International Journal of Research in Engineering and Science (IJRES)
ISSN (Online): 2320-9364, ISSN (Print): 2320-9356
www.ijres.org Volume 1 Issue 8 ǁ Dec 2013 ǁ PP.20-27

Research on Stability of Software Structure Based On Design Patterns

Zheng Liu[1], Qian Zhang[2], Hai Zhao[1]

[1](*College of Information Science and Engineering, Northeastern University, China)*
[2](*Technology Strategy & Development Department, Neusoft Corporation, China)*

ABSTRACT: *In view of metric for stability of software structure, a metric method based on design pattern is presented, which makes use of the influence of design patterns in OO software network as typical microstructure. Firstly, the application of design patterns is studied in software network, and then the stability of microstructure extracted from design patterns is analyzed by Lyapunov method. The concept of Coverage of Design Patterns is presented according to the application of design pattern in software system, and is used to metric for the stability of the whole software structure.*

Keywords - *stability of software structure, design patterns, software network, OO software*

I. Introduction

From the point of view of software design, internal structure of software will bring great influence on the system reliability. Generally, the more complex the software structure is, the worse the structure stability is, and more structural defects are implied in the system which can affect the system reliability. As to structural stability, Elish believes that it is a sign of the ability that maintain their original structural characteristics during the evolution, the greater the stability is, the stronger the ability maintaining original characteristics when some arguments such as scale or structure change[1]. Recently, researchers find that the software network presents typical characteristics of complex network if many reusable elements such as class, object, module, and subroutine are regarded as the nodes and the relationship between these elements are regarded as arcs in network [2-8].

Any complex system is composed of many basic tectonic units according to certain rules, and the stability of these basic tectonic units can affect the stability of the whole system directly. Software design patterns describe common problems appearing during development of OO software and effective solution for these problems. Application of design patterns in software design can improve the openness, compatibility, stability and scalability of software system and make development and maintaining more easily. After researching on a large number of OO software, it is found that design patterns can be found anywhere in each software system, which are used repeatedly in order to resolve some common problems, and they compose a complete system. From the structure of design patterns, each pattern is a microstructure has certain function composed of several classes or interfaces and their mutual relationship. If extracting from these microstructures, corresponding connected subgraphs will be got. Analyzing on these small connected subgraphs can help to understand the global formation mechanism of network structure because they are basic elements to compose the whole network. These small connected subgraph are usually used to study the structure design and evolution of network in the area of complex network. Now this method is used to research the variation of software structure based on design patterns in the area of software engineering that provide new perspective on the study of characteristics of whole software system.

II. Design patterns in software network

2.1 Establish software network

The process of establishing software network is: starting from source code, extracting the structural of system firstly, then abstracting to network model. As to OO software, class is the basic constituent elements and class-level software network is that the nodes are abstracted from classes and the arcs are abstracted from relationship between classes. The specific operation is shown as Fig.1.

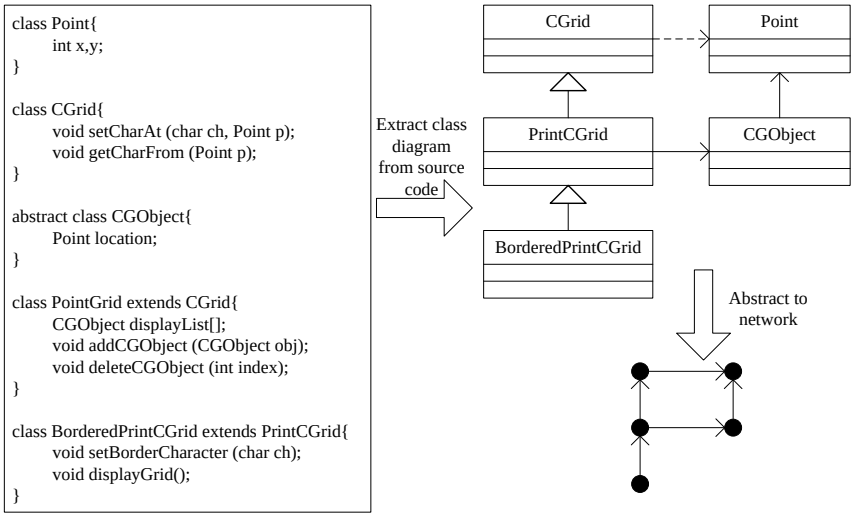

Fig.1 Extraction of software network

This network model is described as: $N=(V, A, T, W)$. Among them, $V(Vertices)=\{v_i \mid i = 1, 2, \ldots, n\}$ represents the set of n nodes in network, and each node represents a class in software system. $A(Arcs)=\{(v_i, v_j, t) \mid v_i \in V, v_j \in V, t \in T, i \neq j\}$ represents the set of arcs in network and each arc represents the relationship between two classes. There is usually not single relationship between two classes in real software system, for example, there is dependency and also generalization between class A and class B, so t is a composite value. $T(Type)=\{t \mid t \in \{G, U, D\}\}$ represents the relationship between two classes. Here G means generalization, D means Dependency and U means usage including association and aggregation. $W(Weight)=\{w(a) \mid w(a)=f(t, \delta, N), a \in A, t \in T, \delta \in (0,1), N \in \{0,1,2\ldots\ldots\}\}$ represents the weight of each arc.

2.2 extracting the network structure of design pattern

There are 23 design patterns for OO software. In order to study the influence on software structure brought by design patterns, the structural network of design patterns are extracted according to the definition of design pattern and its structural description. The specific extracting method is as follows:

(1) describing the design pattern structure by UML according to its definition, and then analyzing the relationship between classes, interfaces and objects.
(2) Abstracting nodes in software network from classes, interfaces and objects in UML class diagram.
(3) Abstracting arcs in software network from the relationship between two elements in UML class diagram, and determined the type value of the arc according to the relationship type.

Different from the extraction of the whole software network, the description of type of arc in design pattern network should be more meticulous. Description the type of arc as generalization, dependency and usage is not enough, and the types of usage and dependency need be differentiated further. According to research needs for design pattern, usage type is described as three types: general usage $U(g)$, static usage $U(s)$ and template usage $U(t)$, and dependency type is described also as three types: parameter dependency $D(p)$, returns dependency $D(r)$ and friend dependency $D(f)$.

Taking Flyweight pattern as example, its structure is shown as Fig.2, and its network model is shown as Fig.3.

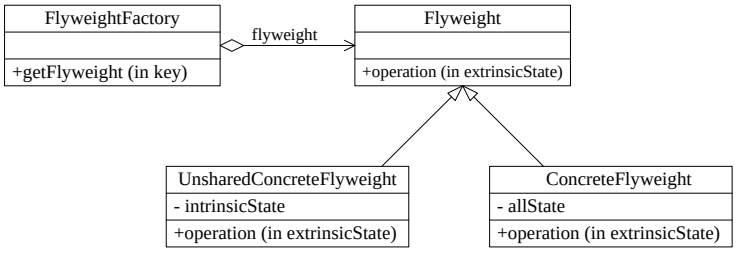

Fig.2 Frame of Flyweight

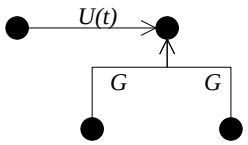

Fig.3 Software network of Flyweight Pattern

2.3 Match design pattern in software network

Among all the 23 design patterns, Singleton pattern, Memento pattern and Template pattern have simple structure due to there's only one or two nodes in their structural networks, so the affection on structure stability is limit. There is only generalization arc in Façade pattern's structural network, which is confused with general generalization in software network during matching and reduces the match accuracy. So there are only 19 patterns will be matched in this paper.

The essence of matching design patterns is searching the exact network structure in software network. If local network structure has the same characteristics with some design pattern, the design pattern is found. In order to match all the design patterns accurately and quickly, the network structure of each pattern should be described mathematically firstly, and then search for a feature arc of pattern structure in software network, compare characteristics of the terminal node and the arc connecting the terminal node and another one in design pattern network with those in software network one by one until find all the nodes and arcs in pattern network.

According to description of mathematical model of design pattern structure, the nodes are represented by *v*, *w*, and the arc between two nodes is represented by <*v*, *w*>. *v* is initial node and *w* is terminal node. Taking the Flyweight pattern as example, the description of its mathematics model is as follows:

There are two class nodes *v* and *w* in software network and the type of arc between *v* and *w* is *U*(*t*), and there is a node set *cw* that satisfied the following conditions:

(1) All the nodes in *cw* are class node.
(2) There is arc between nodes in *cw* and *w* whose type is generalization.
(3) The number of nodes in *cw* is 2.

The structure composed with node *v, w*, node set cw and arc <*v, w*>, <*cw, w*> is called Flyweight pattern.

According to the mathematics description above, the critical points in matching algorithm are:

(1) The type of arc between *v* and *w* is just template usage.
(2) Penetration of *w* whose type is generalization is 2.

So, matching algorithm for Flyweight pattern is as following:

(1) Select any arc <*v, w*> in software network;
(2) Determine if *v, w* are class nodes, if not, turn to (8);
(3) Determine if the type of arc <*v, w*> is *U(t)* and it is single type, if not, turn to (8);
(4) Determine if the penetration of w whose type is generalization is 2, and the penetration of *v* is 0, if not ,turn to (8);
(5) Save *v, w*;
(6) Search all the child nodes of *w* and save;
(7) Update the number of Flyweight pattern;

(8) Determine if have traversed all the arcs in software network, if not ,turn to (1);

(9) End.

III. Analysis on structural stability of design pattern

3.1 Stability of microstructure

Recently, researchers find that the frequency of occurrence of different relationship between nodes is non-randomness, and the frequency is different in different networks along with the deepening of the research on complex network theory. Milo et al called the typical interactive structure appears in real network more frequently than in random network as motif, and believed the motif is the basic unit to compose the whole network and achieves main function of system. Researching on these basic units are benefit for understanding the overall characteristics of the whole network and grasping the formation mechanism of macroscopic characteristics by investigating the way that composing macroscopic topology of software network using motif. According to the research on design patterns in OO software, the pattern structure is just the motif of software network by ignoring the specific context in each pattern.

According to perspective of system science, the structural characteristics and dynamic behavior of a local network topology are determined by connection mode including of the direction or weight of which between nodes. So, the differential equation representing the corresponding dynamic system of a local network structure is as follows:

$$x_i = f_i(x_1, x_2, \cdots, x_n) \quad (i = 1,2,\cdots,n) \tag{1}$$

Among them, x_i represents the status of node i, and f_i represents how much all the other nodes connecting with node i influence on it [9]. Eigenvalue of Jacobian $J = \{a_{ij}\} = \left\{\frac{\partial f_i}{\partial x_i}\right\}$ should be calculated in order to determine the stability of microstructure by Lyapunov method according to literature [10]. a_{ij} represents how much node j influence node i , and its value is the weight of directed arc form node i to node j in software network. Since nodes have no influence on themselves in software network, the value on diagonal of matrix is less than 0. The value is set -1 except in special circumstances so as to research further.

Prill et al find that the robustness of typical microstructure appears when the network structure changes slightly highly correlates with frequency of this microstructure's occurrence in networks in the study on biological networks. At the same time, structural stability score (SSS) is presented to metric on the stability of microstructure [11]. Generally, the value of SSS is related with the number and size of feedback loops in the structure. The larger the number and size are, the smaller the value is, which leads to the stability declining. Complexity and McCabe loop in program have similar relationship in software design. When the number and scale turn to larger, the program will be more complex [12]. So loops should be avoided in order to reduce complexity and improve stability in software design [13].

3.2 structural stability of design pattern

The structural stability of 19 design patterns are analyzed according to metrics discussed above. Jacobians of all the 19 design patterns are listed as follows:

Table 1 Jacobians of design patterns composed of three nodes

Pattern Name	Jacobians	Pattern Name	Jacobians
Builder pattern	$\begin{bmatrix} -1 & 0 & w_{13} \\ w_{21} & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$	Chain of Responsibility pattern	$\begin{bmatrix} -1 & 0 & 0 \\ w_{21} & -1 & 0 \\ w_{31} & 0 & -1 \end{bmatrix}$
Adapter pattern	$\begin{bmatrix} -1 & w_{12} & w_{13} \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$	Interpreter pattern	$\begin{bmatrix} -1 & 0 & 0 \\ w_{21} & -1 & 0 \\ w_{31} & 0 & -1 \end{bmatrix}$
Composite pattern	$\begin{bmatrix} -1 & 0 & 0 \\ w_{21} & -1 & 0 \\ w_{31} & 0 & -1 \end{bmatrix}$	State pattern	$\begin{bmatrix} -1 & 0 & 0 \\ w_{21} & -1 & 0 \\ w_{31} & 0 & -1 \end{bmatrix}$
Proxy pattern	$\begin{bmatrix} -1 & 0 & 0 \\ w_{21} & -1 & 0 \\ w_{31} & w_{32} & -1 \end{bmatrix}$		

Table 2 Jacobians of design patterns composed of four nodes

Pattern name	Jacobians	Pattern name	Jacobians
Factory Method pattern	$\begin{bmatrix} -1 & 0 & 0 & 0 \\ w_{21} & -1 & 0 & 0 \\ 0 & w_{32} & -1 & w_{34} \\ 0 & 0 & 0 & -1 \end{bmatrix}$	Command pattern	$\begin{bmatrix} -1 & 0 & 0 & w_{14} \\ 0 & -1 & 0 & 0 \\ 0 & w_{32} & -1 & w_{34} \\ 0 & 0 & 0 & -1 \end{bmatrix}$
Prototype pattern	$\begin{bmatrix} -1 & 0 & 0 & w_{14} \\ 0 & -1 & 0 & w_{24} \\ 0 & 0 & -1 & w_{34} \\ 0 & 0 & 0 & -1 \end{bmatrix}$	Observer pattern	$\begin{bmatrix} -1 & 0 & 0 & w_{14} \\ w_{21} & -1 & 0 & 0 \\ 0 & w_{32} & -1 & w_{34} \\ 0 & 0 & 0 & -1 \end{bmatrix}$
Flyweight pattern	$\begin{bmatrix} -1 & 0 & 0 & w_{14} \\ 0 & -1 & 0 & w_{24} \\ 0 & 0 & -1 & w_{34} \\ 0 & 0 & 0 & -1 \end{bmatrix}$		

Table 3 Jacobian matrixes of design patterns composed of five nodes

Pattern name	Jacobians	Pattern name	Jacobians
Bridge pattern	$\begin{bmatrix} -1 & 0 & 0 & 0 & w_{15} \\ w_{21} & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & w_{35} \\ 0 & 0 & 0 & -1 & w_{45} \\ 0 & 0 & 0 & 0 & -1 \end{bmatrix}$	Mediator pattern	$\begin{bmatrix} -1 & 0 & 0 & 0 & 0 \\ w_{21} & -1 & w_{23} & w_{24} & 0 \\ 0 & 0 & -1 & 0 & w_{35} \\ 0 & 0 & 0 & -1 & w_{45} \\ w_{51} & 0 & 0 & 0 & -1 \end{bmatrix}$
Decorator pattern	$\begin{bmatrix} -1 & 0 & 0 & 0 & 0 \\ w_{21} & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & w_{35} \\ 0 & 0 & 0 & -1 & w_{45} \\ w_{51} & 0 & 0 & 0 & -1 \end{bmatrix}$	Strategy pattern	$\begin{bmatrix} -1 & 0 & 0 & 0 & w_{15} \\ 0 & -1 & 0 & 0 & w_{25} \\ 0 & 0 & -1 & 0 & w_{35} \\ 0 & 0 & 0 & -1 & w_{45} \\ 0 & 0 & 0 & 0 & -1 \end{bmatrix}$

Itarator pattern	$\begin{bmatrix} -1 & 0 & 0 & 0 & 0 \\ w_{21} & -1 & w_{23} & 0 & 0 \\ 0 & w_{32} & -1 & w_{34} & 0 \\ 0 & 0 & 0 & -1 & 0 \\ w_{51} & 0 & 0 & w_{54} & -1 \end{bmatrix}$

Table 4 Jacobian matrixes of design patterns composed of more than five nodes

Pattern name	Jacobians
Abstract Factory pattern	$\begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ w_{21} & -1 & 0 & 0 & w_{25} & 0 & 0 & w_{28} & 0 & 0 \\ w_{31} & 0 & -1 & w_{34} & 0 & 0 & w_{37} & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & w_{46} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & w_{56} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & w_{79} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & w_{89} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ w_{10,1} & 0 & 0 & 0 & 0 & w_{10,6} & 0 & 0 & w_{10,9} & -1 \end{bmatrix}$
Visitor pattern	$\begin{bmatrix} -1 & w_{12} & 0 & 0 & 0 & 0 & 0 & w_{18} \\ 0 & -1 & 0 & 0 & w_{25} & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & w_{35} & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & w_{45} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & w_{68} \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & w_{78} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}$

Eigenvalue of each the Jacobian above is calculated and it is find that all value are -1, that means structural stability of these design patterns has nothing to do with the connection between nodes in network. According to the definition of SSS, all SSS values of microstructures described by Jacobians are 1. So, all design patters have stable structure.

IV. Metrics on stability of software structure

Because the structures of all design patterns are stable, applying design patterns in software system to realize system's function can not only improve efficiency of development due to reusability of these patterns but also help the system structure maintain its structural stability. The more patterns are applied in a OO software, the higher the structural stability of the system. Definition of coverage of design patterns is presented to metric the proportion of nodes belong to design patterns in the software network as follows:

Definition 1 *Coverage of Design Patterns*: the proportion of nodes belongs to some design pattern among all the nodes in software network is called Coverage of Design Patterns of this software system, which is marked as *C*, and

$$C = \frac{\sum_{i=1}^{19} n_i * f_i}{N}, \quad n_i \in \{3,4,5,8,10\}. \tag{2}$$

Among this, N is the number of nodes in software network, f_i is the number of some design pattern applied, and n_i is the number of nodes in this pattern.

In this paper, 50 open source software are selected as study sample to analyze Coverage of Design Patterns of them, and the statistics result is shown as Fig.4. It can be find that among these software, Octave has the highest value of *C* that is 0.764, and Subversion has the lowest value of *C* that is 0.012. The average value of C is 0.294 and the standard deviation is 0.18.

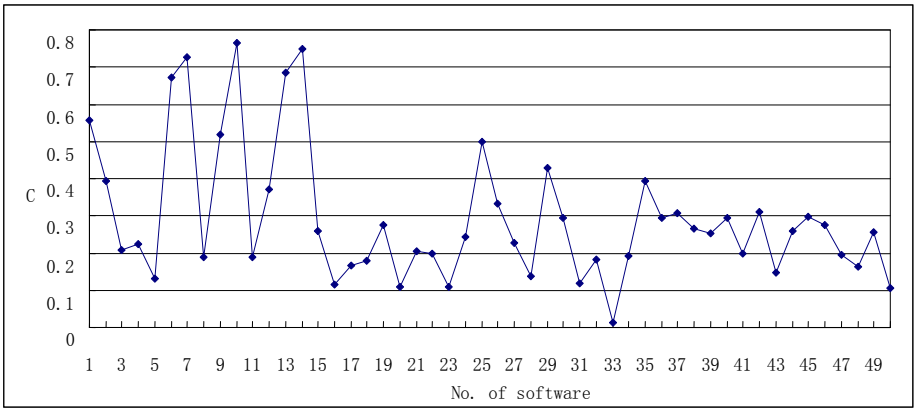

Fig.4 Stat. of Coverage of design patterns

The relationship between *C* and the scale of software is described in Fig.5. The abscissa is the number of nodes in software network that shown in logarithmic form. It is find that software whose *C* value is larger always have fewer nodes, that means microstructure of design patterns have greater influence on the whole network in the software network with fewer nodes. For example, the number of nodes in a software network whose *C* value is larger than 0.5 is always within 500. Leading role that design patterns play on the structure is declining along with the software's scale increasing, and the value of C will keep on a stable level. According to statistical data, it can be believed that if the value of C is larger than 0.4, design patterns have great influence on the software structure and the structure of this software is stable due to the stable structure of design pattern. If the value of C is less than 0.2, the stability of software structure should be metric by other means due to design patterns have little influence on the software system.

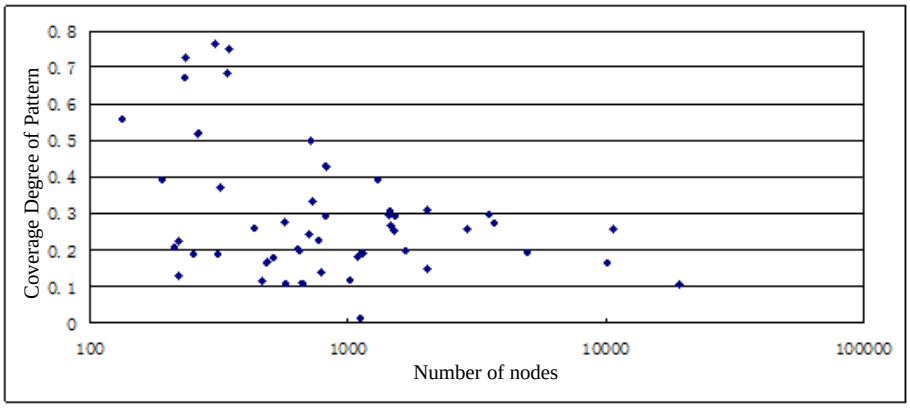

Fig.5 Relationship between the coverage of design pattern and software scale

V. Conclusion

Different kinds of networks have different typical connection mode. The local network compose of these connections is considered as the basic constituent element of the whole network, the structural characteristics of which reflect the overall characteristics of network. Design patterns are basic elements constituting OO software system. The macrostructure stability of OO software system is analyzed according to

the study on structural characteristics of design patterns and patterns' application in the system. It is finds that design patterns have positive impact on stability of software structure described by Coverage of Design Patterns defined in this paper based on analysis on stability of design patterns according to Lyapunov method. It is a new research tool that metric stability of software system by that of microstructure of design patterns according to stable structure of design patterns has positive impact on the stability of whole system structure, but the quantitative relationship between these two kinds of structures needs to be studied further. As to the software in which design patterns are applied infrequently, it is needed other supplementary tools to metric the structural stability. That is what we'll do in further research.

VI. Acknowledgements

The research work was supported by Fundamental Research Foundation of the Ministry of Education of China under Grant No. N110304003

References

[1] Elish M O and Rine D. Indicators of structural stabilitu of object-oriented designs: a case study. *Proc. 29th Annual IEEE/NASA Software Engineering Workshop*, Marriott Greenbelt, MD, 2005, 183-192.

[2] Valverde S, Cancho R F, SoléR. Scale Free Networks from Optimal Design, *Europhysics Letters*, 60(4), 2002, 512-517.

[3] Myers C R. Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs , *Physics Review E, 68*, 2003, 1-15.

[4] Wheeldon R and Counsell S. Power law distributions in class relationships, *Proc. 3rd IEEE International Workshop on Source Code Analysis and Manipulation, Amsterdam*, 2003, 45-54.

[5] Baxter G, Frean M, Noble J, et al. Understanding the shape of Java software, *ACM SIGPLAN Notices, 41(10),* 2006, 397-412.

[6] Potanin A, Noble J, Frean M, et al. Scale-free geometry in OO programs, *Communications of the ACM, 48(5)*, 2005, 99-103.

[7] Moura A P S, Lai Y-C and Motter A E. Signatures of small-world and scale-free properties in large computer programs, *Physical Review E, 68*, 2003, 17-22.

[8] Liu B, Li D Y, Liu J, et al. Classifying Class and Finding Community in UML Metamodel Network. In Proceedings of ADMA 2005, Springer-Verlag LNAI 3584: 690-695.

[9] Keqing He, Yutao Ma, Jing Liu, et al. *Software Network* (Beijing: Science Press, 2008).

[10] Khalil H K. *Nonlinear Systems* (Englewood Cliffs, NJ: Prentice Hall, 2002).

[11] Prill R J, Iglesias P A and Levchenko A. Dynamic properties of network motifs contribute to biological network organization. *PLos Biology*, 3(11), 2005, 1881-1892.

[12] Parnas D L. Designing software for ease of extension and contraction. *Proc. 3rd International Conference on Software Engineering, Atlanta*, USA, 1976, 264-277.

[13] Melton H and Tempero E. An empirical study of cycles among classes in Java. *Technical Report UoASE-2006-1*, Department of Computer Science at University of Auckland, 2006.