Evolutionary Analysis of Software Structure Based on Software Networks

Xinlong Fu
Wulanchabu Vocational College, Wulanchabu, Inner Mongolia, China

ABSTRACT: Evolution is one of the essential characteristics of software system. Understanding their evolutions can lead to better software engineering practices. The existing work usually concerns on the simple statistics such as number of versions, software scale and the number of modules, neglecting the analysis of the evolutionary characteristics of software systems as a whole. This paper introduces the theory and methodology in complex networks to analyze the evolution of object-oriented (OO) software from a whole perspective. It uses the software class network to represent the OO software, and analyzes the evolutionary laws of OO software from the number of nodes and edges, average degree, degree distribution and small-world phenomenon. Empirical results on real-world OO software systems uncover some underlying evolution characteristics which give a better understanding of the complexity of the evolution of object-oriented software systems and are also important inspirations to software evolution modeling.

Keywords: software network; software evolution; software structure; object-oriented software

1 INTRODUCTION

Software evolution refers to the process of software maintenance and updates [1]. The evolution of software system during its life cycle always inevitably undergoes continuous evolution. The change of system requirements, enhanced functions, defect correction and replacement of any operating environment can cause the software evolution [1]. Software system must have the ability to continuously evolve, otherwise it will fail prematurely.

The evolutionary analysis of software dedicated to finding software evolution rules and modalities. It is an important research field of software engineering. It is helpful for software maintenance activities. It extends the software life, reduces the cost of the software, and also provides an important reference for software evolution process modeling [2]. There are already a lot of software evolution analysis works, but most of them concentrated on simple statistical metrics such as the number of software versions, development time, system size, and number of modules [3]. There is a lack of software metrics that can treat the software as a whole to quantify its structure features. Due to the lack of appropriate tools, researchers rarely analyze software evolution from the perspective of software as a whole [4].

Complex networks can be used to describe the skeleton of various systems, providing an overall perspective to analyze the system topology and its dynamics as a whole. It has become an extremely important and challenging scientific frontier. In recent years, researchers apply complex network theory into the field of software engineering.

Software network model abstracts method, class, packages in a specific piece of software as nodes, and the relationships between them such as method calls, inheritance, association and other relations as the edges of the network. A large number of open-source software systems are studied. Fruitful results have been reported. Potanin et al. [5-6] found that the software network has Small World and Scale-free characteristics. The literature [7] proposed an evolution model based on the replication and bifurcation rules. He Keqing [8] put forward the model of software evolution based on the design patterns in the frozen section. Pan Weifeng proposed an evolution model of evolutionary algorithm based on software networks. Pan Weifeng [4] defines several metrics to measure the structure of the software quality. Pan Weifeng [10]
proposed software reconstruction method based on the community detection technique in complex networks.

In the view of this, this paper presents an EASN (Evolutionary Analysis of Software Structure based on Software Networks) method for the analysis of the evolution of software architecture based on software network. EASN abstracts classes and the relationship between them as nodes and edges respectively. It introduces the theory in complex networks such as the scale of networks, the average degree, degree distribution, and Small World theory to analyze the characteristics of the overall structure of the software. At the part of empirical research, we take 4 real Java software systems as the research subjects, analyze their software networks, and demonstrate the process of the application of the principle of EASN method. In the current paper, we reveal many rules of the evolution of these systems. These rules are of great significance for guiding the development and maintenance of software.

This paper is organized as follows. In Section 2, we give the related work which is closely related to this paper, and the basic framework of the method proposed in this paper. In Section 3, we use the method proposed to analyze the software networks constructed from 4 open-source software systems. In Section 5, we conclude the paper.

2 RELATED WORK

Lehman and Belady use the number of modules to describe the size of a specific version of software, define some measures to characterize the difference of continuous software versions, and finally reveal the some basic rules of software evolution such as sustained change, continuously grow and increasing complexity, which are known as Lehman’s laws of software evolution [11-12]. Chong Hok Yuen studied the evolution of Lehman software and found that the Lehman’s laws are not shared by all systems [13-14]. Tamai and Torimitsu used a questionnaire to explore a piece of commercial software in its five years and found that the life of small software will be relatively short [15]. Cook and Roesch tracked the evolution data of a real-time telephone exchange software of a telephone company for 18 months and found similar Lehman’s conclusions [16]. Mockus studied the evolution of Apache server and the Mozilla browser, and found these systems are controlled by the core developers. These core developers decided the development cycle and software architecture [17]. German presents a software (software trails) trajectory based software evolution analysis method [18]. He used information from a mailing list, version control system, documentation, source code, etc. to analyze software evolution. This method can provide detailed historical information of software project evolution. Antoniades put forward a framework for simulating the evolution of open source software systems. The experimental results show that the results obtained by this framework are broadly consistent with [19]. Capiluppi used some basic software project properties (size, number of modules, developers, etc.) to analyze the evolution of 19 open-source projects and to build their evolutionary model [20]. Godfrey used code statistical methods to study the evolution of the Linux kernel within six consecutive years, and found that Linux at the system level was super-linear growth [21].

Although the above studies have achieved fruitful results, most of them concentrated on the number of software versions, the system development/use time, the system size and the number of modules. There is a lack of research work that dedicated to analyze software systems as a whole.

3 EASN METHOD

Figure 1 shows the basic framework of the EASN method. First, EASN takes OO (object-oriented) software as the research subject and analyze the source code grammatically and lexically. And various relationships between classes are extracted and is transformed to a software network. Then EASN introduce the some related principles and methods in the complex networks to analyze the evolution of software systems.

![Figure 1. The framework of EASN.](image)

The following subsections will explain some of the content of the framework, and give the definition of related concepts.

3.1 Software class network

OO software systems usually consist of multiple interacting classes (or interfaces). It achieves the desired functionality through the interaction and collaboration between these classes. In order to use the method of complex network, we firstly abstract object-oriented software system as a software network. Classes in the system are regarded as nodes and the relationship between classes are represented as the edges. Thus the software can be abstracted as a form of a network topology. EASN uses the software class network to abstract the topology structure of software systems.

**Definition 1.** Software Class Network SOCAN

The SOCAN of an object-oriented software system...
can be represented by an undirected graph, that is, \( \text{SOCAN} = (N, E) \). Here, \( N \) is the set of nodes and represents a collection of all classes (or interfaces) in the software; \( E \) is a collection of undirected edges that represent a variety of relationships between classes.

EASN mainly considers several dependencies between a few classes, that is, if any one of the following relationship exists between the two classes, then there is an edge between the corresponding class nodes.

1. Class \( A \) uses key words \textit{extends} to inherit from another class \( B \).
2. Class \( A \) uses key words \textit{implements} to implement the interface \( B \).
3. Class \( B \) has a property of type \( A \).
4. The method of class \( A \) calls the method of an object of class \( B \).

Figure 2 shows a Java code fragment and its corresponding SOCAN.

4 EXPERIMENT DESIGN

4.1 Subjects

To illustrate the specific implementation process of the proposed method, we selected 4 software systems, i.e., JFreeChart, DrJava, Open Swing and Tomcat. JFreeChart is an open diagramming library. It can generate pie charts, histograms, scatter plots, timing diagrams, and other graphics. It is a pretty good Java graphics solution. We select consecutive versions of JFreeChart. DrJava is a lightweight development environments used to write java program. We select 53 consecutive versions of DrJava. Open Swing is an application framework based on Java software development. It provides a complete set of solutions for quick and easy to develop client desktop application. We selected Open Swing consecutive 140 versions. Tomcat is a lightweight web application server. It developed by Apache, Sun and other companies and individuals. We select a total of 39 consecutive versions of Tomcat-6 and Tomcat-7.

Above software systems are open source systems using Java language. The source code of Tomcat can be downloaded from the official website of the Apache (http://tomcat.apache.org/). The remaining 3 software source code can be downloaded from SourceForge (http://sourceforge.net/). As an example, Figure 3 shows the construction of the SOCAN from the JFreeChart.

4.2 Experiment analysis

We use the theory and method of complex network to analyze the structure evolution of above-mentioned 4 software systems. The following sections will be discussed

4.2.1 Node and edge

Nodes and edges are the fundamental unit of the network, which are often regarded as the scale of the network. By researching on the evolutionary laws of software class network from the number of nodes and edges, we can understand the changes in software size. We will analyze the above-mentioned 4 software systems’ corresponding software network from number of nodes and edges with the version of co-evolution, to explore the evolution of software size.

Figure 4 shows the cooperative evolution of the number of nodes and edges. Here, the ordinate is the number of edges of a particular version of the software class network, and the abscissa is the number of nodes in a specific version of the software.

Figure 4 shows that the growth of the number of edges and the number of nodes in the software net-
work are very close to linear relationship, that is, the number of edges increases linearly with the number of nodes. The linear fit of DrJava and Tomcat software is better, but the intercept of them varies. The linear fitting of FreeChart and open swing software is relatively poor. The growth of the two systems is without obvious rules to follow. There are cases that the node number is equal but the edge number is greatly different. It shows that it is very difficult to predict the size of the software by modeling. First, it is difficult to predict the node size according to the release time, such as 34th and 90th versions of Open Swing. Its release time are November 9, 2007 and 2008 December 29. They differ by nearly 14 months, but they are 905 nodes; and for 84th and 85th version, release time differs only seven days, while the number of nodes is 1391 and 825, respectively. The difference is very large. Secondly, it is difficult to predict the number of sides according to the number of nodes. As already mentioned above, the 22th (0.9.11) and 31th (0.9.21) version of JfreeChart system, the number of nodes is 635, while the number of sides, respectively 3562 and 2805. The difference is very large.

4.2.2 Average degree
The degree \( d(n) \) of node \( n \) is defined as the number of other nodes connected to the node. It is an important feature of node. Intuitively, the greater the degree of a node means that the node is more important in a sense. Based on the degree of a node, we can define the average degree of the network as

\[
<k> = \frac{\sum_{n \in N} d(n)}{|N|},
\]

where \( N \) is a specific set of nodes in a network.

This section will analyze the evolution rules in the average degree with the issued serial number RSN (release sequence number) of the software network constructed from 4 software systems, and the results shown in Figure 5.

It can be seen from Figure 5 that the minimum average of 4 software is not less than 6 and the maximum average of them is not more than 13. Fluctuation range of DrJava and Tomcat average of two software node is generally between 1 to 2 degrees. The average degree of volatile system is JFreeChart and Open Swing, which are close to 5 and 3 respectively.

Figure 5 shows that the average degree of a node in the examined 4 software systems does not change unanimously. The average degree of Tomcat has a gradually decreasing trend, while there is a gradual upward trend of DrJava. JfreeChart fluctuated greatly. The average degree of open swing, after the 85th version, changed little. Further the average degree of later versions is less than that of the previous ones.

The average degree reflects the density of the network connection. As can be seen from the evolution of the average degree of the software class network, most of the software remained its internal complexity in the process of evolution, while the degree of internal complexity of some of the software has undergone great changes. Meanwhile, a different system is not consistent with the law in terms of the average degree of evolution.

4.2.3 Degree distribution
Degree distribution reflects the probability that a randomly selected node’s degree is \( k \) in the network. It can be described by a function \( P(K = k) \sim k^{-\alpha} \). Degree distribution is commonly used to detect whether a network is scale-free. If the degree distribution of the network follows a power-law distribution, then the network is a scale-free network [6]. The power law is
usually detected by the cumulative degree distribution. Network is scale-free, indicating the degree of the nodes in the network have no obvious characteristic length. Most of the nodes have only a few connections while a few nodes have a large number of connections to be the hubs of the network.

To explore the changes of degree distribution with the evolution of software systems, each software system selects 3 representative versions in the latest, oldest and intermediate stages. And their distributions will be compared.

Figure 6 shows the cumulative distribution of the three selected versions of the 4 software systems. Here, the dots (black) represent the first selected version. It is the earliest version; triangle (blue) indicates the second selected version which is in the middle of the version’s stage; Cross (red) indicates the selected third version which is the latest version. As we can see from Figure 6, in addition to some of the smaller version of the software (the earliest version of JFreeChart, only have 90 nodes), the degree distribution curves for each different version in the double logarithms are close to be linear, and the distribution curve of the latest version is more close to a straight line. This shows that the scale-free structure of OO software is formed during the continuous evolution, and the software with the structure of scale-free maintained a scale-free feature in the evolutionary process. This characteristic is becoming more and more obvious. In addition, the degree distribution curve of different versions is separated at the tail, indicating the change of the nodes with bigger degree is moderate in the process of evolution. The principle of encouraging reuse in software engineering practice may be the cause of the scale free of the degree distribution, which enables software developers to encapsulate the basic functions which may be universally reused in the later part of the software. In the case where there is a need for similar function modules, we can choose to reuse the already-programmed ones.

The clustering coefficient \( C_i \) of the node \( i \) reflects the possibility that its neighbors become neighbors. If \( l_i \) represents the actual number of edges between node \( i \) and its adjacent nodes, \( k_i \) represents the total number of adjacent nodes of node \( i \), and then \( C_i \) is calculated through \( C_i = \frac{2l_i}{k_i(k_i-1)} \). While the clustering coefficient of the whole network is the average of all the nodes clustering coefficient, i.e., \( C = \frac{1}{N} \sum C_i \).

Many of the complex network systems in the real-world not only have a relatively small average shortest path length, but also have a high clustering which is more obvious than the corresponding random network. They are known as small world networks.

We calculate the average shortest path length and average clustering coefficient for all versions of the software network, and compare the average shortest path length and the average clustering coefficient to the \( D_{\text{rand}} \) and \( C_{\text{rand}} \) of the random network with the same scale. The average shortest path and clustering coefficient of random networks are calculated using \( D_{\text{rand}} = \ln(N)/\ln(<k>) \) and \( C_{\text{rand}} = <C>/N \).

Because the version number is too large, we only take the latest version of the 4 software systems as examples to illustrate. The results are shown in Table 1.

### Table 1. statistical data of the latest version of the software.

<table>
<thead>
<tr>
<th>Software</th>
<th>JFreeChart</th>
<th>DrJava</th>
<th>Open Swing</th>
<th>Tomcat</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N )</td>
<td>596</td>
<td>2368</td>
<td>983</td>
<td>1711</td>
</tr>
<tr>
<td>( E )</td>
<td>2957</td>
<td>10427</td>
<td>3658</td>
<td>8443</td>
</tr>
<tr>
<td>( &lt;k&gt; )</td>
<td>9.92</td>
<td>8.81</td>
<td>7.74</td>
<td>9.87</td>
</tr>
<tr>
<td>( D )</td>
<td>3.38</td>
<td>3.43</td>
<td>3.51</td>
<td>4.19</td>
</tr>
<tr>
<td>( C )</td>
<td>0.41</td>
<td>0.48</td>
<td>0.37</td>
<td>0.44</td>
</tr>
<tr>
<td>( D_{\text{rand}} )</td>
<td>2.81</td>
<td>3.87</td>
<td>3.74</td>
<td>3.43</td>
</tr>
<tr>
<td>( C_{\text{rand}} )</td>
<td>0.02</td>
<td>0.003</td>
<td>0.006</td>
<td>0.005</td>
</tr>
</tbody>
</table>

Comparison of the software network to the same scale random networks in regards to the average path length and clustering coefficient, their average shortest path length is very close, while the clustering coefficient of random network is much smaller. It indicates that the software class network has a small world phenomenon. This indicates that large elements in the system are actually closely related. Table 1 shows that, there is no significant correlation between the clustering coefficient and the scale of the network from different systems.

For a single software system, whether the size of the clustering coefficient is also independent of the changes in the network size? We analyzed the co-evolution relationship of average clustering coeffi-
cient and the network scale of the system. The results are shown in Figure 7.

As we can see from Figure 7, with increase of the system size, the average clustering coefficient of the software network has a decreasing trend, that is to say, the degree of agglomeration will increase with the size decreases. However, this does not seem true for JFreeChart system. For some versions of similar size, their clustering coefficient gap is large. (e.g., the red rectangle points in JFreeChart). In addition, in Tomcat and DrJava two systems, there gathered a large deviation coefficient point (e.g., red circle points in DrJava and Tomcat). The inconsistency of the changes in the degree of aggregation also indicates that the complexity of software evolution is complex, and it is difficult to find a universally evolution rules in software evolution.

5 CONCLUSIONS

In this paper, we abstract software systems as software networks and apply the theories and methods in complex networks to reveal the dynamic characteristics and laws of the software evolution from a global perspective of software architecture. To illustrate the way to use our proposed method, we use multiple successive versions of some open source systems such as JFreeChart, DrJava, Open Swing and Tomcat, as the research subjects. We find some characteristics of software structure, which can be summarized as follows: (1) the growth of the number of edges and nodes in the software network are close to a linear relationship approximately; (2) most of the software has a smaller average degree, and the stability of the average degree is retained in the evolution process of the software; (3) most of the software networks are scale-free; and (4) the class network of most software is small world network. These rules can be used to understand the evolution and trend of software system design, but also has important significance for building software evolution model.

REFERENCES


