International Journal of Grid Distribution Computing
Vol. 8, No.4, (2015), pp.1-10
http://dx.doi.org/10.14257/ijgdc.2015.8.4.01

Modularizing Legacy System through an Improved Bunch Clustering Method in Cloud Migration

Junfeng Zhao[1], Jiantao Zhou[1] and Hongji Yang[2]

[1]*College of Computer Science Inner Mongolia University*
Hohhot, China cszjf@imu.edu.cn, cszjtao@imu.edu.cn
[2]*Centre for Creative Computing Bath Spa University Bath, England*
h.yang@bathspa.ac.uk

Abstract

As a new service mode in cloud computing, SaaS (software as a service) brings many attractive advantages. Legacy systems can be revived through being reengineered to SaaS. In order to achieve reengineering, the analysis and understanding to legacy systems are essential. For this goal, an improved Bunch clustering system is proposed to implement automatic modularization to object-oriented software systems so as to help engineer understand legacy system, including introduction of modular dependency graph with relationship type information, adaptation to initial partition and adjustment to modularization quality. The experiment results show that the improvement of Bunch clustering system is effective. The improved Bunch clustering system can make the clustering results more stable and consistent to the benchmarks.

Keywords: *legacy system, software reengineering, software as a service, clustering, Bunch*

1. Introduction

Cloud computing provides innovative computing modes. More and more users seek to make use of the advantages brought by cloud computing through building their applications in the cloud. In addition to constructing new applications based on the cloud, migrating legacy system to the cloud is also considered favorably. According to the service models of cloud computing, cloud migration can be divided into different categories, one of which is reengineering legacy system to SaaS [1].

For reengineering legacy system to SaaS, the analysis and understanding of legacy system is necessary, and this can be implemented by software architecture recovery. Software architecture embodies the implementation information in high level abstraction. Many architecture recovery techniques have been proposed such as ACDC [2], ARC [3], Bunch [4], LIMBO [5] and WCA [6]. A comparative analysis to these software architecture recovery techniques is carried out [7]. The results indicate that ACDC and ARC are superior to the others among multiple measures. At the same time, the results also show that there has much improvement space in all the clustering techniques.

Experiments results show some clustering techniques can produce higher overall accuracy than Bunch does, but these techniques need more manual participation and artificial analysis of legacy system before clustering so that the automation degree decreases. On the contrary, Bunch implements software architecture recovery more automatically than other clustering techniques do, though the overall accuracy is not the best in existing clustering techniques.

This paper tries to improve the accuracy of Bunch by introducing new ideas, such as using R-MDG (Modular Dependency Graph with Relationship information) as input data of clustering instead of MDG (Modular Dependency Graph). Then three main adaptations are conducted to the existing Bunch clustering algorithms. Finally, one open source

ISSN: 2005-4262 IJGDC
Copyright ⓒ 2015 SERSC

software systems are clustered as legacy systems to test the validity of the adaptations. Experiment results indicate that the adaptations are effective, and improved Bunch clustering algorithm can produce more accurate clustering results.

The structure of the rest of this paper is as follows: Section 2 presents an overview of the related work. Section 3 describes the adaptations to Bunch clustering system. Section 4 introduces the implementation of a prototype tool, and one experiment are carried out to verify the effectiveness of the adaption to Bunch clustering system. Section 5 concludes and suggests future research issues.

2. Related Work

Many automated software architecture approaches have been proposed in recent three decades [8-11]. Implementation-level entities such as files, classes or functions are grouped into clusters by applying these approaches where each cluster represents a component [12, 13]. In the midst of these approaches, SAR (Software Architecture Reconstruction) and component reuse by using the cluster analysis approach are paid more attention. SAR techniques are classified into three automation levels such as quasi-manual, semiautomatic, and quasi-automatic in work [14]. The authors observed that many techniques for architecture recovery, even not wholly based on clustering, employ clustering at some stage to derive and present results.

Rigi used clustering to form cohesive subsystems [15]; Harris et al. used clustering of files to support their bottom-up recovery process [16]; Sartipi et al. used an approach based on association rule mining and clustering to recover system's architecture [17]. Unlike other software clustering tools, Bunch clustering system uses search techniques to perform clustering, which decomposes legacy system into subsystems by partitioning the entity and relation graph of the legacy code. In the process of clustering, a fitness function is adopted to evaluate the quality of graph partition [18]. Work in [19] described a comparative study of six software clustering algorithms including ACDC and Bunch. Experimental results show that the studied algorithms exhibit distinct characteristics. Finally, a conclusion is obtained that current automatic clustering algorithms need to be improved significantly so as to provide continual support for large-scale software projects. To combine the strengths of various clustering algorithms, Cooperative Clustering Technique (CCT) was proposed to allow clustering algorithms to cooperate at intermediate steps during the clustering process [20]. For improving the quality of software clusters, a supervised feature selection technique for unlabeled data was proposed for software clustering [21].

With regards to software systems implemented in object oriented languages, the relationships between classes include different types such as implementation, generalization, association and dependency. Different relationship reflects distinct cohesion degree between classes. But the dependencies between modules of MDG in Bunch are considered equally, which will affect the accuracy of clustering result to object-oriented legacy system in the end. Work in [22] further improved Bunch clustering algorithm by introducing building block partitions. Until now, MDG is still used as data source of clustering in Bunch where all edges are considered equally as before. This paper introduces R-MDG as input data for clustering. Based on R-MDG, the Bunch clustering system will be improved from several aspects.

3. Improving Bunch Clustering Methods

In this paper, we adapt Bunch from several aspects, aiming at recovering software architectures of object oriented software systems. The most important adaptation is that we use R-MDG as input data in the improved Bunch clustering method. For object oriented software systems, a clustering algorithm groups classes into relatively independent clusters in terms of the dependency degree between them. In addition to the

 Copyright ⓒ 2015 SERSC

amount of the relationships, the dependency degree between two classes has something to do with the type of their relationship, which embodies distinct cohesion degree and influences the production of the clusters. For example, under same conditions, two classes with generalization relationship have higher cohesion degree than with dependency relationship, and should be grouped into one cluster compared with dependency relationship.

Differing from the MDG in [4], we transform class diagrams into R-MDG which includes specific relationship type information between the classes. In fact, R-MDG is a kind of directed graph where vertex corresponds to class and edge indicates specific relationship between classes. So a class diagram is mapped to a directed graph $G = (V, E)$ where V is vertex set and E is edge set whose type is an enumeration including realization, generalization, association and dependency. In the transformation process, a class in class diagram is mapped to a vertex in R-MDG. For an edge, the mapping rules need to be considered carefully, which are shown as follows:

- Inheritance — if class A extends class B, a direct edge (A, B) will be added to E whose relationship type is marked as G denoting generalization.
- Realization — If class A implements interface B, a direct edge (A, B) will be added to E whose relationship type is marked as I denoting implementation.
- Association, Aggregation, Composition — If one of these three relationships exists between class A and B, a direct edge (A, B) will be added to E whose relationship type is marked as A denoting association.
- Dependency — if class A depends class B, a direct edge (A, B) will be added to E whose relationship is marked as D denoting dependency.

On the basis of the improvement to input data, we have improved the existing Bunch clustering algorithm as follows, whose pseudo code is shown as Figure 1.

Firstly, the method of initial partition in clustering algorithm is improved. To the hill-climbing search algorithm that has been implemented in Bunch, it starts from a random partition of the MDG. Because the random partition cannot embody any meaningful partition, many follow-up calculations could be redundant. Conversely, a clustering algorithm that starts from a relatively reasonable initial partition will be more efficient to clustering. Based on the relationship type information in R-MDG, a new method for initial partitioning can be proposed. We think that the different relationships reflect diverse cohesion degrees between classes. The cohesion degree sequence from strong to weak is as generalization, implementation, composition, aggregation, association and dependency. Due to high cohesion degree of generalization and implementation, we arrange the classes with these relationships into a same cluster in initial partition, then each of other classes be arranged into different clusters separately. Because the initial partition produced by this approach avoids randomness and imports determinacy, the improved algorithm can improve the efficiency of clustering. The first selected programming statement in Figure 1 calls method *initPartition* to obtain initial partition of software systems. To the implementation of Method *initPartition*, all edges in R-MDG are traversed and their attribute *relation Type* is analyzed. When the relationship type of edge belongs to implementation or generalization, the classes corresponding to the start point and end point of the edge will be arranged into a same cluster, and the other classes without these two relationship types will be arranged into individual clusters separately. In the end, the classes linked by edges with high cohesion degree will form a cluster, and each of the other classes will form its own cluster itself.

Copyright ⓒ 2015 SERSC

```
VerticalClustering(GraphMLSoftwareSystem)
1.  currentPartition=initPartition(SoftwareSystem);
2.  maxMQ=MQCalculate(currentPartition);
3.  while (annealingCondition==true)
4.  {
5.      change=false;
6.      allNeighborPartition=getAllNeighborPartition(currentPartition);
7.      for (partition: getPartOfAllNeighborPartition())
8.      {
9.          mq= MQCalculate(partition);
10.         if (mq>maxMQ)
11.         {
12.             maxMQ=mq;
13.             currentPartition=partition;
14.             Change=true;
15.         }
16.     }
17.     If (change==true)
18.     {
19.         maxPartition=currentPartition;
20.     }
21.     else
22.     {
23.         currentPartition=getRandomPartition();
24.     }
25.     updateAnnealingCondition();
26. }
27. return maxPartition;
```

Figure 1. Pseudo Code of New Clustering Algorithm

Secondly, an innovative objective function is redefined in the improved clustering algorithm. Due to the different cohesion degree of relationship types, their respective contributions to MQ value should be different, but this characteristic cannot be embodied in the clustering algorithm in [4]. So we do effective improvement to the implementation of MQ calculation in our approach. A different weight is assigned to a specific relationship type in the improved clustering algorithm, through which more reasonable clustering results could be obtained by pursuing higher MQ value. Formally, w_G, w_I, w_A and w_D respectively represent the weight of generalization, implementation, association and dependency relationship. G_i, I_i, A_i and D_i respectively represent the amount of the internal edges with generalization, implementation, association and dependency relationship in cluster *i*. $G_{i,j}$ and $G_{j,i}$, $I_{i,j}$ and $I_{j,i}$, $A_{i,j}$ and $A_{j,i}$, $D_{i,j}$ and $D_{j,i}$ respectively represents the amount of the external edges with generalization, implementation, association and dependency relationship between two distinct cluster *i* and *j*. MQ value is defined as Equation (1) in this paper.

$$
\begin{aligned}
MQ &= \mathrm{sum}_{i=1}^{k} CF_i / k \\
i_{in} &= (w_I * I_i + w_G * G_i + w_A * A_i + w_D * D_i) \\
i_{ex} &= \sum_{j=1, j\neq i}^{k} (I_{i,j} + I_{j,i}) * w_I + (G_{i,j} + G_{j,i}) * w_G \\
&\quad + (A_{i,j} + A_{j,i}) * w_A + (D_{i,j} + D_{j,i}) * w_D \\
CF_i &= \begin{cases} 0 & i_{in} = 0 \\ \dfrac{2 * i_{in}}{2 * i_{in} + i_{ex}} & \text{otherwise} \end{cases}
\end{aligned} \tag{1}
$$

The value of MQ is a decimal obtained by dividing sum by k, the number of clusters, which can reflect the average cohesion degree more efficiently, and avoid dividing classes into too many clusters for pursuing the maximal MQ in the process of clustering. Through the usage of the relationship weight, we can partition the graph into clusters more accurately than the existing cluster algorithm. The method *MQCalculate* is designed to compute MQ value of a partition which reflects the quality of the partition. In Fig. 1, the third selected programming statement calls method *MQCalculate* to obtain MQ value of neighbor partition.

 Copyright ⓒ 2015 SERSC

In the end, a heuristic-search technique is adopted to discover acceptable suboptimal results quickly. To large-scale legacy software, the universal set of graph partition will be tremendous. In order to decrease the calculation, only part of the available neighbor partitions is considered by adopting specific strategies in method *get Part of All Neighbor Partition*. Specifically, a threshold value of ratio is defined to determine the amount of partitions that will be calculated, while the choice of the partitions involved in calculation is random in iterations.

4. Prototype Tool and Improvement Evaluation

4.1. Prototype Tool

Based on the improved clustering algorithm, we developed a prototype tool to realize software architecture recovery for software system implemented in Java. One function in this tool is the recovery of R-MDG, and the other core function is the implementation of various clustering algorithms.

According to the thought of Bunch clustering algorithm [4], we implemented the existing algorithm in our tool where specific relationship information in R-MDG is ignored. Then, based on the adaptation to the existing Bunch clustering algorithm, the improved Bunch clustering algorithm is implemented in this tool. In the implementation of the prototype tool, two core problems are considered particularly, and the reasonable value are confirmed.

I. Weights could be assigned to the different types of relationships by a configuration file, which will affect clustering result observably. Through verification by study cases, it is relatively reasonable that the weights are assigned to 1.5, 1.4, 1.2 and 1 for generalization, implementation, association and dependency.

II. In order to decrease the clustering time, a threshold $\tau\,(0\% \le \tau \le 100\%)$ is defined to determine the minimum number of neighbors that must be calculated in each iteration of the clustering process. Through verification by study cases, we confirm that the threshold τ with value 80% is reasonable to the clustering.

In order to verify the validity of our adaptation, we conducted several case studies where existing and improved Bunch clustering algorithms are executed to recover software architecture, one of them is GeoScope.

4.2. Evaluation by GeoScope

GeoScope is a free, open-source Java toolkit for geographical analysis and localization of IP addresses. It was developed in 2004, and now reengineering this legacy system to service is significant. Firstly, the R-MDG of knowledge sharing system is obtained by using the prototype tool. The excerpt of Graphml file corresponding to the produced R-MDG of GeoScope is shown as Figure 2. From Figure 2 we can find that 56 classes and 134 relationships between them are discovered.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE graphml SYSTEM "http://www.graphdrawing.org/dtds/graphml.dtd">
<graphml>
    <graph id="G" edgedefault="directed">
        <node id="1" name="com.eaio.geoscope.GeoScope" />
        <node id="2" name="com.eaio.geoscope.GeoScopeException" />
        ......
        <node id="55" name="com.eaio.util.ui.HorizontalRule" />
        <node id="56" name="com.eaio.util.ui.UIColors" />
        <edge id="1" source="1" target="2" relationType="dependency" />
        <edge id="2" source="1" target="3" relationType="dependency" />
        <edge id="10" source="1" target="11" relationType="association" />
        <edge id="11" source="1" target="12" relationType="association" />
        <edge id="12" source="1" target="13" relationType="dependency" />
        <edge id="13" source="1" target="14" relationType="dependency" />
        <edge id="14" source="2" target="15" relationType="generalization" />
        <edge id="22" source="5" target="18" relationType="generalization" />
        <edge id="31" source="7" target="18" relationType="generalization" />
        <edge id="34" source="8" target="18" relationType="generalization" />
        <edge id="37" source="21" target="18" relationType="implementation" />
        <edge id="38" source="22" target="2" relationType="association" />
        <edge id="39" source="22" target="21" relationType="generalization" />
        ......
        <edge id="133" source="42" target="56" relationType="dependency" />
        <edge id="134" source="55" target="56" relationType="dependency" />
    </graph>
</graphml>
```

Figure 2. Graphml File of Knowledge Sharing System

Then we cluster the R-MDG by using the existing and improved Bunch clustering algorithms for ten times respectively too. The respective clustering results are shown as Table 1 and Table 2. Table 1 indicates that the clustering results produced by the existing Bunch clustering algorithm diverge obviously, and so do MQ value and clustering time. On the contrary, the results generated by the improved Bunch clustering algorithm are relatively stable, and MQ values and clustering time are in the same case.

In order to verify the quality of the clustering result, the reference decomposition of GeoScope is defined as benchmark. Through analyzing the documentation of GeoScope, implementation of GeoScope is divided into thirteen modules according to their function, which is shown as Table 3. MoJo measures similarity between two clusters by calculating the minimum number of module move and cluster join operations required to transform one cluster into the other [23]. By comparing the clustering results with the benchmark, The MoJo distances between clustering results and reference decomposition are shown as Table 4. Through analyzing the data shown in the tables, we can find that the improved clustering algorithm can produce more reasonable clustering results than the existing algorithm.

Table 1. Clustering Result of Geoscope by the Existing Bunch Clustering Algorithm

run	number of clusters in initial partition	number of clusters in result	MQ value	clustering time (millisecond)
1	42	22	6.7473	61567
2	24	21	6.7436	38682
3	10	10	5.5073	11499
4	8	11	5.5442	11970
5	56	24	6.6591	68454
6	19	19	6.6950	25711
7	28	21	6.3869	44175
8	32	22	6.7450	44329
9	37	25	6.4670	50047
10	30	25	6.6494	41190

Table 2. Clustering Result of Geoscope by the Improved Bunch Clustering Algorithm

run	number of clusters in initial partition	number of clusters in result	MQ value	clustering time (millisecond)
1	35	14	0.4390	27952
2	35	15	0.4212	25972
3	35	14	0.4333	22230
4	35	14	0.4336	26660
5	35	15	0.4212	24817
6	35	15	0.4150	23816
7	35	15	0.4150	23134
8	35	14	0.4333	23198
9	35	14	0.4333	22623
10	35	14	0.4262	21811

 Copyright ⓒ 2015 SERSC

Table 3. Reference Decomposition of GeoScope

Module name	Function description	Id of member classes
Module1	provider of relational databases	2, 15, 22
Module2	access to geo data	5, 6, 21
Module3	file conversion from the NGA: GNS GEOnet Names Server format to the GeoScope binary file format	28, 29, 30, 33
Module4	user interface	41, 42, 43, 53, 54, 55, 56
Module5	extraction of country information	11, 24, 12, 46
Module6	fast searching in files	36, 37, 52
Module7	access to registry data	8, 9, 31, 40
Module8	encapsulation to the business logic	1, 3, 4, 14, 16, 35, 45
Module9	extraction of city information	23, 26, 10, 13, 25
Module10	file conversion from the U.S. Census Bureau format to the GeoScope binary file format	17, 27, 38
Module11	conversion to registry data	32, 39, 44, 47
Module12	Simplification to localization and internationalization	34, 48, 49, 50, 51
Module13	persistence to generated data	7, 18, 19, 20

Table 4. MOJO Distance between Clustering Results and Reference Decomposition

Results	Existing algorithm	Improved algorithm
1st clustering result	20	7
2nd clustering result	19	9
3rd clustering result	17	8
4th clustering result	16	8
5th clustering result	22	9
6th clustering result	20	10
7th clustering result	19	10
8th clustering result	20	8
9th clustering result	22	8
10th clustering result	22	9
average	19.7	8.6

4.3. Evaluation Summary

The introduction of relation type information enriches input data of clustering algorithm, which makes initial partition more well-grounded and MQ more meaningful. In addition, the improved measurement of MQ embodies averaged cohesion degree, avoiding producing more clusters for only pursuing the maximum of MQ. Evaluation based on this case study indicates that the improved clustering algorithm can produce more reasonable clustering results and save clustering time.

For showing the improvement degree, we design specific equations to obtain numerical representation. Equation (2) and (3) are defined to calculate the improvement value of modularization accuracy and time efficiency as follows. In order to make the results have

Copyright ⓒ 2015 SERSC

statistical significance, we clustered two experiment cases 100 times respectively, and achieved the average value.

$$imp_{accuracy} = \frac{MOJO_{existing} - MOJO_{improved}}{num_{class}} \times 100\% \quad (2)$$

$$imp_{efficiency} = \frac{t_{existing} - t_{improved}}{t_{existing}} * 100\% \quad (3)$$

In Equation (2), $MOJO_{existing}$ represents the average MOJO distance between the clustering results obtained by the existing clustering algorithm and reference decomposition, and $MOJO_{improved}$ is for the improved decomposition, and num_{class} indicates the number of the classes involved in clustering. By calculation, we can find that the cluster accuracy is improved by 18.3% in the first case, and the value to the second case is 19.6%. In Equation (3), $t_{existing}$ represents the average time spent for clustering in the existing clustering algorithm, and $t_{improved}$ is for the improved decomposition. By calculation, we can find that the time is saved by 20.8% in the first case, and the value to the second case is 37.3%.

5. Conclusions

Reengineering legacy systems to a cloud platform can make existing application revitalized, while understanding to legacy systems is necessary to software reengineering. Software architecture can embody system structure in high abstraction level, so recovering software architectures through reengineering could make developers understand legacy systems better.

In this paper, an improved Bunch clustering algorithm is proposed for software architecture recovery. Firstly, MDG with relationship type information is adopted to describe the implementation of legacy system, which can accurately describe the implementation details. Then an improved clustering algorithm is proposed. Finally, based on the improved Bunch clustering algorithm, a prototype tool is developed to recover software architecture of legacy system in Java. One case study is conducted, and the experiment results show that that the improved clustering algorithm is superior to the existing clustering algorithm. Besides the above positive conclusions, a few issues should be addressed in the future work, so that the algorithm could generate more reasonable clustering results. Firstly, the granularity size of the cluster should be paid more attention, avoiding only pursuing the maximum of MQ. Secondly, a case study of large-scale software systems should be conducted so as to test the validity to complex systems.

References

[1] J. F. Zhao and J. T. Zhou, “Strategies and methods for cloud migration”, International Journal of Automation and Computing, vol. 11, no. 2, **(2014)** April, pp. 143-152.
[2] V. Tzerpos and R. Holt, “ACDC: an algorithm for comprehension-driven clustering”, 7th Conference on Reverse Engineering, **(2000)**, pp. 258-267.
[3] J. Garcia, D. Popescu, C. Mattmann, N. Medvidovic, and Y. Cai, “Enhancing architectural recovery using concerns”, IEEE/ACM 26th International Conference on Automated Software Engineering (ASE), **(2011)**, pp. 552-555.
[4] B. S. Mitchell and S. Mancoridis, “On the automatic modularization of software systems using the bunch tool”, IEEE Transactions on Software Engineering, vol. 32, no. 3, **(2006)** March, pp. 193-208.
[5] P. Andritsos and V. Tzerpos, “Information-theoretic software clustering”, IEEE Transactions on Software Engineering, vol. 31, no. 2, **(2005)** February, pp. 150-165,.
[6] O. Maqbool and H. A. Babri, “Hierarchical clustering for software architecture recovery”, IEEE Transactions on Software Engineering, vol. 33, no. 11, **(2007)** November, pp. 759-780.

 Copyright ⓒ 2015 SERSC

[7] J. Garcia, I. Ivkovic and N. Medvidovic, "A comparative analysis of software architecture recovery techniques", IEEE/ACM 28th International Conference on Automated Software Engineering (ASE), **(2013)**, pp. 486-496.
[8] L. Belady and C. Evangelisti, "System partitioning and its measure", Journal of Systems and Software (JSS), vol. 2, no. 1, **(1981)** February, pp. 23-29.
[9] D. H. Hutchens and V. R. Basili, "System structure analysis: Clustering with data bindings", IEEE Transactions on Software Engineering, v-SE. 11, no. 8, **(1985)** August, pp. 749-757.
[10] R. W. Schwanke, "An intelligent tool for re-engineering software modularity", 13th International Conference on Software Engineering, **(1991)**, pp. 83-92.
[11] R. W. Schwanke and S. J. Hanson, "Using neural networks to modularize software", Machine Learning, vol. 15, no. 2, **(1994)** May, pp. 137-168.
[12] S. W. Kang, S. Lee and D. Y. Lee, "Architecture reconstruction: Tutorial on reverse engineering to the architectural level", Software Engineering, vol. 5413, **(2009)**, pp. 140-173.
[13] T. A. Wiggerts, "Using clustering algorithms in legacy systems remodularization", 4th Working Conference on Reverse Engineering, **(1997)**, pp. 33-43.
[14] S. Ducasse and D. Pollet, "Software architecture reconstruction: A process-oriented taxonomy", IEEE Transactions on Software Engineering, vol. 35, no. 4, **(2009)** April, pp. 573-591.
[15] H. A. Muller, S. R. Tilley and K. Wong, "Understanding software systems using reverse engineering technology perspectives from the Rigi project", Conference of the Centre for Advanced Studies on Collaborative Research: Software Engineering, **(1993)**, pp. 217-226.
[16] D. R. Harris, H. B. Reubenstein and A. S. Yeh, "Reverse engineering to the architectural level", 17th International Conference on Software Engineering, **(1995)**, pp. 186-195.
[17] K. Sartipi, K. Kontogiannis and F. Mavaddat, "Design recovery using data mining techniques", European Conference on Software Maintenance and Reengineering, **(2000)**, pp. 129-139.
[18] S. Mancoridis, B. Mitchell, Y. Chen and E. Gansner, "Bunch: a clustering tool for the recovery and maintenance of software system structures", IEEE International Conference on Software Maintenance (ICSM), **(1999)**, pp. 50-59.
[19] J. W. Wu, A. E. Hassan and R.C. Holt, "Comparison of clustering algorithms in the context of software evolution", 21th IEEE International Conference on Software Maintenance, (2005), pp. 525-535.
[20] R. Naseem, O. Maqbool and S. Muhammad, "Cooperative clustering for software modularization", Journal of Systems and Software, vol. 86, no. 8, **(2013)**, pp. 2045-2062.
[21] Z. Shah, R. Naseem, M. A. Orgun, A. Mahmood and S. Shahzad, "Software clustering using automated feature subset selection", 9th International Conference on Advanced Data Mining and Applications, **(2013)**, pp. 47-58.
[22] B. S. Mitchell and S. Mancoridis, "On the evaluation of the Bunch search-based software modularization algorithm", Soft Computing, vol. 12, no. 1, **(2008)** January, pp. 77-79.
[23] V. Tzerpos and R. C. Holt, "Mojo, A distance metric for software clustering", 6th Working Conference on Reverse Engineering, **(1999)**, pp. 187-193.

Authors

Jun-Feng Zhao received his bachelor degree in computer science from Inner Mongolia normal university, and the master degree in software engineering from National University of Defense Technology, Changsha, and RPC in 2006. He is currently a lecturer and a Ph.D. candidate at College of Computer Science, Inner Mongolia University. His research interests include software reengineering, formal modeling and cloud computing.

Mr.Zhao is a member of CCF.

E-mail: cszjf@imu.edu.cn

Jian-Tao Zhou was born in 1974, and received her PhD degree from Tsinghua University in 2005. She is a professor and PhD supervisor in College of Computer Science, Inner Mongolia University. Her current research interests concentrate on network computing and formal methods.

E-mail: cszjtao@imu.edu.cn

Copyright ⓒ 2015 SERSC

 Copyright ⓒ 2015 SERSC