The Application of Structure and Code Metrics to Large Scale Systems

by

James Thomas Canning

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
DOCTOR OF PHILOSOPHY
in
Computer Science and Applications

APPROVED:

Dr. Dennis Kafura

Dr. Sallie Henry

Dr. Timothy Lindquist

Dr. Bradley Skarpness

Dr. Charles Nunnally

May, 1985
Blacksburg, Virginia

THE APPLICATION OF STRUCTURE AND CODE METRICS TO LARGE SCALE SYSTEMS

by
James Thomas Canning
Committee Chairman: Dr. Dennis Kafura
Computer Science

(ABSTRACT)

This work extends the area of research termed software metrics by applying measures of system structure and measures of system code to three realistic software products. Previous research in this area has typically been limited to the application of code metrics such as : lines of code, McCabe's Cyclomatic number, and Halstead's software science variables. However, this research also investigates the relationship of four structure metrics: Henry's Information Flow measure, Woodfield's Syntactic Interconnection Model, Yau and Collofello's Stability measure and McClure's Invocation complexity, to various observed measures of complexity such as, ERRORS, CHANGES and CODING TIME. These metrics are referred to as structure measures since they measure control flow and data flow interfaces between system components.

Spearman correlations between the metrics revealed that the code metrics were similar measures of system complexity, while the structure metrics were typically measuring different dimensions of software. Furthermore, correlating the

metrics to observed measures of complexity indicated that the Information Flow metric and the Invocation Measure typically performed as well as the three code metrics when project factors and subsystem factors were taken into consideration. However, it was generally true that no single metric was able to satisfactorily identify the variations in the data for a single observed measure of complexity. Trends between many of the metrics and the observed data were identified when individual components were grouped together. Code metrics typically formed groups of increasing complexity which corresponded to increases in the mean values of the observed data. The strength of the Information Flow metric and the Invocation measure is their ability to form a group containing highly complex components which was found to be populated by outliers in the observed data.

ACKNOWLEDGEMENTS

I would like to thank the following people.

, , and for implementing a large portion of the software metrics generator used in this research. and for graciously providing the source code and developmental database necessary for this work. Dr. Sallie Henry, Dr. Timothy Lindquist, Dr. Charles Nunnally, and Dr. Bradley Skarpness who served as members of my committee. For their patience and support, I would also like to thank my wife ; my mother, ; my father, ; and my stepfather, . I thank my teacher, , for preparing me for the many challenges that lie ahead.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter I

INTRODUCTION

The study of software metrics is a relatively new, controversial area of computer science which investigates the existance of quantitative indices of merit. Although the area is in its infancy, the demand for metrics is quite real. In the industrial sector the need for software metrics is expressed in [Deni81].

> " Measurement techniques and parameters are desperately needed for assessing the quality and reliability of software as well as for the prediction and measurement of software production."

A similar need on the part of government organizations was confirmed by a GAO report of June 1978 [Deni81]:

> " There exist no DOD performance criteria to measure quality and to establish a basis for its acceptance or rejection."

to which the Secretary of Defense replied :

> " We concur. We regret and underscore the importance of the need. The department of defense will quickly embrace such measures when they are available."

However, this pressing demand for immediately usable metrics inhibits a process of detailed examination, gradual acceptance, and eventual standardization [Deni81]. Instead, the field has been characterized by some careful and deliberate experimentation confused with a large volume of ad-

hoc and ill-defined stop gap measures [Hame82][Lass81][Shen83]. Controversy has arisen because of a failure to clearly separate these two types of efforts. The problem is further compounded by the fact that researchers cannot agree on a fundamental set of measurable characteristics exhibited by quality software.

The quality of software is the degree to which expectations of the system are ultimately realized. Thus, if software quality is to be measured, expectations regarding the system must first be specified. This implies that the term "software quality" has different connotations for different users of a given software system. For instance, an operating system user, expecting high throughput, evaluates the software on its ability to quickly process jobs. However, systems programmers or maintenance personnel may expect the operating system to be easily maintained, altered or transported. Furthermore, top level management expects quality software to increase profits.

Researchers, probing these different aspects of software quality rely on models to predict and understand the software characteristics under investigation. There are two classes of models, structural and phenomenological, which can be used to predict system behavior [Brow81]. Structural models, used in more scientific disciplines, attempt to un-

derstand the characteristics of a process by hypothesizing interrelationships among components of the underlying system. Phenomenological models, viewing the system as a black box, do not attempt to predict system behavior by examining its structure. Rather, this model characterizes the system by the correlation between system inputs and system outputs. Both styles of modeling are important when trying to understand large, complicated systems. Although good structural models are the mark of a mature engineering discipline [Brow81], factors which influence system behavior may not be easy to pinpoint. Thus, it is hoped that good phenomenological models, in addition to accurate prediction, will help to locate these factors.

The refinement of a structural model is an iterative process which contains the following five steps :

1. Identify the characteristics of the system to be investigated.
2. Identify the factors within the system which effect the characteristics.
3. Create a model which relates the factors.
4. Measure the factors.
5. Validate the model by :
    a) Measuring the desired characteristics of the actual system.

b) Measuring the desired charcteristics of the system model.

c) Comparing the results generated from the actual system and the model system.

The creation of good structural models and their subsequent validation intimately depends on the ability to measure both the characteristics of interest and the influencing factors. Thus, an implicit assumption, underlying step 4 and step 5, is that metrics are available to quantify these characteristics and factors. For instance, consider a system composed of a large software product, a maintenance staff and the interactions between the product and staff. Suppose that management is interested in projecting software modification costs. A study is conducted which determined that the cost will be a function of three factors : the complexity of the software, the skill of the programming staff and the motivation of the staff. A model is then built which relates these three factors. In order to validate this model, measures of software complexity, programming skill, and programmer motivation are required, and are then used as parameters of the model. Furthermore, the final validation step would be to perform the modification, measure the cost incurred, and compare this result with the output from the model. The validated model could then be used to project the maintenance cost of future modifications.

A problem which confronts software metric research is illustrated by the above example. The model's validation is predicated on possessing accurate metrics which measure the assumed underlying factors. In particular, a measurement of software complexity has not yet been standardized or accepted. Here, the term software complexity does not imply computational complexity which concerns algorithm efficiency and machine utilization. Rather, the term implies psychological complexity which suggests the study of software characteristics affecting programmer performance. Thus, much of the research in this area involves the development of metrics which measure the psychological complexity of a software product.

The nature of the complexity metrics which are currently being proposed covers a broad spectrum. This is due to the fact that a software product can be analyzed from a variety of viewpoints. For instance, one researcher may conceptualize an operating system as a single, large, indivisible program, while another researcher sees the system as a collection of individual modules. A third may even view the system as a hierarchy of levels, consisting of a character I/O level, a block I/O level, a file management level and a process handling level. Furthermore, modules and levels may also be considered as measurable software products with their own

collection of subcomponents (i.e. procedures). Correspondingly, a metric of a software product depends upon how the product is viewed. If the product is viewed and measured as an isolated piece of software without regard to its environment, then the metric is called an intra-product or code metric. However, if the product is viewed as a component of a larger system then metrics which consider the interfaces between the component and the remainder of the system are referred to as inter-product or structure metrics. System components, such as modules, may be measured with either code metrics, structure metrics or a hybrid of both.

The inter-product metrics are also called structure metrics because they typically consider the product to be a module or a level in a large scale system. Underlying the measurements generated by these metrics is the assumption that "good" systems decompose into components which are only "loosely" connected to the remainder of the system. By minimizing the connections between components, the system will be made easier to understand and the number of paths along which errors might propogate throughout the system will be decreased [Henr79]. The complexity of a system is affected not only by the number of connections, but also by the degree to which each connection couples two components[Stev74]. The term coupling is used by Myers[Myer75] to

reflect the strength of the association established by a connection from one component to another. The implication is that the time, cost and effort needed to create, understand and maintain large scale systems is directly proportional to the degree to which system components are coupled to each other.

Code metrics are so named because they measure individual system components isolated from their environment. These metrics analyze only the internal composition of system components such as, internal control flow, component length, and various syntactic counts.

Since structure metrics do not require detailed knowledge of the internal mechanisms of system components, they may be applied earlier in the software life cycle than code metrics. System designers could be assisted by structure metrics when comparing the relative merits of alternative designs. Software managers may utilize structure metrics to forecast developmental costs and to establish resource allocations policies. The perceived benefits from code metrics occur during the latter phases of the software life cycle. During system implementation, generated code could be measured and flagged whenever it exceeds pre-specified complexity thresholds. Furthermore, code metrics can be used to signal those system components which need to be tested more thoroughly.

A number of code and structure metrics have been proposed, each of which attempts to capture some dimension of software complexity. Control flow complexity metrics [McCa76], [McCl78], [Schn79] are based on results from graph theory. Entropy metrics [Chan73], [Schu77], [Berl80], [Silv79] are derived from results found in information theory, while a handful of reliabilty based metrics [Musa80],[Litt78] have also been suggested. Some metrics trace the data flow connections[Henr79] throughout the system and still others rely purely on a syntactic analysis of a program [Hals77].

Although numerous metrics have been proposed, many of them remain unvalidated. Validations which have been performed were primarily concerned with code metrics. Preliminary results from these experiments have been positive, suggesting the need for future research. However, other evidence suggests that code metrics may be too sensitive to individual differences among programmers [Schn81]. Furthermore it is not clear that results from code metric experiments generalize to large complex programs. The few validation studies concerning structure metrics are also promising [Kafu81],[Yin78],[Wood80]. The paucity of these validation efforts underscores the need for additional, well organized experimentation. It is the purpose of this work to contribute to such experimentation.

This dissertation contains the description and the results of experiments which applied structure, code and hybrid metrics to three large scale realistic software systems. Before the results of these experiments are presented, a literature review is given in chapter 2. Chapter 3 identifies the set of software metrics used throughout this dissertation. Furthermore, chapter 3 also describes the source of the three software systems and their associated developmental databases. Chapter 4 presents the results of statistical comparisons made between the various software complexity metrics. Chapter 5 reports on experiments which related structure, code and hybrid metrics to observed measures of complexity such as, ERRORS, CHANGES, and CODING TIME. In this chapter, the software metric values and the observed measures of complexity are correlated on both a component by component basis and on a subsystem by subsystem basis. Chapter 6 introduces and presents results of a group level analysis. The final chapter concludes the dissertation by presenting a summary of the results and identifies areas of future work.

Chapter II

LITERATURE REVIEW

During the last decade, the software engineering community has produced an enormous amount of literature concerning the measurement of software. The classification of such a large body of research is difficult, if not impossible, since the literature covers such a broad, multidimensional spectrum of proposed measurements. This difficulty is reflected by the fact that previous surveys of the metric literature have limited their discussion to research dealing with a single metric[Fitz78] or have presented a variety of metrics which can be applied to different stages of the lifecycle[Moha79]. One interesting attempt at categorizing the complexity metrics literature was proposed by Belady[Bel79]. All three of these publications are well written documents, surveying different aspects of the literature in varying degrees of detail. However, it should be noted that each of these articles are at least five years of age. Much activity in the software measurement field has occured since these papers were written. It is the primary purpose of this chapter to describe the more significant metrics and results which have been published during the last ten years.

2.1 MCCABE'S CYCLOMATIC COMPLEXITY

McCabe's code metric [McCa76] denoted V(G), is a graph based metric. Fundamental to the metric is an interpretation of a program as a cluster of strongly connected, directed graphs. Each graph of the cluster depicts the flow of control within a given procedure, where the nodes of the graph represent a block of sequentially processed statements and where the arcs indicate control flow between the blocks. An additional arc from the graph's unique exit node to its unique entry node is drawn to fulfill the strongly connected criterion.

A result from graph theory states that the maximum number of linearly independent circuits of a strongly connected graph equals the graph's cyclomatic number V(G), where

$$V(G) = E - N + 1 \quad \text{and where}$$

E = number of edges in the graph

N = number of nodes in the graph

Thus, McCabe defines the cyclomatic complexity, V(G), of a procedure within a program to be the cyclomatic number of the procedure's associated strongly connected control flow graph.

The intuition behind this definition is that the complexity of a strongly connected graph, hence procedure, is directly proportional to the maximum number of linearly inde-

pendent circuits within the graph. This intuition is reasonable with regard to a procedure's testability since the problem of testing all control paths through a procedure reduces to testing only those paths associated with all linearly independent circuits.

If a program is decomposed into P procedures, then the cyclomatic complexity of the program is equal to the sum of the cyclomatic complexities of its component procedures, and is given by :

$$V(G) = \sum_{i=1}^{p} V(Gi)$$

$$= \sum_{i=1}^{p} (ei - ni + 1)$$

$$= e - n + p$$

where,

ei is the number of edges in procedure i's graph

ni is the number of nodes in procedure i's graph

e is the total number of edges in all graphs

n is the total number of nodes in all graphs

Utilizing results derived by Mills[Mill72] McCabe showed that the cyclomatic complexity of any structured program is the number of simple predicates in the program plus one.

McCabe states that the cyclomatic complexity metric has been useful in an operational environment. In this environment programmers were required to calculate complexity for all modules which they created. Modules whose cyclomatic complexity exceeded ten had to be recoded. This upper bound was established by intuition. In further support of the metric, McCabe also cites a close correlation between an objective cyclomatic complexity ranking of 24 in-house procedures and a corresponding subjective reliability ranking by project members.

2.2 MYERS' INTERVAL MEASURE

Myer's [Myer77] extends McCabe's measure by deriving an interval measure of complexity (L , U). The upper bound U, represents McCabe's cyclomatic complexity while the lower bound, L, is defined to be the number of conditional statements plus one. The interval measure, (L , U), is motivated by the observation that the complexity of the three code segments[Haye80] found below should be ranked

A < B < C

where,

```
A :     IF ( x = 0 )
           THEN ...
           ELSE ...

B :     IF ( ( x = 0 ) and ( y <> 0 ) )
```

```
                THEN  ...
                ELSE  ...

C :       IF (  x = 0 )
                THEN IF ( y <> 1 )
                          THEN  ...
                          ELSE  ...
```

However, McCabe's metric would rank them A < B = C while merely using the lower bound of the interval metric would generate an equally unsatisfactory result of A = B < C. Myer's interval metric, utilizes both measures and induces the proper intuitive ranking :

$$V(A) = (2,2) < V(B) = (2,3) < V(C) = (3,3).$$

2.3 COBBS'S W MEASURE

Cobb [Cobb78] has defined a complexity metric W, which assigns a pair of numbers to each line of code. The first number, termed "FROM", represents the number of paths which emmanate from the line of code, while the second number, "THRU", is the number of paths which pass through a line of code. A path is a sequence of source statements through which the program counter will traverse from a more than one-way branch point or entry, to the next encountered more than one-way branch point. The THRU value indicates the num-

ber of distinct places within the procedure where the environment for the specified line of code must be correctly established. Higher THRU values are a result of nested looping constructs and join points (where two paths join together). Knowledge of join points enables a programmer to locate unstructuredness in a program module. For example, consider the FORTRAN program found in figure 1 and its associated control graph in figure 2.

The control flow graph in figure 2 contains the six distinct paths :

1. PATH 1 : 3 , 4
2. PATH 2 ; 3 , 16 , 17 , 18 , 6 , 7
3. PATH 3 : 4 , 5 , 6 , 7
4. PATH 4 : 4 , 13 , 14 , 15 , 6 , 7
5. PATH 5 : 7 , 10 , 11 , 12
6. PATH 6 : 7 , 8 , 9

The primary join point of order three found at line six appears to be located within the ELSE clause of the IF statement on line four. However, this join point also lies in the last THRU segment of paths two, three and four. Cobb shows how the code can be properly structured without repeating any lines of code.

Cobb concludes that programmers who may only have unstructured languages at their disposal can analyze high THRU

```
                                                        FROM    THRU
 1.          REAL FUNCTION FIXU ( C , Y )                0       0
 2.          REAL*8  C , Y , REL                         0       0
 3.          IF ( Y .EQ. 0.0 ) GOTO 110                  2       0
 4.          IF ( C .EQ. 0.0 ) GOTO 115                  2       0
 5.          REL = 1.0 - C/Y                             0       1
 6. 105      CONTINUE                                    0       3
 7.          IF ( REL .EQ. 0.0 ) GOTO 100                2       0
 8.          FIXU = -DLOG10 ( DABS( REL ) )              0       1
 9.          RETURN                                      0       1
10. 100      CONTINUE                                    0       1
11.          FIXU = 20                                   0       1
12.          RETURN                                      0       1
13. 115      CONTINUE                                    0       1
14.          REL = Y                                     0       1
15.          GOTO 105                                    0       1
16. 110      CONTINUE                                    0       1
17.          REL = C                                     0       1
18.          GOTO 105                                    0       1
19.          END                                         0       1
```

Figure 1: Procedure Fixu

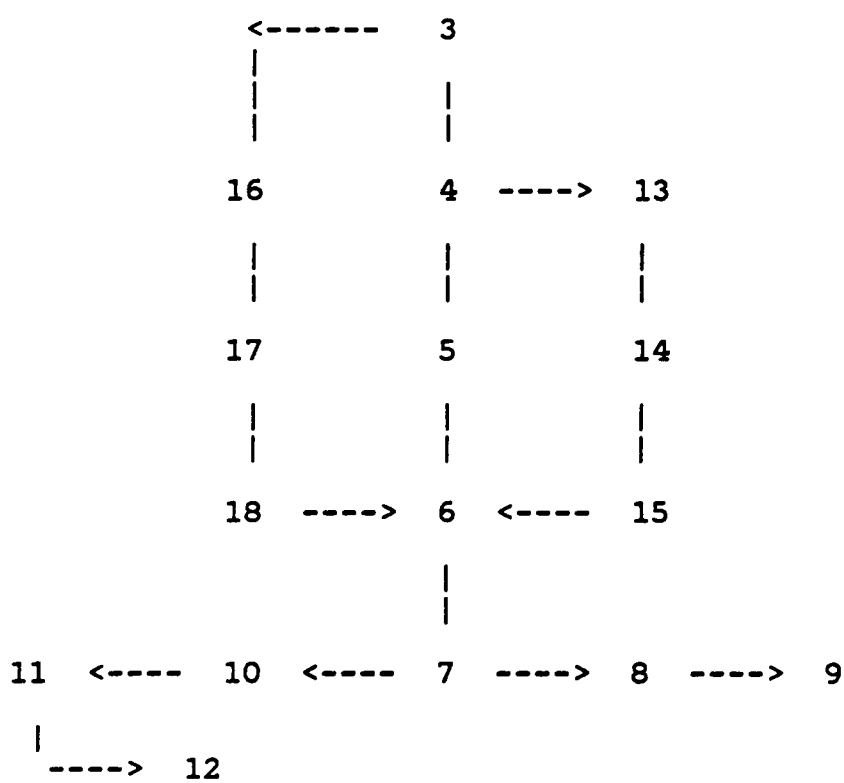


Figure 2: Control Flow Graph of Procedure Fixu

values for indications of code with undue complexity. The W measure may also be viewed as a measure of the control flow complexity existing inside a procedure.

2.4 CHEN'S MAXIMAL INTERSECT NUMBER

In an experiment conducted by Chen[Chen78], program complexity and programmer productivity were related. Chen's program complexity metric, the maximal intersect number (MIN), is another graph theoretic measure which weights nested control structures more heavily than serial control structures. The metric is derived by first forming a control flow diagram of a structured version of a specified program. Additional lines are drawn which will partition the control flow graph into its strongly connected proper subparts, where a strongly connected proper subpart contains no bridges and is not contained within another strongly connected proper subpart. For example, consider the weakly connected control flow graph found in figure 5.

The dashed lines, which partition the graph into its two strongly connected proper subparts, A and B, also divides the two dimensional space into a finite number of regions. The MIN metric is calculated for a subpart by counting the number of graph arcs which are intersected when a continuous line is drawn in such a way that it enters all regions only

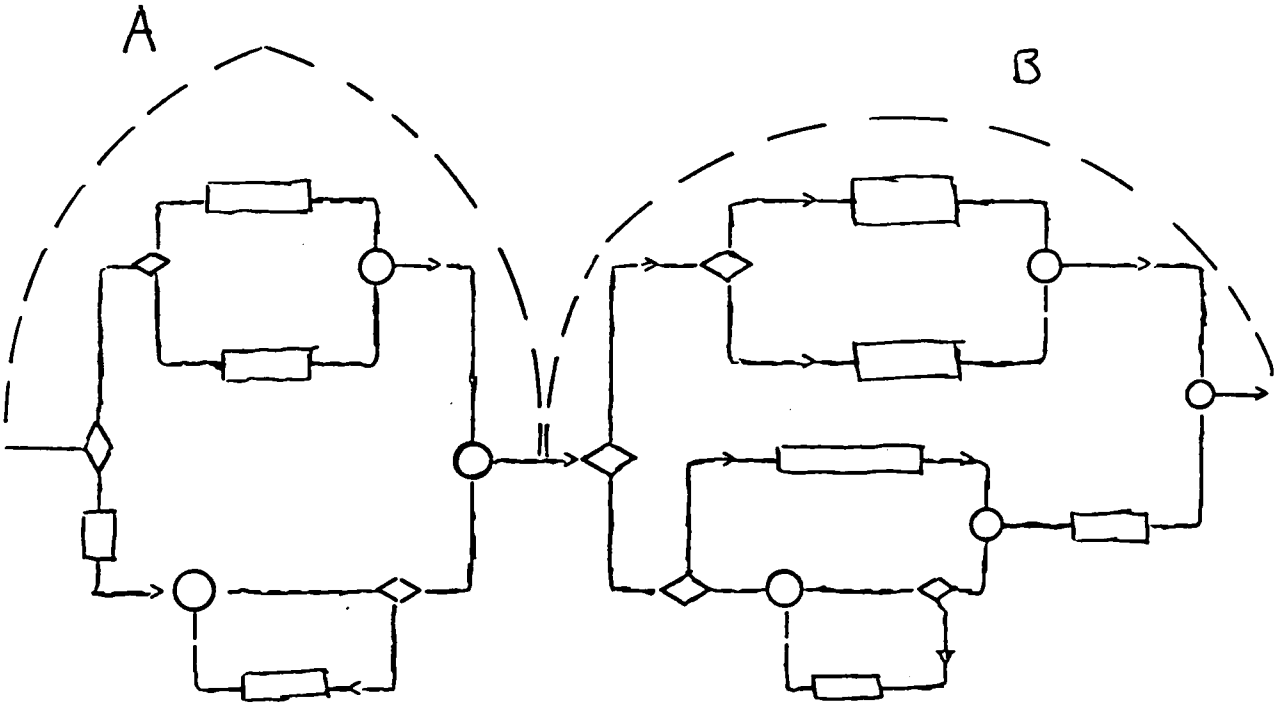


Figure 3: Control Flow Graph with a MIN of 7

once. Thus, subpart A has a MIN value of four, while subpart B has a MIN value of five. Chen proves that the MIN of a weakly connected graph equals the sum of all strongly connected subparts minus, two times the number of subparts plus two. Therefore, the composite graph above has a MIN value of seven (= 4 + 5 - 2(2) + 2).

In general, for a given program containing n decision points, an upper bound of its MIN is n + 1, occuring when all n decision points form a single nested structure of n-level depth. The lower bound of MIN equals two, which occurs when all n decision points are used in sequence.

Chen utilizes the information theoretic notion of entropy to develop a formula which estimates the MIN value of a given program. This formula was then used in a controlled experiment which tested the hypothesis that programmer productivity decreases when producing programs which ultimately have higher MIN values. Chen recorded the time eight novice programmers spent creating eight different programs. For each programming effort, a program productivity index was established by dividing the time spent in creating the program into its number of valid source statements. The programs were then ranked according to both their MIN complexity value and their associated programming productivity index. A comparison of the rankings indicated that programs

with higher MIN values had smaller productivity indexes, thus supporting the above hypothesis.

2.5 SCHNEIDERWIND'S REACHABILITY MEASURE

Another graph based code metric is proposed by Schneiderwind [Schn79]. The complexity of a procedure is determined by first creating the reachability matrix R of the procedure's directed control flow graph, where r[ij] = 1 if a directed path exists between node i and node j of the graph and r[ij] = 0 otherwise. The reachability matrix can be used to locate code which is not executable. This would be indicated by rows in the R matrix which contain only zeroes. The code metric, called average reachabilty, is calculated by summing the elements of the reachabilty matrix R and dividing by the number of nodes.

A study conducted by Schneidewind at the Naval Postgraduate School indicated that procedures which possessed higher average reachabilities were more likely to contain errors than procedures which had a lower average reachabilty.

2.6 HALSTEAD'S SOFTWARE SCIENCE

Halstead [Hals77],[Hals72] takes the view that algorithms have measurable characteristics which are interrelated through laws analogous to physical laws. The developed theory, termed Software Science, states that certain algorithmic properties can be derived from the following four simple counts :

1. n1 = the number of unique operators appearing in a program.
2. n2 = the number of unique operands appearing in a program.
3. N1 = the total number of occurances of all operators in a program.
4. N2 = the total number of occurances of all operands in a program.

The counts, n1 and n2, when summed define the size of the vocabulary of a program and is denoted by n. The length, N, of a program is similarly defined as the sum of N1 and N2.

One fundamental software science metric, called Program Volume, is defined as :

$$V = N * \log_2(n)$$

The derivation of this formula is subject to two interpretations. The first indicates that V is a measure of the number of bits required to specify the program since log2(n) bits

must be used for each of the N program elements . The second interpretation assumes that a programmer makes a binary search through the vocabulary each time an element in the program is chosen. This latter interpretation suggests that the volume V represents the number of mental comparisons required to generate a program.

It is argued that programs written in more abstract languages will generate smaller volume measures. This result is illustrated in [Fitz78], where a FORTRAN implementation of an interchange sort algorithm had a volume of 204.4 bits, while the corresponding PDP-11 assembly language generated a measure of 328.5 bits.

Halstead hypothesized the following conservation law between the level of abstraction and a program's volume :

$$L * V = \text{constant}$$

Here the level, L, of a program is defined as the ratio of potential volume to actual volume :

$$L = V^* / V$$

where V*, the potential volume of a program is the volume measure of the program written in its hypothetical, most abstract form. Since V* cannot be established, an estimator L¬ , for program level is given by :

$$L\neg = (2/n1) * (n2/N2).$$

This estimator reflects the belief that program level will decrease with both an increasing number of distinct operators and an increasing number of recurrent uses of operands. The constant 2, found in the numerator of $L^{\wedge}$, represents the number of operators needed when a program is written in its most abstract form - one to perform the algorithm and one to assign the results to some location. The program level measure can be intuitively thought of as the relative intellectual ease a programmer experiences when making a single mental comparison. The reciprical of L, denoted as D, represents the difficulty of making these mental comparisons and is measured in units referred to as elementary mental discriminations (EMD).

Utilizing the above definitions, Halstead defines a complexity metric E or effort as

$$E = (V) * (D) = (V) * (1 / L)$$

This metric reflects the total number of elementary mental discriminations needed to understand or encode a program. This claim is motivated by the fact that there are V mental comparisons per program and D mental discriminations per mental comparison. A more intuitive argument for E is that as the volume of a program increases and the level of program abstraction decreases, then the effort needed to create the program should increase.

An additional software science metric T, is claimed to be a good estimate for programming time when a given problem is solved by a programmer coding a single module program. The metric is derived from a controversial application of the hypothesis presented by the psychologist John Stroud, which states that the mind is capable of making a limited number of elementary discriminations per second [Stro56]. This number, denoted by S and ranging from 5 to 20, when divided into the effort metric E yields the claimed time measure T. Halstead chose to set S equal to eighteen since this gave the best results during experimentation.

Since its inception, software science has been the source of much controversy. In agreement with the theory, Fitzimmons and Love [Fitz78] review several empirical studies which generate supporting evidence. One such study, conducted by Funami and Halstead [Funa76], used error data collected by Akiyama[Akiy71] on nine system modules to reveal an association between the effort metric E and the number of bugs in a program. The study reports a 0.98 correlation between these two variables. However, Hamer et al.[Hame81] criticize this experiment on the basis that the software science parameters were incorrectly estimated from Akiyama's published data. Ottenstein [Otte79],[Otte81] used software science parameters to develop models which estimate the num-

ber of bugs in a software project. One model [Otte79], which estimates the number of bugs encountered during system integration was supported by three independently published data sets. Another model [Otte81], which predicted the total number of bugs to be found during the entire developement process, fit the published data reasonbly well, but results proved inconclusive when the model was applied to student generated programs.

Although a large amount of empirical evidence supports software science [Funa76], [Elsh76], [Gord79] objections concerning the theory and validity of related experiments have been raised [Lass81],[Hame81],[Shen81]. One criticism [Lass81] recognizes the problems of counting operators and operands. For example, a function reference may serve as an operator and operand at the same time. A second objection concerns the believabilty of assumptions which underline the theory. For instance, the Stroud number used in conjuction with the effort measurement to form the time metric has not been generally accepted among psychologists. Indeed Curtis [Curt80] states,

> "Computer scientists would do well to immediately purge from their memory the Stroud number of 18 mental discriminations per second".

A third criticism concerns the validity and generality of existing empirical studies. Shen and others [Shen81] highlight four problems with such studies :

1. The sample sizes of the experiments were too small.
2. The programs involved were too small.
3. Many of the experiments involved only single subjects. Unless the single subjects were perfectly "typical", the results may not generalize to other programmers.
4. Subjects were generally college students. There is a real concern that results based on this type of subject may not generalize to professional programmers.

The controversy surrounding software science is best captured by the following contradictory quotes :

> " Various studies and experiments have shown that the theory's predictions of the number of bugs in programs and of the time required to implement a program are amazingly accurate . It is a promising theory worthy of much more probing scientific investigation [Fitz78] ".
>
> "... software science seems to be in a very weak state , both theoretically and experimentally . We would strongly advise against any attempt to use it in the practical evaluation and control of software developement " [Hame81].

2.7 HENRY AND KAFURA'S INFORMATION FLOW MEASURE

Henry and Kafura [Henr79],[Henr81b],[Kafu81a],[Kafu81b],[Kafu82] have developed a hybrid metric which is composed of two factors. The first factor is the internal complexity of a procedure, expressed by its length in lines of code, while the second

factor involves the information flow connections of a procedure to its environment. These information flow connections, termed fan-in and fan-out, are determined by first generating a set of relations for each procedure which expresses the flow of information through a procedure from input parameters and global data structures to output parameters and global data structures. From these sets of relations, a forest of trees, called information flow structures, is then built, with the root of each tree representing a global data structure. The leaf nodes of a tree represent data structures or procedures containing values which will ultimately propogate into the data structure of the root node. The internal nodes identify various procedure interfaces through which these data values propogate. Thus, for a given tree, paths from the leaf nodes to the root node define all possible routes along which data may flow into the associated data structure.

The procedure complexity is then given by :

Procedure complexity =length * (fan-in * fan-out) ** 2

The information flow term is squared since it is felt that problems typically encountered when additional connections are added between system components are non-linear in nature. In addition, two other related metrics, a module com-

plexity metric and a level complexity metric, are also defined. A module with respect to a data structure D consists of those procedures that either update or retrieve information from D. Thus, module complexity is defined as the sum of the complexities of the individual procedures within the module. If the measured system is viewed as a hierarchy of levels, then level complexity is the sum of the complexities of those modules which comprise the level.

The authors feel that procedure complexity is helpful in locating the "stress points" of the system (i.e. those procedures with heavy data traffic). In addition, a high fan-in and fan-out together with a high internal code complexity indicates a procedure which has not been adequately refined. The module complexity metric can reveal those data structures which are overloaded or may indicate improper modularization. Level measurements would be used to pinpoint missing levels of abstraction or to compare alternative system designs [Kafu82]

These information flow metrics were applied to the UNIX operating system [Henr81b]. The study indicated a statistical correlation (0.95) between errors and procedure complexity. Furthermore, although similar correlations for McCabe's metric (0.89) and Halstead's effort metric (0.96) were generated, it was found that these measurements were more

highly correlated to each other than to the information flow metric. This result suggested that the information flow method is measuring a different aspect of complexity than are the two code metrics. In addition, the computation of both module and level complexities revealed overloaded data structures and missing levels of abstractions.

2.8 WOODFIELD'S SYNTACTIC INTERCONNECTION MODEL

Woodfield [Wood80] reports on the development and confirmation of a hybrid model of program complexity. The general form of the model is given by :

$$\text{Complexity of Module B} = \text{Internal Complexity of Module B} \quad \text{fan-in} \quad \text{Review Constant}$$

where , fan-in represents the number of times that module B must be reviewed when trying to understand the entire program.

Here module complexity is a function of both the module's internal complexity and its connections to other system modules. The notion of learning is incorporated into this hybrid model to reflect the fact that each time a given module is reviewed, the process of understanding the module is successively less difficult due to the knowledge remembered from the previous reviews of this module.

The specific hybrid model developed by Woodfield is referred to as the " interconnection syntactic model ". This model utilizes a review constant of 2/3, which is a number previously suggested by Halstead [Hals77]. The internal complexity of a module is computed by applying the Logical Module Complexity Metric, which has also been developed by Woodfield. This metric, utilizing a predefined notion of "largeness", partitions modules which are too big into separate logical modules. Each of these logical modules are treated as physically distinct units possessing headings equal to the original module. All modules, both original modules and the logical modules, are then measured using Halstead's effort metric and this measurement is taken to be their internal complexity. Note that the internal complexity of those modules found to exceed the largeness criteria is equal to the sum of the effort measurements of its corresponding logical modules. Woodfield completes his model by defining the fan-in of a module to be the combination of control and data connections for the module. A control connection exists for a module each time it is invoked, while two modules, A and B, are data connected if there is some variable V such that the following two conditions are true :

1. the variable V is modified in B and referenced in A.

2. there exists at least one data path set D(BA), between B and A, such that the same variable V is not referenced in any di where di is a member of D(BA).

where a data path set, D(BA), is an ordered set of one or more modules {d1 , d2 , d3 , ... , dn} such that one of the following conditions is true :

1. B calls D1 ; D1 calls D2 ; ... ; D(n-1) calls Dn ; Dn calls A.

2. A calls D1 ; D1 calls D2 ; ... ; D(n-1) calls Dn ; Dn calls B.

3. There exists some Di in D(AB) such that Di calls both D(i-1) and D(i+1) ; D(i-1) calls D(i-2) ; D(i-2) calls D(i-3) ; ...; D2 calls D1; D1 calls B ; and also D(i+1) calls D(i+2) ; D(i+2) calls D(i+3) ; ... ; D(n-1) calls Dn ; Dn calls A.

The validation of Woodfield's model involved data extracted from advanced student programmers developing thirty small programs (18 - 196 lines of code) in a controlled environment. The time needed to create each program was compared with measurements generated by the interconnection syntactic model. Results indicated that the model was able

to account for 80 percent of the variance in programming time with an average relative error of only 1 percent.

2.9 MCCLURE'S INVOCATION COMPLEXITY

McClure [McC178] proposes a control flow structure metric, which examines the control flow interfaces between modules. For a given module, P, the interface complexity of this module, denoted M(P), is the summation of two factors. The first factor is a measure of the complexity of the circumstances under which module P is invoked. The second factor is a measure of the complexity of module P invoking another module. The complexity of invoking module P is a function of the number of modules which invoke module P as well as the average complexity of all invocation control variable sets which affect the invocation of module P. The term, invocation control variable set, is defined to be the set of control variables upon whose values a particular invocation of a module depends. For example, suppose the following code is used to invoke module P :

```
IF ( V1 = 3 ) AND ( V2 > V3 ) THEN

    WHILE ( V4 = 0 ) DO

        INVOKE P

    END WHILE
```

Then, the set {V1, V2, V3, V4}, forms an invocation control variable set for module P. A module may have multiple invocation control variable sets when it is conditionally invoked in more than one place in the program. The complexity of an invocation control variable set is given by :

$$b \sum_{i=1}^{e} C\ (Vi)$$

where ,

b = 2 if any Vi is a control variable of a repetition structure.

= 1 otherwise .

C (Vi) = Control Variable Complexity *

e = The number of variables in the invocation control variable set .

The complexity of module P invoking another module is a function of the number of modules which P invokes together with the average complexity of all invocation control variable sets used by module P in invoking its direct descendents.

Using the notation found in [McC178], the complexity of a module P is given by :

$$M(P) = [\ Fp * X(p)\ ] + [\ Gp * Y(p)\ ]$$

where,

Fp = The number of modules which invoke module P

Gp = The number of modules which module P involes

X(p)= The average complexity of all invocation control variable set used to invoke module P.

Y(p)= The average complexity of all invocation control variable sets used by module P to invoke other modules .

McClure points out that since the complexity metric, M(P), can be computed for all modules P before the program reaches its implementation phase, program designers could be aided by the use of such a metric. One design strategy recommended by McClure is that the complexity of each module should be minimized while the complexity among modules should be evenly distributed. She claims that overall program readability should improve if these two criteria are satisfied.

2.10 HANEY'S STABILITY METRIC

Haney [Hane72] proposes a technique for modeling the stability of a large system as a function of its internal structure. The technique, called Module Connection Analysis, is motivated by the belief that intermodule connections are the prime causes of cost overruns and delayed delivery dates. These problems occur more frequently in systems where modules are highly connected since any change to a single module typically induces subsequent changes in many of its connecting modules. These second order changes will then begin to ripple through the system necessitating changes in additional modules. The system's resistance against such change propagation is termed system stability.

When modeling a system composed of N modules, Haney's technique assumes the existance of an N x N matrix P, with elements, Pij, representing the one-step probability that a change in module i induces a change in module j. The probability that a change to module i propoagates to module j in two steps is given by P-squared, since the (ij)th element of P-squared is

$$\sum_{k=1}^{N} P(ik)P(kj)$$

which represents the sum of the probabilities that a change in module i is propagated to module j through module k. In general, the (ij)th element of P raised to the Kth power represents the probability that a change in module i will propagate to module j in K steps. The model also assumes the existance of a 1 x N row vector A, where Ai represents the initial number of changes made to module i when it is integrated into the system. Thus, the total number of changes, C, made to all modules during the integration phase is expressed by the equation :

$$C = A (I + P + P + P + \ldots)$$

where, I is the N x N identity matrix .

A result from matrix algebra reveals that the above matrix power series converges to

$$C = A (I - P) \text{ inverse}$$

when the absolute value of the eigenvalues of P are less than one .

Haney suggests that an initial estimate of the P matrix is based on experience and "feel" for the system. These raw values can then be periodically revised during system integration by maintaining a history on all changes, their causes and their aftereffects.

The problem with Haney's method and similar methods [Myer75], [Soon77] is that for large systems, model validation is difficult since model inputs cannot be automatably obtained. However, Yau and Collofello[Yau80] as well as Schutzer [Schu77], outline stabilty techniques which will provide an automable solution.

2.11 YAU AND COLLOFELLO'S STABILITY MEASURE

Yau and Collofello [Yau80] outline an automatable hybrid metric which calculates the stability of modules within a system as well as total system stability. The procedure first calculates two sets of variables for each module. The first set, Vk, identifies all variable definitions in module K. If a variable is defined twice within a module, then Vk contains a unique entry for each defintion. The second set, Tk, representing all variables in module K through which changes may propogate to other modules, contains global variables referenced by module K, input parameters to modules called by module K, and the output parameters of module K. These variables are referred to as interface variables. For each element i, of set Vk, a set Zki is also computed. The set, Zki, a subset of set Tk, identifies those interface variables affected by modifications to variable definition i. Also computed, for each element j, of set Tk, a set X(kj)

consisting of those modules which are affected by a change to interface variable j. Thus, the set of all modules affected by a change to variable definition i in module k can be computed and is referred to as Wki, where

$$Wki = \sum_{j \in Z(ki)} X(kj)$$

The internal complexity of each module in the set Wki is computed using an available code metric and these measurements are subsequently summed to form the logical complexity of modification of variable i in module K, denoted LCMki. Yau and Collofello then define the logical ripple effect (LREk) for module K to be the mean LCMki for all variable definitions i in module K. This logical ripple effect value represents a measure of the expected impact on the system of a modification to a variable in module K. A measure for the stabilty of a module K, denoted by LSk, is then established as LSk = 1 / LREk.

2.12 YIN AND WINCHESTER'S TREE IMPURITIES

Yin and Winchester [Yin78] have defined a set of related structure metrics which attempt to pinpoint sections of a design that may cause problems in the latter phases of the life-cycle. A design is first graphically viewed as a hierarchy of levels, where each level contains processing module

nodes together with data base table nodes. A network of nodes is formed by drawing control and data transfer arcs, within and across levels, as specified by the design. The authors hypothsize that the quality of a design increases as its graph more closely approximates a tree. This hypothesis is supported by the often quoted design objective of minimizing the number of places from which a given procedure is invoked. The more strictly hierarchical (i.e. tree-like) the system graph the more this objective is realized. The calculation of the three structure metrics Ci', Ri' and Di' is given below in figure 6.

The metric Ci' is a monotonically decreasing function indicating the absolute deviation from a tree structure from level 0 to level i. The related metrics Ri' and Di' measure the fraction of tree impurities within the system which occur from level 0 and level i-1 to level i respectively.

In an effort to validate the complexity metrics Di and Ci, two projects containing 1000 modules and 27 modules were partitioned into subsystems. Each subsystem was then divided into levels and measured. Error data collected per module was available and catalogued by levels. Trend analysis between Di' and a normalized error count per level was conducted to determine if fluctuations in the normalized number of errors in each level corresponds to fluctuations in Di. Re-

$$Ci' = Ai' - Ti'$$

$$Ri' = Ci' / Ai'$$

$$Di' = 1 - (\Delta Ti' / \Delta Ai')$$

where,

Ni' = the number of modules and data base references from level 0 to level i.

Ti' = the number of module and data base tree arcs from level 0 to level i.

Ai' = the number of module and data base network arcs from level 0 to level i.

Δ Ti' = Ti' - T(i-1)'

Δ Ai' = Ai' - A(i-1)'

Figure 4: Structure Metrics of Yin and Winchester

sults indicated mild significance for subsystems with a reasonbly large number of levels. A second experiment, hypothesizing that subsystems with higher final complexity C would possess a higher number of total errors, was also conducted. Regression analysis was performed on the generated model yielding high correlation coefficients (0.98) (0.99). Additional experimentation leads Yin and Winchester to conclude that although Ci is an absolute measure, a sharp increase of Ci from one level to the next indicates where a designer may look for defects in the system structure.

2.13 ZOLNOWSKI AND SIMMONS'S COMPLEXITY INDEX

A composite index of complexity which is tailored to a given programming environment is proposed by Zolnowski and Simmons[Zoln81] and later enhanced by McTap[McTa81]. Complexity indices are computed for four categories: instruction mix, data reference, interconnectivity and control flow. Within each category, local experts select a set of software characteristics which are relevent to their own environment. Local threshold values for each software characteristic are established by averaging measurements derived from typical or high quality in-house software products. In order to compute the complexity of a candidate program for a given category, measurements of the specified software char-

acteristics must be taken from the program. These values are then compared with the pre-established threshold measurements. A complexity index, S, for a category is determined by the formula

$$S = \sum_{i=1}^{N} Wi * Ui$$

where,

N is the number of characteristics in the category.

Wi is the weighting factor for the ith characteristic.

Ui equals 1 if the candidate program measurement compares unfavorably with the threshold value.

equals 0 otherwise

2.14 ADDITIONAL EMPIRICIAL STUDIES

Curtis, et al. [Curt79a], [Curt79b] have conducted a series of controlled human factor experiments which investigate the relationships of programmer performance to three code metrics (Halstead's effort, McCabe's Cyclomatic metric and length of a program measured in lines of code). They hypothesized that the Halstead and McCabe metrics were more

closely associated with factors affecting psychological complexity than were lines of code. In the first experiment [Curt79a] subjects were required to recreate previously studied programs. Regression analysis was then performed using the number of functionally correct statements recalled as the dependent variable and each of the three code metrics as independent variables. In the second experiment [Curt79a] subjects were asked to modify programs. Again regression techniques were applied, with two dependent variables (accuracy of the modification and time to the complete the modification) being related to the metrics. Since the correlations in the raw data were not large, ranging from -0.13 to 0.46, and lines of code generally predicted performance better than the other two metrics the results of the experiment were considered to be disappointing. However, it was felt that these two experiments contained several limitations. First, all programs were short (35-55 lines of code) and second, individual differences among programmers exerted significant effects on the experiment. Thus, a third experiment which overcame many of the previous problems was conducted [Curt79b]. In this experiment the code metrics were related to the time needed to locate preplanted bugs. Regression analysis indicated that all three code metrics predicted performance at the subroutine level, but at the

program level Halstead's effort metric was superior than either McCabe's metric or lines of code. Furthermore, substantial intercorrelations were observed among the three metrics at the subroutine level, but at the system level, the intercorrelation between Halstead's metric with the remaining two code metrics was not as pronounced.

Basili [Basi81] [Basi83] utilizing data extracted from the Software Engineering Laboratory at Goddard Space Flight Center, studied a host of code metrics. From these experiments it appeared that there does exist some relationship between complexity metrics and effort as well as errors. However, most of the metrics generally did not outperform predictions based on lines of code. Furthermore, the intercorrelations between many of the metrics was significantly high, suggesting the need to find a set of orthogonal metrics.

Hayes[Haye80] conducted a study to empirically validate several proposed code metrics. Among the metrics were the software science parameters: N1, N2, N, n1, n2, n, E, V, variations of McCabe's cyclomatic number, the number of card images, and nesting levels of conditional statements. These metrics were applied to two sets of data. The first source of data was extracted from 51 subroutines taken from a Radar Trainer System. During the development of the sys-

tem, the number of changes made to each subroutine during system integration was recorded and served as the dependent variable for a variety of statistical analysis. The second source of data was code written by a masters candidate who classified error data in the 31 Algol routines he himself was writing. The total length of time in seconds it took to correct the errors in a particular routine and the total number of errors a given routine served as the dependent variables. Statistical analysis indicated that although both Halstead's effort metric and McCabe's cyclomatic number correlated reasonably well with the dependent variables. Hayes also found that the number of card images also provided satisfactory results. Factor analysis of the Radar Trainer data showed the existance of two complexity factors. One factor, which seem to involve size measurements, clustered around the independent variable N, while the second factor clustered around McCabe's V(G).

2.15 ADDITIONAL METRICS AND RELEVENT WORK

A number of other software metrics can be found throughout the literature. Benyon-Tinker[Beny79] developed a metric which is based on the calling hierarchy of a system. Chapin[Chap79] defines a software metric, Q, which is purported to be an index of the difficulty people have in un-

derstanding the function implemented by the software. The metric, Q has the advantage that it can be computed from a detailed input-output table available at design time. However, the metric has been criticized since it utilizes some non-motivating weighting factors[Perl81b]. Troy and Sweben [Troy81] outline an experiment which assesses design documents in order to determine those design principles which affect system quality. Roberts[Robe80] describes a method for deriving a quality metric from a SADT[Ross77] representation of system design. The method is based upon the coupling and cohesion characteristics [Myer75] of each individual module. Moreover, Roberts introduces and defines a measurement for the RISK incurred in the implementation of the design as formulated. Szulewski and others [Szul81] have proposed a technique for identifying and counting software science parameters from design mediums. Channon[Chan73] defines what is meant by program structure and then creates an entropy measurement of such structure. Although Channon's measure has a solid mathematical foundation, applying it to all but small programs is impractical.

Belady and Lehman [Bela76], [Bela80], [Lehm80], have done some interesting and extensive work concerning the development and maintenance of large programs. From their experiences they postulate the following three laws which all large systems obey:

1. Law of continuing change : A system that is used undergoes continuing change until it is judged more cost effective to freeze and recreate it.
2. Law of increasing entropy : the entropy of a system (its unstructuredness) increases with time, unless specific work is executed to maintain or reduce it.
3. Law of statistically smooth growth : Growth trend measures of global system attributes may appear to be stochastic locally in time and space but , statistically, they are cyclically self-regulating, with well defined long range trends.

By analyzing the consequences of these laws they derived some statistical and formal models of the programming process which may help to control the complexity in an evolving system. They define the complexity of evolving software as the fraction of modules impacted by some standard dose of change.

Chapter III

BACKGROUND

The major roadblock inhibiting comprehensive studies of structure and implementation metrics applied to large scale realistic systems has been the non-availability of three vital resources. The first vital resource needed for this form of experimentation is an automated software metrics tool capable of collecting a wide range of both structure and implementation measurements. Most of the past research in this area has involved the measurement of small student sized programs where metric collection can be hand performed. Those studies using larger software systems either restrict themselves to analysis of code metrics, where the construction of a metrics tool is relatively simple or employ a single structure metric in the analysis [Yin78][Henr79]. A second resource that is usually difficult to obtain is the source code of non-trivial realistic systems. Firms, businesses and large corporations are reluctant to loan or give their software to an interested researcher for close scrutiny. Even if such software were available, its value would be minimized without the third vital resource -- access to an error and effort historical database cataloguing important aspects of the software and the process from which it

was developed. It is the purpose of this chapter to profile the resources which were used in this research and briefly describe how they were acquired.

3.1 DESCRIPTION OF THE SOFTWARE METRICS GENERATOR

The software metrics generator used in this research, capable of extracting structure, code and hybrid measurements from software systems, can be logically viewed as a two step machine. The front end of the machine performs a lexical and syntactic analysis of FORTRAN programs outputting a number of generic files. The second phase of the generator is a language independent backend which assimilates this generic information and computes the following metrics for each subroutine and function within the FORTRAN package. The code metrics and the hybrid metrics are referred to as implementation metrics since they can only be taken after the software has been written.

STRUCTURE METRICS

1. Unweighted Information Flow Metric (INFOFLOW)
2. Unweighted Stability Measure (STABILITY)
3. Unweighted Syntactic Interconnection Model (REVIEW)
4. Invocation Complexity Metric (INVOKE)

IMPLEMENTATION METRICS

1. Halstead's Mental Effort (EFFORT).

2. Lines of Code (LOC)
3. McCabe's Cyclomatic Complexity (CYCLO).
4. Information Flow weighted by Halstead's Effort (INFO-EFF).
5. Information Flow weighted by McCabes V(g) (INFO-CYCLO).
6. Information Flow weighted by LOC (INFO-LOC).
7. Stability Measure weighted by Halstead's Effort (STAB-EFF).
8. Stability Measure weighted by McCabe's V(g) (STAB-CYCLO).
9. Stability Measure weighted by LOC (STAB-LOC).
10. Syntactic Interconnection Model weighted by Halstead's Effort (REVIEW-EFF).
11. Syntactic Interconnection Model weighted by McCabe's V(g) (REVIEW-CYCLO).
12. Syntactic Interconnection Model weighted by LOC (REVIEW-LOC)

The generator itself is approximately seven thousand source lines of Pascal code. Construction of the metrics generator began under the direction of Dr. Sallie Henry who, while at the University of Wisconsin at LaCrosse oversaw the development of the front end and those portions of the back-end needed to compute Halstead's effort, McCabe's V(g), Lines of Code and the Information Flow Metric. The enhancement and augmentation of this tool to incorporate the additional three structure metrics was performed at Virginia Polytechnic Institute and State University.

3.2 DESCRIPTION OF NASA/GODDARD'S CONTRIBUTION

The remaining two vital resources necessary for this project were graciously provided by the Software Engineering Laboratory (SEL) headed by Frank McGarry, Jerry Page and Victor Basili. This organization[Nasa82] formed in 1976, is composed of three members:

1. Nasa/Goddard Space Flight Center(GSFC).
2. The University of Maryland (Computer Science Department).
3. Computer Science Corporation (Flight Systems Operation).

Upon creation, the organization defined their primary objectives as:

1. To understand the software development process in the GSFC environment.
2. To measure the effect of various methodologies, tools, and models on this process.
3. To identify and then to apply successful development practices.

With these goals in mind, the SEL defined and implemented an extensive monitoring and data collection process by which the details of all aspects of the software development process and product could be extracted for analysis[McGa80].

Through the cooperation of the SEL management, source code and historical development data for three of the larger FORTRAN projects were made available for this research. These two important resources are described in the following two subsections.

3.2.1 Nasa/Goddard Source Code

The Nasa/Goddard source code used in this research can be logically viewed as the three level hierarchy depicted in figure 5. As shown, the measured software consists of three separate projects, each of which can be catagorized as scientific in nature. These three software systems will be referred to as PROJECT A, PROJECT B, and PROJECT C. Each project is further subdivided by Nasa/Goddard into subsystems. The largest project, PROJECT A, has nine subsystems, while PROJECT B and PROJECT C possess seven and six subsystems respectively. Throughout this dissertation individual subsystems will be identified by a number appended to its associated project name. Thus, subsystem three in PROJECT A will be referred to as subsystem A.3 . Each subsystem is a collection of FORTRAN subroutines, functions and data blocks, known as components. Within a given project, subsystems generally do not share common components. That is, routines from one subsystem rarely call routines from anoth-

er subsystem. Communication between subsystems is usually acheived through the use of global variables. However, all subsystems share a pool of utility routines written in FORTRAN and assembly language.

3.2.2 Nasa/Goddard Developmental Database

In addition to the three FORTRAN systems, SEL also provided a developmental database associated with each of the three projects. The database itself contains numerous files and a wide range of information. Selected for this study were the following variables:

Count Based Dependent Variables

1. Component Changes: A modification to a component made either to correct an error, to improve system performance, to add capability, or to implement a requirements change.
2. Component Errors: A discrepency between a specification and its implementation. The specification might be requirements, design specification, or coding specification.

Time Based Dependent Variables

1. Design Time: The time recorded by SEL personnel to create, to read and to review the design of an individual component or subsystem.

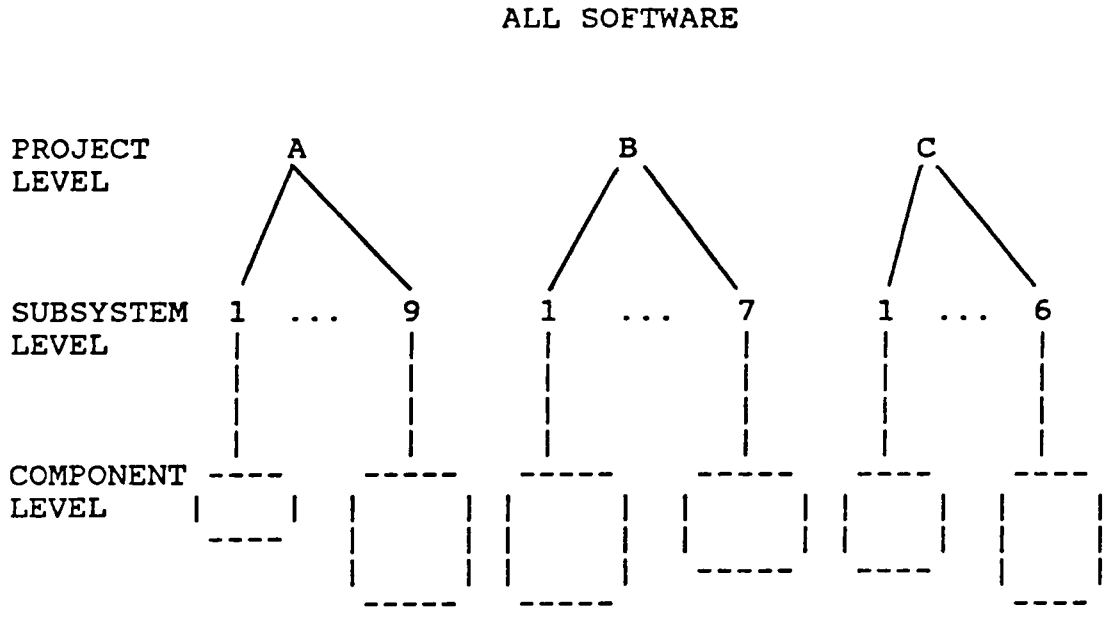


Figure 5: Software Hierarchy for Nasa/Goddard

2. Total Code Time: The time recorded by SEL personnel to implement, to read and to review the coding of an individual component or subsystem.
3. Coding Time: The time recorded by SEL personnel just to implement an individual component or subsystem.
4. Test Time: The time recorded by SEL personnel to test an individual component or subsystem, including unit testing, integration testing, and testing reviews.
5. Total Time: The time attributed to the entire development of a component or subsystem.

In addition to the two count based variables mentioned above, a third count based variable, WEIGHTED CHANGES, was derived for the purposes of this research. This is a measure of the total amount of effort spent either to fix an error or to make a change to a given component[Basi83]. In the Nasa/Goddard environment, programmers classify each change by estimating the amount of effort needed to isolate the change/error and to implement the change/error. The four possible classifications recorded in the developmental database are:

1. Less than one hour.
2. One hour to one day.
3. One day to three days.

4. Over three days.

Corresponding to each of these classifications are the four weights suggested by Basili: 0.5, 4.5, 16.0, or 32.0 hours. For each change, the appropriate weight is divided equally among all components involved in the change. Thus, a component's WEIGHTED CHANGE value is derived by summing the hours attributed to the component for every change with which it was involved.

Chapter IV

INITIAL COMPARISONS BETWEEN METRICS

The purpose of this chapter is to present results of statistical comparisons made between various software complexity metrics. These results were generated by experiments which applied both the structure metrics and the implementation metrics to three fortran systems constructed at Nasa/ Goddard Space Flight Center. Througout the chapter, Spearman correlation coefficients will be presented and used to form the basis of all comparisons.

The usefulness of these comparisons are their potential to identify the different factors contributing to software complexity during the design and implementation phases of the lifecycle. As such, it was important for this study to partition the experiments into two separate categories. One category of experimentation correlates the four structure metrics with themselves. The other category associates the implementation metrics with both themselves and with the structure metrics. This distinction is necessary if the relationships between metrics which are applied during the same phase of the life cycle are to be understood. The structure metrics are different from the implementation metrics since they can be extracted during the design phase of

the life cycle. However, structure metrics may also be taken during latter phases of development and therefore need to be compared with the implementation metrics as well.

The existance of the multiple factors complicating both the design and implementation processes is generally accepted. The existance of these factors during the design phase can be deduced by the large number of design methodologies. One methodology, that of Jackson[Jack75], builds a software design by using a data structure orientation. Other methodologies such as the Structured Analysis and Design Technique (SADT) or the Systematic Activity Modeling Method (SAMM) rely on data flow diagrams to convey the intent of the software. The existance of multiple factors contributing to software complexity during the implementation phase of the life-cyle has also been experimentally verified. Sime et al.[Sime73], reported that programming effort varied with the utilization of different control flow constructs. Dunsmore and Gannon [Duns80], reported that at least five software properties, including nesting levels and data dependent factors were also related to programming effort.

These and similar experiments have pioneered the identification of potentially important software metrics which can be applied during the implementation phase of the life cycle. Furthermore, intuition leads us to believe that soft-

ware is indeed multi-dimensional. For years, the simplistic Lines of Code metric used to measure programmer performance has been grossly inadequate. Two programs of equal size but possessing different control and data flow characteristics will not, in general, be of equal complexity. It is also logical to assume that metrics which incorporate the measurement of a component's interface properties will differ from those which merely measure the component's internal composition.

Underlying the definition of each metric used in this research are intuitions about one or more factors contributing to software complexity. The Unweighted Information Flow Metric, INFOFLOW, and the Unweighted Stability Measure, STABILITY, identify two methods of quantifying data flow within a design. McClure's Invocation Complexity, INVOKE, measures systemwide control flow properties, while Woodfield's Unweighted Interconnection Model, REVIEW, counts both the data and control flow characteristics of a software product. No previous research has been in a position to apply these four unique structure measures to the same software system and determine the actual similarities and differences in their behavior. Furthermore, the code metrics offer additional methods of identifying complexity. Halstead's EFFORT and the Lines of Code (LOC) measure are two

size based metrics, while McCabe's cyclomatic complexity, CYCLO, proposes a technique to measure internal control flow. By using these metrics in combination, as well as individually, software designers, implementors, and maintainers may better comprehend the ramifications of their decisions. For example, small increases in control flow complexity could lead to dramatic reductions in data flow complexity. Before understanding the tradeoffs between each of the proposed metrics, it is first necessary to classify those complexity measures which exhibit different and similar behavior.

As previously mentioned, the following sections of this chapter will present Spearman correlation coefficients reflecting the association among the measurements obtained by software metrics from three Fortran projects constructed as Nasa/Goddard. The following section, section 3.1, reports on experimentation which explored the impact of various weighting terms on the Information Flow Metric, the Interconnection Model and the Stability Measure. In the original definition of each of these three measures, the author either used or arbitrarily suggested a given code metric as an appropriate weight. The purpose of the subsequent two sections is to confirm and augment results found by earlier researchers. Section 3.2 provides results on the comparison

between the LOC metric, Halstead's EFFORT measure and McCabe's CYCLO measure. A number of previous studies [Hayes80], [Basi81], [Curt79a], [Craw82] have compared various code metrics typically finding that these count based measurements identify similar properties of complexity. Section 3.3 presents correlations which associate the four structure metrics with the three code metrics. Promising work by Kafura and Henry [Kafu82] introduced evidence that a given structure metric (Information Flow) provides a view of a software system (Unix) which is independent of those views generated by the three code metrics. The findings in this section concur with this result and expand it to encompass the other structure metrics. Section 3.4, completing the analysis involving code metrics, describes the correlations between them and the hybrid metrics. Closing out the presentation, section 3.5 presents one table which compares the structure metrics with themselves and a second table which relates the hybrid metrics with both themselves and the structure metrics. The chapter concludes with a brief summary.

4.1 IMPACT OF CODE METRICS ON HYBRID METRICS

Recall that hybrid metrics are a composite of structure values and code measurements. The code measurements are used to give weight to a components interface complexity. This section reports on the effect the three code metrics: Lines of Code(LOC), Halstead's EFFORT and McCabe's Cyclomatic Number(CYCLO), have on the various hybrid metrics. The following three tables contain the Spearman correlation coefficients generated when analyzing the three projects independently as well as collectively. Each entry in all three tables contain the four coefficients respectively corresponding to Project A, Project B, Project C and all three projects together. For example, in table 1 the coefficients between the unweighted information flow metric and the information flow metric weighted by McCabe's Cyclomatic Number are: .982 (Project A), .977 (Project B), .982 (Project C) and .981 (All Projects).

The most striking fact found in table 1 are the tremendously high Spearman coefficients implying that all four metrics are strongly associated. There appears to be little difference between the three information flow hybrid metrics. Moreover, the results of this experiment indicate that the differences between the unweighted information flow metric and the three information flow hybrid metrics are

neglible. This result, due to the dominance of the squared structure term in the flow metric's definition, allows us to omit without any loss both the INFO-CYCLO metric and the INFO-EFF metric in the remainder of this work. Only the unweighted information flow metric, INFOFLOW, and the information flow metric weighted by lines of code, INFO-LOC, will be used in subsequent analyses. The INFOFLOW metric was retained because the information flow metric was initially proported to be a design metric. Since this metric does not incorporate an implementation dependent weighting term, it can indeed be measured at design time. The INFO-LOC metric was also not omitted since Henry incorporated the LOC weighted term in the original definition[Henr79] This will allow for a more direct comparison between the previous UNIX study [Henr84] and the results of this research.

The results comparing the four metrics derived from Woodfield's Interconnection Model are presented in table 2. Again we find that the hybrid forms of the model vary little. The coefficients between the metrics weighted by either Lines of Code, Halstead's Effort, or McCabe's Cyclomatic Complexity range from .844 to .989. Although the association between the metrics are not as strong as in the case of the information flow hybrids, it is certaining strong enough to warrant the elimination of two of these hybrid metrics

TABLE 1

Effect of Code Metrics on Information Flow

	INFOFLOW	INFO-EFF	INFO-CYCLO	INFO-LOC
INFOFLOW	1.00	.986	.982	.985
	1.00	.986	.977	.984
	1.00	.987	.982	.984
	1.00	.986	.981	.985
INFO-EFF		1.00	.990	.998
		1.00	.992	.998
		1.00	.995	.998
		1.00	.992	.998
INFO-CYCLO			1.00	.992
			1.00	.993
			1.00	.995
			1.00	.981
INFO-LOC				1.00
				1.00
				1.00
				1.00

KEY: $p < 0.01$

from further consideration. The selected hybrid metric, arbitrarally chosen as the REVIEW-LOC metric, and the REVIEW metric will both be retained for further study because the association between these two metrics is not nearly as pronounced. Coefficients between the REVIEW metric and the REVIEW-LOC metric range between .332 and .558 for three of the correlations. The somewhat anomalous 0.796 coefficient for Project C does not provide enough evidence to warrant the elimination of either metric.

The results showing the effect of incorporating various code measurements into the Yau and Collofello stability measure are reported in table 3. Once again the values in this table indicate that all three implementation dependent weights have similar effects on the overall measurement. As such, only the STAB-LOC metric along with the unweighted stability metric, STABILITY, will be used in subsequent analyses.

In this section the results of experimentation showing the effect various code metrics had on different hybrid metrics were presented. From these experiments it was determined that certain metrics were redundant. In all cases, the weighting factors: Lines of code(LOC), Halstead's effort(EFFORT) and McCabe's cyclomatic complexity(CYCLO) influenced the hybrid metrics in extremely similar manners.

TABLE 2

Effect of Code Metrics on Interconnection Model

	REVIEW	REVIEW-EFF	REVIEW-CYCLO	REVIEW-LOC
REVIEW	1.00	.510	.526	.516
	1.00	.303	.291	.332
	1.00	.794	.778	.796
	1.00	.543	.522	.558
REVIEW-EFF		1.00	.844	.978
		1.00	.932	.968
		1.00	.956	.989
		1.00	.895	.977
REVIEW-CYCLO			1.00	.871
			1.00	.924
			1.00	.950
			1.00	.899
REVIEW-LOC				1.00
				1.00
				1.00
				1.00

KEY: $p < 0.01$

TABLE 3

Effect of Code Metrics on the Stability Measure

	STABILITY	STAB-EFF	STAB-CYCLO	STAB-LOC
STABILITY	1.00	.639	.633	.642
	1.00	.715	.704	.704
	1.00	.715	.726	.712
	1.00	.671	.664	.672
STAB-EFF		1.00	.964	.992
		1.00	.981	.991
		1.00	.980	.994
		1.00	.972	.992
STAB-CYCLO			1.00	.971
			1.00	.980
			1.00	.980
			1.00	.974
STAB-LOC				1.00
				1.00
				1.00
				1.00

KEY: $p < 0.01$

Thus, the results of this section have pruned the list of proposed metrics to the following ten measurements:

1. Unweighted information flow metric (INFOFLOW)
2. Unweighted interconnection model (REVIEW)
3. Unweighted stability measure (STABILITY)
4. Invocation complexity measure(INVOKE)
5. Halstead's effort(EFFORT)
6. McCabe's cyclomatic complexity(CYCLO)
7. Lines of code(LOC)
8. Information flow metric weighted with LOC (INFO-LOC)
9. Interconnection model weighted with LOC (REVIEW-LOC)
10. Stability measure weighted with LOC (STAB-LOC)

Additional comparisons among these ten software metrics are contained in the following sections.

4.2 CODE METRICS CORRELATED WITH CODE METRICS

The Spearman rank correlation coefficients showing the strength of association between the three code metrics are provided in table 4. As in the previous section the first three coefficients in each table entry correspond to values obtained by analyzing each of the three projects individually and the fourth coefficient corresponds to the combined components of all three projects. The extremely high values shown in table 4 are in accord with previous research

[Henr79][Redd84][Craw85] and once again illustrates the strong relationships between Halstead's effort, McCabe's cyclomatic complexity and the Lines of code metric. This strong correlation exists regardless of the project being measured. The results from these three projects together with past research on drastically different software systems undermine the usefulness of a collection of only code metrics.

4.3 STRUCTURE METRICS CORRELATED WITH CODE METRICS

Table 5 contains the Spearman rank coefficients of the four structure metrics correlated with the three code metrics. The most striking difference between table 4 and table 5 is the presence of consistently smaller coefficients. This reaffirms Henry and Kafura's[Henr81b] hypotheses that structure metrics are measuring different properties of component complexity than their code metric counterparts.

The Spearman coefficients associating the information flow metric with the three code metrics range from 0.50 - 0.70. This is somewhat higher than the range of 0.35 - 0.38 cited in Henry and Kafura's earlier paper. However, these smaller coefficients were generated by using the Pearson correlation method. In this study, the more conservative non-parametric Spearman method was employed since the Goddard data did not

TABLE 4

Code Metrics Correlated With Code Metrics

	LOC	CYCLO	EFFORT
	---	-----	------
LOC	1.00	.846	.972
	1.00	.895	.961
	1.00	.902	.978
	1.00	.864	.967
CYCLO		1.00	.808
		1.00	.893
		1.00	.889
		1.00	.855
EFFORT			1.00
			1.00
			1.00
			1.00

KEY: $p < 0.01$

TABLE 5

Structure Metrics Correlated With Code Metrics

	LOC	EFFORT	CYCLO
INFOFLOW	.70	.68	.66
	.50	.59	.46
	.63	.65	.55
	.62	.63	.55
REVIEW	?	?	?
	?	?	?
	?	?	?
	.20	.18	.16
STABILITY	.52	.50	.51
	.39	.50	.41
	.55	.59	.51
	.49	.51	.46
INVOKE	.43	.40	.44
	.59	.51	.41
	.47	.47	.40
	.46	.44	.40

KEY: ? p > 0.01, otherwise p < 0.01

consistently exhibit properties necessary for the use of parametric statistics. Let it be noted for the sake of comparison, that the associated Pearson coefficients for the unweighted information flow metric correlated with the three code metrics fall within the 0.26 - 0.45 range. The results correlating the remaining three structure metrics with the code metrics are certainly encouraging. Spearman coefficients for McClure's invocation complexity and the code measurements range between 0.40 - 0.59, while correlations between the unweighted stabilty measure and the code metrics also fall near this range. Three of the experiments utilizing Woodfield's unweighted interconnection model yielded results which were not significant at the 0.01 level, however, when all components were analyzed collectively, the interconnection model proved to be highly unrelated to the code metrics with values ranging form 0.16 - 0.20 .

4.4 CODE METRICS CORRELATED WITH HYBRID METRICS

The Spearman coefficients in table 6 confirm and highlight the differences between the hybrid version of the interconnection model measure (REVIEW-LOC) and the hybrid version of the stability measure(STAB-LOC). In the interconnection model, the dominant terms of the metric are the code values extracted from the component being measured.

TABLE 6

Code Metrics Correlated With Hybrid Metrics

	LOC	EFFORT	CYCLO
REVIEW-LOC	.97	.94	.84
	.97	.93	.85
	.50	.51	.44
	.84	.81	.71
STAB-LOC	.31	.30	.32
	.26	.36	.28
	.39	.44	.44
	.31	.35	.41
INFO-LOC	.70	.70	.68
	.63	.70	.57
	.70	.71	.65
	.74	.74	.65

KEY: $p < 0.01$

Thus, correlations between Woodfield's REVIEW-LOC metric and associated code metrics tend to be quite strong. Only the somewhat smaller values generated by analysis of PROJECT C indicate the possible presence of orthogonality between the metrics. Contrast these values with the noticebly smaller correlations between the STAB-LOC measure and the various code metrics. This lesser degree of association is explainable since the LOC metric which gives weight to the stability measure are not taken from the component being measured, but from those components with which it interfaces. Intuitively, it is not to be expected that any relationship should exist between the LOC value of a given component and the LOC value of those components with which it interfaces. The Spearman correlations relating the three code metrics and the INFO-LOC metric range between 0.57 and 0.74 . These values indicate that the information flow metric weighted by lines of code relates more closely to the code metrics than the STAB-LOC metric. This is due to the fact that the LOC value for the INFO-LOC metrics is taken from the component being measured. However, the correlations between the INFO-LOC metric and the assorted code metrics are not as high as those with the REVIEW-LOC metric because the information flow measure is not dominated by lines of code term, as is the case for the interconnection model.

4.5 STRUCTURE/HYBRID METRICS CORRELATED WITH STRUCTURE/HYBRID METRICS

No previous work has investigated the relationship between various structure metrics or their relationship to hybrid metrics. It is the purpose of this section to present Spearman rank coefficients derived by conducting such an investigation. Two tables are given in this section. The first table contains coefficients relating the structure metrics to themselves, while the second table lists the coefficients generated by correlating the hybrid metrics with themselves as well as with the structure metrics.

The coefficients listed in table 7 were obtained by analyzing the components across all three Nasa/Goddard projects. Results from the analysis of the three individual projects were extremely similar and are not given. These results are indeed encouraging. The absence of the extremely high coefficients found in table 4 indicate that unlike the implementation dependent code metrics, structure metrics are measuring different properties of software complexity. The intuitive differences between a control flow metric, McClure's invocation measure, and data flow measurements such as the information flow and stability metrics, are borne out by the Spearman coefficients. In addition, there is an even weaker association between Woodfield's Interconnection model and the three remaining structure metrics. It

is not too surprising that the strongest association (0.711) among all structure metrics is found between the data flow based measurements of Henry/Kafura and Yau/Collofello.

This section concludes with the presention of table 8, containing Spearman coefficients generated from analysis of the components from all three projects. The coefficients in table 8 contain correlations indicating the relationship between themselves and their relationship to the four structure metrics. Once again the results of similar experimentation applied to the three individual projects is analogous. The correlations present in this table show that the three hybrid metrics are generally independent measures of complexity. The STAB-LOC metric and the REVIEW-LOC metric have a particularly weak relationship (0.298). Furthermore, McCLure's INVOKE measure is not strongly related to any of the hybrid metrics.

4.6 SUMMARY AND CONCLUSIONS

Some significant observations can be made from experimental results reported in this chapter. First, the original information flow metric proposed by Henry and Kafura contained an unnecessary code metric weighting term. The data in table 1 shows that the dominant term of the metric is the one which captures the data flow interfaces between compo-

TABLE 7

Structure Metrics Correlated With Structure Metrics

	INFOFLOW	STABILITY	REVIEW	INVOKE
INFOFLOW	1.000	0.711	0.383	0.462
STABILITY		1.000	0.266	0.479
REVIEW			1.000	0.203
INVOKE				1.000

KEY: $p < 0.05$

TABLE 8

Hybrid Metrics Correlated With Structure/Hybrid Metrics

	STAB-LOC	REVIEW-LOC	INFO-LOC
INFOFLOW	0.450	0.598	0.985
STABILITY	0.672	0.555	0.714
REVIEW	0.179	0.558	0.336
INVOKE	0.363	0.365	0.492
STAB-LOC	1.000	0.298	0.446
REVIEW-LOC	0.298	1.000	0.685

KEY: $p < 0.01$

nents. This is a satisfying result since the elimination of the code metric weighting term now allows the formation of a measurement which can be taken at design time. Second, it was shown that many of the hybrid metrics formed by using different code metric weights produce redundant measurements. Third, results in this chapter confirm previous research which state that code metrics are highly related to each other. This implies that only one code metric need be included in the complete correlation of metrics used to measure the various factors influencing software systems. Fourth, encouraging evidence was presented emphasizing the potential orthogonal properties between structure metrics and implementation dependent code metrics. This result agrees with the pilot study of Henry and Kafura and is in accord with the intuition that structure metrics quantify interface complexity while code metrics capture internal complexity. Even more pleasing was the fact that structure metrics tend to be orthogonal to each other. This potentially will allow designers of software systems more than a single view of software complexity. The chapter has presented encouraging results which indicate the existance of potentially orthogonal metrics. In subsequent chapters, results showing the performance of these metrics when compared to various operationally defined complexities will be given.

Chapter V

COMPONENT LEVEL AND SUBSYSTEM LEVEL ANALYSIS

This chapter reports on experiments which related a set of software complexity metrics to data collected by the SEL personnel during the development of three software systems. The general format of these experiments is depicted in figure 6. As illustrated, during the construction of software systems at Nasa/Goddard, information quantifying various aspects of the software process and product is recorded in a database. For our purposes, two types of measures extracted from this database were used as dependent variables: count based measures and time based measures. Count based measures such as, ERRORS, CHANGES and WEIGHTED CHANGES were reported for each component. Time based measures such as, DESIGN TIME, CODING TIME and TESTING TIME were also reported for each component. Furthermore, time based measures were reported for each subsystem within a given project. The times attributed to the development of an entire subsystem are generally higher than the summation of the times attributed to components within that subsystem. This discrepency arises because reporters of time based data do not always attribute their efforts to individual components. For example, much of the design time and testing time data was re-

ported on a subsystem level since significant design and testing activity involves collections of components.

The initial experiments conducted for this research, comparing the set of software metrics with observed measures of software complexity recorded in the database, are partitioned into two classes. The first class of experiments, referred to as component level analysis, compared individual component complexities, as determined by the ten software metrics, with their reported count based and time based data. The second class of experiments, known as subsystem level analysis, related individual subsystem complexities with their associated count based and time based values. A subsystem's software complexity is defined by summing the metric values of its individual components. Count based values were similarly established for each subsystem. Time based measures of entire subsystems reflect the time spent developing individual components as well as any overhead time attributed to the subsystem as a whole. In both classes of experiments, Spearman rank correlation coefficients formed the basis of the comparison. Of course, underlying each of these analyses is the assumption that observed data, such as error counts and time to develop software, is highly associatied with complexity. For example, as software complexity increases it is assumed that the number of errors

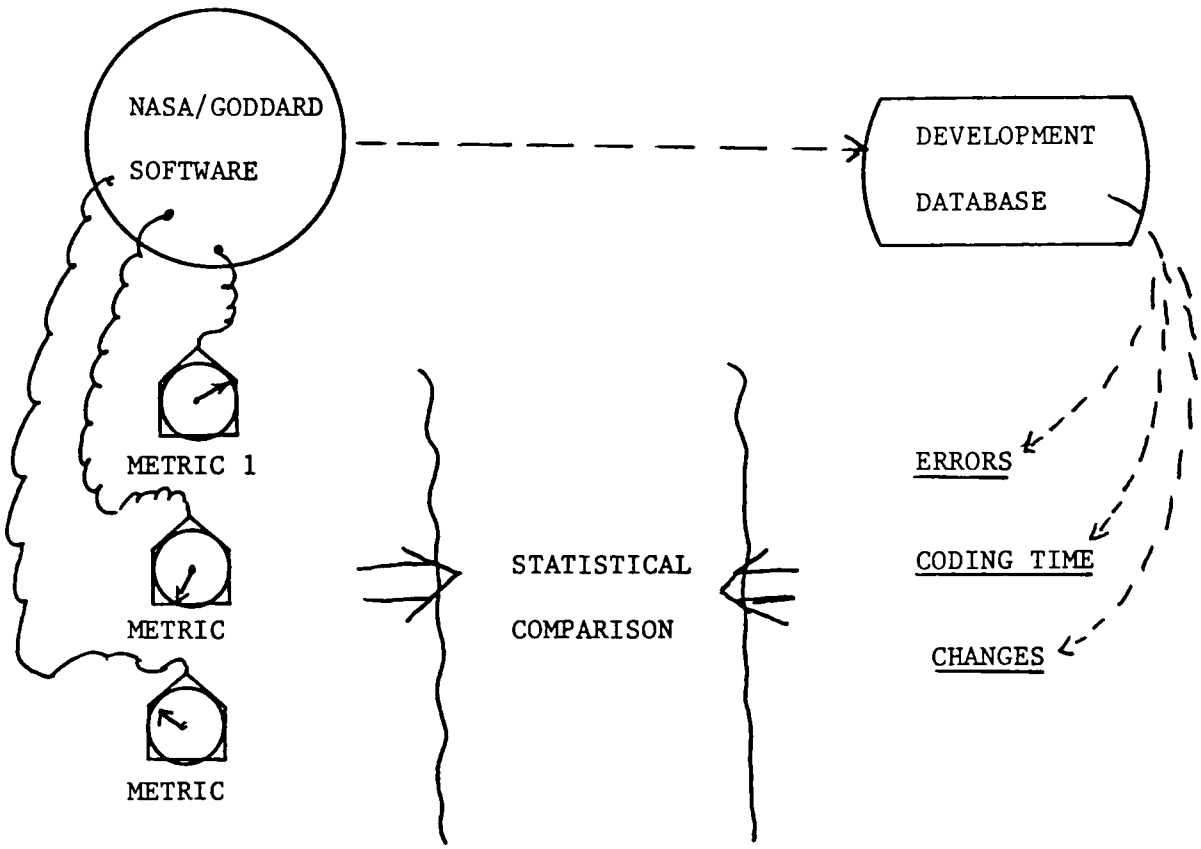


Figure 6: General format of Experimentation

found in the software or the time to develop the software should also increase.

The remainder of this chapter is organized into the following four sections. The next section identifies the two subsets of the Nasa/Goddard data which were used for these experiments. Section three provides the results of the component level analysis. The somewhat low correlations presented in section three are consistent with past research [Bas83]. This may indicate that software metrics may not be sensitive to the subtle differences between components. Section four presents the results of the subsystem level analysis. The final section of the chapter provides a summary of the presented results.

5.1 THE DATA

Because of the nature of the data collection process and the enviroment in which the systems were constructed, not all of the components from the three projects were used in this experimentation. This section will describe the two subsets of the Nasa/Goddard data that were used in the following analyses. One subset of components, referred to as DATASET A, was utilized in those experiments which compared the metrics with the count based dependent variables (ERRORS, CHANGES, WEIGHTED CHANGES). In particular, DATASET

A does not include those components which were entirely or largely reused from prior projects. The second subset of components, known as DATASET B, was used in those experiments which related the metrics to time base dependent variables. This set of components is a subset of DATASET A. The selection process, or "filter", used to produce DATASET B, was found to be necessary in order to adjust for problems with the data reporting mechanisms. Each of these two sets of components is described in more detail in the following subsections.

5.1.1 Components Used for Count Based Experimentation:Dataset A

One subset of components (DATASET A) used for this research contains components which will be used when analyzing the relationship between the metrics and the count based measures. DATASET A is a collection of those subroutines and functions which were either totally new components or extremely modified old components. The SEL database classifies each component as either: new code, extremely modified old code, slightly modified old code, or an exact copy of old code. It is necessary to omit slightly modified or duplicated code from consideration since error counts and development times for reused components do not accumulate from project to project in the SEL reporting mechanism. These

reused components, having already undergone rigorous testing and debugging when intially implemented, would distort the count based measurements. Similarly, the time based measurements would also be biased since they would not reflect the original effort expended in the development of these slightly modified or duplicated components. It was decided that extremely modified components should be incorporated into the analyses since these components undergo a significant transformation. It was felt that count based measures and time based measures would approximate the values of such a component had it been completely constructed anew. In support of these decisions, table 9 below provides the mean values for the four measures across the four component classifications. An analysis of variance was performed using dependent variables (ERRORS, CHANGES, WEIGHTED CHANGES, TOTAL TIME) to test the null hypothesis that mean values for the four component classes were not statistically different. In each of the four analyses of variance the null hypothesis was rejected (alpha=0.05). Furthermore, Fisher's Protected Least Significance Test (LSD) was used to identify those means which were significantly different. In all four tests, it was found that means from component class (1) and component class (2) were not significantly different, but the means for these two classes were significantly different from both class (3) and class (4).

TABLE 9

Data Means for Various Component Types

Component Class	Errors	Changes	Weighted Changes	Total Time
(1) New Code Components	2.03	2.11	44.20	16.09
(2) Extremely Modified Components	1.47	2.62	43.57	13.32
(3) Slightly Modified Components	0.43	1.11	18.98	6.06
(4) Duplicated Components	0.17	0.52	3.24	3.37

5.1.2 Components Used for Time Based Experimentation:Dataset B

Another subset of the Nasa/Goddard components(DATASET B) was used for experiments involving time based dependent variables. This collection of components not only omits slightly modified routines and duplicated code from the experimentation, but also eliminates components which fail to pass two additional criteria. Both of these criterion provide a validity check on the reporting of time based dependent variables.

The first criteria eliminates those components which were constructed by programmers who were identified as poor or inconsistant reporters of time based dependent variables. Programmers were considered poor reporters if the ratio, Vm, found below, was less than eighty percent. This ratio, first suggested by Basili et al. [Bas83], utilizes the partial redundancy built into the SEL monitoring process. Each week every programmer files a Component Status Report (CSR) describing the time they spent on each module. Also on a weekly basis, the project manager files a Resource Summary Form (RSF) recording the time each programmer spent on the project during that week. Basili et al. have indicated that the manager reported information found in the Resource Summary Form is considered to be the more accurate. Thus, if a given programmer fails to submit a Component Status Report

for a given week, this would lower his Vm ratio. The Vm ratio is defined as:

$$Vm = \frac{\text{Number of weekly CSR's submitted by programmer}}{\text{Number of weeks programmer appears on RSF's}}$$

The second criterion is based on the fact that time based dependent variables are not only reported for individual components, but are also reported for individual subsystems. Programmers typically report their time spent on an individual component basis. However, a programmer may choose to attribute work hours to an entire subsystem if these hours cannot be accurately partitioned among that subsystem's individual components. When the time reporting is done on a subsystem level, information at the component level is lost. If less than eighty percent of a subsystem's reported time is attributed to individual components, then these components are not incorporated into DATASET B.

5.2 COMPONENT LEVEL ANALYSIS

This section presents the Spearman rank coefficients obtained by correlating, on a component by component basis, the ten complexity metrics with four operational measures of complexity: ERRORS, CHANGES, WEIGHTED CHANGES and CODING TIME. The goal of this type of analysis is to determine how strongly the proposed complexity metrics relate to errors and actual effort encountered during the development of software in a realistic environment [Bas83]. The other time based measures, design time and testing time, were not included in this analysis since these values were largely reported on a subsystem basis. The tables containing the Spearman coefficients are presented in the following two subsections. The first subsection contains the data relating the complexity metrics with the count based measurements, while the second subsection contains the data relating the metrics to coding time.

5.2.1 Metrics Related to ERRORS, CHANGES and WEIGHTED CHANGES

In this subsection, Spearman rank coefficients relating the complexity metrics to three count based variables are presented. As previously stated, not all components from the three projects were analyzed. Since, the metrics were related to count based variables, only newly developed or

extremely modified components (DATASET A) were used in this experimentation. The components of DATASET A can be partitioned hierarchically by project and by subsystem, thus giving rise to three separate but similar experiments. The first experiment collectively measures all the components of DATASET A yielding correlations which span across all three projects. The second experiment attempts to filter out project related factors by correlating components within the same project. Finally, components within the same subsystem were collectively measured. Correlating components within the same subsystem helped to filter out programmer related factors since in the Nasa/Goddard environment, subsystems are typically developed by one or two programmers.

The Spearman coefficients derived by collectively measuring components which span across projects are found in table 10. The correlations between the metrics and both the ERROR data and the CHANGE data are rather low indicating little or no relationship between them. An attempt to improve the correlations was made by weighting each change, as described in chapter 2, and rerunning the experiment. However, no significant improvements in the correlations could be obtained. Although these results are somewhat disappointing, they were not totally unexpected. Previous research by Basili et al. [Bas83], found similar results when comparing er-

rors with a host of software science metrics. An identical weighting criteria was also applied to errors resulting in similar correlations being generated. These authors noted that the discrete nature of error reporting, with 52 percent of their tested components having zero errors, could account for the low Spearman correlations. Analogously, the distribution of the error and change data used in this experimentation is similarly skewed.

In an attempt to account for project dependent effects, the components of DATASET A were partitioned and analyzed by project. Previous experiments by Basili[Bas83], analyzing different data generated from the same environment(Nasa/Goddard) pointed toward the existance of project related factors. The Spearman correlations between the metrics and the error data for the three individual projects are presented in table 11. For comparison, this table also includes the across project correlations. The correlations presented in table 11 indicate that almost all the metrics show stronger associations with ERRORS for PROJECT A and PROJECT C than they did for the across project analysis. In particular the three code metrics: LOC (.627), CYCLO(.513), and EFFORT(.610) , exhibit noticeably higher correlations with ERRORS when applied to the components of PROJECT C. Of the three hybrid metrics, the INFO-LOC metric was the only

TABLE 10

Metrics Correlated With Count Based Data

	ERRORS	CHANGES	WEIGHTED CHANGES
LOC	.470	.497	.372
CYCLO	.373	.455	.402
EFFORT	.467	.512	.413
INFO-LOC	.364	.452	.338
REVIEW-LOC	.254	.285	.347
STAB-LOC	.219	.272	.276
INFOFLOW	.321	.415	.313
INVOKE	.398	.440	.360
REVIEW	?	?	?
STABILITY	.174	.267	.232

KEY: ? P > .05 , OTHERWISE P < .01

measure to show significant improvements in the correlations. Again, these improvements are present for both PROJECT A and PROJECT C. The pure structure metrics have somewhat weaker associations to ERRORS than do either the code or hybrid metrics. However, of four structure metrics, Henry's INFOFLOW and McClure's INVOKE measure exhibit the strongest relationship to ERRORS.

Although, the correlations found in table 11 are not extremely high, they usually indicated stronger relationships between the metrics and the error data when project related factors are minimized. The unexplained exception to this trend are the correlations between the code metrics and errors for project B components. It is interesting to note that Basili's previous work also noted an anomalous project.

Similar observations can be made when comparing the metrics to the other two count based metrics. The correlations between the metrics and CHANGES for the three projects are presented in table 12. The correlations found in this table are noticeably higher for PROJECT C. In particular, the correlations for the three code metrics and the INFO-LOC metric are all relatively high, exceeding 0.600 . The correlations generated by the unweighted information flow metric, INFOFLOW have also improved, jumping from 0.415 to 0.582 . It is interesting to note that the correlations between

TABLE 11

Component Analysis Within Projects -- ERRORS

	ALL PROJECTS	PROJ. A	PROJ. B	PROJ. C
LOC	.470	.506	.257	.627
CYCLO	.373	.390	?	.513
EFFORT	.467	.494	.189	.610
INFO-LOC	.364	.496	?	.536
REVIEW-LOC	.254	.509	.281	?
STAB-LOC	.219	.175	.267	.257
INFOFLOW	.321	.462	?	.489
INVOCATION	.398	.331	.406	.479
REVIEW	?	.250	.209	-.249
STABILITY	.174	.228	?	.287

KEY: ? P > .05 , OTHERWISE P < .01

CHANGES and the metrics for PROJECT A are highest for the two information flow based metrics. Three metrics, STABILITY, REVIEW, and STAB-LOC consistently generate correlations which indicate weaker associations with CHANGES than the other metrics.

Table 13 contains for the Spearman values for comparisons made between the metrics and WEIGHTED CHANGES for only PROJECT A and PROJECT B, as weighted change data was not available for PROJECT C. The correlations obtained by analyzing components for PROJECT A are somewhat stronger than those obtained from PROJECT B's components. However, in either case, the within project correlations are not noticeably higher than the across project correlations.

Athough correlations between the metrics and the count based data were generally stronger when components were partitioned by project, these Spearman values were still found to be somewhat low. Hypothesizing the existance of "within projects" effects, the data was further partitioned by subsystems. Correlating components within the same subsystem helped to filter out programmer related factors since in the Nasa/Goddard environment, subsystems are typically developed by one or two programmers. The results of these correlations are reported in table 14, table 15 and table 16. The Spearman coefficients found in table 14 result from correlating

TABLE 12

Component Analysis Within Projects: CHANGES

	ALL PROJECTS	PROJ. A	PROJ. B	PROJ. C
LOC	.497	.487	.309	.682
CYCLO	.455	.444	.210	.603
EFFORT	.512	.468	.304	.679
INFO-LOC	.452	.540	.269	.630
REVIEW-LOC	.285	.477	.352	?
STAB-LOC	.272	.239	.381	.318
INFOFLOW	.415	.527	.240	.582
INVOKE	.440	.455	.437	.482
REVIEW	?	.212	.291	-.233
STABILITY	.267	.338	.242	.370

KEY: ? P > .05, OTHERWISE P < .01

TABLE 13

Component Analysis Within Projects: WEIGHTED CHANGES

	ACROSS	PROJ. A	PROJ. B
LOC	.372	.409	.298
CYCLO	.402	.394	.308
EFFORT	.413	.390	.386
INFO-LOC	.338	.469	.313
REVIEW-LOC	.347	.402	.305
STAB-LOC	.276	.248	.443
INFOFLOW	.313	.463	.282
INVOKE	.360	.427	.358
REVIEW	?	.163	?
STABILITY	.232	.297	.365

KEY: ? P > .05, OTHERWISE P < .01

the code metric values with the CHANGE data for components within fifteen subsystems. Six subsystems were omitted from the table since all their correlations had significance levels much greater than 0.05 . For many of the subsystems the correlations between the code metrics and the CHANGE data are dramatically higher than those obtained from previous experiments. This increase may indicate the presence of "within subsystem" effects. Similarly, the within subsystem correlations associating the CHANGE data with the hybrid metrics are noticeably higher. Comparing the correlations in table 14 with table 15 it appears that both INFO-LOC and REVIEW-LOC typically perform as well as the code metrics, and at times indicate stronger associations to CHANGES than do the code metrics. The correlations in table 16 give the within subsystem correlations between the four structure metrics and the CHANGE data. Three metrics: INFOFLOW, STABILITY, and INVOKE typically possess higher coefficients than the across project or within project correlations involving structure metrics.

The Spearman coefficients given in table 17 show the association between the ERROR data and selected metrics for components within a given subsystem. These metrics were selected since they indicated the strongest association to ERRORS for the three metric classes: code, hybrid and struc-

TABLE 14

Code Metrics Correlated with CHANGES:Within Subsystem

SUBSYSTEM	LOC	CYCLO	EFFORT
A.1	.771	?	.680
A.3	.695	.702	.644
A.4	.574	.640	.540
A.5	.487	.488	.450
A.6	.613	.545	.590
A.7	.550	.441	.518
B.1	.714	.642	.742
B.2	.571	.532	.631
B.3	.658	.569	.665
B.4	.815	?	.815
B.5	.716	.681	.728
C.1	?	?	.948
C.2	.400	.374	.524
C.3	.434	.655	.518
C.6	.357	.318	.409

KEY: ? p > .05, otherwise p < .05

TABLE 15

Hybrid Metrics Correlated With CHANGES:Within Subsystem

SUBSYSTEM	INFO-LOC	REVIEW-LOC	STAB-LOC
A.1	.862	.680	?
A.3	.695	.725	.405
A.4	.467	.511	.484
A.5	.612	.493	?
A.6	.553	.618	?
A.7	.552	.541	.509
B.1	.664	.687	.450
B.2	.491	.502	?
B.3	.726	.700	?
B.4	.926	.741	?
B.5	.632	?	.327
C.2	.588	.427	.644
C.3	.531	?	.604
C.6	.509	.344	.354

KEY: ? p > .05, otherwise p < .05

TABLE 16

Structure Metrics Correlated With CHANGES:Within Subsystem

SUBSYSTEM	INFOFLOW	REVIEW	STABILITY	INVOKE
A.1	.862	?	.669	.596
A.3	.684	?	.391	.393
A.4	.451	?	.595	.694
A.5	.630	.344	?	.616
A.6	.477	.276	.261	.474
A.7	.476	?	?	?
A.8	.402	?	.554	.453
B.1	.613	?	.652	.619
B.2	.467	?	?	?
B.3	.732	?	.476	?
B.4	.963	?	?	?
B.5	.544	?	.485	.372
C.2	.577	?	.623	?
C.3	.543	?	.526	?
C.6	.434	?	.397	.418

KEY ? p > .05. otherwise p < .05

TABLE 17

Selected Metrics Correlated With ERRORS

SUBSYSTEM	LOC	INFO-LOC	INVOKE
A.1	.743	.844	?
A.3	.752	.721	.447
A.4	.556	?	.611
A.5	.323	.462	?
A.6	.452	.466	.409
B.1	.709	.637	.558
B.3	.652	.728	?
B.5	.684	.558	.342
C.2	.402	.562	?
C.3	.570	.628	?
C.6	.382	.401	.393

KEY: ? p > .05, otherwise p < .05

ture. Again the data in this table points toward the existance of subsystem related factors. It is interesting that the correlation between the ERROR data and the LOC metric for the three PROJECT B subsystems are relatively high, ranging from 0.652 to 0.709 . These coefficients are markedly higher than the 0.257 coefficient generated by relating the ERROR data and the LOC metric to all PROJECT B components. However, notice that no metric consistently reports strong associations with the components in all subsystems. For example, the coefficient between the LOC metric and subsystem A.3 is 0.752, while for subsystem A.5 it is 0.323 . Furthermore, notice that no single metric consistently reports stronger association to ERRORS than the other metrics. Of the eleven subsystems given in table 17, the INFO-LOC indicates the strongest association with the ERROR data for seven of the subsytems, while the LOC metric has the highest coefficients for three of the subsystems. McClure's INVOKE measure relates best to the ERROR data for subsytem A.4, with a coefficient of 0.611 .

5.2.2 Metrics Related to a Time Based Dependent Variable

In this subsection, the Spearman rank coefficients relating the complexity metrics to a single time based variable, CODING TIME, are presented. The other time based dependent

variables, such as DESIGN TIME and TESTING TIME, were not correlated with the metric values since these "times" were not typically reported on a component by component basis. As previously mentioned in subsection 1.1.2, since CODING TIME is the developmental data used in this analysis, only components from DATASET B will be measured. Once again, the components were partitioned hierarchically by project and by subsystem giving rise to three separate but similar experiments.

The results of the first two experiments are summarized in table 18. The first experiment collectively measured all the components of DATASET B yielding correlations which span across all three projects. The results of these correlations are given in in the column labelled ALL PROJECTS in table 18. From the data presented in this table, it appears that code metrics relate more strongly to CODING TIME than either structure or hybrid metrics. Among the structure and hybrid metrics, the INFOFLOW metric (.437) and the INFO-LOC metric (.479) indicate the strongest association to CODING TIME.

Also presented in table 18 are the results of correlations which partitioned the components of DATASET A by project. This experiment was conducted in order to control for project related factors. As can be seen, the coefficients for PROJECT A indicate stronger association to CODING TIME

TABLE 18

Spearman Correlations: Metrics With CODING TIME

	ALL PROJECTS	PROJ. A	PROJ. B	PROJ. C
LOC	.544	.634	.475	.576
CYCLO	.583	.674	.442	.581
EFFORT	.557	.583	.523	.369
INFO-LOC	.479	.616	.465	.517
REVIEW-LOC	.164	.583	.424	?
STAB-LOC	.271	.409	.320	.266
INVOKE	.286	.293	?	.350
INFOFLOW	.437	.599	.421	.468
STABILITY	.303	.468	.392	.306
REVIEW	-.180	?	?	-.327

KEY: ? P > .05, otherwise P < .05

for every metric. Furthermore, for some metrics, the correlations with CODING TIME also increases for PROJECT C components. As before, in the experiments with the count based data, improvements in the correlations are not so consistent for Project B.

The coefficients found in table 19, table 20, and table 21 were obtained by correlating the CODING TIME values with the ten metric values for components within the same subsytem. The coefficients found in table 19 indicate that the three code metrics all relate to CODING TIME in approximately the same manner. For any given subsystem small differences in the coefficients of the code metrics do exist. However, from subsystem to subsystem, fluctuations in the coefficients are generally consistent. For example, the coefficients relating the code metrics with CODING TIME for subsystem B.1 range between 0.562 and 0.595, while the coefficients for subsystem B.5 range between 0.745 and 0.773 .

The Spearman values found within table 20 were generated by correlating the three hybrid metrics with CODING TIME. These values indicate that no single hybrid metric is uniformly superior when relating to CODING TIME. For two of the subsystems (A.7 & B.1), the INFO-LOC metric correlated more strongly with CODING TIME than either the REVIEW-LOC

TABLE 19

Code Metrics Correlated With CODING TIME

SUBSYSTEM	LOC	CYCLO	EFFORT
A.3	.559	.498	.517
A.7	.438	.564	.400
B.1	.595	.562	.585
B.3	?	?	?
B.5	.773	.745	.770
C.3	.415	.493	.468
C.6	.522	.408	.527

KEY: ? P > .05, otherwise P < .05

TABLE 20

Hybrid Metrics Correlated With CODING TIME

SUBSYSTEM	INFO-LOC	REVIEW-LOC	STAB-LOC
A.3	.466	.555	.314
A.7	.668	.467	.523
B.1	.659	.595	.385
B.3	.586	?	?
B.5	.439	?	?
C.3	.595	?	?
C.6	.311	.484	.320

KEY: ? P > .05, otherwise P < .05

TABLE 21

Structure Metrics Correlated With CODING TIME

SUBSYSTEM	INVOKE	REVIEW	STABILITY	INFOFLOW
A.3	?	?	?	.454
A.7	.630	?	.687	.687
B.1	.602	.380	.572	.622
B.3	?	?	?	.626
B.5	.513	?	?	?
C.3	.521	?	?	.572
C.6	.459	?	?	?

KEY: ? P > .05, otherwise P < .05

metric or the STAB-LOC metric, while the REVIEW-LOC metric correlated more closely with CODING TIME for two other subsystems (A.3 & C.6).

The correlations between the four structure metrics and CODING TIME for components within the same subsystem are presented in table 21. The data in this table shows that the INFOFLOW metric yields the strongest correlations to CODING TIME for five of the seven subsystems. Of the two subsystems which the INFOFLOW metric did not yield a statistically significant result, McClure's INVOKE metric generated correlations of 0.513(B.5) and 0.469(C.6). It is difficult to assess the relationship between CODING TIME and both the REVIEW metric and the STABILITY metric since they often gave correlations which were statistically insignificant.

5.3 SUBSYSTEM LEVEL ANALYSIS

The results of the previous section were derived by correlating the the software metrics with the developmental data on a component by component basis. The purpose of this section is to present correlation coefficients between the software metrics and the developmental data where the unit of measure is not a single component, but rather an entire subsystem. In order to obtain such coefficients software complexity measures and developmental data values had to be

defined for each of the twenty-one subsytems used in this research. This was not difficult since in the Nasa/Goddard environment, components are grouped according to the subsystem in which they are defined. As previously mentioned, this grouping is well defined since components do not belong to more than a single subsystem. Communication between subsystems is typically achieved by exchanging data via system wide tables or by passing data through upper level modules. Thus software complexity measures for a given subsystem were derived by summing the complexities of each of its individual components. Similarly, the count based measures for a subsystem (ERRORS, WEIGHTED CHANGES) were established by totaling the count based measures for its individual components. Time based measures were obtained by not only summing the times associated with the development of individual components, but by also adding any overhead time attributed to an entire subsystem. This overhead time is primarily due to the inability of programmers to attribute their effort to individual components. As earlier stated, this often occurred during the design and testing phases of the software life-cycle, rendering component level analysis for these two activities impossible. However, for the subsystem level, five time based measures were incorporated into the analysis. These time measures are:

1. Design Hours: This is the time attributed to the creation of a substystem's design including design reading and design reviewing.
2. Code Total Hours: This is the time attributed to the coding of a subsystem including code reading and code reviewing.
3. Code Hours: This is the time attributed to the coding of a subsystem excluding code reading and code reviewing.
4. Test Hours: This is the time attributed to the testing of a subsystem. This includes unit testing, integration testing and testing reviews.
5. Total Time: This is the time attributed to the entire development of a subsystem.

There are four reasons why software systems need to be examined at the subsystem level. One motivation for studying software at the subsystem level is that typical large scale systems are oftened partitioned into sets of related components, i.e. subsystems. Usually these subsystems have a well defined structure of their own which can be isolated and evaluated as separate entities. It is important to identify those subsystems which have poor substructures and need to be redesigned. Furthermore, management decisions are not often tuned to the development of a single component, but

rather are guided by subsystem related information. For example, subsystems possessing an undue amount of complexity will be more difficult to design, build and maintain, hence will require more staffing. A second reason for studying the software at the subsystem level is that count based and time based data is difficult to collect and to verify at the component level. It is often not possible to identify a single component as the cause of an error or to correctly estimate the time spent working on a particular component. To achieve accurate data collection at such a fine level of detail may not be cost effective for most development environments. It would seem that collecting accurate count based and time based data at the subsystem level is more plausible. A third motivation for incorporating an analysis at the subsytem level is to investigate the relationships between the metrics and development times which span multiple phases of the software life-cycle. Recall, in the component level analysis this was not possible because development time data was available only for the coding phase. However, development time data was reported for the design, implementation and test phases of the lifecycle at the subsystem level. In particular, it would be of interest to identify metrics which are meaningful at all phases of a system's life-cycle. The fourth motivation for studying software at

the subsystem level is a direct result of the component level analysis presented earlier. The correlations in the previous section, although consistent with previous research, were disturbingly low. These disappointing numbers could exist for a variety of reasons. One reason could be that count based and time based data, being difficult to collect and to verify at the component level is too perturbed by random effects. Another, equally plausible reason for such poor relationships between the metrics and the development data at the component level is the metrics may not be able to detect subtle differences which exist between individual components. A third reason for such weak relationships found in the component level analysis will be discussed in the next chapter.

The results of the subsystem level analysis given below describes two similar but distinct experiments. The first experiment correlated the subsystem defined complexities with the subsystem defined data using all twenty-one subsytems available. The second experiment performed analogous correlations using only nine of the subsystem. Twelve of the subsystems were filtered out since they contained components which were constructed during the development of previous projects. Once again the conservative Spearman rank technique was used to generate correlations which indicated the

relationship between the metrics and both the time based and count based measures.

The Spearman coefficients obtained by correlating the five development times with the ten metrics for all twenty-one subsystems are given in table 22. One observation that is immediately apparent is that these Spearman rank coefficients indicate a much stronger relationship between the metrics and the time based data than did the corresponding analysis at the component level. The Spearman rank coefficients in table 22 indicate that the three code metrics: LENGTH, EFFORT, and CYCLO each induce an ordering on the subsystems that is highly similar to the ordering induced on them by any of the development times. Furthermore, with Spearman correlations ranging between 0.63 and 0.81, McClure's invocation complexity, INVOKE, is apparently the metric most closely related to five development times. Of the remaining structure based metrics, the stability metric of Yau and Collofello also indicates a strong rank association with TOTAL HOURS (0.62) and DESIGN HOURS (0.72). The ability of the information flow metrics and the review metrics to rank order the subsystems according to the various development times is generally weaker than the other proposed metrics.

TABLE 22

Complexity Metrics Correlated With Time Based Data
(N=21)

	TOTAL HOURS	DESIGN HOURS	CODE TOTAL HOURS	CODE HOURS	TEST HOURS
LOC	.697	.740	.607	.645	.603
EFFORT	.690	.710	.614	.633	.607
CYCLO	.707	.661	.636	.644	.684
INFOFLOW	.484	.580	?	?	.511
STABILITY	.620	.720	.540	.510	.444
REVIEW	.488	.587	?	?	?
INVOKE	.790	.811	.750	.692	.632
INFO-LOC	?	?	?	?	?
REVIEW-LOC	?	.528	?	?	.444
STAB-LOC	.624	.696	.542	.545	.468

KEY: ? p > 0.05, otherwise p < 0.05

The Spearman rank correlation coefficients in table 23 indicate relationships between the metrics and the count based data. Once again, it is apparent that collectively, the metrics display a much stronger relationship with ERRORS and CHANGES at the subsystem level than they do at the component level. These results indicate that code metrics possess a strong rank association with subsystem-wide errors and subsystem-wide changes. Collectively, the structure based metrics have a somewhat weaker relationship with ERRORS and CHANGES than do the code metrics. Of these, McClure's INVOKE measure relates the most strongly with errors(0.70) and changes (0.81). Furthermore, the STAB-LOC metric also indicates a rather strong relationship with ERRORS(0.67) and CHANGES(0.70).

In the above analysis, all twenty-one subsystems were used in the experiments. However, many of these subsystems contained components which were previously developed. As described above in subsection 4.1.1, count based and time based data for these components will likely be skewed. In an effort to control for this effect, the above experiments were redone using only the nine subsystems which contained no more than ten percent reused code. The Spearman correlation coefficients between the metrics and the time base data are presented below in table 24. The subsequent table con-

TABLE 23

Complexity Metrics Correlated With Count Based Data
(N=21)

	ERRORS	CHANGES
LOC	.823	.800
EFFORT	.754	.781
CYCLO	.669	.740
INFOFLOW	.635	?
STABILITY	.553	.650
REVIEW	?	.515
INVOKE	.700	.815
INFO-LOC	.613	?
REVIEW-LOC	.568	.562
STAB-LOC	.679	.707

KEY: ? p > .05, otherwise p < .05

tains the Spearman rank coefficients generated by correlating the metrics with the count based variables.

These tables indicate a much stronger association between the metrics and the data than the previous experiment which included all twenty-one subsystems. These increased correlations are attributed to the improvement in the quality of the data. Those subsystems which were largely made up of reused code contained components whose time and count data was artificially low, thus perturbing the experiment.

5.4 SUMMARY AND CONCLUSIONS

In this chapter, results were presented which related both structure and implementation metrics to observed measures of complexity. Initially the data had to be filtered to accomodate problems with the data collection facility. The correlation between metrics on both a component by component basis and on a subsystem by subsystem basis. When performing component level analysis; "across project", "within project" and "within subsystem" experiments were done. For the across project experiments, it was generally true that no metric satisfactorily explained the behavior of the count based or time based data. Partitioning the components on a "within project" basis generally enhanced the correlations for two of the three projects. Although the

TABLE 24

Complexity Metrics Correlated With Time Based Data
(N=9)

	TOTAL HOURS	DESIGN HOURS	CODE TOTAL HOURS	CODE HOURS	TEST HOURS
LOC	.816	.850	.683	.733	.733
EFFORT	.816	.850	.683	.733	.733
CYCLO	.850	.484	.783	.833	.766
INFOFLOW	.816	.850	.700	?	.816
STABILITY	.800	.900	?	.712	.700
REVIEW	.816	.850	.683	.733	.733
INVOKE	.733	.766	.667	.716	?
INFO-LOC	.716	.716	?	?	.766
REVIEW-LOC	.800	.833	.667	.750	.716
STAB-LOC	.667	.750	?	?	?

KEY: ? p > .05, otherwise p < .05

TABLE 25

Complexity Metrics Correlated With Count Based Data
(N=9)

	ERRORS	CHANGES
LOC	.928	.966
EFFORT	.928	.966
CYCLO	.861	.933
INFOFLOW	.778	.783
STABILITY	.928	.966
REVIEW	.928	.966
INVOKE	.711	.800
INFO-LOC	.769	.733
REVIEW-LOC	.895	.933
STAB-LOC	.878	.883

KEY: $p > 0.05$, otherwise $p < 0.05$

correlations were not extremely high, the code metrics as a class seemed to relate best to the count and time based data. Of the four structure metrics, the information flow metric indicated the strongest association with the observed data. Furthermore, when the information flow metric was weighted by lines of code, as in Henry's original definition, the correlations were somewhat higher. It seems that the unweighted REVIEW metric is not a very meaningful measure of complexity, but the REVIEW-LOC metric is able to show some association to the data for PROJECT A. The ability of some metrics to correlate with ERRORS,CHANGES, and CODING TIME generally improved when "within subsystem" factors were taken into consideration. When correlated with CHANGES, the INFOFLOW metric typically provided the strongest coefficients of all the structure metrics. However, for some subsystems, either the STABILITY metric or the INVOKE metric yield the highest coefficients. The inability of a single metric to consistently give the highest coefficients also occurs when the "within subsystem" correlations were performed between the metrics and both the ERROR and CODING TIME data. These results indicate that more that one metric is needed to explain the various dimensions of software complexity.

When the unit of measurement was an entire subsystem rather than a single component, the Spearman coefficients increased dramatically. Initially all twenty-one subsystems available in this research were analyzed. For this analysis, McClure's INVOKE measure consistently reported the highest coefficients with the time based data. As a rule, the correlations improved when only those subsystems with the more reliable data were analyzed. However, since the number of data points is relatively small (N=9), it is prudent not to infer too much from the high coefficients generated from the subsystem level analysis.

The high correlations generated by the subsystem level analysis may, at best, suggest that the metrics are better able to identify trends in the data when individual components are logically grouped. It may be that individual metrics are not able to detect subtle differences existing between individual components. This is especially true when using a conservative non-parametric statistic to judge the "goodness" of a particular metric. The next chapter investigates the notion of "grouping" components together in order to more clearly observe potential trends existing between the metrics and the developmental data.

Chapter VI

GROUP LEVEL ANALYSIS

The purpose of this chapter is to report on experiments which identified and compared groups of components. This group analysis is in contrast to the component by component correlations used in the previous chapter and in most prior software metric studies. The motivation for grouping components together is to identify overall trends in the data which may not be evident from simple linear or rank correlation techniques. Recall that the results of the previous chapter, and Basili's careful and extensive analysis[Basi83] on similar data both report correlation coefficients that were disappointingly low. Conclusions drawn from these studies might indicate that software metrics have little or no value. However, past research [Henr81a] which grouped components, identified a strong correspondence between information flow based groups and their error properties. Furthermore, the discussion and the results which follow generally indicate that positive trends do in fact exist between some software metrics and the developmental data.

There are three reasons for studying the behavior of a group of components. First, measurement of component groups does not require the metrics to differentiate between compo-

nents having approximately the same complexity value. Small differences in complexity values and developmental data readings between components should not mask out any overall trends in the data. This masking is possible when using the non-parametric Spearman rank correlation technique as a measure of metric validation. For example, consider the hypothetical data given in table 26 for components A through J. Notice, that the rankings imposed on these ten components by the CODING TIME data and by the information flow metric, INFOFLOW, are quite different yielding a rank coefficient of -0.03. From this result, one may conclude that CODING TIME and the INFOFLOW metric are highly unrelated. However, a group analysis would determine that a strong relationship does exist between these two measures. The two orders of magnitude difference between the reported CODING TIME for components A through H and both component I and component J is reflected in their large difference in INFOFLOW values. The inability of the non-parametric Spearman Rank method to identify some of the relationships between the metrics and the data occurs because it does not consider the distances between the various components. However, parametric techniques which account for the distances between data points, such as Pearson's correlation method also offered little insight since the distribution of the data was typically asymetrical.

TABLE 26

Hypothetical Data

COMPONENT	CODING TIME	INFOFLOW
A	0.50	9.3 x 10**2
B	0.73	8.5 x 10**2
C	1.03	8.2 x 10**2
D	1.05	7.7 x 10**2
E	1.11	7.3 x 10**2
F	1.22	5.9 x 10**2
G	1.29	4.7 x 10**2
H	1.33	4.5 x 10**2
I	126.00	3.8 x 10**6
J	133.86	3.4 x 10**6

A second reason for studying groups is to better understand those components which are considered to be outliers. It may be unrealistic, and even unnecessary or inefficient, to assume that the development staff has the resources and motivation to give each component extra consideration. By identifying those components which have extremely high values, management can better assess how to distribute the workload for an project.

A third motivation for performing a group analysis is that it is less sensitive to anomalous data points. These anomalies occur either when a component has a high metric reading with a low developmental data value or when a component has a low metric reading with a high developmental data value. The existance of these anomalies often leads to linear correlations which indicate weak associations between the metrics and the developmental data. Given the state of the art in both software metrics development and data collection systems, it is not surprising that these anomalies exist. Those components which possess a high metric reading, but have little or no reported errors, may have been more thoroughly tested before being submitted for configuration management. It is equally possible that despite employing data filtering techniques, the developmental data for a few components may not have been accurately reported. By

analyzing the components according to groups, these anomalies can be better identified. Furthermore, their existance will not inhibit the identification of any trends which may exist in the data.

This chapter contains four sections organized as follows. The next section, entitled GROUPING METHODS, describes three techniques which were used to establish component groups. More than one grouping technique was initially explored to investigate the sensitivity of the results to a particular grouping method. Section 6.2, entitled DEVELOPMENTAL DATA MEANS WITHIN GROUPS, contains a subset of the results which indicated that group level analysis show trends in the data. In particular, mean values for ERRORS, WEIGHTED CHANGES and CODING TIME were generally higher for groups containing components of higher complexity. The third section of this chapter, DEVELOPMENTAL DATA GROUPS CROSSED WITH METRIC GROUPS, presents additional evidence that some software metrics are related to both count based and time based measures. In this third section, data from a collection of two-way crosstabulation tables is displayed. The fourth and last section of the chapter provides a brief summary of the major points in this chapter.

6.1 GROUPING METHODS

The initial group analysis utilized three grouping techniques. It was important to investigate the effect of different grouping methods to gain confidence that overall results were not sensitive to a particular grouping technique. One grouping method, referred to as the Logged Data technique, identifies six distinct groups. Two of the groups consist of those components with values lying within one standard deviation from the mean (plus or minus). A second pair of groups contain components with values lying within two standard deviations of the mean(plus or minus). Finally, those components lying more than two standard deviations from the mean comprise the last two groups. However, since most of the original data is asymmetric with positive skew the usefulness of univariate summary statistics, such as the mean and the standard deviation is minimal. Meaningful summary statistics can be obtained, however, for asymmetric data, by first (a) transforming the data to acheive a symmetric distribution, (b) calculating the mean m, and standard deviation s of the transformed data, and then (c) applying the inverse transform to the values: m-2s, m-s, m, m+s, m+2s to determine the group boundaries of the original data. This method of transforming the data to obtain a non-skewed distribution has been used by Crawford et

al.[Craw85] to analyze static metrics. In their original work, as is done in this research, a logarithmic transformation is applied to the basic data.

The second grouping technique, referred to as Clustering, is based on a hierarchical clustering scheme proposed by Johnson[John67]. Basically, the algorithm begins by forming one cluster for each observation in the data. Components are then grouped according to a distance function, clustering together those components which are "neighbors". Clusters are determined by considering the distances between components within a given cluster versus the distances between the clusters. Ideally, the algorithm will choose the optimal number of clusters, but this research will arbitrarily fix the number of clusters to be six. Fixing the number of clusters constant at six, facilitates any comparisons made between this grouping technique and the other two techniques.

The third grouping technique, referred to as Rank Grouping, is the simplest of the three. Once again the number of groups was held fixed at six for ease of comparison. This method defines the groups according to the ranking of the components imposed by a given metric. The set of components are first ranked in ascending order by a given metric and then placed into groups according to rank. Components whose

rank was less than one-sixth the total number of components were assigned to group 1, components whose rank was greater than one-sixth the total number of components but less than two-sixths the total number of components were assigned to group 2 and so on. Whenever two or more components had the same complexity measurement their ranks were redefined as the mean of their corresponding ranks. Since it is possible that numerous components may have the same complexity value, and in some instances this is likely(ie. REVIEW Metric) not all the groups will necessarily contain the same number of members. In the worst case, a rank order group may contain no components when this simple rule is used.

6.2 TRENDS IN DEVELOPMENTAL DATA MEANS FOR EACH METRIC GROUP

The purpose of this section is to present results of initial experiments which grouped components together. These group level experiments are classified by the following three factors:

1. Grouping Technique
2. Complexity Metric
3. Developmental Data

Since there are three distinct grouping techniques, ten different complexity metrics and three types of developmental data used in this research, a total of ninety results

were generated and analyzed. This large amount of information has been pruned and summarized in the following five tables. These tables are representative of the total ninety cases. The first three tables contain the mean number of ERRORS for each of the three grouping techniques. By comparing these three tables any difference in the grouping techniques can be seen. The last two tables contain the mean number of CODING HOURS and the mean number of WEIGHTED CHANGES for only the Logged Data grouping method. These two tables were included so as to present results which spanned across all three of the developmental data variables utilized in this experiment. The rows of each of the five tables indicate which metric was used to identify group boundaries, while the columns represent the six different groups, numbered from 1 (least complex components) to 6 (most complex components).

Table 27 contains the mean number of ERRORS for groups generated by the the Logged data grouping technique. The results presented in this table largely indicate that those groups containing components with higher complexity values have a higher mean number of ERRORS associated with them. An example of this trend can be seen by observing the LOC metric. Note the steady increase in the mean number of ERRORS across the six groups. The other two code metrics,

TABLE 27

Mean Errors Per Group for Logged Data Technique

	GROUP NUMBER					
	1	2	3	4	5	6
LOC	0.10	0.65	1.12	2.93	4.10	4.28
CYCLO	0.23	1.21	1.09	3.29	3.46	3.12
EFFORT	0.67	0.78	1.57	2.95	2.80	2.55
INFOFLOW	1.07	1.62	1.49	1.98	3.58	23.50
STABILITY	1.25	1.51	1.90	2.62	2.50	---
REVIEW	5.05	---	1.37	2.24	2.80	---
INVOKE	1.25	0.67	1.17	2.19	5.85	20.75
STAB-LOC	1.08	1.53	1.57	2.48	4.91	0.20
REVIEW-LOC	5.34	0.48	0.98	2.28	4.08	3.55
INFO-LOC	0.43	1.50	1.34	2.03	3.91	17.00

McCabe's CYCLO measure and Halstead's EFFORT measure also show a similar tendency. Of the four structure metrics, only Woodfield's REVIEW metric fails to exhibit this behavior. It is striking to note the dramatic jump in the mean number of errors between Group 5 and Group 6 for both the INFOFLOW metric(from 3.58 to 23.5) and McClure's INVOKE complexity measure(from 5.85 to 20.75). Of the three hybrid metrics, the INFO-LOC measure and the STAB-LOC measure both impose groupings on the components which relate to the average number of errors reported. The trend is consistent for the first five groups established by the STAB-LOC metric, while the sixth group exhibits a decrease in the mean number of errors. The sharp increase between group 5 and group 6 is also present for the INFO-LOC groups, jumping from 3.91 mean errors to 17.0 mean errors. The REVIEW-LOC metric fails to show any strong relationship between the mean number of errors and the reported complexity. In fact, these results give conterintuitive evidence that components which have the smallest REVIEW-LOC complexity will, on the average contains more errors that any other group. Three entries in the table contain dashes implying that the Logged Data grouping technique defined group boundaries in such a manner that these groups were empty.

Table 28 contains the mean number of ERRORS per group for each of the ten software metrics when the Clustering technique is employed.

The results in this table are for the most part, similar to those found in the previous table when the Logged Data technique was employed. Once again a large increase in the mean number of ERRORS exists between adjacent groups for the INFOFLOW metric, the INFO-LOC metric, an the INVOKE metric. For both information flow based metrics this large increase exists between groups 4 and 5 as well as groups 5 and 6. For McClure's metric, the mean number of errors for group 5 (22.00) and group 6 (13.00) are noticeably higher than the means for the four lower groups. Furthermore, the mean number of errors across clustered groups for the LOC metric, the CYCLO metric, the STABILITY metric, the REVIEW metric, and the STAB-LOC metric all exhibit behaviors which are similar to their Logged Data counterparts. However, the behavior of the mean number of errors across clustered groups was somewhat different than Logged Data formed groups for two metrics. Halstead's EFFORT metric generates a decrease in the mean number of errors between clustered group 3 and clustered group 4. Also, Woodfield's REVIEW-LOC metric formed clustered groups which does in fact agree with intuition.

TABLE 28

Mean Errors Per Group for Clustering Technique

	GROUP NUMBER					
	1	2	3	4	5	6
LOC	0.88	2.49	4.29	4.90	4.00	4.66
CYCLO	0.98	3.03	3.50	4.23	2.60	2.66
EFFORT	2.12	2.57	2.64	0.50	4.00	2.00
INFOFLOW	1.84	4.64	3.77	3.00	23.00	44.00
STABILITY	1.39	1.97	2.23	3.20	2.00	---
REVIEW	6.07	1.09	1.37	2.42	3.33	1.77
INVOKE	1.03	2.27	4.72	6.93	22.00	13.00
STAB-LOC	1.44	2.65	2.76	5.00	5.91	0.25
REVIEW-LOC	1.60	2.44	3.53	4.70	4.00	5.00
INFO-LOC	1.87	4.40	4.00	2.00	17.00	44.00

The data presented in table 29 contains the mean number of ERRORS found for the six groups when the Rank grouping technique was used. Although not as pronounced, the data still indicates that components belonging to higher complexity groups will tend to have, on the average, a larger number of reported errors. It is encouraging to find that even a naive grouping technique is sufficient to expose this general trend. For example, the ERROR mean for group 1 components for each of the three code metric groups(0.52, 0.84, 0.66) fall below one error, while the corresponding means for the group 6 components (4.03, 4.01, 4.76) all exceed four errors. Furthermore, the error means for the four intermediate groups are consistently increasing. This trend is not as consistent for either the structure metrics or the hybrid metrics. Once again Woodfield's REVIEW and REVIEW-LOC metrics do not exhibit any consistent trend. However, both information flow based measures and McClure's INVOKE metric possess noticeably higher mean errors for group five and group six components than they do for component belonging to the lower four groups.

As mentioned above, the following two tables contain results obtained from forming groups using only the Logged Data Technique. Results for the Clustering and Ranking methods were similar and are not presented. Table 30, found

TABLE 29

Mean Errors Per Group for Ranked Data Technique

	GROUP NUMBER					
	1	2	3	4	5	6
LOC	0.52	0.75	1.73	2.15	3.85	4.03
CYCLO	0.84	1.08	1.25	2.48	3.24	4.01
EFFORT	0.67	0.67	1.18	2.71	3.06	4.76
INFOFLOW	1.46	1.35	1.59	1.64	2.50	4.50
STABILITY	1.45	1.41	2.43	1.68	3.13	2.95
REVIEW	5.05	---	1.37	---	2.24	---
INVOKE	1.25	0.82	1.03	1.50	2.51	5.95
STAB-LOC	1.03	1.52	1.58	2.24	2.62	4.03
REVIEW-LOC	3.09	0.55	1.25	1.94	2.21	4.03
INFO-LOC	1.35	1.04	1.87	1.67	2.27	4.88

below, contains the mean number of WEIGHTED CHANGES for each of the groups established by the Logged Data Technique.

These numbers, with the exception of a few values, generally show that some metrics do place components into groups which will have increasing weighted change means. Only Woodfield's REVIEW metric fails to indicate any relationship with WEIGHTED CHANGES. The three code metrics, together with the STAB-LOC and REVIEW-LOC each report a decrease in mean number of WEIGHTED CHANGES between group 5 and group 6. This decrease exists because these metrics positioned the single component with the highest WEIGHTED CHANGE value into group 5. McClure's INVOKE measure and both the INFOFLOW and INFO-LOC measures correctly placed this component into group 6, yielding the expected increase in the mean WEIGHTED CHANGES between group 5 and group 6.

The results given in table 31 illustrate the relationship between the mean number of CODING TIME hours and the six groups. The group were established using the Logged Data Technique. Although not everywhere consistent, once again, the pattern exists where those components belonging to higher groups require, on the average, a longer period of time to implement. Two of the three code metrics, LOC and Halstead's EFFORT, form groups which have ascending CODING TIME means. Three structure metrics, INFOFLOW, STABILTY, and

TABLE 30

Mean Weights Per Group for Logged Data Technique

	GROUP NUMBER					
	1	2	3	4	5	6
LOC	0.10	21.63	33.64	50.10	75.18	55.92
CYCLO	8.31	29.47	26.81	60.48	71.75	43.35
EFFORT	3.50	21.07	42.68	52.88	55.25	36.25
INFOFLOW	33.33	26.54	33.99	40.15	65.96	515.16
STABILITY	24.81	29.66	39.00	52.60	53.75	---
REVIEW	43.95	---	39.40	41.86	57.26	---
INVOKE	23.57	25.81	24.13	49.02	122.24	759.50
STAB-LOC	19.00	31.18	33.15	49.45	109.68	13.10
REVIEW-LOC	0.16	19.85	34.17	48.85	73.79	64.66
INFO-LOC	30.61	26.28	33.11	39.97	68.94	338.10

INVOKE complexities each exhibit the same consistent trend. It appears that Woodfield's REVIEW metric is unable to impose a grouping upon the components that will relate to CODING TIME means. Of the three hybrid metrics, both the INFO-LOC metric and the STAB-LOC metric exhibit increasing CODING TIME means across the different groups. The REVIEW-LOC metric also generates increasing CODING TIME means for groups two through six. However, this metric again attributes its least complex group of components with a relatively high average coding time (12.91).

In order to statistically support the hypothesis that the ten software metrics impose a grouping on the components which yield significant differences in the three developmental data means(ERRORS,CODING TIME,WEIGHTED CHANGES), thirty analyses of variance were performed. The results of these ANOVA's are summarized in table 32 found below. The results contained within this table indicate that twenty-five of the thirty analyses of variance rejected the null hypothesis that the group means were all equal(alpha=.05). The five anova's which failed to reject the null hypothesis are each denoted in the table with an asterisk.

Although rejection of the null hypothesis statistically indicates that not all of the population means are equal, it does not indicate which group means are statistically diffe-

TABLE 31

Mean CODING TIME Per Group for Logged Data Technique

	GROUP NUMBER					
	1	2	3	4	5	6
LOC	2.33	3.10	3.73	8.38	16.95	34.26
CYCLO	1.48	3.37	3.99	9.10	18.81	13.26
EFFORT	3.30	3.48	5.20	8.67	17.06	4.25
INFOFLOW	3.79	4.44	8.76	14.17	14.42	---
STABILITY	5.53	4.12	4.66	9.97	11.55	---
REVIEW	13.85	---	6.18	8.61	8.56	4.51
INVOKE	6.33	5.21	8.08	7.55	11.37	26.66
STAB-LOC	5.08	5.80	7.53	8.90	10.53	---
REVIEW-LOC	12.91	3.67	3.99	6.56	16.12	24.84
INFO-LOC	0.50	3.85	4.05	8.87	14.93	14.42

TABLE 32

ANOVA Results Between Metrics and Developmental Data

	ERRORS	WEIGHTED CHANGES	CODING TIME
LOC	REJECT	REJECT	REJECT
CYCLO	REJECT	REJECT	REJECT
EFFORT	REJECT	*	REJECT
INFOFLOW	REJECT	REJECT	REJECT
INVOKE	REJECT	REJECT	REJECT
REVIEW	REJECT	*	REJECT
STABILITY	*	*	REJECT
INFO-LOC	REJECT	REJECT	REJECT
REVIEW-LOC	REJECT	REJECT	REJECT
STAB-LOC	REJECT	REJECT	*

rent from each other. In order to identify those groups with statistically different means, a multiple comparisons procedure was applied to the data. In particular, Fisher's Protected Least Significant Difference (LSD) [OTT77], method was chosen since it is one of the few multiple comparison procedure's which allow for unequal group sizes. Table 33, table 34, and table 35 found below summarize the results of the Protected LSD multiple comparisons analysis used to determine which groups had significant differences in their ERROR means, their CODING TIME means and their WEIGHTED CHANGES means.

Within each table, group numbers are ordered, from left to right, by descending developmental data means. Those groups not underlined by a common line are declared to be significantly different according to the least significant criterion. For example, in table 33 McClure's INVOKE metric defines the mean number of ERRORS for Group 6 to be significantly different than the mean number of ERRORS for the other five groups. Furthermore, the ERROR mean for Group 5 was also considered different than the ERROR means for all other groups. However, the ERROR means for Group 4, Group 1, Group 3 and Group 2 were not found to be statistically different. In a very similar manner, the INFO-LOC metric declares Group 6 ERROR means to be statistically different

TABLE 33

LSD Summary for ERROR Means

```
Metric                     Group Number
------                     ------------
LOC                6     5     4     3     2    1
                  ***************   **************
                              ***************

EFFORT             4     5     6     3     2    1
                  *********************
                       ***************************

CYCLO              5     4     6     2     3    1
                 ***************   **************
                             *****************

INFOFLOW           6     5     4     2     3    1
                  ***   *********************
                              *********************

INVOKE             6     5     4     1     3    2
                  ***   ***   ********************

REVIEW             1     5     4     3
                  ***   ********
                             ***********

INFO-LOC           6     5     4     2     3    1
                  ***   ********
                             *********************

REVIEW-LOC         1     5     6     4     3    2
                  ***************   **************
                        ****************

STAB-LOC           5     4     3     2     1    6
                  ***   ********************
                              **********************
```

from the remaining five groups. Although the Group 5 ERROR mean is not statistically different from the Group 4 ERROR mean, it is declared to be different from the means of the lower three groups.

Three observations can be made regarding the data found in these three tables. First, four metrics: LOC, INVOKE, INFOFLOW and INFOFLOW-LOC form more meaningful group boundaries than do the other six metrics. These metrics will generally place those components with higher development data values into the more complex groups. This is unlike the REVIEW based metrics which typically assigns components with a high mean value to its least complex group. Second, the LOC,INVOKE,INFOFLOW and INFO-LOC metrics are not able to identify significant differences between the means of the least complex groups. Third, the strength of McClure's metric and to a great extent the two information flow metrics are their ability to identify groups of highly complex components which have mean values significantly higher than the mean values for the less complex groups.

TABLE 34

LSD Summary For CODING TIME means

LOC	6	5	4	3	2	1
	***	***	**********			
EFFORT	5	4	3	6	2	1
	********					
		**********				
CYCLO	5	6	4	3	2	1
	********	**********				
		********	**********			
INFOFLOW	6	5	4	3	2	
	**********					
			**********			
INVOKE	6	5	3	4	1	2
	***	**********				
STABILITY	5	4	1	3	2	
	********		**********			
REVIEW	1	4	5	3	6	
	***	**********				
INFO-LOC	5	6	4	3	2	1
	**********					
		**********				
REVIEW-LOC	6	5	1	4	3	2
	***	********		**********		

TABLE 35

LSD Summary for WEIGHTED CHANGE Means

```
LOC          5     6     4     3     2     1
            ******************
                              ************
                  ******************

CYCLO        5     4     6     2     3     1
                        ******************
            ******************
                  ******************

INFOFLOW     6     5     4     3     1     2
            ***   ************************

INVOKE       6     5     4     2     3     1
            ***   ********
                        ******************

INFO-LOC     6     5     4     3     1     2
            ***   ************************

STAB-LOC     5     4     3     2     1     6
            ***    ************************

REVIEW-LOC   5     6     4     3     2     1
            ******************
                        ******************
                  ******************
```

6.3 METRIC GROUPS CROSSED WITH DEVELOPMENTAL DATA GROUPS

The purpose of this section is to provide a more detailed view of the groups defined in section 6.2 by comparing groups formed by selected metrics with groups formed by the developmental data. The results presented in this section were obtained from the formation of two-way crosstabulation tables generated by the Statistical Analysis System (SAS). It is useful to present crosstabulation tables since they show combined frequency distributions for two variables. By observing the frequency distribution, the relationship between the two measures can be better identified. Furthermore, it is possible to isolate any anomalous components. For example, those components which belong to a high ERROR group, but also belong to a low INVOKE group would be considered anomalies. In addition to displaying the frequency distributions, SAS also outputs row, column and table percentages for each entry in a given crosstabulation table. These percentages will be used in this section to identify trends in the group level data.

For this analysis, it was necessary to also define group boundaries for ERRORS, WEIGHTED CHANGES, and CODING TIME. Since each of the three grouping techniques generally identified the same trend in the data, only one grouping method will be used in this analysis. The Logged Data technique,

suggested by Crawford et.al., was chosen as a representative grouping method since it is more sophisticated than a simplistic rank grouping (ie. uses the distances between components) and it attempts to compensate for the asymmetric properties inherit in the data.

The following three tables contain two-way crosstabulations between selected metric groups(LOC, INFO-LOC, INVOKE) and groups defined by the ERROR data. Tables for only these three metrics are presented since they more clearly indicate trends in the data.

Each table entry contains four numbers. From top to bottom, the first number represents the frequency count, the second number represents the table percentage, the third number represents the row percentage and the final number represents the column percentage. For example, in table 36 where LOC groups are crossed with ERROR groups, there were nine components which were classified as group 1 components for both the LOC metric groupings and the ERROR data groupings. These nine components represented 1.60% of the population, 90% of all components classified as ERROR group 1 components, and 3.85% of all components classified as LOC group 1 components.

The crosstabulation data given in table 36, table 37, and table 38 indicate some interesting relationships between the

TABLE 36

LOC Groups Crosstabulated With ERROR Groups

LOC GROUPS / FREQ / PERCENT / ROW PCT / COL PCT	ERROR GROUPS 1	2	3	4	5	6	TOTAL
1	9	1	0	0	0	0	10
	1.60	0.18	0.00	0.00	0.00	0.00	1.78
	90.00	10.00	0.00	0.00	0.00	0.00	
	3.85	0.86	0.00	0.00	0.00	0.00	
2	42	3	3	6	1	0	55
	7.49	0.53	0.53	1.07	0.18	0.00	9.80
	76.36	5.45	5.45	10.91	1.82	0.00	
	17.95	2.59	3.95	7.89	2.13	0.00	
3	120	45	21	15	9	1	211
	21.39	8.02	3.74	2.67	1.60	0.18	37.61
	56.87	21.33	9.95	7.11	4.27	0.47	
	51.28	38.79	27.63	19.74	19.15	8.33	
4	45	55	34	32	23	7	196
	8.02	9.80	6.06	5.70	4.10	1.25	34.94
	22.96	28.06	17.35	16.33	11.73	3.57	
	19.23	47.41	44.74	42.11	48.94	58.33	
5	18	11	17	20	12	4	82
	3.21	1.96	3.03	3.57	2.14	0.71	14.62
	21.95	13.41	20.73	24.39	14.63	4.88	
	7.69	9.48	22.37	26.32	25.53	33.33	
6	0	1	1	3	2	0	7
	0.00	0.18	0.18	0.53	0.36	0.00	1.25
	0.00	14.29	14.29	42.86	28.57	0.00	
	0.00	0.86	1.32	3.95	4.26	0.00	
TOTAL	234	116	76	76	47	12	561
	41.71	20.68	13.55	13.55	8.38	2.14	100.00

TABLE 37

INFO-LOC Groups Crosstabulated With ERROR Groups

INFO-LOC GROUP / FREQ / PERCENT / ROW PCT / COL PCT	ERROR GROUP 1	2	3	4	5	6	TOTAL
1	13	0	2	1	0	0	16
	2.32	0.00	0.36	0.18	0.00	0.00	2.85
	81.25	0.00	12.50	6.25	0.00	0.00	
	5.56	0.00	2.63	1.32	0.00	0.00	
2	44	14	4	7	8	0	77
	7.84	2.50	0.71	1.25	1.43	0.00	13.73
	57.14	18.18	5.19	9.09	10.39	0.00	
	18.80	12.07	5.26	9.21	17.02	0.00	
3	95	36	16	14	9	2	172
	16.93	6.42	2.85	2.50	1.60	0.36	30.66
	55.23	20.93	9.30	8.14	5.23	1.16	
	40.60	31.03	21.05	18.42	19.15	16.67	
4	70	49	35	26	16	3	199
	12.48	8.73	6.24	4.63	2.85	0.53	35.47
	35.18	24.62	17.59	13.07	8.04	1.51	
	29.91	42.24	46.05	34.21	34.04	25.00	
5	12	17	18	27	12	5	91
	2.14	3.03	3.21	4.81	2.14	0.89	16.22
	13.19	18.68	19.78	29.67	13.19	5.49	
	5.13	14.66	23.68	35.53	25.53	41.67	
6	0	0	1	1	2	2	6
	0.00	0.00	0.18	0.18	0.36	0.36	1.07
	0.00	0.00	16.67	16.67	33.33	33.33	
	0.00	0.00	1.32	1.32	4.26	16.67	
TOTAL	234	116	76	76	47	12	561
	41.71	20.68	13.55	13.55	8.38	2.14	100.00

TABLE 38

INVOKE Groups Crosstabulated With ERROR Groups

INVOKE GROUPS / FREQ / PERCENT / ROW PCT / COL PCT	ERROR GROUPS 1	2	3	4	5	6	TOTAL
1	54	16	7	11	6	0	94
	9.63	2.85	1.25	1.96	1.07	0.00	16.76
	57.45	17.02	7.45	11.70	6.38	0.00	
	23.08	13.79	9.21	14.47	12.77	0.00	
2	40	12	5	3	1	0	61
	7.13	2.14	0.89	0.53	0.18	0.00	10.87
	65.57	19.67	8.20	4.92	1.64	0.00	
	17.09	10.34	6.58	3.95	2.13	0.00	
3	65	43	15	10	4	1	138
	11.59	7.66	2.67	1.78	0.71	0.18	24.60
	47.10	31.16	10.87	7.25	2.90	0.72	
	27.78	37.07	19.74	13.16	8.51	8.33	
4	65	38	37	38	21	1	200
	11.59	6.77	6.60	6.77	3.74	0.18	35.65
	32.50	19.00	18.50	19.00	10.50	0.50	
	27.78	32.76	48.68	50.00	44.68	8.33	
5	10	7	12	13	14	8	64
	1.78	1.25	2.14	2.32	2.50	1.43	11.41
	15.63	10.94	18.75	20.31	21.88	12.50	
	4.27	6.03	15.79	17.11	29.79	66.67	
6	0	0	0	1	1	2	4
	0.00	0.00	0.00	0.18	0.18	0.36	0.71
	0.00	0.00	0.00	25.00	25.00	50.00	
	0.00	0.00	0.00	1.32	2.13	16.67	
TOTAL	234	116	76	76	47	12	561
	41.71	20.68	13.55	13.55	8.38	2.14	100.00

metric based groups and the ERROR based groups. First, notice that the row percentage for ERROR group 1 components typically decreases as the metric group number increases. The ERROR group 1 components are precisely those components which have been found to be error free. These percentages indicate that a larger fraction of components have errors for the higher metric groups than they do for the lower metric groups. For example, 90% of the components found in LOC Group 1 were not reported to have any errors. However, only 76% of the components in LOC Group 2 were found to be errorless, 56% percent for LOC Group 3 and so on. Interestingly, all components belonging to the highest LOC group did contain at least one error. The results presented in table 39 show the percentage of components within each group which were found to contain no errors for all ten software metrics.

This trend is also everywhere consistent for McCabe's CYCLO metric ranging from 76%(Group 1) down to 0%(Group 6). The third code metric, Halstead's EFFORT generally exhibits this same property, but does contain an anomalous jump between the Group 5 and Group 6 boundaries. Furthermore, of the four structure metrics, only Woodfield's Review complexity fails to show any pattern. The strongest relationship between the various structure metric groups and fraction of

TABLE 39

Percentage of Errorless Procedures Across Groups

	GROUP NUMBER					
	1	2	3	4	5	6
LOC	90%	76%	56%	22%	21%	0%
CYCLO	76%	63%	53%	26%	25%	0%
EFFORT	80%	64%	54%	29%	18%	33%
INFOFLOW	73%	51%	53%	36%	17%	0%
STABILITY	60%	51%	40%	38%	31%	---
REVIEW	14%	---	53%	36%	17%	0%
INVOKE	57%	65%	47%	32%	15%	0%
STAB-LOC	56%	49%	40%	35%	26%	80%
REVIEW-LOC	20%	80%	61%	22%	22%	11%
INFO-LOC	81%	57%	55%	35%	13%	0%

errorless components exist with the INFOFLOW groups. These percentages are steadily decreasing, ranging from 73%(Group 1) to 53%(Group 3) to 0%(Group 6). Of the three hybrid metrics, only the INFO-LOC metric defines groups with consistently decreasing percentages. The REVIEW-LOC metric exhibits this desirable trend for Group 2 through Group 6, however it does report only 20% of the components to be errorless for its least complex group of components.

A second relationship between the metric based groups and the ERROR based groups, can be inferred from the crosstabulation data. In the three crosstabulations presented, the upper right portion of each table is relatively sparse. This indicates that those components which were highly error prone, were typically the most complex components of the system. In particular, notice that eleven of the twelve components with the highest number of reported errors were located in the higher three groups for both LOC metric and the INVOKE metric. The information flow based metric, INFO-LOC also performed quite well, indentifying ten of the twelve most error prone components. Table 40, found below summarizes the ability of each of the ten metrics to classify the error prone components as more complex. The entries found in this table represents the percentage of error prone components (Error groups 4,5 & 6) which were classified as the

most complicated (Error groups 4,5 & 6). The percentages in table 40 indicate that the code metrics identify a higher percentage of the error prone components than either of the other two classes of metrics. Of the remaining seven metrics, McClure's INVOKE metric and both the information based metrics identify approximately 70% of the most error prone components as the more complicated.

A third trend in the crosstabulation data is graphically displayed in the form of "city block" charts. A few of these charts are illustrated in figure 7 and figure 8 found below. Each of the following city block charts contain "buildings" representing the row percentages found in associated two-way crosstabulation tables. Observe the behavior of percentage distributions across metric groups. For example, consider the block chart found in figure 7 (a). Notice that 90% of all LOC Group 1 components also reside in ERROR Group 1. However 10% of the LOC Group 1 components have shifted into ERROR group 2. Notice further that only 76.36% of all LOC Group 2 components reside in ERROR Group 1. The remaining 23.64% of the components have "seeped" into ERROR groups 2 through 5. More and more components seep out of ERROR Group 1 as the LOC group number increases. This trend was previously highlighted in table 36. However, this city block chart graphically emphasizes that as the LOC

TABLE 40

Percentage of Error Prone Components Identified as Complex

Metric	Percentage
LOC	76%
CYCLO	74%
EFFORT	77%
INVOKE	73%
INFOFLOW	69%
STABILITY	61%
REVIEW	47%
INFO-LOC	70%
STAB-LOC	63%
REVIEW-LOC	59%

group number increases, more and more components generally seep into the higher ERROR groups. Ultimately, for LOC Group 6, all the components have seeped into the higher ERROR Groups with 71.33% of the components residing in either ERROR Group 4 or ERROR Group 5. It is somewhat surprising that none of the LOC Group 6 components seeped into the highest ERROR group. Notice that this was not the case for the other three city block charts found in figure 7. These three charts indicate that McClure's INVOKE measure and both of Henry's Information Flow measures also exhibit the seeping effect. The rate in which components seep into the higher ERROR groups appears to be somewhat gradual for the first three or four metric based groups. This is illustrated by the line which is superimposed on each of the four city block charts. This line, connecting the tallest buildings within each of the metric based groups, roughly indicates the rate of seepage across the six metric based groups. Notice that the line does not jump sharply until the transition from Group 4 to Group 5 is made. The gradual seepage of components for the first four groups, followed by larger amounts of seepage for the last two groups is consistent with the previous analysis using the LSD multiple comparison test. Recall that Fisher's Protected LSD test indicated that the ERROR means of the lower complexity groups

were not found to be significantly different but did declare significant differences to exist for the ERROR means of the higher complexity groups.

The two city block charts comprising figure 8 graph the row percentages found in the crosstabulations between groups based on CODING TIME and groups based on both the LOC metric and the INFO-LOC metric. Both of the block charts found in figure 8 also exhibit a shift in the distribution of percentages across the metric based groups. Each of these diagrams illustrate the fact that components belonging to higher complexity based groups will generally take longer to implement. The trend is particularly noticeably in figure 8 (a), where the tallest buildings all lie on the main diagonal.

Finally, the information contained in the two-way crosstabulations between the metric based groups and the WEIGHTED CHANGES based groups has been summarized in the following table. The data in table 41 captures the ability of the metrics to identify components which will be highly affected by changes in the system. To obtain the percentages given in this table, the components were partitioned into two classes: severely impacted components and weakly impacted components. Severely impacted components are precisely those routines which possessed the highest WEIGHTED

Figure 7: Row Percentages Between Metrics and Error Groups

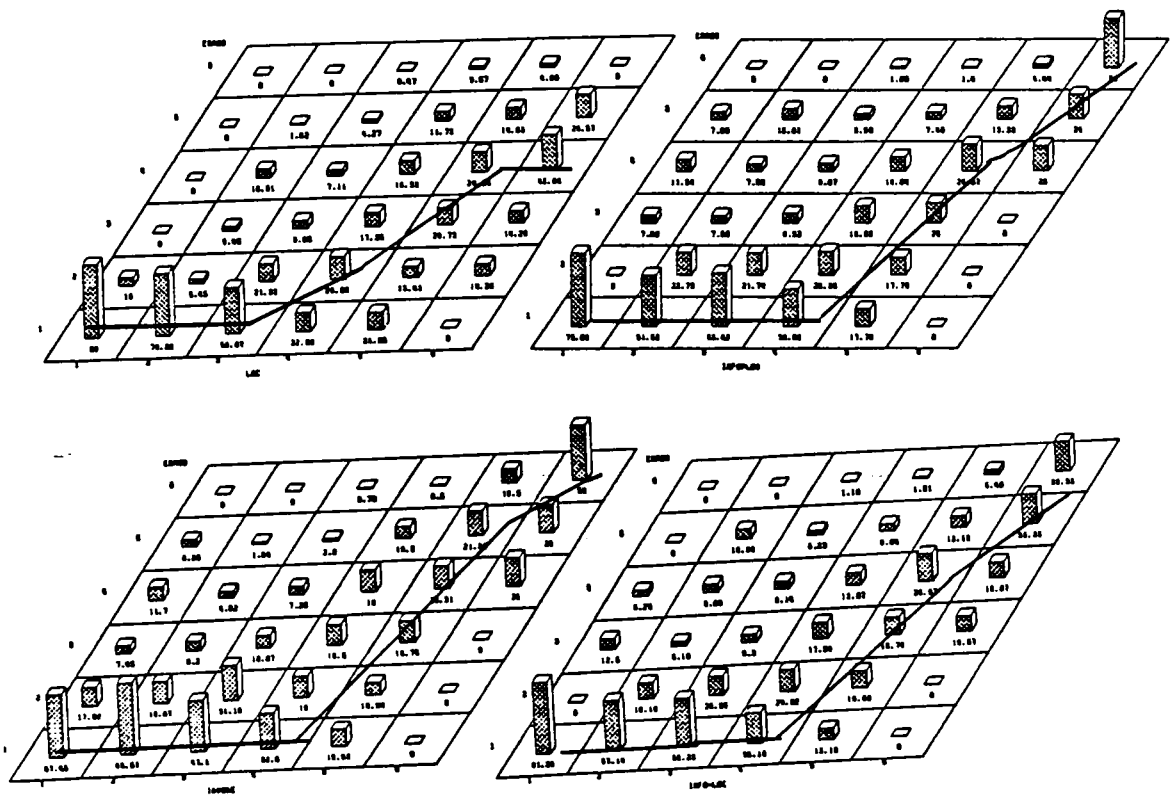
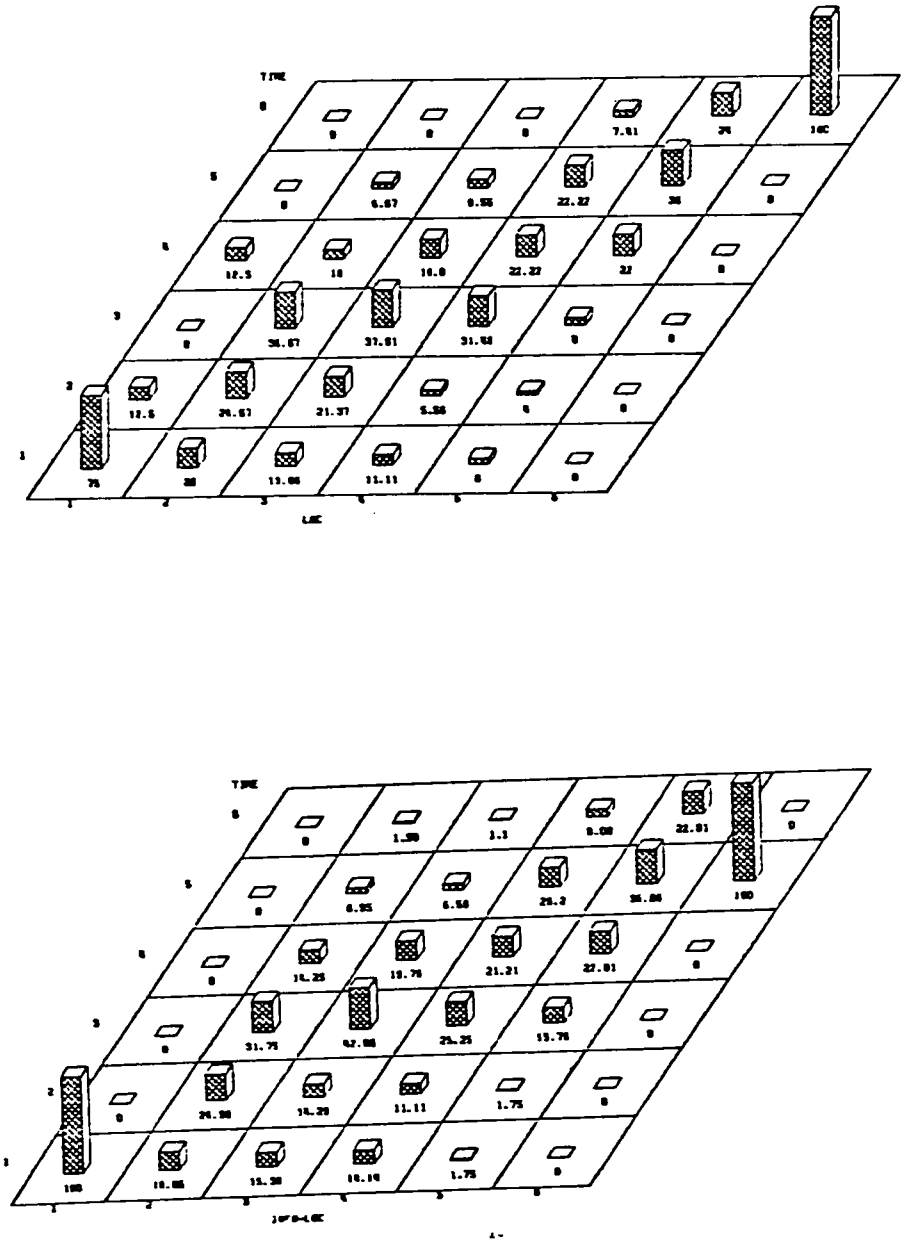


Figure 8: Row Percentages Between Metrics and Time Groups



CHANGE values. In particular, a component possessed a "high" WEIGHTED CHANGE value if the Logged Data grouping method positioned the component into either Group 4, Group 5, or Group 6. The entries in table 10 indicate the percentage of components within each metric based group, which were severely impacted by changes in the system. For example,72% of the components in LOC group 5 were also identified to be severely impacted components, while 100% of the components in INVOKE group 6 were severely impacted components. The percentages given in the last two columns of table 10 generally indicate that components placed into the higher metric groups are likely to be severely impacted by system changes. Only the group 6 percentage for the STAB-LOC metric falls below 56%. Furthermore, the three code metrics indicate that components placed in the lower metric groups are not as likely to be severly impacted. Unfortunately, the structure metrics do not show a similar trend. Notice that for group 1 components, each of the structure metrics still report rather high percentages of severely impacted components, ranging from 36 to 70 percent.

TABLE 39

Fraction of Severely Impacted Components

	GROUP NUMBER					
	1	2	3	4	5	6
LOC	0%	35%	43%	54%	72%	100%
CYCLO	15%	25%	43%	63%	66%	100%
EFFORT	5%	39%	48%	50%	81%	62%
INFOFLOW	58%	33%	41%	60%	77%	100%
STABILITY	40%	35%	47%	55%	63%	---
REVIEW	70%	---	44%	53%	56%	0%
INVOKE	36%	31%	40%	60%	82%	100%
STAB-LOC	29%	50%	49%	58%	65%	20%
REVIEW-LOC	0%	29%	44%	51%	70%	100%
INFOFLOW LOC	55%	34%	42%	45%	85%	100%

6.4 SUMMARY AND CONCLUSIONS

This chapter has reported on experiments which identified and compared groups of components. Three different grouping techniques: Logged Data, Clustering and Rank, were employed, each of which generally produced similar results. It was found that many of the metrics were able to partition the components into groups which possessed significantly different means for ERRORS, WEIGHTED CHANGES, and CODING TIME. Furthermore, the two information flow metrics and McClure's invocation complexity were particularly able to separate out those components which possessed extremely high developmental data readings.

This chapter also presented results from two-way crosstabulations between the component groups formed by the metrics and component groups formed by the observed measures of complexity. One result of these crosstabulations indicated the relationship between metric based groups and their percentage of errorless components. It was found that some metrics: LOC, CYCLO, INFOFLOW, STABILITY and INFO-LOC were able to define increasingly more complex groups which had a decreasing percentage of errorless components. Furthermore, the six metrics: LOC, CYCLO, EFFORT, INFOFLOW, INVOKE, AND INFO-LOC were able to identify approximately seventy percent of the most error prone components.

The crosstabulation data was also graphically displayed in the form of "city block" charts which revealed percentage distributions of the data across selected metric groups. These distributions once again illustrated that as more complex groups are considered, a larger fraction of the population will shift into the higher ERROR or CODING TIME groups.

Chapter VII

SUMMARY AND FUTURE WORK

The main goal of this research was to investigate the performance of a variety of structure, code and hybrid metrics which have been proposed in the literature on the same large scale systems. This was achieved by applying four structure metrics, three code metrics and three hybrid metrics to software systems constructed and monitored at the Nasa/Goddard Space Flight Center. As such, inferences derived from the experimentation described in the previous chapters can only apply to software developed in the Nasa/Goddard environment. However, the work is significant because it contributes to an extremely small pool of results eminating from metric experiments on large scale systems.

From the data presented in chapter three, it was concluded that the three code metrics: LOC, CYCLO and EFFORT were all identifying similar properties of software complexity. This conclusion is in accord with past research which measured software from a similar enviroment[Basi83] and past research which measured software from a different environment[Henr81][Redd84]. Furthermore, it was concluded that the structure metrics as a class were different measures of complexity. This augments a similar result from a pilot stu-

dy performed by Henry and Kafura [Henr81b] which reported that the information flow metric was orthogonal to typical code metrics. In addition, it was determined that the structure metrics tend to be orthogonal to each other.

In chapter five the complexity values of each component, as determined by the ten software metrics were correlated with various observed values of component complexity such as ERRORS and CODING TIME. From the results contained within this chapter, it was concluded that no single metric is able to explain the behavior of a single observed value of complexity. This result was particularly true when the correlations were performed on components spanning three different software systems. Only when "within project" and "within subsystem" factors were considered did a component's metric value relate more strongly to its observed data value. The improvement in these correlations was attributed to differences between individual programmers since in the Nasa/Goddard environment subsystems are typically implemented by one or two different programmers. It was also found that no single metric consistently reported the highest coefficients between its complexity value and a given observed data value. This is further evidence that all the metrics are not measuring the same dimensions of software complexity.

From the Group Level analysis described in chapter six, relationships between some of the metrics and the developmental data that was obscured by using linear correlation techniques became more apparent. In this chapter components were partitioned into six groups of increasing metric complexity and the behavior of the developmental data across groups was observed. It was concluded that the strength of the two information flow metrics and McClure's invocation measure was their ability to place components into highly complex groups (Groups 5 and 6) which possessed noticeably higher developmental data readings. However, it was also observed that the lower four groups defined by these three metrics typically possessed similar ERROR means, CODING TIME means and WEIGHTED CHANGE means. This suggests that these metrics do not do a very good job of separating out the "easy" components from themselves, but are better able to distinguish the "easy" components from the most "difficult" components.

In addition, the crosstabulation information presented in chapter six also led to the conclusion that for most metrics, the percentage of errorless procedures is higher for groups containing the simpler components than it is for groups containing the more complex components. This was especially valid for groups defined by the three code me-

trics as well as groups defined by the two information flow metrics. Furthermore, the more complex groups, as defined by most of the metrics, were largely composed of components which were severly impacted by changes in the system. Only the REVIEW metric and the STAB-LOC metrics failed to exhibit this property.

As is the case with most works of this ilk, a number of additional avenues for future investigation are possible. One consideration for further research would be to study the performance of metrics taken in combination. It was determined in chapter five that no single metric fully explained the behavior of a single observed measure of complexity. Not only may it be important to consider the metrics in combination, but it may be equally important to consider the observed measures of complexity in combination. Work in this area has already been initiated by Kafura and Canning[Kafu85]. Another area of future study would be to explore the relationship of the structure and code metrics to data extracted from the maintenance phase of the software life cycle. One limitation of this research was that maintenance data was not available. A third possibility for additional work, would be to conduct experiments which more thoroughly explored various grouping policies. The grouping policies discussed in chapter six were sufficient to indi-

cate trends in the data, but research that indicated how to better identify meaningful groups would be useful.

REFERENCES

[Akiy71]Akiyama,F., "An Example of Software System Debugging," Proceedings IFIP Congress, pp. 353-358, 1971.

[Basi81]Basili,V. "Evaluating Software Development Characteristics: Assessment of Software Measures in the Software Engineering Laboratory," Proceedings of the Sixth Annual Software Engineering Workshop, Dec. 2, 1981.

[Basi83]Basili,V.,Selby R.,Phillips T., "Metric Analysis and Data Validation Across Fortran Projects" IEEE Transactions on Software Engineering, Vol. SE-9,No. 6. November 1983

[Bela76]Belady,L.A., Lehman,M.M., "A Model of Large Program Development," IBM Systems Journal, No. 3, 1976, pp. 225-252.

[Bela80]Belady,L.A., "Experiments with Modifiability," Life-Cycle Management, Infotech State of the Art Report, (series 8, number 7) 1980.

[Beny79]Benyon-Tinker,G., Complexity Measures in an Evolving Large System, Workshop on Quantitative Software Models, October 9-11, 1979

[Berl80]Berlinger,ELi, "An Information Theory based Complexity Measure," AFIPS Conference Proceedings , 1980 National Computer Conference , May 19 .

[Boeh79]Boehm,B., "Software Engineering: R & D Trends and Defense Needs," Research directions in Software Technology Peter Wegner,editor; The MIT Press.

[Brow81]Browne,J.C.; Shaw,Mary, "Toward a Scientific Basis for Software Evaluation," Software Metrics, Perlis, Sayward, and Shaw, editors, The MIT Press, 1981.

[Chan73]Chanon,R., "On a Measure of Program Structure," Technical Report, Department of Computer Science, Carnegie-Mellon University, 1973.

[Chap79]Chapin, Ned, "A Measure of Software Complexity," Proceedings AFIPS 1979 National Computer Conference , pp 995-1002,AFIPS, June 1979

[Chen78]Chen,Edward, "Program Complexity and Programmer Productivity," IEEE Transactions on Software Engineerinf, May 1978,pp 187-194.

[Cobb78]Cobb,G.W., "A Measurement of Structure for Unstructured Programming Languages," Proceedings of the Software Quality and Assurance Workshop, Nov. 1978, pp140-147.

[Craw85]Crawford,S.,McIntosh A., and Pregibon D., "An Analysis of Static Metrics and Faults in C Software" The Journal of Systems and Software, Vol. 5 #1, Feb. 1985

[Curt79a]Curtis,W., Sheppard,S.,Milliman,P.,Borst,M.,Love,T. "Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics," IEEE Transactions on Software Engineering, Vol. SE-5,No.2, March 1979.

[Curt79b]Curtis,W. , Sheppard,S.B. , and Milliman,P.M. "Third Time Charm : Stronger Prediction of Programmer Performance by Software Complexity Metrics , " Fourth International Conference on Software Engineering September 1979 , Munich , Germany , 356-360 .

[Curt80]Curtis,Bill "Measurement and Experimentation in Software Engineering" Proceedings of the IEEE, Vol.68, No. 9, Sept. 1980.

[Deni81]Denicoff,Marvin; Grafton,Robert, "Software Metrics: A Research Initiative," Software Metrics, Perlis, Sayward, and Shaw, editors, The MIT Press, 1981

[Dijk79]Dijkstra,E., "The Humble Programmer," Classics in Software Engineering Yourdon,E.,editor;,1979,pp113-128.

[Duns80]Dunsmore,H., Gannon J., "Analysis of the Effects of Programming Factors on Programming Effort" The Journal of Systems and Software 1, 1980

[Elsh76]Elshoff,J.L., "Measuring Commercial PL/1 programs using Halstead's Criteria," ACM Sigplan Notices (May 1976).

[Fitz78]Fitzimmons,A. and Love,T., "A Review and Evaluation of Software Science," ACM Computing Surveys, 10, 1 (March 1978), 3-18.

[Fox82]Fox,Joseph M., Software and Its Development, Prentice Hall, 233-253.

[Funa76]Funami,Y., and Halstead,M.H. , "A Software Physics Analysis of Akiyama's Debugging Data," Proceedings of the Symposium on Computer Software Engineering , April 1976 , New York , NY , 133-138 .

[Gord79]Gordon,R.D. "Measuring Inprovements in Program Clarity," IEEE Transactions on Software Engineering, SE-5,2 (March 1979) ,79-90.

[Hals72]Halstead,M.H., "Natural Laws Controlling Algorithm Structure," Sigplan Notices, Vol. 7, Num 2, Feb. 1972.

[Hals77]Halstead,M.H. Elements of Software Science , Elsevier North-Holland, Inc. , New York , NY , 1977.

[Hame81]Hamer, Peter, Frewin, Gillian, "M.H. Halstead's Software Science - A Critical Examination," ITT Technical Report No. STL 1341, July 1981.

[Hane72]Haney, Frederick, "Module Connection Analysis - A Tool for Scheduling Software Debugging Activities,"

[Haye80]Hayes,Mary, An Empirical Study of Measures of Psychological Complexity in Software, Masters Thesis, Univ. of California at Irvine, 1980.

[Henr79]Henry,Sallie, Information Flow Metrics for the Evaluation of Operating Systems' structure Ph.d Dissertation,Iowa State University, 1979.

[Henr81a]Henry,Sallie and D. Kafura, "Software Structure Metrics Based on Information Flow", IEEE Transactions on Software Engineering, Vol. SE-7, No. 5, pp. 510-518, September, 1981.

[Henr81b]Henry, Sallie, D. Kafura, and K. Harris, "On the Relationships Among Three Software Metrics", Performance Evaluation Review, Vol. 10, No. 1, pp. 81-88 Spring 1981.

[Henr84]Henry, Sallie and D. Kafura, "The Evaluation of Software Systems' Structure Using Quantitative Software Metrics", Software: Practice and Experience (to appear)

[Jack75] Principles of Program Design, Academic Press, New York 1975

[Kafu81a]Kafura,D. , Henry,S. , "Software Quality Metrics Based on Interconnectivity," The Journal of Systems and Software , Vol. 2 , pp. 121-131, 1981.

[Kafu81b]Kafura,D. Henry,S., "A Viewpoint on Software Quality Metrics: Criteria, Use, and Interpretation, Proceedings of Sigsoft Sponsored Software Engineering Symposium, June 1981.

[Kafu84]Kafura D., Canning J., and Reddy G., "The Independence of Software Metrics Taken at Different Life-Cyle Stages" Proceedings: Ninth Annual Software Engineering Works Goddard Space Flight Center, Nov. 28, 1984

[Kafu85]Kafura D.,Canning J., "A Validation of Software Metrics Using Many Metrics and Many Resources" Proceedings of the International Conference of Software Engineering August 1985

[Lehm80]Lehman,M.M., "Programs, Life-cycles, and Laws of Software Evolution," Proceedings of the IEEE, Vol. 68, No. 9, Sept. 1980.

[Litt78]Littlewood,Bev., "Validation of a Software Reliability Model," 2nd Software Life Cycle Management Workshop , August 1978, pp 146-152.

[McCa76]McCabe,T,J, "A Complexity Measure , " IEEE Transactions on Software Engineering , SE-2 , 4 (December 1976) , 308-320.

[McCl78]McClure, C. "A Model for Program Complexity Analysis," Proceedings Third International Conference on Software Engineering , Atlanta, Ga. May 1978 , 149-157.

[McGa80]McGarry,F., "The Software Engineering Laboratory: An Approach to Measuring Software Techonlogy," Proceedings from the Fifth Annual Software Engineering Workshop, Nov. 24, 1980.

[McTa81]McTap,John, "The Complexity of an Individual Program," AFIPS Conference Proceedings, 1981 National Computer Conference, May 1981.

[Mill72]Mills,H.D., "Mathematical Foundations for Structured Programming," Federal System Division, IBM Corp., Gaithersburg, MD., FSC 72-6012, 1972.

[Musa80]Musa,J.D. "Software Reliability Measurement," The Journal of Systems and Software , vol. 1, no. 3,1980, pp. 223-241.

[Myers75]Myers,G.J., Reliable Software Through Composite Design, Petrocelli/Charter,publisher 1975.

[Myer77]Myers,G.J. "An Extension to Cyclomatic Measure of Program Complexity," ACM Sigplan Notices , vol. 12 , no 10, October 1977,pp. 61-64.

[Nasa82] Guide to Data Collection Software Engineering Laboratory Series, SEL-81-101 August 1982

[Otte79]Ottenstein,L.M. "Quantitative Estimates of Debugging Requirements," IEEE Transactions on Software Engineering , SE-5, 5 (September 1979) , 504-514.

[Otte81]Ottenstein, Linda, "Predicting Software Developement Errors Using Software Science Parameters," Performance Evaluation Review: ACM Workshop/Symposium on Measurenent and Evaluation of Software Quality, March 25-27,1981.

[Perl81a]Perlis,Sayward,Shaw,editors; Software Metrics pp. 1-4.

[Perl81b]Perlis,Sayward,Shaw,editors; Software Metrics Annotated Bibliography page 297.

[Redd84]Reddy G., Analysis of a DataBase Management System Using Software Metrics, M.S. Thesis, Computer Science Department, Virginia Polytechnic Institute and State University, June 1984.

[Robe80]Roberts,Joyce, Design Structure Quality Metrics Master Thesis, University of California,Irvine, 1980.

[Ross77]Ross,Douglas T., "Structured Analysis (SA): A Language for Communicating Ideas," IEEE Transactions on Software Engineering, Jan. 1977.

[Schn81]Schneider,G.M., Sedlmeyer,R.L. , Kearney,J., "On the Complexity of Measuring Software Complexity," AFIPS Conference Proceedings 1981

[Schn79]Schneidewind,N.F., "Software Metrics for Aiding Program Development and Debugging," Proceedings AFIPS 1979 National Computer Conference, June 1979, pp. 989-994.

[Schu77]Schutzer, "On the Specification and Quantification of Software Performance Objectives," Proceedings 1977 Annual ACM Conference, pp. 181

[Silv79]Silver,Aaron, "A Computer Analysis Tool for Structural Decomposition Using Entropy Metrics," Proceedings AFIPS 1979 National Computer Conference, pp. 83-88,1979.

[Sime73]Sime M.,Green T., and Guest D., "Psychological Evaluation of Two Conditional Constructions Used in Computer Languages," International Journal of Man-Machine Studies 5(1):105-113, 1973

[Soon77]Soong, N.L., "A Program Stability Measure," Proceedings 1977 Annual ACM Conference, pp. 163-173.

[Stev74]Stevens,W.P.,Myers,G.J.,Constantine,L.L., "Structured Design," IBM Systems Journal, Vol IV., No. 2 pp. 115-139.

[Stro56]Stroud,J.M., "The Fine Structure of Psychological Time," Information Theory in Psychology, The Free Press, Chicago,Ill.,1956.

[Szul81]Szulewski,Paul,Whitworth,Mark,Buchan,P.,Dewolf,J., "The Measurement of Software Science Parameters in Software Designs," ACM Workshop/Symposium on Measurement and Evaluation of Software Quality, March 25-27,1981.

[Troy81]Troy,D.A.,Zweben,S.H., "Measuring the Quality of Structured Designs," Proceedings Sigsoft Sponsored Software Engineering Symposium, June 1981.

[Warn74]Warnier, Jean-Dominique, Logical Construction of Programs, H.E. Stenfert Kroese BV, Publisher 1974.

[Wood79]Woodfield,S.N., "An Experiment on Unit Increase in Problem Complexity," IEEE Transactions on Software Engineering . SE-5, 2 (March 1979) , 76-79 .

[Wood80]Woodfield,S.N., Enhanced Effort Estimation by Extending Basic Programming Models to Include Modularity Factors, Ph. D. Thesis, Purdue University, Computer Science Dept., 1980.

[Yau80]Yau S., Collofello J., "Some Stability Measures for Software Maintenance," IEEE Transactions on Software Engineering, Vol. SE-6,No.6,Nov.1980.

[Yin78]Yin,B.H., Winchester,J.W., "The Establishment and Use of Measures to Evaluate the Quality of Software Designs," Sigsoft Software Engineering Notes, Vol 3, No. 5, 1978, pp. 45-52.

[Zoln81]Zolnowski,J.,Simmons,D., "Taking the Measure of Program Complexity," AFIPS Conference Proceedings, 1981 National Computer Conference, May 1981.

The vita has been removed from the scanned document