Dynamical Evolution Analysis of the Object-Oriented Software Systems

Huan Li, Beibei Huang, and Jinhu Lü

Abstract— **Software evolution and update play a vital role in software engineering. It has many advantages, such as improving the efficiency of programming, reducing the cost of maintenance and promoting the development of software systems. This paper further analyzes the evolution and update processes of three typical kinds of real-world object-oriented software systems by using the tools of complex networks. It discovers some underlying dynamical evolution characteristics and rules of the object- oriented software systems. These results are very useful for the design and development of the object-oriented software systems.**

I. Introduction

It is well known that software engineering is a systematic and disciplined approach to developing software. In detail, it applies computer science, engineering principles and practices to the generation, operation, and maintenance of software systems. There are many key processes in software engineering. In particular, the software evolution and update play a vital role in software engineering. In fact, most development effort and expenditure is allocated to the evolution and update of those existent versions. Software is ceaselessly changed - maintained, evolved and update - more often than it is written, and changing software is extremely costly. Therefore, exploring the nature of the software evolution is a challenging problem for the software engineers. Fortunately, there are some substantial advances in this extensive field over the last four decades [1]-[21]. Most of the existing works focus on how to analyze the certain aspects of the development history of software engineering. Normally, the software systems become more and more complex as the evolution and update of the software systems. Such increasing complexity confronts much more challenges in system robustness and adaptability, which are predeterminative qualifications in order to ensure their long-term safety and reduce the cost of maintenance. It seems that it is very difficult to control the complexity of the whole system and discover the relationship between the structure complexity and software evolution by using traditional theories, methods and tools of software engineering.

Many real-world software systems are rather complex and can be regarded as complex networks [6]-[11], in which components are nodes and the relationships between two nodes are links. Moreover, we find that most software networks have the typical small-world property and scale-free character [12]-[16]. The so-called small-world property refers to a high degree of clustering as in the regular networks and a small average distance compared with the whole network size [17]. The scale-free character reveals the large inhomogeneity of the network structure, where the node degree distribution is very wide and usually obeys the power law distribution [18]. It means that most nodes have very few connections but a few nodes have many connections. Although these nodes are rare, they significantly dominate almost all network properties, which clearly different from those of the random networks.

Most of these existing works concentrate on investigating various network characteristics or analyzing the topology structures of the different kinds of software systems by using complex network theory. However, the underlying reason that why these different software networks have similar network characteristic is not clear up to now. The software evolution is a typical continuous process which requires the collective efforts of all users and software engineers. Thus, the dynamic change and update is inevitable. For example, object-oriented (OO) software becomes increasingly developed by means of evolutionary-development processes, such as inheritance, polymorphism and overloading. Without question, there exist various advantages in the study of evolution laws of real-world software systems by using complex network theory. This paper further investigates the evolution and update processes of three typical kinds of real-world object-oriented software systems by using the tools of complex networks. It reveals some interesting dynamical evolution characteristics and rules of the object-oriented software systems rather than those existing research.

The left part of this paper is then organized as follows. Some preliminary knowledge is introduced in Section II. In Section III, we apply the complex network theory to further study three kinds of real-world OO software networks with a serial of versions in detail. Conclusions are finally given in Section IV.

II. PRELIMINARY KNOWLEDGE

In this section, we introduce some preliminary knowledge of the complex networks [12]-[16]. Then we use the complex

H. Li and B. Huang are with the State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, P.R. China (email: {lihuan2501, hbb21st}@163.com).

J. Lü is with the Institute of Systems Science, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100080, P.R. China, and also with the State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, P.R. China (email: jhlu@iss.ac.cn).

This work was supported by the National Natural Science Foundation of China under Grants No.60772158 and No.60221301, the Scientific Research Startup Special Foundation on Excellent PhD Thesis and Presidential Award of Chinese Academy of Sciences, the Important Direction Project of Knowledgement Innovation Program of Chinese Academy of Sciences under Grant No.KJCX3-SYW-S01, and the National Key Basic Research and Development 973 Program of China under Grant No.2007CB310805.

978-1-4244-1823-7/08/$25.00

networks theory to characterize the general OO software systems.

As a kind of artificial system, OO software represents a human interpretation and solution of a problem. An OO software system consists of class that is an abstract set of objects, and the relationships between two classes are inheritance, association, dependency, etc. We model an OO software system as a software network, which is a directed graph denoted as $G = (N, L)$, where N is the set of all classes of the specific OO software system, and L is the set of all relationships between two classes. Here, $(i, j) \in L$ means that class i either inherits from, or is associated with, or depends on class j . Fig. 1 shows a simple example of the object-oriented software network.

```
public class Shape {}

public class Circle extends Shape{}

public class Square extends Shape{}

public class Paint
{
 public Circle myCircle;
 public Square mySquare;
}
```

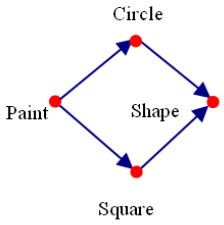

Fig. 1. Illustration of a simple OO software network

III. DYNAMICAL EVOLUTION ANALYSIS OF THE OO SOFTWARE SYSTEMS

In this section, we apply the complex network theory to further analyze the evolution structural quality of the consecutive complex software networks.

A. Statistical parameters of the software networks

As the most prevalent software, Java software runs on more types of embedded devices, PCs and servers than any other software. Unlike C++, which is an OO version of c and has held some feature of old non-object-oriented language, java is an efficient, fast, secure and reliable OO language. In order to obtain comprehensive conclusion of software network evolution, we have collected three series of open source software systems that have 41 different versions written in Java. JEdit is a mature programmer's text editor with hundreds of person-years of development behind it. Apache Tomcat is the servlet container that is used in the official Reference Implementation for the Java Servlet and JavaServer Pages technologies. Azureus implements the BitTorrent protocol using java language and comes bundled with many invaluable features for both beginners and advanced users. Figs. 2-11 show the dynamical evolution characteristics of these consecutive software networks, where

N= number of nodes,
L = number of links,
d = average shortest length,
D = diameter,
k = average degree of nodes,
K_{in}= maximum in-degree;
K_{out}=maximum out-degree;
In = power-law exponent of in-degree distribution,
Out = power-law exponent of out-degree distribution,
C = clustering coefficient.

B. Growth of Node and Link

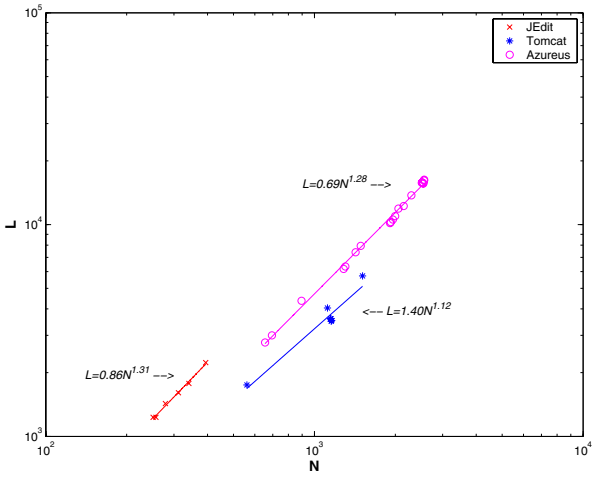

Fig. 2. Log-log plot of the number of nodes vs the number of links

It is an inevitable and cyclical course that the software is evolved. Environmental change, new organizational requirement and discovery of defeat have impelled the software to be revised and upgraded. This transformation continues promoting the environment to change, also produce new demand and defeat. The designers spend a large amount of time on revising and improving on the existing software version in order to release the new one. Some new classes and links have been added into old versions or those existing components have been improved. Fig. 2 shows that the upgrading course of the software version is the process that class and link increases. It implies that the number of class and the number of link is the simplest metric of scale, function and complexity of software system. In the three sets of software systems that we have selected, the minimum number of nodes is 251 and the maximum number of nodes is 5726. Fig. 2 shows that these software networks obey following empirical relation in the log-log plot as follows:

$$\log_{10} L = a \log_{10} N + b.$$

Then we can get

$$L = 10^b N^a.$$

That is

$$L = \alpha N^{\beta} \tag{1}$$

where α and β are constants.

The reason the number of links must be smaller than the complete graph with $N(N-1)$ links must cause $1 < \beta < 2$. We can obtain the following differential equation:

$$\frac{dL}{dN} = \alpha\beta N^{\beta-1} \tag{2}$$

According to Fig. 2, β is often close to one. The empirical equation (2) shows that every time programmer implement a new class, an approximate equal quantity of links $\alpha\beta N^{\beta-1}$ will be also added into system. Software system will keep the growth of the node and links in harmony.

C. Distance among Nodes

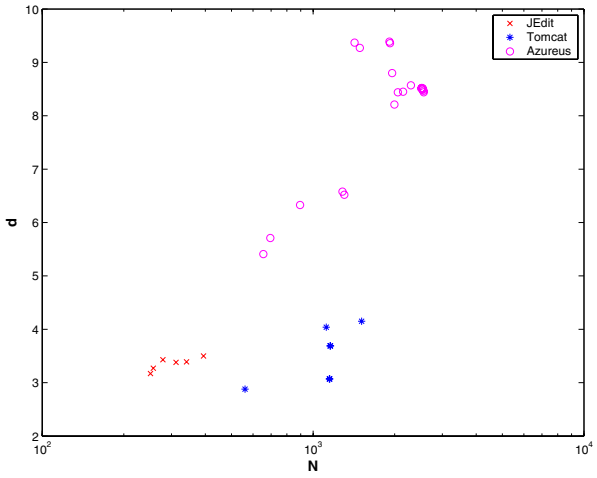

Fig. 3. Semi-log plot of the number of nodes vs average shortest length

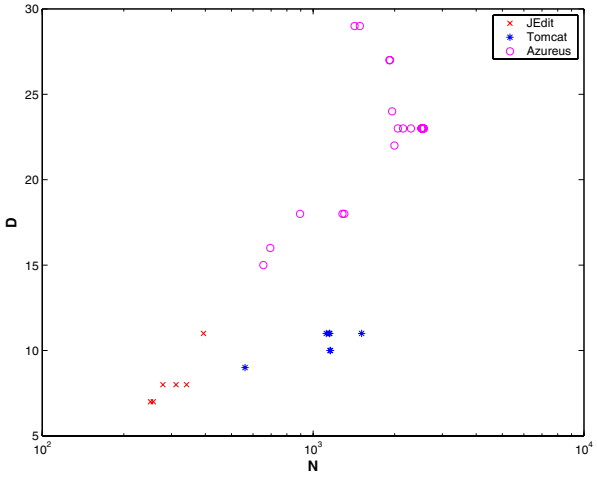

Fig. 4. Semi-log plot of the number of nodes vs diameter

In graph theory, the distance between node i and node j is the minimum number of links along the shortest path connecting them. Traversing all paths, you will get the average shortest length and the diameter that is the maximum number of all distance. The Designer should not only process various classes but also control the relation among them in phase. The relation between classes mainly has the following three ways, where object, as instance of class, have contributed two ways.

(1) An object read or writes data stored in another object;
(2) An object calls a method in another object;
(3) A class inherits from another class.

The situation that distance between classes is very long will increase the complexity of the procedure, hide potential hazard and incur awful readability. These disadvantages are unfavorable to the programmers to code, modify and debug programs and at the same time efficiency of transmitting message can reduce due to overlong distance, so that the designers will do their best to reduce connection of long distance. For example, if depth of inheritance is excessive, they will carry out a bran-new one in order to escape from embarrassment. In Figs. 3 and 4, we find the scales of the software networks are increasing constantly, but the average shortest length d and diameter D of three networks increase slowly, even negative growth has appeared in Azureus. Although different kinds of network have different range of value of diameter D, these values restrain connective way of additive nodes. These logarithmic figures describe that the growth rate of the average shortest length d and diameter D are roughly identical, which have similar linear relation with logarithm of network scale N. If the number of N is up, acceleration of logarithm of N will become down. In spite of having a great deal of nodes, software network strongly depart from the Poissonian graph and demonstrate some characteristics of the small world structure.

D. Degree and Scale Free Property

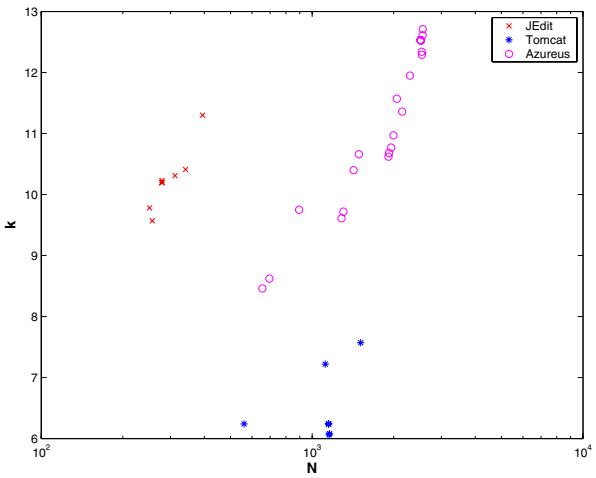

Fig. 5. Semi-log plot of the number of nodes vs average degree of nodes

In graph theory degree represent not only simple but also important attribute of node. If the number of the degree of

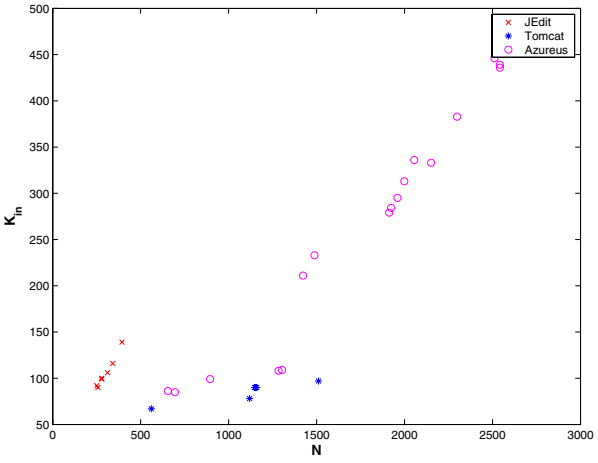

Fig. 6. Plot of the number of nodes vs maximum in-degree of nodes

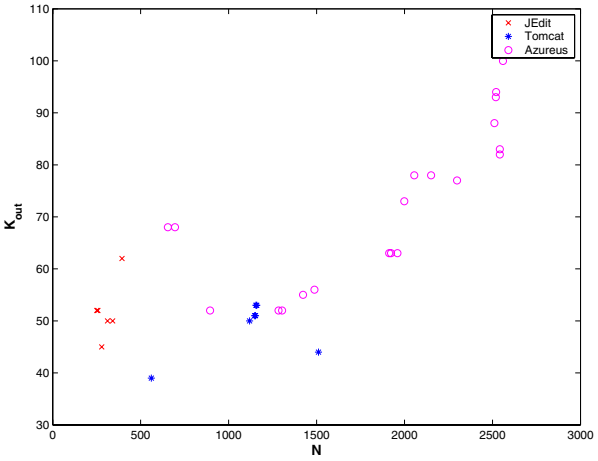

Fig. 7. Plot of the number of nodes vs maximum out-degree of nodes

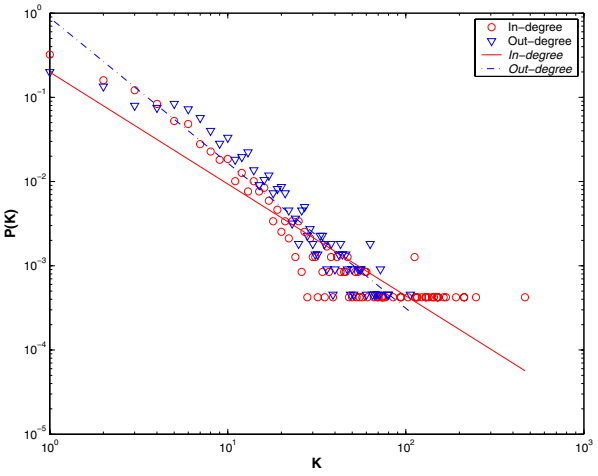

Fig. 8. Log-log plot of the in-degree and out-degree of nodes vs its probability

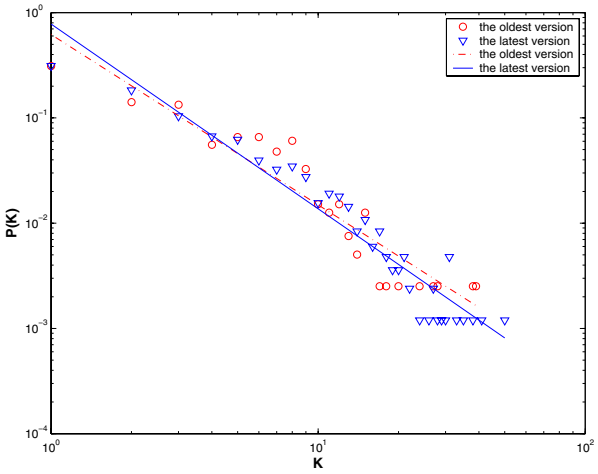

Fig. 9. Log-log plot of the out-degree of nodes vs the probability of the oldest and the latest version of Tomcat

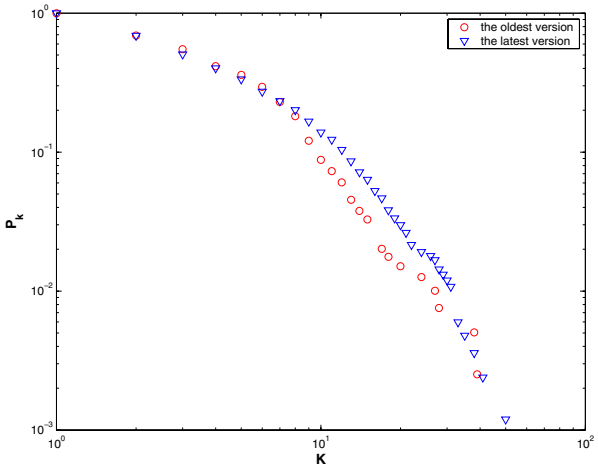

Fig. 10. Log-log plot of the cumulative out-degree distribution of the oldest and the latest version of Tomcat

node is large, it shows that one is important in a certain aspect. The correlation between WMC that shows complicated characteristic of class or LCOM that shows cohesion among classes and the out-degree of node is obvious [19]-[21]. The relationship between the average degree and the logarithm of network scale exist a certain one linear to correlated with as shown in fig. 5, which demonstrate the growth characteristic of small world network similar to the average path length and diameter. Figs. 6 and 7 show that the maximum number of in-degree and the maximum number of out-degree will follow the increase of the scale of network, but sometimes the maximum number of out-degree appear negative growth.

In many complex networks the distribution of nodes can be described by a power law form. The scale-free character reveals the large inhomogeneity in the network structure, where the distribution of degree of a node is very wide, is

usually described by a power law form, i.e.

$$P(k) \propto k^{-\gamma} \tag{3}$$

where the scaling exponent γ is usually constrained to a range $2 \leq \gamma \leq 3$. Although there are greater differences on function and scale in three kinds of software systems, we find that in the evolving course of software systems the all of maximum numbers of in-degree are greater than the maximum number of out-degree, all of these networks are scale free network, the scaling exponents of degree distribute function keep similar numbers and the scaling exponents of in-degree are lower than that ones of out-degree. Here we figure two curves of logarithm of distribution of in-degree and out-degree that belong to latest edition of Azureus. As the scaling exponents of in-degree are lower than that ones of out-degree, Fig. 8 shows that the distribution shape of in-degree drop more slowly than that of out-degree, the probability of in-degree of nodes is lower than that of out-degree of nodes when nodes have less degree, the probability of in-degree of nodes is higher than that of out-degree of nodes when nodes have much more degree and there are much more dispersed points of in-degree of nodes at their tails of curves than those of out-degree of nodes. The situation that the curve of in-degree of nodes has much more point at its tails agrees with another observation that that maximum number of in-degree of nodes is higher than the ones of out-degree of nodes. As designers can produce class by many reused means, such as inheritance, polymorphism and overloading, class has high inclination that can be used by others. Specially, based classes often have much more in-degree and less out-degree.

Scale free networks have reflected evolution rule of the competition and selection in the nature. Obviously, if the distribution of degree is for strict power-law and the power-law exponent does not change, and even if the scale of network changes, the slope of curve of cumulative degree distribution

$$P_k = \sum_{k'=k}^{\infty} P(k') \tag{4}$$

would remain stable. Figs. 9 and 10 show the curves of distribution of power-law and of cumulative out-degree distribute of two Tomcat versions, which one is the oldest version and another is the latest version that we have collected. Through these curves we may observe the variation tendency of probability of distribution of degree in the course of software evolution. Because the power-law exponents of both networks are approximate equal, tendencies of decline of probability appear basically similar as out-degree of nodes increase. Although some dispersed point appear in tail of curve of the latest version, their number of probability are very low in fig. 9. This case reveals that probability is high as degree is low in the oldest network but this high probability in low degree drop on a very small quantity in the latest network. Fig. 10 shows that as the scale of network in increase, probability of cumulative degree in low degree low will keep in a steady proportion and only a very small amount of nodes will evolve the ones that have large number of degree. We conclude that rule of scale free network always dominate the course of software evolution and topology of software network evolve steadily according to rules of preferential attachment.

E. Small World Property

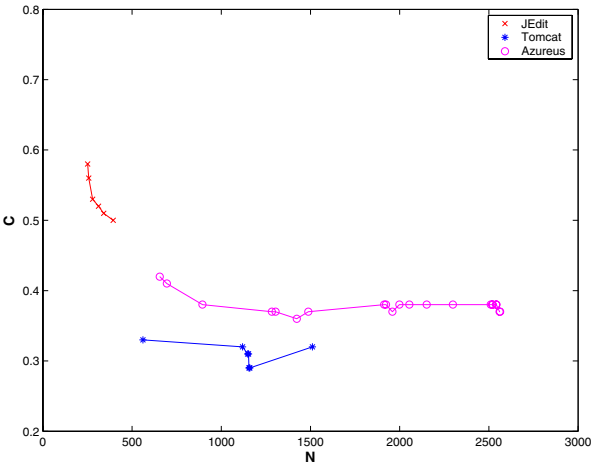

Fig. 11. Plot of the clustering coefficient vs the number of nodes

Although random graphs keep the average path length low, its clustering feature remain low, the property of which lead that it could not reproduce important characteristics of the real world network. Recently, there have been many discoveries on research of complex theory that lots of real world network, such as Internet, social network and software network, display characteristics of small world, which have short average length and high clustering feature. The clustering coefficient of one node

$$C_i = 2E_i/k_i(k_i - 1) \tag{5}$$

presents local feature in the network, where k_i is the degree of node i and E_i is the quantity of the links between neighbors of node i. The mean clustering coefficient of nodes reflect clustering feather of the whole network.

On above section, we have mentioned that all software networks that we have selected have short average length and small average degree that increase slowly. Fig. 11 shows that their clustering coefficient range from 0.29 to 0.58 that exceed random graphs far away. High clustering feature indicate that a great number of interconnected nodes exists in local area, which just correspond to the characteristics of modularization of software system. As modularization is one of the most important OO software design principles, the software network will inevitable demonstrate the high clustering characteristic. Fig. 11 also shows that their clustering coefficient declines as the scale of network increase. Specially, clustering coefficient of JEdit decline fast, but the coefficient of its latest version remains high. Although different software networks have different ranges of clustering

coefficient, all of them show clustering coefficient of late node is close to the one of old nodes and designers try their best to maintain the hierarchical structure topologically.

IV. Conclusions

In this paper, we have further investigated the evolution and update processes of the three typical kinds of real-world object-oriented software systems with several continuous versions by using the tools of complex networks. It reveals some underlying dynamical evolution characteristics and rules of the object- oriented software systems. These results are very useful for the design and development of the object-oriented software systems, such as improving the efficiency of programming, reducing the cost of maintenance and promoting the development of software systems. The traditional system design principle consists of modularization, minimum coupling and maximum cohesion. However, the modern software design principle has been appended characteristics of competition, preferential attachment and clustering.

References

[1] Z. C. Xing and E. Stroulia, "Analyzing the evolutionary history of the logical design of object-oriented software," *IEEE Trans. Software Eng.*, vol. 31, pp. 850-869, Oct. 2005.

[2] M. M. Lehman and J. F. Ramil, "An approach to a theory of software evolution," *Proc. 4th Int. Workshop Principles of Software Evolution*, Vienna, Austria, pp. 70-74, 2001.

[3] M. M. Lehman and L. A. Belady, *Program Evolution-Processes of Software Change*. London: Academic Press, 1985.

[4] S. G. Eick, T. L. Graves, A. F. Karr, J. S. Marron, and A. Mockus, "Does code decay? Assessing the evidence from change management data," *IEEE Trans. Software Eng.*, vol. 27, pp. 1-12, Jan. 2001.

[5] E. J. Barry, C. F. Kemerer, and S. A. Slaughter, "On the uniformity of software evolution patterns," *Proc. 25th Int'l Conf. Software Eng.*, pp. 106-113, 2003.

[6] R. Albert and A.-L. Barabsi, "Statistical mechanics of complex networks," *Rev. Mod. Phys.*, vol. 74, no. 1, pp. 47-97, 2002.

[7] R. V. Sol, R. Ferrer-Cancho, J. M. Montoya, and S. Valverde, "Selection, tinkering and emergence in complex networks," *Complexity*, vol. 8, pp. 20-33, 2002.

[8] J. Zhou, J. Lu, and J. Lü, "Adaptive synchronization of an uncertain complex dynamical network," *IEEE Trans. Auto. Contr.*, vol. 51, no. 4, pp. 652-656, 2006.

[9] J. Lü and G. Chen, "A time-varying complex dynamical network model and its controlled synchronization criteria," *IEEE Trans. Auto. Contr.*, vol. 50, no. 6, pp. 841-846, 2005.

[10] J. Lü, X. Yu, G. Chen, and D. Cheng, "Characterizing the synchronizability of small-world dynamical networks," *IEEE Trans. Circuits Syst. I*, vol. 51, no. 4, pp. 787-796, 2004.

[11] J. Zhou, J. Lu, and J. Lü, "Pinning adaptive synchronization of a general complex dynamical network," *Automatica*, 2008, in press.

[12] C. R. Myers, "Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs," *Phy. Rev. E*, vol. 68, no. 4, pp. 046116, 2003.

[13] S. Valverde and R. V. Sol, "Hierarchical small worlds in software architecture," Arxiv preprint cond-mat/0307278, 2003.

[14] A. Potanin, J. Noble, M. Frean, and R. Biddle, "Scale-free geometry in OO programs," *Commun. ACM*, vol. 48, no. 5, pp. 99-103, 2005.

[15] A. MacCormack, J. Rusnak, and C. Y. Baldwin, "Exploring the structure of complex software designs: An empirical study of open source and proprietary code," *Management Sci.*, vol. 52, pp. 1015-1030, 2006.

[16] K. He, R. Peng, J. Liu, F. He, P. Liang, and B. Li, "Design methodology of networked software evolution growth based on software patterns," *J. Syst. Sci. & Complexity*, vol. 19, no. 2, pp. 157-181, 2006.

[17] D. J. Watts and S. H. Strogatz, "Collective dynamics of small-world," *Nature*, vol. 393, pp. 440-442, 1998.

[18] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, pp. 509-512, 1999.

[19] S. R. Chidamber, C. F. Kemerer, and C. Mit, "A metrics suite for object oriented design," *IEEE Trans. Software Eng.*, vol. 20, no. 6, pp. 476-493, 1994.

[20] G. Chastek and R. Ferguson, "Toward Measures for Software Architectures," *Tech. Note CMU/SEI-2006-TN-013*, 2006.

[21] J. Liu, K. He, R. Peng, and Y. Ma, "A Study on the Weight and Topology Corrlation of Object Oriented Software Coupling Network," *Dyna. Continu. Discret. Impul. Syst. Ser. B*, vol. 13, no. Suppl. S, pp. 955-959, Dec. 2006.