Complexity Metrics for Component-based Software Systems

Jianguo Chen, Hui Wang, Yongxia Zhou, Stefan D. Bruda


Complexity Metrics for Component-based Software Systems

1*Jianguo Chen; 2Hui Wang; 3Yongxia Zhou; 4Stefan D. Bruda

1,2,3 College of Information Engineering, China Jiliang University, Hangzhou 310018, China
jgchen@cjlu.edu.cn, 29706087@qq.com, zhouyongxia@cjlu.edu.cn
4 Department of Computer Science, Bishop’s University, Sherbrooke J1M 1Z7, Canada
bruda@cs.ubishops.ca
doi:10.4156/jdcta.vol5.issue3.24

Abstract

In recent years, the software engineering community has put considerable efforts into the design and development of component-based software system (CBSS) in order to manage the software increasing complexity and to maximize the reuse of code. This paper presents some of such efforts by investigating the improved measurement tools and techniques, i.e., through the effective software metrics. Upon the research on the classical evaluation measures for software systems, we argue the traditional metrics are not suitable for CBSS. Therefore we provide an account of novel software measures for component by adequate coupling, cohesion and interface metrics. The complexity metrics combined with three metrics on the CBSS level is also investigated. The advantages of our method are discussed as well through a case study in this paper.

Keywords: Complexity Metrics, Coupling, Cohesion, Interface, Component-based Software System

1. Introduction

The core of component-based software system (CBSS), as known as component-based software engineering (CBSE), is to reuse software components. The pressure for reducing software development life cycles and costs has lead to an increasing interest in CBSS that not only facilitates the process of software development but also changes the ways to develop software applications. Nowadays CBSS is getting accepted in the industry as a new effective software development paradigm [1].

Most of the CBSS research has been inclined towards methods and approaches in the development and in comparison of software systems [2]. Some work tries to evaluate the complexity of tools used to create the software artifacts [3]. A very little work has been made for the development of measures/metrics that can be used to evaluate the complexity of components being developed, and the software quality using component integration [4]. However, many of software failures are caused by the inherent complexity of the software development process. One solution to ameliorate such failures can be achieved by improving our software management capabilities by the development of improved software metrics and improved utilization of such metrics.

Software metrics as a subject area is near 40 years old, but it has barely penetrated into the mainstream of software engineering. A key reason for this is that most software metrics activities have missed the most important requirement: providing information to support quantitative managerial decision-making during the software lifecycle. This paper will attempt to provide the quantitative software metrics for CBSS.

Software metrics can usually be classified as being one of two types: (1) process metrics - They are quantifiable attributes of the development or maintenance processes and their environment; (2) product metrics - they are quantifiable features of the software product. Both process and product metrics may be further categorized as: (1) result metric - quantifies some attributes of a completed project, or part project; (2) predictor metric - is used to make an estimate of the value of a result metric. This paper will discuss the product metrics on component and CBSS levels with the further category in the result metric.

Two common vulnerabilities related product metrics are coupling and cohesion both at the stages of design and code. In general, the coupling between components is the degree of mutual interdependence and the component cohesion is the property of connectivity among the aggregate elements of a single
component. It is believed that high coupling and low cohesion make understanding, developing, testing, and maintaining CBSS difficult, leading to the introduction of vulnerabilities [5].

In CBSS, after components have been created or possibly reusable components have been extracted, these components must be evaluated and certified through a software metrics system because the predictability of system quality attributes will depend on the quality attributes of the constituent components. Therefore component evaluation becomes a very important task for CBSS. The widespread reuse of a software component with poor quality may literally lead to vulnerabilities or even disasters.

On the other hand, the evaluation should also be performed on assemblies, rather than only on individual components. The overall quality of the produced system, rather than the individual quality of the components, is more important.

Therefore, in this paper, we will investigate the software metrics on the component and CBSS levels. We will begin by discussing the most commonly used software metrics and review their use in constructing model development process. More importantly, our focuses will first be on the evaluations in the individual component such as cohesion metrics, coupling relations and interface complexity between the components. Then the focus should shift to the overall best solution of CBSS with the individual component assessment acting as a part of the component assembly evaluation.

The rest of this paper is organized as follows. We briefly introduce common traditional software metrics and their properties in section 2. Section 3 discusses the component’s definition and the metrics on the component level, including coupling, cohesion and interface metrics. In Section 4, we will present component assembly metrics, i.e., metrics on the system level. Section 5 is a case study. The related works are discussed in section 6, followed by a summary of the key ideas in this paper in section 7.

2. Common traditional software metrics

As applied to the software product, software metrics that are usually related to the software quality basically measure or quantify the characteristics of the software. Traditional metrics have been applied to the measurement of software complexity of structured systems since 1976. Some common traditional software metrics are: 1) source lines of code (SLOC); 2) cyclomatic complexity; 3) function point analysis (FPA); 4) bugs per lines of code; and 5) code coverage.

2.1. Source lines of code

The simplest software metric is the number of lines of code (LOC or KLOC for thousands of lines of code) used to measure the size of a software program by counting the number of lines in the text of the program's source code. It was, and still is, used routinely to predict the amount of effort that will be required to develop a program, i.e., Effort = f (LOC), as well as to estimate programming productivity such as LOC/person-month or cost ($/LOC) once the software is produced.

However, differences in underlying definitions and counting techniques may make LOC impossible to compare. If different programming languages are involved, metrics involving LOC, if not carefully interpreted, will lead to incorrect conclusions and thereby conceal the real significance of the data. Therefore, the need for more discriminating metrics to measure the complexity of software becomes especially urgent with the increasing diversity of programming languages.

2.2. Cyclomatic complexity

Cyclomatic complexity directly measures the number of linearly independent paths through a program's source code. It is used to measure code complexity by taking into account the program flow graph under the assumption that the effective complexity of a program lies in its structure rather than in a mere statement count.

Given any computer program, we can draw its control flow graph that may consist of several sub-graphs. In each sub-graph, each node corresponds to a block of sequential code and each arc
corresponds to a branch or decision point in the program. The cyclomatic complexity can be computed as (1) [6]:

\[ M = E - N + 2P \]  

where

- \( M \) = cyclomatic complexity.
- \( E \) = the number of edges of the graph.
- \( N \) = the number of nodes of the graph.
- \( P \) = the number of sub-graphs.

### 2.3. Function point analysis

A function point is a unit of measurement to express the amount of functionality the system provides to the user. FPA metric is to compute the total function point value for the system based upon the number of user inputs, outputs, files, inquiries and interfaces [7]. These function-point counts are then weighed (multiplied) by their degree of complexity, e.g., the weights for the five counts can be 4, 5, 10, 4 and 7 at the average complexity, respectively:

\[ \text{FPA} = 4 \times \text{input} + 5 \times \text{output} + 10 \times \text{files} + 4 \times \text{inquiries} + 7 \times \text{interfaces} \]  

### 2.4. Bugs or faults per line of code

The number of bugs or defects observed in a software product provides a metric of software quality. Some alternative measures have been proposed since there is no effective procedure for counting the bugs or defects in the program: (1) number of design changes, (2) number of errors detected by code inspectors, (3) number of errors detected in program tests and (4) number of code changes required.

### 2.5. Code coverage

Code coverage, an indirect measure of quality, is a quantitative measure used in software testing. It measures the code lines that are executed for a given set of software tests and describes the degree to which the source code of a program has been tested. Meanwhile, it finds areas of a program not exercised by a set of test cases and asks for additional test cases to increase coverage. It is a form of testing that inspects codes directly and is therefore a form of white box testing.

### 2.6. Limitations of traditional software metrics

Traditional software metrics are usually applicable to small programs, whereas the metrics for CBSS should depend mainly on the granularity and interoperability aspects of the components.

Size of a component is normally not known to the component developers, whereas most of the traditional size metrics such as SLOC and bugs (faults)/code line and code coverage are based on lines of code, which is not applicable to CBSS.

Traditional cyclomatic complexity metric suite cannot be applicable either in CBSS because operator and operand counts are not known in CBSS and the number of linearly independent paths cannot be measured.

FPA depends on the weights that were developed in a particular environment, which arises about the validity of this method for general application even though some improved measures like adjusting the counting method have been taken.

There are many inherent differences in CBSS and non-CBSS so that the traditional software metrics are inappropriate for CBSS [8]. Besides, the traditional software metrics do not address the interface complexities and integration-level metrics, which are also not applicable to CBSS.
3. Component definition and metrics

3.1. Component definition

The traditional software metrics focus on non-CBSS and are inappropriate to CBSS mainly because the component size is normally not known in advance. Inaccessibility of the source code for some components prevents comprehensive testing. Therefore we will use other metrics to guide the software development process of CBSS.

Software component is usually regarded as a part of the starting platform for service orientation throughout software engineering. In this paper, a component can be regarded as a reusable software element such as a function, file, class, module or subsystem.

In CBSS, each acquired component should be validated at a very early stage in the software development process such as the design stage and coding one. The quality attributes of the component must be measured using validated metrics that should provide a correct estimate of the attributes. Here we will mainly discuss component metrics and system measures at the design stage of life cycle.

3.2. Component general metrics

The important and relevant metrics applicable for the component quality analysis during design stage are [9]:

- **Component Size Metric (CSM):** CSM should be based on the total number of sub-components such as classes or use cases.
- **Weighted Methods per Component (WMC):** the number of local methods defined in the component. WMC is related to size complexity. WMC is the indicator of development and maintainability complexity.
- **Depth of Inheritance Tree (DIT):** the maximum depth of the component in the inheritance tree. The deeper the component is in the inheritance hierarchy, the greater the number of methods it is likely to inherit, making it more complex to predict the component’s behavior.
- **Number Of Children (NOC):** the number of immediate sub-components of a component or the count of derived components. NOC measures inheritance complexity.
- **Count of Base Components (CBC):** the number of base components. Like NOC, CBC measures inheritance complexity.
- **Response set For a Class (RFC):** the set of methods that can potentially be executed in response to a message received by an object of that component. RFC is simply the number of methods in the set, including inherited methods.

Additionally, we will present three more metrics for the component in CBSS: coupling, cohesion and interface metrics. Cohesion is a measure of how strongly-related the various responsibilities of a software component are. Coupling is usually contrasted with cohesion. Low coupling often correlates with high cohesion, and vice versa. Interface metric provides an estimate of the complexity of interfaces.

3.3. Coupling metric

Coupling between components is the number of other components coupled to this component. In CBSS, coupling will be defined as: two components are coupled if and only if at least one of them acts upon the other. Since coupling is the extent to which the components are interdependent, a quantitative measure is to count the way in which one component may dependent on the other.

In order to develop a coupling metric, we begin by regarding any CBSS as a directed graph. The components comprising CBSS are the vertices in the graph. Suppose such a system comprises a set of components \( C = \{C_1, C_2, \ldots, C_m\} \). Let \( M_i \) and \( V_i \) be the sets of methods and instance variables of component \( C_i \). \( MV_{ij} \) is the set of methods and instance variables in class \( C_i \) invoked by class \( C_j \). \( MV_j \), the set of all methods and instance variables in other components, \( 1 \leq i \leq m \) and \( i \neq j \), that are invoked by component \( C_j \), can be defined [10]:

- \( MV_{ij} \)
- \( MV_j \)
Complexity Metrics for Component-based Software Systems  
Jianguo Chen, Hui Wang, Yongxia Zhou, Stefan D. Bruda  

There are usually two kinds of coupling: afferent coupling and efferent coupling. Afferent coupling is the number of other components that depend upon sub-components within the component and is an indicator of the component's responsibility. Efferent coupling is the number of other components that the sub-components in the component depend upon and is an indicator of the component's independence.

### 3.4. Cohesion metric

Cohesion specifies the similarity of methods in a component. It is a measure of the extent to which the various functions performed by a component are related to one another.

Suppose a component j such as a class has a set of method members \( M_j(C) = \{ m_{j1}, m_{j2}, ..., m_{jm} \} \) and a set of instance variables \( V_j(C) = \{ v_{j1}, v_{j2}, ..., v_{jn} \} \). \( E_j(C) \) is the set of pairs \((v_j,m_j)\) for each instance variable \( v \) in \( V(C) \) that is used by method \( m \) in \( M_j(C) \). The cohesion metric for a component \( j \), also named as Cohesion of Methods (COM), is defined as:

\[
\text{COM}_j(C) = \frac{|E_j(C)|}{|V_j(C)| \cdot |M_j(C)|}
\]

### 3.5. Interface metric

Afferent coupling and efferent coupling can also be represented by component interface metric (CIM). CIM is defined as three values:

- Available number of incoming interactions (II).
- Available number of outgoing interactions (OI).
- A ratio of II to OI, i.e., II / OI.

Another interface metric related to a component \( j \) is actual interactions metric (AIM) that measures the interface density in a component. AIM is the ratio between the actual numbers of interactions over potential ones:

\[
\text{AIM}_j = \frac{II_j + OI_j}{II_{j,\text{max}} + OI_{j,\text{max}}}
\]

Here, \( II_{j,\text{max}} \) and \( OI_{j,\text{max}} \) are maximum numbers of input and output interactions in a component \( j \). We will use AIM instead of CIM as the interface metric for a component in this paper.

### 3.6. Sole component complexity metric

If a sole component complexity metric (SCCM) is required, we may combine above three component metrics with different weights for each metric.

\[
\text{SCCM}_j = \alpha \cdot MV_j + \beta \cdot \text{COM}_j + \gamma \cdot \text{AIM}_j
\]

where \( \alpha \), \( \beta \) and \( \gamma \) are the weights for the coupling metric, cohesion and interface metrics of component \( j \) with the condition as \( \alpha + \beta + \gamma = 1 \).
4. Component assembly metric

Combining component-level metrics to obtain system level indicators of quality is a challenging issue. Component assembly evaluation can be of two types: qualitative or quantitative. The former can be performed by an expert, but is prone to be subjective since different experts may produce conflicting evaluations and the comparison between evaluations performed on different assemblies is difficult.

On the other hand, the quantitative evaluation can be objective and repeatable. Through the usage of well-defined metrics, a quantitative quality model of CBSS may facilitate the comparison of the results of evaluations performed on different assemblies.

Having established the measures for the strength of coupling between pairs of components, the cohesion degree within the components and the interface metrics, the final step is to use them as a basis for the measure of the total complexity of CBSS. This is readily achieved by summing all the measures and dividing by the total number of the components in CBSS.

4.1. System coupling metric

The system coupling metric (SCOUP) for CBSS will be:

$$SCOUP = \frac{\sum_{j=1}^{m} MV_j}{m}$$  \hspace{1cm} (7)

Here $MV_j$ is the coupling metric for component $j$ and $m$ is the number of the components in CBSS.

4.2. System cohesion metric

The system cohesion metric (SCOH) for CBSS is:

$$SCOH = \frac{\sum_{j=1}^{m} COM_j}{m}$$  \hspace{1cm} (8)

Similarly, $COM_j$ is the cohesion metric for component $j$.

4.3. System actual interface metric

Based on the measurement of actual interactions for a component $j$, system actual interface metric (SAIM) is the integration of the interface metrics of the total number of components:

$$SAIM = \frac{\sum_{j=1}^{m} AIM_{j}}{m}$$  \hspace{1cm} (9)

4.4. Sole system complexity metric

Similarly, if we need a sole system complexity metric (SSCM), we may combine above three system metrics with different weights for each items.

$$SSCM = \alpha' \times SCOUP + \beta' \times SCOH + \gamma' \times SAIM$$  \hspace{1cm} (10)
where $\alpha'$, $\beta'$ and $\gamma'$ are the weights for system coupling metric, cohesion and interface metrics with the condition as $\alpha' + \beta' + \gamma' = 1$.

5. Case study

Suppose we have a CBSS that consists of four components as Fig. 1. We know that there are some interactions between component A and B (both directions), from A to C (single direction), from A to D (single direction), between B and D (both directions) and from D to C (single direction). The relative data are assumed among four components as Table 1.

![Figure 1. A CBSS consisting of four components](image)

| Component | j | A | B | C | D | II max | OI max | $|E_j(C)|$ |
|-----------|---|---|---|---|---|--------|--------|----------|
| A         | 7 | 6 | 3, 2 | 3, 3 | 4, 2 | 10     | 9      | 20       |
| B         | 4 | 5 | 2, 4 | 3, 2 | 8    | 7      | 5      |          |
| C         | 5 | 6 | 0    | 6    | 9    | 10     |        |          |
| D         | 6 | 5 | 3, 4 | 4, 3 | 8    | 9      | 25     |          |

Two data for $MV_{ij}$ in Table 1 are the numbers of methods and instance variables in class $C_i$ invoked by class $C_j$. According to Fig. 1 and Table 1, we may calculate the coupling and interface metrics on the levels of the individual component CBSS.

5.1. Coupling metrics for the component and system

According to (3), we will get the coupling metric for each component in CBSS, shown in Table 2.

<table>
<thead>
<tr>
<th>Component</th>
<th>j</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>MV j</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>7</td>
<td>6</td>
<td>3, 2</td>
<td>3, 3</td>
<td>4, 2</td>
<td>1.31</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
<td>5</td>
<td>2, 4</td>
<td>3, 2</td>
<td>8</td>
<td>1.22</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>6</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>6</td>
<td>5</td>
<td>3, 4</td>
<td>4, 3</td>
<td>8</td>
<td>1.27</td>
</tr>
</tbody>
</table>

From (7), the coupling metric for the system is:

$$SCOUP = \frac{\sum_{i=1}^{n} MV_j}{m} = 0.95$$
5.2. Cohesion metrics for the component and system

According to (4), we will get the cohesion metric for each component in CBSS, shown in Table 3.

| Component | M_i | V_i | |E_j (C)| COM_j |
|-----------|-----|-----|----------------|---------|
| A         | 7   | 6   | 20             | 0.48    |
| B         | 4   | 5   | 5              | 0.25    |
| C         | 5   | 6   | 10             | 0.33    |
| D         | 6   | 5   | 25             | 0.83    |

From (8), system cohesion metric (SCOH) will be:

\[ SCOH = \frac{\sum_{j=1}^{m} \text{COM}_j}{m} = 0.47 \]

5.3. Interface metrics for the component and system

According to the definitions for CIM and AIM, see (5), we will have the interface metric for each component in CBSS, shown in Table 4.

<table>
<thead>
<tr>
<th>Component</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>II /OI</th>
<th>MV_j,i</th>
<th>MV_j,i</th>
<th>MV_j,i</th>
<th>MV_j,i</th>
<th>II_j,max</th>
<th>OI_j,max</th>
<th>II_j</th>
<th>OI_j</th>
<th>II /OI</th>
<th>AIM_j</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3, 2</td>
<td>3, 3</td>
<td>4, 2</td>
<td>10</td>
<td>9</td>
<td>7</td>
<td>17</td>
<td>0.41</td>
<td>1.26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>2, 4</td>
<td>3, 2</td>
<td>8</td>
<td>7</td>
<td>5</td>
<td>12</td>
<td>0.42</td>
<td>1.13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>9</td>
<td>13</td>
<td>0</td>
<td>Null</td>
<td>0.87</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>3, 4</td>
<td>4, 3</td>
<td>8</td>
<td>9</td>
<td>11</td>
<td>14</td>
<td>0.79</td>
<td>1.47</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

From (9), system actual interface metric (SAIM) will be:

\[ SAIM = \frac{\sum_{j=1}^{m} \text{AIM}_j}{m} = 1.18 \]

5.4. Combined sole system complexity metric

If we give the weights for system coupling, cohesion and interface metrics as 0.5, 0.2 and 0.3, from (10), we will get the SSCM of the exampled CBSS as:

\[ SSCM = \alpha' \times SCOUPL + \beta' \times SOCM + \gamma' \times SAIM = 0.92 \]

5.5. Combined sole component complexity metric

Let’s go back to compute the sole component complexity metric (SCCM) for CBSS. From Table 2 to Table 4, we will have SCCM metrics for each component as Table 5 if we assign the coupling, cohesion and interface weights as 0.5, 0.2 and 0.3, referring to (6).
Table 5. Complexity metrics for the component

<table>
<thead>
<tr>
<th>Component</th>
<th>$MV_j$</th>
<th>$COM_j$</th>
<th>$AIM_j$</th>
<th>$SCCM_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1.31</td>
<td>0.48</td>
<td>1.26</td>
<td>1.129</td>
</tr>
<tr>
<td>B</td>
<td>1.22</td>
<td>0.25</td>
<td>1.13</td>
<td>0.999</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0.33</td>
<td>0.87</td>
<td>0.327</td>
</tr>
<tr>
<td>D</td>
<td>1.27</td>
<td>0.83</td>
<td>1.47</td>
<td>1.242</td>
</tr>
</tbody>
</table>

From Table 5, we know that the component D has the highest complexity metric and needs the most investment such as time/cost, while component C has the lowest complexity value. The complexity metric of component B is less than that of component A.

Similarly as above comparisons of complexity metrics among the components, the system complexity data from section 5.1 to 5.4 such as 0.95 (SCOUP), 0.47 (SCOH), 1.18 (SAIM) and 0.92 (SSCM) have hardly any meaning for themselves. However, when such data are used to compare the complexity levels among several software systems, the developers will know which CBSS needs more people and more time during the coding and testing stages, or they may expect the vulnerabilities will happen in which component according to the complexity metrics.

6. Related works

Some proposals for software metrics in CBSS both concerning individual components and component assemblies have been presented [11]. These proposals provide useful insights on the specificities to consider when developing metrics for CBSS. However they do not contribute with concrete metrics. Several authors made proposals (component centric) for the evaluation of component interfaces and dependencies [12]. Hoek et al. proposed metrics to assess service utilization in component assemblies. Narasimhan and Hendradjaya introduced metrics to assess component integration density and component interface density [13].

All above referred metric proposals included informal specifications to some extent. Formalization in software metrics for CBSS has begun. Goulao et al. presented formal definition of composition assessment metrics for CBSS, using an extension of the CORBA Component Model meta-model as the ontology for describing component assemblies [14]. We also made an attempt in formal metrics for CBSS and presented the novel direct and indirect component coupling metrics with formal specifications [15].

The complexity metrics presented in this paper provides formal specifications of software metrics and novel quantitative software measures by combining coupling, cohesion and interface metrics on the levels of the component and CBSS.

7. Conclusions

The pressure for reducing software development life cycles and costs has lead to an increasing interest in CBSS. In order to manage the software increasing complexity and to maximize the reuse of code, adequate metrics to quantify component quality on the levels of both individual component and system are necessary. Though there are some proposals aiming at establishing requisites and guidelines for CBSS metrics, most current metrics proposals include informal specifications.

Upon the research on the classical evaluation measures for software systems, we argue that the traditional metrics that mainly rely on the lines of codes are not suitable for CBSS. Therefore we provide an account of novel security measures for component and CSS by adequate coupling, cohesion and interface metrics in this paper.

Our focuses are to evaluate both individual component and assembly relation between components at the design stage of CBSS development life cycle. We believe that our efforts may help to manage the complexity of CBSS and to validate the component/system at a very early stage in the software development process. The case study shows that the methods presented in the paper are effective by comparing the complexity metrics for each component. We believe the complexity measures for CBSS must be also useful in a similar way.
The software metrics research for CBSS is still in the early stages in the long run. We will continue our effort in software metrics for CBSS such as to optimize the coupling metric by combining indirect coupling into direct coupling. More empirical research by applying our novel metrics in the real CBSS systems is also one of our future works.

8. Acknowledgment

This study has been carried out as a part of the project “High performance computing of realistic fluid based on GPU”, financed by the National Nature Science Foundation of China (Grant No.10702067). The research was supported by 2010 returned scholar scientific activity project selective assistance with the project title as “Research of P2P Network Security Based on Natural Computation and Trust Management”, which was funded by the Department of Human Resources and Social Security of Zhejiang Province. This work was also supported by the key course project, high education project and the scientific research startup project of China Jiliang University Jiliang University, Natural Sciences and Engineering Research Council of Canada and Fond Québécois de recherché sur la nature et les technologies. The comments from three colleagues are also gratefully acknowledged.

9. References