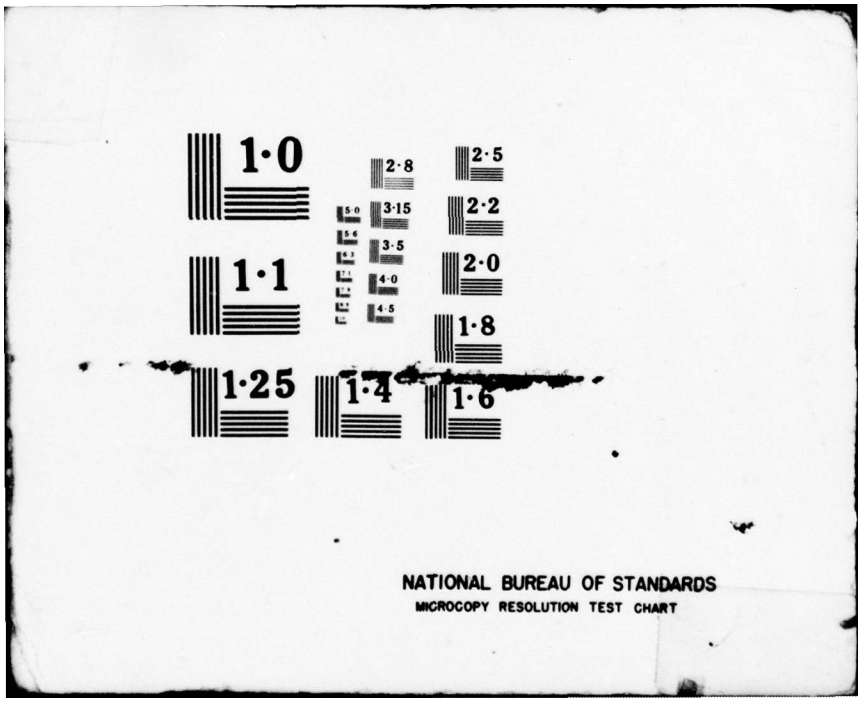
AD-A068 196 GENERAL ELECTRIC CO ARLINGTON VA F/G 9/2
FACTORS AFFECTING PROGRAMMER PERFORMANCE IN A DEBUGGING TASK.(U)
FEB 79 S B SHEPPARD, P MILLIMAN, B CURTIS N00014-77-C-0158
UNCLASSIFIED TR-79-388100-5 NL

1 OF 1
ADA
068196

END
DATE FILMED
6-79
DDC


1·0
1·1
1·25
1·4
1·6
1·8
2·0
2·2
2·5
2·8
3·15
3·5
4·0
4·5
5·0
NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

(12) LEVEL II

TR-79-388100-5

FACTORS AFFECTING PROGRAMMER PERFORMANCE IN A DEBUGGING TASK

by

Sylvia B. Sheppard,
Phil Milliman,
and
Bill Curtis

February 1979

SOFTWARE MANAGEMENT RESEARCH
GENERAL ELECTRIC
INFORMATION SYSTEMS PROGRAMS
ARLINGTON, VIRGINIA

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: TR-79-388100-5	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): Factors Affecting Programmer Performance in a Debugging Task.	5. TYPE OF REPORT & PERIOD COVERED: Technical Report
6. PERFORMING ORG. REPORT NUMBER: 388100-5	
7. AUTHOR(s): S.B. Sheppard, B. Curtis, P. Milliman	8. CONTRACT OR GRANT NUMBER(s): N0014-77-C-0158
9. PERFORMING ORGANIZATION NAME AND ADDRESS: General Electric Company, 1755 Jefferson Davis Highway, Arlington, VA 22202	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: NR197-037
11. CONTROLLING OFFICE NAME AND ADDRESS: Office of Naval Research, 800 North Quincy Street, Arlington, VA 22217	12. REPORT DATE: 2/19/79
	13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited. Reproduction in whole or in part is permitted for any purpose of the U.S. Government.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

This research was supported by Engineering Psychology Programs, Office of Naval Research.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Debugging, Structured Programming, Software Psychology Metrics, Control Flow Complexity, Modern Programming Practices, Program Errors.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report is the third in a series investigating characteristics of software which are related to its psychological complexity. Three independent variables, length of program, complexity of control flow, and type of error, were evaluated for three different Fortran programs in a debugging task. Fifty-four experienced programmers were asked to locate a single → (over)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

409 446

(cont)

bug in each of three programs. Documentation consisted of input files, correct output, and erroneous output. Performance was measured by the time to locate and successfully correct the bug.

Small but significant differences in time to locate the bug were related to differences among programs and presentation order. Although there was no main effect for type of bug, there was a large program by error interaction suggesting the existence of context effects. Among measures of software complexity, Halstead's 'E' proved to be the best predictor of performance followed by McCabe's 'v(G)' and the number of lines of code.

Number of programming languages known and familiarity with certain programming concepts also predicted performance. As in the previous experiments, experiential factors were better predictors for those participants with three or fewer years experience programming in Fortran.

A

1473B

TR-79-388100-5

FACTORS AFFECTING PROGRAMMER PERFORMANCE IN A DEBUGGING TASK

by

Sylvia B. Sheppard,
Phil Milliman,
and
Bill Curtis

Software Management Research
Information Systems Programs
General Electric Company
1755 Jefferson Davis Highway
Arlington, VA 22202

Submitted to:

Office of Naval Research
Engineering Psychology Programs
Arlington, VA 22217

Contract: #N00014-77-C-0158
Work Unit: NR 197-037

February 1979

Software Complexity Research Program

Department of Defense (DOD) software production and maintenance is a large, poorly understood, and inefficient process. Recently Frost and Sullivan (The Military Software Market, 1977) estimated the yearly cost for software within DOD to be as large as $9 billion. DeRoze (1977) has also estimated that 115 major defense systems depend on software for their success. In an effort to find near-term solutions to software related problems, the DOD has begun to support research into the software production process.

A formal 5 year R&D plan (Carlson & DeRoze, 1977) related to the management and control of computer resources was recently written in response to DOD Directive 5000.29. This plan requested research leading to the identification and validation of metrics for software quality. The study described in this paper represents an experimental investigation of such metrics and is part of a larger research program seeking to provide valuable information about the psychological and human resource aspects of the 5 year plan.

DOD is also initiating the development of a more powerful, higher order language for general use by all services (Department of Defense, 1977). With a language-independent measure of the complexity of software, we can evaluate not only program A versus program B, but also the individual constructs of a language (cf. Gordon, 1977). Thus, an objective, quantitative theory based on sound experimental data can replace idiosyncratic, subjective evaluations of the psychological complexity of software. Long term benefits of this effort involve improved software system reliability and reduced development and maintenance costs.

The challenge undertaken in this research program is to quantify the psychological complexity of software. It is important

to distinguish clearly between the psychological and computational complexity of software. Computational complexity refers to characteristics of algorithms or programs which make their proof of correctness difficult, lengthy, or impossible. For example, as the number of distinct paths through a program increases, the computational complexity also increases. Psychological complexity refers to those characteristics of software which make human understanding of software more difficult. No direct linear relationship between computational and psychological complexity is expected. A program with many control paths may not be psychologically complex. Any regularity to the branching process within a program may be used by a programmer to simplify understanding of the program.

Halstead (1977) has recently developed a theory concerned with the psychological aspects of computer programming. His theory provides objective estimates of the effort and time required to generate a program, the effort required to understand a program, and the number of bugs in a particular program (Fitzsimmons & Love, 1978). Some predictions of the theory are counterintuitive and contradict some results of previous psychological research. The theory has attracted attention because independent tests of hypotheses derived from it have proven amazingly accurate.

Although predictions of programmer behavior have been particularly impressive, much of the research testing Halstead's theory has been performed without sufficient experimental or statistical controls. Further, much of the data were based upon imprecise estimating techniques. Nevertheless, the available evidence has been sufficient to justify a rigorous evaluation of the theory.

Rather than initiate a research program designed specifically to test the theory of software science, a research strategy was

chosen which would generate suggestions for improving programmer efficiency regardless of the success of any particular theory. This research has focused on four phases of the software life-cycle: understanding, modification, debugging, and construction. Since different cognitive processes are assumed to predominate in each phase, no single experiment or set of experiments on a particular phase would provide a sufficient basis for making broad recommendations for improving programmer efficiency. Each experiment in the series comprising this research program has been designed to test important variables assumed to effect a particular phase of software development. Professional programmers have been used in these experiments to provide the greatest possible external validity for the results (Campbell & Stanley, 1966). In addition, Halstead's theory of software science and other related metrics have been evaluated with these data.

ABSTRACT

This report is the third in a series investigating characteristics of software which are related to its psychological complexity. Three independent variables, length of program, complexity of control flow, and type of error, were evaluated for three different Fortran programs in a debugging task. Fifty-four experienced programmers were asked to locate a single bug in each of three programs. Documentation consisted of input files, correct output, and erroneous output. Performance was measured by the time to locate and successfully correct the bug.

Small but significant differences in time to locate the bug were related to differences among programs and presentation order. Although there was no main effect for type of bug, there was a large program by error interaction suggesting the existence of context effects. Among measures of software complexity, Halstead's E proved to be the best predictor of performance, followed by McCabe's v(G) and the number of lines of code.

Number of programming languages known and familiarity with certain programming concepts also predicted performance. As in the previous experiments, experiential factors were better predictors for those participants with three or fewer years experience programming in Fortran.

TABLE OF CONTENTS

INTRODUCTION

Debugging programs is one of the most expensive, time-consuming activities in the development of a software system. Only a few laboratory experiments have investigated the relative difficulty of locating different types of bugs or the most effective search strategies. Youngs (1974) found that experience contributed to differences among types of errors made in a construction experiment. Wescourt and Hemphill (1978) described a model of the debugging process, but the model was not entirely supported by the available data. Gould and his associates (Gould and Drongowski, 1974; Gould, 1975) found that the type of bug influenced debugging performance on short programs. Specifically, assignment bugs were more difficult to locate than array or iteration bugs, probably because the former required a greater understanding of the algorithm used by the program.

The difficulty of debugging a program may be associated with coding practices used during its development. One factor which may influence the ease of finding a bug is the complexity of a program's control flow. Two previous experiments by the authors investigated the effects of structured control flow in understanding and modification tasks (Sheppard, Curtis, Borst, Milliman, & Love, 1979). Programmers performed their tasks more efficiently on code which exhibited a straightforward, top-down control flow than on an unstructured, convoluted control flow. A rigorously structured control flow (Dijkstra, 1972) did not produce significantly better performance than a naturally structured version which allowed limited unstructured constructs (e.g., exits from loops). Thus the overall top-down quality of the control flow appears to influence performance, while minor deviations from the tenets of structured code do not appear to influence performance significantly. This result

may reflect the innate awkwardness of implementing strictly structured code in standard Fortran.

Factors other than the structuredness of the control flow may influence the complexity of a computer program and, thus, the difficulty programmers experience in performing their tasks. Some of these factors have been quantified in the software complexity metrics developed by Halstead (1977) and McCabe (1976). Halstead's metric purportedly represents the number of mental discriminations involved in developing a program, while McCabe's metric measures the number of elementary control path segments comprising a program. In experiments on understanding and modification, these software complexity metrics were evaluated for their usefulness as predictors of programmer performance (Curtis, Sheppard, Milliman, Borst, & Love, 1979). The results observed in those experiments were modest. The correlations in the raw data were not large, and the number of lines of code usually predicted programmer performance better than the Halstead or McCabe metrics. Several limitations in the experimental procedures employed to obtain the data may have produced these results. First, all of the programs studied were short (35-55 lines of code). The limited range of metric values calculated on programs of this length may not have been sufficient for an adequate test of the predictive worth of the metrics. Second, individual differences among programmers exerted significant effects on the results obtained. When the data from the first experiment were transformed in an attempt to control for differences among programs and programmers, a correlation of -.73 ($p \leq .001$) was obtained between the performance criterion and Halstead's E. However, the issue is not whether theories can be validated with mystical transformations of data, but whether the results of these heuristic transformations can be replicated in an experiment designed to overcome the limitations of previous research.

The present experiment evaluated the difficulty of locating three types of errors under controlled programming conditions. In order to compare the effects on performance of different methods of structuring code, programs in the present experiment were implemented in three types of control flow, all of which exhibited a generally top-down flow. This experiment also evaluated the ability of software complexity metrics to predict performance over a wider range of program sizes. To investigate the effects of length, the three programs in this experiment were subdivided into functional subroutines so that they could be presented in three different lengths: approximately 50, 125, and 200 lines of code. Finally, the present experiment attempted to relate programming performance to experiential factors, such as familiarity with other programming languages or relevant programming tools and concepts.

METHOD

Participants

Fifty-four professional programmers at six different locations participated in this experiment. Thirty were civilian employees, while 24 were employees of the military. The participants averaged 6.6 years of professional experience programming in Fortran, ranging from 1/2 year to 25 years (SD = 6.1).

Experimental Design

In order to control for individual differences in performance, a within-subjects, 3^4 factorial design was employed. Three types of control flow were defined for each of three programs, and each of these nine versions was presented in three lengths with three different bugs, for a total of 81 different experimental conditions. The first 27 participants each saw three of the programs, exhausting the 81 conditions (Figure 1). The second set of 27 participants replicated the conditions exactly except that the order of presentation of the tasks was different in each case.

Learning effects were expected on the basis of results obtained in previous experiments of this type (Sheppard, Curtis, Borst, Milliman, & Love, 1979; Sheppard & Love, 1977). Therefore, the order of presentation of conditions was counterbalanced to assure that each level of each independent variable appeared as the first, second, or third task an equal number of times.

Procedure

A packet of materials prepared for each participant included: 1) written instructions on the experimental tasks (Appendix A), 2) a short tutorial of commands used in Fortran

PROGRAM	LENGTH	NATURALLY STRUCTURED			GRAPH-STRUCTURED			FORTRAN 77			← CONTROL FLOW
		1	2	3	1	2	3	1	2	3	← BUG
1 ROOTS	SHORT	1	23	12	20	15	3	18	2	26	
	MEDIUM	19	11	7	14	9	25	8	22	17	
	LONG	10	4	27	6	24	13	21	16	5	
2 ACCT	SHORT	13	8	21	7	27	16	24	10	9	
	MEDIUM	5	26	15	23	18	4	12	6	20	
	LONG	22	14	2	17	1	19	3	25	11	
3 GRADER	SHORT	25	17	6	11	5	22	4	19	14	
	MEDIUM	16	3	24	2	21	10	27	13	1	
	LONG	9	20	18	26	12	8	15	7	23	

EACH CELL REPRESENTS ONE OF THE THREE TASKS GIVEN TO A PARTICIPANT

Figure 1. Assignments of 27 Participants in One Replication of the Experimental Design

77 (Appendix A), 3) a short preliminary task (Appendix B), 4) three experimental tasks, and 5) a questionnaire concerning previous experience (Appendix C).

All tasks included input files, a listing of the Fortran program with the embedded bug, a correct output, and the erroneous output produced by this program. All differences between the correct and erroneous output were circled on the erroneous output. Also included were explanatory descriptions of any subroutines or functions not presented in the listing but referenced by the program.

The 54 participants were divided into two groups of 27, each of which represented a complete replication of the design. Within a group all participants were given the same preliminary task. Group 1 worked with an algorithm to find the greatest common divisor of two numbers and Group 2 was given a simple sort algorithm. These preliminary tasks were provided to reduce learning effects on the experimental tasks and to provide a basis for comparing the abilities of the participants to perform a task of this nature.

Following the initial exercises, participants were presented with three separate programs comprising their experimental tasks. Participants were allowed to work at their own pace, signalling the experimenter when they believed they had identified and corrected the bug. The experimenter verified all corrections, and in the case of a mistake the participant was instructed to try again until the task was successfully completed. The maximum time participants were allowed to work on a particular program was 45 minutes for the preliminary task and 60 minutes for each experimental task. Time was measured to the nearest minute.

Independent Variables

Program. Three programs were selected for the generality of their content and their understandability to programmers. The first program sorted and categorized alphabetic response data to a questionaire (Veldman, 1967). The second program, an accounting routine, produced income and balance statements (Nolen, 1971). Program 3 kept track of students' test grades and calculated their semester averages (Brooks, 1978). All programs were tested prior to the experiment.

Length. The inclusion of additional subroutines made it possible to present each program in three different lengths. The shorter programs had 25-75 statements, medium programs contained 100-150 statements, and the longer programs contained approximately 175-225 statements. (One Fortran 77 version exceeded the 225 line limit by 8 lines because of the number of ELSE and ENDIF statements required).

Program listings included a two or three line explanation of any routine or function that was called by a program but not presented in the experimental materials. Participants were told to assume that missing routines worked correctly. All of the input and output files were presented regardless of the length of the program. That is, for the shorter version, some of the input was read in and some of the output was produced by subroutines which were not presented.

Complexity of control flow. Three versions of control flow performing identical tasks were defined for each program. Two types of structures were implemented in Fortran IV, naturally structured and graph-structured. A third version was written in Fortran 77 (Brainerd, 1978), which includes the IF-THEN-ELSE, DO-WHILE, and DO-UNTIL constructs.

The Fortran 77 version of each program was implemented in a precisely structured manner. All flow proceeeded from top to

bottom, and only three basic control constructs were allowed: the linear sequence, structured selection, and structured iteration (Figure 2).

The graph-structured version of each program was implemented in Fortran IV from the Fortran 77 version, replacing the special constructs but producing code for which the control flow graphs of the two versions were identical. All nested relationships could be reduced through structured decomposition to a linear sequence of unit complexity. A full discussion of reducibility is presented by McCabe (1976).

Structured constructs are awkward to implement in Fortran IV (Tenny, 1974). In order to test a more naturally structured flow, limited deviations were allowed in a third version of each program. These deviations included such practices as branching into or out of a loop or decision and multiple returns. Control flow graphs and the code for a section of a routine implemented in all three versions of control flow are presented in Figures 3 and 4.

Each program was indented following the nesting patterns presented in the code. Thus, all DO loops and branching instructions were indented. For naturally structured versions, decisions were made arbitrarily about the importance of various constructions, and indenting was necessarily less standardized than for the graph-structured and Fortran 77 versions.

Type of Bug. Three types of semantic bugs were chosen from a classification developed by Hecht, Sturm, and Trattner (1978): computational, logical, and data errors. Bugs in each category were defined for each of the three programs in order to maximize the similarity of bugs from a single category across programs. Computational bugs involved a sign change in an arithmetic expression. Logic bugs were implemented by using the wrong logical operator in an IF condition. Data bugs involved

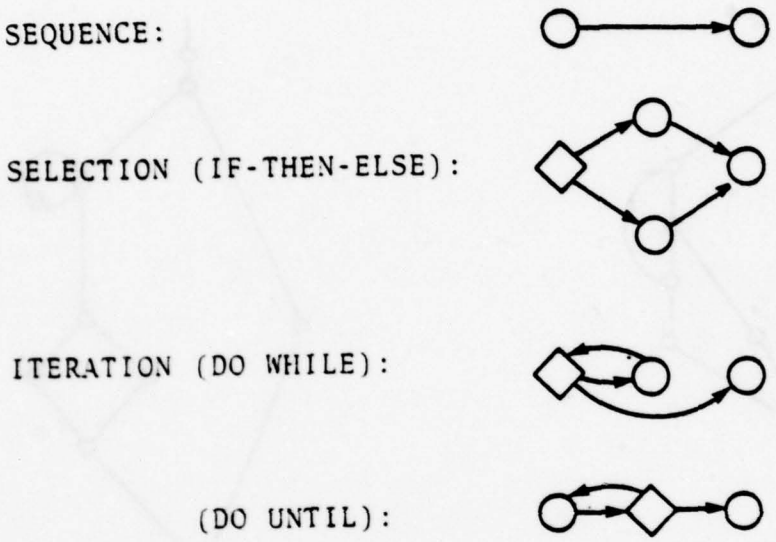
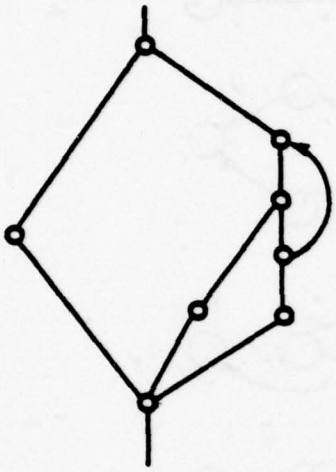
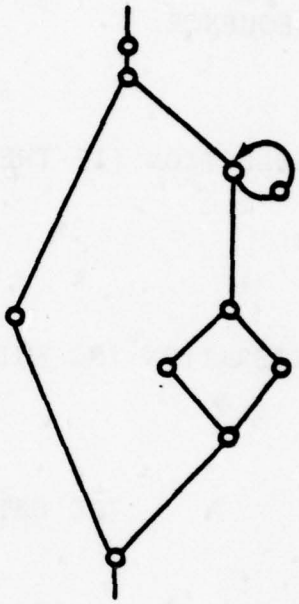


Figure 2. The Basic Structured Constructs



Naturally structured



Fortran 77 and
Graph-Structured
Fortran IV

Figure 3. Control Graphs for All Versions of Control Flow

NATURALLY STRUCTURED

```
      IF (ASNUM .LT. 1 .OR. ASNUM .GT. NASSGN) GO TO 420
         DO 400 K=1,NSTUDN
            IF (CURID .EQ. ID(K)) GO TO 440
400         CONTINUE
         PRINT 410,CURID
410      FORMAT (1H0,30X,' ID NUMBER NOT IN FILE:  ',I8)
         GO TO 450
420      PRINT 430, CURID,ASNUM
430      FORMAT (1H0,30X,'ID ',I8,'ILLEGAL ASSIGNMENT',I3)
         GO TO 450
440   SCORE(K,ASNUM)=VAL
450   CONTINUE
```

GRAPH-STRUCTURED

```
      K=1
      IF (ASNUM .LT. 1 .OR. ASNUM .GT. NASSGN) GO TO 420
400      IF (CURID .EQ. ID(K) .OR. K .GT. NSTUDN) GO TO 405
            K=K+1
            GO TO 400
405      IF (K .LE. NSTUDN) GO TO 415
            PRINT 410,CURID
410         FORMAT (1H0,30X,' ID NUMBER NOT IN FILE:  ',I8)
            GO TO 450
415         SCORE(K,ASNUM)=VAL
            GO TO 450
420      PRINT 430, CURID,ASNUM
430      FORMAT (1H0,30X,'ID ',I8,' ILLEGAL ASSIGNMENT ',I3)
450      CONTINUE
```

FORTRAN 77

```
      K=1
      IF (ASNUM .GE. 1 .AND. ASNUM .LE. NASSGN) THEN
            DO 400 WHILE (CURID .NE. ID(K) .AND. K .LE. NSTUDN)
400            K=K+1
               IF (K .GT. NSTUDN) THEN
                  PRINT 410,CURID
410               FORMAT (1H0,30X,' ID NUMBER NOT IN FILE:  ',I8)
               ELSE
                  SCORE(K,ASNUM)=VAL
               ENDIF
         ELSE
            PRINT 430, CURID,ASNUM
430         FORMAT (1H0,30X,'ID ',I8,' ILLEGAL ASSIGNMENT ',I3)
         ENDIF
450   CONTINUE
```

Figure 4. Examples of the Three Types of Control Flow

wrong index values for variables. Examples of these bugs and the routines in which they were inserted are presented in Appendix D.

Each bug in this experiment was purposely designed to affect only a limited area of code. That is, each calculation containing a bug occurred near the corresponding WRITE and FORMAT statements. In no case did a bug produce errors in routines other than the one in which it was embedded, and each bug appeared in only one line of code.

Individual Differences Measures

Scores on the preliminary exercise were used as a measure of programming ability related to the experimental task. Participants were also asked to complete a questionnaire about their programming experience. The information requested included specific type of experience, number of years programming professionally in Fortran, number of statements in the longest Fortran and non-Fortran programs written, the first programming language learned, and number of languages learned. In addition, various programming concepts that appeared relevant to the experimental programs were listed, and participants were asked to mark those with which they were familiar.

Complexity Metrics

Halstead's E. Using a program based on Ottenstein (1976), Halstead's effort metric (E) was computed from the source code listings of the 27 experimental programs, representing three distinct programs at three levels of structure and three different lengths. The computational formula was:

$$\underline{E} = \frac{n_1 N_2 (N_1 + N_2) \log_2 (n_1 + n_2)}{2n_2}$$

where,

n_1 = number of unique operators
n_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) = # edges - # nodes + 2 (# connected components).

McCabe presents two simpler methods of calculating v(G): the number of predicate nodes plus 1 or the number of regions computed from a planar graph of the control flow.

Length. The length of the program was the total number of Fortran statements, excluding comments. The total number of executable statements was found to be highly correlated with number of statements (r = .99, $p \leq .001$).

Dependent Variable

The dependent variable was the number of minutes necessary for the participant to locate and correct the bug.

Analysis

The analysis of data was conducted in two phases. The first phase was an experimental test of the independent variables, while the second phase evaluated the software complexity metrics. In the first phase, experimental data were analyzed in a hierarchical regression analysis. In this analysis, domains of variables were entered sequentially into a multiple regression equation to determine if each successive domain significantly

improved the predictive capability of the equation developed from domains already entered. Thus, the order in which domains were entered into the analysis was important. Variables representing the different conditions of experimentally manipulated variables were effect-coded (Kerlinger & Pedhazur, 1973).

The second phase of analysis investigated relationships between the time to find the bug and the metrics, Halstead's E, McCabe's v(G), and number of statements in the program. All correlations are Pearson product-moment correlations.

RESULTS

Preliminary Tasks

Group 1 (Participants 1-27) and Group 2 (Participants 28-54) were given different preliminary tasks. The two algorithms were of varying difficulty, producing significant differences in both time to completion and percent of completions. Finding the bug in the greatest common divisor algorithm required an average of 23.8 minutes with 22% failing to find the bug in 45 minutes, while the sorting algorithm required only 14.6 minutes with only 4% failing to find the bug. However, no significant differences in performance between the two groups occurred on the experimental programs.

Experimental Manipulations

The average time to locate bugs across all experimental conditions was 20.1 minutes (SD = 16.2). All but six of the 162 experimental tasks comprising this experiment were completed successfully during the allotted 60 minutes. These six conditions were not associated with any particular factor.

Despite the use of a preliminary task to familiarize the participants with the experiment, a significant order effect occurred ($p \leq .04$), indicating that learning took place during the first of the three experimental tasks (Figure 5).

Results of a hierarchical regression analysis of the independent variables on the time to find the bug are presented in Table 1. Differences in solution time for the three programs were significant ($p \leq .01$). Finding the bug in the accounting program required an average of 15.1 minutes, 20.0 minutes in the program that sorted questionnaire data, and 25.0 minutes in the grade-scoring program. Increasing the length of the programs had a modest effect ($p \leq .06$) on the time to locate and correct

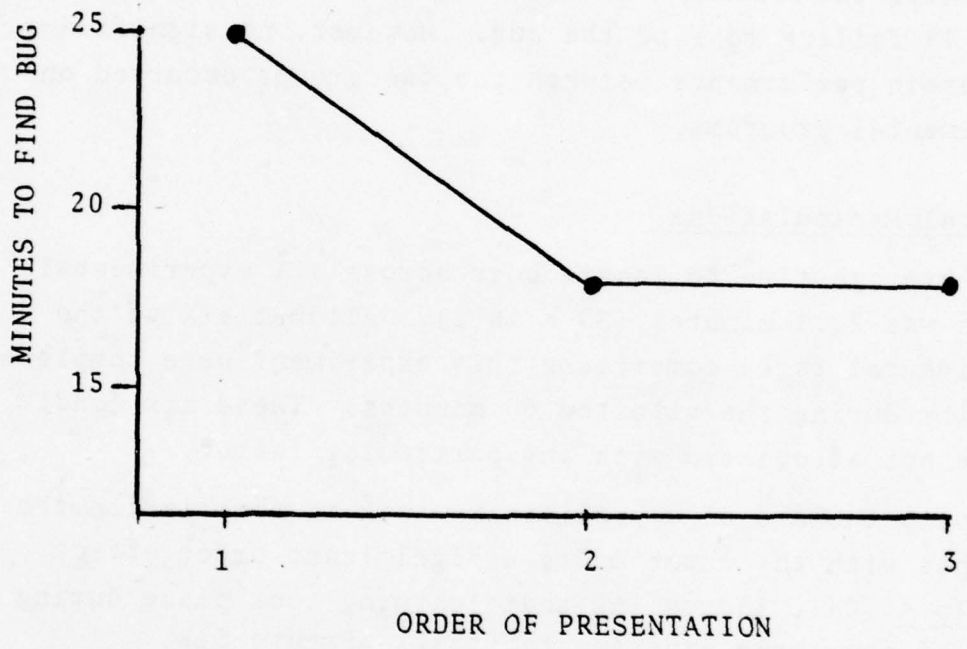


Figure 5. Order Effect on the Three Experimental Tasks

TABLE 1

Hierarchical Regression Analysis
for Time to Find Bug

	Variable	df	R^2	ΔR^2
(1)	Program	2	.06**	.06**
(2)	Presentation order	2	.04*	.04*
(3)	Type of bug	2	.00	.00
(4)	Program X bug interaction	4	.26***	.26***
(5)	Complexity of control flow	2	.02	.02
	All variables	12		.38***

Note: $n = 162$. R^2 column represents the separate regression for each domain.

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

the error. The average time for the short program was 16 minutes, while the medium and long programs required a mean of 21 and 23 minutes, respectively.

Averages for the three error categories were not significantly different from one another. However, a very large interaction occurred between type of bug and program (Figure 6). This interaction accounted for the largest percent of variance (26%) of any of the experimental relationships studied. No significant differences in performance resulted from the three types of control flow.

Software Complexity Metrics

Intercorrelations among the three measures of software complexity were computed from the 27 different versions of the programs at both the subroutine and program levels (Table 2). Substantial intercorrelations were observed among Halstead's E, McCabe's v(G), and length at the subroutine level. When computed on the total program, the correlation between length and McCabe's v(G) increased, while the correlations for Halstead's E with these two measures were substantially smaller, especially with lines of code.

Correlations between time to find the bug and the complexity metrics were calculated for unaggregated data (three experimental tasks for each of the 54 participants, $n = 162$) and for data averaged over the six scores obtained for each program (Table 3). Correlations for the aggregated data were much higher than those for the unaggregated scores. All three metrics predicted performance equally well at the subroutine level. At the program level, however, E was the best predictor, accounting for more than twice the variance in performance than did the length (56% versus 27%, respectively). The variance accounted for by v(G) fell between these values (42%). A stepwise multiple

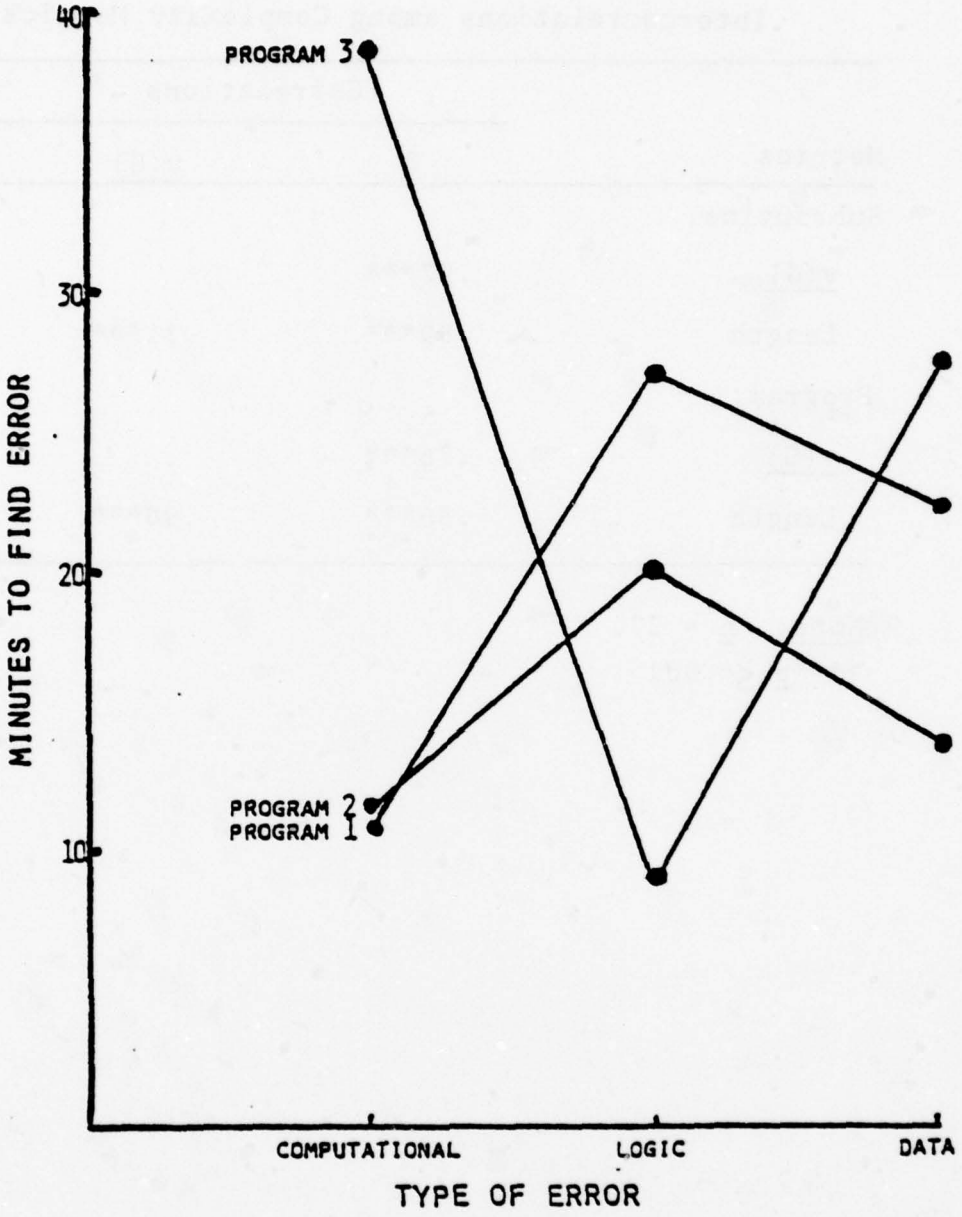


Figure 6. Program by Error Interaction

TABLE 2

Intercorrelations among Complexity Metrics

Metrics	Correlations	
	E	v(G)
Subroutine:		
v(G)	.92***	
Length	.89***	.81***
Program:		
v(G)	.76***	
Length	.56***	.90***

Note: $n = 27$.

*** $p \leq .001$

TABLE 3

Correlation Between Performance Time and Complexity Metrics

Metric	Correlations	
	Unaggregated (n = 162)	Aggregated (n = 27)
Subroutine:		
Halstead's E	.25***	.66***
McCabe's $v(G)$	.24***	.63***
Length	.25***	.67***
Program:		
Halstead's E	.28***	.75***
McCabe's $v(G)$	.25***	.65***
Length	.20**	.52**

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

regression analysis indicated that length and v(G) added no increments to the prediction afforded by E.

The scatterplot of performance with Halstead's E presented in Figure 7 suggested the existence of a curvilinear trend in the data. The significance of this trend was tested using the second degree polynomial regression approach suggested by both Cohen and Cohen (1975) and Kerlinger and Pedhazer (1973) for investigating curvilinear relationships. A multiple correlation coefficient of .84 indicated that the curvilinear trend accounted for an additional 15% ($p \leq .001$) of the variance beyond that accounted for by a linear relationship. The prediction equation generated from these data was:

$$\text{minutes to find bug} = 9.837 + .00239E - .00000000079E^2$$

However, with few data points in the right tail of this distribution for Halstead's E, it is difficult to extrapolate to the exact shape of the curvilinear trend. No curvilinear trend was detected with either the lines of code or McCabe's v(G).

Experiential Factors

The relationship between complexity metrics and performance was investigated within groups of programmers differing in years of professional experience programming in Fortran. As a heuristic, the participants were divided into two groups of approximately equal numbers: those with three or fewer years experience and those with more than three years experience. The results presented in Table 4 indicate that the complexity measures were more predictive of performance for less experienced programmers, especially when computed at the subroutine level.

Two measures of experience were also found to be related to the performance of less experienced programmers (Table 5), but not to the performance of experienced programmers. The first such measure was the number of programming languages the

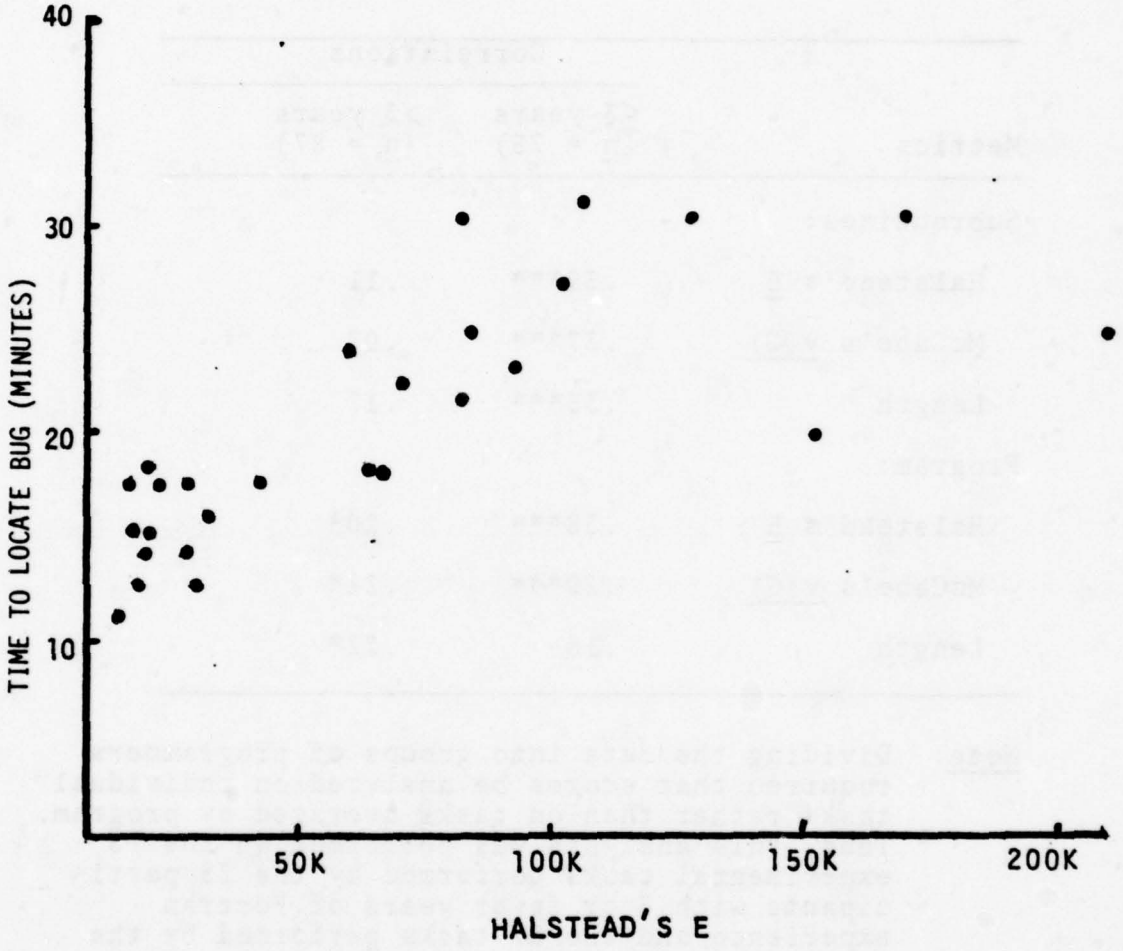


Figure 7. Scatterplot of Halstead's E and Performance

TABLE 4

Correlations between Performance and Complexity Metrics Moderated by Years of Fortran Experience

Metrics	Correlations	
	≤3 years (n = 75)	>3 years (n = 87)
Subroutines:		
Halstead's E	.39***	.11
McCabe's v(G)	.37***	.07
Length	.33***	.17
Program:		
Halstead's E	.38***	.20*
McCabe's v(G)	.29***	.21*
Length	.18	.22*

Note: Dividing the data into groups of programmers required that scores be analyzed on individual tasks rather than on tasks averaged by program. Thus, this analysis was performed on the 75 experimental tasks performed by the 25 participants with 3 or fewer years of Fortran experience and the 87 tasks performed by the 29 participants with more than 3 years experience.

*$p \leq .05$

**$p \leq .01$

***$p \leq .001$

TABLE 5

Relationships of Experiential Factors to Performance for Programmers Differing in Fortran Experience

Relevant experience	≤3 years (n = 25)	>3 years (n = 29)	Total (n = 54)
# of programming languages	-.49**	-.03	-.19
Questionnaire score	-.48**	-.11	-.33**

**$p \leq .01$

participant knew. The second metric was the number of items checked on the experience questionnaire (Appendix C). The moderating effects of programmer experience may have been the result of greater variability in performance for programmers with less experience (Figure 8). This greater variability would increase the ability of correlational tests to detect significant relationships (Cohen & Cohen, 1975).

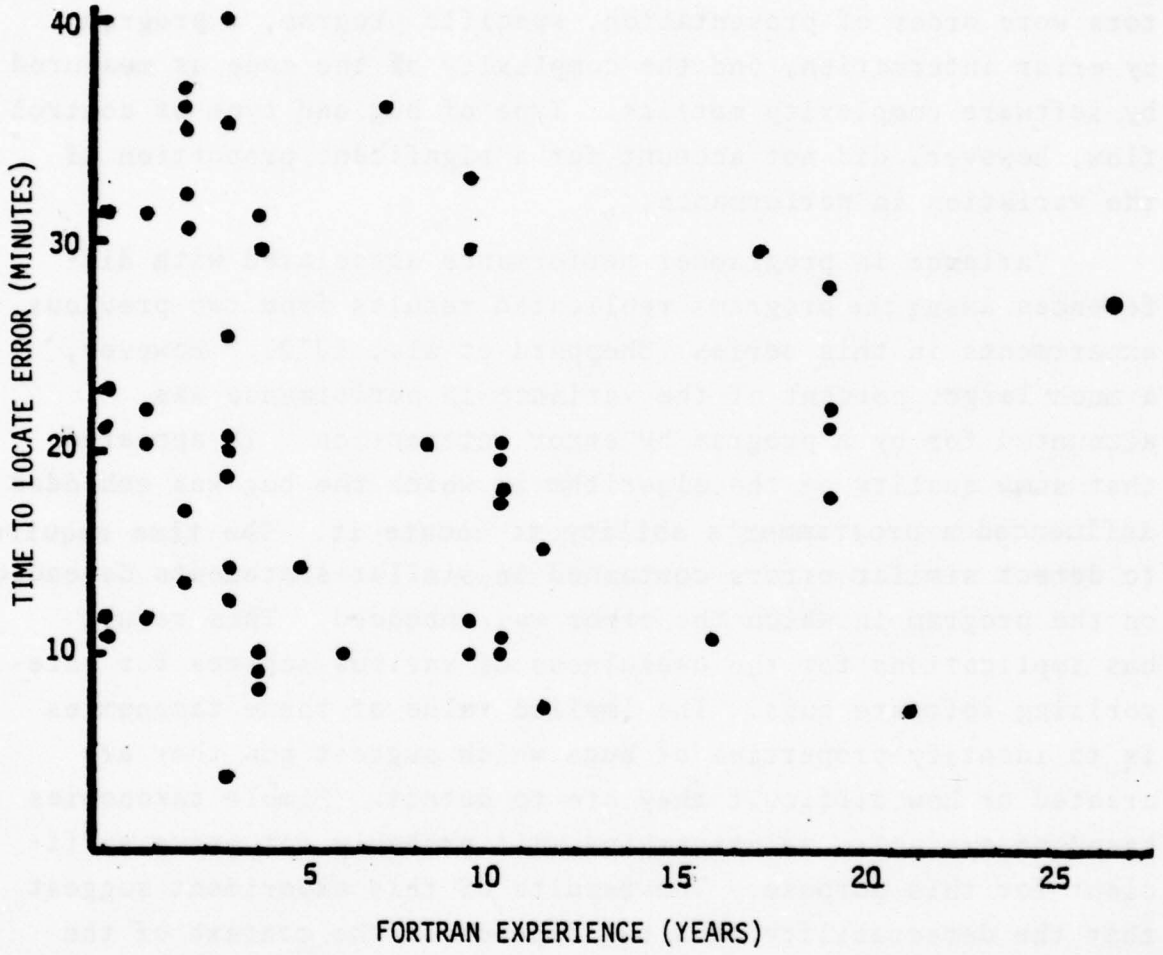


Figure 8. Scatterplot of Experience and Performance

DISCUSSION

Four factors were found to influence the speed with which programmers could find a bug in a computer program. These factors were order of presentation, specific program, a program by error interaction, and the complexity of the code as measured by software complexity metrics. Type of bug and type of control flow, however, did not account for a signficant proportion of the variation in performance.

Variance in programmer performance associated with differences among the programs replicated results from two previous experiments in this series (Sheppard et al., 1979). However, a much larger percent of the variance in performance was accounted for by a program by error interaction. It appeared that some quality of the algorithm in which the bug was embedded influenced a programmer's ability to locate it. The time required to detect similar errors contained in similar statements depended on the program in which the error was embedded. This result has implications for the usefulness of various schemes for categorizing software bugs. The implied value of these taxonomies is to identify properties of bugs which suggest how they are created or how difficult they are to detect. Simple taxonomies based on syntactic relationships will probably not prove sufficient for this purpose. The results of this experiment suggest that the detectability of a bug depends on the context of the algorithm surrounding it. This contextual effect may determine the optimal search strategy for finding the bug, and it is this search strategy that needs to be understood if debugging performance is to be improved.

In the last section of the post-session questionnaire, the participants were asked to describe their searching strategies for locating the bugs. Typically, one of two approaches was

described. In the first strategy the programmer tried to understand the whole program from beginning to end before searching for the section with the bug. In the second strategy the programmer used appropriate clues in the output to go directly to the section containing the bug. The latter appeared to be a much quicker strategy for debugging, but there were insufficient data for a meaningful statistical analysis. In order to improve the debugging performance of programmers it will be important not only to identify effective search strategies, but also to identify conditions under which they will be differentially effective.

No significant differences were evident among the three types of top-down control flow tested in this experiment. This finding agrees with previous results (Sheppard et al., 1979) where differences were found between top-down and convoluted control flow, but not between types of top-down control flow. The minor deviations from strictly structured coding allowed in the naturally structured version of this experiment did not adversely affect performance. Summarizing the combined results of the three experiments, it would appear that the overall top-down quality of the control flow is important to performance, but careful attention to strict structuring does not appear to improve programmer performance significantly.

Since no difference was found between the graph-structured and Fortran 77 program versions, it would appear that the newer constructs provide little additional aid in a debugging task beyond that provided by a top-down flow. Only five of the 54 participants had previously used Fortran 77, so a lack of familiarity with the new constructs may have prevented them from finding the bug more quickly in Fortran 77 than in Fortran IV. However, immediately prior to the experiment a short training session was conducted with each group of participants in

which the new Fortran 77 constructs were discussed in detail. These constructs were similar to those implemented in Fortran IV, and the participants' previous lack of familiarity with them was probably not a significant factor in their performance.

Most laboratory studies exhibit a certain degree of artificiality that is necessary for experimental control. In this experiment participants were told there was only one bug in a program. While this situation differs from a normal programming environment, it should not have affected participant's ability to perform the tasks. These experimental tasks may have been simpler to perform than typical debugging problems since there was greater certainty about the bugs. Further, differences between the correct and erroneous output were clearly marked on the erroneous output, reducing the amount of comparison necessary to discover what problems had occurred.

During a typical debugging problem a programmer could refer to the functional specifications for a program or to comments included in the code. However, no such aids were made available in this experiment. The participant's comprehension of the program's function had to be gleaned from the code or from the input and output listings. The latter were designed to be self-explantory, with each section labeled appropriately; e.g., "FINAL COURSE GRADE" or "TRIAL BALANCE". Although adding some artificiality to the experimental situation, the absence of documentation was an attempt to equalize the amount of information provided by materials other than the code.

Software Complexity Metrics

The results of this experiment not only replicated the results obtained in our previous research, but also demonstrated that more viable results could be obtained when limitations in our earlier experimental procedures were overcome. For instance,

our previous research was conducted exclusively on small-sized (35-55 lines of code) programs, which seems to have limited the results in three ways. First, the range of values on the factors studied in those programs seems to have been too restricted to detect the size of relationships observed here. Second, the curvilinear relationship observed in this experiment between Halstead's E and performance would not have been observed if longer programs had not been used in the experimental tasks. Third, the extremely high intercorrelation between length and Halstead's E at the subroutine level suggests that both are measuring program volume. With larger programs the information measured appears to differ; that is, Halstead's E measures something in addition to, but inclusive of, factors measured by length.

Many small-sized programs can be grasped by the typical programmer as a cognitive gestalt. The psychological complexity of such programs is adequately represented by the volume of the program in terms of the number of lines of code. When the code grows beyond a subroutine, its complexity to the programmer is better assessed by measuring constructs other than the number of lines of code. This may result partly because programmers cannot grasp the entire program within their mental spans at a single time. For larger programs the difficulty programmers experience is better represented by counts of operators, operands, and control paths. Thus, as the size of a program increases, Halstead's E seems to be a better measure of its psychological complexity.

One possible explanation for the superior predictive ability of Halstead's E is that the relationship between program size and performance is curvilinear, and the algorithmic transformation with the Halstead measure captures this relationship while lines of code does not. There was no evidence in

these data of a curvilinear relationship between lines of code and performance. On the other hand, a curvilinear relationship did exist between Halstead's E and performance. This trend suggests that as Halstead's E grows larger, a program becomes more psychologically complex, but the increments in difficulty grow smaller and smaller. In the experimental task used in this debugging experiment, there seemed to be an amount of time that was typically required to locate a bug within a subroutine once the correct subroutine had been identified (approximately 16 minutes). Added to this baseline rate was the time required to identify the proper subroutine. The curvilinearity of the relationship between time to find the bug and Halstead's E appeared to result from the time required to isolate the problem subroutine.

The moderating effects of experiential factors also replicated the results found in the earlier experiments. The metrics again proved to be better predictors of performance for programmers with three or fewer years experience in Fortran than for those with more than three years experience. It was also possible to predict the performance of an individual programmer from job history data. Several important factors seemed to be the number of languages a programmer had used and familiarity with certain programming concepts. These predictions from job history were also more valid for programmers who had three or fewer years of experience in Fortran. Future work is needed to refine the use of experiential questionnaires for use in personnel functions such as selection, assessment for training needs, and placement.

Code which is more psychologically complex may also be more error-prone and difficult to test. The results of this experiment provide evidence that the software complexity metrics developed by Halstead and McCabe are related to the difficulty

programmers experience in locating errors in code. Thus these metrics appear to be capable of satisfying several practical applications. They can be used in providing feedback both to programmers about the complexity of the code they have developed and to managers about the resources that will be necessary to maintain particular sections of code. Further evaluative research needs to assess the validity of these uses in ongoing software projects.

ACKNOWLEDGEMENTS

The authors are grateful to Judy McWilliams and Mary Anne Borst who helped with this experiment and to Beverly Day for manuscript preparation. We are also grateful to Dr. Gerald Hahn for advice on experimental design, to Drs. Tom Love and Ben Shneiderman for advice on the experimental tasks and procedures, and to Dr. John O'Hare for his careful review of this report. We are especially appreciative of the efforts of Earl North and Leo Pompliano of General Electric; Jan Gombert of Applied Urbanetics; Mrs. Joan Shields, Cols. William Eglington, Earl Goetze and Richard Blair, and Lt. Col. Pat Harris of the U.S. Air Force; and Capt. Webster and J. Rehbehn of the U.S. Navy in providing the participants for this research. The support and encouragement of both Gerald Dwyer and Lou Oliver has been vital to the success of this research.

REFERENCES

Brainerd, W. Fortran 77. Communications of the ACM. 1978, 21, 806-820.

Brooks, R. Unpublished algorithm. Irvine, CA: University of California at Irvine, Computer Science Department, 1978.

Campbell, D., & Stanley, J.C. Experimental and quasi-experimental designs for research. Chicago: Rand-McNally, 1967.

Carlson, W.E., & DeRoze, B. Defense system software research and development plan. Unpublished manuscript, Arlington, VA: Defense Advanced Research Projects Agency, September 1977.

Cohen, J., & Cohen, P. Applied multiple regression/correlation analysis for the behavioral sciences. New York: Wiley, 1975.

Curtis, B., Sheppard, S.B., Milliman, P., Borst, M.A., & Love, T. Measuring the psychological complexity of software maintenance tasks with the Halstead and McCabe metrics. IEEE Transactions on Software Engineering, 1979, 5, 95-104.

Department of Defense requirements for high order computer programming languages: Revised "IRONMAN". SIGPLAN Notices, 1977, 12, 39-54.

DeRoze, B. Software research and development technology in the Department of Defense. Paper presented at the AIIE Conference on Software, Washington, D.C.: December 1977.

Dijkstra, E.W. Notes on structured programming. In Structured programming, O.J. Dahl, E.W. Dijkstra, and C.A.R. Hoare, (Ed.) New York: Academic, 1972.

Fitzsimmons, A.B., & Love, L.T. A review and evaluation of software science. ACM Computing Survey, 1978, 10, 3-18.

Gordon, R.D. A measure of mental effort related to program clarity. Unpublished doctoral dissertation, Purdue University, 1977.

Gould, J.D. Some psychological evidence on how people debug computer programs. International Journal of Man-Machine Studies, 1975, 7, 151-182.

Gould, J.D., & Drongowski, P. An exploratory study of computer program debugging. Human Factors, 1974 16, 258-277.

Halstead, M.H. Elements of software science. New York: Elseiver North-Holland, 1977.

Hecht, H., Sturm, W.A., & Trattner, S. Reliability measurement during software development. Redondo Beach, CA: Aerospace Corp., 1978.

Kerlinger, F.N., & Pedhazur, E.J. Multiple regression in behavioral research. New York: Holt, Rinehart, & Winston, 1973.

McCabe, T.J. A complexity measure. IEEE Transactions on Software Engineering, 1976, 2, 308-320.

Nolen, R.L. Fortran IV computing and applications. Reading, MA: Addison-Wesley, 1971.

Ottenstein, K.J. A program to count operators and operands for ANSI-FORTRAN modules (Tech. Rep. CSD-TR-196). West Lafayette, IN: Purdue University, Computer Science Department, 1976.

Sheppard, S.B., Curtis, B., Borst, M.A., Milliman, P., & Love, L.T. First year results from a research program on human factors in software engineering. In Proceedings of the 1979 National Computer Conference, Montvale, NJ: AFIPS, 1979.

Sheppard, S.B., & Love, L.T. A preliminary experiment to test influences on human understanding of software. In Proceedings of the 21st Meeting of the Human Factors Society. Santa Monica, CA: Human Factors Society, 1977.

Tenny, T. Structured programming in FORTRAN. Datamation, 1974, 20, 110-115.

The military software market (Rep. 427). New York: Frost & Sullivan, 1977.

Veldman, D.J. Fortran programming for the behavioral sciences. New York: Holt, Rinehart, & Winston, 1967.

Wescourt, K.T., & Hemphill, L. Representing and teaching knowledge for troubleshooting/debugging (Tech. Rep. 292). Stanford, CA: Stanford University, Institute for Mathematical Studies in Social Science, 1978.

Youngs, E.A. Human errors in programming. International Journal of Man-Machine Studies, 1974, 6, 361-376.

APPENDIX A

INSTRUCTIONS TO PARTICIPANTS

Instructions To Participants

HELLO

Today we are going to ask you to participate in an experiment which we hope will be both entertaining and challenging. This study is being sponsored by GE and the Office of Naval Research to examine the properties of bugs in computer programs. To accomplish this, we will give you several different programs and ask you to find a bug in each one. Our purpose is to evaluate characteristics of programs which make them easier to debug. It is not to evaluate computer programmers. Your performance on a program will be compared only to your performance on other programs, and no form of competition is involved. We hope you will assist us in what we believe is important research in software engineering. However, your involvement is voluntary and you are free to withdraw from participation at any time. All programs and papers that you will be handed are carefully numbered so it is not necessary for you to put your name on any of these. These numbers are solely for the purpose of identifying different programs and cannot be used to identify you as an individual. Your work will remain completely anonymous and data collected in this study will be used for research purposes only.

For each task, you will be given a program, the input files, and both the correct and incorrect output produced by the program you have. Your job is to identify the bug and correct it. Each bug can be corrected by inserting, deleting or correcting one line of code. When you believe you have corrected the bug, please inform the monitor by raising your hand.

During this experiment, each of you will be working on a different program. If others seem to finish earlier than you, don't be concerned. They may have been working on a program which did not require as much time.

We will begin with a short introductory program. Raise your hand as soon as you have found the bug and corrected it. Because of the concentration required for this task, we ask you to make an extra effort to remain quiet so that others will not be distracted. When you have completed all three experimental programs you are free to leave, but please do not discuss any of the programs you worked on with anyone else until after we have completed all experimental sessions. We request this of you only to insure that our results are valid.

If there are any questions, please ask them at this time.

One of the programs you see will be in Fortran-77. It is very similar to standard Fortran except for the addition of three constructs.

F77 allows:

1. if:

```
      IF (condition)  THEN
         any statement
               .
               .
               .
         ENDIF
```

or:

```
      IF (condition)  THEN
         any statement
               .
               .
               .
         ELSE
          any statement
               .
               .
               .
         ENDIF
```

2. do while:

```
      DO statement # WHILE (condition)
         any statement
               .
statement #    .
```

3. repeat until:

```
      DO statement # UNTIL (condition)
          any statement
               .
               .
statement #    .
```

Miscellaneous:

input and output files may be referenced by a string

spaces are not important

line lengths are not important

& after the line number indicates a continuation line

Fortran 77 IF's can be nested

program order will be the following:

input

program

correct output

incorrect output with bad results circled

APPENDIX B

PRETESTS

Greatest Common Divisor Algorithm

SOURCE CODE LISTING

```
110         INTEGER GCD, REMAIN
115         I=0
120         READ("EUCDAT",1) M,N
130 1       FORMAT(2I5)
140         IF (M.EQ.0) THEN
150         GCD=N
160         ELSE
170         IF (N.EQ.0) THEN
180         GCD=M
190         ELSE
200         IG=M/N
210         REMAIN=M-N*IG
220         DO 2 WHILE (REMAIN.NE.0 .AND. I.LT.100)
230         M=N
240         M=REMAIN
250         IG=M/N
260         REMAIN=M-N*IG
265         I=I+1
270 2       CONTINUE
275         GCD=N
280         ENDIF
290         ENDIF
294         IF (I.LT.100) THEN
295         PRINT 3,GCD
296 3       FORMAT(" GCD = ",I5)
297         ELSE
298         PRINT 4
299 4       FORMAT(" TOO MANY ITERATIONS")
300         ENDIF
301         STOP
310         END
```

INPUT

EUCDAT

```
30   25
```

INCORRECT OUTPUT

```
TOO MANY ITERATIONS
```

CORRECT OUTPUT

```
GCD =     5
```

Sorting Algorithm

INPUT

DATAPRE

```
 25
110
 30
 31
  1
153
193
 62
 78
 16
  1
193
 62
 78
 74
168
192
199
999
  5
 78
 79
 56
  9
 57
  3
```

```
100         IMPLICIT INTEGER(A-Z)
110         DIMENSION A(50),B(50)
115         READ("DATAPRE",10) N
116         DO 5 I = 1, N
120     5   READ("DATAPRE",10) A(I)
130    10   FORMAT(I3)
140         DO 100 J = 1, N
160         SMALL = A(1)
170         M = 1
180         DO 20 K = 2,N
190    15   IF(A(K) .LT. SMALL) GO TO 20
200         SMALL = A(K)
210         M = K
220    20   CONTINUE
230         B(J) = SMALL
240         A(M) = 1000
250   100   CONTINUE
251         DO 101 I = 1, N
260   101   PRINT 110, B(I)
261   110   FORMAT(2X,I4)
270         STOP
280         END
```

CORRECT OUTPUT

```
  1
  1
  3
  5
  9
 16
 30
 31
 56
 57
 62
 62
 74
 78
 78
 78
 79
110
153
168
192
193
193
199
999
```

INCORRECT OUTPUT

```
 999
1000
1000
1000
1000
1000
1000
1000
1000
1000
1000
1000
1000
1000
1000
1000
1000
1000
1000
1000
1000
1000
1000
1000
1000
```

APPENDIX C

EXPERIENCE QUESTIONNAIRE

SUMMARY

QUESTIONNAIRE

We would like you to answer the following questions for our research purposes:

1. How long have you been programming in FORTRAN professionally?

   _____years _____months

2. Please circle one of the following: Has your experience primarily been with

   a. Engineering
   b. Statistical
   c. Non-Numeric
   d. Business
   e. Other (Please describe_____

   Also, please briefly describe your specific areas of programming experience.

   _____

   _____

3a. Approximately how many source code instructions were in the longest FORTRAN program that you have ever written? Please exclude blank lines and comments_____.

   b. What is the length of the longest non-FORTRAN program you have ever written?_____

   What language?_____

4. Place a check in the appropriate blank for each of the following languages you have used:

   a. FORTRAN _____
   b. FORTRAN 77 _____
   c. COBOL _____
   d. PL/1 _____
   e. BASIC _____
   f. PASCAL _____
   g. APL _____
   h. ALGOL _____
   i. JOVIAL _____
   j. assembler _____
   k. RPG _____
   l. SNOBOL _____
   m. LISP _____
   n. other _____

5. What was the first programming language you learned?_____

6. Place a check in the appropriate blank for each of the following you have used when coding:

DO statement	_____	DATA statement	_____
arrays	_____	conversion from alpha to string variables	_____
CALL with parameters	_____	IF of more than 1 condition	_____
COMMON	_____	decimal to integer conversions	_____
READ statement	_____	percentile computation	_____
PRINT statement	_____	DO WHILE (concept)	_____
WRITE statement	_____	DO UNTIL (concept)	_____
FORMAT statement	_____	weighting numbers	_____
'X' format specification	_____	rounding numbers when don't have rounding function	_____
'A' format specification	_____	used an array reference as an index to another array	_____
'I' format specification	_____	finding Maximum value in array	_____
'F' format specification	_____	finding mean of values in an array	_____
continuation lines	_____	printing titles in an output	_____
'H' format specification	_____	computing frequencies of items	_____
implicit data types	_____	running SUMS	_____
IF THEN ELSE (concept)	_____	Bubble SORT	_____
CREDITS in monetary trans.	_____	implied DO	_____
DEBITS in monetary transactions	_____	equivalenced arrays	_____
Financial transactions	_____	String variables	_____
TRIAL BALANCE computation	_____	used the binary equivalent of characters	_____
GENERAL LEDGER Accounting	_____	Interactive debugger	_____
REAL NOTATION (Ø.Ø1)	_____	symbolic debugger	_____
Tax computation	_____	TRACE mechanism	_____
carriage control Holerith	_____	Octal or Hex dumps	_____
2 or more dimensional arrays	_____	Double Precision	_____
using " in output formats	_____	free field I/O	_____
IMPLICIT statement	_____	matrix inversion	_____
HEAP sorts	_____	pattern matching	_____
stacks	_____	device drivers	_____
tree search	_____	batch systems	_____
NAMELIST statement	_____	interactive systems	_____
'T' format specification	_____	list handling languages	_____
interrupt handlers	_____		
parsers	_____		
lexical analyzers	_____		
graphics drivers and handlers	_____		

7. Please indicate in the space provided any other particulars which you feel may have an effect on your performance(for instance, if most of your work is involved in debugging systems we would like to know that).

8. Please indicate your reactions to the experiment and anything that might help us understand how you undertook the task. Please include any problems or insights you may have had.

APPENDIX D

SUBROUTINES WITH ERRORS

Questionnaire Scoring Program

```
2500      SUBROUTINE SCORE(LX, MSEX, L)
2510      INTEGER MSEX(100, 2), CAT(100), RLENG(100)
2520      ALPHA LX(100,3), ROOT(100,3), KA, KB, KC
2530      DIMENSION F(25,2)
2550      READ("DATC12", 4) NR, NC
2560    4 FORMAT(2I3)
2570    5 FORMAT (I2, 1X, 3A4)
2590      DO 10 I = 1,NR
2600         READ("DATC12", 5) CAT(I), (ROOT(I,J), J= 1,3)
2620   10    CONTINUE
2640      DO 20 I = 1,NR
2650         RLENG(I) = LGTH(ROOT(I, 1),ROOT(I, 2), ROOT(I, 3))
2670   20    CONTINUE
2680      PRINT 1
2690      PRINT 6
2700    1 FORMAT(///17H0ECHO INPUT ROOTS)
2710    6 FORMAT (1H0,11X,5HROOTS,8X,8HCATEGORY)
2720    7 FORMAT(10X,3A4,I10)
2740      DO 30 I = 1, NR
2750         PRINT 7,(ROOT(I,J), J = 1, 3), CAT(I)
2770   30    CONTINUE
2790      DO 40 I = 1, NC
2800         F(I, 1) = 0.0
2810         F(I, 2) = 0.0
2830   40    CONTINUE
2850      DO 60 I = 1,L
2860         KA = LX(I, 1)
2870         KB = LX(I,2)
2880         KC = LX(I,3)
2890         LL = LGTH(KA, KB, KC)
2900         CALL ROOTER(KINDEX,ROOT, RLENG, NR,KA, KB, KC, LL)
2910         IF(KINDEX .NE. 0) THEN ──────────────────► .EQ.   LOGIC
2920               J = CAT(KINDEX)
2940               DO 50 II=1,2
2950                  F(J,II) = F(J, II) + MSEX(I,II) ──► 1    DATA
2970   50             CONTINUE
2980            ENDIF
3000   60    CONTINUE
3010      PRINT 3
3020    3 FORMAT(///10X,31HCATEGORY  TOTAL   MALE   FEMALE)
3040      DO 90 I = 1,NC
3050         T = F(I,1) + F(I,2) ──────────────────► -    ASSIGNMENT
3060         PRINT  8, I, T, F(I,1), F(I,2)
3080   90    CONTINUE
3090    8 FORMAT (13X,I1,2(4X,F5.0),2X,F5.0)
3100      RETURN
3110      END
```

NOTE: The program is correct as printed. Handwritten changes indicate the errors the participants saw.

Accounting Program

```
      SUBROUTINE TRBAL
      COMMON IACCT1(100),IACCT2(100),IACCT3(100),IACCT4(100),BAL(100),N
      PRINT 400
  400 FORMAT (1H0,20X,25H***** TRIAL BALANCE ***** ///)
      PRINT 410
  410 FORMAT (1H ,5HACCT.,28X,5HDEBIT,9X,6HCREDIT)
      SDEBIT=0.0
      SCRD=0.0
      PRINT 420
  420 FORMAT (1H ,70(1H-))
      DO 480 I =1,N
         IF (BAL(I) .EQ. 0.0) GO TO 480 ------------------> .NE.  LOGIC
            IF (I .GT. 20) GO TO 450
               IF (I .EQ. 4 .OR. I .EQ. 13 .OR.
     &             I .EQ. 15) GO TO 460 ------------------> (1)  DATA
  430             SDEBIT=SDEBIT+BAL(I)
                  PRINT 440, I,IACCT1(I),IACCT2(I),IACCT3(I),IACCT4(I),BAL(I)
  440             FORMAT (1H ,I3,2X,4A4,5X,F12.2)
                  GO TO 480
  450       CONTINUE
            IF (I .GT. 60) GO TO 430
               IF (I .EQ. 53) GO TO 430
  460             SCRD=SCRD+BAL(I) ------------------> +  ASSIGNMENT
                  PRINT 470, I,IACCT1(I),IACCT2(I),IACCT3(I),IACCT4(I),BAL(I)
  470             FORMAT (1H ,I3,2X,4A4,20X,F12.2)
  480    CONTINUE
      PRINT 420
      PRINT 490, SDEBIT,SCRD
  490 FORMAT (1H ,26X,F12.2,3X,F12.2)
      PRINT 420
      PRINT 420
      RETURN
      END
```

Grading Program

```
2694        SUBROUTINE GRADR2(SCORE,PGRADE,FREQ,HIGH,PERCNT,PT,TOTAL)
2695        IMPLICIT INTEGER(A-Z)
2696        COMMON NSTUDN,NASSGN,ID,CURID
2697        DIMENSION SCORE(300,20),PERCNT(5),PGRADE(5),LB(5),
2698     &  TOTAL(100),FREQ(100),ID(300)
2700        PRINT 680
2705    560 FORMAT (1H ,///)
2710    680 FORMAT (1H0,33X,"OVERALL SCORE",3X,"FREQUENCY")
2720        I=HIGH
2730    690 IF (FREQ(I) .GT. 0) PRINT 700, I,FREQ(I)
2740    700 FORMAT (1H0,38X,I3,10X,I3)
2750        I=I-1
2760        IF (I .GT. 0) GO TO 690
2770        PRINT 560
2780        PRINT 710
2790    710 FORMAT (1H0,30X,"LOWER BOUNDS FOR EACH GRADE")
2800        SUM=FREQ(HIGH)
2810        CUT=HIGH
2820        DO 760 PT=1,4
2830    720    QUOTA=IFIX(FLOAT(SUM)/FLOAT(NSTUDN)*100+.5)
2840           IF (QUOTA .GE. PERCNT(PT)) GO TO 740
2850    730       CUT=CUT-1
2860              IF (CUT .LT. 1) GO TO 770
2870                 IF (FREQ(CUT) .LT. 1) GO TO 730
2880                    SUM=SUM+FREQ(CUT)
2890                    GO TO 720
2900    740    LB(PT)=CUT
2910           PRINT 750, PGRADE(PT),LB(PT)
2920    750    FORMAT (1H0,41X,A1,2X,I3)
2930           SUM=0
2940    760    CONTINUE
2950        PT=PT+1
2960        GO TO 790
2970    770 DO 780 I=PT,4
2980    780    LB(PT)=0
2990    790 LB(5)=0
3000        PRINT 750,PGRADE(PT),LB(PT)
3010        PRINT 560
3020        PRINT 800
3030    800 FORMAT (1H0,36X,"FINAL COURSE GRADE")
3040        DO 850 I=1,NSTUDN
3050           PRINT 810, ID(I),(SCORE(I,J),J=1,NASSGN)
3060    810    FORMAT (1H0,31X,I8,20(1X,I3))
3070           DO 820 J=1,   5
3080              IF (TOTAL(I) .GE. LB(J)) GO TO 830
3090    820       CONTINUE
3100    830    PRINT 840, TOTAL(I),PGRADE(J)
3110    840    FORMAT (1H0,32X," OVERALL SCORE  = ",I3,"   GRADE = ",A1)
3120    850    CONTINUE
3130        GTOTAL=0
3140        DO 860 I=1,NSTUDN
3150    860    GTOTAL=GTOTAL+TOTAL(I)
3160        MEAN=IFIX(FLOAT(GTOTAL)/FLOAT(NSTUDN)+.5)
3170        PRINT 870, MEAN
3180    870 FORMAT (1H0,31X,"MEAN SCORE = ",I3)
3190        RETURN
3200        END
```

.LE. LOGIC

\- ASSIGNMENT

(S) DATA

OFFICE OF NAVAL RESEARCH

Code 455

TECHNICAL REPORTS DISTRIBUTION LIST

OSD

CDR Paul R. Chatelier
Military Assistant for Training and
Personnel Technology
Office of the Deputy Under Secretary
of Defense
OUSDRE (E&LS)
Pentagon, Room 3D129
Washington, D.C. 20301

Col. Richard L. Blair
Director of OSD Systems
AFDSC/GN
Pentagon
Washington, D.C. 20330

Mrs. Shields
Director Systems Support
AFDSC/SF
Room 1 Delta 1039
Pentagon
Washington, D.C. 20330

Col. Goetze
AFDSC/(GL)
Pentagon
Washington, D.C. 20330

Col Eglington
Headquarter USAF, AF/PACA
Pentagon
Washington, D.C. 20330

Lt. Col. Patrick L. Harris
AFDSC/GNP Room 2D279
Pentagon
Washington, D.C. 20330

ONR, Code 455, Technical Reports Distribution List

Department of the Navy

Director
Engineering Psychology Programs
Code 455
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217 (5 cys)

Director
Information Systems Program
Code 437
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Director
Physiology Program
Code 441
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Special Assistant for Marine
Corps Matters
Code 100M
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Commanding Officer
ONR Branch Office
ATTN: Dr. J. Lester
Building 114, Section D
666 Summer Street
Boston, MA 02110

Commanding Officer
ONR Branch Office
ATTN: Dr. C. Davis
536 South Clark Street
Chicago, IL 60605

Commanding Officer
ONR Branch Office
ATTN: Dr. E. Gloye
1030 East Green Street
Pasadena, CA 91106

Office of Naval Research
Scientific Liaison Group
American Embassy, Room A-407
APO San Francisco 96503

Director
Naval Research Laboratory
Technical Information Division
Code 2627
Washington, D.C. 20375 (6 cys)

Dr. Bruce Wald
Communications Sciences Division
Code 7500
Naval Research Laboratory
Washington, D.C. 20375

Dr. Robert G. Smith
Office of the Chief of Naval
Operations, OP987H
Personnel Logistics Plans
Washington, D.C. 20350

ONR, Code 455, Technical Reports Distribution List

Department of the Navy

Mr. Arnold Rubinstein
Naval Material Command
NAVMAT 98T24
Washington, D.C. 20360

Commander
Naval Air Systems Command
Human Factors Programs
NAVAIR 340F
Washington, D.C. 20361

Commander
Naval Air Systems Command
Crew Station Design,
NAVAIR 5313
Washington, D.C. 20361

Mr. Phillip Andrews
Naval Sea Systems Command
NAVSEA 0341
Washington, D.C. 20362

Dr. James Curtin
Naval Sea Systems Command
Personnel & Training Analysis Office
NAVSEA 074C1
Washington, D.C. 20362

Commander
Naval Electronics Systems Command
Human Factors Engineering Branch
Code 4701
Washington, D.C. 20360

Capt. Horace M. Leavitt
Naval Electronics Systems Command
Room 554 JP1
Washington, D.C. 20360

Bureau of Naval Personnel
Special Assistant for Research Liaison
PERS-OR
Washington, D.C. 20370

CDR R. Gibson
Bureau of Medicine & Surgery
Aerospace Psychology Branch
Code 513
Washington, D.C. 20372

LCDR Rober Biersner
Naval Medical R&D Command
Code 44
Naval Medical Center
Bethesda, MD 20014

Dr. Arthur Bachrach
Behavioral Science Department
Naval Medical Research Institute
Bethesda, MD 20014

LCDR T. Berghage
Naval Medical Research Institute
Behavioral Sciences Department
Bethesda, MD 20014

ONR, Code, 455, Technical Reports Distribution List

Department of the Navy

Dr. George Moeller
Human Factors Engineering Branch
Submarine Medical Research Lab
Naval Submarine Base
Groton, CT 06340

Dr. H. G. Steubing
Naval Air Development Center
Code 5030
Warminster, PA 18974

Mr. John Rehbehn
NARDAC, Bldg. 196
Washington Navy Yard
Washington, D.C. 20374

Mr. Frank Figlozzi
NARDAC - Code 44 B
Washington Navy Yard
Washington, D.C. 20374

Chief
Aerospace Psychology Division
Naval Aerospace Medical Institute
Pensacola, FL 32512

Dr. Fred Muckler
Navy Personnel Research and
Development Center
Manned Systems Design, Code 311
San Diego, CA 92152

Naval Personnel Research and
Development Center
Code 305
San Diego, CA 92152

Navy Personnel Research and
Development Center
Management Support Department
Code 210
San Diego, CA 92152

CDR P. M. Curran
Human Factors Engineering Division
Naval Air Development Center
Warminster, PA 18974

Mr. Stephen Fikas
NOSC
San Diego, CA 92152

Human Factors Section
Systems Engineering Test
Directorate
U.S. Naval Air Test Center
Patuxent River, MD 20670

Human Factors Engineering Branch
Naval Ship Research and Development
Center, Annopolis Division
Annapolis, MD 21401

Naval Training Equipment Center
ATTN: Technical Library
Orlando, FL 32813

Human Factors Department
Code N215
Naval Training Equipment Center
Orlando, FL 32813

Dr. Alfred F. Smode
Training Analysis and Evaluation
Group
Naval Training Equipment Center
Code N-00T
Orlando, FL 32813

ONR Code 455, Technical Reports Distribution List

Department of the Navy

Dr. Gary Poock
Operations Research Department
Naval Postgraduate School
Monterey, CA 93940

Dean of Research Administration
Naval Postgraduate School
Monterey, CA 93940

Mr. Warren Lewis
Human Engineering Branch
Code 8231
Naval Ocean Systems Center
San Diego, CA 92152

Dr. A. L. Slafkosky
Scientific Advisor
Commandant of the Marine Corps
Code RD-1
Washington, D.C. 20380

Commanding Officer
MCTSSA
Marine Corps Base
Camp Pendleton, CA 92055

Department of the Army

Mr. J. Barber
HQS, Department of the Army
DAPE-PBR
Washington, D.C. 20546

Dr. Joseph Zeidner
Technical Director
U.S. Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Director, Organizations and
Systems Research Laboratory
U.S. Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Dr. Edgar M. Johnson
Organizations and Systems Research
Laboratory
U.S. Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Technical Director
U.S. Army Human Engineering Labs
Aberdeen Proving Ground, MD 21005

ONP, Code 455, Technical Reports Distribution List

Department of the Air Force

U.S. Air Force Office of
Scientific Research
Lite Sciences Directorate, NL
Bolling Air Force Base
Washington, D.C. 20332

Dr. Donald A. Topmiller
Chief, Systems Engineering Branch
Human Engineering Division
USAF AMRL/HES
Wright-Patterson AFB, OH 45433

Air University Library
Maxwell Air Force BAse, AL 36112

Other Government Agencies

Defense Documentation Center
Cameron Station, Bldg. 5
Alexandria, VA 22314 (12 cys)

Dr. Stephen J. Andriole
Director, Cybernetics Technology
Office
Defense Advanced Research Projects
Agency
1400 Wilson Blvd
Arlington, VA 22209

Dr. Stanely Deutsch
Office of Life Sciences
National Aeronautics and Space
Administration
600 Independence Avenue
Washington, D.C. 20546

Director, Information Processing Techniques
DARPA
1400 Wilson Blvd
Arlington, VA 22209

Other Organizations

Dr. William A. McClelland
Human Resources Research Office
300 N. Washington, Street
Alexandria, VA 22314

Dr. Jesse Orlansky
Institute for Defesne Analyses
400 Army-Navy Drive
Arlington, VA 22202

Dr. Arther I. Siegel
Applied Psychological Service, Inc.
404 East Lancaster Street
Wayne, PA 19087

Victor H. Richard
Room 602
Business Administration Bldg.
Pennsylvania State University
University Park, PA 16802

Foreign Addressees

Director, Human Factors Wing
Defense & Civil Institute of
Environmental Medicine
Post Office Box 2000
Downsville, Toronto, Ontario
CANDA