System Structure Analysis: Clustering with Data Bindings

DAVID H. HUTCHENS, MEMBER, IEEE, AND VICTOR R. BASILI, SENIOR MEMBER, IEEE

Abstract—This paper examines the use of cluster analysis as a tool for system modularization. Several clustering techniques are discussed and used on two medium-size systems and a group of small projects. The small projects are presented because they provide examples (that will fit into a paper) of certain types of phenomena. Data bindings between the routines of the system provide the basis for the bindings. It appears that the clustering of data bindings provides a meaningful view of system modularization.

Index Terms—Cluster, coupling, data binding, module, measurement, system structure.

I. Introduction

An aspect of complexity that has long been recognized, but seldom measured, is the complexity associated with system modularization, i.e., the grouping of procedures into modules within the system. It has been argued by many [1], [2] that system modules should have small interfaces (parameters and shared data) and that the internal components of the modules should be strongly connected. It has also been suggested that modules should be developed so that a fault is contained within a small module. Faults might be used as a means of determining if modularization techniques have placed together those procedures that are often sharing faults.

The analysis of the interface between the small components of the system can be used to determine the modularization that those interfaces define. This analysis is called clustering, and the modules so defined will be referred to as clusters. Armed with this knowledge, one might ask questions about how closely the current modularization (as described in the documentation) corresponds with the modularization defined by the clustering. One might also consider the strength and coupling of the modularization of the system defined by the clustering. The information gained from this work should be of interest to designers and maintainers. It may also be used to obtain a modularization of a system that has no (or little) existing high level documentation, giving maintainers a handle on the structure of the system.

The closer that the objectively defined modules correspond to the modules defined by the developer, the better one should feel about the design. However, it is unlikely that the two views of the system will correspond exactly. Something can be learned about a system from the differences. It may also be possible to derive some basic measurements of the quality of the modularity from the results of this analysis. These measures may provide a means of comparing various design proposals and monitoring systems during maintenance.

Each of these possibilities will be considered in the following sections. Having stated the research goals, it is now appropriate to consider the work that has been done by others and provide the foundations of this work.

II. Background

Data organization metrics are measures of data use and visibility. Several types of data organization metrics appear in the literature. Some of these are briefly mentioned here. Data binding [3], [4] is an example of a module interaction metric. Span [5] measures the proximity of references to each data item. As such it qualifies as a data organization metric. Slicing [6] can also be considered a data organization metric. A slice is that (not necessarily consecutive) portion of code that is necessary to produce some prescribed partial output from the program. Fan-In [7] measures the number of procedures that pass data, through parameters or globals, into a given procedure. Fan-Out is the number of procedures receiving data from the given procedure. Yao and Collofello [8] use detailed data flow analysis to determine a measure they call stability.

A. Data Bindings

Data bindings will be used in this paper to measure the interface between the components of a system. In order to compare this work to other work that has used data bindings, several levels of data bindings will be defined.

A *potential data binding* is defined as an ordered triple (p,x,q) where p and q are procedures and x is a variable within the static scope of both p and q. Potential data bindings reflect the possibility of a data interaction between two components, based upon the locations of p, q, and x. That is, there is a possibility that p and q can communicate via the variable x without changing or moving the definition of x. Whether or not x is mentioned inside of p or q is irrelevant in the computation of potential data bindings.

A *used data binding* is a potential data binding where p and q use x for either reference or assignment. The used data binding requires more work to calculate than the potential data binding as it is necessary to look inside the components p and q. It reflects a similarity between p and q (they both use the variable x).

An *actual data binding* is defined as a used data binding where p assigns a value to x and q references x. The actual data binding is slightly more difficult to calculate as a distinction between reference and assignment must be maintained.

Manuscript received August 31, 1983; revised April 10, 1985.

This work was supported in part by the Air Force Office of Scientific Research under Contract AFOSR-F49620-80-C-001 to the University of Maryland. Computer support was provided in part by the Computer Science Center at the University of Maryland.

D. H. Hutchens is with the Department of Computer Science, Clemson University, Clemson, SC 29631.

V. R. Basili is with the Department of Computer Science, University of Maryland, College Park, MD 20742.

Thus more memory is required but there is little difference in computation time. The actual data binding only counts those used data bindings where there may be a flow of information from p to q via the variable x. The possible orders of execution for p and q are not considered.

A *control flow data binding* is defined as an actual data binding where there is a "possibility" of control passing to q after p has had control. The possibility is based on a fairly simple control flow analysis of the program. To be more precise, a possibility is said to exist whenever either 1) there exists a chain of calls from p to q or vice versa, or 2) there exists a procedure r such that there are chains of calls from r to p and from r to q and there exists a path in the directed control flow graph of r connecting the call chain p with the call chain to q. The solution to the general problem of allowable control flow sequences (where allowable means there exists data which will cause the sequence to be followed) is known to be uncomputable. However, one might improve on this measure by using techniques of data flow analysis to prove more paths impossible and thereby remove more data bindings. It seems unlikely that this added effort will yield enough improvement to justify the effort. This binding requires considerably more computation effort than actual data bindings because static data flow analysis must be performed. Note that a control flow data binding of the form (p, x, q) may exist even though q can never execute after p (because of the dynamic properties of the program).

As an example consider the following portions of code. The parameter of the call is assumed to be call by value.

```
INT a, b, c, d
PROC p1
   /* uses a, b */
   /* assigns a */
   . . .
   CALL p2
   . . .
PROC p2
   /* uses a, b */
   /* assigns b */
   . . .
   CALL p3 (x)
   . . .
   CALL p4
   . . .
PROC p3 (int e)
   /* uses c, d, e */
   /* assigns c */
   . . .
PROC p4
   /* uses c, d */
   /* assigns d */
   . . .
START p1
```

In this example, the potential data bindings are

$(p1, a, p2), (p1, a, p3), (p1, a, p4),$

$(p2, a, p1), (p2, a, p3), (p2, a, p4),$

$(p3, a, p1), (p3, a, p2), (p3, a, p4),$

$(p4, a, p1), (p4, a, p2), (p4, a, p3),$

$(p1, b, p2), (p1, b, p3), (p1, b, p4),$

$(p2, b, p1), (p2, b, p3), (p2, b, p4),$

$(p3, b, p1), (p3, b, p2), (p3, b, p4),$

$(p4, b, p1), (p4, b, p2), (p4, b, p3),$

$(p1, c, p2), (p1, c, p3), (p1, c, p4),$

$(p2, c, p1), (p2, c, p3), (p2, c, p4),$

$(p3, c, p1), (p3, c, p2), (p3, c, p4),$

$(p4, c, p1), (p4, c, p2), (p4, c, p3),$

$(p1, d, p2), (p1, d, p3), (p1, d, p4),$

$(p2, d, p1), (p2, d, p3), (p2, d, p4),$

$(p3, d, p1), (p3, d, p2), (p3, d, p4),$

$(p4, d, p1), (p4, d, p2), (p4, d, p3),$

$(p1, e, p3), (p2, e, p3), (p4, e, p3).$

However, the used data bindings are only

$(p1, a, p2), (p2, a, p1), (p1, b, p2), (p2, b, p1),$

$(p3, c, p4), (p4, c, p3), (p3, d, p4), (p4, d, p3),$

$(p2, e, p3)$

actual data bindings are

$(p1, a, p2), (p2, b, p1), (p3, c, p4), (p4, d, p3), (p2, e, p3)$

and control flow data bindings restricts the set to just

$(p1, a, p2), (p2, b, p1), (p3, c, p4), (p2, e, p3).$

1) Using Data Bindings: The Belady and Evangelisti study [9] applied used data bindings in determining modules for a system. Based on the study of an IBM operating system, they concluded that certain metrics of modularity could be derived from clustering. They used a technique that gave a flat (nonhierarchic) partitioning of the components of the system into modules.

The use of data bindings to determine the appropriate modularization has its drawbacks. A module that hides a data structure is easily found by a data bindings modularization technique. However, a module that defines an abstract data type and has no local data that is shared among the operations on the type will not be located using this method. The reason is that there is no *direct* data binding between the operations of the module. All of the interactions are indirect through the procedures that use the abstraction. Hence, there are relatively few data bindings between them, and they do not tend to cluster.

It would seem that the abstract data type modules need a different measure of connectivity. However, only explicit syntax such as the package of Ada (TM) [10] or the module of MODULA [11] allow abstract data types to be automatically recognized as utility functions and removed from the analysis. Except as noted for specific utility routines, this issue will be ignored in the rest of this paper.

B. Definition of Clustering

Since the components will be grouped based on the strength of their relationships with each other, a reasonable starting point is *mathematical taxonomy*, often referred to as *clustering*. The idea has been used [9] to partition a large system into subsystems.

Because there are so many clustering methods and algorithms that have been published, e.g., [12]-[15], the choice of techniques to use is not easy. Due to the authors' belief that systems and programs are best viewed as a hierarchy of modules and their hope that the levels of the hierarchy will provide fertile ground for the definition of measures, this paper will concentrate on clustering methods that exhibit their results in this fashion. This section will give a formal definition of ä hierarchic clustering method based on the one presented in [16].

A *dissimilarity matrix* d for an ordered set P of n elements is defined to be an $n \times n$ matrix such that

1) $d(a, b) \geqslant 0$

2) $d(a, b) = d(b, a)$

3) $d(a, a) = 0$

for all a, b $1 \leqslant a \leqslant n$, $1 \leqslant b \leqslant n$. That is, d is a nonnegative, real, symmetric matrix with zeros on the main diagonal.

D is defined to be a *dendrogram* over P if

1) D : [0, infinity) $\text{-->} E(P)$

2) $0 \leqslant x \leqslant y \Rightarrow D(x) \leqslant D(y)$

3) there exists $x > 0$ such that $D(x) = P \times P$

4) given $x \geqslant 0$ there exists $y > 0$ such that $D(x + y) = D(x)$

where $E(P)$ is the set of equivalence relations on P. That is, 1) given any nonnegative real number x (a level), D yields an equivalence relation. The clusters defined for level x are the equivalence partitions of P defined by $D(x)$. Furthermore, 2) given level $y > x$, each cluster at level x is contained in some cluster at level y. Also, 3) there exists some level at which all of P is in a single cluster. Lastly, 4) is just a uniqueness technicality to handle the ambiguity at those levels where D is discontinuous. Hence a dendrogram might be pictured as follows:

```
a ------------
b -------      |----------
        |-----            |
c -------                 |------->
d -------                 |
        |-----------------
e -------
___________________________________
0     5     10    15    20    25
```

This dendrogram shows (a), (b, c), and (d, e) forming clusters at level 5, (a, b, c) and (d, e) at level 10, and all collapsing together at level 20.

A dendrogram may be represented as a tree where the branches of the tree are the clusters with associated levels and the leaves are the elements of P. The tree for the above dendrogram might be given in Lisp-like notation as

(20 (10 a (5 b c)) (5 d e)).

A *hierarchic cluster method* is a function from the set of dissimilarity matrices to the set of dendrograms over an ordered set P. The basic algorithms that will be used to implement the cluster methods are *agglomerative*, or bottom-up. They iteratively create larger and larger groups, until the elements have coalesced into a single cluster. The elements chosen for grouping are the ones with the smallest dissimilarity. Given an algorithm, a cluster method is determined, although the converse is not true. Algorithms are introduced here because they provide a reasonable way of specifying the cluster methods.

The character of the individual algorithms is determined by the method used to compute the new dissimilarity matrix at each iteration. The dissimilarity between two elements should not change during an iteration. However, at each iteration some elements are replaced by a single element representing a newly formed cluster. It is the dissimilarity between the newly formed clusters and the other elements (including other newly formed clusters) that must be specified.

The classical algorithms include "single-link" which takes the smallest dissimilarity between the elements of each pair of newly formed clusters as the new dissimilarity coefficient between the them. This gives clusters whose elements are connected at the given level. Another algorithm uses the largest dissimilarity between the elements as the new coefficient and gives clusters that are completely connected at the given level. (Strictly speaking, the clusters are completely connected only if all of the elements that combine into a single cluster during one iteration are pairwise related by the same dissimilarity.) Other well-known algorithms use the average dissimilarity or the weighted average dissimilarity as the new coefficient.

C. Data Sources

The Software Engineering Laboratory (SEL) [17] data were collected during the development of production software for the NASA Goddard Space Flight Center. The systems used in this work are ground-support systems for satellites and were written in Fortran by Computer Sciences Corporation. The developers have a large amount of experience in building this type of system. Both the users and the developers feel that the overall system designs are fairly good.

Data concerning effort, errors, methods, reused code, and other relevant information have been collected for several projects. Most of the data are supplied by the developer on forms prepared for use by the SEL. These forms are normally filled out by the programmer or manager most closely involved with the subject of the form as the knowledge required becomes available. The data have been used to investigate many aspects of system development.

III. A Modularity Study

A technique will be presented that automatically produces a hierarchic module decomposition for a system. The technique is based on data bindings and clustering algorithms. There are several choices to be made in determining the best technique for this application. Some reasonable choices are presented and analyzed. The techniques are then used on some sample systems and the results are given.

A. Specialized Clustering Techniques

The use of data bindings to determine dissimilarity requires that we abstract the data to give a symmetric matrix. Let $b(i, j)$ be the number of control flow data bindings of the form (i, x, j) or (j, x, i) for some program variable x. A dissimilarity matrix may be computed from the binding matrix b in several ways.

1) Recomputed Bindings: One way was chosen that captures the intuitive notion that if a component of the system is entirely connected to just one other component, that connection should be computed as a lower dissimilarity than any connection that is not complete. It is based on the percentage of the bindings that connect to either of the two components and are shared by the two components. That is, let p be the dissimilarity matrix defined by

$$p(i,j) = \frac{(\text{sum}i + \text{sum}j - 2\,b(i,j))}{(\text{sum}i + \text{sum}j - b(i,j))}$$

where sum$i$ is the number of data bindings in which i occurs and sum$j$ is the number of data bindings in which j occurs. Since

$$\text{sum}i + \text{sum}j - b(i,j)$$

is the number of data bindings in which either i or j occur and

$$\text{sum}i + \text{sum}j - 2\,b(i,j)$$

is the number of data bindings in which either i or j occur but not both, $p(i, j)$ is the probability that a random data binding chosen from the union of all bindings associated with i or j is not in the intersection of all bindings associated with i and j. Note that if components i and j have no external connections, then

$$\text{sum}i = \text{sum}j = b(i,j)$$

and $p(i, j) = 0$. Note also that if i and j share no common data then $b(i,j) = 0$ and $p(i,j) = 1$.

For an example, consider the program in Section II-A with the actual bindings

$$(p1, a, p2), (p2, b, p1), (p3, c, p4), (p4, d, p3), (p2, e, p3).$$

A binding matrix can be computed such that the (i, j)th entry is the number of data bindings between the ith and the jth procedures giving

$$\begin{array}{ccccc} & p1 & p2 & p3 & p4 \\ p1 & 0 & 2 & 0 & 0 \\ p2 & 2 & 0 & 1 & 0 \\ p3 & 0 & 1 & 0 & 2 \\ p4 & 0 & 0 & 2 & 0. \end{array}$$

Then the dissimilarity matrix computed as outlined above is

$$\begin{array}{ccccc} & p1 & p2 & p3 & p4 \\ p1 & 0 & 1/3 & 1 & 1 \\ p2 & 1/3 & 0 & 4/5 & 1 \\ p3 & 1 & 4/5 & 0 & 1/3 \\ p4 & 1 & 1 & 1/3 & 0. \end{array}$$

Another degree of freedom in clustering is the choice of the algorithm. While any of the previously presented algorithms might be used, they do not correspond to the intuitive notion that the dissimilarity between two clusters should be directly related to the number of data bindings that cross the boundary. To achieve this property, a new dissimilarity matrix is computed from the bindings matrix at each iteration in the clustering process. This, however, introduces another problem. Each of the algorithms stated earlier has the property that if the elements with least dissimilarity are merged (into perhaps several clusters) at one iteration, then the next dissimilarity matrix will have entries that are all greater than the least nondiagonal entry of the last matrix. If the dissimilarity matrix is recomputed at each iteration based on the bindings matrix, it is possible that the new matrix will contain values that are smaller than any that existed in the last matrix. The resulting tree from this new approach is not a dendrogram.

As an example, start with the following binding matrix for the elements (A, B, C):

$$\begin{array}{ccc} 0 & 1 & 2 \\ 1 & 0 & 3 \\ 2 & 3 & 0. \end{array}$$

That is, there are three procedures, A, B, and C, with 1 data binding between A and B, 2 between A and C, and 3 between B and C. This produces the dissimilarity matrix

$$\begin{array}{ccc} 0 & 5/6 & 4/6 \\ 5/6 & 0 & 3/6 \\ 4/6 & 3/6 & 0. \end{array}$$

Joining B and C into a cluster produces a new binding matrix:

$$\begin{array}{cc} 0 & 3 \\ 3 & 0. \end{array}$$

The new dissimilarity matrix is

$$\begin{array}{cc} 0 & 0 \\ 0 & 0 \end{array}$$

causing $(B\ C)$ to be united with (A) at level 0 to get the tree

$$(0\ A\ (1/2\ B\ C))$$

which is clearly not a dendrogram (since the level of a node must be greater than the level of a son and $0 \leqslant 1/2$).

The tree can be converted to a dendrogram in a natural way. If it is assumed that any cluster that was created with a lower value than its son was really included in the same cluster (at the same level) as its son, the tree can be collapsed into a dendrogram.

The above example would thus give the simple dendrogram described by

$$(1/2\ A\ B\ C).$$

The dendrogram obtained from the latter approach has many good properties. Each cluster is based on the bindings to the other clusters regardless of how late in the clustering process it was formed. This method will be called Recomputed Binding Clustering.

2) Expected Bindings: A problem with the proposed clustering method is that the levels are somewhat incomparable. That is, at a point in the algorithm where there are a large number of elements (e.g., 50) there is less likelihood that two components will have a very large percentage of their total bindings occur between them than when there are few elements (e.g., 3). It seems reasonable to attempt to weight the binding levels relative to the total number of elements under consideration in a given iteration. In particular, if there are n elements under consideration and there are k bindings involving either element i or element j, one would expect $k/(n - 1)$ of the bindings to be between i and j were the bindings to be distributed in a uniformly random way. Hence, those with exactly $k/(n - 1)$ interconnections should have a similar level whether $n = 5$ or $n = 250$. One might therefore compute the new dissimilarity as

$$d(i, j) = (k/(\mathrm{n} - 1))/\mathrm{bind}\ (i, j)$$

for each i not equal to j at each iteration. This method will be referred to as Expected Binding Clustering.

Consider the example of the 4 procedures $p1, p2, p3,$ and $p4$ from Section II-A. If we take the used data bindings, the binding matrix is

$$\begin{array}{ccccc} & p1 & p2 & p3 & p4 \\ p1 & 0 & 2 & 0 & 0 \\ p2 & 2 & 0 & 1 & 0 \\ p3 & 0 & 1 & 0 & 2 \\ p4 & 0 & 0 & 2 & 0. \end{array}$$

This gives the dissimilarity matrix of

$$\begin{array}{ccccc} & p1 & p2 & p3 & p4 \\ p1 & 0 & 1/2 & I & I \\ p2 & 1/2 & 0 & 5/3 & I \\ p3 & I & 5/3 & 0 & 1/2 \\ p4 & I & I & 1/2 & 0 \end{array}$$

where I is infinity. The first iteration combines $(p1, p2)$ and $(p3, p4)$ at level 1/2 giving the new binding matrix

$$\begin{array}{cc} 0 & 1 \\ 1 & 0. \end{array}$$

The new dissimilarity matrix happens to be the same as the binding matrix and we get the Expected Binding dendrogram

$$(1\ (1/2\ p1\ p2)\ (1/2\ p3\ p4)).$$

This dendrogram is intuitively satisfying as $p1$ and $p2$ seem to be closely bound and $p3$ and $p4$ seem to be closely bound. Note that if we use the control flow data bindings, $p3$ and $p4$ no longer seem so closely bound. Computing the Expected Binding dendrogram for these bindings yields

$$(1\ p3\ p4\ (1/2\ p1\ p2))$$

reflecting the reduced cohesion between $p3$ and $p4$.

B. System Fingerprints

The clusters that are derived from a system are analogous to a star system. That is, there may be several small subsystems that revolve around the main subsystem. This analogy leads to the naming of some various types of system fingerprints.

Each of these fingerprints will be illustrated by a program chosen from the group of class projects used by the [18] study. These programs implement a small language on a stack based machine, simulated for the students by the three procedures POP, PUSH, and INTERP. Not surprisingly, these three procedures tend to cluster quickly. Control-flow data bindings were computed for these projects and form the basis for the analysis.

Indentation will take the place of parentheses in the examples. That is, the dendrogram

$(8\ A\ (5\ B\ C))$

will be given as

```
8 A
  5 B C.
```

Planetary systems are those that have several subsystems that are connected to form the whole system. These systems may (although not necessarily) have a larger subsystem that acts as the core of the system.

As an example, the following is an Expected Binding Cluster of one of the 19 compiler projects. For Expected Binding Clusters in the following examples, all level numbers were multiplied by 100 and hence are expressed as percentages.

```
66 COMMENT ASIMPID BACKUP
   54 HASH ALLOCATERETURN DUMPSYMBOLS FINDIT
      LOOKUP ALLOCATESEGMENT ALLOCATEARG
      ADDRESS
      27 ACONST
         24 AARRID AID
            19 AFUNCID ASCAN ARPAREN
               AEVALEXP EXPRESSION ALPAREN
               AFLUSH APRODCODE ASTACKOR
               ADOIFLUSH
      45
         42 DCL CODEGEN SEGMENT
             7 POP INTERP PUSH
            41 PROGRAM
               32 STMT CONSTGEN
         13 NEXTCHAR SCAN NEXTSYMB
            SPECIALCHAR IDENTIFIER CONSTANT
```

Notice how the clusters tend to form distinct parts of the compiler. The group at the bottom (at level 13) is the scanner. The group a few lines above it (at level 7) is the interpreter. An interesting group is the large cluster close to the top (at level 27). This cluster was written by one of the members of the programming team (his name began with an A). Also notice how all of the routines fall together when the symbol table routines are added (at level 54).

Black Hole systems have no visible planets. The clustering process finds one key subsystem that then absorbs the rest of the system. This may be a natural phenomenon associated with the way the system is built, or it may be a bias of the clustering scheme. Since the bindings are recomputed at each iteration in the process, a cluster that has already been formed may contain more bindings with other elements than do small elements. Hence the strongest connection that exists may be with the already formed and growing Black Hole. If this happens, the Black Hole may tend to absorb procedures before their relationships with other parts of the system are discovered.

The following example is also an Expected Binding Cluster of one of the 19 compiler projects.

```
100 STMTLIST DCLLIST
    87 HEADING ACTUALLIST
        86 CODEDUMP INPUTCHAR ASSIGN IFSTMT
           WHILESTMT RETURNSTMT READSTMT
           WRITESTMT EXP LOGICALPROD RELATION
           ADDEXP MULTEXP FACTOR
           24 POP
              16 INTERP PUSH
           71 CALLSTMT
              65 SCANNER
                 55 PROGRAM
                    53 ACTUAL VARIABLE
                       PRIMARY
                       46 SEGMENTLIST
                          FORMALPARM
                          41 SEARCHSYMTAB
                             SYMTABDUMP
                             36 ARRAYDCL-
                                LIST IDLIST
```

The symbol table routines, SEARCHSYMTAB, SYMTABDUMP, ARRAYDCLLIST, and IDLIST, seem to dominate this program in a different way from the previous one. Here they are the quickest to cluster and then they form a nucleus about which everything else revolves. It would seem that this group was less effective in isolating the symbol table from the other routines. One might guess, just from the appearance of the clusters, that many of the routines in this program have intimate knowledge of the structure of the symbol table. This is most clearly true of ARRAYDCLLIST and IDLIST which build entries directly in the table.

Gas Cloud systems are those that show no tendency to cluster. These systems are possibly poorly designed as there is no strength to the modules and a large degree of coupling between them. None of the compilers provide a clear example of this type of system. However, the following dendrogram has some of the properties.

```
87 MAIN ASSIGNMENT SEGRETURN STATEMENT IO
   EXPRESSION PROGRAM WHILESTMT IFSTMT LOAD
   JUMP PARSE INITIALIZE DECLARATION SEGMENT
   SEARCH SEGCALL PRIMARY
   76 POP
      73 INTERP PUSH
   83 SCAN NUMBER IDENTIFIER GETNEXTNONBLANK
```

This dendrogram is a recomputed binding clustering of one of the compiler projects. Note that the majority of the procedures fall together at one level. While this system does show modularity with the scanner and interpreter, the rest of the system does not display much modularity.

C. Using Weighted Clustering

If the Black Hole Syndrome is caused by the clustering method, a reasonable approach to correct its bias is changing the weighting of the bindings when producing the dissimilarity matrix for the next iteration. The weighting would cause bindings to large clusters to be discounted slightly to allow the planets to form. This could be done by replacing each occurrence of $b(i, j)$ in the equation that computes the dissimilarity matrix with $b(i, j) * w(i, j)$ where $w(i, j)$ depends on the size of element i and element j.

D. Measurements

Several measurements can be taken on the clustered system. The more obvious ones include the number and sizes of the clusters. Other interesting measurements that may be taken from the dendrogram are the strength and coupling levels of the clusters. These are available from the levels at which the clusters form. If several clusters form at level 76 and then collapse into a single cluster at level 87 then it may be said that they have a strength of 76 and a coupling of 87. For example, note how the preceding compiler examples have a module of the interpreter (containing POP, PUSH, and INTERP) that has a low degree of coupling and a high degree of strength. The numbers just given are from the last example. Note also that even though the interpreter module is essentially the same in all of the projects, the measurements are quite different. This type of analysis appears to be sensitive to the environment of the module. The values of these measures may have more meaning if they are viewed for a single system as it changes over time.

The stability of the system with respect to data interactions can be examined by evaluating the changes in the dendrograms as data bindings are added or removed from the binding matrix. In fact, whole procedures could be removed from the analysis to see what the system structure is without them. This may be particularly useful if parts of the system are built as virtual machines, utility functions, or data abstractions. A particular layer of the system could then be examined while assuming all of the lower layers act as primitives. This approach is best taken by removing the lower layers of the system and treating calls to them as references and definitions of the global variables that they use.

E. Case Studies

The clustering techniques presented have been applied to two medium size systems that are part of the SEL [17] data collection effort. The systems consist of approximately 100 000 and 64 000 lines of Fortran source code, including comments. The larger one was designed as two distinct load modules, one of which has two distinct functions which are not used together in a single execution. Thus there are essentially three programs to analyze. The three programs were not independent, however, as they contained several common subsystems. The actual data bindings between the routines were computed for each of the two systems and used as the basis for the clustering. Actual data bindings were used because they are much cheaper to calculate than control flow data bindings.

Several routines that were designated as utility routines were removed from the analysis as described in the preceding section. This removal was helpful in determining the true relationships among the remaining routines. Without the removal of the utility routines, they provided a second-order relationship between the routines that called them. For example, there was a user-written utility routine that converted one form of date to another. When two very different routines both called this date routine, a two step path was created between them (e.g. (p, x, date) and (q, x, date) are data bindings where p and q each call date with parameter x). Even though there was no actual data relationship between them, they were pulled together in the clustering algorithm. After removal of the utility routine, their direct relationship emerged. The location and removal of utility routines is not automatable. However, these routines tend to be ones that do all of their communication via parameters and return values and are called by more than one other routine. Hence it is possible to automate part of the search for these routines.

The second project, while smaller than the first, is not broken into independent portions so it actually provides a larger example. The second project also has more errors which involve multiple modules so it is better suited for some of the analysis which follows.

1) Finding Functional Clusters: One of the goals of the study was to determine if any of the methods were able to pick out logical modules in the software. The system was designed as several subsystems, and these subsystems were further refined with the major emphasis of design placed on functionality. If the clustering approach is to be useful, the modularization given by the clustering techniques should be similar to the developers' subsystems. There was a close correspondence between the two views of the system. This may be seen by the dendrogram of the smaller system in the Appendix. The two capital letters preceding each of the Fortran procedure names designate the subsystem in which the designers placed the routine.

An interesting note can be made about the places where the cluster and the subsystem designation differ. In a talk with one of the developers, it became clear that these differences occurred with routines which operated on data which were different from those used by the rest of the routines in the subsystem. From this it may be concluded that there is, in this environment, a strong relationship between the functionality of routines and the data usage of the routines. But at the same time, something can be gained by looking at the system from another viewpoint. That is, functionality is not the only view of the system. The maintainer should also be aware of the data usage that actually exists. This information is not necessarily contained in the calling chart documentation even if the documentation is current.

2) Error Analysis: The study of errors involving changes to more than one routine can yield insights into the effectiveness of the clustering techniques. The NASA-SEL database contains error histories for the systems being studied. For a given clustering, the errors that involved more than one routine were attached to the smallest cluster that contained all of the routines involved (that is, it is attached to the smallest cluster which covers the error). The number of errors attached to each cluster was multiplied by the number of routines in the cluster and the products summed over the clusters. The resultant number is an indication of how well a given technique places all of the routines that were involved in a given error into a single cluster at a low level. A low number indicates that many errors were contained in small clusters.

The results of the error study were inconclusive because the NASA-SEL environment tends to generate a small number of interface errors and because only two projects have been examined. The majority of such errors were confined to the developers' subsystems so they tended to be somewhat localized by the clustering techniques as well.

3) Clustering Technique Comparison: Another goal of this study was to determine if there was a difference between clusters that are generated by the various clustering techniques presented earlier. Based on these two case studies, it appears that the Expected Binding Cluster and the Recomputed Binding Cluster are similar to each other and better than the other methods tested. Better here refers to 1) locality of errors and 2) clusters that capture the developers' subsystems and place the individual routines with reasonable siblings at the lower clustering levels.

The following chart illustrates the differences among the clustering techniques as measured by error*module_size count. Smaller values indicate that errors were contained in smaller modules.

	SL	RB	EB	WB
Large-1	780	571	653	605
Large-2	827	571	629	727
Large-3	6014	4635	5172	6462
Small	13 571	14 765	14 175	25 321

The abbreviations are SL = single-link, RB = recomputed binding, EB = expected binding, and WB = weighted binding. The weighted binding did not perform well according to this test. For the large system recomputed binding seems to have been superior, but for the small system single link did better. It must be remembered that Large-1, Large-2, and Large-3 contain portions of common code so there are not four independent observations in these results.

VI. Conclusions

Several clustering methods have been presented and analyzed on some small and medium size programs. It appears that clustering by data bindings can select the logical modules of a system.

This study has not produced sufficient evidence to determine which module generating techniques are best at reducing the scope (i.e., the size of an encompassing module) of development errors. All three subsystems of the larger case study favor the recomputed bindings technique. However, the other case study favors the standard shortest link method. Further work in this area should focus on the selection of the algorithm. In particular, the algorithms should be tried on some very large systems to see if they still work well. One should be wary of the application of random cluster methods to this (or any) domain.

The dendrograms resulting from the data bindings counts can provide fingerprints of some basic design decisions. In particular, examples were shown which distinguish between the use of data hiding versus the global use of data structures.

The case studies show a large degree of correspondence between the automatically generated module structures and those defined by the developers. The places where these differ are instructive in the explanation of procedure connectivity.

The value of clustering may be greatest when it is used on a single system as it evolves over time. Such a use would allow the maintenance personnel to be aware of the changing relationships among the components of the system. Clustering may also be used to test the hypothesis that system modularity tends to deteriorate over time. Once the modules have been determined it is possible to use clustering to determine measures of the strength and coupling of the modules. The dendrogram gives a fingerprint or classification of the system.

Several measures have been proposed for evaluating the automatically derived module hierarchy. These measures have not been adequately evaluated due to the lack of a proper set of data.

There were some inconclusive tests conducted with respect to modularization. Among these is the question of which technique provides the best description of the system modularity. Perhaps the choice should depend on whether the goal is to localize errors, mimic the designer, or compute measures of strength and coupling. Indeed, the question of how to compute measures of strength and coupling is still unresolved. The use of clustering analysis and data bindings holds promise of providing meaningful pictures of high level system interactions. One might readily ask if these pictures are useful to the designer or maintainer of the software. The answer lies in further research and experimentation.

Appendix

The following is an expected binding dendrogram of the small system. The numbers before the parentheses are the levels defined and discussed in the paper. The two numbers inside the parentheses are 1) the number of errors for which this cluster is a minimal cover and 2) the number of subroutines in the cluster.

```
50(11, 246) UAcvtgra TPhex BIbiasin OAopinit
  45(13, 217) TPinitr8 TPchkmod TPlodipd
    42(4, 209) TPmbhs TPhskbhs TPiniti4 TPhsksyn
               TPtimpos TPcrinit TPhexdmp TPhbhs
      41(22, 55)
        40(3, 53) TPspnprd TPcosync TPconadr
           1(0, 3) TPunhedm TPhedck TPunhedp
          39(0, 39) TPqalchk TPconbhs
            31(4, 37) TPstore TPmsun
              30(8, 35) TPmnfchk
                22(11, 34)
                   7(0, 12) TPtpread
                     2(2, 4)
                       1(0, 2) TPrdtelm UAadlchk
                       1(0, 2) TPdrvadl TPredadl
                     3(0, 7)
                       2(3, 5) TPtrans TPpretrn
                              TPtpinit
                         1(0, 2) TPdsktap TPrevse
                  19(5, 22)
                    18(0, 9)
                       3(0, 3) TPpltmsn
                         2(0, 2) TPlodmsn
                                TPunpmsn
                       5(0, 6) TPintvr8
                         1(0,2) TPcreint TPintvi4
                         2(0, 3) TPunpack
                                TPlodhsk
                                TPunphsk
                    18(45, 13)
                       7(2, 11) TPcodsai TPcvtipd
                               TPhsun TPcvthsk
                               TPtppocc
                         4(2, 4)
                           0(0, 2) TPtpfnal
                                  TPtpwrsm
                           3(0, 2) TPdrivtp
                                  TPtpdisp
                         5(0,2) TPcvtmsn
                                TPtpipd
                       3(0,2) TPpocbld TPsungmt
           7(0, 2) TPchktim TPmilsec
           2(0, 6) TPprered
             1(0, 5) TPmxmni4 TPnomint TPnormi4
                   TPnormr8 TPmxmnr8
         7(0, 2) TPdebug TPplthsk
      34(2, 146) UAcurtim DAadjin
        22(0, 144) DCgstat1
          21(2, 143) DAadtape
            18(8, 142) OCplotin
              13(6, 141) BIbset BIbprint BIbrecur
                        BIbinit BIbobsp BIbstate
                        BIbdisp BIbfprt DAnadang
                        OAdtchck
                 12(0, 118) OAeph DRstatus BIbupdat
                    4(7, 32) DRreport
                      3(1, 31) AZchkgap AZgapfil
                             AZrdhdrs AZazcomp
                             AZrecur AZsumary
                             AZexpwrt AZallpts
                             AZazlist
                        2(0, 2) AZpresmo AZextend
                        2(0, 3) AZazdriv AZazview
                               AZoveral
                        1(0, 2) AZreadat AZpadgap
                        2(0, 9) AZazrate
                          1(0, 8) AZredriv TPreinit
                                 AZobsmod AZrefilt
                                 AZabinit AZabdriv
                                 AZabfilt AZsmooth
                        1(0, 4) AZazmuth AZhsaz
                               AZsunaz AZprovec
                        1(0, 2) AZinitop AZmemnam
                   11(19, 81)
                      2(1, 6) OCocobs OCocprnt
                            OCocstat
                         1(0, 3) OCplotoc OCzeroxo
                               OCocpre
                      8(1, 38) OAattdt3 OAattdt4
                              OAattone OAuncl
                              OAcone1 OAcones
                         7(0, 7) OAunccon
                           3(0, 4) OAroots2 OAunc2
                                 OAattdt7 OAattdt6
                           2(0, 2) OAattdet OAattdt1
                         7(0, 2) OAroots1 OAattdt5
                         5(0, 7) DCdcdriv
                           2(1, 6) DCdcinpt
                             1(0, 5) DCdcangl
                                   DCld1hor
                                   DClddih
           DCldcone
           DClddual
7(0,16) OAsindis
  4(0,15) OAoawrit
    3(0,6) OAoasys
           OAuarfix
       2(0,4) OAoasmon
         1(0,3) OAblkavg
                OAspnavg
                OAchoose
      1(0,6) OAdisplt
             OAfill2 OAfillup
             OAfill1
             OAfootnt
             OAcba921
      0(0,2) OAoaplot
             OAoaplt1
 7(7,37) DAephem DAinitcf
        DAwrtazm
        DAwrtoab DAdaint
        DAadjust
   1(1,5) DAedit DAfree
          DAcopyb
     0(0,2) DAsmthvl
            DAfitdri
   5(7,24) DAfirst
     3(5,9)
       0(0,2) DAdurat
              DAdurchk
        1(0,5) DAnadir
               DAdangle
               DAdotdri
               DAdottst
               DAvaldat
        0(0,2) DAchklm
               DAlimchk
      3(1,7) DAoutat
             DAtwerk
             DAoutoab
             DAtimsel
             DAoutaz
        1(0,2) DAdatadj
               DAoutput
      1(1,3) DApreavg
             DAreduce
             DAsift
      3(0,2) DAdaread
             DAplotz
      0(0,2) DArddata
             DAsample
    6(0,2) DAcopya
           DAadjint
  5(0,2) BIbframe BIoabias
0(0,2) TPrdadl BIbobs
4(0,6) DRdridea DRresult
  3(0,4) DRdschck DRtiming
    2(1,2) DRdeadri DRdrinit
1(0,5) DCfinal2 DCgdccon
       DCblkinv
  0(0,2) DCcofsm DCdccons
```

```
   1(0,3) TPtolang TPtptolr TPtolspn
   0(0,2) TPdrvucl TPucltrn
18(0,10) DRnladj DRnlaz DRnldcc DRnldri DRnloab
         DRnloas DRnloc DRnltp
   3(0,2) UArename DRwritit
 0(0,15) DRrewine DRrecall UAmodd TPtpsupr TPtpnld
         DRdafile DAinitfg DAprcent DCunitdc
         TPovride TPselgmt TPtimseq UAintcnv
         UAdspshr UAdspmod
```

References

[1] G. J. Myers, *Composite/Structured Design.* New York: Van Nostrand Reinhold, 1978.

[2] E. Yourdon, *Techniques of Program Structure and Design.* Englewood Cliffs, NJ: Prentice-Hall, 1975.

[3] V. R. Basili and A. J. Turner, "Iterative enhancement: A practical technique for software development," *IEEE Trans. Software Eng.*, vol. SE-1, pp. 390-396, Dec. 1975.

[4] W. P. Stevens, G. J. Myers, and L. L. Constantine, "Structural design," *IBM Syst. J.*, vol. 13, no. 2, pp. 115-139, 1974.

[5] J. L. Elshoff, "An analysis of some commercial PL/1 programs," *IEEE Trans. Software Eng.*, vol. SE-2, pp. 113-120, June 1976.

[6] M. D. Weiser "Program slicing," in *Proc. 5th Int. Conf. Software Eng.*, San Diego, CA, 1981.

[7] S. Henry and D. Kafura, "Software quality metrics based on interconnectivity," *J. Syst. Software*, vol. 2, no. 2, pp. 121-131, 1981.

[8] S. S. Yau and J. S. Collofello, "Some stability measures for software maintenance", *IEEE Trans. Software Eng.*, vol. SE-6, pp. 545-552, Nov. 1980.

[9] L. A. Belady and C. J. Evangelisti, "System partitioning and its measure," *J. Syst. Software*, vol. 2, no. 1, pp. 23-29, Feb. 1982.

[10] *Reference Manual for the Ada Programming Language*, U.S. Dep. Defense, draft revised MIL-STD 1815, July 1982.

[11] N. Wirth, "MODULA: A programming language for modular multiprogramming," *Software Practice Experience*, vol. 7, pp. 3-35, Jan. 1977.

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

[13] B. S. Duran and P. L. Odell, *Cluster Analysis: A Survey.* New York: Springer-Verlag, 1974.

[14] M. H. Van Emden, *An Analysis of Complexity.* Amsterdam, The Netherlands: Mathematical Centre Tracts, 1975.

[15] B. Everitt, *Cluster Analysis.* London, England: Heinemann, 1974.

[16] N. Jardine and R. Sibson, *Mathematical Taxonomy.* New York: Wiley, 1971.

[17] *The Software Engineering Laboratory*, Software Eng. Lab., NASA Goddard Space Fight Center, Rep. SEL-81-104, Feb. 1982.

[18] V. R. Basili and R. W. Reiter, "A controlled experiment quantitatively comparing software development approaches," *IEEE Trans. Software Eng.*, vol. SE-7, May 1981.

David H. Hutchens (M'84) received the B.S. degree in mathematics from Western Carolina University, Cullowhee, NC, in 1977, the M.S. degree in mathematical sciences from Clemson University, Clemson, SC, in 1979, and the Ph.D. degree in computer science from the University of Maryland, College Park, in 1983.

He is currently an Assistant Professor of Computer Science at Clemson University. His research interests include measurement, evaluation, and modeling of the software development process and its product.

Dr. Hutchens is a member of the Association for Computing Machinery and the IEEE Computer Society.

Victor R. Basili (M'83-SM'84), for a photograph and biography, see p. 168 of the February 1985 issue of this TRANSACTIONS.