International Journal of Software Engineering
and Knowledge Engineering
Vol. 14, No. 4 (2004) 429–439
© World Scientific Publishing Company

APPLICATION OF CLUSTERING TECHNIQUES TO SOFTWARE COMPONENT ARCHITECTURE DESIGN

SHU-CHUAN LO* and JIA-HUI CHANG

Department of Industrial Engineering and Management,
National Taipei University of Technology, Taipei, Taiwan 10643, R.O.C.
**sclo@ntut.edu.tw*

This paper describes the application of two variants of clustering techniques to a matrix of requirements (use cases) vs. design units (classes), with the aim of augmenting the design of software component architectures. One algorithm used a modified COMO matrix-clustering algorithm, while the other is average-linkage agglomerative clustering based on similarity among classes as a measure of distance. Our results provide a shortcut for software engineers with limited experience in Component-Based Software Development (CBSD) who are constructing component architectures. This paper also evaluates the correlations between these two clustered results. We solicited feedback from experienced software developers who confirmed that the inferred hierarchies conformed to their conceptualizations of the software structure.

Keywords: Clustering analysis; software component; CBSD; object-oriented programing.

1. Introduction

Component-based software engineering is now widely accepted by software designers. Components follow the object principle of combining functions and related data into a single unit. The major challenge in component-based system development is managing change. Many people think the primary rationale of components is re-use. Programmers tend to design software that can be developed once and then re-used in different contexts. As a result, productivity can be increased, best-in-class solutions can be obtained and quality can be improved. These are admirable objectives, but the principal stimulus today is that conditions keep changing, and often — as with business-to-business electronic commerce — there is no longer any hope that centralized control can be exercised. In such an environment, one of the primary characteristics of a component is that it must be easily *replaceable* — either by a completely different implementation of the functions or by an upgraded version of the current implementation. In this paper, rather than seeking to ensure that individual components are reusable by multiple component systems, we focus on system architecture and management, as various components evolve and requirements change.

Currently, component architectures for component development (such as EJB and COM+) are generating tremendous interest in the software community. Some systematic development processes and detailed guidelines for building OO components have also been proposed, such as Catalysis [4], Advisor [1], UML Component [3] and COMO [8]. Engineers using these model-based approaches in the design and construction of enterprise-scale component-based software usually need intensive theoretical and practical training. Some engineers have learned EJB and COM+, but their component architectures are limited to class-size components because classes are their preferred design. If the architecture of components is based on the engineers' habits and preferences, then the time span from requirement to design must be shortened and improved. This is the main objective of our study.

The concept of clustering has been applied in the software engineering field. Belady and Evangelisti [2] presented a method to perform automatic clustering of a large number of program modules and data structures with the objective of reducing complexity. Hutchens and Basili [5] presented the use of cluster analysis as a tool for software partition. Ince and Shepperd [6] presented the use of system design metrics to monitor maintenance changes based on a single-link clustering algorithm. Selby and Basili [10] applied a clustering technique to data bindings to generate a hierarchical system description from source code. Raz and Yaung [9] applied two variations of agglomerative clustering to the analysis of planning data for an organization-wide information system design project. The above approaches demonstrate that clustering is an effective and practical technique for system structure analysis in the software life cycle.

In this paper, we describe the application of two variations of clustering algorithms to component structure analysis. One is modified from the COMO algorithm [8], the other is average linkage — an agglomerative hierarchical method. In addition, we focus on their application to a realistic situation and evaluate the results they support. Following a background discussion, this paper is organized into three major sections: examination of the clustering techniques developed here (Secs. 3 and 4), and analysis of the relationships between the two sets of clusters (Sec. 5). The results are compared with experienced programmers' comments. Following these three sections, we provide concluding remarks.

2. Background

In this research, we present a medium-scale program — a music web system — for our case study. There are five subsystems in this system. The system architecture is shown in Fig. 1. We study the MakingFriends subsystem to identify components by clustering methods. Six use cases were analyzed from the MakingFriends subsystem, shown in Fig. 2 Use Case Diagram. After a series of discussions with developers of this web system, we identified ten classes that are related to these six use cases. Figure 3 is the Class Diagram of the MakingFriends subsystem. It is convenient to assign an ID for each class name and use case names in clustering

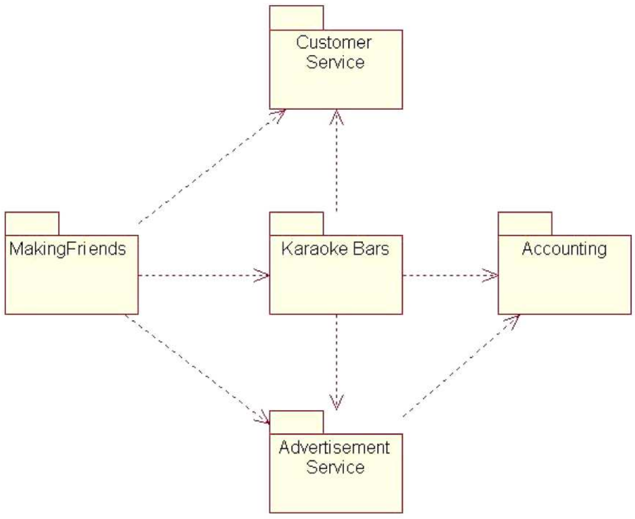


Fig. 1. System architecture of music web case.

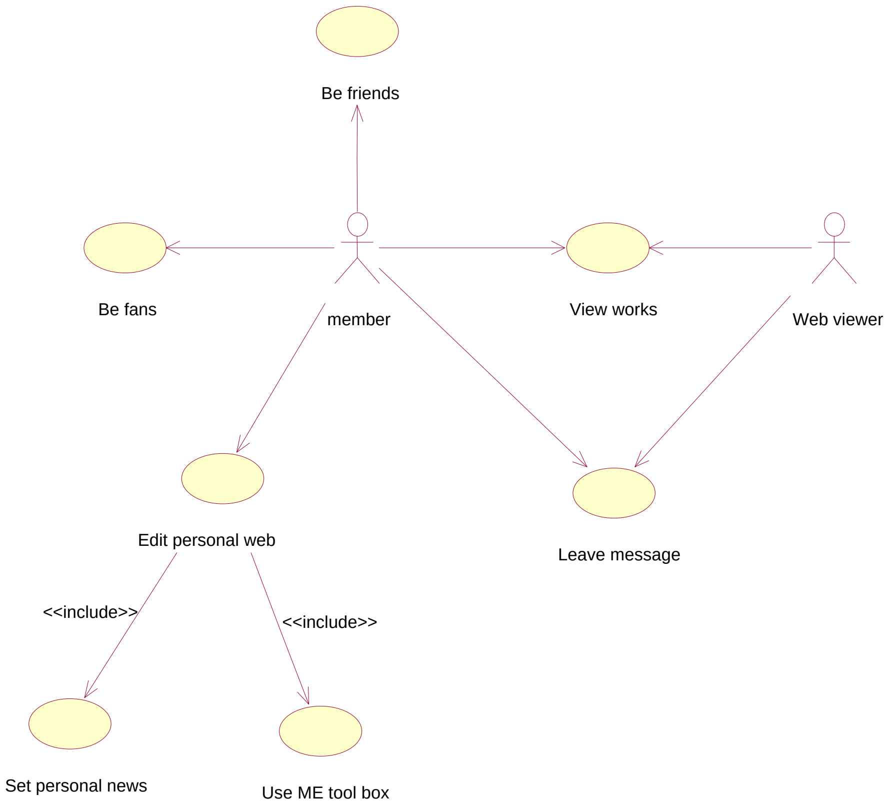


Fig. 2. Use case diagram for the MakingFriends subsystem.

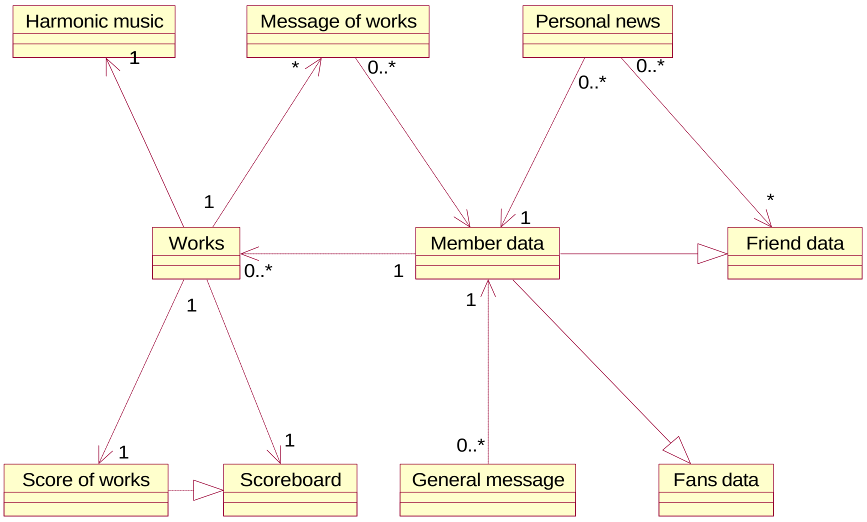


Fig. 3. Class diagram for the MakingFriends subsystem.

Table 1. ID table for the MakingFriends subsystem.

ID	Name
Class 1	Message of works
Class 2	Score of works
Class 3	Scoreboard
Class 4	General message
Class 5	Member data
Class 6	Friend data
Class 7	Fans data
Class 8	Works
Class 9	Personal news
Class 10	Harmonic music
Use Case 1	Be friends
Use Case 2	Be fans
Use Case 3	View works
Use Case 4	Leave message
Use Case 5	Use ME tool box
Use Case 6	Set personal news

operations. We present an ID table for the MakingFriends subsystem in Table 1.

We assume a highly cohesive component related to a group of classes that have similar data usage characteristics. We use a use-case/class matrix to present the relationship between requirement and design. Based on this assumption we attempt to group the highly similar classes into a component. Before clustering, in order to describe the relationship of use cases and classes in the matrix, we defined five types:

Table 2. Original use-case/class matrix.

Class / Use Case	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Class 10
Use Case 1	N	N	N	N	W	C	W	N	N	N
Use Case 2	N	N	N	N	W	W	C	N	N	N
Use Case 3	C	C	C	R	R	W	W	R	N	R
Use Case 4	N	N	N	C	W	W	W	R	N	N
Use Case 5	R	R	R	N	W	R	R	C	N	C
Use Case 6	N	N	N	N	W	R	R	N	C	N

None (N): use case did not use any data from the class
Read (R): use case reads data from the class
Write (W): use case updates and reads data from the class
Delete (D): use case deletes and reads data from the class
Create (C): use case creates and reads data from the class

The resulting information about the usage of data from each class by each use case is summarized in a two-dimensional association matrix, shown in Table 2.

3. Modified COMO Algorithm

Lee *et al.* [8] categorized the relationship between use case and class into four types: "Create", "Delete", "Write", and "Read". They also assigned priority to the relationship types as: "Create > Delete > Write > Read". We add a new type, "None", in the relationship of the use-case/class matrix. The priority of the relationship types is defined as: "Create > Delete > Write > Read > None". The type "None" means no data usage between use case and class. To add this type accords with real situations better than the COMO model. The clustering algorithm is shown in Fig. 4.

4. Average Linkage Agglomerative Hierarchical Method

Hierarchical clustering techniques proceed by either a series of successive mergers or a series of successive divisions. Agglomerative hierarchical methods start with the individual objects. Thus, there are initially as many clusters as objects. The most similar objects are first grouped, and these initial groups are merged according to their similarities. Eventually, as the similarity decreases, all subgroups are fused into a single cluster.

Average linkage treats the distance between two clusters as the average distance between all pairs of items where one member of a pair belongs to each cluster. The input to the average-linkage algorithm is distances or similarities. The algorithm proceeds as shown in Fig. 5. We search the similarity matrix $\mathrm{D} = \{d_{ik}\}$ to find the nearest (most similar) objects (for example, U and V). These objects are merged to form the cluster (UV). For step 3 of the agglomerative algorithm, the distances between (UV) and any other cluster W are determined by: $d_{(UV)W} = \frac{\sum_i \sum_k d_{ik}}{N_{(UV)} N_w}$

```
int init_C, init_R, end_C, end_R=1      //Set up row 1 to be the initial given row.
int n = # of classes                    //Set up column 1 to be the initial given column.
int m = # of use cases
int rept = 1                            //Bound columns with 'C' to 'R' together for repetition 1.
 for ( rept = 1  rept  2   rept + + ) { //Bound columns with 'C' to 'W' together for repetition 2.
   while (init_C < n && init_R < m) {
     for ( I = end_C   I     n   I + + ) {
      if ((rept = = 1 && M ( init_R , I ) = = 'C'   'D'     'W' | 'R')
          (rept = = 2 && M ( init_R , I ) = = 'C'   'D'     'W')) {
              temp = Column ( end_C )
              Column ( end_C ) = Column ( I )
              Column ( I ) = temp
              end_C + +
              }                         //Swap prior columns together from column 1 to n
      }                                 //for a given row from initial row 1 to m
      for ( j = init_C   j < end_C   j + + ) {
        for ( k = end_R   k     m   k + + ) {
          if ((rept = = 1 && M ( init_R , I ) = = 'C'   'D'     'W' | 'R') |
              (rept = = 2 && M ( init_R , I ) = = 'C'   'D'     'W')) {
               temp = Row ( end_R )
              Row ( end_R ) = Row ( k )
              Row ( k ) = temp
              end_R + +
              }                         //Swap prior rows together from row 1 to m
        }                               //for a given column from initial column 1 to end-column
      }
     if ( init_C = = n ?   n   init_C + + )
     if (init_R = = m ?   m   init_R + + )
   }
}
 ( Note )
        init_C      Start location of column
        end_C       End location of column
        init_R      Start location of row
        end_R       End location of row
        C   "Create"   D   "Delete"   W   "Write"
        Column ( end_C )    Column of current end_C
        Row ( end_R )    Row of current end_R
```

Fig. 4. The modified clustering algorithm from the COMO model.

where d_{ik} is the distance between object i in the cluster (UV) and object k in the cluster W, and $N_{(UV)}$ and N_W are the number of items in clusters (UV) and W, respectively.

In this paper we use the similarity measure of Raz and Yaung [9] to redefine the use-case/class matrix. Different numerical weights were assigned to these five relationship types to reflect the intensity of usage: 0 for None, 1 for Read, and 2 for other types. The resulting matrix of the usage of data from each class by each use case is summarized in Table 3. Of the 60 cells in the matrix, 47% had the weight 0; 20% had the weight 1; 33% had the weight 2. The similarity measure is defined as follows: The similarity between two classes i and j is the ratio of the number of

1. Start with N clusters, each containing a single entity and an $N \times N$ symmeric matrix of distances (or similarities) $D = \{d_{ik}\}$.
2. Search the distance matrix for the nearest (most similar) pair of clusters. Let the distance between "most similar" clusters U and V be d_{UV}.
3. Merge clusters U and V. Label the newly formed cluster (UV). Update the entries in the distance matrix by: (a) deleting the rows and columns corresponding to clusters U and V, and (b) adding a row and column giving the distances between cluster (UV) and the remaining clusters.
4. Repeat steps 2 and 3 a total of $N - 1$ times. (All objects will be in a single cluster at termination of the algorithm.) Record the identity of clusters that are merged and the levels (distances or similarities) at which the mergers take place.

Fig. 5. The average linkage agglomerative hierarchical algorithm.

Table 3. The weight matrix of use cases and classes.

	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Class 10
Use Case 1	0	0	0	0	2	2	2	0	0	0
Use Case 2	0	0	0	0	2	2	2	0	0	0
Use Case 3	2	2	2	1	1	2	2	1	0	1
Use Case 4	0	0	0	2	2	2	2	1	0	0
Use Case 5	1	1	1	0	2	1	1	2	0	2
Use Case 6	0	0	0	0	2	1	1	0	2	0

use cases that both i and j relate to, and to the number of use cases that either i or j relate to.

$$SM(i,j) = \frac{\sum_{k=1}^{n} I(i,j)_k}{\sum_{k=1}^{n} U(i,j)_k} = \frac{\sum_{k=1}^{n} \text{Min}\{W_{ik}, W_{jk}\}}{\sum_{k=1}^{n} \text{Max}\{W_{ik}, W_{jk}\}}, \tag{1}$$

where W_{ik} is the weight of the relationship between class i and class k, $I(i,j)_k$ is the intersection of the relationship between class i and use case k and the relationship between class j and use case k and $I(i,j)_k = \text{Min}\{W_{ik}, W_{jk}\}$, such as $I(3,4)_3 = \text{Min}\{2,1\} = 1$. $U(i,j)_k$ is the union of the relationship between class i and use case k and the relationship between class j and use case k and $U(\{i,j\})_k = \text{Max}\{W_{ik}, W_{jk}\}$, such as $U(3,4) = \text{Max}(2,1) = 2$. The similarity of class 3 and class 4 is $(0+0+1+0+0+0)/(0+0+2+2+1+0) = 1/5 = 0.2$. The similarity matrix is shown in Table 4.

5. Results of Clustering Algorithms

In this section we present the results of the two clustering algorithms noted in Secs. 3 and 4, respectively. We applied the modified COMO clustering algorithm to the original use-case/class matrix. The clustered matrix is depicted in Table 5.

Table 4. The similarity matrix of classes.

	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Class 10
Class 1	1	1	1	0.2	0.17	0.3	0.30	0.4	0	0.5
Class 2	1	1	1	0.2	0.17	0.3	0.3	0.4	0	0.5
Class 3	1	1	1	0.2	0.17	0.3	0.3	0.4	0	0.5
Class 4	0.2	0.2	0.2	1	0.27	0.3	0.3	0.4	0	0.2
Class 5	0.17	0.17	0.17	0.27	1	0.75	0.75	0.45	0.18	0.27
Class 6	0.3	0.3	0.3	0.3	0.75	1	1	0.36	0.09	0.18
Class 7	0.3	0.3	0.3	0..3	0.75	1	1	0.27	0.09	0.18
Class 8	0.4	0.4	0.4	0.4	0.45	0.36	0.27	1	0	0.75
Class 9	0	0	0	0	0.18	0.09	0.09	0	1	0
Class 10	0.5	0.5	0.5	0.2	0.27	0.18	0.18	0.75	0	1

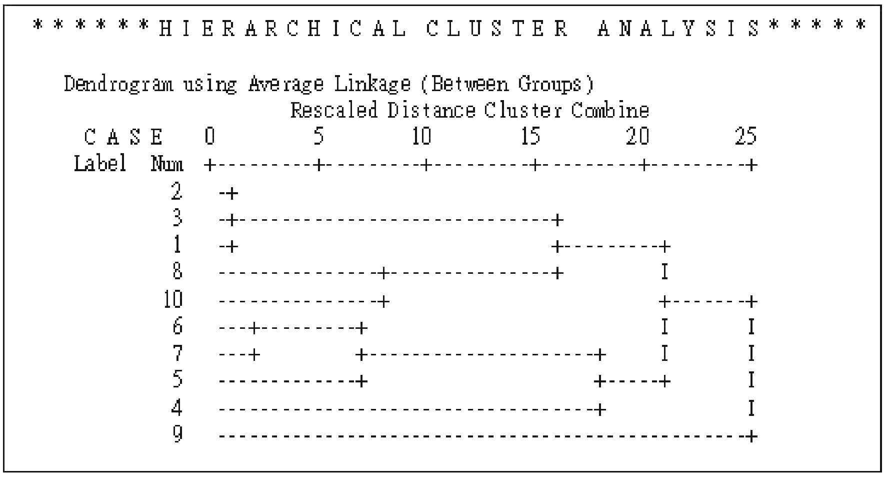
Table 5. The clustered use-case/class matrix.

	Class 5	Class 6	Class 7	Class 4	Class 8	Class 10	Class 9	Class 1	Class 3	Class 2
Use Case 1	W	C	W	N	N	N	N	N	N	N
Use Case 2	W	W	C	N	N	N	N	N	N	N
Use Case 4	W	W	W	C	R	N	N	N	N	N
Use Case 5	W	R	R	N	C	C	N	R	R	R
Use Case 6	W	R	R	N	N	N	C	N	N	N
Use Case 3	R	W	W	R	R	R	N	C	C	C

We used the well-known statistical package SPSS10.0 to execute the average-linkage clustering algorithm for the data shown in Table 4. The dendrogram of the similarity matrix is represented in Fig. 6. From Table 5 and Fig. 6 we can see that the clusters of the two algorithms are quite consistent. If we set the rescaled distance at approximately 15, then we can see that there are five clusters:

Cluster 1: class 1, class 2 and class 3
Cluster 2: class 8 and class 10
Cluster 3: class 5, class 6 and class 7
Cluster 4: class 4
Cluster 5: class 9

In our case these two algorithms yielded the same clusters. We distribute these clusters to components as shown in Table 6. We interviewed 18 OO programmers with approximately four years of experience about these clusters. Most of them are graduate students who learned OO programming in their first or second college year. We displayed the specifications of these ten classes and five components. We



```
* * * * * * H I E R A R C H I C A L   C L U S T E R   A N A L Y S I S * * * * * *

 Dendrogram using Average Linkage (Between Groups)
                     Rescaled Distance Cluster Combine
  C A S E     0         5        10        15        20        25
  Label  Num  +---------+---------+---------+---------+---------+
          2    -+
          3    -+-----------------------------+
          1    -+                             +---------+
          8    ---------------+---------------+         I
         10    ---------------+                         +-------+
          6    ---+---------+                           I       I
          7    ---+         +---------------------+     I       I
          5    -------------+                     +-----+       I
          4    -----------------------------------+             I
          9    -------------------------------------------------+
```

Fig. 6. Dendrogram of classes.

Table 6. Component clusters.

Component	Class
Works Management	Harmonic music, Works, Message of works
Friend Management	Friend data, Member data, Fans data
Evaluating Management	Score of works, Scoreboard
Personal News Management	Personal news
Message Management	General message

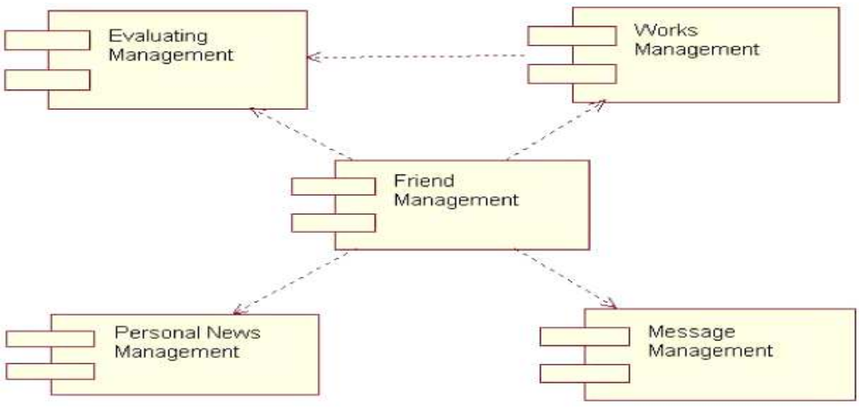


Fig. 7. Component architecture in the MakingFriends subsystem.

asked them to distribute classes into components. The result shows that 92% of their class distributions are the same as our clusters. Figure 7 is the component architecture for the MakingFriends subsystem from our clustering analysis.

Table 7. Comparison of clustering techniques.

Clustering Techniques Criteria	Modified COMO Algorithm	Average Linkage Hierarchical Method with Raz's Similarity Definition
Algorithm Automation	Matrix shift	Numerical Operation
Practicality	High	High
Extensity	Medium	High
Effectiveness	Good	Good

Both these two clustering algorithms have same basic ideas. That is, a cohesive component related to similar data usage classes. And their clustering weight policies are equivalent also. That is, "Create", "Delete" and "Write" have the same weight level, higher than "Read" and "None". We compare these two clustering algorithms based on the following criteria: Algorithm Automation, Practicality, Extensity, and Effectiveness. These are shown in Table 7 for this web program case. The clustering algorithm of the COMO model uses exchanges between columns and rows in use-case/class matrix to cluster classes together with similar weight of data usage. Matrix operation is less flexible than numerical operation when the matrix becomes large in complex case. Even the results of these two algorithms are identical to those of our experimental case. We believe the average linkage hierarchical method with Raz's similarity definition is more useful for complex cases. Both algorithms show good quality in this study but we need to study more cases to support their effectiveness.

6. Conclusion

The results of this study illustrate the value that clustering techniques can add to component architecture design. The two clustering algorithms applied to this web subsystem offer a new process from software requirement to component design, namely, a way to obtain component structure through an auto-clustering mechanism applied to the matrix of requirements (use cases) vs. design units (classes). These two clustering algorithms, based on intuitive principles, are easy to implement and require only modest computational resources. The assumption of a highly cohesive component, depending on groups of classes that have similar data usage characteristics, fits our experience in component design. The associative nature yielded the same output for these two different algorithms in our case; one is based on the shifts of columns and rows and the other is based on the numerical similarity among classes. Results may not be the same when we apply these two algorithms to other problems. These algorithms provide a shortcut for software engineers with limited experience in CBSD to construct their component architectures.

Although this study highlights the value of clustering techniques in component architecture design, we need more cases to fully validate the appropriateness of component clusters. We will therefore explore other clustering techniques applied to the use-case/class in a future study.

Acknowledgements

The authors are grateful to the National Science Council, Taiwan, R.O.C., for supporting this research project number NSC 91-2622-E-027-010-CC3. We also appreciate data support from `www.surfingfun.net`.

References

1. Advisor, http://www.advisor.com
2. L. A. Belady and C. J. Evangelisti, System partitioning and its measure, *J. Systems and Software* (1981) 23–29.
3. J. Cheesman and J. Daniels, *UML Components*, Addison-Wesley, 2001.
4. D. F. D'souza and A. C. Wills, *Objects, Components, and Components with UML*, Addison-Wesley, 1998.
5. D. H. Hutchens and V. R. Basili, System structure analysis: Clustering with data bindings, *IEEE Trans. Software Eng.* **SE-11**(8) (1985) 749–758.
6. D. C. Ince and M. J. Shepperd, The use of cluster techniques and system design metrics in software maintenance, in *Proc. UK IT'90 Conf. Southampton*, IEE/DTI, UK, 1990, pp. 139–142.
7. R. A. Johnson and D. W. Wichern, *Applied Multivariate Statistical Analysis*, 3rd ed., Prentice Hall, 1992.
8. S. D. Lee, Y. J. Yang, E. S. Cho, S. D. Kim, and S. Y. Rhew, COMO: A UML-based component development methodology, in *Proc. Sixth Asia Pacific Software Engineering Conference* (APSEC '99), 1999, pp. 54–61.
9. T. Raz and A. T. Yaung, Application of clustering techniques to information systems design, *Information and Software Technology* **37**(3) (1995) 145–154.
10. R. W. Selby and V. R. Basili, Error localization during maintenance: generating hierarchical system descriptions from source code alone, *Proc. IEEE Conf. Software Maintenance*, 1988, pp. 192–197.