http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 7 (1): 56-62, 2008
ISSN 1812-5638

A New Approach of Component Identification Based on Weighted Connectivity Strength Metrics

Xinyu Wang, Xiaohu Yang, Jianling Sun and Zhengong Cai
College of Computer Science, Zhejiang University, Zheda Road 38, Hangzhou, Zhejiang, 310027, China

Abstract: Once software is in production, maintenance works are inevitable e.g., bug fixes, requirement changes, etc. During the long-term software maintenance, documents become gradually inconsistent with the system and the source code becomes the most reliable source for design recovery. Currently there exist many approaches to identify component from source code, which adopt different component metrics and clustering algorithms. However, these approaches are not satisfying in the precision and efficiency. This study proposes a new component extraction approach based on 0 metrics and hierarchical clustering algorithm, which greatly improves the precision and efficiency.

Key words: Component, clustering algorithm, component metrics, connectivity strength, component complexity

INTRODUCTION

Once software is in production, maintenance works are inevitable, e.g., bug fixes, requirement changes, etc. This requires engineers be familiar with the design of the system. In the long-term software maintenance, documents gradually become inconsistent with the system and therefore the source code becomes the most reliable source for design recovery.

Currently, a component-based software development pattern is formed on the basis of object-oriented software developing technologies. Component technology becomes increasingly important in software development (Holz and Kath, 2001; Huber *et al*., 1998) and hence it is important in design recovery to obtain component information of the system. There are many approaches to component extraction from source code, which adopt different component metrics and clustering algorithms, but these approaches are not satisfying in the precision and efficiency.

Lee *et al*. (2003) defined criterions of component metrics, including connectivity strength, component complexity, etc. In the definition of connectivity strength, Lee assigned equal weight to all user-defined types. However, the complexity of user-defined types in a real system varies greatly, which is not reflected in Lee,s definition and results in low precision of component classification. Cho *et al*. (2001) proposed a more comprehensive component metrics and formalized the computation of these metrics, but still lacked in suitable methods of selecting certain parameter weight; Xu *et al*. (2002) analyzed class cohesion through their attributes and methods, formalized measuring of cohesion and satisfied the requirement of component independence; Allen and Khoshgo Ftaar (1999) proposed metrics of component coupling and cohesion, explained complete component attributes of coupling and cohesion, as well as the computation principles. But these two methods did not provide quantified metrics scale.

Jain (2001) and Lee *et al*. (2005) proposed an approach to construct business components for object-oriented system. This approach first analyzed relation of composition and inheritance among classes to obtain primitive components and these primitive components were classified into system component through hierarchical clustering algorithm. But for metrics of connectivity strength, no formalized description was provided; Washizaki and Fukazawa (2005) proposed automatic refactoring-based component extraction approach, which analyzed connectivity strength between classes qualitatively. But the precision of component extraction is not guaranteed without quantitative analysis of this dependency. Meng *et al*. (2005) proposed a component identification approach based on business model and hierarchical clustering algorithm; Lee *et al*. (2003) proposed an approach based on analysis of class cohesion and inter-class coupling ess-domain knowledge on engineers.

This study proposes a nto recover system components. These two approaches place very high

Corresponding Author: Xinyu Wang, College of Computer Science, Zhejiang University, Zheda Road 38, Hangzhou, Zhejiang, 310027, China

requirement of businew component extraction approach, which is optimized both in clustering metrics and clustering algorithm. On the basis of connectivity strength and considering variation of user-defined types, this study proposes a new measure of component metrics, Weighted Connectivity Strength (WCS). WCS reflects differences of user-defined classes in the system better and assigns high weight to crucial classes, enlarges the connectivity strength of classes related with crucial classes. This helps closely related classes to be easily clustered into a component and obtains components with better service.

WEIGHTED CONNECTIVITY STRENGTH

In complex systems, the differences of complexity and importance between classes are obvious and the process of component extraction shall reflect this kind of difference. Zero metrics (Lee *et al*., 2003) assigned equal weight to all user-defined classes and did not reflect this difference. This study proposes weighted connectivity strength metrics to measure the connectivity between components. This approach assigns higher weight to crucial classes in the system and such closely related crucial classes are more inclined to cluster into a component. Such metrics can be formalized as:

If S' denotes a system and C_k denotes one component in the system, then their relation can be represented as:

$$S' = \{C_1...C_k\}, k \geq 1$$

WCS may quantify the connectivity strength between components, formalized as:

$$WCS(C_i, C_j) = \sum_{c_u \in CSET(C_i)} \sum_{c_v \in CSET(C_j)} WCS(c_u, c_v)$$

$$WCS(c_u, c_v) = \sum_{m_x \in MSET(c_u)} \sum_{m_y \in MSET(c_v)} WCS(m_x, m_y)$$

$$WCS(m_x, m_y) = \begin{cases} Pri\_Count(m_y) * w_{pri} + \sum_{i=0}^{Abs\_Count(m_y)} w_a, & \text{if } m_x \text{ calls } m_y \\ 0, \text{ Otherwise} \end{cases}$$

Where:

C_i, C_j = Components
C_u, C_v = Classes
CSET (C_i) = The set of classes in C_i
w_{pri} = The weight of a primitive type
W_a = The weight of a user-defined class a
m_x, m_y = Member methods
MSET (c_u) = The set of member methods of class c_u
Pri_Count (m_y) = The number of parameters of primitive type in method (m_y)
Abs_Count (m_y) = The set of parameters of user-defined type in method (m_y)

The connectivity strength between classes not only depends on the amount of function invocation, but also the complexity of the invocation. Invocation of crucial classes in the system indicates closer relations between these classes and these classes shall take precedence in clustering into a component.

The advantage of WCS metrics is analyzed with examples. Suppose we have four classes A, B, C and D and their methods are invoked as: A's methods are invoked by both B and C, represented by I_{BA} and I_{CA}; D's methods are invoked by C, represented by I_{CD}. These invocations take equal number of parameters, 2 user-defined parameters and 2 primary parameters, but differ in parameter complexity. I_{BA} has very high complexity parameters, I_{CA} has very low complexity parameters, I_{CD} has high complexity parameters and there is no invocation between B and C. Using CS metrics, equal weight is set for all user-defined parameters and another weight value for all primary weight. So we can set weight of user-defined parameter 64 and primary parameter 8. Then All of CS (A, B), CS (A, C) and CS (C, D) values are 64*2+8*2 = 144. As illustrated in Fig. 1, with clustering algorithm, A and B are first clustered into a component, temporarily called M1. CS values for C-M1 and C-D are the same, so C may be clustered into M1 with great possibility to get a new component, called M2. Finally, D is clustered into M2 if no other constraints. In fact, C and D are obviously more connected and should be clustered into one component M' before C is clustered into M1.

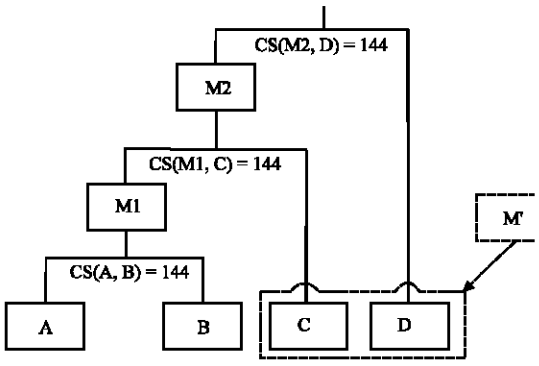

Fig. 1: The clustering processes using CS

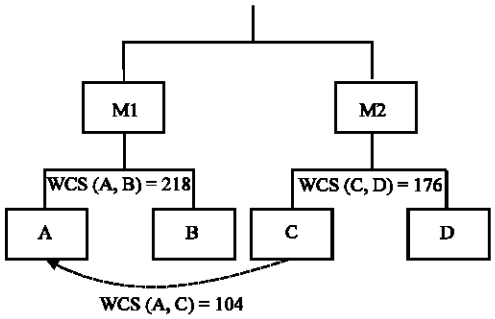

Fig. 2: The clustering processes using WCS

After using WCS metrics and adopting a new weight strategy, the un-weighted COX is used as the weight of this class, which will be discussed later. The weights of the user-defined parameters of I_{BA} are 128 and 74, those of I_{CA} are 24 and 64 and those of I_{CD} are 84 and 76. In this way, the value of WCS (A, B) is 218. Similarly, WCS (A, C) value is 104 and WCS (C, D) value is 176. Therefore, using clustering algorithm, as shown in Fig. 2, Classes A and B are firstly combined into one component, called M1 temporarily. Then in the left components and classes, WCS (C, D) has the largest value 176. So C and D are merged into one component M2 before clustering of M1 and M2 is considered.

Therefore, WCS metrics can better express difference of user-defined classes in the system, cluster closely related classes into a component and improve component cohesion and coupling.

COMPONENT IDENTIFICATION

Component identification includes two steps: First analyze the relation of composition and inheritance between classes to obtain a group of primitive components; then analyze WCS between primitive components and cluster the pair of components with highest WCS into a new component until all WCS between any pair of components fall below a specified threshold.

Relation analysis of composition and inheritance: The aim of component classification is to reduce the coupling of components and increase the cohesion of each single component (Jain, 2001). Classes with composition can be viewed as a function unit and be classified into a component to increase component cohesion. Steps are explained in detail: For classes in a system, if class a is not contained in any component, create a new component comp_a and move class a into this component; if b is a class in component comp_a, then move all classes having composition relation with b into this component.

Classes with inheritance relation are generally strongly relevant and are more suitable to be clustered into a component. If a parent class and its child classes are distributed among different components, then the dependency between components will increase greatly. For primitive components and classes obtained after analysis of composition, subsequent analysis of their inheritance is carried out: If a class in this component is inherited from other classes not in this component, then all parent classes of this class shall be copied to this component; if a class have one or more child classes, then create a new component and copy or move this class as well as its child classes to this newly created component.

Identification algorithm: After a group of primitive components is obtained through analysis of their composition and inheritance, WCS between these primitive components are analyzed and components are optimized with following algorithm, which combines algorithms in (Saeed *et al*., 2003; Wiggerts, 1997):

- Take N components with complexity under certain specified threshold as the initial state and compute WCS between each component:
- While WCS exceeds the specified threshold:

```
Do
  Find the pair of components with highest WCS;
  While (WCOX is lower than the specified
  threshold after clustering those two components)
Do
  Find the pair with highest WCS in remaining pairs
  that have not been examined;
  If (the number of pairs that have not been
  examined = 0)
   Goto the end of the program;
End
  Cluster two components into a single component;
  Re-compute WCS between all components;
End
```

To work with the metrics of WCS, we propose Weighted Component Complexity (WCS), which introduces weight into 0 metrics (Lee *et al*., 2003) to reflect the complexity of components. The principle of WCOX is similar to that of WCS and it will not be explained in detail here.

RESULTS AND ANALYSIS

We choose five programs (http://www.javaresearch.org/oss/jsq/readme.html) (http://www.eclipse.org/downloads/) with different sizes as test cases. The reason why we choose these programs is that we are familiar with

Table 1: Five test programs

Programs	Source code (lines)	Class No.
Jsq	5105	51
Editor	12007	124
Viewer	27015	295
UI	73213	744
Jdt	154372	1500

these programs and the extraction precision of components can be evaluated easily. Test programs in the experiment are shown in Table 1.

Comparison of extraction precision between WCS and CS: For each test program, both CS and WCS are applied for comparison. The test results of component extraction for five Java open source programs in Table 1 are shown in Fig. 3.

Compared with CS, WCS has 20% higher precision in component extraction and has better performance.

To extract components with clustering algorithms based on WCS, certain manual adjustment is inevitable and engineers may adjust component classification to reach a satisfying state according to the characteristic of the real system.

Discussion about weight and threshold: There are many approaches to weight selection, such as the number of member variables, the total number of member variables and member methods, the total number of member variables and statements in member methods, etc. However, these approaches still have some problems on computing weight. For example, the first approach represents the class using number of member variables without operations; The second approach considers the operations of a class, but treats operation and variable as the same, that means, it ignores the difference between variable and operation; The third approach considers the difference and uses statement number to represent operations, however, statement number is easily influenced by code style. This study uses un-weighted COX (Lee *et al.*, 2003) as its weight. COX considers both the member variables and operations. It calculates the sum of operation parameters weight to represent an operation and then calculates the weight of this class. This approach can represent the weight of the class more accurately. The computing process of WCS and WCOX is: first compute the COX for each user-defined class, then take the value as weight of classes and compute weighted complexity and weighted connectivity strength of system components.

$$COX(C_u)=\sum_{a\in TYPE(C_u)}\begin{cases} w_{pri} & \text{if a is primitive type}\\ w_{abs} & \text{if a is not the member of } C_u\\ COX(a)*w_{abs} & \text{Otherwise}\end{cases}$$

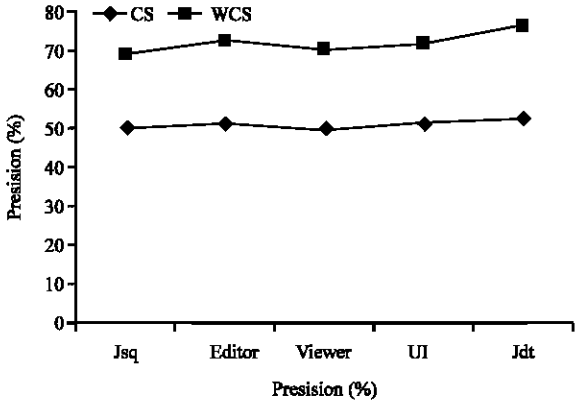

Fig. 3: Comparison of component classification precision

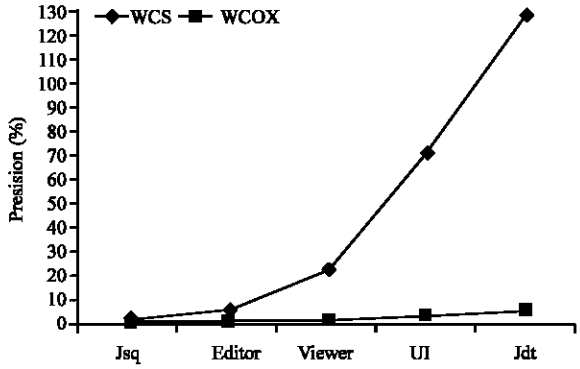

Fig. 4: Threshold for optimum results of system component extraction

Where:

c_u = A user-defined class;

w_{abs} = The weight of user-defined classes. We assign equal weight to all user-defined classes, so as to reflect grossly the status of these classes in the system

For threshold, the test selects certain discrete values to extract components and then discuss the selection of threshold through comparison of these extraction results. In real projects, engineers may select suitable thresholds according to the characteristic of the system. Figure 4 shows thresholds in component extraction for five open source programs in Table 1. It is clear that with the increase of the system size, the increase of WCOX is mild, while the increase of WCS is rapid.

As analyzed from the experimental results, WCOX threshold is linear with the number of classes. In systems with less than 500 classes, the linear slope falls in the range of 30-40 and when the number of classes increases to 500-1500, the slope turns down to 10-20. It is not difficult to understand that the slope of WCOX threshold

decreases as the size of the system enlarges. The component complexity of a system will not increase unlimitedly and normally it is the number of components in a system, rather than complexity of every component, that increases linearly as the system size enlarges. WCS is obtained by computing connectivity strength between classes. WCS (C_i, C_j) of component C_i and C_j are the sum of all the WCS of classes pairs (c_u, c_v), where c_u is from C_i and c_v is from C_j. If there are N classes in each component, WCS of N^2 class pairs (c_u, c_v) are computed to get WCS (C_i, C_j). So WCS (C_i, C_j) has quadratic relation with the class number N. As discussed above, the component complexity threshold is nearly linear with T, which is the number of classes in system. So N is nearly linear with T since the component complexity is also linear with the number of classes in this component obviously. Therefore, the WCS threshold has quadratic relation with the number of classes in the system. After preliminary examination of test data, we found that for systems with 0-1500 classes, the quadratic coefficient falls in the 0-10 range and increases as the number of classes increases. However, the linear coefficient decreases greatly as the number of class increases. For systems with the number of classes below 200, the linear coefficient is 0-200, but for system with 200-1500 classes, the linear coefficient reduced rapidly to the range of -1000 to 0. This might help engineers determine the right threshold.

The relation between WCOX threshold and the number of classes can be formalized as:

$$WCOX = kn+b$$

$$\text{Where, } k \in \begin{cases} (30,40), & n \in (0,500) \\ (10,20), & n \in (500,1500) \end{cases}$$

The relation between WCS threshold and the number of classes can be formalized as:

$$WCS = an^2+bn$$

$$\text{Where, } a \in (0,10), b \in \begin{cases} (0,200) & 0 < n < 200 \\ (-1000,0) & 200 \le n < 1500 \end{cases}$$

Optimization of clustering algorithm: We first analyze the time complexity of hierarchical clustering algorithm: suppose N primitive components are obtained after analysis of composition and inheritance, then there are N^2 pairs of primitive components as cluster candidates; select the pair with greatest connectivity strength from N^2 pair of primitive components, with time complexity of $O(N^2)$; the time complexity processing all possible component pairs is $O(N^4)$ and suppose the average time complexity to compute the connectivity strength of a component pair is O(M), then the time complexity for the whole process is approximately $O(M*N^4)$.

To compute connectivity strength between two components, suppose the number of classes in a system is T and then the average number of classes in every primitive component is T/N and suppose every class contains L statements in average and then the time complexity to compute WCS between components is 2*(T/N)* L, i.e., O(L*T/N). Therefore, the number N of primitive components has great influence on the hierarchical clustering algorithm.

Improvement to clustering algorithm: Compared with hierarchical clustering algorithm, the optimized hierarchical clustering algorithm (Chen *et al*., 2005a, b) reduces the computation complexity while ensuring computation precision. The main idea behind this optimization is:

Select a group of K components as initial center points; then select one component from remaining components to compute its WCS with each component from the group of K central components respectively and merge this component with the initial central component with which it has the highest WCS; then select another component from remaining components and repeat the process above until all remaining components are merged into these K components and finally we have the intermediate result with K components. Take this intermediate result as input and we get final result of component extraction through hierarchical clustering algorithm (i.e., the extraction algorithm in 3.2).

The computation complexity of the optimized hierarchical clustering algorithm, in which K center points are selected and all components are merged into K components, is (N-K)*K*L*T/N, denoted as O(L*T*K). Time complexity of the subsequent hierarchical clustering algorithm is $O(L*T*K^3)$, so time complexity of the whole process is about $O(L*T*K^3)$.

Optimization of WCS computation: In analyzing invocations among all these classes, we do not compute the connectivity strength of every pair of classes one by one, but compute connectivity strength of one class with all other classes and then combine all these connectivity strength together as the connectivity strength of the whole system or certain components.

Compare the complexity of algorithm before and after optimization: suppose there are N primitive components in the system and each component has L statements. When computing the connectivity strength of the whole system, the original algorithm shall traverse all N*(N-1)duplets (c_i, c_j) for all components, analyze the invocation of methods in c_j by L statement in c_i and then add all these connectivity strength together as WCS of the whole system; the complexity is $O(N^2 * L)$. After optimization, we only traverse statements in these N components once, then analyze statements in this component one by one, which has L statements and then compute the connectivity strength between this class and other classes based on the total number of invocations of methods of other classes by each statement in this class, with complexity of only O(N*L). For large systems, normally the value of N is large, so the complexity of the algorithm is greatly reduced.

Analysis of the test result: We take five test programs in Table 1 as test cases; compare the result of algorithms before optimization, after optimization of hierarchical clustering algorithm and with optimized WCS computation based on the last optimization. Comparison of efficiency is shown in Table 2.

It can be found through comparison that the computation time of optimized hierarchical clustering algorithm is reduced to about 50% of that before optimization and the computation time for optimized 0 computation and optimized clustering algorithm is 1/3 to 1/5 of that before optimization.

Table 2: Comparison of Efficiency before and after Optimization (Unit: Minute)

	Test programs	Option I	Option II	Option III
Jsq	Thresholds (WCOX, WCS)	1.5 k; 7.5 k	3k; 15 k	6 k; 30k
	Before optimization	3	5	8
	Optimization I	2	3	5
	Optimization II	1.5	1.5	3.5
Editor	Thresholds (WCOX, WCS)	4 k; 30 k	8 k; 60 k	16 k;120 k
	Before Optimization	9	12	16
	Optimization I	5	6	10
	Optimization II	2	3	6
Viewer	Thresholds (WCOX, WCS)	5 k; 50 k	10 k;100 k	20 k;200 k
	Before optimization	12	22	28
	Optimization I	8	10	16
	Optimization II	4.5	5	7
UI	Thresholds (WCOX, WCS)	4 k; 30 k	8 k; 60 k	16 k;120 k
	Before optimization	15	20	24
	Optimization I	10	13	16
	Optimization II	5	7	9
Jdt	Thresholds (WCOX, WCS)	5 k; 50 k	10 k;100 k	20 k;200 k
	Before optimization	30	39	46
	Optimization I	16	21	25
	Optimization II	10	13	16

CONCLUSIONS

To effectively extract component information of the system from source code, we propose a component extraction approach based on WCS metrics. This approach takes classes as the smallest units in component classification, takes WCS as measures of connectivity between components, clusters classes with high WCS into a component through hierarchical clustering algorithm, so as to reduce the coupling of components in the system and increase component cohesion. It is proved in our experiment that this approach can better reflect the difference of user-defined classes and the precision of component extraction from source code can be as high as 70%; moreover, we propose optimizations for clustering algorithm and WCS computation, which improves the efficiency of component identification by 3 to 5 times.

This approach still needs further improvement. Such as, it does not take into account the concept of software layer, certain classes in foundation libraries of the system may be classified together with business layer classes because of their close relation and therefore it may have an impact on the layer structure of the system; besides, certain thresholds in the extraction process shall be given by experienced engineers to have expected effect; the effect in the selection of center points during the process of clustering is not ideal and so forth. Further study is needed to address these problems.

REFERENCES

Allen, E.B. and T.M. Khoshgoftaar, 1999. Measuring coupling and cohesion: An information-theory approach. In: Proceeding of the 6th International Software Metrics Symposium, Florida, USA., pp: 119-127.

Chen, B., C.T. Phang, R. Harrison and Y. Pan, 2005. Novel hybrid hierarchical K-means clustering method (H-K-means) for microarray analysis. Proceeding of the 2005. IEEE Computational Sytems Bioinformatics Coference Workshops (CSBW), pp: 105-108.

Chen, T., T. Tzu-Hsin, Y. Tzu Chen, L. Chin-Chiang, C. Rong-Chang, L. Huan-Yow and C. Hsin-Yi, 2005. A combined K-MEANS and hierarchical clustering method for improving the clustering efficiency of microarray. Proceedings of 2005 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS), pp: 405-408.

Cho, E.S., M.S. Kim and S.D. Kim, 2001. Component metrics to measure component quality. Software engineering conference. APSEC 2001. 8th Asia-Pacific, pp: 419-426.

Holz, E. and O. Kath, 2001. Manufacturing Software Components from Object-Oriented Design Models. In: Proceedings of 5th IEEE International Enterprise Distributed Object Computing Conference, pp: 262-272.

Huber, F., A. Rausch and B. Rumpe, 1998. Modeling dynamic component interfaces. In: Proceedings of Technology of Object-Oriented Languages, pp: 58-70.

Jain, H., 2001. Business component identification. A formal approach. In: Proceedings of 5th IEEE International Enterprise Distributed Object Computing Conference, pp: 183-187.

Lee, E., B. Lee, W. Shin and C. Wu, 2003. A Reengineering Process for Migrating from an Object-oriented Legacy System to a Component-based System. In: Proceedings of the 27th International Computer Software and Applications Conference (COMPSAC) (Dallas, TX, USA, Nov 3-6). IEEE. Comput. Soc. Press, pp: 336-341.

Lee, E., W. SHIN, B. LEE and C. WU, 2005. Extracting components from object-oriented system: A Transformational Approach. IEICE. Trans. Inform. Syst. E88-D(6), pp: 1178-1190.

Meng, F., Z. Den-Chen and X. Xiao-Fei, 2005. Business component identification of enterprise information system: A hierarchical clustering method. Proceeding of the 2005. IEEE International Conference on e-Business Engineering (ICEBE, 2005), pp: 473-480.

Saeed, M., O. Maqbool, H.A. Babri, S.Z. Hassan and S.M. Sarwar, 2003. Software clustering techniques and the use of combined algorithm. In: Proceedings of the 7th European Conference on Software Maintenance and Reengineering (CSMR) (Benevento, Italy, Mar 26-28). IEEE Comput. Soc. Press, pp: 301-306.

Washizaki, H. and Y. Fukazawa, 2005. A technique for automatic component extraction from object-oriented programs by refactoring. Sci. Comput. Programming, 56: 99-116.

Wiggerts, T.A., 1997. Using clustering algorithms in legacy systems remodularization. In: Proceedings of the 4th Working Conference on Reverse Engineering (WCRE) (Amsterdam, Netherlands. IEEE. Comput. Soc. Press, pp: 33-43.

Xu, B., C. Zhengqiang and Z. Yuming, 2002. Measuring class cohesion based on dependence analysis. Proceedings of International Conference on Software Maintenance, pp: 377-384.