An Empirical Study of Software Metrics

H. F. LI, MEMBER, IEEE, AND W. K. CHEUNG

Abstract—**Software metrics are computed for the purpose of evaluating certain characteristics of the software developed. A Fortran static source code analyzer, FORTRANAL, was developed to study 31 metrics, including a new hybrid metric introduced in this paper, and applied to a database of 255 programs, all of which were student assignments. Comparisons among these metrics are performed. Their cross-correlation confirms the internal consistency of some of these metrics which belong to the same class. To remedy the incompleteness of most of these metrics, the proposed metric incorporates context sensitivity to structural attributes extracted from a flow graph. It is also concluded that many volume metrics have similar performance while some control metrics surprisingly correlate well with typical volume metrics in the test samples used. A flexible class of hybrid metric can incorporate both volume and control attributes in assessing software complexity.**

Index Terms—**Control complexity, cross correlation, empirical study, Halstead's software science, hybrid metrics, software metric.**

I. Introduction

As software scientists attempt to understand software processes and products, it is natural for them to characterize and measure those aspects of programs that seem to affect cost. Software maintainability [1] is the degree to which characteristics that impede maintenance are present. It is driven primarily by software complexity [1], a measure of how difficult a program is to comprehend and work with. Their relationship is roughly depicted in Fig. 1.

The inherent complexity of a task may differ greatly from the apparent complexity of the program developed for the task [1]. Hence, software complexity should be measured by objective, reliable, valid, and convenient metrics but not by intuition. Such metrics may be useful in 1) preparing quality specifications or contractual obligation, 2) checking compliance of software specification, and 3) making design tradeoffs between maintenance and development costs. To study the validity and relationship of such metrics, some experimentation involving developed program codes is necessary.

There are different scales of measurement that could be adopted: 1) ordinal, 2) interval, 3) ratio [2]. In the ordinal scale, the complexities of different programs are to be ranked to provide a rather crude comparison. In the interval scale, the difference in complexity of two programs is expressed in units, for example, program B is 6 units more complex than program A. In the ratio scale, the ratio of the complexities of two programs is determined. The ratio scale is flexible but impractical. The interval scale requires some definition of units. Thus, the ordinal scale seems more convenient and practicable.

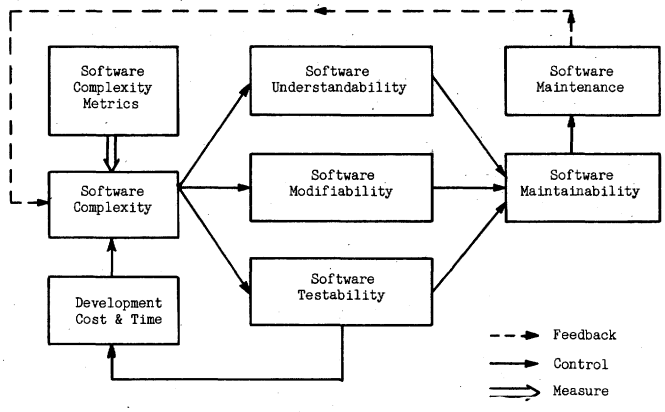


Fig. 1. Importance of software complexity.

Many complexity metrics have been developed and they can be classified into two basic types: 1) static and 2) dynamic. In the former case, measurement of the product is done by static analysis of the source code, while in the latter case, it is collected at run time and may vary from one execution to the other [16], [17]. In this paper, the attention will be concentrated on static measures which can in turn be divided into three types:

1) Volume: measures the size of a product.

2) Data Organization: measures the usage and visibility of data as well as their interactions.

3) Control Organization: measures the comprehensibility of control structures.

Fig. 2 illustrates the classification using some common measures of interest [3]–[10]. Most of these measures have been used in some way but do not gain full acceptance partly because it is not certain what aspects of the software life cycle the metrics describe and partly because of the difficulty in parameterization. Empirical evaluation of these measures was also reported by several researchers [11]–[14].

This paper is concerned with the analysis and experimental validation of the relationship among static metrics, including one proposed which incorporate both volume and control organization indexes. Table I lists the 31 metrics implemented and tested. A Fortran static code analyzer, FORTRANAL, was developed to compute the met-

Manuscript received September 28, 1984; revised May 30, 1986. This work was completed while the authors were with the University of Hong Kong.

H. F. Li is with the Department of Computer Science, Montreal, P.Q. H3G 1M8, Canada.

W. K. Cheung is with the Department of Electrical Engineering, University of Hong Kong, Hong Kong.

IEEE Log Number 8714442.

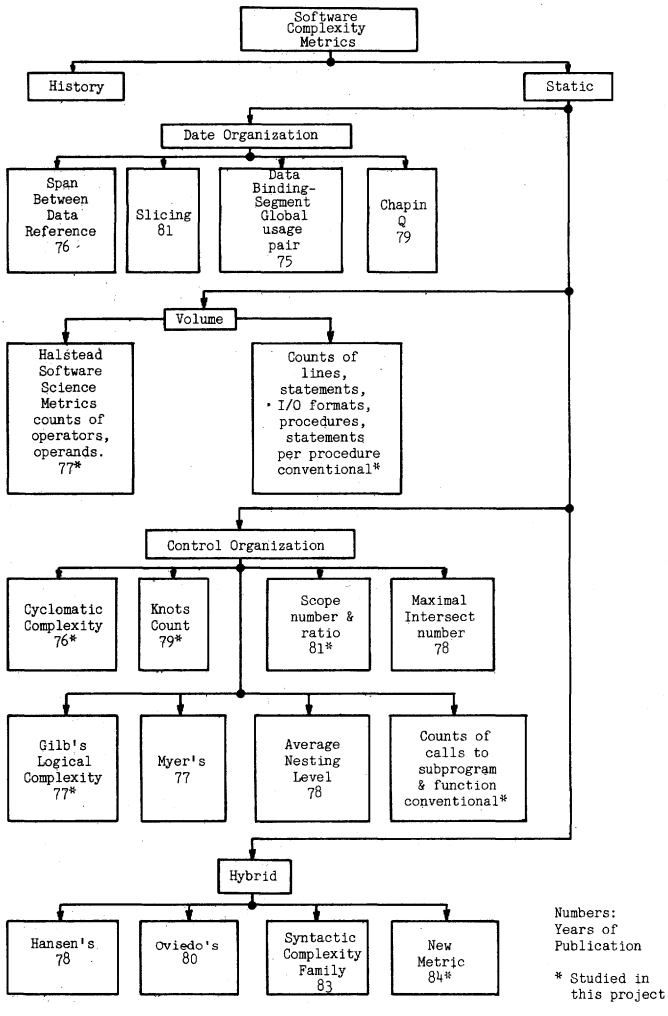


Fig. 2. Classification of complexity metrics.

rics. Fig. 3 summarizes the types of student assignments used as test programs and their related statistics. By establishing such empirical evidence of the appropriateness of candidate metrics, other techniques and methodologies may be developed to apply the use of metrics in software development [18], [19]. For example, [14] demonstrated that Halstead's and McCabe's metrics, in addition to statements, calls, and jumps, are related to development effort and error. However, the latter issue is beyond the scope of the paper.

II. Principles

Static metrics are not directly related to the execution of an algorithm or program, because only the machine language versions are normally executed. These measures are obtained by analyzing statical structures of the program concerned.

Three factors can affect maintenance: program size, data organization, and control structure. A number of measures have been developed to consider more than one. There exist various ways to count these measures and this section interprets the counting procedures used by FORTRANAL and defines each of the metrics relative to Fortran.

TABLE I
Metrics Investigated

METRICS	BRIEF DEFINITIONS	SUB-CLASS
1. LINES	lines of code count	Volume
2. STMTS	statements count	Volume
3. FOMTS	I/O formats count	Volume
4. LN-CM	total number of lines excluding comments	Volume
5. UNITS	programming modules count	Volume
6. STM/U	mean number of statements per module	Volume
7. NODES	count of nodes in planar flow graph	Vol/Con
8. EDGES	count of edges in planar flow graph	Vol/Con
9. McCBE	cyclomatic complexity	Control
10. SCOPE	scope metric	Control
11. SCORT	scope ratio	Control
12. n1	unique operator count	Volume
13. n2	unique operand count	Volume
14. N1	total operator count	Volume
15. N2	total operand count	Volume
16. n	vocabulary n1+n2	Volume
17. N	program length N1+N2	Volume
18. N^	calculated program length n1Logn1+n2Logn2	Volume
19. V	program volume NLogn	Volume
20. L^	calculated program level 2.n2/n1/N2	Volume
21. IC	intelligence content VL^	Volume
22. D2	calculated program difficulty 1/L^	Volume
23. E^	calculated program effort 1 V/L^	Volume
24. E^^	calculated program effort 2 N^Logn/L^	Volume
25. CALLS	number of calls to subroutines & functions	Control
26. NEW_1	new metric (scope with Halstead's)	Hybrid
27. CA+BD	sum of procedural calls and binary decision	Control
28. CL	Gilb's (absolute logical complexity)	Control
29. cL	Gilb's (relative logical complexity)	Control
30. KNOT1	Knot count metric 1 (real knots count)	Control
31. KNOT2	Knot count metric 2 (possible knot count)	Control

CLASSES	COURSE	ASSIGNMENTS	CODE
E.Eng.I	85110 Computer Programming Techniques	1. area of triangle	A
		2. Newton-Raphson Iteration	B
		3. Lagrangian Interpolation	C
Computer Sciences I	CS0101	1. Income, expense & profit	D
		2. Convert Decimals to Reals	E
		3. Definite Integrals	F

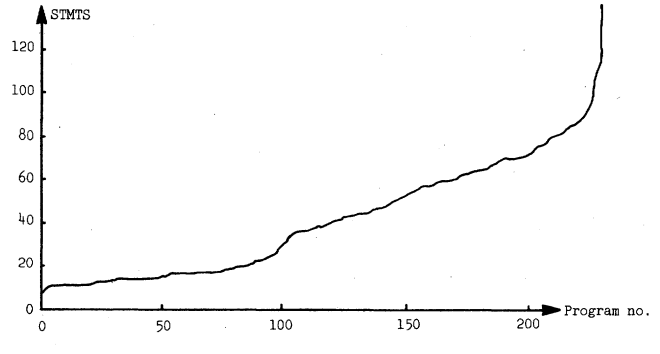


Fig. 3. Data investigated.

A. Conventional Volume Metrics

The most straightforward and widely applicable approach is based on program size. Very large programs incur problems just by virtue of the volume of information that must be absorbed to understand the program. These are easy to be calculated with definable measures, as exemplified in Fig. 4:

1) LINES counts all source lines including comments,

```
LINE   LABEL   STMT   NODE

   1       0      0          $JOB WATFOR
   2       0      0          C
   3       0      0          C PROGRAM TO FIND THE ROOT OF THE EQUATION X**X = 10
   4       0      0          C ITERATION
   5       0      0          C
   6       0      0          C CHAN CHI HUNG
   7       0      0          C EI
   8       0      0          C
   9       0      1      1         READ, X, E, 0
  10       0      2      1         IF(X.LT.0) GOTO 20
  11       0      3      2         I=1
  12      10      4      3   10    Z=X**X*(ALOG(X)+1)
  13       0      5      3         IF(ABS(Z).LT.D) GOTO 30
  14       0      6      4         Y=X-(X**X-10)/Z
  15       0      7      4         IF(Y.LT.0) GOTO 40
  16       0      8      5         I=I+1
  17       0      9      5         IF(ABS(X-Y).LT.E) GOTO 60
  18       0     10      6         IF(I.GT.30) GOTO 50
  19       0     11      7         X=Y
  20       0     12      7         GOTO 10
  21      20     13      8   20    PRINT, 'Initial guess smaller than zero'
  22       0     14      8         GOTO 70
  23      30     15      9   30    PRINT, 'Derivation of the function vanishes'
  24       0     15      9       A, Newton-Raphson interation invalid'
  25       0     16      9         GOTO 70
  26      40     17     10   40    PRINT, 'Invalid initial guess, next approx'
  27       0     18     10         GOTO 70
  28      50     19     11   50    PRINT, 'Number of interations exceeds 30
  29       0     19     11       A,guess'
  30       0     20     11         GOTO 70
  31      60     21     12   60    WRITE(6, 80)Y
  32      80      0          80    FORMAT(11X, 'Root of equation = ',E15.6)
  33      70     22     13   70    STOP
  34       0     23     14         END
  35       0      0          $DATA

                NUMBER OF LINES    35          LINES=35
              NUMBER OF STATEMENTS 23          STMTS=23
                NUMBER OF COMMENTS  7
                NUMBER OF FORMATS   1          FOMTS=1
            SOURCE LINES- COMMENTS 28          LN-CM=28
      NUMBER OF PROGRAMMING UNITS   1          UNITS=1
      MEAN NO. OF STATEMENTS/UNIT 23.0         STM/U=23
```

Fig. 4. Example—conventional volume metrics.

specifications, I/O formats, compiler control, debug, and executable statements.

2) STMTS counts executable statements such as assignments, control, I/O, and arithmetric statement functions.

3) FOMTS is the number of I/O formats.

4) LN-CM counts all source lines excluding comments.

5) UNITS counts programming modules: subroutines, functions, etc.

6) STM/U is the average length of a programming module, STMTS/UNIT.

Thus, LINES, STMTS, and LN-CM measure the physical size and the rest measures the logical size of a program.

B. Halstead's Software Science [3]

It is one of the most widely accepted measures with several empirical studies. It involves metrics defined by some key constituents of a program implementing an algorithm.

1) Operators & Operands: An algorithm consists of operators and operands, and of nothing else. Operators fall into three classes:

a) Basic —+ − * ** / // = ( ) .GT. .GE. .LT. .LE. .NE. .EQ. .NOT. .AND. .OR. .EQV. .XOR. .NEQV.

b) Keyword—IF THEN ELSE ELSEIF ENDIF DO DOWHILE GOTO ASSIGN CONTINUE ENDDO READ WRITE TYPE PRINT ACCEPT EOS (implicit statement delimiter)

c) Special —Names of subroutines, functions.

Operands consist of all variable names and constants such as, .TRUE. .FALSE. and Esnn(real). Then Halstead's metrics can be defined based on the number of distinct operations $n1$, the number of distinct operands $n2$, the total number of operators $N1$, and the total number of operands $N2$.

2) Derived Metrics: Halstead defines the vocabulary of the program as $n = n1 + n2$ (the total number of distinct operators and operands) and the implementation length as $N = N1 + N2$. These are meaningful volume measures of a program. He hypothesizes an estimator $\hat{N} = n1\text{Log}n1 + n2\text{Log}n2$ for N. A program volume metric V defined as $N\text{Log}n$ characterizes the size of an implementation, which can be regarded as the number of bits necessary to encode the whole module.

The potential volume $V^* = n^*\text{Log}n^*$ represents the

17 OPERATORS	
READ	1
EOS	21
IF	5
( )	10
.LT.	4
GOTO	10
=	5
**	2
*	1
ALOG	1
+	2
ABS	2
-	3
/	1
.GT.	1
PRINT	4
WRITE	1
TOTAL	74

15 OPERANDS	
X	9
0	2
20	1
I	4
I	3
Z	3
D	1
30	2
Y	4
10	2
40	1
E	1
60	1
50	1
70	4
TOTAL	39

n1	n2	N1	N2	n	N	N^	V
17	15	74	39	32	113	128.090	565.000

L^	IC	D2	E^	E^^
0.04525	25.566	22.10000	12486.500	14153.971

Fig. 5. Example—Halstead's software science.

minimum algorithm representation in a language where the required operation is "built in." Hence, the potential vocabulary $n^* = n1^* + n2^* \geq 2 + n2^*$ because in such a minimal form, the number of operators is two: the algorithm name and ( ).

To evaluate the programming effort, propensity of error, and ease of understanding, the program level L of an implementation is defined as V^*/V which has the maximum value of unity and can be approximated by $\hat{L} = 2 \cdot n2/n1/N2$. It follows that only the most succinct expression can have a level of unity. Program difficulty D is the difficulty of coding an algorithm. $D = 1/L$ by definition and can be estimated by $D2 = 1/\hat{L}$.

Halstead hypothesizes that LV remains invariant under translation from one language to another. LV can therefore be regarded as the intelligence context IC of the algorithm which increases only as the complexity of problem solution increases.

The effort required to generate an algorithm is $E = V/L$. It is suggested that E can measure the effort required to comprehend an implementation and is a measure of CLARITY. Effort E can be approximated by $\hat{E} = V/\hat{L} = (n1N2 \cdot N\mathrm{Log}n)/(2n2)$ or $\hat{\hat{E}} = (n1N2 \cdot \hat{N}\,\mathrm{Log}n)/(2 \cdot n2)$.

For the program in Fig. 4, Halstead's software science metrics are exemplified in Fig. 5.

C. Directed Graph and Control Flow

The effects of control flow on program complexity attracted much research over the past ten years. A 50-line program with 25 IF-THEN-ELSE's has 33 million control paths! To expose the control flow topology, we represent the program as a flow graph, $G = (V, E)$. The directed graph consists of a set of nodes V and a set of directed edges E.

a) *Node*—A sequential block of code with unique entrance and exit but no internal branch or loop.
b) *Edge*—Flow of control between the various nodes.

For an edge (u, v), node u is the initial node and node v is the terminal node. The outdegree of node u is the number of edges emanating from u; the indegree of node is the number of edges incident at u. If edge (u, v) exists, then u is immediately preceding v and v is immediately succeeding u. If a path exists from u to v then u precedes v and v succeeds u.

Using those basic structures in Fig. 6 as guidelines, the flow graph of the example program can be constructed as shown in Fig. 7. Various control metrics can be formulated which characterize the control complexity of a given flow graph.

1) McCabe and Gilb [4], [5]: McCabe's cyclomatic complexity is well accepted, intuitively reasonable, and easily calculated by $V(G)$ = EDGES − NODES + 2*UNITS. In a strongly connected graph, this cyclomatic number is the number of linear independent circuits. For programs with single entry and single exit, $V(G)$ is one plus the number of decisions. The graph theoretic metric is independent of the program size but depends only on the decision structure. Decision making of a program affects its error probability and development time and cost.

Alternately, Gilb proposes two metrics: CL, absolute logical complexity (number of binary decisions) and cL, relative logical complexity (ratio of CL to STMTS) which have been supported by some empirical evidence. The latter can be considered as an improvement over pure control metrics as it also takes into account some volume metric. For the example program, the McCabe and Gilb's metrics are shown in Fig. 8.

In addition, two conventional metrics related to Gilb's can be defined:

a) *CALLS* —The number of subroutine and function invocations.
b) *CA + BD* —The total number of calls and binary decisions.

2) KNOTS Count [6]: A knot occurs when two control transfers intersect, as depicted in Fig. 9. Since each node is a sequence of statements with no internal branches, a knot occurs if node b includes at least one line in the example.

Two related metrics can be further defined:

a) *KNOT1*—The number of knots that can be verified.
b) *KNOT2*—The total number of potential knots, assuming every node contains one statement.

It is conceivable that a program with many knots is more complex to comprehend and the KNOT metrics are defined to reflect this.

3) SCOPE Metric and Ratio [7], [8]: Nodes with an outdegree 0 or 1 are RECEIVING nodes. Those with an

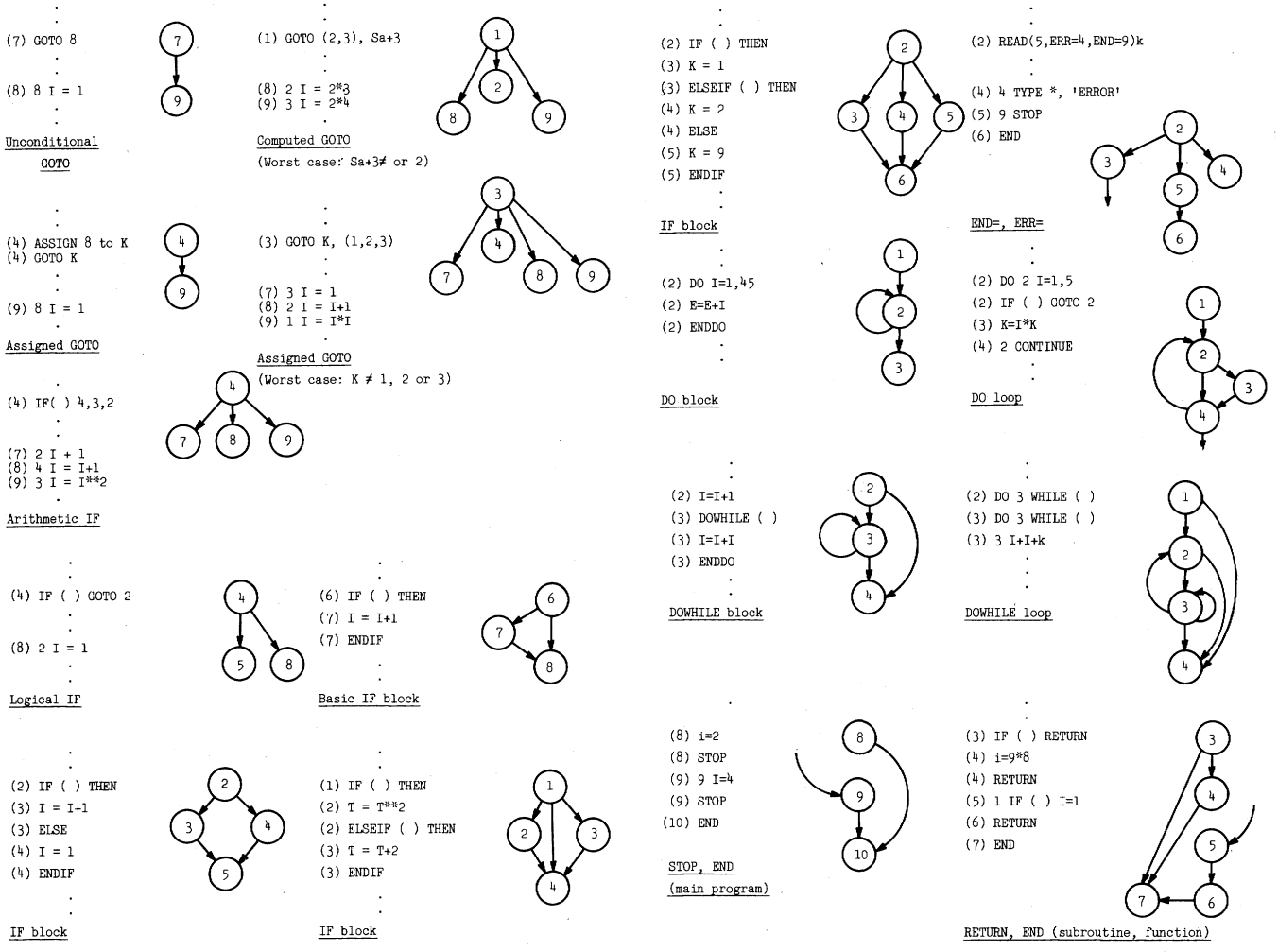


Fig. 6. Basic control structures in Fortran.

NUMBER OF NODES 14 NODES = 14

NUMBER OF EDGES 18 EDGES = 18

NODE NUMBER	STARTING STATEMENT	FINISHING STATEMENT	OUTDEGREE	IMMEDIATE SUCCESSOR
1	1	2	2	2, 8,
2	3	3	1	3,
3	4	5	2	4, 9,
4	6	7	2	5, 10,
5	8	9	2	6, 12,
6	10	10	2	7, 11,
7	11	12	1	3,
8	13	14	1	13,
9	15	16	1	13,
10	17	18	1	13,
11	19	20	1	13,
12	21	21	1	13,
13	22	22	1	14,
14	23	23	0	

Fig. 7. Example—numerical flowchart of the program.

McCabe's metric

UNIT	COMPLEXITY
1	6

McCabe'S CYCLOMATIC COMPLEXITY: 6

Gilb's metrics

UNIT	ABSOLUTE LOGICAL COMPLEXITY	RELATIVE LOGICAL COMPLEXITY
1	5	0.217
TOTAL	5	21.73%
	[CL]	[cL]

Conventional metrics

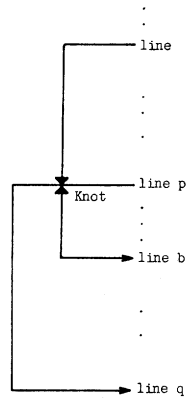
NUMBER OF CALLING FUNCTIONS/SUBROUTINES 3

TOTAL NUMBER OF BINARY DECISIONS AND CALLS 8

McCBE = 6 CL = 5 cL = 21.7% CALLS = 3 CA+BD = 8

Fig. 8. McCabe's and Gilb's metrics of program in Fig. 4.

outdegree greater than 1 are SELECTION ones. Given a selection node, we can find at least one "lower bound" node which succeeds every immediate successor of the selection node. The lower bound node that precedes every

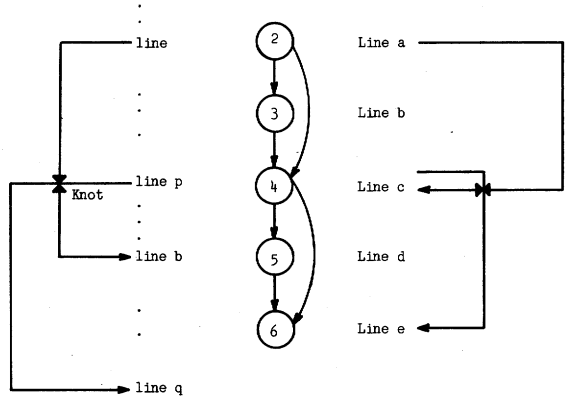


Control Flow knot occurs:
transfer (a,b) interrupted
by transfer (p,q)

UNIT	REAL KNOTS	POSSIBLE KNOTS
1	0	1
TOTAL	0	1

KNOT1 = 0 KNOT2 = 1

A knot occurs at node 3 (Consult Fig. 10) if it contains only one statement.



Control Flow knot occurs:
transfer (a,b) interrupted
by transfer (p,q)

Flow graph with knot. A knot occurs when transfer between two nodes is interrupted.

UNIT	REAL KNOTS	POSSIBLE KNOTS
1	0	1
TOTAL	0	1

KNOT1 = 0 KNOT2 = 1

A knot occurs at node 3 (Consult Fig. 10) if it contains only one statement.

Fig. 9. Knot count metrics of the program.

other lower bound is the GREATEST LOWER BOUND (GLB). The number of nodes preceding the GLB and succeeding the selection node, plus 1, yields the ADJUSTED COMPLEXITY (AC) of that selection node. It reflects the scope of "influence" of the selection node. Summing up the adjusted complexity of each node, the SCOPE metric is formed.

SCORT, the scope ratio metric, is defined as:

$(1.0 - N/\text{SCOPE}) * 100\%$, where N = number of nodes in the flow graph excluding the terminal node.

SCORT increases towards 100 (percent) as complexity increases. For the example, the resulting metrics are shown in Fig. 10.

NODE	SCOPE								REMARKS
1	2	3	4	5	6	7	8		
	9	10	11	12					
								COMPLEXITY 12	GLB = node 13
2								COMPLEXITY 1	Receiving node
3	4	5	6	7	3			COMPLEXITY 6	GLB = node 9
4	3	5	6	7	4			COMPLEXITY 6	GLB = node 10
5	3	4	6	7	5			COMPLEXITY 6	GLB = node 12
6	3	4	5	7	6			COMPLEXITY 6	GLB = node 11
7								COMPLEXITY 1	Receiving node
8								COMPLEXITY 1	.
9								COMPLEXITY 1	.
10								COMPLEXITY 1	.
11								COMPLEXITY 1	.
12								COMPLEXITY 1	.
13								COMPLEXITY 1	Receiving node
14								COMPLEXITY 0	terminal node
								COMPLEXITY OF UNIT 44	SCOPE = 44
								SCOPE RATIO OF UNIT 70.455 %	SCOPE = 70

Fig. 10. SCOPE and SCORT metrics of the program.

NODE	n	N	V	RAW COMPLEXITY	RAW COMPLEXITY	COMPLEXITY
1	9	10	31.699	700.553	8092.196	
2	4	4	8.000	176.800	176.800	
3	16	23	92.000	2033.200	8364.042	
4	15	21	82.045	1813.188	8144.030	
5	15	18	70.324	1554.161	7885.004	
6	8	8	24.000	530.400	6861.243	
7	6	7	18.095	399.894	399.894	
8	4	5	10.000	221.000	221.000	
9	4	5	10.000	221.000	221.000	
10	4	5	10.000	221.000	221.000	
11	4	5	10.000	221.000	221.000	
12	2	2		0.000	0.000	
13	0	0		0.000	0.000	
14	0	0		0.000	0.000	
SUM				8092.196	40807.211	80.170%
TOTAL				8092.196	40807.211	80.170%

GLOBAL LEVEL 0.04525

Fig. 11. A hybrid metric—NEW__1 of program.

D. Hybrid Metric—NEW__1

We propose a new hybrid metric which integrates software science with the scope of measure, and reflects both volume and control organization. The raw complexity of a node V_j is $\hat{E_j}$:

$$E_j \triangleq N_j \operatorname{Log} n_j / \hat{L}$$

where N_j, n_j are local parameters of node V_j, and $\hat{L}$ is a global parameter defined previously.

The adjusted complexity for a selection node is the sum of $\hat{E_j}$ values of every node within the SCOPE of that selection node, plus the value of the selection node itself. A receiving node has an adjusted complexity equal to its raw complexity. The complexity of the overall program is the sum of the adjusted complexities of every node. The intuitive justification for proposing this metric is to incorporate volume metric into an otherwise pure graph theoretic metric, resulting in a hybrid that may be more useful in characterizing software complexity.

We define the metric NEW__1 as:

NEW__1 = (1.0 − Σ Raw Complexities/Σ Adjusted Complexities) * 100%

NEW__1 increases towards 100 (percent) as complexities increases.

For the sample program, its NEW__1 metric is shown in Fig. 11.

In fact, this hybrid metric is a special instance of a class of metrics derivable by combining the SCOPE idea with software science in general. The raw complexity of a node could be represented by one of the volume metrics characterizing the volume "complexity" of the node which when merged into the SCOPE to form the aggregate adjusted complexity may reflect naturally the desirable mix of two different parameters in the metric. For the purpose of this experiment, we have chosen the above definition of raw complexity in the evaluation. The actual use of this class of hybrid metric may depend on the environment and objective of the measurement or estimation conducted.

To derive quantitative comparison among different metrics and in particular, to demonstrate their inherent relationship, the correlation coefficient of their values for various sample programs may be obtained. In general, this coefficient measures the relationship between two random variables X and Y. It plays a major role in many bivariate data analysis problems. It might be said that sample estimates of correlation coefficient close to unity imply good CORRELATION or LINEAR ASSOCIATION between X and Y, while values near zero indicate little or no correlation.

To gain insight on the degree of linear relationship between X and Y, standard definition of the correlation coefficient is [15]:

$$\text{correlation coefficient} = \frac{S_{xy}}{\sqrt{S_{xx}S_{yy}}}$$

where

$$S_{xx} = \sum_i (X_i - \bar{X})^2$$

$$S_{yy} = \sum_i (Y_i - \bar{Y})^2$$

$$S_{xy} = \sum_i (X_i - \bar{X})(Y_i - \bar{Y}).$$

III. Result and Analysis

A four-pass program, FORTRANAL, is developed for evaluating the objective metrics and the new one proposed for the 255 sample programs under test. Briefly stated, the computations performed in each pass are:

Pass-1:

1) Compute LINES, STMTS, FOMTS, LN-CM.
2) Collect the names of subroutines and functions.
3) Eliminate nonexecutable statements.

Pass-2:

1) Compute UNITS, STM/U, NODES, EDGES, SCOPE, SCORT, McCaBE, CL, cL, KNOT1, and KNOT2.
2) Construct the flow graph.

Pass-3:

1) Count the number of operators and operands of the whole program.
2) Obtain $n1$, $n2$, $N1$, $N2$, n, N, $\hat{N}$, V, $\hat{L}$, IC, $D2$, $\hat{E}$, $\hat{\hat{E}}$, CALLS, CA + BD.

Pass-4:

1) Count the number of operands and operators of each node.
2) Evaluate NEW__1.

The major computation in FORTRANAL is the construction of a flow graph which requires the detection of all possible branching and looping paths and special data structure. Individual reports for a certain program can be obtained by executing FORTRANAL once. The output contains tables as exemplified in Figs. 4, 5, and 7–11.

The output file includes:

1) Program listing of the source code with line number, statement label, statement number, and node number.
2) Report on LINES, STMTS, FOMTS, LN-CM, UNITS and STM/U.
3) Listing of programming units: starting and finishing statements; starting and finishing nodes.
4) Listing of nodes: starting and finishing statements, outdegree, and all immediate successors.
5) Report on McCabe's complexity metric of each unit and the whole program.
6) Report on the adjusted complexities of nodes, SCOPE, and SCORT of programming units and the whole program.
7) Halstead's metrics of each node and the whole program; tables of operators and operands of the program.
8) Raw and adjusted complexities of the nodes, NEW__1 of the programming units and the whole program.
9) Gilb's logical complexity metrics of each programming unit and the whole program.
10) Report on $CA + BD$.
11) Report on the knot count metrics of each programming unit and the program.

The results of the six sets of programs (255 programs in total) were analyzed using another program, ANALYSIS. It evaluates the correlation coefficient between any pair of complexity metrics.

Among the 7905 values measured, there are certain general relationships that can be observed. For example, after calculating the correlation coefficient between each pair of metrics a group of metrics whose correlations are high can be extracted, as revealed in Table II. It shows a close relationship in VOLUME (e.g., STMTS, LN-CM, Halstead's, and CL) and CONTROL ORGANIZATION (e.g., SCOPE, McCBE, NODES, and EDGES) metrics.

A. Halstead's Metrics

This family of metrics seems to possess reasonable internal consistency, that is, with correlation coefficient close to unity, as can be observed in Table II. This is further evidence to previous studies such as that in [14].

1) Length Equation: Halstead suggests that $N = N1 + N2$ can be approximated by $\hat{N} = n1\text{Log}n1 + n2\text{Log}n2$. This length equation appears to be program-size dependent in our study (Fig. 12). $\hat{N}$ tends to be high for small programs and low for larger ones.

In our environment, $\hat{N}$ significantly overestimates N for $N < 170$ and underestimates N for $N > 200$. To show

TABLE II
CORRELATION COEFFICIENTS AMONG 18 SELECTED METRICS

	STMTS	LN-CM	NODES	EDGES	McCBE	SCOPE	n2	N1	N2	n	N	N^	V	IC	E^	E^^	CL
STMTS																	
LN-CM	.983																
NODES	.924	.906															
EDGES	.914	.875	.982														
McCBE	.908	.891	.964	.971													
SCOPE	.848	.797	.910	.947	.892												
n2	.898	.877	.896	.889	.872	.826											
N1	.977	.971	.916	.898	.905	.833	.925										
N2	.942	.933	.917	.903	.915	.828	.953	.976									
n	.907	.893	.920	.899	.886	.832	.987	.933	.950								
N	.968	.960	.921	.906	.915	.836	.943	.996	.992	.946							
N^	.896	.878	.913	.898	.881	.837	.989	.925	.947	.998	.940						
V	.960	.949	.927	.914	.918	.852	.956	.990	.992	.959	.997	.958					
IC	.865	.834	.810	.824	.796	.780	.956	.882	.891	.907	.891	.912	.900				
E^	.914	.913	.905	.873	.897	.805	.845	.940	.937	.884	.944	.880	.947	.728			
E^^	.886	.882	.917	.881	.892	.813	.887	.914	.925	.931	.924	.931	.938	.748	.976		
CL	.878	.830	.930	.978	.969	.932	.851	.862	.872	.848	.872	.850	.880	.803	.830	.828	
KNOT2	.871	.830	.919	.948	.923	.877	.855	.861	.872	.848	.871	.845	.873	.815	.803	.799	.943

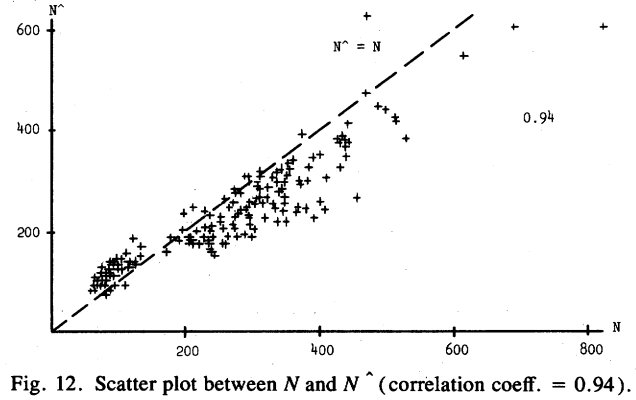


Fig. 12. Scatter plot between N and $N\hat{}$ (correlation coeff. = 0.94).

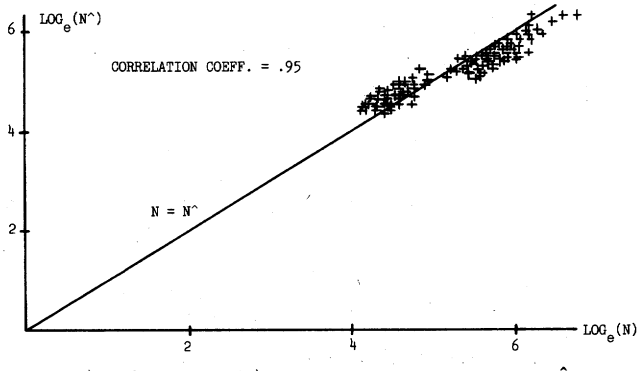
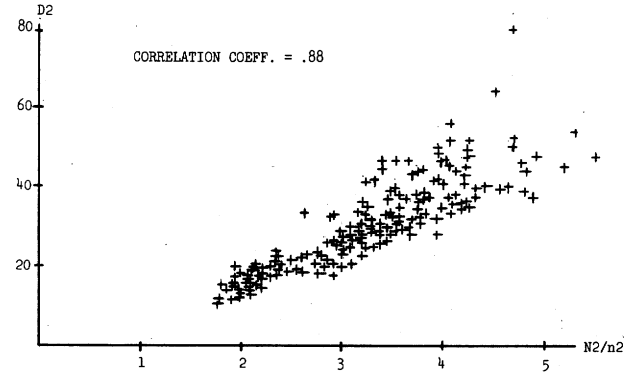


Fig. 13. Scatter plot of natural logarithms of N and $N\hat{}$.



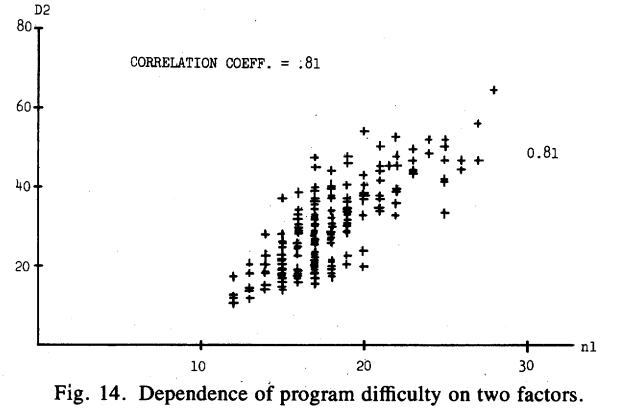


Fig. 14. Dependence of program difficulty on two factors.

moderate improvement on the correlation coefficient, $\mathrm{Log}(N\hat{})$ is plotted against $\mathrm{Log}(N)$ in Fig. 13. The natural logarithms of estimated and observed length correlate better and their relation changes less with program size.

2) Program Level and Difficulty: Halstead hypothesizes that $D2 = 1/L\hat{} = n1N2/2N2$. There are two factors affecting $D2$: unique operator count $n1$ and the average operand usage $N2/n2$. It may be demonstrated that in our environment, the latter factor dominates slightly (Fig. 14).

From the figures, apparently the operator count is a minor factor. Clearly, in these assignments, the number of unique operators tends to remain relatively constant with respect to length. Hence, the factor $N2/n2$ is the main contributor to the program difficulty $D2$ (Fig. 15).

3) Program Effort: Halstead attempts to use effort metric to quantify the effort required to comprehend the

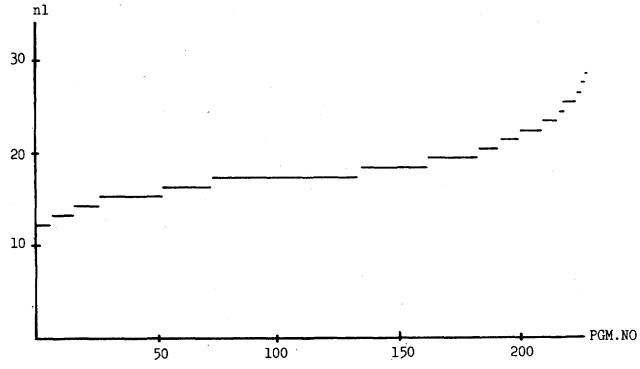


Fig. 15. Unique operator count seems to be relatively constant.

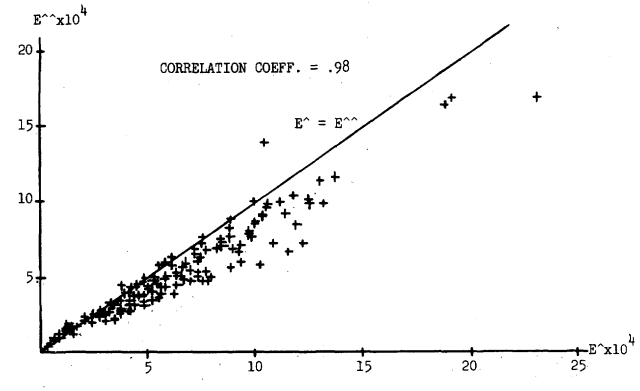


Fig. 16. Scatter plot between two metrics: $E\hat{}$ and $E\hat{}\hat{}$.

implementation of an algorithm. The two estimators $E\hat{}$ and $E\hat{}\hat{}$ are given by $E\hat{} = (N\text{Log}n)/L\hat{}$ and $E\hat{}\hat{} = (N\text{Log}n)/L\hat{}$. However, they correlate better than the corresponding pair N and $N\hat{}$ (Fig. 16).

Actually, Halstead's metrics are based on a refinement of measuring program size by counting the number of lines of codes. Hence, we should expect that $n2$, $N1$, $N2$, n, N, $N\hat{}$, V, IC, $E\hat{}$, and $E\hat{}\hat{}$ form a group of metrics with good internal consistency. Observe that $n1$ is relatively constant in our environment and cannot correlate well with the others.

From the analysis, Halstead's metrics do form a group of consistent metrics suggesting that one of them may replace the others in application. However, Halstead's suggestion of using different metrics to explain different life cycles of software development would prevent us from doing so.

Table III shows the correlation of Halstead's metrics with other selected metrics. Referring to the table, N and V have similar correlations and correspond quite well with most of the other metrics. $E\hat{}$ and $E\hat{}\hat{}$ is another pair but they do not correlate as well with the other 10 metrics. Halstead's metrics do not correlate well with CALLS. This suggests that CALLS may not be a reliable VOLUME metric.

Most of Halstead's metrics, such as $N1$, $N2$, $n2$, n, N, $N\hat{}$, V, IC, $E\hat{}$, and $E\hat{}\hat{}$, depend on program size (represented by STMTS). Of these, $N1$, $N2$, N, V are best correlators with STMTS, because they are typical program "size" metrics. E seems to be a better correlator than $E\hat{}\hat{}$ and similarly N correlates better than $N\hat{}$. Among the three size metrics (LINES, STMTS, and LN-CM), most of Halstead's metrics correlate best with STMTS. This suggests that STMTS is a better VOLUME metric than the other two.

TABLE III
CORRELATION COEFFICIENT MATRIX BETWEEN HALSTEAD'S METRICS AND 10 SELECTED CLASSICAL METRICS

	LINES	STMTS	McCBE	In-CM	NODES	EDGES	SCOPE	CA+BD	CALLS	KNOT2
n1	.69	.62	.63	.65	.70	.62	.56	.69	.41	.51
n2	.79	.90	.87	.88	.90	.89	.83	.70	.14	.85
N1	.86	.98	.91	.97	.92	.90	.83	.81	.27	.86
N2	.83	.94	.92	.93	.92	.91	.83	.77	.21	.87
n	.82	.91	.89	.89	.92	.90	.83	.75	.20	.85
N	.85	.97	.92	.96	.92	.91	.84	.80	.25	.87
N^	.81	.90	.88	.88	.91	.90	.84	.73	.18	.85
V	.84	.96	.92	.95	.93	.92	.85	.79	.23	.87
L^	-.73	-.73	-.70	-.77	-.70	-.65	-.56	-.73	-.40	-.63
IC	.71	.87	.80	.83	.81	.82	.78	.61	.07	.82
D2	.78	.77	.75	.81	.76	.69	.59	.79	.45	.64
E^	.84	.91	.90	.91	.91	.87	.81	.81	.30	.80
E^^	.83	.89	.89	.88	.92	.88	.81	.78	.27	.80

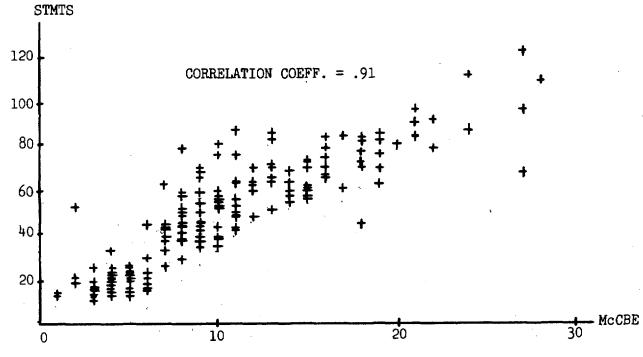


Fig. 17. Scatter plot between STMTS and McCBE.

B. McCabe's Metric

This graph-theoretic complexity measure has unity as a lower bound but has no upper bound. It grows faster than UNITS because the addition of a module always increases $V(G)$ by at least one. The cyclomatic complexity can be viewed as a control organization metric (i.e., number of control paths) and to a lesser extent, a volume metric (i.e., number of decisions + 1). Here, it correlates well with module size STMTS (Fig. 17).

Actually, from Table II, McCabe's metric correlates well with Halstead's, Gilb's, Knot Counts, SCOPE, EDGES, and NODES metrics. So, the cyclomatic complexity metric seems to bridge the gap between the two categories: VOLUME and CONTROL ORGANIZATION metrics.

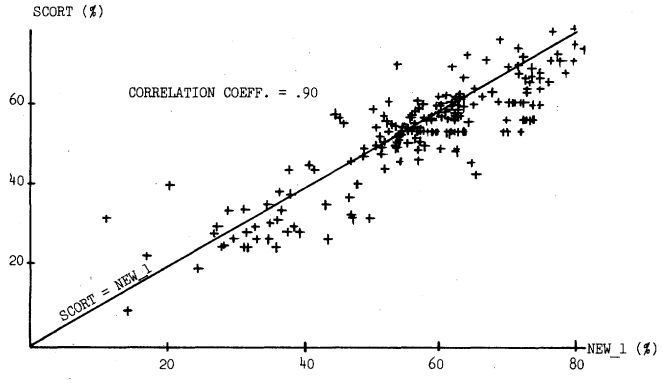


Fig. 18. Scatter plot between SCORT and NEW__1 metrics.

C. SCOPE Metric

The scope number, in essence, is dependent on the number of nodes in the flow graph. Obviously this measure cannot always be reliable, since some programs can be trivially rearranged to give flow graphs with different scope measures.

From our observation, SCOPE correlates well with NODES (correlation coefficient = 0.91) and EDGES (correlation coefficient = 0.95). Moreover, its relationship with most of Halstead's metrics is fair.

D. SCORT and NEW__1 Metrics

The scope ratio and our hybrid metric tend to remain constant and cannot correlate well with the other size metrics. The new hybrid metric integrates software science with the scope ratio measure and is found to be dominated by the latter. For this reason, NEW__1 correlates well with SCORT (Fig. 18).

In Fig. 18, both SCORT and NEW__1 remain relatively constant around 60 percent for most samples in the test. This strong correlation may occur when the volume measures of each node tend to be uniform in the samples leading to similar values for both measures. In cases where the raw complexities of a node can differ significantly from those of others, the correlation may deteriorate!

E. Knot Count Metrics

KNOT2 is a much better correlator than KNOT1 with the traditional measures as shown in Table IV(a). It may be due to the definition of KNOT2 which is based on the static control structure, as are most of the rest. (KNOT2 is the number of possible knots). Notice that KNOT1 counts the number of knots detected and is much less than KNOT2. The step discontinuity of KNOT1 is much more pronounced than that of KNOT2 as shown in Table IV(b).

F. Gilb's Logical Complexity Metrics

Similar to the Knot metrics, the absolute logical complexity CL correlates better with those traditional metrics than the relative logical complexity cL, which takes into account the program size. In fact, CL is the number of binary decisions in the program's logic and can be regarded as a special volume metric too, as is observable in Table V.

TABLE IV

(a) CORRELATION COEFFICIENTS AMONG KNOT COUNT METRICS AND EIGHT SELECTED MEASURES. (b) RANGE OF KNOT COUNT METRICS MEASURED—KNOT1 HAS GREATER STEP DISCONTINUITY.

	STMTS	NODES	EDGES	McCBE	SCOPE	N	V	CL
KNOT1	.74	.73	.75	.69	.70	.69	.68	.74
KNOT2	.81	.92	.95	.92	.88	.87	.87	.94

Correlation coefficient between KNOT1 and KNOT2 = .81

(a)

Program Set	A	B	C	D	E	F	Overall
mean STMTS	15	20	62	66	72	46	48
mean KNOT1	0	.1	2.2	2.8	3	.5	1.2
maxim. KNOT1	0	1.	4.0	6.0	7	1.	7.0
minim. KNOT1	0	0.	0.0	0.0	0	0.	0.0
mean KNOT2	.9	1.	7.4	5.5	9.6	3.	4.1
maxim. KNOT2	3.	2.	19	14	16	5	19.0
minim. KNOT2	0	0	0.0	2.0	4	1	0.0
no. of programs	54	56	60	22	30	33	255
Cor. Coef. (KNOT)	-	.5	.6	.7	.6	.2	.81

Program Set	P	Q	R	S	T
no. of programs	51	51	51	51	51
STMTS	min.				max.
Correl. Coeff. (KNOT)	-	.2	.7	.7	.3

(b)

TABLE V

CORRELATION COEFFICIENTS AMONG GILB'S LOGICAL COMPLEXITY METRICS AND EIGHT SELECTED MEASURES

	STMTS	NODES	EDGES	McCBE	SCOPE	N	V	KNOT2
cL	0.5	.24	.35	.35	.37	.07	.09	.33
CL	.88	.93	.98	.97	.93	.87	.88	.94

G. Traditional VOLUME Metrics

Referring to Table VI(a), among STMTS, LINES, LN-CM, FOMTS, UNITS, and STM/U, the best volume metric that correlates well with all the others should be STMTS. Typical Halstead's N and V measures correlate best with STMTS and LN-CM.

Obviously, LINES can be arbitrarily made larger by adding comments. FOMTS can also be adjusted as we wish. UNITS and STM/U depend on our own choice of subprogram sizes. Hence, only STMTS and LN-CM are fair indexes of program sizes.

H. Traditional CONTROL ORGANIZATION Metrics

Referring to Table VI(b), among EDGES, NODES, CALLS AND CA + BD, NODES, and EDGES correlate very well with conventional VOLUME metrics. However, CALLS and CA + BD are isolated from the rest.

TABLE VI
(a) CORRELATION COEFFICIENTS AMONG CONVENTIONAL VOLUME METRICS AND TWO TYPICAL HALSTEAD'S MEASURES. (b) CORRELATION COEFFICIENTS AMONG FOUR CONVENTIONAL CONTROL ORGANIZATION METRICS WITH EIGHT SELECTED MEASURES.

	STMTS	LINES	LN-CM	FOMTS	UNITS	STM/U
N	.97	.85	.96	.71	.38	.64
V	.96	.84	.95	.69	.36	.65
LINES	.85	.				
LN-CM	.98	.89				
FOMTS	.75	.65	.81			
UNITS	.33	.53	.44	.25		
STM/U	.69	.44	.61	.55	-.37	

(a)

	STMTS	LN-CM	McCBE	SCOPE	N	V	CL	KNOT2
EDGES	.91	.88	.97	.95	.91	.91	.98	.95
NODES	.92	.91	.96	.91	.92	.93	.93	.92
CALLS	.18	.30	.13	-.07	.25	.23	-.07	-.01
CA+BD	.75	.81	.78	.60	.80	.79	.64	.65

(b)

IV. CONCLUSION

The validity of any proposed measure of software complexity cannot be assessed with precision. Obviously, different people view software differently and interpret its complexity differently. In general, an index is said to be valid if it succeeds to reflect what it purports to measure. Thus, changes in what it purports to measure should be reflected faithfully in the measure. The validity of some measures is sometimes questionable. In this study, we compare the proposed NEW__1 measure and the other new metrics (e.g., KNOT1, KNOT2, etc.) with the conventional ones (e.g., Halstead's, McCabe's). As no external validation of these new metrics was performed, we can at most conclude that the new metrics seem as good as any of the conventional ones in terms of validity.

In general, metrics based on measures of program size have been the most successful to date, with experimental evidence indicating that larger programs have greater maintenance costs than smaller ones. They are good nominal scale metrics to use in classifying programs into different "complexity category," but it is difficult to distinguish between different programs within the same category.

STMTS, LN-CM, $n2$, $N1$, $N2$, n, N, $\hat{N}$, V, IC, $\hat{E}$, and $\hat{\hat{E}}$ were observed to form an internally consistent group of VOLUME metrics. So any one of them may be adopted to measure program size. $n1$ was found to be relatively constant. They are easily computed, and widely applicable. However, as Chapin [9] has pointed out, the usage difficulty or convenience make some metrics more distinguished. Volume metrics can only be measured after the design has been carried out fully to the debugged code. But by then, it is usually too late to take corrective action!

Some metrics were found to be unreliable, e.g., LINES, FOMTS, UNITS, and STM/U. Programs of the same complexity (defined by other measures) can have a large range of these measures because comments, formats, and subprograms can be added or deleted at will without affecting much on the inherent complexity of the programs!

The length equation proposed by Halstead's was observed as program-size dependent, $\hat{N}$ overestimates N for small programs and underestimates N for large programs. However, this length equation suggests that the human brain obeys a more rigid set of rules than it has been aware of, and that the parameters $n1$, $n2$, $N1$, and $N2$ may serve as useful elements in eliciting further relationships, e.g., V, $\hat{E}$, $\hat{L}$.

A very significant portion of effort over the past 10 years is devoted to studying the effects of control flow on program complexity. Several CONTROL ORGANIZATION metrics were observed to correlate well with VOLUME metrics, e.g., McCABE, SCOPE, CL and KNOT2, NODES, EDGES. This is perhaps because some of them also reflect volume in a special context. However, this observation may not hold across other database. Those metrics with unit "%" were observed to remain relatively constant for our database, e.g., cL, SCORT, and NEW__1.

In general, the control flow metrics fail to be comprehensive and do not consider the contribution of any factor except control flow complexity. However, these metrics can differentiate between two programs of similar VOLUME metrics and certainly are related to the software quality. Hence, a useful approach is to use VOLUME metrics for prior classification and then to use CONTROL ORGANIZATION measures to evaluate the programs in detail.

Hybrid metrics attempt to remedy one of the shortcomings of the single-factor complexity metrics in use. The new hybrid metric NEW__1 considers two properties that are thought to contribute to software complexity-VOLUME and CONTROL ORGANIZATION. Our approach is to borrow part of the measure from an existing metric. NEW__1 combines a measure of control flow and program size, i.e., SCORT and $\hat{E}$ are considered together. The reasons for this choice are:

1) These two metrics are orthogonal, i.e., they measure different aspects of the complexity. From our observation, correlation coefficient between SCORT and $\hat{E}$ is only -0.032.

2) $\hat{E}$ is the most context-sensitive of VOLUME metrics. In fact, software science received a lot of empirical evidence and has a good internal consistency.

3) SCORT is sensitive to nested decisions and does not depend on program size.

Our samples show that SCORT dominates our new proposed hybrid metric. That is to say, NEW__1 is found to be slightly different from SCORT measures. $\hat{E}$ has negligible effect on the metric proposed. This is explainable from the uniformity of size of the modules in the samples, but may not hold true for other more heterogeneous samples.

The hybrid approach to measure complexity is, however, the most sensible approach. Software complexity is caused by so many different factors that measuring only one of them cannot help but give unreliable results for a general case. The measures studied in the project appear quite reliable, deterministic, and valid. Moreover, they can be implemented and tested, as in the FORTRANAL developed.

Most metrics lack context sensitivity. For example, EDGES, McCabe and CL, NODES consider only the node and edge counts and fail to consider the context of each edge and node. Halstead's count operators and operands only and cannot take into account the flow of control. Hence, most metrics lack comprehensiveness. Hybrid metrics are developed just to remedy these. Many are widely applicable, although of course, they may work better on some types of programs than others. In our case, only short Fortran programs were analyzed and the metrics investigated are applicable to all data.

Many metrics are supported by empirical evidence (e.g., Halstead's, McCabe's) but some have not been extensively tested, (e.g., KNOT, SCOPE, etc.). Our work is an attempt to evaluate the relationship among metrics which can guide us in adopting them in practice. The validation process must continue before metrics can be effectively adopted in the characterization and evaluation of software and in the prediction of its attributes, as well as aiding software development in its early phases in the form of well conceived guidelines and strategies. Only by understanding the properties and relationship of these metrics can we assertively apply them in both the development and assessment of software.

References

[1] W. Harrison, K. Magel, R. Kluczny, and A. DeKock, "Applying software complexity metrics to program maintenance," *Computer*, vol. 15, pp. 65–79, Sept. 1982.

[2] J. C. Baird and E. Noma, *Fundamentals of Scaling and Psychophysics.* New York: Wiley, 1978, pp. 1–6.

[3] M. H. Halstead, *Elements of Software Science.* New York: North-Holland (Elsevier Computer Science Library), 1977, pp. 1–60.

[4] T. J. McCabe, "A complexity measure," *IEEE Trans. Software Eng.*, vol. SE-2, pp. 308–320, Dec. 1976.

[5] T. Gilb, *Software Metrics.* Cambridge, MA: Winthrop, 1977.

[6] M. Woodward, M. Hennel, and D. Hedley, "A measure of control flow complexity in program TEXT," *IEEE Trans. Software Eng.*, vol. SE-5, pp. 45–50, Jan. 1979.

[7] W. Harrison and K. Magel, "A complexity measure based on nesting level," *ACM SIGPLAN Notices*, pp. 63–74, Mar. 1981.

[8] ——, "A topological analysis of computer programs with less than three binary branches," *ACM SIGPLAN Notices*, pp. 51–63, Apr. 1981.

[9] N. Chapin, "A measure of software complexity," in *Proc. Nat. Comput. Conf.*, 1979, pp. 995–1002.

[10] M. R. Woodward, M. A. Hennel, and D. Hedley, "A measure of control flow complexity in program text," *IEEE Trans. Software Eng.*, vol. SE-5, pp. 45–50, Jan. 1979.

[11] V. R. Basili and D. H. Hutchens, "An empirical study of a syntactic complexity family," *IEEE Trans. Software Eng.*, vol. SE-9, pp. 664–672, Nov. 1983.

[12] H. E. Dunsmore and J. D. Gannon, "Experimental investigations of programming complexity," in *Proc. ACM-NBS 16th Annu. Tech. Symp.: System Software*, Washington, DC, June 1977, pp. 117–125.

[13] B. Curtis, S. B. Sheppard, P. Milliman, M. A. Borst, and T. Love, "Measuring the psychological complexity of software maintenance tasks with the Halstead and McCabe metrics," *IEEE Trans. Software Eng.*, vol. SE-5, pp. 96–104, Mar. 1979.

[14] V. R. Basili, R. W. Selby, Jr., and T. Y. Phillips, "Metric analysis and data validation across Fortran projects," *IEEE Trans. Software Eng.*, vol. SE-9, pp. 652–663, Nov. 1983.

[15] R. Walpole and R. H. Meyers, *Probability and Statistics for Engineers and Scientists.* New York: MacMillan, 1979, pp. 299–304.

[16] A. Schroeder, "Integrated program measurement and documentation tools," in *Proc. Int. Conf. Software Eng.*, 1984, pp. 304–313.

[17] M. De Prycker, "On the development of a measurement system for high level language program statistics," *IEEE Trans. Software Eng.*, vol. C-31, pp. 883–891, Sept. 1982.

[18] M. Itakura and A. Takayanagi, "A model for estimating program size and its evaluation," in *Proc. Int. Conf. Software Eng.*, Sept. 1982, pp. 104–109.

[19] V. Basili and R. W. Selby, "Calculation and use of an environment's characteristic software metric set," in *Proc. Int. Conf. Software Eng.*, 1985, pp. 386–391.

H. F. Li (S'73–M'75) received the B.S. degree with highest honors from the University of California, Berkeley, in 1972, and remained there until 1975 when he completed the Ph.D. degree in the area of parallel-pipelined computer architecture.

Subsequently he joined the University of Illinois, Urbana-Champaign, as an Assistant Professor in the Computer Engineering Group, Department of Electrical Engineering. He returned to Hong Kong in 1977 and served as Lecturer and later, Senior Lecturer, in the Department of Electrical Engineering, University of Hong Kong, where he developed both an undergraduate and a graduate program in computer engineering. Currently he is an Associate Professor in the Department of Computer Science of Concordia University, Montreal, P.Q., Canada. His research interests cover the general areas of parallel and distributed processing, algorithms and architectures, and system specification.

W. K. Cheung, photograph and biography not available at the time of publication.