Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics

BILL CURTIS, MEMBER, IEEE, SYLVIA B. SHEPPARD, PHIL MILLIMAN, M. A. BORST, AND TOM LOVE

***Abstract*—Three software complexity measures (Halstead's E, McCabe's $v(G)$, and the length as measured by number of statements) were compared to programmer performance on two software maintenance tasks. In an experiment on understanding, length and $v(G)$ correlated with the percent of statements correctly recalled. In an experiment on modification, most significant correlations were obtained with metrics computed on modified rather than unmodified code. All three metrics correlated with both the accuracy of the modification and the time to completion. Relationships in both experiments occurred primarily in unstructured rather than structured code, and in code with no comments. The metrics were also most predictive of performance for less experienced programmers. Thus, these metrics appear to assess psychological complexity primarily where programming practices do not provide assistance in understanding the code.**

***Index Terms*—Commenting, complexity metrics, documentation, Halstead's E, human factors in software engineering, McCabe's $v(G)$, mnemonic variable names, modern programming practices, modifications, software science, structured programming.**

I. Introduction

CURRENTLY, more time is spent maintaining existing software than in developing new code. In fact, resources invested in maintenance have been estimated to be three times higher than those required during development [1]. Metrics computed from the initial code which could estimate either the reliability of modifications or the time required to implement them would prove invaluable to software managers who must allocate the time and resources necessary for software maintenance. Such metrics should relate both to the difficulty experienced by programmers in understanding a program and to the speed and accuracy with which they implement modifications.

Recently, several metrics have been developed to assess the complexity of programming tasks. For instance, in 1972 M. Halstead published his theory of software physics (renamed software science) stating that algorithms have measurable characteristics analogous to physical laws. According to Halstead [2]-[5] the amount of effort required to generate a program can be derived from simple counts of distinct operators and operands and the total frequencies of operators and operands. From these four quantities Halstead calculates the number of mental comparisons required to generate a program. Correlations often greater than 0.90 [5], [6] have been reported between Halstead's metrics and such measures of programmer performance as the number of bugs in a program [6]-[8], programming time [9], [10], and the quality of programs [11]-[13].

More recently, T. McCabe [14] developed a definition of complexity based on the decision structure of a program. McCabe's complexity metric is the classical graph theory cyclomatic number indicating the number of regions in a graph, or in the current usage, the number of linearly independent control paths comprising a program. Simply stated, McCabe's metric counts the number of basic control path segments which, when combined, will generate every possible path through the program. McCabe related his metric to the difficulty of testing a program, and presented it as a measure of computational complexity. Nevertheless, the number of basic control paths indexed by McCabe's metric may also relate to the mental difficulty of a programming task, since additional control paths could make a program more difficult to understand.

There is no exact mathematical relationship between the metrics developed by Halstead and McCabe, since no causal relationship exists between the number of operators, operands, and control paths. Yet, as the number of control paths increases there would be an anticipated increase in the number of operators, and a significant correlation between the Halstead and McCabe metrics across different programs would not be surprising.

In using these metrics, it is important to distinguish between the computational and psychological complexity of software, since the reasons for assessing them differ. *Computational complexity* refers to "the quantitative aspects of the solutions to computational problems" [15], such as comparing the efficiency of alternate algorithmic solutions. For example, as the number of distinct control paths through a program increases, the computational complexity may increase. *Psychological complexity* refers to characteristics of software which make it difficult to understand and work with. Thus, computational complexity assesses the difficulty of verifying an al-

Manuscript received August 18, 1978; revised November 20, 1978. This study was supported by the Office of Naval Research, Engineering Psychology Programs, under Contract N00014-C-77-0158. The views expressed in this paper, however, are not necessarily those of the Office of Naval Research or the Department of Defense. Expanded reports of each experiment can be obtained by writing B. Curtis at: General Electric Company, Suite 200, 1755 Jefferson Davis Highway, Arlington, VA 22202.

B. Curtis, S. B. Sheppard, P. Milliman, and M. A. Borst are with the Information Systems Programs, Space Division, General Electric Company, Arlington, VA 22202.

T. Love is with the Advanced Systems Development Group, Information Services Business Division, General Electric Company, Rockville, MD 20850.

gorithm's correctness, while psychological complexity assesses human performance on programming tasks. No simple relationship between computational and psychological complexity is expected. For example, a program with many control paths could be psychologically simple if any regularity existed in the program's branching process.

This report investigates the extent to which the Halstead and McCabe metrics assess the psychological complexity of understanding and modifying software. Primary emphasis is focused on Halstead's metric since it was proposed as an absolute measure of psychological complexity (i.e., number of mental discriminations). Although it was not formulated in psychological terms, McCabe's metric may prove to be a correlated measure of psychological complexity. Conditions which moderate relationships between these metrics and programmer performance will also be reported.

The two experiments reported are part of an ongoing research program investigating human factors in software engineering. These experiments were designed to investigate factors which influence two aspects of the software maintenance process; namely, understanding an existing program (Experiment 1) and accurately implementing modifications to it (Experiment 2). These factors included structured programming techniques, program documentation, and software complexity. While the first two factors were manipulated experimentally, no systematic attempt was made to manipulate software complexity. Manipulation of the first two factors, however, required preparing multiple versions of several programs. Since separate versions generated different values for the complexity metrics, it was possible to correlate complexity metrics with performance on controlled experimental tasks. Professional programmers were employed to provide the greatest possible external validity for the results [16].

II. Method

Participants

Each experiment involved 36 programmers from several General Electric locations. Participants in Experiment 1 had a working knowledge of Fortran and averaged 6.8 years of professional programming experience ($SD = 5.8$). In Experiment 2, the participants had a working knowledge of Fortran, averaged 5.9 years of professional programming experience ($SD = 4.0$), and had not participated in the previous experiment. The majority of participants possessed an engineering background.

Procedures

Introductory exercise: In both experiments, a packet of materials was prepared for each participant with written instructions on the experimental tasks. As a preliminary exercise, all participants were presented with the same short Fortran program (an implementation of Euclid's algorithm) and a brief description of its purpose. In Experiment 1, they studied this program for 10 min and were then given 15 min to reconstruct a functional equivalent from memory. In Experiment 2, participants were allowed unlimited time to complete a specified modification. This introductory program was intended to provide a common basis for comparing the skills of participants and to diminish learning effects prior to the experimental tasks. This latter point is important, since a pilot study [17] indicated that learning may occur during such tasks.

TABLE I
CONTROL STRUCTURES ALLOWED IN THREE TYPES OF CONTROL FLOW

	Programming conventions allowed		
Control structure	Structured	Naturally structured	Unstructured
DO	Forward exit only	Forward exit only	Expanded loop, forward and backward exits
IF	Logical only	Logical only	Logical and arithmetic
GO TO	Forward only	Forward and judicious use of backward	Unrestricted forward and backward
Computed GO TO	Forward only, single entry and exit	Unrestricted	Unrestricted
RETURN	Single	Multiple	Multiple

Experimental tasks: Following the initial exercise, participants were presented in turn with three separate programs comprising their experimental tasks. In Experiment 1, they were allowed 25 min to study each program, during which they were permitted to make notes or draw flowcharts. At the end of the study period, the original program and all scrap paper were collected. Each participant was then given 20 min to reconstruct a functional equivalent of the program from memory on a blank sheet of paper, but was not required to reproduce the comment section. In Experiment 2, a separate modification was indicated for each of the three programs and was described on a sheet accompanying the program listing. Participants were allowed to work at their own pace, taking as much time as needed to implement the modification. A 15 min break occurred before the last program was presented in each experiment.

Independent Variables

Program class: Three general classes of programs were used in Experiment 1: engineering, statistical, and nonnumerical. Three programs were chosen for each class from among many solicited from programmers at several locations. These nine programs varied from 36 to 57 statements and were considered representative of programs participants might actually encounter. All experimental programs were compiled and executed using appropriate test data. Experiment 2 used three of the nine programs from Experiment 1.

Complexity of control flow: Control flow structures at three levels of complexity were defined for each program in both experiments. Table I presents a summary of the constructs allowed under each type of control flow. The constructs allowed under structured control flow were generally consistent with the principles of structured programming described by Dijkstra [18].[1] When the rules for structured programming are applied

[1] Although this type of control flow is referred to as structured, the designation refers to standard Fortran IV and does not imply the use of a special compiler with IF-THEN-ELSE, DO-WHILE, or DO-UNTIL constructs. These structures were not available on the system used.

rigorously, awkward constructions may occur in standard Fortran, such as DO loops with dummy indices [19]. In a second version of each program, these awkward constructions were largely eliminated with a more naturally structured control flow. These conventions included multiple returns and judiciously used backward GO TO's. In the unstructured version of each program the control flow was not straightforward. Expanded DO loops, arithmetic IF's, and unrestricted use of GO TO's were allowed.

Variable name mnemonicity: In Experiment 1, three levels of mnemonicity for variable names were manipulated independently of program structure. Several nonparticipants were shown the programs and asked to assign names to the variables. The names chosen most frequently were used in the most mnemonic condition. The medium level consisted of less frequently chosen names. In the least mnemonic condition, names consisted of one or two alphanumeric characters.

Comments: Three levels of commenting were manipulated in Experiment 2: global, in-line, and none. Global comments appeared at the front of a program and provided both an overview of its function and a definition of the primary variables. In-line comments were interspersed throughout the program and described the specific functions of small sections of code.

Modifications: Three types of modifications were selected for each program in Experiment 2 as typical changes a programmer might be expected to implement. The level of difficulty for seven of the nine modifications increased with the number of new lines that had to be inserted to achieve a correct implementation, and the hardest modifications for each program required the most additional lines.

Experimental design: In order to control for individual differences in performance, a within-subjects, 3^4 factorial design was employed in each experiment [20]-[22]. In Experiment 1, three types of control flow were defined for each of nine programs, and each of these 27 versions was presented in three levels of variable mnemonicity, for a total of 81 programs. In Experiment 2, three levels of control flow were defined for each of the three programs. Each of these nine versions was presented in three levels of commenting. Modifications at three levels of difficulty were developed for each program, generating a total of 81 programs. The first 27 participants in each experiment exhausted the 81 separate programs and the final 9 participants repeated 27 of the previous experimental tasks. Programmers at each location were randomly assigned to experimental conditions, but in such a way that over the course of their three experimental programs, every participant had experienced each level of each independent variable. That is, they had worked with a program from each class, with each type of structure, with each type of commenting, etc. Each of the first 27 participants experienced unique combinations of the levels of independent variables across the three experimental tasks. The order of presentation of the three programs was assigned randomly to each participant.

Complexity Measures

Halstead's E: Halstead's effort metric (E) was computed precisely from a program [23] whose input was the source code listings of the 27 different versions of programs in each experiment. In Experiment 1, these 27 versions represented nine separate programs each written in three different types of control flow. In Experiment 2, the 27 versions represented three different modifications performed on three different programs, each of which was written in three different types of control flow. The computational formula was:

$$E = \frac{\eta_1 N_2 (N_1 + N_2) \log_2 (\eta_1 + \eta_2)}{2\eta_2}$$

where,

η_1 = number of unique operators,
η_2 = number of unique operands,
N_1 = total frequency of operators,
N_2 = total frequency of operands.

McCabe's $v(G)$: McCabe's metric is the classical graph-theory cyclomatic number defined as:

$$v(G) = \# \text{ edges} - \# \text{ nodes} + 2\ (\# \text{ connected components}).$$

McCabe presents two simpler methods of calculating $v(G)$. McCabe's $v(G)$ can be computed as the number of predicate nodes plus 1, or as the number of regions in a planar graph of the control flow.

Length: The length of the programs was computed as the total number of Fortran statements excluding comments.

Dependent Variables

Experiment 1: Current literature [24]-[26] suggests that the most sensitive measure of whether programmers understand a program is their ability to learn its structure and reproduce a functionally equivalent program without notes. Thus, the percent of statements correctly recalled became the dependent variable in Experiment 1. The criterion for scoring the reconstructed programs was the functional correctness of each separately reconstructed statement. Variable names and statement numbers which differed from those in the original progam were counted as correct when used consistently. Control structures could be different from the original program so long as the statements performed the same function.

Three judges scored each of the 108 reconstructed programs independently. Interjudge correlations of 0.96, 0.96, and 0.94 were obtained across the three sets of scores. The average of the judges' scores on each program (mean percent of statements correctly reconstructed) was used as the performance measure in Experiment 1.

Experiment 2: The dependent variables in Experiment 2 were the accuracy of the implemented modification and the time taken by the participant to perform the task. The individual steps necessary for correct implementation of each modification had been delineated in advance and assigned equal weights. That is, prototypes of each version of a program with each modification correctly implemented were established as the criteria against which participants' work would be compared. An accuracy score reflecting the percent of steps correctly implemented in each modification was computed by comparing each participant's changes with the criteria. All of the implemented modifications were scored by the same grader. The time to implement a modification was measured to the nearest minute by an electronic timer. Thus, the performance measures were the percent of changes correctly

TABLE II
MEANS, STANDARD DEVIATIONS, AND INTERCORRELATIONS FOR SOFTWARE COMPLEXITY MEASURES

Metric	Mean	Standard deviation	Correlations E	Correlations v(G)
Experiment 1 (n = 27)				
Halstead's E	80.9	49.0		
McCabe's v(G)	14.0	5.7	.84***	
Length	46.0	6.0	.47**	.64***
Experiment 2				
Unmodified (n = 9)				
Halstead's E	65.4	36.6		
McCabe's v(G)	13.0	4.3	.85**	
Length	41.9	7.3	.97***	.90***
Modified (n = 27)				
Halstead's E	62.9	35.5		
McCabe's v(G)	12.1	4.0	.88***	
Length	42.4	8.3	.92***	.89***

Note: Halstead E values are reported in thousands of mental discriminations.

*$p \leq .05$
**$p \leq .01$
***$p \leq .001$

implemented to a program and the number of minutes required to complete them.

Analyses

In the initial set of analyses, the three measures of software complexity were related to the different dependent variables using the Pearson product-moment correlation coefficient. In the second phase of analysis, the previous analyses were reconducted on data generated at each level of an independent variable and for participants at different levels of experience. These analyses were conducted to determine whether structured programming, documentation, or experience moderated the relationship between complexity measures and performance.

III. RESULTS

Experimental Manipulations

Experiment 1: A complete report of the results of the experimental manipulations in both experiments is presented elsewhere [27]. Briefly, a mean of 51 percent of the statements were correctly recalled across all experimental tasks ($SD = 25$ percent). Substantial differences in performance were observed across the nine programs. Significant differences in performance were also found across the three types of control flow. Performance on naturally structured programs was superior to that on unstructured programs. Differences in the mnemonicity of variable names had no effect on performance.

Experiment 2: An average accuracy score of 62 percent was achieved over all implemented modifications to the experimental programs ($SD = 31$ percent). The 108 accuracy scores ranged from five scores of 0 percent to 24 scores of 100 percent, and were negatively skewed. The average time to complete the modifications was 17.9 min ($SD = 11.4$), ranging from 2 to 59 min with a positive skew. Accuracy and time were uncorrelated.

On two of the three programs employed, the accuracy of the implementations was modestly affected by the difficulty of the modification and the use of structured programming techniques. More accurate modifications were implemented to strictly structured rather than unstructured programs. More difficult modifications took longer to implement, although the use of structured programming techniques did not reduce the time to completion. No effect on accuracy or time was observed for type of comments.

Distributional Information on Complexity Measures

Means, standard deviations, and intercorrelations for Halstead's E, McCabe's $v(G)$, and length from programs in both experiments are presented in Table II.[2] The average number of Fortran statements in Experiment 1 was 46 and in Experi-

[2]In this and subsequent tables, n indicates the number of data points for each variable over which the correlations were computed, not the number of participants. The levels of significance for correlations are reported in terms of p values which indicate the proportion of times a correlation of this magnitude could be expected to be obtained by chance in a sample of given size when the correlation in the underlying population was zero. It is traditional in research with human participants to accept a correlation as significant if it is large enough to have occurred no more than once in 20 times by chance. Asterisks identify significant correlations and the severity of the significance test they could satisfy.

TABLE III
CORRELATIONS OF COMPLEXITY METRICS WITH PERCENT OF STATEMENTS CORRECTLY RECALLED IN EXPERIMENT 1

	Correlations		
Criterion	E	v(G)	Length
Unaggregated data (n = 108)	-.19*	-.34***	-.47***
Aggregated data (n = 27)	-.13	-.35*	-.53**
Exceptional group removed (n = 24)	-.36*	-.55**	-.61***
Transformed scores (n = 27)	-.10	-.24	-.38*
Exceptional group removed and transformed scores (n = 24)	-.73***	-.21	-.65***

$^*p \leq .05$
$^{**}p \leq .01$
$^{***}p \leq .001$

ment 2 was approximately 42 for both modified and unmodified programs (some modifications involved deletions or substitutions). The programs in Experiment 1 consisted of an average of 14 basic control path segments, while the average for unmodified programs in Experiment 2 was 13 segments. The only apparent difference in the average score for complexity measures between the two experiments occurred on Halstead's E. In Experiment 1, there was an average of approximately 80 900 mental discriminations per program (influenced by an extreme score of 250 000 on one program), while in Experiment 2, E averaged only 65 400 mental discriminations on unmodified programs and 62 900 on modified programs.

Substantial intercorrelations were observed among the complexity metrics in both experiments. In Experiment 1, the Halstead and McCabe metrics were strongly related, while their correlations with length were moderate. In Experiment 2, however, all three measures were strongly correlated on both the unmodified and modified programs.

Correlations with Performance

Experiment 1: Since different levels of variable mnemonicity and type of commenting neither affected performance, nor caused any change in the value of the complexity metrics for a particular program, some of the data reported in sections on performance predictions were aggregated over the three levels of mnemonicity in Experiment 1 and three types of commenting in Experiment 2. Thus, when analyses are reported for 27 data points, each datum represents the average of at least three performance scores.

The correlations between percent of statements correctly recalled and complexity metrics for both aggregated and unaggregated data are presented in Table III. These correlations were all negative, indicating that fewer lines were correctly recalled as the level of complexity represented by these three measures increased. Little difference was observed between the correlations in the aggregated and unaggregated data. Length and McCabe's $v(G)$ were moderately related to performance, while little relationship was found for Halstead's E.

Fig. 1 presents the scatterplot of Halstead's E with performance; data collected on different versions of the same program are united by lines. There were three data points (circled in Fig. 1) which were developed by averaging across three participants who consistently outscored others on both the pretest and the experimental tasks. It was as likely that the high scores on these three data points resulted from the failure of random assignment to separate these participants into different groups as from factors inherent in these three versions of the experimental programs. With the three data points of the exceptional group removed, the correlations for all three complexity metrics improved (third row of Table III).

The considerable differences in difficulty among programs evident in the experimental results were also apparent in Fig. 1. As a heuristic device to determine whether the complexity metrics were more predictive of performance within programs than across them, a transformation was applied separately to the data for each metric in an attempt to establish similar baselines for each program. The lowest value obtained for a metric among the three versions of each of the nine separate programs was set to zero. This lowest value was also subtracted from the values of the metric on the other two versions. Similarly, the percent of statements correctly recalled for the version with the lowest value of the metric was also set to zero, and this percentage was subtracted from the percentage scores on the other two versions. Correlations on transformed data indicated whether performance diminished as a function of increasing complexity when initial differences in difficulty and complexity among programs had been removed.

Correlations between performance and each of the transformed metrics with the data from the exceptional group removed are presented in the last row of Table III. Although Halstead's E was unrelated to performance in the raw data, a strong correlation was observed after corrections were made for differences among programs and participants. Such an improvement was not observed in the results for $v(G)$ or length.

Experiment 2: Correlations between the three complexity metrics and the two dependent variables in Experiment 2 are shown in Table IV. The correlations computed from the aggregated data were slightly larger than those computed from the unaggregated data. The complexity metrics were generally more strongly correlated with time to completion than with the accuracy of the implementation, especially on the modified programs.

The largest number of significant correlations were observed for metrics computed from the modified programs.[3] The metrics were moderately related to both criteria on modified programs. The only significant correlations between performance measures and metrics computed from the unmodified

[3]All values for complexity metrics for modified programs were computed on the prototype programs with correct implementations rather than from code generated by the participants. Had the metrics been computed on the participants' code, they would be measuring the complexity of the participants' understanding of the modified algorithm. Computing metrics on the participants' code should tend to inflate the correlations with performance measures.

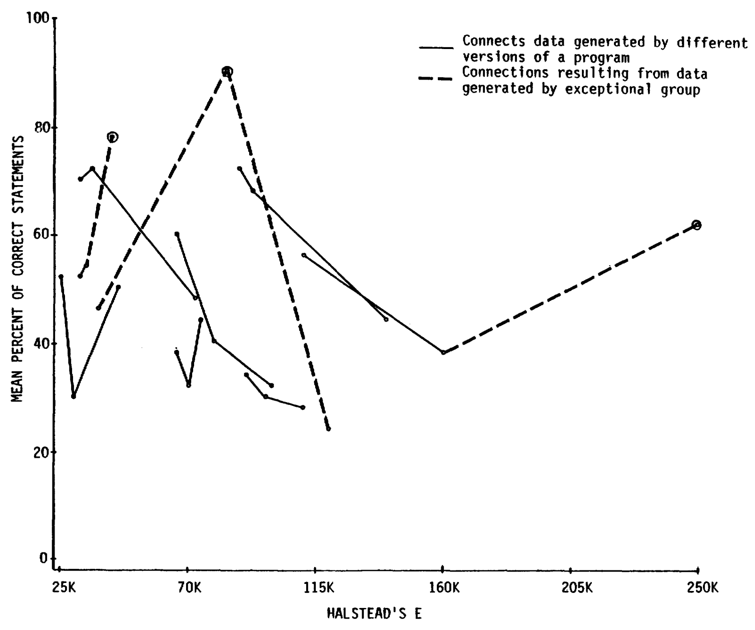


Fig. 1. Scatterplot of Halstead's E with mean percent of correct statements in Experiment 1.

TABLE IV
CORRELATIONS OF COMPLEXITY METRICS WITH ACCURACY AND TIME IN EXPERIMENT 2

Criterion	Correlations E	v(G)	Length
Unaggregated (n = 108)			
Accuracy			
Unmodified	-.12	-.21*	-.17*
Modified	-.17*	-.21*	-.20*
Time to completion			
Unmodified	.16*	.15	.13
Modified	.28**	.24**	.30***
Aggregated (n = 27)			
Accuracy			
Unmodified	-.21	-.36*	-.28*
Modified	-.29	-.36*	-.34*
Time to completion			
Unmodified	.25	.23	.20
Modified	.44**	.38*	.46**

*p ≤ .05
**p ≤ .01
***p ≤ .001

TABLE V
CORRELATIONS BETWEEN PERFORMANCE MEASURES AND COMPLEXITY METRICS UNDER DIFFERENT TYPES OF CONTROL FLOW

Criterion and type of control flow	Correlations E	v(G)	Length
Experiment 1 (n = 36)			
% recalled			
Unstructured	-.45***	-.55***	-.68***
Naturally structured	.07	-.08	-.20
Structured	-.01	-.11	-.51***
Experiment 2 (n = 36)			
Time to completion			
Unstructured	.38*	.24	.37*
Naturally structured	.28*	.20	.34*
Structured	.08	.21	.12

*p ≤ .05
**p ≤ .01
***p ≤ .001

programs represented a tendency for higher McCabe values and greater lengths to be associated with lower accuracy scores.

Moderator Effects

In order to determine the effects of possible moderators, correlations between performance measures and software complexity metrics were computed under different types of control flow and commenting, and at different levels of programmer experience. These analyses were conducted on unaggregated data in both experiments, and on modified programs in Experiment 2.

Correlations between performance measures and software complexity metrics under different types of control flow are presented in Table V. In Experiment 1, Halstead's E and McCabe's $v(G)$ correlated significantly with performance only on unstructured programs. While a similar pattern of correlations emerged in Experiment 2 between Halstead's E and time to complete the modification, differences among these corre-

TABLE VI
CORRELATIONS BETWEEN PERFORMANCE MEASURES AND COMPLEXITY METRICS UNDER DIFFERENT TYPES OF COMMENTING IN EXPERIMENT 2

	Correlations		
Criterion and type of comments	E	v(G)	Length
Accuracy (n = 36)			
None	-.34*	-.35*	-.37*
Global	-.18	-.31*	-.23
In-line	.03	.04	.03
Time (n = 36)			
None	.47**	.44**	.55***
Global	.21	.18	.18
In-line	.16	.11	.16

*$p \leq .05$
**$p \leq .01$
***$p \leq .001$

TABLE VII
CORRELATIONS BETWEEN PERFORMANCE MEASURES AND COMPLEXITY METRICS AMONG PROGRAMMERS DIFFERING IN EXPERIENCE

	Correlations		
Criterion and level of experience	E	v(G)	Length
Experiment 1			
% recalled			
≤ 3 years (n = 42)	-.35*	-.47***	-.55***
> 3 years (n = 60)	-.03	-.18	-.31**
Experiment 2			
Time to completion			
≤ 3 years (n = 32)	.55***	.52***	.56***
> 3 years (n = 75)	.20*	.15	.22*

Note: Two participants in Experiment 1 did not report their years of experience

*$p \leq .05$
**$p \leq .01$
***$p \leq .001$

lations were not significant. The moderating effects did not appear to result from restrictions of range on the variables involved. That is, means and variances for complexity metrics were identical across types of control flow, and although mean performance scores differed across types of control flow, no significant differences were observed in variances.

Significant moderating effects were also found for types of commenting. Table VI presents correlations between performance measures and complexity metrics for data generated under each type of commenting in Experiment 2. All but one of the significant correlations occurred when no comments were included in the code. Differences in correlations between in-line and no-comment conditions either achieved or bordered on significance in all cases. There were no significant differences in means and variances among performance measures or complexity metrics across types of commenting.

Finally, relationships between complexity metrics and performance measures were moderated by the participants' years of professional programming experience. The dividing point between three or fewer years and more than three years experience was arbitrary and represented a compromise between minimizing the years of experience in the less experienced category and having sufficient participants for a correlational analysis. As is evident in Table VII, the complexity metrics were more strongly related to performance among less experienced programmers in all cases. Differences among correlations with time to completion between the two groups all achieved significance. Neither means nor variances differed significantly for performance measures or complexity metrics among the two experience levels.

IV. DISCUSSION

The two experiments comprising this study produced empirical evidence that software complexity metrics were related to the difficulty programmers experienced in understanding and modifying software. The correlations observed in this study, however, were not as high as those reported by Halstead [5] in other verifications of his theory. While the correlations reported here will not seem large to perusers of the engineering or physical science literature, their magnitudes are typical of significant results reported in human factors research.

Stronger relationships may have been obscured by variation in performance scores related to differences among participants and programs which were enhanced by the economical multifactor designs employed in these experiments. When data in the first experiment were transformed in an attempt to reduce these differences, Halstead's E predicted performance substantially better than it had in the untransformed data. Further, uniformity in the sizes of programs employed may also have limited these results. The range of values assumed by complexity metrics may not have been sufficient to allow correlational tests [28] to detect the strong relationships that have been reported in other verifications of these theories. Studies reporting higher correlations for Halstead's E usually involved a broader range of program sizes [5], [6].

Differences among programs played an important role in these experiments. The Halstead and McCabe metrics provided some information about program differences, but there were other factors unassessed by these metrics which influence the psychological complexity of the programs. The metrics predicted programmer performance better on versions of programs which were unstructured or uncommented. By reducing the cognitive load on a programmer, information available from structured code and comments altered the psychological complexity of a program such that it was no longer accurately reflected by the complexity metrics. Further, neither Halstead's nor McCabe's metrics consider the level of nesting within various constructions (e.g., three DO loops in succession will result in metric values similar to those for three nested DO loops). However, nesting may influence psychological complexity.

The number of statements in the program proved to be as strongly related to performance on the experimental tasks as the Halstead and McCabe metrics, contrary to results obtained by Gordon [29]. Correlations between the metrics and length suggested substantial overlap in the constructs they measured on the modular-sized programs studied. In these experiments, the complexity metrics and length appeared to provide three different methods of quantifying an underlying construct of program volume. Had these correlations been smaller, the predictive worth of the metrics could have been better compared. The number of statements may not be as highly correlated with the Halstead and McCabe metrics for programs of substantially greater length, written in languages other than Fortran, or implementing other types of applications.

Much of the research on Halstead's software science [5] or related theories have predicted performance criteria from metrics computed on programs in final form. Thus, it was not surprising that correlations with performance criteria in Experiment 2 were usually higher for metrics computed on modified rather than unmodified code. These results suggest that the complexity metrics provide information for projecting manpower needs and costs primarily when their values on accurately modified programs can be estimated prior to the implementation. The difficulty of this task does not detract from the value of using the metrics as feedback tools to programmers concerning the complexity of their code [30].

A distinguishing characteristic of psychological complexity is the interaction between program characteristics and individual differences, such as programming experience. Chrysler [31] demonstrated the value of experiential variables in predicting the time to complete a programming task. In the current study, the complexity metrics were more highly related to the performance of less experienced programmers. Thus, the complexity metrics may not represent the most important constructs for predicting the performance of experienced programmers. These programmers probably conceptualized programs at a level other than that of operators, operands, and basic control paths. They may have fit the program into a schema they recognized from previous experience, just as chess players learn to visualize the gameboard differently with experience. Results did not indicate, however, that more experienced programmers performed the tasks more efficiently. Since the ratio comparing good to poor programmer performance is often as great as 28 to 1 [32], the selection, training, and placement of programmers can have a significant impact on project costs and product quality.

Halstead's software science has provided some important initial directions for investigating psychological complexity. Subsequent work by McCabe [14] and Laemmel and Shooman [33] has also proven relevant to this purpose. Yet, assessing the psychological complexity of software appears to require more than a simple count of operators, operands, and basic control paths. If the ability of complexity metrics to predict programmer performance is to be improved, then metrics must also incorporate measures of phenomena related by psychological principles to the memory, information processing, and problem solving capacities of programmers. In identifying a set of psychological principles relevant to programming tasks, it will be important to determine methods for quantifying factors in the code which represent information concerning the program. This approach might not only generate improved metrics for assessing psychological complexity, but might also identify programming practices which could lead to simplified, more easily maintained software.

ACKNOWLEDGMENT

The authors gratefully acknowledge the assistance of A. Fitzsimmons in implementing this research, of Dr. G. Hahn in developing the experimental design, and of B. Day in preparing the final report. Careful reviews of this report by Dr. J. O'Hare, Dr. M. Halstead, and T. McCabe have resulted in substantial improvements. The support and encouragement of G. Dwyer has also been greatly appreciated.

REFERENCES

[1] L. H. Putnam, "Measurement data to support sizing, estimating and control of the software life cycle," in *Proc. COMPCON '78*, IEEE, New York, 1978.
[2] M. H. Halstead, "Natural laws controlling algorithm structure," *SIGPLAN Notices*, vol. 7, 1972, pp. 19–26.
[3] —, "A theoretical relationship between mental work and machine language programming," Comput. Sci. Dep., Purdue University, Tech. Rep. CSD-TR-67, 1972.
[4] —, "Software physics: Basic principles," Thomas J. Watson Res. Center, IBM, Yorktown Heights, NY, Tech. Rep. RJ-1582, 1975.
[5] —, *Elements of Software Science.* New York: Elsevier, 1977.
[6] A. B. Fitzsimmons and L. T. Love, "A review and evaluation of software science," *ACM Computing Surveys*, vol. 10, pp. 3–18, 1978.
[7] Y. Funami and M. H. Halstead, "A software physics analysis of Akiyama's debugging data," Comput. Sci. Dep., Purdue University, Tech. Rep. CSD-TR-144, 1975.
[8] L. Cornell and M. H. Halstead, "Predicting the number of bugs expected in a program module," Comput. Sci. Dep., Purdue University, Tech. Rep. CSD-TR-205, 1976.
[9] M. H. Halstead, "Using the methodology of natural science to understand software," Comput. Sci. Dep., Purdue University, Tech. Rep. CSD-TR-190, 1976.
[10] R. D. Gordon and M. H. Halstead, "An experiment comparing FORTRAN programming times with the software physics hypothesis," Comput. Sci. Dep., Purdue University, Tech. Rep. CSD-TR-167, 1975.
[11] M. H. Halstead, "An experimental determination of the 'purity' of a trivial algorithm," Comput. Sci. Dep., Purdue University, Tech. Rep. CSD-TR-73, 1972.
[12] N. Bulut and M. H. Halstead, "Impurities found in algorithm implementation," Comput. Sci. Dep., Purdue University, Tech. Rep. CSD-TR-111, 1974.
[13] J. L. Elshoff, "Measuring commercial PL/1 programs using Halstead's criteria," *SIGPLAN Notices*, vol. 11, pp. 38–46, 1976.
[14] T. J. McCabe, "A complexity measure," *IEEE Trans. Software Eng.*, vol. SE-2, pp. 308-320, Dec. 1976.
[15] M. O. Rabin, "Complexity of computations," *Commun. Ass. Comput. Mach.*, vol. 20, pp. 625–633, 1977.
[16] D. Campbell and J. C. Stanley, *Experimental and Quasi-Experimental Designs for Research.* Chicago: Rand-McNally, 1976.
[17] S. B. Sheppard and L. T. Love, "A preliminary experiment to test influences on human understanding of software," in *Proc. 21st Annu. Meeting of the Human Factors Society*, vol. 21, 1977, pp. 167–171.
[18] E. W. Dijkstra, "Notes on structured programming," in *Structured Programming*, O. J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, Ed. New York: Academic, 1972.

[19] T. Tenny, "Structured programming in FORTRAN," *Datamation*, vol. 20, pp. 110-115, 1974.

[20] G. J. Hahn and S. S. Shapiro, "A catalogue and computer program for the design and analysis of orthogonal symmetric and asymmetric fractional factorial experiments," Corporate Res. Dev., General Electric Company, Schenectady, NY, Tech. Rep. 66-C-165, 1966.

[21] R. E. Kirk, *Experimental Design Procedures for the Behavioral Sciences.* San Francisco: Freeman, 1968.

[22] G. E. P. Box, W. G. Hunter, and J. S. Hunter, *Statistics for Experiments.* New York: Wiley, 1978.

[23] K. J. Ottenstein, "A program to count operators and operands for ANSI-FORTRAN modules," Comput. Sci. Dep., Purdue University, Tech. Rep. CSD-TR-196, 1976.

[24] L. T. Love, "Relating individual differences in computer programming performance to human information processing abilities," Ph.D. dissertation, Dep. Psychology, Univ. Washington, 1977.

[25] B. Shneiderman, "Measuring computer program quality and comprehension," *Int. J. Man-Machine Studies*, vol. 9, pp. 465-478, 1977.

[26] —, "Human factors experiments for developing quality software," in *The Infotech State of the Art Report on Software Engineering.* Berkshire, England: Infotech International, Ltd., 1977.

[27] S. B. Sheppard, M. A. Borst, B. Curtis, and T. Love, "Factors influencing the understandability and modifiability of computer programs," Inf. Syst. Programs, General Electric Company, Arlington, VA, 1978.

[28] A. L. Edwards, *An Introduction to Linear Regression and Correlation.* San Francisco: Freeman, 1976.

[29] R. D. Gordon, "A measure of mental effort related to program clarity," Ph.D. dissertation, Comput. Sci. Dep., Purdue University, 1977.

[30] J. L. Elshoff, "A review of software measurement studies at General Motors Research Laboratories," in *Proc. 2nd Software Life Cycle Management Workshop*, IEEE, NY, 1978.

[31] E. Chrysler, "Some basic determinants of computer programming productivity," *Commun. Ass. Comput. Mach.*, vol. 21, pp. 472-483, 1978.

[32] H. Sackman, W. J. Erickson, and E. E. Grant, "Exploratory experimental studies comparing on-line and off-line programming performance," *Commun. Ass. Comput. Mach.*, vol. 11, pp. 3-11, 1968.

[33] A. Laemmel and M. Shooman, "Software modeling studies: Statistical (natural) language theory and computer program complexity," Rome Air Development Center, Griffiss AFB, Tech. Rep. RADC-TR-78-4, vol. II, 1978.

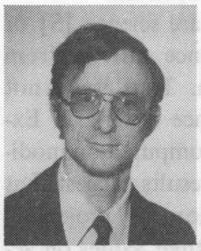
Bill Curtis (M'79) was born in Meridian, TX, in 1948. He received the B.A. degree in psychology, mathematics, and drama from Eckerd College, St. Petersburg, FL, in 1971, the M.A. degree in clinical psychology from the University of Texas, Austin, in 1974, and the Ph.D. degree in industrial psychology and statistics from Texas Christian University, Ft. Worth, in 1975.

From 1975 to 1977 he was a Research Assistant Professor in the Organizational Research Group at the University of Washington, where he taught statistics in the Department of Psychology. His research during this period involved sports psychology and the development of leadership training programs. During 1977 he was the Staff Psychologist for Weyerhaeuser Company where his work involved the development of a performance appraisal system, employee selection tests, and employee attitude surveys. In 1978 he joined Information Systems Programs in the Space Division of the General Electric Company, Arlington, VA, where he is currently Manager of the Software Management Research Unit. His current research involves human factors in software engineering, project management in software development, and optimal design strategies for large systems.

Sylvia B. Sheppard received the B.A. degree in mathematics from Rutgers University, New Brunswick, NJ, in 1958.

From 1958 to 1962, she was an Associate Member of the Technical Staff in the Human Factors Research Department, Bell Laboratories, Murray Hill, NJ. She was involved in research on human factors and information theory. From 1977 to the present, she has been an Associate Research Scientist with Information Systems Programs in the Space Division of the General Electric Company, Arlington, VA. Her research interests involve human factors in software engineering and design strategies for large software systems. Recently, she has pursued graduate coursework in computer science and information systems management at the University of Maryland, College Park.

Ms. Sheppard is a member of Phi Beta Kappa.



Phil Milliman received the B.S. degree in psychology from the University of Washington, Seattle, in 1973.

From 1973 to 1974, he was Programmer with Consolidated Electronics, Inc. From 1975 to 1978, he was a Programmer Analyst in the Computer Center at Anderson College. In 1978, he joined Information Systems Programs in the Space Division of the General Electric Company, Arlington, VA, where he is an Associate Research Scientist. His current research interests involve human factors in software engineering, the effects of modern programming practices on software development projects, and design strategies for large software systems. He is currently pursuing graduate coursework in computer science at George Washington University, Washington, DC.

M. A. Borst received the B.S. degree in psychology from the University of Washington, Seattle, in 1974.

From 1977 to the present she has been an Associate Research Scientist with Information Systems Programs in the Space Division of the General Electric Company, Arlington, VA. Her research interests have been in cognitive psychology and human factors in software engineering.

Ms. Borst is a member of Phi Beta Kappa.

Tom Love received the B.S. degree in psychology and mathematics from the University of Alabama, Birmingham, in 1970, and the M.S. and Ph.D. degrees in cognitive psychology from the University of Washington, Seattle, in 1973 and 1977, respectively.

While at the University of Washington he received a National Institute of Mental Health Research Fellowship and was a Research Assistant at the Battelle Human Affairs Research Center.

In 1976 he joined Information Systems Programs in the Space Division of the General Electric Company, Arlington, VA, as a Senior Software Engineer. In 1977 he became Manager of the Software Psychology Research Unit and obtained research contracts involving human factors in software engineering and the effects of modern programming practices on software development projects. In 1978 he joined the Advanced Systems Development Group in the Information Services Business Division of General Electric where he is currently involved in designing a comprehensive environment for software development.