An Analysis of Several Models of System Structure.

Robert V. Binder

EECS 481
Professor Tsai
Spring 1988

This paper is an examination of several models for analyzing the structure of abstract systems. There are four main parts: (1) the problem of abstract system assessment is discussed in the Introduction, (2) some relevant features of the Data Flow model are summarized, (3) four quantitative models of hierarchic decomposition are presented, (4) these models are analyzed, and (5) some general conclusions are drawn.

I. Introduction.

The abstract systems of interest are software system designs. Since the primary use of these abstract systems is to facilitate the implementation of operable software systems, we will examine their utility as tools for the designer of a software system. Our discussion will be limited to abstract system models, and will not address issues related to the behavior of implementations of an abstract system model.

The creation of a software system design is an intellectual process. Hierarchic organization is a means to manage the complexity of a large collection of components of a solution. The components of a solution may be found by a refinement process, which starts with a general statement of a goal and proceeds by identifying means to attain the goal. These means will contain sub-goals that may require additional refinement. Refinement continues until a suitable level of detail has been reached. There are at least two conditions for terminating the means-end refinement.

The first condition obtains when an elemental level of description has been reached. Further refinement becomes meaningless within the context of the problem. The second condition is that the refinement has descended to an axiomatic level: a representation has been

found that is sufficiently similar to some proven, trusted, or self-evident transformation or structure. The representation can be assumed without further analysis.

We would like to have some means to verify the decomposition of a problem and the associated hierarchy of the solution model for several reasons. First, a good decomposition will result in a good implementation: lower development cost, fewer defects, and lower maintenance costs. This is the essential observation made by Yourdon and Constantine in their discussion of the rationale for Structured Design.[1] Second, we would like an objective means to assess the goodness of the decomposition, which would provide answers to questions like: Does the collection of elemental components still correspond to the original problem? Assuming some criterion for partitioning, for example, minimize complexity of the interfaces, maximize "functional cohesion", how well has the model been constructed? Objective analysis of the abstract model then serves two purposes: it would provide a tool for developing the abstract system model, and would facilitate general improvement in the resulting software.

We would like to make this assessment without heavy reliance on subjective, qualitative considerations. Ideally, the abstract system would be analyzed by quantitative models with little or no reliance on subjectively scaled parameters. Subjective evaluations are useful and necessary, but are often skewed by lack of agreement on appropriate criteria, personal stylistic preferences, bias motivated by non-technical considerations, and cognitive lapses.

II. Heuristic Models of Hierarchic Decomposition.

The abstract system models under consideration are generally known as Functional Decomposition Data Flow Diagrams. There are many variants of the notation and semantics of

[1] Edward Yourdon and Larry L. Constantine, Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design (Englewood Cliffs: Prentice-Hall, Inc., 1979), see chapters 1 and 5.

the Data Flow Diagram model. We will use the widely accepted DeMarco notation and semantics in this paper.

A Data Flow Diagram model consists of a network of data transformation processes and data stores (graph vertices), connected by data flows (graph edges). This network is partitioned into subgraphs to gain improved understanding of the design problem. The subgraphs are represented as a composite processes, at successively greater degrees of abstraction until the system is represented as single process accepting input from external sources and delivering output to external sinks. This is the "top" level contains a single process and is called the Context Level diagram. In practice, the Context level is often the first part of the model prepared, and lower levels are produced by successive refinement. This general strategy is often called "top-down" development.

Some of the heuristics suggested for top-down development of a Data Flow Diagram are:

> [1] Listen to the data -- if it cries out to be distributed in some fashion, allow it to affect your partitioning. A partitioning that is in this sense "natural" is often easier to understand than one that stays within artificial limits. [2] Partition to create conceptually meaningful interfaces. [3] Expect the extent of partitioning (number of bubbles, for instance) to be somewhat greater at the very top level. ... [4] At any level, partition as much as you can without hurting readability. The more you partition, the fewer diagrams you will have to use ... [5] If you need an artificial limit, use seven; diagrams partitioned into approximately seven pieces are usually workable. [6] Diagrams that are clear are better than diagrams that are not, regardless of "absolute" partitioning rules.[2]

Some guidelines for refinement of a model derived by this approach are suggested as well:

> 1. Build an expanded diagram by combining all of the children of the diagram that needs repartitioning. Connect the interfaces. ... 2. ... divide [the expanded diagram] into a workable number of subsets within minimal interfaces among them. Your decision to include a given bubble in one set or another should be made purely on topological grounds -- in other words, to minimize interest communication. 3. Recreate the upper level by allocating one bubble for each set. Copy the inter-set interfaces from the expanded diagram onto the upper level. [3]

---

[2] Tom DeMarco, Structured Analysis and System Specification (Englewood Cliffs: Prentice-Hall Inc., 1978), pp. 82-83.

[3] DeMarco, op. cit., pp. 115-116.

It is evident from these succinct and effective heuristics that the process of top-down decomposition is highly subjective. If practiced by skilled designer, a model of relatively high utility will result. If performed by an amateur, the model will be of limited use for design validation or as a implementation specification.

III. Quantitative Models of Hierarchic Decomposition.

MODEL 1: ALEXANDER'S DESIGN PROGRAM.

Our first model was developed in a theoretical study of methods for Architectural design and Urban Planning by Alexander.[4] The model provides a means for quantitative analysis and hierarchic partitioning of the requirements for a building (or any object), the relationship between requirements, and an algorithm for creating a n-level design hierarchy from the requirements. The purpose of the model is identify groups of requirements that have the most interaction, so that they can analyzed and designed as a unit.

The designer begins by stating the qualitative requirements in a consistent fashion, and then identifies dependencies among the requirements. The result of this analysis is represented as a graph. Each vertex represents a requirement, each edge represents a dependency between requirements. The requirements:

> ... must be chosen (1) to be of equal scope, (2) to be as independent of one another as is reasonably possible, and (3) to be as small in scope and hence as specific and detailed and numerous as possible.[5]

With regard to the dependencies between requirements:

> Roughly speaking, two requirements interact (and are therefore linked), if what you do about one of them in a design necessarily makes it more difficult or easier to do anything about the other. [6]
>
> The search for causal relations of this sort cannot be mechanically experimental or statistical; it requires interpretation: to practice it we must adopt the same kind of common sense that we have to make use of all the time in the inductive part of

---

[4] Christopher Alexander, Notes on the Synthesis of Form (Cambridge: Harvard University Press, 1964). Some substitution of variable names has been made.

[5] Alexander, op. cit., p. 115.

[6] Alexander, op. cit., p. 107.

science. [7]

Each dependency is assigned an integer weight. A weight may take the value -1 or 1. A negative weight indicates mutual exclusion. A positive weight indicates that both requirements must be met.

The design problem is then represented as a graph G(M,L), with a weight matrix V.

The major elements of the model are as follows:

M A set of requirements, or "design attributes", each stated so that it be decided whether it is met by a given design or not. Each element of M is a vertex in a graph G.

m_i A requirement, i = 1,N

L A set of associations on requirements, or "links", that represents the dependency of one requirement on another. Each element of L is an edge in a graph G.

l_{ij} A link between m_i and m_j.

Z Number of links, or number of edges in G(M,L).

v_{ij} A weight on l_{ij}, that represents the dependency of requirement m_i and mj. If this weight is zero, no dependency exists. If positive, the requirements must be met jointly. If negative, the requirements are mutually exclusive.

S An arbitrary subgraph of G.

P A partition of M, consisting of a set of disjoint subgraphs, S, of M.

I(P) Interface count for partitioning P, equal to the total number of edges spanning two subgraphs in P.

R(P) An index of the goodness of decomposition of a given P.

[7] Alexander, op. cit., p. 109.

A proof is offered that G(M,L) and V are sufficient to compute the Shannon information entropy on the entire system, **H(M)**, and on any subset **H(S)**[8]. Then,

> ... we may use the difference [∑ H(S)] - H(M) as a measure of the strength of the connections severed by the partition P. The larger it is, the weaker the connections are, and the less information transfer there is across the partition.[9]

After some additional transformations to account for redundant edges, and to provide computational tractability, the goodness of decomposition index, based on the Shannon Entropy, is given as:

$$R(P) = \frac{0.5N(N-1)\sum V_{ij} - ZI(P)}{\sqrt{I(P)(0.5N(N-1) - I(P)}}$$

The partition P that minimizes R(P) over all possible P is argued to be the best decomposition, since amount of information transfer across the partitions is minimized. Two algorithms for computing the minimal R(P) are given:

> To find the best partition of a set **S**, we use a hill-climbing procedure which consists essentially of taking the partition into one-element subsets, computing the value of R(P) for this partition, and then comparing with it all those partitions which can be obtained from it by combining two of its sets. Whichever of these partitions has the lowest value of R(P) is then substituted for the original partition; and the procedure continues. It continues until it comes to a partition whose value of R(P) is lower than that of any partition which can obtained from it by

---

[8] The Shannon Entropy is a measure of the number of bits contained in a given message. This measure can be derived for any observed frequency distribution over a finite set of variables or from a continuous probability distribution. Since not all elements of M are assumed to be necessary, Alexander argues that the weights on G(M,L) applied to each possible subset of M (there are 2^m subsets) yields the probability distribution necessary for calculating the Shannon Entropy:

$$H(M) = N \log_2 + (d^2/2) > V_{ij}^2$$

d is a scaling constant in the zero/one range. See Alexander, op. cit., pp. 174-189.

[9] Alexander, op. cit., p. 187.

combining two sets.

> Another hill-climbing procedure, which finds a tree of partitions directly, goes in the opposite direction. It starts with whole set **S**, and breaks it into its two most independent disjoint subsets, by computing R(P) for a random two-way partition, and improving the partition by moving one variable at time from side to side, until no further improvement is possible. It then repeats this process for each of the two subsets obtained, breaking each of them into two smaller subsets, and so on iteratively, until the entire set **S** is decomposed.[10]

MODEL 2: BSP/INFORMATION SYSTEM MODEL.

The second model is used in developing a general design for all of the data processing systems in a large organization. This approach was developed by IBM Corporation and is called Business Systems Planning (BSP). A BSP study identifies the major processes of an organization and the major categories of data that are created or used by these organizational processes.

An important part of BSP analysis is to identify data processing subsystems. This is done by looking for clusters of creating and using processes. The result of this clustering is a two level hierarchy. Figure 1 gives an example of an unclustered model. Figure 2 shows how the same model might be clustered into sub-systems.

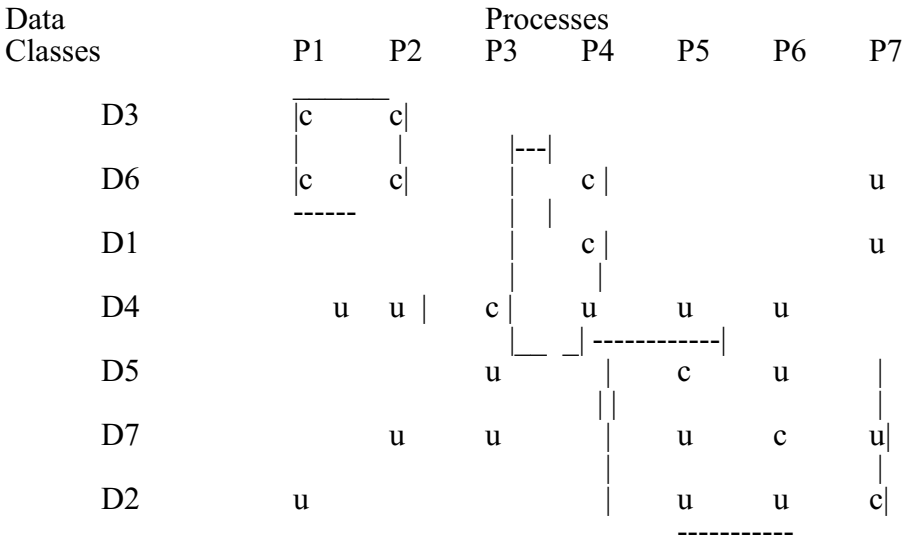
[10] Alexander, op.cit., pp. 190-191.

Figure 1. Data Class/Business Process Matrix.

Data Classes	P1	P2	P3	P4	P5	P6	P7
			Processes				
D1				c			u
D2	u				u	u	c
D3	c	c					
D4		u	u	c	u	u	u
D5			u		c	u	
D6	c	c		c			u
D7		u	u		u	c	u

u: Process n Uses Data Class m c: Process c Creates Data Class m

Figure 2. Sub-system Clustering.



As BSP was originally conceived, subsystem clustering was performed by visually grouping creating processes. No algorithm was provided to accomplish this. IBM has recently developed a program product to manage the large quantity of data typically produced in a BSP study. This tool is called ISMOD (Information System Model). ISMOD provides some improved means for identifying the subsystem clusters.[11]

ISMOD requires that a detailed data model be developed along with the process model. The Creating/Using processes are identified for each data element. This allows the access frequency between processes to be tabulated. The tabulation of access frequency is used to calculate indices of connectivity between processes. The basic variables are:

P_i A Process, i = 1,n

L_i The number of Local data elements in P_i. A Local data element is created and used within a process. No other process creates or uses these data elements.

G_i The number of Global data elements in P_i. A Global data element is created and used over more than one process.

The indices of connectedness are calculated from these variables. These indices may be calculated on a single Process, or between any two Processes.

I_{ij} Isolation Ratio. This is the ratio of Local to Global data elements:

$I_{ij} = L_{ij} / L_{ij} + G_{ij}$

An Isolation ratio of 0.0 means that a process is totally dependent on other processes. An Isolation ratio of 1.0 means that a process is completely independent of other processes.

[11] K. Peter Hein, "Information System Model and Architecture Generator," IBM Systems Journal, v. 24, n. 3/4, 1985: 213-235. Some substitution of variable names has been done.

IM_{ij} Isolation Mean. This is the average Isolation ratio for any two process P_i and P_j:

$IM_{ij} = I_i + I_j / 2.0$

IE_{ij} Isolation Extended. This is the Isolation ratio that would result when any two process P_i and P_j are combined causing some previously Global data items become Local to the new Process.

DI_{ij} Difference in Isolation Ratio. This is the difference in the Isolation Ratio that would result if any two process P_i and P_j were combined. This value represents the extent that process P_i and P_j have shared data elements:

$DI_{ij} = IE_{ij} - IM_{ij}$

Sub-systems are composed by collecting processes with relatively high values of interaction. Conversely, processes with low interaction (a high Isolation value) might be candidates for decomposition. However, no explicit heuristic or algorithm is provided to accomplish this:

> If the interaction [between processes] is high, two processes can be looked upon as a couple and hence part of the same subsystem. If the interaction is low, the two would not be considered to be ideal for common grouping.[12]
>
> The determination as to subsystem assignment should not be made on the basis of the [indices] alone, but should also require some understanding of the process and actual data used to determine whether the coupling and clustering suggested by the [indices] is sensible.[13]

MODEL 3: DESIGN ASSISTANT MODEL.

[12] Hein, op. cit., p. 226.

[13] Hein, op. cit., p. 227.

Our third model was developed to support an automated design system CAPO (Computer Aided Process Organization). This system:

> ... would not replace the designer, but rather would support the design activities and provide a unified approach to the design process. The tool should also provide a quantitative measure of design quality in order to facilitate the design evaluation by the analyst. The measure of the goodness should be derived from the degree to which a particular design satisfied the design heuristics.[14]

The analysis begins with a Data Flow Diagram. The relevant attributes of the Data Flow Diagram are represented in six matrices and several variables as follows:

P_i A process, i = 1,N

f_j A data store, j = 1,k

E Incidence Matrix. This relates the ith Process to the jth Data Store as follows:

e_{ij} = 1, where f_j is an input to P_i

$_{eij}$ = 0, where f_j is not connected to P_i

e_{ij} = -1, where f_j is an output of P_i.

P Precedence Matrix. This relates the ith Process to the jth Process as follows:

P_{ij} = 1, where P_i is a direct Precedent of P_j.

$_{Pij}$ = 0, where P_i is a not direct Precedent of P_j.

R Reachability Matrix. This relates the ith Process to the jth Process as follows:

R_{ij} = 1, where P_i is precedent of P_j, by a path of any length.

$_{Rij}$ = 0, where P_i is never a precedent of P_j, by a path of any length.

---

[14] J. Karimi and B. R. Konsynski. "An Automated Software Design Assistant", *IEEE Transactions on Software Engineering*, February 1988, v.14, n.2, p 197.

R^* Partial Reachability Matrix. This relates the ith Process to the jth Process as follows:

R^{*i}_j = 1, where P_i is precedent of P_j, by at least two paths of any length.

R^{*i}_j = 0, where P_i is never a precedent of P_j, by a more than one path of any length.

G Feasible Process Pair Grouping Matrix. This relates the ith Process to the jth Process as follows:

G_{ij} = 2, P_i and P_j share data files and can be merged.

G_{ij} = 1, P_i and P_j have a direct precedence link and can be merged.

G_{ij} = 0, P_i and P_j do not have a direct precedence link and can be merged.

G_{ij} = -1, P_i is precedent of P_j, by at least two paths of any length, i.e., R^{*i}_j is non-zero.

T Timing Relationship Matrix. This relates the ith Process to the jth Process as follows:

T_{ij} = 1, P_i and P_j are active in the same time interval.

T_{ij} = 0, P_i and P_j are not active in the same time interval.

With the Data Flow Diagram represented in the above matrices, a relative measure of the strength of each link (a weight) can be computed and stored in the weight matrix.

W Interdependency Matrix. This relates the ith Process to the jth Process as follows:

W_{ij} = 0.9, if G_{ij} = 1 (Sequential Cohesion assumed).

W_{ij} = 0.7, if G_{ij} = 2, and e_{i1} = 1 and e_{j1} = 1 shared data stores (Communicational Cohesion assumed).

W_{ij} = 0.5, if G_{ij} = 2, and $e_{i1} \neq 1$ or $e_{j1} \neq 1$, then common input data flow but no

shared data store (Procedural Cohesion assumed).

W_{ij} = 0.3, if $G_{ij} = 0$, and $T_{ij} = 1$ simultaneous activation but no other linkage (Temporal Cohesion assumed).

W_{ij} = 0.0, if $G_{ij} = 0$, and $T_{ij} = 0$, no common relationship (Coincidental Cohesion assumed).

The Design Assistant Model provides for a identification of subgraphs, ranking of the subgraphs, and an algorithm for creating a complete hierarchy. This procedure moves in a "bottom-up" manner. Primitive level processes are examined for "similarity" by a graph theoretic measure, and then grouped into subgraphs. The subgraphs are in turn checked for similarity. Each subgraph is evaluated for goodness of partitioning.

The identification of subgraphs is accomplished by ranking processes according to their "similarity". This measure of similarity will tend to generate subgraphs where there is a relatively high number of connections. The number of connections as a selection criterion is augmented or diminished by the weights calculated for each connection. A "goodness" index for each partition is computed for each subgraph.

The cells of a matrix Z contain the Similarity index between any two processes P_i and P_j, computed as follows:[15]

$$Z_{ij} = (|X| / |Y|)(U_{ij}^2 / V_{ij})$$

Where:

X is the set of Processes connected to either P_i or P_j, including P_i and P_j.

[15] Karimi and Konsynski, op. cit., p. 200. I have made some substitutions in the variable names to clarify the author's confusing and redundant presentation.

Y is the set Processes connected to both P_i and P_j, including P_i and P_j.

U_{ij} is the average weight on edges in the set X.

V_{ij} is the average weight on edges in the set Y.

The Z_{ij} are used to identify candidate subgraphs in the following manner:

> ... the procedure for ... hierarchical clustering [proceeds by viewing] each node as separate cluster. ... The similarity matrix [Z] is searched to find the most "similar pair of" clusters. ... The cluster pair with the largest similarity value is then merged to form a single cluster, producing the next level up in the clustering tree. The joining of similar pair [sic] is continued until the number of clusters is reduced to 1 (the entire set of objects). ... the goodness measure [is calculated after each cluster is created] and the information is recorded in order to find a particular decomposition exhibiting the highest objective function.[16]

The goodness measure is "needed for assessment of partition graph strength and subgraph coupling."[17] It is calculated as follows:

Let:

Lp number of links in subgraph p

N_p number of nodes in subgraph p

L_{pq} number of links connecting nodes in subgraph p to subgraph q.

K number of non-overlapping subgraphs.

W_p sum of the W_{ij} in subgraph p.

W_{pq} sum of the W_{ij} connecting nodes in subgraph p to subgraph q.

Then:

Strength of subgraph p:

$$L_p - (N_p - 1) \qquad W_p$$

[16] Karimi and Konsynski, op. cit., p. 201

[17] Karimi and Konsynski, op. cit. p. 200. Again, some substitution of variable names has been made.

$$S_p = \frac{}{(N_p\ (N_p - 1)/2) - (N_p - 1)} \quad \frac{}{L_p}$$

Coupling between subgraph p and subgraph q:

$$C_{pq} = W_{pq} / N_p N_q$$

Goodness measure of a subgraph p as a partition:

$$M_p = [\ \sum p=1,K\ \ Sp\ ]\ -\ [\ \sum p=1,K-1\ \ \sum q=K+1,K\ \ C_{pq}\ ]$$

It can be observed that the Goodness Measure M is the difference of the "strength" of subgraph p and the coupling of subgraph p to subgraph q.

MODEL 4: DATA FLOW ALGEBRA MODEL.

The fourth model is based on an algebra for formal description of a Data Flow Diagram and an associated set of formal transformations and statements. "The objective of the algebra is to provide a rigorous technique for process decomposition ... "[18]

The decomposition process begins with the Context Level diagram. All input and output data items are tabulated in an "I/O Matrix", as shown in Figure 3. Row 1 indicates that Input a is used to produce output y and z. Column 2 indicates that Output y is produced from Inputs a,b, and c. Cell 3,3 indicates that Input C is used to produce Output z.

Figure 3. I/O Matrix.

		Outputs x		y		z	
Inputs:	a			*		*	
	b	*		*		*	
	c		*		*		*
	d			*			
	e					*	

The decomposition proceeds by generating strings in the algebra, and applying operations until a stopping condition is met.

Strings in the algebra are of the form <input> -> <output>, which means that some collection of input variables is used to produce some collection of output variables.

Several operations are defined in these strings. *Substitution* allows redundant variables to be combined, over unconnected processes. *Substitution* allows redundant variables to be

[18] Mike Adler. "An Algebra for Data Flow Diagram Process Decomposition", IEEE Transactions on Software Engineering, February 1988, v.14, n.2, p 169.

combined, over connected processes. *Substitution* allows redundant variables to be combined, over connected processes. *Connection* allows process variables to be connected to allow nesting to be continued. All operations except substitution may result in the generation of a new matrix element. This is necessary because the decomposition cannot proceed unless there is a process with at least two inputs and two outputs.

Decomposition begins with the initial I/O matrix and directly derivable statements. Then,

> The actual decomposition is performed by applying the operators of the algebra to the initial sequence of the decomposition. The operators are applied in the following order: 1) Absorb, 2) Collect, 3) Substitute exact, 4) Substitute subset, 5) Substitute weak, 6) Connect. The operators are applied to each transform or pair of transforms in the transform list to produce a minimal sentence. As each list (input and output) of the new sentence is constructed, the operators are reapplied to it. This gives the best chance of finding a minimal sentence in each case. After minimization, the transform or pair of transforms is replaced in the transform list by this minimized sentence and the operators are reapplied to the transform list to continue the minimization process.[19]

The process continues until there are no more decomposable statements. In Adler's terms this means that all statements have been reduced to sentences with a single data variable in either the left or right hand side (or both sides).

Adler identifies three formal categories for the utility of the resulting sentences. A trivial sentence is one that is identical to the initial sentence or term. An Optimal sentence is one that is equivalent to the initial sentence and not Trivial. A Feasible sentence is one not equivalent to the initial sentence and not Trivial.

IV. Discussion of the Four Models.

Our comparison of the four models will be based on a list of characteristics. This list contains the questions raised in the introduction, and several related issues. Each model requires

[19] Adler, op. cit., p. 178.

that the analyst prepare an organized input. This input is called the **Conceptual Design**. Each model produces a restructured output. This output is called the **Formal Design.** The list has two major categories: the model as a model, and the model as a design tool. As a design tool, we are interested in the utility of the model for producing the Formal Design, given the Conceptual Design. As an abstract object, we are interested in the consistency, generality, and elegance of the model. The list and a discussion of each characteristic follows.

A. The Model as a Tool: Design Characteristics.

1. Information Required. What kind of Conceptual Design is needed to begin using the model?
2. Influence of Conceptual Design. What is the influence of the accuracy, detail, and completeness of the Conceptual Design?
3. Domain Limitation. Is any class of Conceptual Design excluded or poorly supported?
4. Life Cycle Window. At what point in the development life cycle can the model be used?
5. Capacity Constraint. Are there any limits on the number of objects in the Conceptual or Formal Design?
6. Interactive Inputs. Does the model require the analyst to supply information in addition to the Conceptual Design?
7. Metrics. Does the model provide an objective measure of the Formal Design?
8. Sensitivity Analysis. Does the model provide for interactive refinement, based on an objective measure of the extent that one variable determines another?
9. Automated Support. Is, or could there be, an automated implementation of

the model?

10. Number of Levels. How many levels can be represented in the Formal Design?

B. The Model as a Model: Abstract Characteristics.

11. Number of Variables. How many variables does the model encompass?
12. Modeled Objects. What is represented by the model?
13. Major Determinants. What objects or relationships dominate the model?
14. Partitioning Direction. Top-down or bottom-up?
15. Partitioning Scheme. What is the general logic of the model?
16. Partitioning Goal/Objective Function. What criteria are used to select a partition?
17. Manual Computability. Could the model transformation be accomplished without the aid of software?

The answer to the seventeen questions about the four models are presented in tabular format. Most of the entries in the tables are self-explanatory. Some additional discussion of three of the characteristics is warranted.

Influence of Conceptual Design is an indication of how sensitive the Formal Design is to variations in the level of detail and consistency in the Conceptual Design. For example, results of the model would vary if a large data structure (a record or an array) was represented as a single data flow, or as a separate data flow for each cell or field. This is a structural problem. A given structure could also have a wide semantic range. Since the models are sensitive to structure, but decomposition is more related to semantics, the resulting Formal Design could misrepresent or obscure the semantics of the Conceptual Design.

Sensitivity Analysis is concept from quantitative optimization. For a given optimization

over some set of variables, constraints, and an objective function, a set of "shadow prices" is computed that give the marginal cost of increasing or decreasing the optimal value of any constrained variable. A similar indicator of the impact of re-partitioning would be useful.

Major Determinants of the model are the parts of the Conceptual design that have the most influence on the creation of the Formal Design. In all four models, the structure of the Conceptual Design is the key determinant. While the semantics of the transformations, the information content of the data (in the Shannon sense of entropy) are typically used in the intellectual process of design, they are not easily subjected to quantitative deterministic analysis. The four model presented here assume the structure of the Conceptual Design is a reasonable proxy for its semantics. This may or may not be the case.

The tabular comparison follows. Table 1 shows the Design Characteristics of each model. Table 2 shows the Abstract Characteristics of each model.

Table 1. Design Characteristics.

	Alexander	Hein	Karimi	Adler
Information Required	Normalize requirements, identify links	Model data and processes spec	Detail DFD, Timing Diagram	I/O Matrix Context
Influence of Conceptual Design and consistent	Needs consistent definition of Requirements and Links	Needs consistent definition of Data Elements (degree of Process detail)	Needs consistent definition Flows	Needs correct analysis of I/Omapping, of Data definition of Data Flows
Domain Limitation	None	None	None	None
Life Cycle Window	Requirement Definition, Detail Design	Requirement Definition	Detail Design	Requirement Definition, Detail Design
Capacity Constraint	None	None	None	None
Interactive Inputs	No	Yes	Yes	Yes
Metrics conditions	Yes R(P)	Yes DI, does not measure decomposition	Yes M_p	Yes Abstract, formal

Sensitivity Analysis	No	No	No	No
Automated Support	Yes	Yes	Yes	Yes
Number of Limit Levels	No Limit	2	No Limit	No

Table 2. Abstract Characteristics.

	Alexander	Hein	Karimi	Adler
Number of Variables	8	6	24	1
Modeled Objects	Requirements	Data Classes	Data Stores	Data Stores
	Links	Data Elements	Data Flows	Data Flows
	Weights	Processes	Processes	Processes
Major Determinants:				
Structure	Dominant	Dominant	Dominant	Dominant
Data	No affect	No affect	No affect	Secondary
Process	No affect	No affect	No affect	No affect
Partitioning Direction	Top-down or Bottom-up	Bottom-up	Bottom-up	Top-Down
Partitioning Scheme	Minimize cut set, Maximize partition independence	None, suggests grouping by highest D_{ij}	Identify highly coupled set of processes	Identify process with complex input or output
Partitioning Goal/Objective Function	Good clustering of similar requirements	Group processes with common data	Identify feasible, good, subsystems	Formally verifiable decomposition
Manual Computability:				

Small System	Yes	Yes	No	Yes
Larger System	No	No	No	Yes

V. Conclusions.

In nature, the organization of components into structure is a fundamental paradigm. Sub-atomic particles are structured into atoms, atoms into compounds, compounds into cells or crystalline lattices, and so on to the social and ecological organization of species or to galaxies of stellar objects. This organization is not arbitrary. It is in large part the result of a low-level equilibrium enabling an assembly of the next higher level of organization. If this assembly is stable, it may become a component of a larger assembly of like components.

In the intellectual process of system design, organization of components is also a paradigm with significant utility. By providing views of a complex system at a manageable level of detail, we can express and solve structural and semantic problems with greater ease and confidence. However, in contrast to nature, human organization tends to be subjective and arbitrary. This is a major limitation on the correctness of a design and the productivity of its creators.

Each of the models considered here provides a viable, objective means to organize a Conceptual Design. Each is limited by the fact that the semantic content of the Conceptual Design is not directly represented, and is not considered in creating the Formal Design.

Bibliography.

Adler, Michael. "An Algebra for Data Flow Diagram Process Decomposition", *IEEE Transactions on Software Engineering*, February 1988, v.14, n.2, pp 169-183.

Alexander, Christopher. *Notes on the Synthesis of Form*. Cambridge: Harvard University Press, 1964.

DeMarco, Tom. *Structured Analysis and System Specification*. Englewood Cliffs: Prentice-Hall Inc., 1978.

Hein, K. Peter. "Information System Model and Architecture Generator", *IBM Systems Journal*, 1985, v.24, n.3/4, pp 213-235.

Karimi, J. and Konsynski, B. R. "An Automated Software Design Assistant", *IEEE Transactions on Software Engineering*, February 1988, v.14, n.2, pp 194-210.