Using Clustering Technique to Restructure Programs

Chung-Horng Lung

Dept. of Systems and Computer Engineering

Carleton University, Ottawa, Ontario, Canada

chlung@sce.carleton.ca

Marzia Zaman

Cistel Technology

Ottawa, Ontario, Canada

marzia@cistel.ca

Abstract

Program restructuring or refactoring is often required when a function becomes too large or is involved in multiple activities and therefore exhibits low cohesion. A critical factor in restructuring is to increase cohesion and decrease coupling. There are many existing methods which measure cohesion and coupling but do not provide much information as to how to restructure the program, while some other methods exist which only deals with restructuring the functions. The paper presents a simple but effective approach to function restructuring based on the experimental research on cohesion and coupling. Measure for software cohesion could be the first step of reengineering a software system to identify the functions with low cohesion. The next step is to restructure the identified functions. A clustering technique is presented in this paper which can assess the cohesiveness of a function and also gives indicaiton as to how to decompose a function to multiple high-cohesive functions. Several examples are presented to demonstrate the concept.

1. Introduction

Software inevitably evolves over time due to changes in technologies and requirements. As a result, a program may become large or complex, and consist of multiple functions or features. Consequently, the program is difficult to maintain. Program restructuring or refactoring [16] can be used to transform such programs or poorly-designed ones to another form that is better organized and easier to understand, without changing the behavior of the programs. The new software will usually be smaller, easier to change, and less costly for further evolution. More importantly, a desirable restructuring should achieve high cohesion and low coupling, so that all the elements in one component are closely related for the realization of a certain function, and changes made to that component will have as little impact as possible on other components.

Cohesion refers to a component's internal strength, that is, the strength that holds the internal elements in a component together to perform a certain functionality. While cohesion is an intra-component property, coupling measures the interdependence among components. Alexander [1] also postulated that the major design principle which is common to all engineering disciplines is

the relative isolation of one component from other components. Partitioning plays a crucial role in system design and is highly related to cohesion and coupling. In fact, the most important partitioning heuristics is to minimize external coupling and maximize internal cohesion [28]. Partitioning is to decompose a system into lower level components from the top down. Clustering, on the other hand, is a bottom-up method. With clustering, similar components are grouped together to form clusters or subsystems. Those clusters or subsystems are partitions which consistitute a system. The main objective of partitioning and clustering is the same..

Clustering techniques have been used successfully in many areas to assist grouping of similar components and support partitioning of a system. The technique can facilitate better understanding of the observations and the subsequent construction of complex knowledge structures from features and component clusters. The key concept of clustering is to group similar things together to form a set of clusters, such that intra-cluster similarity (cohesion) is high and inter-cluster (coupling) similarity is low. The objective – high cohesion and low coupling – is similar in software design. In fact, clustering of components has been discussed extensively in the software engineering literature to support software restructuring and reengineering [4,5,10,12,14,16,22,23,25,26,32,33,34, 36]

Lung et al, [27] borrow some clustering ideas from established disciplines, and tailor them to support software architecture partitioning, recovery, and restructuring. The approach is based on numerical taxonomy or agglomerative hierarchy. There are several reasons for adopting numerical taxonomy. First, the method is conceptually and mathematically simple, as will be demonstrated in Section 2. Secondly, although the concept is simple, no scientific study has shown that numerical taxonomy is inferior to other more complex multiversity methods [31]. Thirdly, the approach can be applied to various levels of abstraction (architecture, design, and maintenance) and can be used in round-trip engineering (e.g., both forward engineering and reverse engineering).

This paper focusses on how clustering technique can be used to measure the cohesiveness of a program as well as to assist the restructuring activity at the function level by dividing a large or low-cohesive program into cohesive functions. The approach adapts our previous work on software clustering to functions. As stated above, a

program or a function often becomes loosly cohesive over time. As a result, the system becomes difficult to maintain, understand, and/or change. Restructuring, although, could be very challenging and difficult, may be unavoidable.

The work presented in this paper does not attempt to automatically restructure a function. As reported in [10], there are problems assoicated with automatic restruturers. It is also the authors' belief that there is risk using automatic restructurers. A program, if incorrectly translated into a different representation, will create more problems than the original form. There are cases where low-cohesion might be required to meet specific requirements, e.g., performance. The approach presented in this paper is used to provide some indications as to (i) whether a function exhibits low cohesion; and (ii) if it does, how to improve the function by restructuring and dividing it into cohesive functions. We have developed a prototype tool which will aid in the process of restructuring. However, designer's involvement in this process is recommended to validate the results from the tool and make the final decision.

The rest of paper is organized as follows: Section 2 presents an overview of the clustering technique and discusses the method adopted for this research and the rationale behind it. Section 3 demonstrates how this technique relates to the standard cohesion types as suggested in [35]. Section 4 discusses the implemention of the method and experimental observations. Section 5 highlights some related work. Finally, section 6 presents the summary and discusses future directions.

2. Clustering

This section first briefly describes the general concept behind the numerical taxonomy clustering. Following that, we will discuss the method adopted in this research.

2.1 General Clustering Concepts

Many clustering methods have been presented [2,15,31,36] and many applications of clustering analysis can be found in various disciplines. In this paper, we focus on numerical taxonomy or agglomerative hierarchical approaches. Those approaches comprise the following three common key steps:
- Obtain the data set.
- Compute the resemblance coefficients.
- Execute the clustering method.

An input data set is a component-attribute data matrix. Components are the entities that we want to group based on their similarities. Attributes are the properties of the components. For example, the components could be software modules; the attributes, a set of features.

In this paper, a resemblance coefficient for a given pair of components indicates the degree of similarity between these two components. For instance, the data may be

represented by means of a binary variable. A 1 value indicates that the component has the property. A resemblance coefficient could be qualitative or quantitative. The simplest form of qualitative value is binary representation; e.g., the value is either 0 or 1. Qualitative attributes can also be multistate such as red/blue/green. A quantitative coefficient measures the literal distance between two components when they are viewed as points in a two-dimensional array formed by the input attributes.

There are many different methods to calculate the resemblance coefficients. This paper does not discuss those in detail. Rather, we briefly illustrate one algorithm that is closely related to our work. The algorithm is based on qualitative input data. Table 1 shows three components with eight attributes. A 1 entry indicates that the attribute is present in the corresponding component and 0 means that it is absent. Component x in Table 1 consists of attributes 1, 3, 4, and 8; component y is positive to attributes 1, 2, 3, and 7. Components x and y share two common attributes 1 and 3, or these two components have two 1-1 matches. In other words, a 1-1 match means that the same attribute is coded 1 for both components. Similarly, there are 1-0, 0-1, and 0-0 attribute matches between two components. Let a, b, c, and d represent the number of 1-1, 1-0, 0-1, and 0-0 matches between two components.

Table 1 Input Data Matrix: an Illustration

attri comp	1	2	3	4	5	6	7	8
x	1	0	1	1	0	0	0	1
y	1	1	1	0	0	0	1	0
z	0	1	1	0	1	0	1	0

Therefore, based on the definition, we obtain for components x and y that a = 2, b = 2, c = 2, and d = 2. Similarly, for components x and z, we obtain that a = 1, b = 3, c = 3, and d = 1; components y and z, a = 3, b = 1, c = 1, and d = 3.

To ascertain the similiarity between two components, we calculate the proportion of relevant matches between the two components. In other words, the more relevant matches there are between two components, the more similar the two components are. There are different methods to count relevant matches and there exist many algorithms to calculate the similarity. Some heuristics are presented as to how to choose a particular algorighm [31]. Here, we only illustrate the Sorenson algorithm. Let c_{xy} be the resemblance coefficient for components x and y. Sorrenson coefficient is defined as $c_{xy} = 2a / (2a + b + c)$. Note that d is not used in the fomula.

By applying the Sorenson matching coefficient to the example in Table 1, we get $c_{xy} = (2 \times 2) / (2 \times 2 + 2 + 2) = 1/2$.

Likewise, $c_{xz} = 1/4$ and $c_{yz} = 3/4$. This procedure is repeated for each component pair in order to obtain the resemblance matrix. For this particular data representation, the higher a coefficient, the more similar the two corresponding components represent. Hence, components y and z in this example are the most similar pair, since the resemblance coefficient c_{yz} is the largest.

Given a resemblance matrix, calculated from either quantitative or qualitative data, a clustering method is then used to group similar components. In essence, a clustering method is a sequence of operations that incrementally groups similar components into clusters. The sequence begins with each component in a separate cluster. At each step, the two clusters that are closest to each other are merged and the number of clusters is reduced by one. Once these two clusters have been merged, the resemblance coefficients between the newly formed cluster and the rest of the clusters are updated to reflect their closeness to the new cluster. An algorithm called UPGMA (unweighted pair-group method using arithmetic averages) [31] is commonly used to find the average of the resemblance coefficients when two clusters are merged.

For the example shown in Table 1, components y and z are first grouped as a new cluster (y, z), since c_{yz} is the largest resemblance coefficient. Recall that c_{xy} and c_{xz} are 1/2 and 1/4, respectively. The resemblance coefficient between the new cluster (y, z) and component or cluster x is then the average of c_{xy} and c_{xz}, which is $(1/2 + 1/4) / 2 = 3/8$. The process repeats until all clusters are exhausted or a pre-defined threshold value has been reached.

Figure 1 illustrates the concept. In this example, the clustering steps are (a, c), (b, d), ((a, c), e), and finally ((a, c, e), (b, d)). The dendrogram grasps the relative degree of similarity among components or clusters. In general, the lower the level, the more similar the components or clusters.
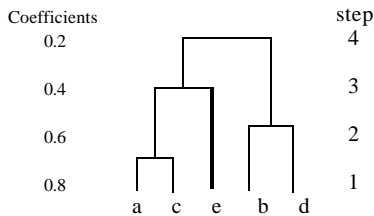

Figure 1. An Example of a Dendrogram

2.2 Input Data Set and Resemblance Coeffcient for Function Restructuring

This paper adopts the concept of qualitative clustering, as described above, and tailors it to accommodate different types of dependance appeared in software. The input data is obtained by parsing a program at the function level, and mapping of the logical statements in that function and its data dependance and/or scope in that function. The idea will be explained in detail later in this section.

We have started with the Sorenson coefficient for software clustering [25,26]. Although the Sorenson method was also successfully used to classify a number of simulation models into a set of generic models [24,27], we have made some changes to the calculation for restructuring purpose. Specifically, statements are viewed as components and variables are considered as attributes. Also, scope of variables (i.e., if, else, loop) are also taken into account from the control aspect. They were treated like variables but with less weight. The input data set is then a representation of statement-variable/scope data matrix. Statements are the entities that we want to group based on their similarities in terms of their variable uses (data dependance) as well as the scope of their useages. All statements are considered with few exceptions, e.g., statements where a local variable is incremented (e.g., a statement like i++) in a loop are igonred.

Modifiction to the Sorenson method is conducted to accommodate different possible association of function statement with the data and control. Therefore, multistate qualitative data (4/3/2/1/0) are used for input, where each value represents a different type of cohesion or association. A value of 4 or 3 represents direct data dependency, but 4 is adopted when a variable is updated, whereas 3 is chosen when a variable is only used. For instance, a 4 value will be entered for varialbe *sum*, but a 3 value is used for variable *num*, in the statement: *sum = sum + num*. A 2 value represents an indirect control dependancy of a statement on a variable. For example, a statement under a loop is dependent on the upper limit of the loop. Finally, a 1 value represents control dependency in a statement being under a decision making block (if/else block) or inside an iterative loop (for, while, do etc.).

We have chosen five different types of associations or matches (a, b, c, d, e, f) and give them different weights when calcualting the coefficients. The matches between two components are defined as: a is the total numbers of 4-4 matches; b, total numbers of 4-3 and 3-3 matches; c, total number of 2-2 matches; d, the total number 1-1 matches, e, total number of 4-2, 2-4, 3-2 or 2-3 matches; and f, total number of mismatches (i.e., patterns 4-0, 0-4, 3-0, 0-3, 2-0, 0-2, 1-0, 0-1) between two statements. Higher weights are given for direct data dependency and lower weights are used for indirect control dependency.

The resemblance cofficient is modified as follows:

$$(w_a a + w_b b + w_c c + w_d d + w_e e)/(w_a a + w_b b + w_c c + w_d d + w_e e + f)$$

where w_a, w_b, w_c, w_d are the weights. The weights used in this paper are primarily obtained from experiments. Specially, $w_a = 16$, $w_b = 8$, $w_c = 2$, $w_d = 1$, and $w_e = 2$.

Once the input data matrix is generated and the initial resemblance coefficients are calculated, the clustering method is used as mentioned in the previous section. As

the clustering progresses, the statements of a function are grouped together and the resemblance coefficients are re-calculated in each step. The coefficients will drop as the clustering progresses. A considerable drop would be an indication as to whether there is multiple functionalities within a single function. The process will also create several groups of statements where each group of statements would generally map to a new function. The result of such method when combined with designer's input can produce effective restructuring of a function.

3. Cohesion Types and Resemblance Coefficients

This section demonstrates the concept with examples based on cohesion types [35]. This paper considers each statement in a function as the processing element. The dependence relation of a statement is obtained from its variables and the scope within a module. The "statement" level is chosen because our final goal is to provide further retructuring options as we measure cohesion. Using statements also reduces the chance of introducing errors. In addition, the approach presented here for gathering the input data does not involve generating the dependance graph, which could be complex and time consuming. Rather, a simple script is used to parse the module of interest to get the information required. The entire process is simple, repititive and easily modifiable in case the weights of the association types need to be changed.

In order to validate the effectiveness of our clustering technique, we use the sample programs which depict the different cohesion types from [20] and obtain the input data matrix to be used for clustering. The example code for coincidental, logical, procedural, and communicational cohesion types can be easily decomposed into two cohesive functions. For sequential and functional cohesion types, the result depits one strongly related cluster. Due to the page limit, only some of the cohesion type and the results are presented in Figures 2 and 3.

The example in Figure 2 shows the computation of the sum of first m numbers if the flag is true, else the product of first n numbers. In the input data shown in Figure 2(b), artifical variables, such as *if* and *loop*, are introduced to represent association due to the control dependency. For instance, both lines 6 and 8 have a 1 entry for if (column 2), since both of them belong to the same *if-then* block. Line 8 has a 2 entry under variable *m*, since *sum* is indirectly associated with *m* due to the loop control dependency. Note also that the input shown in Figure 2(b) does not contain variables (e.g., i in lines 5, 9, 13, ad 17) that are used as loop count, neither does it have a row entry for the *if*, *while* or *for* statements (e.g., lines 4, 7, and 15).

It is shown in Figure 2(c) that lines 6 and 8 have a high cohesion value (0.89); similarly lines 14 and 16 are closely related. However, these two clusters, (6,8) and (14,16), are

independent (the associated resemblance coefficient is 0). This also explains why the loop variables are not considered in the clustering. If the second while loop also uses variable *i* as the counting variable, then these two while loops will reveal some false resemblance.

```
1 procedure sum_or_product (m,n,flag:
  integer; var sum,prod: integer);
2 var i,j: integer;
3 begin
4  if flag = 1 then begin
5     i := 1;
6     sum := 0;
7     while i <= m do begin
8        sum := sum + i;
9        i := i + 1
10    end
11 end
12 else begin
13    j := 1;
14    prod := 1;
15    while j <= n do begin
16       prod := prod * j;
17       j := j + 1
18    end
19 end
20 end
```

(a)

line#	if	loop 1	m	else	loop 2	n	sum	prod
6	1	0	0	0	0	0	4	0
8	1	1	2	0	0	0	4	0
14	0	0	0	1	0	0	0	4
16	0	0	0	1	1	2	0	4

(b)


```
0
0.89
0.89
         6  8  14  16
```
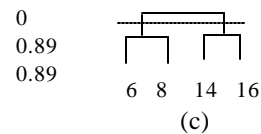(c)

Figure 2. (a) A module computing the sum of first m numbers or product of first n numbers; (b) input data matrix; (c) dendrogram and resemblance coefficients. The module represents logical cohesion.

We have applied the technique to all the examples in [20] and the results are encouraging but are not presented here due to page limit. Figure 3 shows another sample of non-cohesive code [23]. For simplicity, the input data matrix is not presented.

```
  1 procedure sale_pay_profit (days: integer;
cost:float; var sale: int_array; var pay:
float; var profit: float; process: Boolean);
  2  var i: integer; total_sale, total_pay:
float;
  3  begin
  4  i := 0;
  5  while i < days do begin
  6     i := i + 1;
  7     readln (sale[i]);
  8  end;
  9  if process = True then begin
 10     total_sale := 0;
 11     total_pay := 0;
 12     for i := 1 to days do begin
 13       total_sale:=total_sale + sale[i];
 14       total_pay:=total_pay+0.1*sale[i];
 15       if (sale[i] > 1000) then
 16         total_pay := total_pay + 50;
 17     end;
 18     pay := total_pay / days + 100;
 19     profit := 0.9 * total_sale – cost;
 20 end;
 21 end;
```
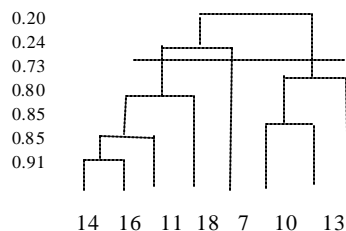
```
    total_pay := 0;
    for i := 1 to days do begin
      total_pay := total_pay + 0.1 * sale[i];
      if (sale[i]) > 1000) then
        total_pay := total_pay + 50;
    end;
    pay := total_pay / days + 100;
  end;
```

Figure 4. Restructured program.

4. Implementation and Experiments

A prototype tool was built for concept evaluation and demonstration. The tool consists of three modules. Module 1 is a parser for C programs written in Perl, which generates the input data matrix. Module 2 performs the clustering. Module 3 draws the dendrogram.

This approach is primarily experimental research. The multistate values and weights were chosen based on experiments on various examples. The main idea is to devise a sheme that distinguishes different association types using different weights. The results were encouraging, as demonstrated in the previous section. We have applied the approach to many functions of a C parser and a telecommunication system developed in industry, and some concurrent programs written by students. Generally, we have achieved our goal – transforming low-cohesive functions into smaller and cohesive ones.

5. Related Work

Numerious papers have discussed cohesion and coupling at different levels (e.g., design vs. code) [6,7,11,13,17] or for different paradigms (procedural vs. OO languages) [9]. Most of those papers focus on measurement of cohesion and coupling. The idea of restructuring is to transform a system or a progrma into more organized form to support evolution.

Lakhotia and Deprez [22,23] provided a thorough discussion on related work. Readers are referred to read their papers for a review of this area. The objective of their approach is identical to ours. However, their restructuring technique computes pairwise cohesion between output variables of a function using rule-based measure of cohesion. Our approach is a variation of the numeric clustering technique. The method or tool can be easily adapted if the definition of the weighing policy is modified.

The clustering idea is also related to program slicing. Each column or a collection of columns of the input data matrix is similar to a program slice. The clustering algorithm is then used to find closeness among the slices.

6. Conclusions and Futurre Work

We presented a conceptually simple clustering technique and adapted it to program restructuring The result of the clustering can provide useful feedback to


```
0.20
0.24
0.73
0.80
0.85
0.85
0.91

       14  16  11  18  7   10   13
```

Figure 3. Sample non-cohesive code and the dendrogram output.

The dendrogram and the associated resemblance coefficients clearly show three groups: (14, 16, 11, 18), (10, 13, 19), and (7). The coefficient between (14,16,11,18) and (7) is very small, so they belong to two different clusters. In other words, the program can be decomposed into the three sub-functions based on those three clusters. The following shows those three procedures (for brevity, variable declarations are not shown here):

```
procedure read_input:        procedure compute_profit:
begin                        begin
  i := 0;                        total_sale := 0;
  while i < days do begin       for i := 1 to days do begin
    i := i + 1;                    total_sale := total_sale + sale[i];
    readln (sale[i]);          end;
  end;                           profit := 0.9 * total_sale – cost;
end;                         end;

    procedure compute_avg_pay:
    begin
```

convert a large or low-cohesive code segment into smaller and cohesive functions, without changing its behavior. A tool was developed to assist the parsing, clustering, and drawing. We demonstrated the concept with examples of different cohesion types. The concept can be easily tailored to meet other definitions or different calculations. The restructuring effort can be automated further, however, we advocate human intervention to be involved in the restructuring process to avoid errors. More experiments need to be conducted to further validate the approach for large and complex software systems.

References

[1] Alexander, C., *Notes on the Synthesis of Form*, Harvard University Press, Cambridge, MA, 1964.

[2] Anderberg, M.R., *Cluster Analysis for Applications*, Academic Press, New York, NY, 1973.

[3] Anquetil, N., Lethbridge, T., "Extracting concepts from file names: a new file clustering criterion", *Proc. of International Conf. on Software Eng.*, 1973, pp. 84-93.

[4] Anquetil, N., Lethbridge, T., "Experiments with clustering as a software modularization", *Proc. of the 6th Working Conf on Reverse Engineering*, 2000, pp. 235-255.

[5] Arnold, R.S, *Software Reengineering*, IEEE Computer Society Press, Los Alamitos, California, 1993.

[6] Bieman, J.M., Ott, L, "Measuring functional cohesion", *IEEE Trans. Sw. Eng.* 20 (8), 1994, pp. 644-657.

[7] Bieman, J.M., Kang, B.-K., "Measuring design-level cohesion", *IEEE Trans. Sw Eng*, 24 (2), 1998, pp. 111-124.

[8] Briand, L., Morasca, S., Basili, "Property-based software engineering measuremen", *IEEE Trans. on Sw. Eng*, 22, (1), 1996, pp. 68-86.

[9] Briand, L., Devanbu, P., Melo, W., "An investigation into coupling measures in C++", *Proc. of Int'l Conf on Sw. Eng*, 1997, pp. 412-421.

[10] Calliss, F.W., "Problems with automatic restructurers", *SIGPLAN Notices* 23 (3), 1988, pp. 13-21.

[11] Card, D.N., Glass, R.L., *Measuring Software Design Quality*, Prentice Hall, Eaglewood Cliffs, NJ, 1990.

[12] Davey, J., Burd E., "Evaluating the suitability of data clustering for software remodularization", *Proc. of the 7th Working Conf. on Reverse Eng*, 2000, pp. 268-276.

[13] Dhama, H., "Quantitative models of cohesion and coupling in software", *J. of Sys & Sw* 29, 1995, pp. 65-74.

[14] Dromey, R.G., "Cornering the Chimera", *IEEE Software*, 1996, pp. 33-43.

[15] Everitt, B., *Cluster Analysis*, Heinemann Educational Books, Ltd., London, 1980.

[16] Fowler, M., et al., *Refactoring: Improving the Design of Existing Code*, Addison Wesley, 1999.

[17] Heyliger, G., "Coupling", In *Encyclopedia of Software Eng*, J. Marciniak (ed.), 1994.

[18] Hutchens, D., Hutchens, Basili, V.R., "System structure analysis: clustering with data bindings" *IEEE Trans. Sw Eng* 11 (8), 1985, pp. 749-757.

[19] Kim, H. S., Kwon, Y. R., Chung, I. S., "Restructuring programs through program slicing", *Int'l J of Sw Eng and Knowledge Eng* 4 (3), 1994, pp. 349-368.

[20] Lakhotia, A., "Rule-based approach to computing module cohesion", *Proc. of the 15th Int'l Conf on Software Eng*, 1993, pp.35-44.

[21] Lakhotia, A., "A unified framework for expressing software subsystem classification techniques", *J. of Systems and Sw* 36, 1997, pp. 211-231.

[22] Lakhotia, A., Deprez, J, "Restructuring programs by tucking statements into functions" *J. of Info and Sw Technology* 40 (11-12), 1998, pp. 677-689.

[23] Lakhotia, A., Deprez, J., "Restructuring functions by low cohesion", *Working Conf on Reverse Eng*, 1999.

[24] Lung, C.-H., Cochran, J.K., Mackulak, G.T., Urban, J.E., "Computer simulation software reuse by Generic/Specific domain modeling approach", *International Journal of Sw Eng and Knowledge E*ng 4 (1), 1994, pp. 81-102.

[25] Lung, C.-H., "Software architecture recovery and restructuring through clustering techniques", *Proc. of the 3rd Int'l Workshop on Sw Architecture*, 1998, pp. 101-104.

[26] Lung, C-H, Zaman, Z., A. Nandi, "Applying clustering techniques to software architecture partitioning, recovery and restructuring", to appear *J. of Sys and Sw*.

[27] Mackulak, G.T., Cochran, J.K., "Generic/Specific modeling: an Improvement to CIM simulation techniques. in *Optimization of Manufacturing Systems Design*, D. Shunk (ed.), Elsevier Science Publishers, '90, pp. 331-363.

[28] Maier, M.W, Rechtin, E., *The Art of Systems Architecting*, 2nd edition. CRC Press, 2000.

[29] Mancoridis, S., Mitchell, B.S., Rorres, C, Chen, Y., Gansner, E.R., "Using automatic clustering to produce high-level system organizations of source code", *Proc. of Int'l Workshop on Program Comprehension, 2001.*

[30] Patel, S., Chu, W., R., Baxer, X, "A measure for composite module cohesion", *Proc. of Int'l Conf. on Sw Eng*, 1992, pp. 38-48.

[31] Romesburg, H. C., *Cluster Analysis for Researchers*. Krieger, Malabar, Florida, 1990.

[32] Sartipi, K., Kontogiannis K., "Component clustering based on maximal association", *Proc. of the 8th Working Conf. on Reverse Eng,* 2001, pp. 103-114.

[33] Selby, R.W., Reimer, R.M., Interconnectivity analysis for large software systems, *Technical Report, (UCIrv-95-PROC-CSS-001)*, 1995, Univ. of California at Irvine.

[34] Snelting, G., "Software reengineeirng based on concept analysis", *Proc. of the European Conf. on Software Maintenance and Reeng.*, 2000, pp. 1-8.

[35] Stevens, W.P., Myers, G.J., Constantine, L.L., *Sturctured design, IBM Sys J*. 13 (2), 1974, pp. 115-139.

[36] Wiggerts, T.A., "Using clustering algorithms in legacy systems modularization", *Proc. of the Working Conf. on Reverse Eng*, 1997, pp. 33-43.