade in software structure analysis using complex network theory. Last but not least, some concluding opinions are drawn in Section V.

Weifeng Pan is with the School of Computer Science and Information Engineering, Zhejiang Gongshang University, Hangzhou 310018, Zhejiang, P. R. China, e-mail: wfpan@mail.zjgsu.edu.cn.

II. Motivation: Traditional Software Structure Metrics and Their Limitations

Till now, software programming styles have undergone many phases such as procedure-oriented programming (POP), object-oriented programming (OOP) and aspect-oriented programming (AOP). The most important two are POP and OOP while AOP is emerging. Software metrics proposed are usually suited for a specific kind of programming style. So once the programming style changes, the corresponding metrics should also be adapted. In the following subsections we will first review the state-of-the-art in the traditional software metrics research mainly from the POP and OOP perspective, and then we will point out their limitations which are also the motivation (reasons) of why we apply complex network theory in software structure analysis.

And the rationales for the choice to focus on the metrics of POP and OOP is threefold: POP and OOP all have been the widely used development paradigm separately for a long time; There are a lot of software systems developed following the two main programming styles; And the recent work of applying complex network theory in software structure analysis mainly focus on the two domains.

A. Traditional Software Structure Metrics

1) Metrics for Procedure-Oriented Software: POP was very popular in the 1970s, and became the leading programming paradigm. The software metrics at that time mainly proposed to quantify some aspects of procedure-oriented software.

In 1974, R. W. Wolverton proposed Lines of Code (LOC or KLOC for thousands of lines of code) metric to measure programmer productivity and effort [8].

In 1976, T. J. McCabe created the Cyclomatic Complexity, $v(G)$, which is derived from a flow graph and is mathematically computed using graph theory (i.e. it is found by determining the number of decision statements in a program) [9]. $v(G)$ has been applied in many areas such as code development risk analysis, change risk analysis in maintenance, and test planing.

In 1977, M. H. Halstead created Halstead metric by calculating the number of operators and the number of operands in a specific program [10]. And it is also applicable to estimate development efforts and module risk. Halstead metric is very

different from the McCabe metrics. Because the former is based on the mathematical relationships among the number of variables, the complexity of the code and the type of programming language statements, while the latter is determined by the code complexity especially the control paths. Halstead metric received many criticisms for they only measure lexical and/or textual complexity.

In 1978, B. H. Yin and J. W. Winchester proposed two metric groups, primary metrics and secondary metrics, to evaluate the software design [11]. The primary metrics are based on coupling and simplicity and expressed through values extracted from the specification of design. It is reported these metrics can be successfully used in error-prone areas. The secondary metrics try to measure the communication strength between every pair of components. The fan-in and fan-out metrics in them can be used to estimate the worst-case communication complexity of a specific component.

In 1978, C. L. McClure, in [12], defined another complexity metric to quantify the control structures and variables. This metric assigned a specific component with a small invocation complexity if it is invoked unconditionally by only one other component and with a higher complexity if it is invoked conditionally.

In [13] published in 1980, N. Woodfield used a decreasing function to weight the complexity for reviewing a given component and proposed to use the sum of all weights to measure the complexity of the component.

In 1981, S. Henry and D. Kafura proposed a new set of metrics which are based on the measurement of information flow across system components [14]. These metrics are mainly for procedure complexity, module complexity, and module coupling, and are successfully applied to reveal various types of structure flaws in the design and implementation.

In 1984, K. C. Tai proposed a new metric which is based on data flow to quantify the complexity of software [15].

Based on the above mentioned metrics many other new metrics has been created. But a large part of them do not receive many attentions, so here we omit them. For more details, please refer to [1].

2) Metrics for Object-Oriented Software: From 1990s, object oriented design is becoming more popular in software development environment. Compared to structural development, object oriented design is a comparatively new technology and requires a different way of thinking, with many features such as data abstraction, encapsulation, messaging, modularity, polymorphism, and inheritance. So the metrics useful for evaluating procedure-oriented software, may perhaps not fit in with the software developed using OO language.

Till now, a significant number of object oriented metrics have been proposed in literature. For example, metrics proposed by B. F. Abreu and R. Carapuca [16], CK metrics [17], W. Li and S. Henry metrics [18], MOOD metrics [19, 20], M. Lorenz and J. Kidd metrics [21], etc. CK metrics are the most popular (used) among them. Another comprehensive set of metrics is MOOD metrics. This section will focus on the most influential two metric suites, namely CK metrics and MOOD metrics.

CK metrics, first presented by R. Chidamber and F. Kemerer in [17], are the most widely used metrics to evaluate complexities of OO softwares from inheritance (DIT, NOC), coupling between classes (RFC, CBO) and complexity within each class (WMC, $LCOM$). These metrics offer informative insight into whether developers are following OO principles or not in their design. It is reported that using the CK metrics collectively is helpful for design decision making. Many researches have sought to analyze the ability of CK metric suite in estimating the bug proneness of classes [22-25]. Results show that most of the metrics in CK metrics are good indicators to predict bug prone classes.

F. B. Abreu et at. defined MOOD (Metrics for Object Oriented Design) metrics [19, 20]. MOOD refers to a basic structural mechanism of the OO paradigm as encapsulation (MHF, AHF), inheritance (MIF, AIF), polymorphism (POF), and message passing (COF). In MOOD metrics, two main features are used in every metric, i.e., methods and attributes. Methods are used to perform operations so as to obtain or modify the status of objects. Attributes are used to represent the status of each object in the system. Each method or attribute is either visible or hidden from a given class.

B. Limitations

The traditional structural metrics ($i.e.$, procedure- and OO software metrics) mainly focus on the local features of a specific software system, e.g., the number of classes, the number of methods, etc. But they fail to deeply explore the rich information in software topological structure. Due to the lack of suitable tools and theories, people seldom investigate the software structure as a whole, making themselves be in dark about the nature of software.

III. Complex Network

Complex systems and complexity science, in recent years come into the people's attentions, and are viewed as the "21st Century Science" by the founder of Santa Fe Institute, George Cowan. His basic view is that the structure determines the function, emphasizing the view of the system as a whole [26]. With the development of computer technology and its wide application in complex systems studies, people can capture, store and analyze data with an unprecedented scale. The complex network study is emerging (see fig. 1). People used network model to represent and explore many real-world systems, i.e., nodes represent system elements, while edges represent the interaction between them.

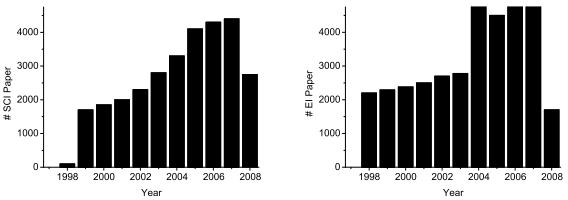

Fig. 1. Statistics of the published SCI (left) and EI (right) papers [27]

It is believed that promoting the study wave of complex networks are two famous papers. In 1998, D. J. Watts and

his advisor S. H. Strogatz in the Cornell University of United States published a paper titled *Collective dynamics of 'small-world' networks* in *Nature*, revealing the small-world feature shared by many real-world networks such as grid and actors network [28]. By saying small-world, it means the network has small average path length and large clustering coefficient. At the same time, they proposed a small-world network model to explain the emergence of this feature. A year later, Professor A. L. Barabási and his Ph.D student R. Albert published another famous paper, *Emergence of scaling in random networks* in *Science*, revealing the scale-free feature shared by many real-world networks [29]. By saying scale-free, it means the degree of nodes obeys the power-law degree distribution. At the same time, they believe "growth" and "preferential attachment" together are the main reasons to the emergence of scale-free feature, and propose BA scale-free network model to explain the emergence of this feature.

The outstanding work of D. J. Watts, S. H. Strogatz, A. L. Barabási and R. Albert's make the complex network research enter a new era, namely the "new science of networks" [30-32], receiving many attentions from a wide range of experts in various fields. The number of papers and monographs on complex networks are emerging. A wide range of real-world networks, from the Internet [33, 34] to WWW [29,35], from Large power networks [28] to the global transportation network [36, 37], from the organism in the brain [38] to a variety of metabolic networks [39], from the scientific collaboration network [40] to a variety of economic [41], etc., and even language [42], numbers [43], music [44], earthquakes [45], which are not networks in the eyes of ordinary people, can also be studied from the perspective of complex networks. It was found that although these networks from different areas, representing different systems, but all have a similar "small world" and "scale-free" features, and other statistical characteristics of the topology have also been revealed. People built a lot of network evolution models to explain the emergence of these features. It can be found that complex network theory has become a powerful tool to analyze the structure as a whole and its dynamical properties of various types of complex systems.

IV. Software Systems as Complex Networks

The research content between complex network theory and software metrics is mainly involved in characterizing the shared topological features of software networks, modeling the growth of software networks, measurement of software networks, and their applications in software practices. In the following subsections, we will give a simple review of the research work in each direction.

A. Characterization of Software Networks

The research work on characterizing software networks focus on discovering and validating the topological features such as small world, scale-free, etc., and exploring the shared features in software structures.

In 2002, S. Valverde et al. are the first to introduce complex network theory into software analysis [46]. They abstract the class diagram as an undirected network, where the nodes denote classes, and the edges denote relationships (inheritance, association, etc.) between every pair of classes (see fig. 2). Then they use such a method to abstract JDK 1.2 and UbiSoft ProRally 2002 and employ complex network theory to analyze the statistic properties of these systems. They found that these two software network all have small-world and scale-free properties. They also make an expectation that these interesting phenomenon may be related to the locally optimization process in software development.

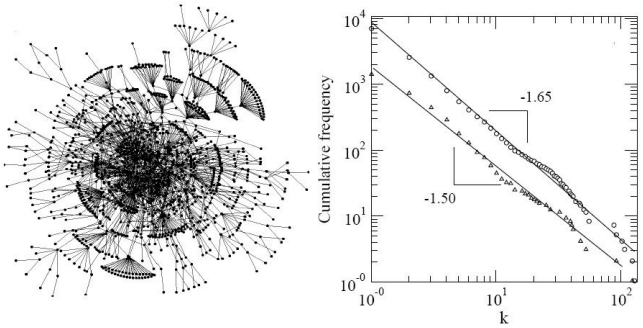

Fig. 2. The software network of JDK 1.2 (left) and its cumulative degree distribution (right) [46]

In software systems, the relationships between software entities (classes, methods, etc.) such as collaboration, invoking, etc. reflecting the control flow. So the directions in software network is meaningful and should be taken into considerations [47]. By this line, in 2003, some researchers use directed networks to represent the topological structures of software systems. C. R. Myers used directed networks to represent software systems, and analyze class collaboration graph of 3 OO software systems, and static procedure class graph of 3 process-oriented software systems. They found that 3 interesting phenomenons: a) Though these software networks from different systems, and even programmed by different languages, they all share small-world and scale-free features; b) The imbalance existing in the exponents of in/out-degree distribution, i.e., the exponent of out-degree distribution is larger than that of in-degree distribution; And c) the negative correlation between in degree and out degree. They expected it may owe to the reuse in software development, i.e., simple classes with small out-degree are easy to be reused, so their in-degrees are larger. Then S. Valverde and R. Soé also use direct network to describe class diagram and they not only found the small-world and scale-free features, but also found these software network shared hierarchical and modularity [48].

Later, people began to analyze software structure from multi-levels of granularity, such as method level, class level, package level, file level, etc. A. P. S. Moura et al. use network model to represent software systems programmed by C and C++ language, i.e., the head files are nodes, and their co-occurrence relationships are edges. They found such a head file network also displays a scale-free feature and has a small-world structure. Meanwhile, they show the scale-free feature due to network growth and small-world feature is the result of performance optimization of the program [49]. N. LaBelle and E. Wallingford show the software network at package level also are of small-world and scale-free type [50]. S.

Valverde and R. V. Solé explore the software network at class and method level with classes and methods are nodes and the reference relationships between them are edges [51]. They also found scale-free and small-world features in such a network. Other people, like D. Li, W. Pan, G. Qi and H. Zhao, also carried out many empirical studies in different software systems, and validate the scale-free and small world features of software networks [52-56].

In recent years, people began to analyze software structural features from the perspective of software evolution [57, 58-60]. They further find many evolution rules such as distance among nodes increase, and diassortative hierarchical structure in addition to scale-free and small-world properties. It provides a new way to study software structure.

B. Modeling of Software Networks Growth

Since software systems share many features, what inner mechanisms make them to be so? The research work on modeling of software networks growth try to elaborate the underline reasons.

C. R. Myers realized the fact that refactoring can improve code evolvability by reorganizing its internal structure, and proposed a simple model for evolving software systems, based on a few refactoring techniques [47]. This simple refactoring model can capture many of the salient features of the observed systems.

S. Valverde et al. assume the complex network features of software systems arise from the conflicting constraints in design [46, 61]. And they further analyzed many software systems programmed by C++, and found that a) the causal relationship between the size of software network and the number of subgraph, and b) software growth is very similar to cellular network. and has duplication and rewiring phenomenon. So, they finally proposed a simple model of software network by duplication and rewiring [62].

K. He et al. explored the growth of software systems from the perspective of design patterns. They categorized software pattern into two groups, frozen spots and hot spots. Frozen spots provide the coordinative dependency relationship between roles of pattern classes, and can not be modified, while hot spots provide the rules for extension and modulation in pattern. And they finally present a software evolution model based on frozen spots of software pattern [63].

B. Li et al. proposed a software evolution model CN-EM by combination of complex network theory and evolutionary algorithm [64]. Experimental results shown CN-EM can well describe the emergence process of complex network properties for many practical software systems.

H. Li et al. introduced two scale-free network models with accelerating growth for undirected and directed software networks [65]. Case studies on two kinds of software network shown its goodness in predicting the emergence of power-law growth and scale-free features.

W. Pan et al. presented a software evolution model with weighted edges to simulate activities in real software development process. Empirical results on eight software systems shown its effectiveness on the description of software evolution and the emergence of their complex network characteristics.

C. Measurement of Software Networks

Recently, some researchers began to propose some metrics to quantify the structural properties of OO software systems using complex network theory. In general, these metrics are useful to evaluate some aspects of software design, and help developers to identify problematic structures in software networks.

R. Vasa et al. studied the change of software structure according to the relationship between the number of nodes and edges, predicting the software size and cost required to construct the software [67].

Y. Ma et al. defined structural entropy according to the degree of nodes to quantify and analyzed the orderly of software structure. At the same time, they explored the relationships between structural entropy, software robustness and the efficiency of the communication, providing the basis for the optimization of software structure [68].

A. Girolamo et al., based on betweenness, proposed a suit of metrics for OO software systems to identify and detect the defects in software structure and problematic classes from different levels of granularity such as class level, network level and design level [69].

In 2006, Y. Ma et al., based on the basic properties of software - cohesion and coupling, integrating complex network Parameters, object-oriented metrics and code-level metrics, proposed a hierarchical system of metrics. And they used such a metric system to analyze the software complexity from the perspective of macro (network), meso (community) and micro (nodes) [70].

S. Jenkins et al. proposed a new metric, I_{cc} to quantify the stability of object-oriented software systems at the class level of granularity [57].

R. Vasa et al. proposed many metrics to quantify the stability of software system in development. They found that the size and complexity of class changed little over time, and classes with a large degree are tend to be modified [71].

Melton et al. studied the dependencies between classes of 81 open source systems, and found that classes can be accessed from other classes are easy to form dependency rings, which make the software complexity increase and stability decrease [72].

Y. Ma et al. used network motif to study the stability of software [73]. They found that sub-graphs with high statistical importance are more stable and hard to form rings. Therefore, they suggested programmers, in the development process, should avoid the generation of rings.

D. Applications in Software Engineering Practices

Over the past decade, although many theoretical results are obtained, however, the potentials for their applications in real-world are not fully explored. Our research group has tried to promote this direction, and has done some primary work on the optimal selection of software structures and software refactoring.

1) Optimal Selection of Software Structures: The same user requirements can be meet by software systems with different inner structures. The quality of the software, however, may

vary. So how to select software with an optimal structure is a problem facing many persons, especially with the popular of open source software development.

In [2], W. Pan et al. tried to solve such a problem with a metric-based approach. They proposed a new metric, $SQOS$, for quantitatively measuring the structural quality of Object-Oriented (OO) softwares. With the aid of $SQOS$, the optimal software structure can be obtained. The metric $SQOS$ can be calculated according to formula (1):

$$SQOS = \frac{\sum_{i=1}^{|N^c|} CI(i) \times BPIC_i}{N^2 - N}, \quad (1)$$

where $|N^c|$ is the number of classes in the OO software systems, $CI(i)$ is the class influence of class i, and $BPIC_i$ is the $BPIC$ (Bug Proneness Index of Classes) of the class i. The authors use formula (2) and (3) respectively to calculate $CI(i)$ and $BPIC_i$.

$$CI(i) = \sum_{j=1, j\neq i}^{|N^c|} M_p^c(i,j)_{updated}, \quad (2)$$

where $M_p^c(i,j)_{updated}$ is the weight of the corresponding edge.

$$BPIC_i = \alpha \frac{WMC_i}{WMC_{sum}} + \beta \frac{CBO_i}{CBO_{sum}} + \gamma \frac{RFC_i}{RFC_{sum}}, \quad (3)$$

where $BPIC_i$ is the $BPIC$ of the class i. WMC_i, CBO_i, and RFC_i are the WMC, CBO, RFC of the class i, respectively. WMC_{sum}, CBO_{sum}, and RFC_{sum} are the sum of WMC, CBO, RFC of all classes, respectively. α, β, and γ are the weights for the corresponding metrics and meet α+β+γ=1.

$SQOS$ is a scalar whose value not smaller than 0. A lower *SQOS* indicates a software structure with a better quality, and bugs can not easily propagate between its classes.

In order to obtain the value of $SQOS$, they first created a new type of software networks at the class level of granularity, weighted class dependency networks (WCDN), in which software components (classes) are nodes and the interaction between every pair of nodes if any is a directed edge which is annotated with a weight corresponding to the probability that a bug in one component (class) will propagate to the other (see fig. 3 as an illustration).

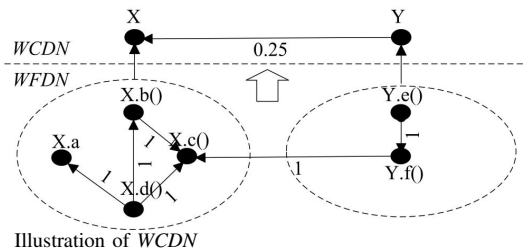

Fig. 3. Illustration of *WCDN*

And then they analyze the bug propagation process in such a WCDN together with the bug proneness of each class, and based on which, $SQOS$ is proposed to measure the structural quality of OO softwares. The empirical results on several case studies validate the effectiveness of the proposed metric. And they also implement a tool to automate the calculation of $SQOS$.

2) Software Class Structure Refactoring: The quality of a software system always degrade over the software evolution. In order to keep its quality, the software should be reconditioned from time to time.

Considering the existing methods are very complex and resource-consuming when doing this task, the authors in [7] present an approach to recondition the class structures of OO software systems from the perspective of complex software networks.

They first use attribute-method networks (see fig. 4) and method-method networks (see fig. 5) to represent attributes, methods and dependencies between them. Then they propose a guided community detection algorithm to obtain the optimized community structures in the method-method networks. Such optimized community structures, in its essence, correspond to the optimized class structures. So by comparison with the original class structures, a list of refactorings can be detected. The authors validate their approach on a famous open-source software, JHotDraw 5.1, and the advantages of our approach are illustrated in comparison with existing methods.

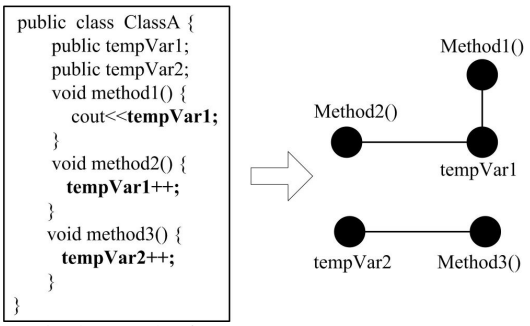

Fig. 4. A simple example of AMN

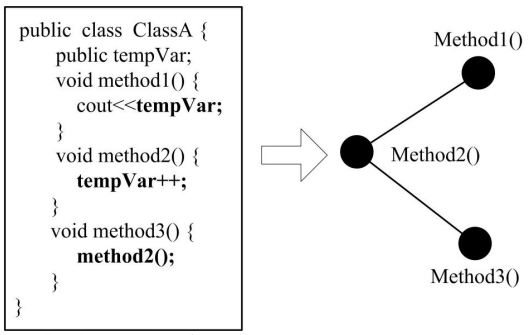

Fig. 5. A simple example of MMN

V. Conclusions

We have talked about the method to use the complex network theory to analyze software structure, and briefly reviewed

the main advances. First, we reviewed the traditional software structure metrics, including metrics for procedure-oriented software and object-oriented software. Then we pointed out the limitations existing in the traditional software metrics, presenting the motivation to use complex network theory in software structure analysis. Then we surveyed the interdisciplinary research work in detail from three dimensions: a) Characterization of software networks; b) Measurement of software networks; and c) Applications in software engineering Practices.

The interdisciplinary research between complex networks and software engineering is an emergent research topic. Exploring the software from the perspective of complex networks and form the software networks, can enable deep understandings with respect to the nature of software. The work presented here summarizes the related work over the last years. However, it is impossible for us to summarize all relative results in this mini-review due to the space limitation. We hope that this brief review can benefit for advancing the work of using complex network theory in solving software engineering problems.

REFERENCES

[1] W. Pan, *Software Networks-Based Analysis of Software Static Structure and Its Applications*. Wuhan: Wuhan University, 2011.
[2] W. Pan, B. Li, Y. Ma, Y. Qin, and X. Zhou, "Measuring Structural Quality of Object-Oriented Softwares via Bug Propagation Analysis on Weighted Software Networks", *Journal of Computer Science and Technology*, vol. 25, no. 6, pp. 1202-1213, 2010.
[3] L. Costa, O. Oliveria, and G. Travieso, "Analyzing and Modeling Real-World Phenomena with Complex Networks: a Survey of Applications", *arXiv:*0711.3199v2, 2008.
[4] J. Lü and G. Chen, "Analysis, Control and Application of Complex Networks: a Brief Overview", *Pro. of the 2009 IEEE International Symposium on Circuits and Systems*, pp. 1601-1604, 2009.
[5] J. Lü and D. Liu, "A Brief Overview of the Complex Biological and Engineering Networks", *Pro. of the 2007 IEEE International Symposium on Circuits and Systems*, pp. 2634-2637, 2007.
[6] W. Pan, B. Li, Y. Ma, and Jing Liu, "Multi-Granularity Evolution Analysis of Software using Complex Network Theory", *Journal of Systems Science and Complexity*, vol. 24, pp. 1-15, 2011.
[7] W. Pan, B. Li, Y. Ma, J. Liu, and Y. Qin, "Class Structure Refactoring of Object-Oriented Softwares using Coommunity Detection in Dependency Networks", *Frontiers of Computer Science in China*, vol. 3, no. 3, pp. 396-404, 2009,
[8] R. W. Wolverton, "The Cost of Developing Large-Scale Software", *IEEE Transactions on Computers*, vol. C-23, no. 6, pp. 615-636.
[9] T. J. McCabe, "A Complexity Measure", *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308-320, 1976.
[10] M. H. Halstead, "Elements of Software Science", *Operating, and Programming Systems*, vol. 7, pp. 128, 1977.
[11] B. H. Yin and J. W. Winchester, "The Establishment and Use of Measures to Evaluate the Quality of Software Designs", *Software quality assurance workshop on Functional and performance*, pp. 45-52, 1978.
[12] C. L. McClure, "A Model for Program Complexity Analysis", *Proc. of the 3rd International Conference on Software Engineering*, pp. 149-157, 1978.
[13] N. Woodfield, "Enhanced Effort Estimation by Extending Basic Programming Models to Include Modularity Factors", West-Lafayette, USA, 1980.
[14] S. Henry and D. Kafura, "Software Structure Metrics Based on Information Flow", *IEEE Transactions on Software Engineering*, pp. 510-518, 1981.
[15] K. C. Tai, "A Program Complexity metric based on Data Flow Information in Control Graphs", *Proc. of the 7th International Conference on Software Engineering*, pp. 239-248, 1984.
[16] B. F. Abreu and R. Carapuca, "Candidate Metrics for Object-Oriented Software within a Taxonomy Framework", *Journal of systems software*, vol. 26, no. 1, pp. 87-96, 1994.
[17] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object-Oriented Design", *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476-493, 1994.
[18] W. Li and S. Henry, "Object-Oriented Metrics that Predict Maintainability", *Journal Of Systems And Software*, vol. 23, no. 2, pp. 111-122, 1993.
[19] F. B. Abreu, "The MOOD Metrics Set", *Proc. of the ECOOP95 Workshop on Metrics*, 1995.
[20] F. B. Abreu, "Design Metrics for OO Software System", *Proc. of the ECOOP95 Quantitative Methods Workshop*, 1995.
[21] M. Lorenz and J. Kidd, *Object-Oriented Software Metrics: a Practical Guide*. NJ: Prentice Hall PTR, 1994.
[22] R. Subramanyan and M. S. Krishnan, "Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects", *IEEE Transactions on Software Engineering*, vol. 29, no. 10, pp. 297-310, 2003.
[23] V. R. Basili, L. C. Briand, and W. L. Melo, "A Validation of Object-Oriented Design Metrics as Quality Indicators", *IEEE Transactions on Software Engineering*, vol. 22, no. 10, pp. 751-761, 1996.
[24] K. EI Emam, S. Benlarbi, N. Goel, "The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics", *IEEE Transactions on Software Engineering*, vol. 27, no. 6, pp. 630-650, 2001.
[25] T. Gyimóthy, R. Ferenc, and I. Siket, "Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction", *IEEE Transactions on Software Engineering*, vol. 31, no. 10, pp. 897-910, 2003.
[26] B. Zhang, "Network and Complex Systems", *Scientific Chinese*, vol. 10, pp. 37-37, 2004.
[27] G. Chen, "Introduction to Complex Networks and Their Recent Advances", *Advances in Mechanics*, vol. 38, no. 6, pp. 653-662, 2008.
[28] D. J. Watts and S. H. Strogatz, "Collective Dynamics of Small World Networks", *Nature*, vol. 393, pp. 440-442, 1998.
[29] A. L. Barabási and R. Albert, "Emergence of Scaling in Random Networks", *Science*, vol. 286, pp. 509-512, 1999.
[30] Committee on Network Science for Future Army Application, *Network Science*, Washington DC: National Academies Press, 2006.
[31] A. L. Barabási, *Linked: the New Science of Networks*, Cambridge MA: Perseus Publishing, 2002.
[32] D. J. Watts, "The "new" Science of Networks", *Annual Review of Sociology*, vol, 30, pp. 243-270, 2004.
[33] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law Relationships of The Internet Topology", *Computer Communication Review*, vol. 29, no. 4, pp. 251- 262, 1999.
[34] G. Siganos, M. Faloutsos, P. Faloutsos, et al., "Power Laws and The AS-Level Internet Topology", *IEEE/ACM Transactions on Networking*, vol. 11, no. 4, pp. 514-524, 2003.
[35] L. A. Adamic and B. A. Huberman, "Power-Law Distribution of The World Wide Web", *Science*, vol. 287, pp. 2115a, 2000.
[36] R. Guimerà, S. Mossa, A. Turtschi, et al., "The Worldwide Air Transportation Network: Anomalous Centrality, Community Structure, and Cities' Global Roles", *Proceedings of the National Academy of Science USA*, vol. 102, pp. 3394-7799, 2005.
[37] W. Li and X. Cai, "Statistical Analysis of Airport Network of China", *Physical Review E*, vol. 68, no. 4: 46106, 2004, .
[38] O. Sporns, "Network Analysis, Complexity, and Brain Function", *Complexity*, vol. 8, no. 1, pp. 56-60, 2002.
[39] H. Jeong, B. Tombor, R. Albert, et al., "The Large-Scale Organizationn of Metabolic Networks", *Nature*, vol. 407, pp. 651-654, 2000.
[40] M. E. J. Newman, "Scientific Collaboration Networks. I. Network Construction and Fundamental Results", *Physical Review E*, vol. 64, no. 1, pp. 16131, 2001.
[41] M. A. Serrano and M. Boguñá, "Topology of The World Trade Web", *Physical Review E*, vol. 68, pp. 015101, 2003.
[42] A. E. Motter, A. P. S. Moura, Y. C. Lai, et al., "Topology of The Conceptual Network of Language", *Physical Review E*, vol. 65, pp. 065102, 2002.
[43] G. Corso, "Families and Clustering in a Natural Numbers Network", *Physical Review E*, vol. 60, no. 3, pp. 036106-036110, 2004.
[44] X. Liu, C. K. Tse, and M. Small, "Complex Network Structure of Musical Compositions: Algorithmic Generation of Appealing Music", *Physica A*, vol. 389, no. 1, pp. 126-132, 2010.
[45] S. Abe and N. Suzuki, "Scale-Free Network of Earthquakes", *Europhysics Letters*, vol. 65, no. 4, pp. 581-586, 2004.
[46] S. Valverde, R. F. i Cancho, and R. Sole, "Scale Free Networks from Optimal Design", Europhysics Letters, vo. 60, pp. 512-517, 2002.
[47] C. R. Myers, "Software Systems as Complex Networks: Structure, Function, and Evolvability of Software Collaboration Graphs, *Physical Review E*, vol. 68, 2003.

[48] S. Valverde and R. Solé, "Hierarchical Small Worlds in Software Architecture", Working Paper of Santa Fe Institue, SFI/03-07-44, 2003.
[49] A. P. S. 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, pp. 017102, 2003.
[50] N. LaBelle and E. Wallingford, "Inter-Package Dependency Networks in Open-Source Software", *ArXiv*: Cs.SE/0411096, 2004.
[51] S. Valverde and R. V. Sole, "Universal Properties of Bipartite Software Graphs", *Proc. of the 9th IEEE International Conference on Engineering of Complex Computer Systems*, 2004.
[52] M. Han, D. Li, C. Liu, et al. "Networked Characteristics in Software and its Contribution to Software Quality", *Computer Engineering and Application*, vol. 42, no. 20, pp. 29-31, 2006. (in Chinese)
[53] J. Liu, K. He, Y. Ma, et al., "Scale Free in Software Metrics", *Proc. of the 30th Annual International Computer Software and Application Conference*, pp. 229-235, 2004.
[54] J. Liu, K. He, R. Peng, et al., "A Study on the Weight and Topology Correlation of Object-Oriented Software Coupling Network", *Proc. of the 1st International Conference on Complex Systems and Applications*, PP. 955-959, 2006.
[55] D. Yan, G. QI, "The Scale-free Feature and Evolving Model of Large-Scale Software systems", *Acta Phisica Sinica*, vol. 55, no. 8, pp. 3799-3084, 2006.
[56] H. Zhang, H. Zhao, W. Cai, et al., "Using the k-core Decomposition to Analyze the Static Structure of Large-Scale Software Systems", *The Journal of Supercomputing*, vol. 53, no. 2, pp. 352-369, 2010.
[57] S. Jenkins and S. R. Kirk, "Software Architecture Graphs as Complex Networks: a Novel Parttion Scheme to Measure Stability and Evolution", *Information Sciences*, vol. 177, no. 12, pp. 2587-2601, 2007.
[58] M. Shi, X. Li, and X. Wang, "Evolving Topology of Java Networks", *Proc. of the 6th World Congress on Control and Automation*, PP. 21-23, 2006.
[59] L. Wang, Z. Wang, C. Yang, et al., "Linux Kernels as Complex Networks: a Novel Method to Study Evolution", *Proc. of the 25th International Conference on Software Maintenance*, PP. 41-50, 2009.
[60] H. Li, B. Huang, and J. Lu, "Dynamical Evolution Analysis of the Object-Oriented Software Systems", *Proc. of the 2008 IEEE World Congress on Computational Intelligence*, PP. 3035-3040, 2008.
[61] R. V. Sole and S. Valverde, "Information Theory of Complex Networks: on Evolution and Architectural Constraints", *Proc. of the International Conference on Complex Networks*, pp. 189-207, 2004.
[62] S. Valverde and R. V. Sole, "Network Motifs in Computational Graphs: a Case Study in Software Architecture", *Physical Review E*, vol. 72, pp. 026107, 2005.
[63] K. He, R. Peng, J. Liu, et al., "Network Motifs in Computational Graphs: a Case Study in Software Architecture", *Journal of Systems Science and Complexity*, vol. 19, no. 2, pp. 157-181, 2006.
[64] B. Li, H. Wang, Z. Li, et al., "Software Complexity Metrics Based on Complex Networks", *Acta Electronica Sinica*, vol. 34, no. 12A, pp. 2371-2375, 2006. (in Chinese).
[65] H. Li, "Scale-Free Network Models with Accelerating Growth", *Frontier of Computer Science in China*, vol. 3, no. 3, pp. 373-380, 2009.
[66] W. Pan, B. Li, Y. Ma, and J. Liu, "A Novel Software Evolution Model Based on Software Networks", *Complex (2)*, 1281-1291, 2009.
[67] R. Vasa, J. G. Schneider, C. Woodward, et al., Detecting Structural Changes in Object Oriented Software Systems, *Proc. of the International Symposium on Empirical Software Engineering*, PP. 479-486, 2005.
[68] Y. Ma, K. He, and D. Du, "A Qualitative Method for Measuring the Structural Complexity of Software Systems based on Complex Networks", *Proc. of the 12th Asia-Pacific Software Engineering Conference*, PP. 257-263, 2005.
[69] A. Girolamo, L. I. Newman, and R. Rao, "The Structure and Behavior of Class Networks in Object-Oriented Software Design", www.eecs.umich.edu/ leenewm/ documents/classnetworks.pdf, 2005.
[70] Y. Ma, K. He, D. Du, et al., "A Complexity Metrics Set for Large-Scale Object-Oriented Software Systems", *Proc. of the 6th International Conference on Computer and Information Technology*, PP. 257-263, 2006.
[71] R. Vasa, J. G. Schneider, and O. Nierstrasz, "The Inevitable Stability of Software Change", *Proc. of the 23nd IEEE International Conference on Software Maintenance, Paris France*, PP. 4-13, 2007.
[72] H. Melton and E. Tempero, "Static Members and Cycles in Java Software", *Proc. of the 1st International Symposium on Empirical Software Engineering and Measurement*, PP. 136-145, 2007.
[73] Y. Ma, K. He, and J. Liu, "Network Motifs in Object-Oriented Software Systems", *Dynamics of Continuous, Discrete and Impulsive System (Series B: Applications and Algorithms) Special Issue on Software Engineering and Complex Networks*, vol. 16, no. S6, pp. 166-172, 2007.