Association for Information Systems
AIS Electronic Library (AISeL)

ICIS 1983 Proceedings

International Conference on Information Systems (ICIS)

1983

Computer-Aided Process Organization In Software Deslgn

Jahangir Karimi
University of Cincinnati

Benn R. Konsynski
University of Arizona

Computer-Aided Process Organization In Software Design

Jahangir Karimi
University of Cincinnati

Benn R. Konsynski
University of Arizona

ABSTRACT

As the complexity of systems increase, the need for computer-aided techniques in software system definition, design, and construction becomes apparent. It is the ultimate task of software engineering to develop tools and procedures which reduce the effort involved in production of effective software. Effective software must possess characteristics of correctness, reliability, efficiency, documentation, and flexibility. This paper deals with the development of a computer-aid for one portion of the software system design problem, namely, the determination of process organization in program module specification.

INTRODUCTION

The purpose of computer-aids in design is to reduce manpower and time investments while improving the quality of design by evaluating more alternatives and accommodating the complexity that humans cannot. Secondary benefits include: improved operation of the design process, documentation of both the system requirements and design decisions, and ease of modification and determination of the impact due to change.

This paper deals with the development of a computer-aid for one portion of the information system design problem -- the determination of program modules in design of the software. While the discussion will focus on program module design for information systems, the theory and principles are directly applicable to basic design of large application programs.

Whether the development model is a "life cycle" model or a "prototyping" model, the determination of an effective modularization of the processing activities is an important element. The focus of this current research in the PLEXSYS (Konsynski and Nunamaker, 1981) effort is the identification and specification of program modules.

PROGRAM MODULE DESIGN

A system is composed of interacting parts that operate together to achieve some objective or purpose. Information system requirements can therefore be described in terms of characteristics of the system components and the relationship between those components. From the standpoint of program module determination, the focus is on the information system as a system of interacting processes which act to transport and modify data. The term "process" can thus be used to describe the activities of the system. Processes and their interrelationships necessarily exhibit a high degree of com-

plexity in large-scale information systems. The purpose of this paper is to present a general model &or analysis and organization of a complex system of processes in determination of effective modularization.

A process is a logical unit of computation or manipulation of data. Under this definition, a process could be a simple primitive operation, such as an addition, or a system itself being a process. If the scope is too small, the complexity of the problem is considerable and the benefits of extensive analysis are not increased appreciably. If the scope is too large most of the benefit of the analysis is lost.

Modularization is a factoring of the system into interacting modules such that the modules together perform as the system. In terms of a system of processes, modularization is the determination of subsets of the set of processes that satisfy some evaluation criteria such as reducing the intermodule interface. The subsets may be hierarchical and may overlap. For our present discussion, we will consider modularizations that form a cover for the set of processes and constitute a partitioning.

We can make several observations concerning modularization and present practices in obtaining them:

- There exist many meaningful modularizations - The number of alternative feasible modularizations is very large and different designs are more favorable for alternative design criteria. The number of alternative designs is a function of the number of processes and, more importantly, the number of logical inter-process links. A significant number of these feasible designs could each be the choice for logical organization based on differing criteria with differing significance.

- The use of a single criteria in modularization would give a distorted view of process interrelationships. Processes possess many different attributes and there exist many different types of process relations. If one or a small subset of attributes or relations is emphasized above all others, through the analyst's bias, the result could be disasterous in terms of the neglected or weakly weighted criteria.

- A hierarchical approach limits succeeding organizations - A top-down approach is a convenient means of arriving at a clear logical statement of requirements, but we should not give it complete authority.

- The same modularization may be derived in different ways - Even under our manual procedures of modularization, an output decomposition approach may yield an organization identical to one derived through a functional decomposition approach.

- The complexity of operating environments suggests use of computer-aided techniques in determination of effective modularizations.

CURRENT APPROACHES

This design phase is frequently performed by professional staff who are experts in the application area that the software will serve. Several approaches have been proposed to support the activities at this stage. Much effort has been devoted to the attainment of modules that have three specific properties, expressed by White and Booth (1976) as properties they "would like to see a design possess": (1) components are relatively independent, (2) existing dependencies can be easily understood, and (3) there are no hidden or unforseen interactions between components.

Myers (1975) and Constantine (Yourdon, 1979) have proposed a series of qualitative rules and guidelines for obtaining software modules meeting these properties. In particular they introduced the terms internal module strength or cohesion which refers to level of association between component elements of a module, and module coupling referring to a measure of the strength of interconnection between modules.

Myers (Myers, Stevens, and Constantine, 1974) recognized seven levels of cohesion. They state, "these levels have been distinguished over the years through experiment, theoretical argument, and the practical experience of many designers." The seven levels are, in order of increasing strength or cohesion: Coincidental, Logical, Temporal, Procedural, Communicational, Sequential, and Functional.

Dunsmore (1979) found no relationships between the perceived difficulty of understanding a computer program (Psychological Program Complexity) to the number of modules in a program. Both Dunsmore (1979) and Gelperin (1979) did find a perceived complexity relationship between the amount and nature of inter-module communication and the single-function nature of the module. In particular, Gelperin (1979) found data that partially confirms Yourdon's hypothesized effects of module coupling: information content, type of connection, and type of communication.

In spite of the difficulty in gathering controlled experimental data, results are now available concerning perceived inter-module complexity. Gelperin (1979) has partially confirmed Yourdon's hypothesized effects of intelligent module design on program understanding. Another study (Willis and Jensen 1979) extended the application of structured design principles as published by Myers, et al. (1974), to military embedded software systems. It was found that the principles of structured design, if applied correctly, can result in lower cost, more reliable software systems.

While principles of cohesion and coupling can be useful guides in evaluating the structure of a program, they do not provide an unambiguous methodology for attaining programs with high levels of cohesion or coupling.

SYSTEMATIC PROGRAM MODULE DESIGN

A major difference between a good and poor design structure is the associated complexity. Complexity can be reduced by: 1) partitioning the system into parts having identifiable and understandable boundaries, 2) representing the system as a hierarchy, and 3) maximizing the independence among the parts of the system.

Although partitioning aids in the comprehension of the system, arbitrary partitioning can actually increase the complexity by having parts of the system perform many unrelated functions. Another result may be an increase in inter-part connection complexity. Hierarchy in turn helps in both understanding and constructing a system. However, the hierarchy by itself is often not an effective solution for other design criteria. What is needed are methods for decomposition of a system into a hierarchy such that each part (module) is as independent from all modules as possible. A measure of independence is module strength (cohesion) which classifies internal relationships of a module and module coupling which defines the direct inter-module relationships among all modules.

Although there exist several design methodologies few have been extensively tested. Three types of methodologies (Yourdon 1979) which have been frequently used are: 1) functional

design 2) data flow design, and 3) data structure design method.

Functional decomposition, often called stepwise refinement method, is a top-down approach to problem solving. Under this approach, the problem is divided into functional activities. By reexpressing each function as properly equivalent connected structures, each function is divided into subfunctions. A major problem with functional decomposition method is the unavailability of a clean strategy to help the designer in decomposing the problem. The question that frequently arises is "decomposition with respect to what?." Different decomposition methods result in different designs and the number of potential deompositions is large. The criteria selected for decomposition has a major effect on the "goodness" of the resulting design.

Data flow design method is reported in Yourdon (1979) and Myers (1978). This approach has also been called Transform Centered Design and Composite Design. The initial activity in this strategy is to draw the correct data flow diagram which represents the flow of data in the system. Next, the program structure chart is derived which represents the input, process, output transforms corresponding to steps in the data flow diagram.

One major problem with the data flow design is the lack of correspondence between the structure of the design and the structure of the problem. A hierarchy is imposed on the design structure by appropriate scheduling and/or linking of parts that has little to do with modeling the problem in hierarchical fashion. Frequently, a module is artificially created to control the function of the subsequent modules. This may or may not result in a structure that models the problem environment accurately. There is a similarity between the methods of function decomposition and data flow design. Where the modules in functional decomposition tend to be attached by a "uses" relationship, the steps in a data flow diagram could be labelled "becomes."

The data structure design method was developed in slightly different forms by Jackson (1975) and Warnier (1974). The strategy can be summarized under three basic steps. First, the structures for the data that are to be processed are defined. Second, the program structure which corresponds to the data structure is derived. Finally, elementary operations are defined in order to perform each specified task. For small problems with well defined problem specification this approach would result in similar designs when applied by different people. The utility of this approach is being studied by Konsynski (1982) and Martin (1983).

A major shortcoming in application of current techniques to design of large-scale systems is that they are not suitable for designing software modules as they are for designing within program modules. As Myers (1978) points out:

> "If the product being developed is a system, rather than a single program, there is another design process that must occur between the external design process and the use of "composite design." This process, called system design, is the decomposition of the system into a set of individual subsystems or individual programs. Although some of the ideas of composite design are appropriate here, and some people have claimed to have used composite design for this process, composite design does not appear to be directly applicable to system design. Therefore, when designing a system, as opposed to an individual program, the designer must

first partition the system into distinct subsystems or programs. Then the methodology of composite design can be used to produce the structure of these individual pieces."

There is a growing need for a design tool that can be applied to the detailed design process whether the product being developed is a program or a system. The function of the tool need not be to suppress the designer, but rather to support his activities and to provide explicit design guidelines. Furthermore, the tool should provide a unified approach to design process regardless of the size of the problem. The tool should also provide a quantitative measure of "goodness" for a design in order to facilitate the design evaluation by the analyst.

In addition to the structure of the individual module, the collective structure assumed by those modules must also be considered. Structural design guidelines (Yourdon, 1979), imply that a "good" structure is one in which modules are structured in "hierarchy," where the modules on a given lower level may or may not be shared by the modules on the higher level. In addition to the extent of sharing, there are restrictions on the number of levels relative to total size of the system, and the number of intermediate subordinates for a given module at each level. Although, these are characteristics of a "good" design, there is no clear methodology for the designer to follow during the design process.

As Constantine and Yourdon (1979) state, "... these heuristics can be extremely valuable if properly used, but actually can lead to poor design if interpreted too literally ... results often have been catastrophic ...."

PROPERTIES OF SYSTEMS

In order to determine an objective for design we must first explore the properties of systems relevant to design. Generally, we accept the basic qualitative nature of those overlapping properties. The set of properties might include:

Understandability (documented, logical organization)
Flexibility (portability and adaptability)
Maintain ability (subject to normal evolution)
Testability (correctness, conciseness)
Functionality (usability, capability, and operational unity)
Reliability (security, accuracy, consistency, completeness, recoverability)
Efficiency (performance, cost-effective, resource usage)

The list is not intended to be definitive, merely representative. It should be pointed out that no general agreement on the relative importance of the various properties is expected to arise. Most may initially agree with the notion that the properties are all of equal importance, yet, experience has shown that in different situations and different organizations, certain aspects take on more significance.

As the defined properties are basically qualitative, we lack any generally accepted measure of the system quality. We can, however, examine the design decisions that must be made and arrive at a set of design criteria which are, to some degree, measurable. We then must study the influence of those criteria on the various sytem quality properties.

DESIGN CRITERIA

Following examination of the design decisions that must be made with respect to the earlier discussed design factors, we arrive at a partial list of software design criteria:

Data Utilization - an analysis of data transport. Has significant impact on processing time which is a nondecreasing function of the transport volume.

Reference Distribution - an analysis of the relative location of data references or locality of reference. This analysis takes on special significance when a hierarchical memory organizations are involved.

Information Distribution - deals with the distribution of design knowledge among the modules. By "hiding" design information the complexity of development and change can be reduced considerably.

Control Transfer - analysis of the probability of in-line control transfer between processes. Minimization of control transfers can have significant impact on both process and data organization.

Operational Invocation - an analysis of the periodicity of processing and invocation analysis.

Parallelism - through analysis of invocation procedures and data usage we can determine precedence relations and assess potential savings from parallel invocation of processes. This criteria takes on special importance when mixed or multiple processors and/or pipelining are involved.

Next, we will discuss several of the design criteria in more detail.

Data Utilization

Data utilization or transport volume refers to the volume of data that is involved in processing each module. The volumes deal with the logical data organizations that may or may not be realized in the resulting physical design. The higher the volume of data transported between modules, the higher will be the associated processing time of executing module. We can design a system with lower transport volume by grouping processes into program modules in order to eliminate multiple passes and unnecessary creation of the intermediate files and databases.

Transport volume can be used as a measure of comparison between candidate designs. It should not be taken as the only measure for the design as lower transport volume does not influence logical consistency of each module or the structure of the system as a whole.

Reference Distribution

In programming environments, reference distribution is defined (Ferrari, 1976) as the pattern of references by statements to data and statements to each other. The concept is especially important in virtual memory environments, in which analysis of the program reference string will show how, under the given paging algorithm, the behavior of program could be improved by changing its reference distribution.

In the case of a system of modules, the reference behavior is far more predictable than that of program statements. In a modular system, reference distribution depends on the structure of the system and the extent of sharing of lower level modules by higher level modules. Locality of reference is insured if sharing is lim-

ited. This would be more possible if the structure lends itself more to "pure" tree structure than hierarchy structure.

Information Distribution

The concept of information distribution is described by Parnas (1972) into subsystems. As Parnas describes, in decomposing a system into subsystems, attention has to be given to design interfaces or "connections" which are any sort of interrelationships or interdependencies between subsystems. This includes the distribution of information within the system and includes the flow of control, passed parameters, and shared data structures. Parnas points out that in decomposing a system, each design interface or "connection" should contain as little information as possible to correctly specify it and each subsystem should "hide" assumptions about the solution that are likely to change. Elements that are likely to change include the data structure along with its format codes, linkage access storage, the transforms and their sequence in each subsystem. Parnas states that by "hiding" this information, the complexity of development and change is reduced considerably through the reduction in interdependencies of the subsystems.

Parnas does not offer a procedure to arrive at a "good" structure nor does he explain how to translate the subsystems and their connections into sets of interconnected modules. Information Distribution guidelines can be used after the completion of an overall structured design as a design refinement procedure in order to further optimize the design by reducing the interdependencies between modules.

A set of measurements based on the flow of information connecting system components have been defined by Henry (1979). The measurements are said to be useful in "evaluation of the complexities of the procedures and modules within the system, and the complexities of the interfaces between the various components of the system." The measures are useful in detection of difficulties in the areas of poor functional decomposition of procedures and modules, improper modularization, poorly designed data structure, and modifiability.

Control Transfer

In a paged memory, multiprogramming multi-processed, pipelined or overlayed environment, software logic is frequently required to be segmented into discrete portions. The control transfer model (Lowe, 1969) analyzes the probability of the online control transfer between segments of executable code. The objective is to segment programs in a manner that reduces the frequency of the inter-segment control transfers within cyclic program structures.

Elements of the control transfer model include control units, passive units, and transfer penalties. Control unit refers to program segment, module, or set of operations based on the scope of the design. Each control unit (in the course of system operation) references other control units and/or passive units. Passive units are the files or collections of data needed for the execution of the control units. There are penalties associated with the transfer of control between the control units. A penalty has a corresponding cost which may be fixed or variable, depending on which boundary has been transgressed. Boundaries that separate the set of control units (modules) are imposed by physical constraints or limitations.

THE MODULE DESIGN PROCESS

The program module specification process consists of analysis of the process characteristics and interprocess relationships in determination of a partitioning of the set of processes into modules maximizing satisfaction of the software properties discussed earlier.

To automate the process organization phase of design, a framework is needed for a computer-aided methodology to incorporate the properties of a good design at design time. The objective of our approach is a system of modules with the following properties weighted highly:

a) high level of cohesion

b) low level of coupling

c) low level of reference distribution

d) low level of information distribution

COMPUTER-AIDED PROCESS ORGANIZATION (CAPO)

In order to systematize this phase of the design process, a process structuring workbench has been developed to organize the activities in the detailed design stage of software life cycle. Information on process attributes and inter-process relations are made available from earlier design stages (Konsynski, 1981) or from inverse translation of source libraries. Graphs are derived representing the network of processes within the system. Each node represents a separate transformation on data, decisions have to be made with regard to grouping of these processes to form separate modules.

The level of cohesion of each module depends on the processes which constitute that module. A module that consists of several logically related processes would be more cohesive than a module that consists of fragments of several processes. Depending on the size of the module and the size of the processes that constitute the module, alternative process groupings will result in different levels of cohesiveness. A critical element in the structure of a system is the connection between the modules and the connections of elements within a module.

Figure 1 depicts the overall structure of the Computer-Aided Process Organization (CAPO) for grouping processes to form software modules. The system provides interactive design, relates to other analyzers (i.e., PSA), and fits within PLEXSYS methodology (1981).

After producing a graph representing the network of processes, five matrices are generated and the relationships between processes are examined in order to determine the extent of interdependencies. From this information we generate an interdependency weight which is assigned to the link joining each pair of processes.

In order to assign interdependency weights to links joining each pair of processes, a special weighting scheme was derived. The objective of assigning weights to links joining two processes is to assess the impact in grouping processes in a single module. Modules are to be designed with a high level of cohesion and low level of coupling. A high level of cohesion results when processing elements within a module have strong data or functional relationships. This objective is used in the current CAPO installation (1983).

The resulting weighted graph must be decomposed into a set of non-overlapping subgraphs according to the objective criteria. There are a number

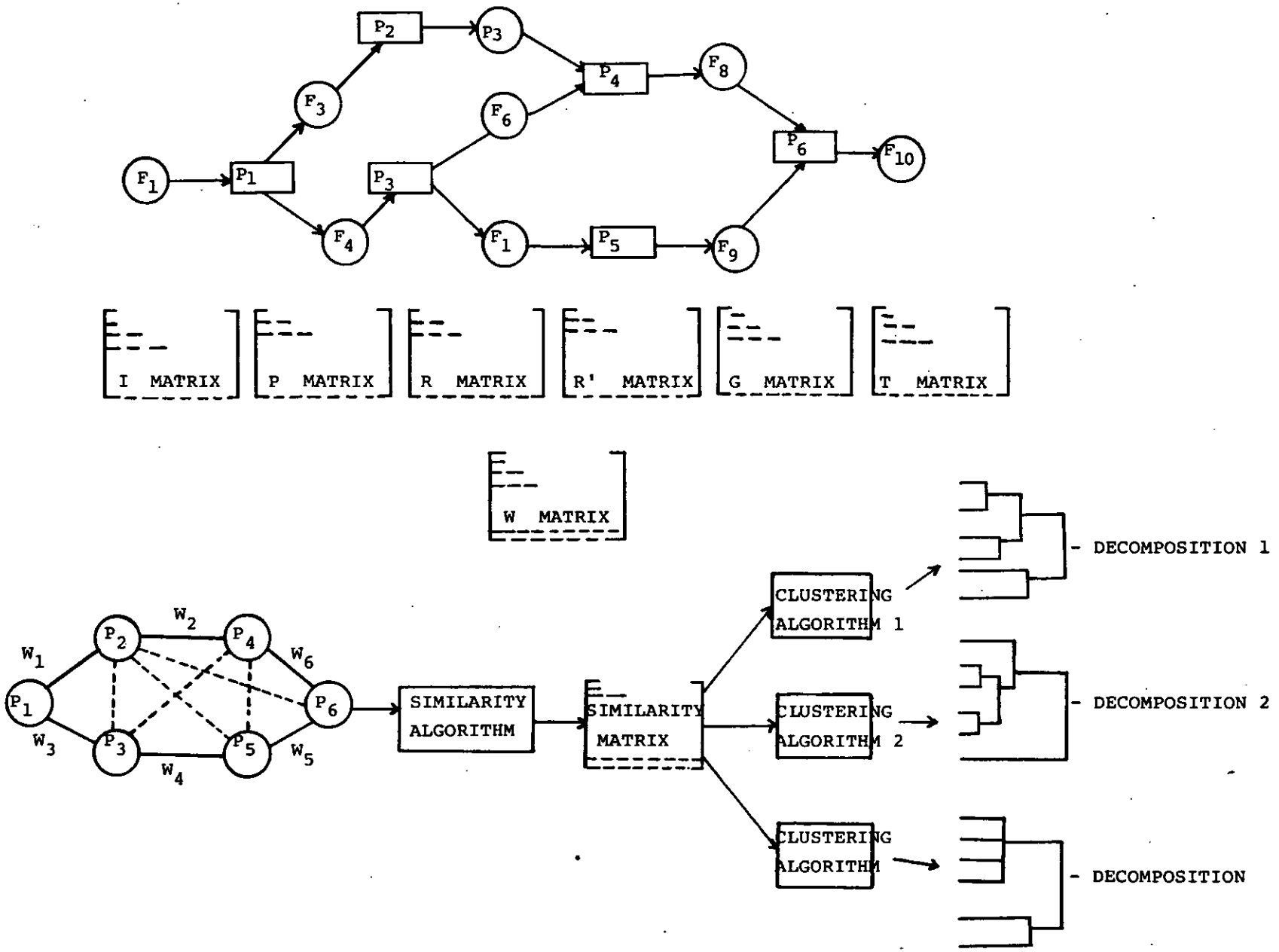


Figure 1. CAPO Overview

METHOD FOR GENERATING PARTITIONS	TECHNIQUES			APPLICABILITY
GRAPH THEORETIC APPROACH	MAXIMAL COMPLETE SUBGRAPH CONCEPT			NOT APPLICABLE
	CUT SET THEORY			NOT APPLICABLE
HEURISTIC APPROACH	CLUSTER ANALYSIS	HIERARCHICAL (AGGLOMERATIVE)	SINGLE LINKAGE	NOT APPLICABLE
			COMPLETE LINKAGE	
			AVERAGE LINKAGE	
			WITHIN NEW GROUP AVERAGE LINKAGE	
			BETWEEN MERGED GROUPS	
			CENTROID CLUSTERING	
			MEDIAN METHOD	
		NON-HIERARCHICAL (PARTITIONING)	LEADER	NOT APPLICABLE
			ITERATIVE	NOT APPLICABLE
			INTERCHANGE	NOT APPLICABLE
	HEURISTIC GRAPH DECOMPOSITION	CORE SET CONCEPT IS USED TO IDENTIFY A SET OF HIGH STRENGTH SUBGRAPH		NOT APPLICABLE DIRECTLY, CORE SET CONCEPT USED TO DEFINE SIMILARITY MATRIX USED IN HIERARCHICAL CLUSTERING

Figure 2. Methods for Generating Graph Partitions

of different methods available to accomplish the organization evaluation. These techniques are divided into two main categories: 1) graph theoretic approach and 2) heuristic approach. Figure 2 below summarizes the various methods which are available for generating graph partitions.

Cluster analysis has been defined as "grouping similar objects." One major assumption made in using any clustering methods deals with the characteristics of the information employed to define similiarities among objects. The procedure used to define similarity depends on the characteristics of the objects and their attributes. The objects are considered to be similar or dissimilar with respect to their contributions to the objective of the clustering analysis.

One major difficulty in performing a cluster analysis is deciding on the number of clusters of the objects. A class of clustering techniques, Hierarchical, gives a configuration of every number of clusters from one (the entire set of objects) to the number of objects (each cluster has only one member). Depending on the size of the cluster, the level of coherence would also change (i.e., clusters with only one member have a maximum level of coherence). On the other hand, some algorithms begin with a selected number of clusters and alter this number as indicated by certain criteria, with the objective of simultaneously determining both the number of clusters and their configurations. Several parameters can be used in order to limit the range of the solution.

A CLUSTER ANALYSIS APPROACH

The method used to generate the similarity matrix for the process-graph was first suggested by Gotlieb (1968) in the context of clustering index terms in library management. The approach is based on the concept of "core set" which is used in heuristic graph decomposition techniques to find "high strength" subgraphs. The core set of a given node (i) in a graph is set of all nodes connected to (i) in the graph including (i) itself. The weighted links, then determine core set values. The concept is illustrated in Figure 3.

The larger the node core set the stronger is the connection of the node to any other node in the core set and the weaker is the connection of the node to any node not in the core set. Stronger connection would bring about higher similarity weights which in turn results in higher probablity that the two nodes should be grouped in a single cluster or module. The procedure in CAPO is to generate the similarity matrix for the process-graph using the analysis package. The core set concept has been used by Huff and Madnick (1979) to define the similarity measure between pairs of nodes in a graph in SDM.

A goodness measure is needed for assessment of partition subgraph strength and subgraph coupling. A separate measure has been developed for each component of the objective function. The strength measure should take into account whether nodes within a particular subgraph are tightly coupled. The process uses the number of links joining the nodes within the subgraph and the cardinality of the subgraph itself. A similar approach has been used to derive the coupling measure. A coupling measure has been derived using the number of links joining nodes in two different subgraphs normalized by the cardinality of the two subgraphs. The "goodness" measure for a partition involves adding the strength of all the subgraphs in the partition and subtracting the result from the coupling associated with all possible pairs of subgraphs. Goodness measures for a

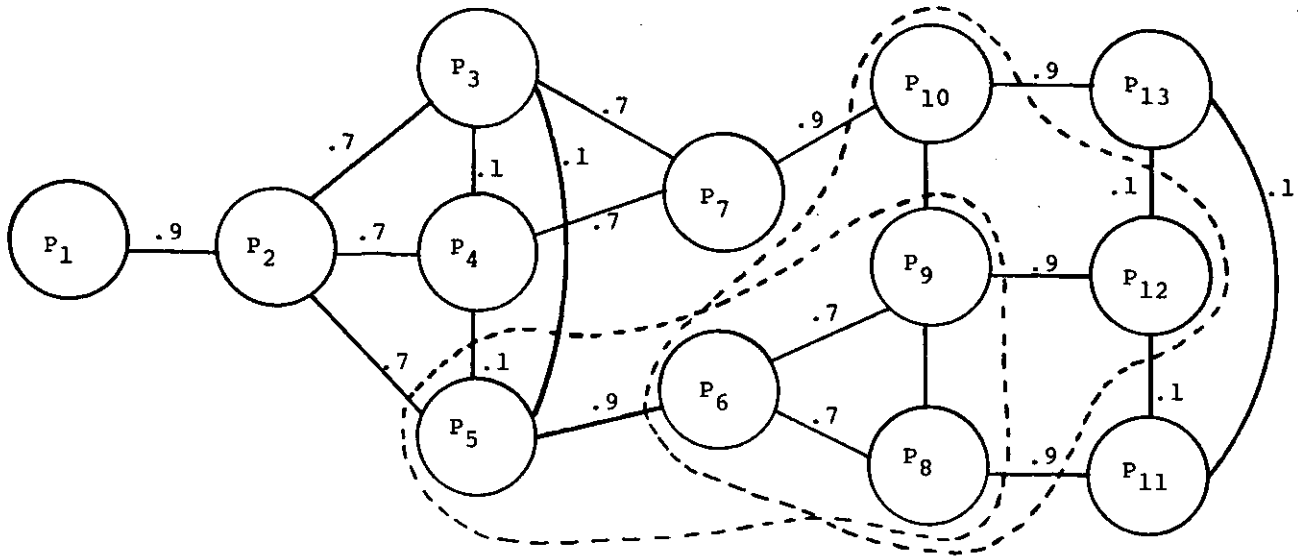


Figure 3. The Core Set Concept

partition have been used previously (see for example, Estabrook, 1966; Hubert, 1974; Andreu, 1978; Huff and Madnick, 1979).

Using the CAPO analysis package, the analyst can ask the system to compute the goodness measure for each stage of clustering and for any of the different clustering methods which are available. Success in producing designs that result in reliable software, even using structured design techniques, is dependent on the experience level of the designer. CAPO provides a quantitive measure of quality necessary in order to ease the dependence on the rare availability of expert designers.

As mentioned earlier, one property of a "good" design is lower data transport volume in the system. Lower transport volume results in lower processing time and lower data organization complexity. Using the CAPO analysis package, the analyst can ask the system to provide volume of data transported between processes and determine total transport volume within the system. This would indicate the necessity of grouping of any pair of processes and/or the effect of grouping of any number of processes on the total transport volume of data within the system.

A CASE STUDY USING A SMALL APPLICATION SYSTEM

To illustrate the application of the various techniques discussed in this paper, including the use of CAPO analysis package, a small design problem is presented below. The problem addressed is the updating of a master inventory file. Figure 4 depicts the graphical representation of this design problem.

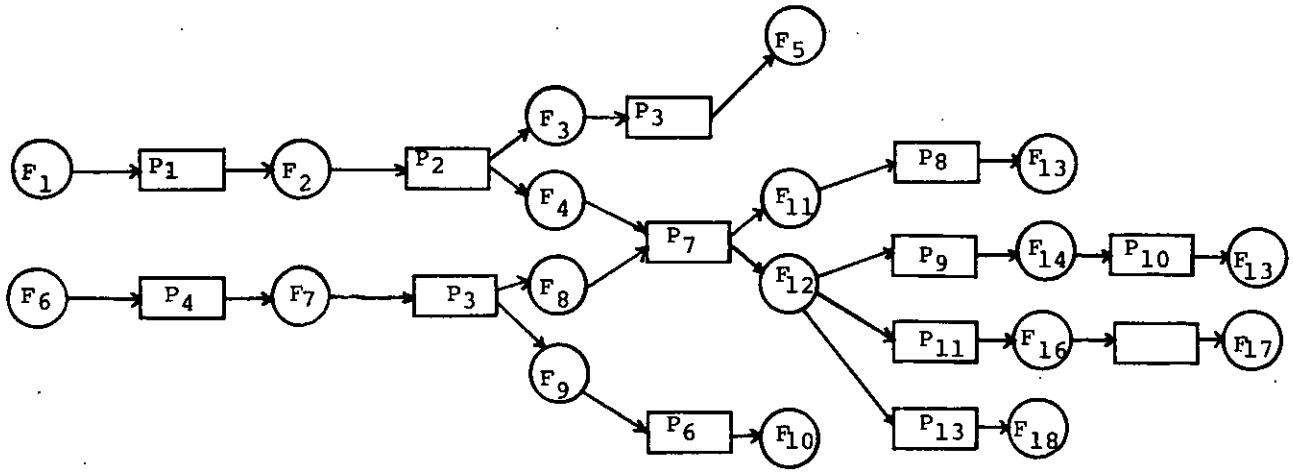
It is easy to present examples of Coincidental and Logical cohesion in this data flow graph. Grouping of two processes, POR-GET-MASTER-RECS and PRO-MATCH-TRANS-AND-MAST-RECS results in Coincidental cohesion: any relationship among the processing elements is purely coincidental. On the other hand, logical cohesion results if two processes, PRO-VERIFY-ORDER-RECS and PRO-VERIFY-MASTER-RECS are grouped in a single module.

As the data flow graph is basically non-procedural, it is not easy to show the Temporal and Procedural cohesion. Grouping of two processes, PRO-GET-ORDER-RECS and PRO-GET-MASTER-RECS is an example of a module involving temporal cohesion.

A Procedural cohesion module results from grouping processes, PRO-MATCH-TRANS-AND-MAST-KEYS, PRO-UPDATE-MASTER-RECS and PRO-COMPUTE-AMT-ORDER-PER-PART. Included in the module with the looping logic itself are the continuing portions of computations and merge logic. Elements of procedural cohesion like the temporal cohesion are related in time, procedure, and sequence-oriented associative principles.

Communication and Sequential cohesion are illustrated on the data flow graph with their problem orientation. Process PRO-MATCH-TRANS-AND-MAST-KEYS has a communicational association on the input side with the two processes, PRO-VERIFY-ORDER-RECS and PRO-VERIFY-MASTER-RECS. On the output side, however, the same process is communicationally related with the four other processes. An example for a sequential cohesion module would be the module which results from grouping two processes PRO-GET-MASTER-RECS and PRO-VERIFY-MASTER-RECS.

The information represented by the data flow graph are used by CAPO analysis package. The information includes the input and output information for each process, the characteristics of



	FILE-NAME
1	TRANS-FILE
2	ORDER-RECS-FILE
3	EXCLUDED-ORDER-RECS-FILE
4	VALID-ORDER-RECS-FILE
5	EXCEPTION-REPORT-A-FILE
6	MASTER-INVENTORY-FILE
7	MASTER-RECS-FILE
8	VALID-MASTER-RECS-FILE
9	EXCLUDED-MASTER-RECS-FILE
10	EXCEPTION-REPORT-S-FILE
11	INVALID-PART-RECS-FILE
12	MATCH-PART-RECS-FILE
13	INVALID-PART-ORDER-FILE
14	UPDATED-MASTER-RECS-FILE
15	UPDATED-MASTER-FILE
16	PART-ORDER-RECS-FILE
17	PART-ORDER-FILE
18	PART-OUT-OF-ORDER-FILE

	NAME
1	PRO-GET-ORDER-RECS
2	PRO-VARIETY-ORDER-RECS
3	PRO-WRITE-EXCEPTION-REPORT-A
4	PRO-GET-MASTER-RECS
5	PRO-VARIETY-MASTER-RECS
6	PRO-WRITE-EXCEPTION-REPORT-S
7	PRO-MATCH-TRANS-AND-MAST-KEYS
8	PRO-WRITE-INVALID-PART-RECS
9	PRO-UPDATE-MASTER-RECS
10	PRO-WRITE-UPDATE-MASTER-RECS
11	PRO-COMPUTE-AMT-ORDER-PER-PART
12	PRO-SUMMARIZE-PER-PART-NO
13	PRO-WRITE-OUT-OF-ORDER-PARTS

Figure 4 Data flow diagram for a master inventory file updating design problem

the files and databases (keys, size of records, number of records, etc.) and the precedence relationship between processes. The system provides information on each process, the data set(s) used by the process as input or output and the processes which are linked to each process. A separate screen shows the characteristics of the data sets used by the processes.

Further, it shows the matrix of the process to data set relationship (incidence graph) and computes the total transport volume of data within the system. Figures 5 through 8 depict the information associated with the process for updating master inventory.

CAPO computes the volume of data transported between processes, the precedence matrix (process to process relationships), the reachability matrix, matrix of partial reachability, matrix of feasible grouping, matrix of probability of control transfer between processes (probability matrix), and the required invocation time of each process with respect to the other processes. Using the above matrices and the weighting scheme, the interdependency weight matrix is computed. As a result, the process-graph is transformed to a weighted down graph, shown in Figure 9.

The interdependency weight matrix is used to produce a measure of similarity between the nodes. The core set concept is suitable as a similiarity measure since the "closeness" of a node (say X) to any other node (y) in the graph is a function of a) the number of nodes which are connected to node (X) and b) the strength of the connection of node (X) to any other node in the graph (including Y). The similiarity matrix is presented in Figure 10.

The derived similarity matrix is used as input to six different hierarchical clustering methods. These are Single Linkage Clustering, Complete Linkage Clustering, Median Method of Gower, Centroid Sorting, Average Linkage within the New Group, and Average Linkage between the Merged Groups. There is no obvious way of determining which clustering method would produce the best partition with respect to the objective function. All of the algorithms are included in the CAPO analysis workbench package and the analyst may apply whichever one he chooses, or all six. Table 1 shows the clustering results for the example.

After each stage of clustering in each of the above methods, the measure (M) is computed. This measure quantifies the extent of coupling and strength within the clusters in the system Comparing the six clustering methods, the single linkage clustering produced the best overall decomposition, with an objective function value of $M = 5.253$. The objective function values for the Complete Linkage Clustering, Average Linkage within New Group, and Average Linkage Between the Merged Groups equal 4.919. The remaining two methods also produced the similar decomposition with the objective function value of 4.450.

The clustering methods provide a collection of partitions ranging from each process as a cluster to one cluster involving all the processes as members. In order to compare the different partitions and investigate the sequence in which the clusters are formed, the information generated by the clustering algorithms are used to draw hierarchical trees. Trees provide an effective visual aid of the clustering results which permit the analyst to grasp the hierarchical relationships and visualize the membership of each cluster at any level of aggregation. In order to draw the tree, the range of the criterion is divided into 25 equal segments and all merges whose criterion values fall within a

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	1	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	1	-1	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	1	-1	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	1	-1	-1	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	1	-1	0	0	0	0	0	0	0	0
7	0	0	0	1	0	0	0	1	0	0	-1	-1	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	1	0	-1	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	1	0	-1	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	1	-1	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	-1	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	-1	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1

FILE NO.	1	2	3	4	5	6	7	8	9	10
NUM ACCES	1	2	2	2	1	1	2			
FLSIZE/1000	76.000	76.000	15.200	60.800	15.200	1210.000	1210.000	1064.800	24.200	24.200
VOL/1000	76.000	152.000	30.400	121.600	15.200	1210.000	2420.000	2129.600	48.400	24.200

FILE NO.	11	12	13	14	15	16	17	18
NUM ACCES	2	4	1	2	1	2	1	1
FLSIZE/1000	9.800	90.750	9.800	90.750	90.750	72.600	72.600	18.150
VOL/1000	7.600	363.000	9.800	181.500	90.750	145.200	72.600	18.150

TOTAL TRANS. VOLUME 7110.000

Figure 5. Incidence Graph of Processes

1	-1	1	-1	0	0	0	-1	-1	-1	-1	-1	-1	-1
2	1	-1	1	0	2	0	1	-1	-1	-1	-1	-1	-1
3	-1	1	-1	0	0	0	2	0	0	0	0	0	0
4	0	0	0	-1	1	-1	-1	-1	-1	-1	-1	-1	-1
5	0	2	0	1	-1	1	1	-1	-1	-1	-1	-1	-1
6	0	0	0	-1	1	-1	2	0	0	0	0	0	0
7	-1	1	2	-1	1	2	-1	1	1	-1	1	-1	1
8	-1	-1	0	-1	-1	0	1	-1	2	0	2	0	2
9	-1	-1	0	-1	-1	0	1	2	-1	1	2	0	2
10	-1	-1	0	-1	-1	0	-1	0	1	-1	0	0	0
11	-1	-1	0	-1	-1	0	1	2	2	0	-1	1	2
12	-1	-1	0	-1	-1	0	-1	0	0	0	1	-1	0
13	-1	-1	0	-1	-1	0	1	2	2	0	2	0	-1

Figure 6 Matrix of feasible grouping

1	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	0.00	0.00	1.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.00	1.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00	0.00	1.00	0.00	1.00
8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
11	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
12	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
13	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Figure 7 Probability matrix

1	0.0	0.9	0.0	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.9	0.0	0.9	0.0	0.7	0.0	0.9	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.9	0.0	0.0	0.0	0.3	0.7	0.0	0.0	0.0	0.0	0.0	0.0
4	0.3	0.0	0.0	0.0	0.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.7	0.0	0.9	0.0	0.9	0.9	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.3	0.0	0.9	0.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.9	0.7	0.0	0.9	0.7	0.0	0.9	0.9	0.0	0.9	0.0	0.9
8	0.0	0.0	0.0	0.0	0.0	0.0	0.9	0.0	0.7	0.0	0.7	0.0	0.7
9	0.0	0.0	0.0	0.0	0.0	0.0	0.9	0.7	0.0	0.9	0.7	0.0	0.7
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.9	0.0	0.0	0.3	0.0
11	0.0	0.0	0.0	0.0	0.0	0.0	0.9	0.7	0.7	0.0	0.0	0.9	0.7
12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	0.9	0.0	0.0
13	0.0	0.0	0.0	0.0	0.0	0.0	0.9	0.7	0.7	0.0	0.7	0.0	0.0

Figure 8 Interdependency weight matrix

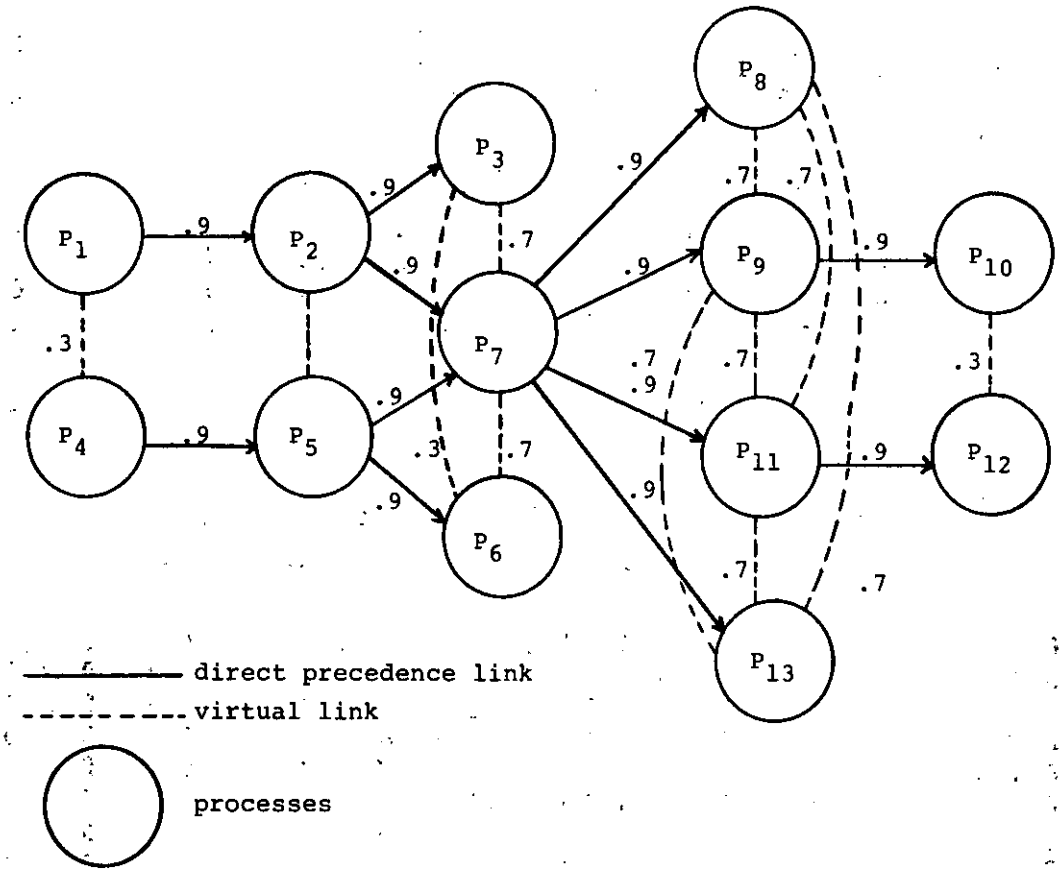


Figure 9. Resulting Weighted Directed Graph

given segment of the range are treated as having occurred together.

The task that remains, then, is to study the decomposition, to formulate a set of specifications for structuring modules required to implement the design. At the same time, effort should be concentrated on identification of anomalies, counterintuitive results, etc., that might indicate errors in assessments of interdependency weights.

ANALYSIS OF CLUSTERING RESULTS

All the clustering routines started by grouping two process (7) and (11) as their first cluster. The reason being that the two processes have the highest level of similarity measure, .518, (see the similarity matrix above). The high level of the similarity measure is the result of their interdependency weight, the number of processes which are connected to them and their interdependency link weight. At the next stage of clustering, processes 9 or 8 are added to the same cluster. Note that, by adding more processes to the same module, the structure of module has changed from sequential cohesion to a communicational cohesion. Going one step higher on the hierarchy tree, processes (13), (2), and (5) are also added to the same cluster. Based on method one, the resulting module has now processes (2, 5, 7, 9, 11, 13).

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	0.271	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.128	0.298	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4	0.125	0.189	0.128	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
5	0.189	0.340	0.218	0.271	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
6	0.128	0.218	0.182	0.128	0.298	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
7	0.329	0.497	0.418	0.329	0.497	0.418	1.000	0.000	0.000	0.000	0.000	0.000	0.000
8	0.173	0.263	0.202	0.173	0.263	0.202	0.481	1.000	0.000	0.000	0.000	0.000	0.000
9	0.210	0.300	0.239	0.210	0.300	0.239	0.518	0.344	1.000	0.000	0.000	0.000	0.000
10	0.086	0.177	0.115	0.086	0.177	0.115	0.317	0.160	0.279	1.000	0.000	0.000	0.000
11	0.210	0.300	0.239	0.210	0.300	0.239	0.518	0.344	0.381	0.198	1.000	0.000	0.000
12	0.086	0.177	0.115	0.086	0.177	0.115	0.317	0.160	0.198	0.074	0.279	1.000	0.000
13	0.173	0.263	0.202	0.173	0.263	0.202	0.481	0.308	0.344	0.160	0.344	0.160	1.000

Figure 10 Similarity matrix of process

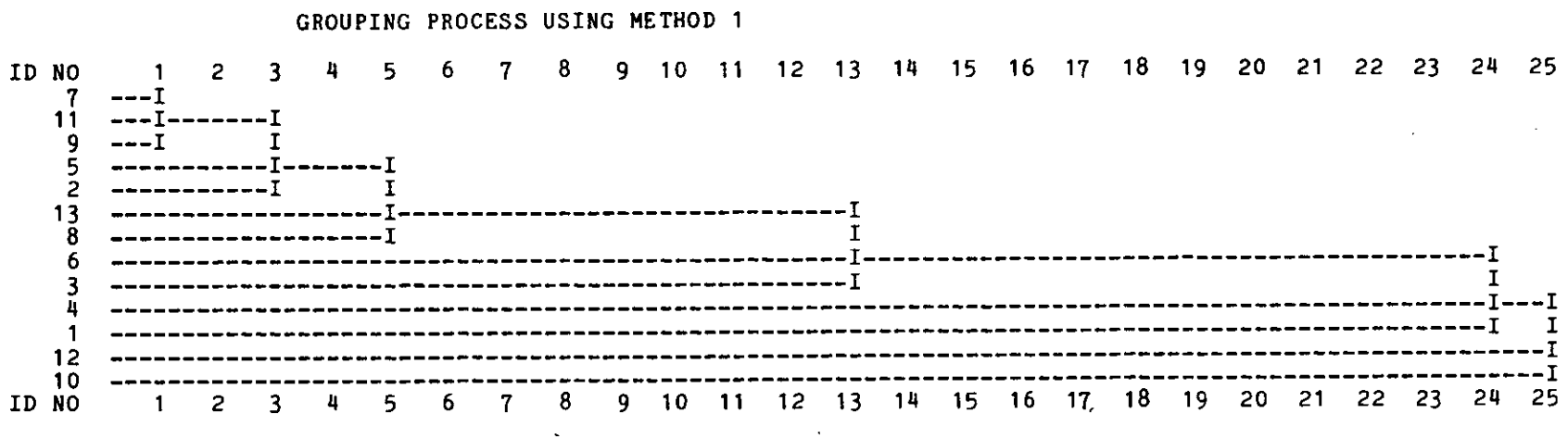


Figure 11 Hierachical clustering tree for single linkage clustering method

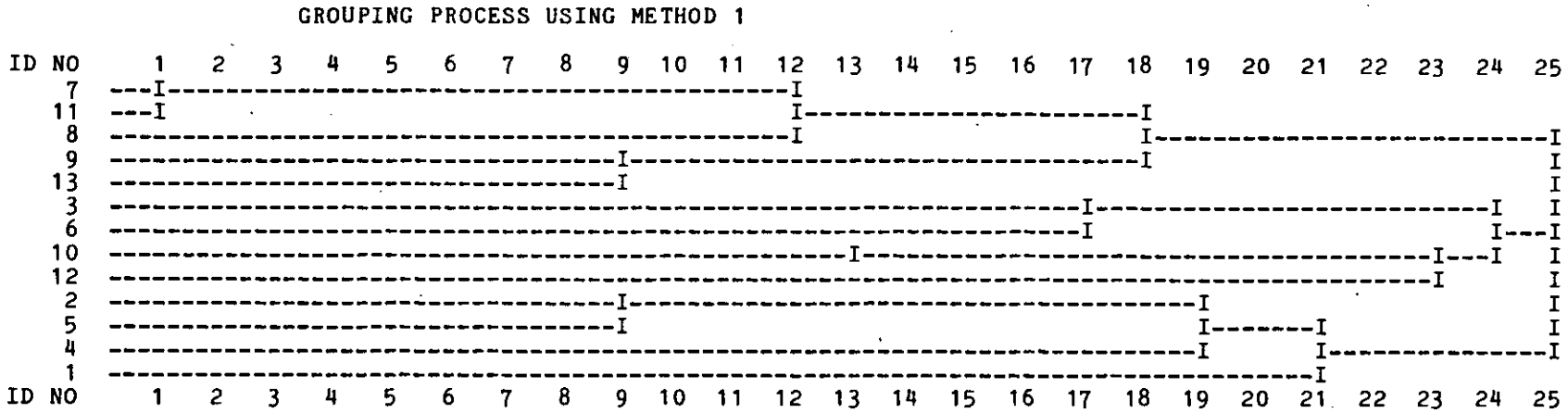


Figure 12 Hierachical clustering tree for average linkage within the merged grouped method

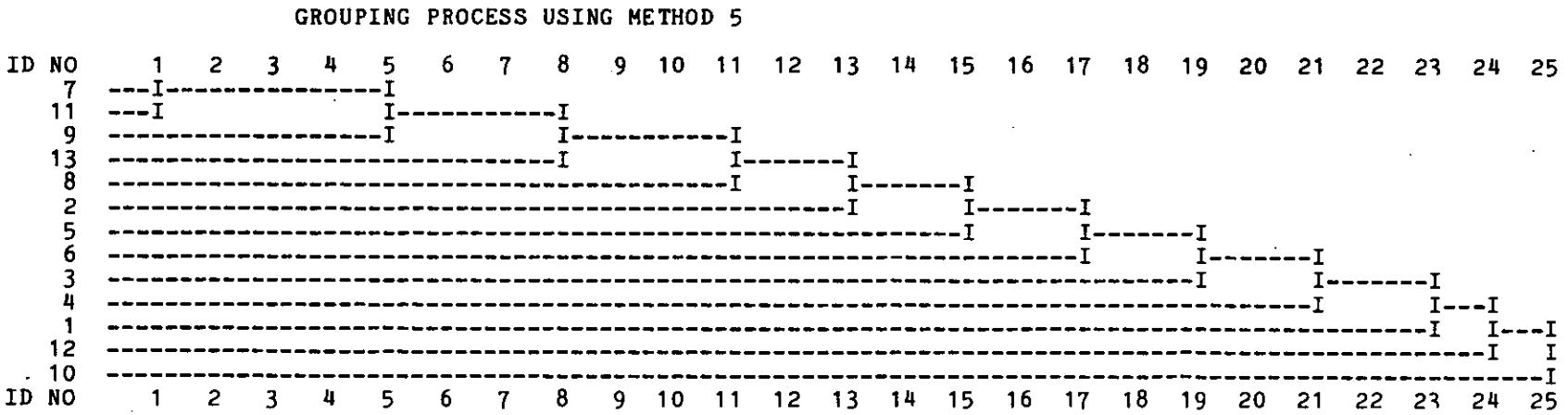


Figure 13 Hierachical clustering tree for average linkage between the merged grouped method

Table 1. Comparison of Clustering Results of six Decompositions Algorithms

Method			
*Single Linkage Clustering	5.252	8	(1), (2, 5, 7, 9, 11, 13) (3). (3). (4). (6), (8), (10), (12)
*Complete Linkage *Average Linkage Within New Group *Average Linkage Between Merged Groups	4.919	7	(1), (2, 5, 7, 8, 9, 11, 13) (3). (4), (6), (10), (12)
*Centroid Clustering *Median Method	4.450	7	(1), (2, 5), (3, 6), (4), (7, 8, 9, 11, 13), (10), (12)

Here, process (7) has a communicational cohesion on the input side with processes (2) and (5) and communicational cohesion on the output side with three processes (9), (11), (13).

As mentioned above, all the processes eventually will be collapsed in a single cluster. The objective function of design (low coupling and high strength) is used to determine the optimum design. The criteria for optimality once again is the measure M. The higher the M, the higher the strength and the lower the coupling in the system. If there is a constraint with regard to the size of the module, the constraint is used as a stopping criteria.

The level of transport volume is also used as an alternative criteria to aid the designer in selecting from among alternative designs. Where two designs have a very close M value, the design that has lower transport volume would be preferred as the candidate design for the system.

CONCLUSION

The result of the analysis is an effective modularization of the system of processes and a specification of program modules for either manual (programming) or automatic code generation (Konsynski, 1981), each of which has significant and differing influence on weighting the properties.

The analysis reveals inherent organization among the set of processes and imposes organization on borderline cases. This eliminates the problems that often arise from prior knowledge or bias on the part of the analyst.

Thus, the analyst is aided in objectivity of design.

Extensions of the overviewed techniques and further design applications are being developed. These extensions include applications in query processing, database creation and reorganization, code generation, system testing, project management, storage allocation, retrofit and portability, and documentation.

REFERENCES

Andreu, R. "A Systematic Approach to the Design and Structuring of Complex Software Systems," Ph.D. Dissertation, Sloan School of Management, M.I.T., Cambridge, Massachusetts, February 1978.

Anderberg, M.R. Cluster Analysis for Applications, Academic Press, New York, New York, 1973.

Balazer, R.M. "Imprecise Program Specifications," USC Information Science Institute Research Report RR-75-36, December 1975.

Boehm, B.W. "Software and its Impact: A Quantatative Assessment," Datamation, Volume 14, Number 5, May 1973, pp. 48-59.

Boehm, B., et al. "Characteristics of Software Quality," TRW Technical Report, TRW-SS-73-09, 1973.

Boehm, B.W. "Software Engineering-As It Is," Proceedings of 4th International Conference on Software Engineering, September 1979, Germany, IEEE Catalog No. 79CH1479-SC.

Bohem, B.W., et al. "Some Experience with Automated Aids to the Design of Large-Scale Reliable Software," IEEE Transaction on Software Engineering, March 1975, pp. 125-133.

Dijkstra, E.W. A Discipline of Programming, Prentice-Hall, New York, New York, 1976.

Dunsmore, H.E. Influence of Programming Factors on Programming Complexity, Ph.D. Dissertation, University of Maryland, 1979.

Estabrook, G.F. "A Mathematical Model in Graph Theory for Biological Classification," Journal of Theoretical Biology, Number 12, 1966.

Ferrari, D. "Improving Locality by Critical Working Sets," Communications of the ACM, Volume 17, Number 11, November 1974.

Ferrari, D. "The Improvement of Program Behavior," Computer, November 1976, pp. 39.

Gelperin, D. Testing Maintainability, Software Engineering Notes 4: 7-12, ACM-SIGSOFT, April, 1979.

Gotileib, C. and Kumar, S. "Semantic Clustering of Index Terms," Journal of the ACM, Volume 15, Number 4, October 1968.

Hartigan, J. Clustering Algorithms, John-Wiley, New York, New York, 1975.

Henry, S.M. "Information Flow Metrics for the Evaluation of Operating Systems' Structure," Ph.D. Dissertation, Iowa State University, 1979.

Horning, J. and Randell, B. "Process Structuring," Computing Surveys, Volume 5, Number 1, March 1973.

Huff, S.I. "Decomposition of Weighted Graph Using The Interchange Partitioning Algorithm," Technical Report No. 8 , Center for Information Systems Research, Sloan School of Management, M.I.T., Cambridge, Massachusetts, March 1979.

Huff, S.L. and Madnick, S.E. "Analysis Techniques for Use With the Extended SDM Model," M.I.T. Sloan School Technical Report Number 9, February 1979, Cambridge, Massachusetts.

Jackson, M.A. Principles of Program Design, Academic Press, New York, New York, 1975.

Karimi, J. "Computer-Aided Process Organization," Ph.D. Dissertation, University of Arizona, May 1983.

Konsynski, B.R. and Nunamaker, J.F., Jr. "A Generalized Model for Computer-Aided Process Organization in Design of Information Systems," Proceedings of the Sixth Pittsburgh

Conference on Modeling and Simulation, April 1975.

Konsynski, B.R. "A Model of Computer-Aided Definition and Analysis of Information System Requirements," Ph.D. Dissertation, Purdue University, December 1976.

Konsynski, B.R. "Quantitive Factors in Software Design," Western American Institute of Decision Science, April 1977.

Konsynski, B.R. and Nunamaker, J.F. "PLEXSYS: A System Development System," Advanced System Development/Feasibility Techniques, eds., J.D. Couger, M. Coulter, and R. Knapp, Wiley and Sons, 1981, pp. 399-423.

Konsynski, B.R. "Information Engineering in Enterprise Analysis," MIS Technical Report, 1982.

Kottemann, J. and Konsynski, B. "Complexity Assessment: A Design and Management Tool for Information System Development," Information Systems, Volume 4, 1984, pp. 12.

Lowe, T.C. "Analysis of Boolean Program Models for Time Shared Paged Environments," Communications of the ACM, Volume 12, Number 4, April 1969.

Lowe, T.C. "Automatic Segmentation of Cyclic Program Structures Based on Connectivity and Processor Timming," Communications of the ACM, Volume 12, Number 1, January 1970.

Lowe, T. "Analysis of an Information System Model with Transfer Penalties," IEEE Transactions on Computers, Volume C-22, N5, May 1973.

Madnick, S.E. and Huff, S.L. "An Extended Model for a Systematic Approach to the Design of Complex Systems," Technical Report No. 7, Sloan School of Management, M.I.T. (NTIS NO. A058565), Cambridge, Massachusetts, July 1978.

Myers, G.J. "Characteristics of Composite Design," Datamation, Volume 19, September 1973, pp. 100-102.

Myers, G. Reliable Software Through Composite Design, Petro-Cell Charter, New York, New York, 1975.

Myers, G. "The Need for Software Engineering," Computer, February 1978.

Nunamaker, J., Nylin, W. and Konsynski, B., "Processing Systems Optimization Through Automatic Design and Reorganization of Program Modules," Tou, Information Systems, Plenum, 1974.

Nunamaker, J. and Konsynski, B. "From Problem Statement to Automatic Code Generation," Systemeering 75, Student Litteratur, Lund, Sweden, 1975.

Nunamaker, J., Ho, T., Konsynski, B. and Singer, C. "Computer Aided Analysis and Design of a Financial Information System," Communications of the ACM, 1976.

Parnas, D.L. "On the Criteria to be Used in Decomposing Systems into Modules," Communication of ACM, Volume 15, Number 12, December 1972, pp. 1053-1058.

Stevens, W.P., Myers, G.I., and Constantine, L.L. "Structured Design," IBM Systems Journal, Volume 13, Number 2, May 1974, pp. 115-139.

Teichroew, D. "Improvements in the System Life Cycle," Proceedings IFIPS, 1974.

Teichroew, D. "A Survey of Languages for Stating Requirements for Computer-based Information Systems," Proceedings AFIPS, 1972, FJCC, pp. 1203-1244.

Warnier, J.D. The Logical Construction of Programs, 3rd ed., Trans. B.M. Flanagan, Van Nostrand Reinhold, New York, New York, 1975.

Wasserman, A.I. "On the Meaning of Discipline in Software Design and Development" In Tutorial on Software Design Techniques, P. Freeman and A.I. Wasserman, eds., IEEE Catalog No. 76CH1145-2C.

White, J.R. and Booth, T.L. "Toward an Engineering Approach to Software Design," Proceedings Second International Conference on Software Engineering, 1976, IEEE Catalog No. 76CH1125-4C.

Whitiney, D.E. and Milley, M.K. "CADSYS: A New Approach to

Computer-Aided Design," IEEE Transactions on Systems, Man, and Cybernetics, 4-1 January 1974.

Yourdon, E. and Constantine, L.L. Structured Design Fundamentals of a Discipline of Computer Program and Systems Design, Prentice-Hall, Englewood Cliffs, New Jersey, 1979.

Willis, R. R. and Jensen, E.P. "Computer Aided Design of Software Systems" Proc. 4th. International Conference on Software Engineering, September 17-19, 1979, Munich, Germany, IEEE Catalog No. 79CH1479-SC.