Applications of Clustering to Early Software Life Cycle Phases

Chung-Horng Lung
Department of Systems and Computer Engineering
Carleton University
Mackenzie Building 4456
Ottawa, Ontario, K1S 5B6, Canada
Tel: (613) 520-2600 ext. 2642
Fax: (613) 520-5727
Email: chlung@sce.carleton.ca

Amit Nandi
Marzia Zaman
Nortel Networks
P.O. Box 3511, Station C
Ottawa, Ontario K1Y 4H7, Canada
anandi@nortelnetworks.com

Abstract

This paper presents empirical studies of applying numerical taxonomy to component clustering and software partitioning. The technique is based on cohesion and coupling information of a software system as these two properties have great impact on other software quality attributes such as maintainability, portability, scalability and reusability. The technique has been successfully applied to several real-time software systems in telecommunications and computer networks at various levels of abstraction or software life-cycle phases. Numerical taxonomy is mathematically simple and yet it provides a useful mechanism for component clustering and software partitioning. This paper presents experiments of the clustering techniques to various types of application and demonstrates a visualization tool that facilitates the analysis process.

Keywords: clustering, software partitioning, use cases, requirements analysis, cohesion and coupling, patterns, software architecture

1. Introduction

Alexander [64] postulated that the major design principle which are common to all engineering disciplines is the relative isolation of one component from other components. This idea has also been a topic of interest in software engineering. System partitioning plays a vital role in overall design, as it greatly affects the downstream development phases. Clustering techniques can be used to support partitioning. Clustering and partitioning are considered as two sides of a coin in our view. Partitioning is a top-down approach to divide a system into subsystems with an aim of high cohesion within a subsystem and low coupling among subsystems. Clustering, on the other hand, is like a bottom-up approach to group similar components as clusters. Collection of clusters form a subsystem or a system.

Software engineering is a relatively new area compared to other well established disciplines, such as mechanical engineering and manufacturing. Effective partitioning is also a paramount goal in those disciplines. Clustering techniques have been successfully used in many areas to assist grouping of similar components and effective partitioning of a system. In fact, classification or clustering analysis has been of long-standing interest and is one of the most fundamental methods used in science and engineering to facilitate better understanding of the observations and the subsequent construction of complex knowledge structures from features and component clusters. For instance, the technique has been used to classify botanical species and mechanical parts. The key concept of clustering is to group similar things together to form a set of clusters, such that intra- cluster similarity is high and inter-cluster similarity is low. As mentioned earlier, an effective software partitioning technique also aims for the same criteria i.e. high cohesion and low coupling.

Various clustering techniques have been studied in software engineering, particularly in the area of reverse engineering [ICSM, IWCRE, IWPC]. In this paper, we borrow ideas of clustering techniques from other established disciplines and tailor them to early softwar life phases, e.g., use cases analysis and requirements analysis.

The clustering techniques adopted in this paper are based on numerical taxonomy. Numerical taxonomy uses numerical methods to make classifications of components. The main reason that we adopt numerical taxonomy is its conceptual and mathematical simplicity, as will be demonstrated in Section 2. However, no scientific study has shown that numerical taxonomy is inferior to other more complex multiversity methods. Because each method has potential for revealing insight that may be lacking in other methods [Romesburg90].

The objective of this paper is to examine existing numerical clustering techniques used in other well established disciplines and apply the techniques to software engineering requirements and use cases analysis phase. The main idea is to better support system partitioning at the early stage. The paper is organized as follows: Section 2 includes a brief overview of the clustering technique adopted for this research. Section 3 highlights some related work in the area of software engineering. Section 4 demonstrates several practical applications of the clustering techniques. Finally, Section 6 presents summary and future directions.

2. Overview of Clustering

Applications of clustering analysis can be found in many disciplines. Many clustering methods have been presented [Anderberg73, Everitt80, Romesburg90], but they comprises the following three common key steps:

- Obtain the data set.
- Compute the resemblance coefficients for the data set.
- Execute the clustering method.

An input data set is an component-attribute data matrix. Components are the entities that we want to group based on their similarities. Attributes are the properties of the components.

A resemblance coefficient for a given pair of components shows the degree of similarity or dissimilarity between these two components, depending on the way the data represents. A resemblance coefficient could be qualitative or quantitative. A qualitative value is a binary representation, e.g., the value is either 0 or 1. 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 various methods for calculating the resemblance coefficients. This paper does not discuss those in detail. The interested reader can refer to [Romesburg90]. Instead, we briefly illustrate one algorithm adopted in this paper. In general, there are two types of algorithms for calculating resemblance coefficients based on the scales of measurement used for the attributes. One type of resemblance coefficients can be computed based on qualitative input data or nominal scales for attributes, the other one is based on quantitative input data or the attributes are measured on ordinal, interval, or ratio scales.

This paragraph explains how one algorithm based on binary relations or norminal scales works. Let a, b, c, and d represent number of the pair of 1-1, 1-0, 0-1, and 0-0 matches between two components and assume the following component-attribute input data set for an eight-attribute case.

$$i = \{1, 0, 1, 1, 0, 0, 0, 1\}$$
$$j = \{1, 1, 1, 0, 0, 0, 1, 0\}$$
$$k = \{0, 1, 1, 0, 1, 0, 1, 0\}$$

A 1-1 match between two components indicates that they share this specific attribute. Based on the definition, we can obtain for components i and j that a = 2, b = 2, c = 2, and d = 2. The Sorenson coefficient is adopted in this research and is defined as follows.

$$c_{ij} = 2a / (2a + b + c)$$

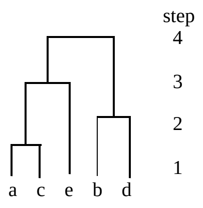
where c_{ij} is resemblance coefficient for components i and j. Selection of this particular algorithm is discussed in [Lung02]. By applying the Sorenson matching coefficient we can get $c_{ij} = (2 \times 2) / (2 \times 2 + 2 + 2) = 1/2$. Similarly, $c_{ik} = (2 \times 1) / (2 \times 1 + 3 + 3) = 1/4$ and $c_{jk} = (2 \times 3) / (2 \times 3 + 1 + 1) = 3/4$. The

0-0 match, d, is not used in this particular algorithm. This procedure is repeated for each component pair to obtain the resemblance matrix. For this method, the higher a coefficient, the more similar the two corresponding components represent. Components j and k in this example are more similar, since the resemblance coefficient c_{jk} is the largest.

Given a resemblance matrix, a clustering method is then used to group similar components. Basically, a clustering method is a sequence of operations that incrementally group similar components into clusters. It starts with each component being in a separate cluster. Each step, two clusters that are closest to each other (either the largest or the smallest coefficient, depending on the way you look at it) are merged and the number of clusters is reduced by one. After these two clusters are merged, the resemblance coefficients between the newly formed cluster and the rest of the clusters are updated to reflect the closeness to the new cluster. A commonly used algorithm called UPGMA (unweighted pair-group method using arithmetic averages) [Romesburg90] is adopted to find the average of the resemblance coefficients when two clusters are merged. For the above example, after component j and component k are formed as a new cluster, the resemblance coefficient between the new cluster (j,k) and component or cluster i is the average of c_{ij} and c_{ik}, which is (1/2+1/4) / 2 = 3/8. The process repeats until all clusters are exhausted, or a pre-defined threshold value is reached.

A tree or a dendrogram is commonly used to indicate the clustering process. 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 are.

Figure 1. An Example of a Dendrogram



We use the clustering technique in two different ways. We tailor the technique based on component-component independencies. In this case, the data set now represents the interdependencies or interconnections among components instead of component-attribute relationships. For example, in Table 1, the 1 entries show that the corresponding components are interdependent or interrelated.

Table 1. Matrix Representation for Component Interdependency

	E1	**E2**	**E3**	**E4**	**E5**	**E6**	**E7**	**E8**	**E9**
E1	1	0	1	0	0	1	0	0	1
E2	0	1	1	1	1	0	0	1	0
E3	1	1	1	0	0	1	0	0	0
E4	0	1	0	1	1	1	0	1	0
E5	0	1	0	1	1	1	1	0	1

E6	1	0	1	1	1	1	0	0	0
E7	0	0	0	0	1	0	1	0	1
E8	0	1	0	1	0	0	0	1	0
E9	1	0	0	0	1	0	1	0	1

By applying the Sorenson clusteirng technique to the problem displayed in Table 1, we can obtain the system paritioning shown in Figure 2. The partitioning, however, for the same problem could look like Figure 3, if it is not well designed [Heyliger94]. Evidently, the inter-subsystem connection or coupling is significantly reduced from Figure 2 to Figure 3.

Figure 2. Grouping Based on Clustering of Components

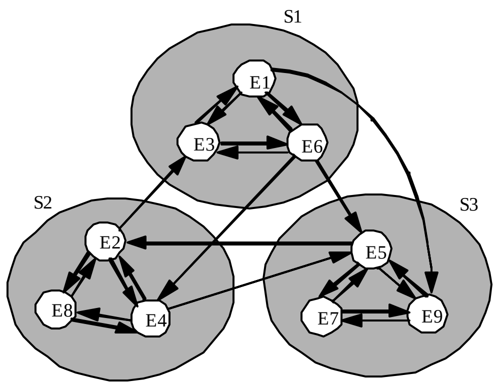
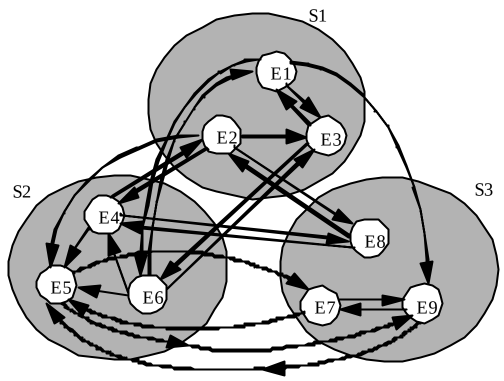


Figure 3. Arbitrary Grouping



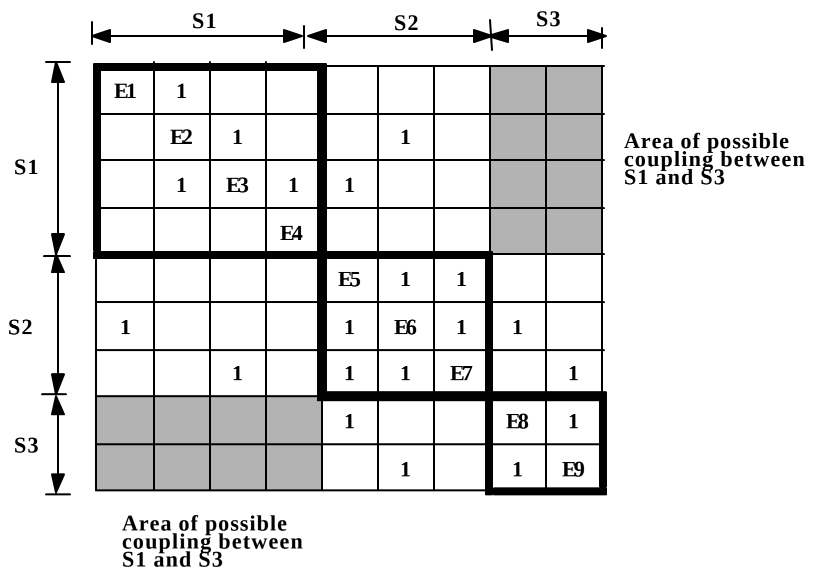
3. Related Work

Applications of the clustering concept specific to the software partitioning have been studied. Andreu [77] applied the partitioning concept to a data base management system to minimumize coupling. Requirements and their interdependencies were first identified and were converted to a graph problem.

Various alternatives for partitioning were examined and a quantitative metric was calculated for each alternative. The alternative with the lowest value of coupling was chosen as the optimal partitioning. The process is time-consuming.

Heyliger [94], on the other hand, has proposed to use N square charts to partition a large system into subsystems. For an N square chart, the rectangles on the main diagonal represent the system partition. Figure 4 shows an example of an N square chart with three subsystems, S1, S2, and S3. The 1's within a subsystem indicate internal strength or cohesion. The 1's outside the partitions or clusters represent coupling among subsystems. Table 1 conveys similar idea.

Figure 4. N-Square Chart Representation: An Illustration (adapted from [Heyliger94])


S1
S2
S3
E1
E2
E3
E4
E5
E6
E7
E8
E9
Area of possible coupling between S1 and S3
Area of possible coupling between S1 and S3

The objective of Heylinger's approach was to incrementally refine the design to maximize cohesion and minimize coupling. Heylinger has identified a set of patterns that characterize specific interfaces among system elements. These patterns provide mechanisms for system structural reorganization and refinement for low inter-subsystem couplings. This process, as depicted by the author, is labor intensive and the rearrangement of the elements is a major problem even for small or modest systems.

Both of these two papers share a common goal, which is to minimize the interconnectivity among components. Selby and Reimer [Selby95] presented an analysis on the interconnections of a software and system errors. They also discussed various approaches to clustering software based on component interconnections. Lakhotia [97] has also conducted a survey on different subsystem classification techniques that have been proposed to classify software into a particular subsystem. The main objective of this paper is to present a unified framework for expressing subsystem classification techniques.

Clustering techniques have also been applied to other software engineering areas. Lung et al. [92, 94] present an application of the clustering technique to domain modeling. Numerious papers discuss software architecture recovery and re-engineering using clustering techniques [ICSM, IWCPC, IWCRE]. Other areas include process management and planning [Raz95].

4. Applications to Early Software Life Cycle Phases

This section presents applications of the clustering method to show how it can be used to support the early software life cycle phases. The objective is to partition the system to obtain high cohesion within subsystems and low coupling among subsystems.

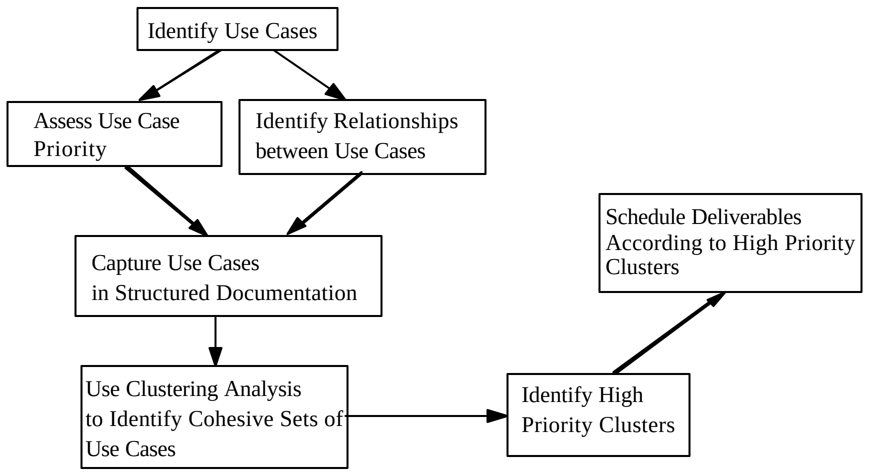
4.1 Application to Use Case

We have used the clustering method in an enterprise development project. Use cases were identified and analyzed to determine development priorities, plan the project, and determine architectural fit between a third party tool and the software system as shown in Figure 5. Each of these steps will be described below:

Identifying Use Cases & Related Information

Jacobson [92] defines use cases as a set of behaviorally related sequence of transactions in a dialogue with the system. For our system, each use case was identified by a short descriptive name and given a unique use case identification number. After all of the use cases were identified, the priority of each was determined along with the dependencies between that use case and the other use cases.

Figure 5. Process of Using Cluster Analysis with Use Cases


Identify Use Cases
Assess Use Case Priority
Identify Relationships between Use Cases
Capture Use Cases in Structured Documentation
Use Clustering Analysis to Identify Cohesive Sets of Use Cases
Identify High Priority Clusters
Schedule Deliverables According to High Priority Clusters

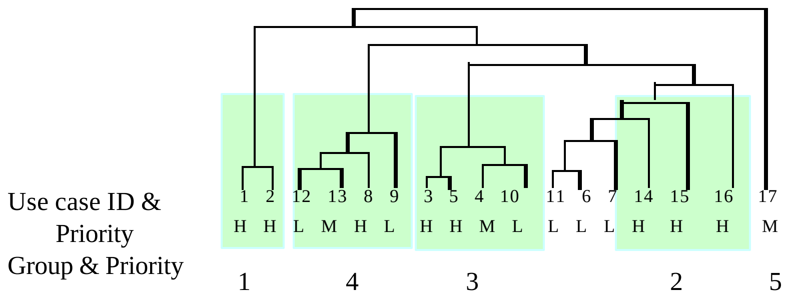
Cluster Analysis of Use Cases

Since all use cases, use case dependencies and priority information were captured. The coupling information between use cases was then used to create a dependency graph that was clustered to give insight into use cases that were functionally related. The idea is described in Section 2. The coupling information is then covered to a table of 1's and 0's, as shown in Table 1. This information is not necessarily obvious from the use case descriptions.

Applications to the System

The next step required to move forward in the design of the system was to divide the work in a logical way - both in terms of overall functionality and in terms of time on a project schedule. Each cluster of use cases consists of one or more use cases with individual priorities. Two forces are considered in determining the priority of each cluster. First force is the leval and number of highest priority use cases within a cluster. The other one is the average priority of all the use cases within a cluster. These two forces determine the order and grouping of the use cases in the cluster for the schedule. This was a useful method for scheduling that reduced the problem of mismatch between the project plan and the technical details of functional dependencies. Figure 6 demonstrates a dendrogram of the use cases along with their corresponding clusters.

Figure 6. Dendrogram Showing System Use Case Cluster and Priority



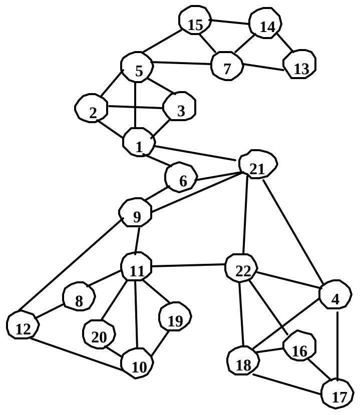
The system in Figure 6 shows the logical partitions or groups of use cases that were prioritized according to the individual priorities of the use cases within each cluster. For this example, the development priorities of the use cases will be (1, 2), (14, 15, 16), (3, 5, 4, 10), (12, 13, 8, 9), (17), and finally (11, 6, 7).

4.1.1 Application to Requirement Analysis

Alexander [64] presented an approach for system partitioning in building a community village. The approach first outlined a number of critical requirements and identified the interdependencies among those requirements. Given the information, these requirements were then grouped into several clusters. An appropriate form or style was then chosen for each cluster. The syntheses of these styles in turn formed a village.

Andreu [77] applied Alexander's partitioning concept to the a data base management system. Figure 8 shows the requirements and their interdependencies for the DBMS example. Table 2 depicts the N-square representation as a result of the clustering process. The table shows five clusters highlighted with bold rectangles. The clusters were then evaluated against the corresponding requirements. Each cluster actually represented a subsystem. A high-level layered architecture was developed based on the analysis result.

Figure 8. Requirements and Their Interdependencies for a DBMS (from Andreu [77])



The clustering method presented in this paper was also applied to Andreu's data base management system at the requirement level. The result was exactly the same as that described in the article. However, the response time is instantaneous and is much faster, because we don't need to identify all possible partitions and calculate the strength for each one of them and select the highest one. This is a time-consuming process, especially if the number of entities is high. Likewise, it is difficult to find the optimal partitions with the N-Square chart approach [Heyliger94].

Table 2. N-Square Representation for the DBMS Partitioning

	1	**2**	**3**	**5**	**6**	**9**	**21**	**8**	**10**	**11**	**12**	19	20	**7**	**13**	**14**	**15**	**4**	**16**	**17**	**18**	**22**
1		1	1	1	1		1															
2	1		1	1																		

3	1	1		1																		
5	1	1	1											1			1					
6	1					1	1															
9					1		1			1	1											
21	1				1	1												1				1
8										1	1											
10										1	1	1	1									
11						1		1	1			1	1									1
12						1		1	1													
19									1	1												
20									1	1												
7				1											1	1	1					
13														1		1						
14														1	1		1					
15														1		1						
4							1													1	1	1
16																				1	1	1
17																		1	1		1	
18																		1	1	1		1
22							1			1								1	1		1	

Determining Potential Architectural Fit

After the the grouping of requirements, Alexander identified an appropriate style for each group. With the increasing applications and experience built with various patterns [Buschmann96, Schmidt00], similar idea to this may be applied to some software applications. We may identify a suitable pattern for each group obtained from the clustering method.

For instance, in concurrent or network applications, requirements may include

- connections setup
- demultiplex and dispatch service requests
- separation of connection and processing of peer services
- decouple processing of incoming and outgoing messages

Those and other relevant requirements can lead to the identification of some design patterns. Some examples include Reactor, Acceptor-Connector, Active Object, and Half-Sync/Half-Reactive. This will improve software design quality and development time.

5. Summary and Future Work

This paper presented an approach for software partitioning based on a numerical taxonomy clustering method. The key value of this approach is that it can support rapid and effective partitioning of a system based on the relationships between components and features or component interdependencies at various levels of abstraction. System partitioning is usually performed by experienced designers in an ad-hoc manner. The method can help designers quickly obtain an outline of the architecture or design. More evaluations could then be conducted to identify potential problems early in the development process.

Further, the method could be used to together as a generative approach to identify appropriate styles or patterns as described by Alexander [64]. As we understand more about architectural styles or patterns [Buchmann96, Shaw96], and the interoperability issues of various styles, the method could support systematic partitioning of software architectures and identification of appropriate patterns, as have practiced by other mature disciplines.

Some other areas are still in progress. We are also working on tools integration. There are various viewpoints that various stakeholders may need. Tools that allow the user to select a view and generate it accordingly will have a lot of values. The other area is to compare various clustering techniques. We also tried to calculate the Euclidean distance coefficients [Romesburg90] which are derived from numeric values rather than binary values as demonstrated in this paper. The clustering method is then applied to the set of resemblance coefficients.

References

[Alexander64] C. Alexander, *Notes on the Synthesis of Form*, Harvard University Press, Cambridge, MA, 1964.

[Anderberg73] M.R. Anderberg, *Cluster Analysis for Applications*, Academic Press, New York, 1973.

[Andreu77] R.C. Andreu and S.E. Madnick, *A Systematic Approach to the Design of Complex Systems: Application to DBMS Design and Evaluation*, Technical Report CISR 32, MIT Sloan School of Management, 1977.

[Buschmann96] F. Buschmann, et al., *Pattern-Oriented Software Architecture*, Wiley, 1996.

[Card90] D.N. Card and R.L. Glass, *Measuring Software Design Quality*, Prentice Hall, 1990.

[Dhama95] H. Dhama, “Quantitative Models of Cohesion and Coupling in Software”, *J. of Systems and Software*, vol. 29, 1995, pp. 65-74.

[Dobbing98] T. Dobbing, “Initial Set of Structured Documentation and Review Templates for the Various Development Phases from which Metrics are Automatically Extracted”, *Proc. of SRE’98*, July 1998.

[Dromey96] R.G. Dromey, “Cornering the Chimera”, *IEEE Software*, Jan. 1996, pp. 33-43.

[Everitt80] B. Everitt, *Cluster Analysis*, Heinermann Educational Books, Ltd., London, 1980.

[Heyliger94] G. Heyliger, “Coupling”, *Encyclopedia of Software Engineering*, J. Marciniak (ed.), 1994.

[Hutchens85] D. Hutchens and V.R. Basili, “System Structure Analysis: Clustering with Data Bindings”, *IEEE Trans. on Software Eng*., vol. SE-11, no. 8, Aug. 1985, pp. 749-757.

[ICSM] *Proceedings of Int’l Conf. on Software Maintenance.*

[IWCPC] *Proceedings of Int’l Working Conf on Program Comprehension.*

[IWCRE] *Proceedings of Int'l Working Conf on Reverse Engineering.*

[Jacobson92] I. Jacobson et al., *Object-Oriented Software Engineering A Use Case Driven Approach*, Addison-Wesley, 1992.

[Kazman94] R. Kazman, L. Bass, G. Abowd, M. Webb, "SAAM: A Method for Analyzing the Properties of Software Architecture", *Proc. of ICSE-16*, 1994, pp. 81-90.

[Kazman98] R. Kazman and S.J. Carriere, "View Extraction and View Fusion in Architectural Understanding", *Proc. of the 5th Int'l Conf on Software Reuse*, April 1998.

[Lakhotia97] A. Lakhotia, "A Unified Framework for Expressing Software Subsystem Classification Techniques", *J. of Systems and Software*, vol. 36, 1997, pp. 211-231.

[Lung92] C.-H. Lung, J.K. Cochran, G.T. Mackulak, and J.E. Urban, "Empirically Analyzing Software Reuse in a Simulation Environment", *Proc. of the Workshop on Software Reuse (WISR)*, Oct. 1992.

[Lung94] C.-H. Lung, J.K. Cochran, G.T. Mackulak, and J.E. Urban, "Computer Simulation Software Reuse by Generic/Specific Domain Modeling Approach", *Int'l Journal. of Software Eng. and Knowledge Eng.*, vol. 4, no. 1, March 1994, pp. 81-102.

[Lung02] C.-H. Lung, M. Zaman, and A. Nandi, "Applying Numerical Taxonomy to Software Architecture Partitioning, Recovery, and Restructuring", *submitted for publication*.

[Monroe97] R. Monroe, A. Kompanek, R. Melton, and D. Garlan, "Architectural Styles, Design Patterns, and Objects", *IEEE Software*, Jan. 1997, pp. 43-52.

[Murphy01] G.C. Murphy, D. Notkin, and K.J. Sullivan, "Software Reflexion Models: Bridging the Gap between Design and Implementation". *IEEE Trans. of Software Eng.*, vol. 27, no. 4, April, 2001, pp. 364-380.

[Neighbors96] J. Neighbors, "Finding Reusable Software Components in Large Systems", *Proc. of Working Conf. on Reverse Engineering*, 1996, pp. 2-10.

[Raz95] T. Raz and A.T. Yaung, "Application of Clustering Techniques to Information System Design", *Information and Software Technology*, vol. 37, no. 3, Mar, 1995, pp. 145-154.

[Romesburg90] H. C. Romesburg, *Cluster Analysis for Researchers*, Krieger, Malabar, Florida, 1990.

[Selby95] R.W. Selby and R.M. Reimer, *Interconnectivity Analysis for Large Software Systems*, Technical Report, UCIrv-95-PROC-CSS-001, Univ. of California at Irvine, 1995.

[Shaw96] M. Shaw and D. Garlan, *Software Architecture Perspectives on an Emerging Discipline*, Prentice Hall, 1996.

[Smidt00] D. Schmidt, et al., *Pattern-Oriented Software Architecture Patterns for Concurrent and Networked Objects*, John Wiley & Sons, 2000.

[Wiggerts97] T.A. Wiggerts, "Using Clustering Algorithms in Legacy Systems Modularization", *Proc. of the 4th Working Conf. on Reverse Engineering*, 1997, pp. 33-43.