PROGRAM COMPLEXITY AND SOFTWARE ERRORS: A FRONT END FOR RELIABILITY

Dr. Bill Curtis
Software Management Research
Information Systems Programs
General Electric Company
Arlington, Virginia

Error analysis and software complexity have received increased attention in software engineering research over the past several years. The study of software errors has been necessitated by the emphasis on software reliability. Models such as the one presented by John Musa in this volume statistically model such phenomena as the mean-time-between-failures or the probability of a failure within a given unit of time. As John indicates, one of the parameters required as input to this model is the number of errors existing in the software.

There are several ways to estimate the number of errors in a piece of software. One is the actuarial approach which assumes there are so many errors in a given number of lines of code. A number frequently passed about is one error per one hundred lines. This approach assumes that all software is created equal and ignores the advances that have been made during recent years in analyzing software characteristics. An alternative approach recognizes these gains in relating software characteristics to such factors as the error-proneness of a section of code or the difficulty which will be experienced in maintaining the code. The purpose of this paper is to review recent research on software complexity metrics to determine whether knowing something about software characteristics improves our ability to predict the number of errors it contains or the amount of effort required to maintain it.

If we can validate the use of software metrics for predicting the number of errors in software and the difficulty experienced in correcting them, then such metrics will prove a valuable addition to both quality assurance and management information systems. During the design phase, metric values can be estimated from relevant design information to predict problems which will be experienced during coding. Values computed on the actual code can be used in predicting testing results, number of delivered bugs, and ease of maintenance. Although a large number of metrics have been presented in the literature, two seem to have received the most attention in empirical research. I will focus on these two metrics in the remainder of this paper.

Thomas McCabe (1976) developed a complexity measure based on the cyclomatic number from graph theory. McCabe counts the number of regions in a graph of the control flow of a computer program. His metric represents the number of basic control path segments which when combined will generate every possible path through the program. Thus, McCabe has measured the complexity of the control structure. Schneidewind and Hoffmann (1979) demonstrated that the cyclomatic number and the reachability measure which can be computed from it were superior to the number of source statements in predicting the number of errors in a section of code and the time required to find and fix them. Feuer and Fowlkes (1979) also demonstrated that the node count was related to the time to repair errors. However, their data indicated that different prediction equations should be used with different types of errors. Separate prediction equations might be possible when we have (1) developed more robust error classification schemes, and (2) progressed past predicting gross errors to predicting types of errors.

Another approach to software complexity was presented by Maurice Halstead (1977) in his theory of Software Science. Halstead maintained that 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. These quantities can be used to calculate the number of mental comparisons required to generate a program. Halstead's effort metric, E, expresses the complexity of computer software in psychological terms. Halstead also developed a metric to estimate the number of delivered errors in a system. This metric is based on the notion that there is a limited amount of code that a programmer can mentally grasp at a single time. When a section of code exceeds this value it is likely that the programmer made at least one mistake in producing it. Halstead predicts the number of errors by dividing the total volume of code by this critical level for error-prone code.

Bell and Sullivan (1974) presented a scatterplot which suggested that there was some validity to Halstead's notion of a critical value for error-free code. In their data no program with a Halstead volume above 260 was error-free, while only one program below this level had an error. Subsequently, both Cornell and Halstead (1976) and Fitzsimmons and Love (1978) found correlations of 0.75 and above between Halstead's metrics and the number of errors found in various software products. In a debugging study we recently completed at G.E. (Curtis, Milliman, and Sheppard, 1979) the Halstead and McCabe metrics were better predictors of the time required to find a bug than was lines of code.

In studying some error data provided us by Rome Air Development Center, Phil Milliman and I (1979) found Halstead's metric a remarkably accurate predictor of delivered bugs in a system developed with modern programming practices and tools. However, the prediction was poor in a system developed with conventional techniques. The types of errors experienced in the former system were typical when compared to the types of errors reported in other systems (in particular to several reported by TRW). Phil and I also observed that the error ratio reported during the final months of development was an excellent predictor of post-development test errors. The error ratio represents the number of failed runs divided by the total number of runs. We observed a linearly decreasing trend in the error ratio during the final 9 months of development. When we extrapolated this trend into post-development testing, we observed a good prediction of the number of errors detected.

We suspect from the data we have observed that the prediction of errors and maintenance resources will be more accurate on projects guided by modern programming practices. We believe that such practices will reduce the amount of variation in performance and quality resulting from such sources as individual differences among programmers, the programming environment, etc. That is, a structured discipline constrains the amount of variation in the way software is developed. Since this variation is a source of error in predictions, the ability to predict various software-related criteria (such as number of errors) should improve.

Based on the brief review of empirical research presented here, I propose the following conclusions, but agree that much more data is needed to substantiate them.

- Measures of software characteristics can be used to predict the number of errors in a portion of code and the effort required to find and correct them. Such measures will be more valuable than an actuarial approach based on lines of code.

- Different predictive plots may be observed for different classes of errors (computational, logic, interface, etc.)

- Metrics should be calculated at the appropriate level (subroutine, module, etc.) for explaining the results.

- The prediction of software reliability and of maintenance requirements can begin early in the software development cycle, and improvements can be made and monitored if feedback is provided for improving software quality.

ACKNOWLEDGEMENTS

I would like to thank Sylvia Sheppard and Elizabeth Kruesi for their comments, Beverly Day for manuscript preparation, and Lou Oliver for his support and encouragement. Work resulting in this paper was supported by the Office of Naval Research, Engineering Psychology Programs (Contract #N000014-79-C-0595) and the General Electric Company (IR&D Project 79D6A02). However, the opinions expressed in this paper are not necessarily those of the Department of the Navy or the General Electric Company.

REFERENCES

Bell, D. E. and J. E. Sullivan, Further investigations into the complexity of software (Tech. Rep. MTR-2874). Bedford, MA: MITRE, 1974.

Cornell, L. M. and M. H. Halstead, Predicting the number of bugs expected in a program module (Tech. Rep. CSD-TR-205). West Lafayette, IN: Purdue University, Computer Science Department, 1976.

Curtis, B. and P. Milliman, A matched project evaluation of modern programming practices (RADC-TR-79, 2 vols.). Griffiss AFB, NY: Rome Air Development Center, 1979.

Curtis, B., S. B. Sheppard, and P. Milliman, Third time charm: Stronger prediction of programmer performance by software complexity metrics. In Proceedings of the Fourth International Conference on Software Engineering. New York: IEEE, 1979.

Feuer, A. R. and E. B. Fowlkes, Some results from an empirical study of computer software. In Proceedings of the Fourth International Conference on Software Engineering, New York: IEEE, 1979.

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

Halstead, M. H., Elements of Software Science. New York: Elsevier North-Holland, 1977.

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

Schneidewind, N. F. and H. M. Hoffmann, An experiment in software error data collection and analysis. IEEE Transactions on Software Engineering, 1979, 5, 276-286.

GENERAL ELECTRIC COMPANY

SPACE DIVISION

NEEDS RELATING TO ERROR ANALYSIS

INFORMATION SYSTEMS PROGRAMS

SOFTWARE MANAGEMENT RESEARCH

NEEDS	USES
PREDICTORS OF THE NUMBER OF ERRORS RESIDENT IN A PORTION OF CODE	INPUTS INTO SOFTWARE RELIABILITY MODELS
PREDICTORS OF THE TIME REQUIRED TO FIND AND CORRECT SOFTWARE ERRORS	ESTIMATION OF TESTING AND MAINTENANCE RESOURCES

GENERAL ELECTRIC COMPANY

SPACE DIVISION

CANDIDATE PREDICTORS

INFORMATION SYSTEMS PROGRAMS

SOFTWARE MANAGEMENT RESEARCH

ACTUARIAL DATA

SOFTWARE CHARACTERISTICS

- CYCLOMATIC NUMBER
- SOFTWARE SCIENCE

DOES KNOWING SOMETHING ABOUT THE CHARACTERISTICS OF THE CODE IMPROVE OUR ABILITY TO PREDICT THE NUMBER OF ERRORS IT CONTAINS?

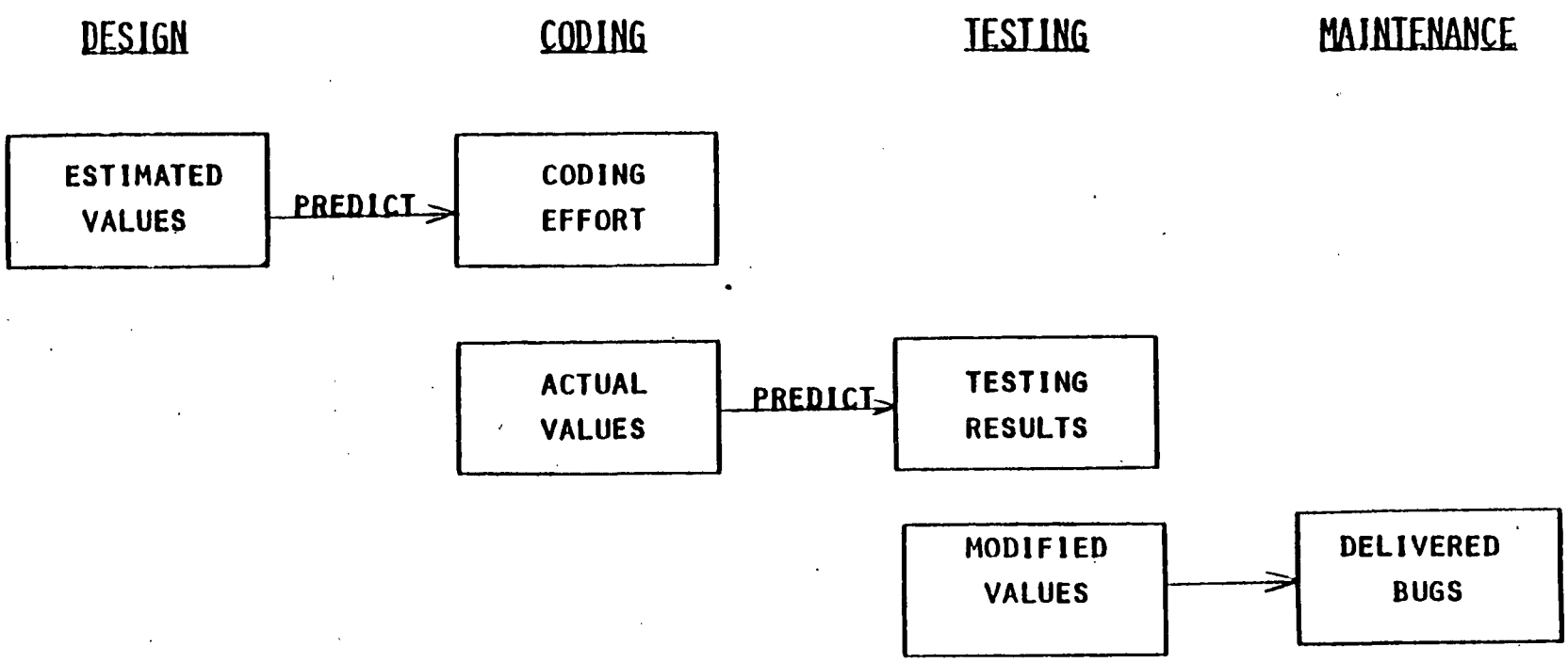
GENERAL ELECTRIC COMPANY

SPACE DIVISION

THE USE OF SOFTWARE METRICS IN A MANAGEMENT INFORMATION SYSTEM

INFORMATION SYSTEMS PROGRAMS

SOFTWARE MANAGEMENT RESEARCH



GENERAL ELECTRIC COMPANY

SPACE DIVISION

THOMAS J. McCABE
A COMPLEXITY MEASURE (1976)

INFORMATION SYSTEMS PROGRAMS

SOFTWARE MANAGEMENT RESEARCH

EQUATION:

$v(G) = \#\ \text{EDGES} - \#\ \text{NODES} + 2(\#\ \text{CONNECTED COMPONENTS})$

OR

$v(G) = \#\ \text{PREDICATE NODES} + 1$

OR

$v(G) = \#$ REGIONS IN A PLANAR GRAPH OF THE CONTROL FLOW.

DESCRIPTION:

McCABE'S METRIC REPRESENTS THE NUMBER OF LINEARLY INDEPENDENT CONTROL PATHS COMPRISING A PROGRAM. THAT IS, THE NUMBER OF BASIC CONTROL PATH SEGMENTS WHICH WHEN COMBINED WILL GENERATE EVERY POSSIBLE PATH THROUGH THE PROGRAM. McCABE'S v(G) REPRESENTS A MEASURE OF COMPUTATIONAL COMPLEXITY.

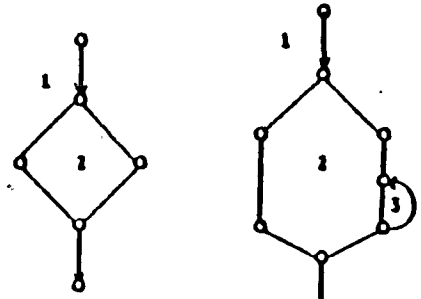
GENERAL ELECTRIC COMPANY

SPACE DIVISION

COMPUTATION OF McCABE'S v(G)

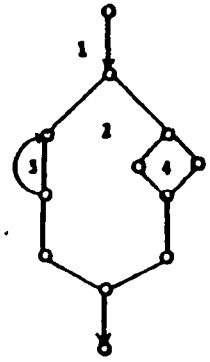
INFORMATION SYSTEMS PROGRAMS

SOFTWARE MANAGEMENT RESEARCH

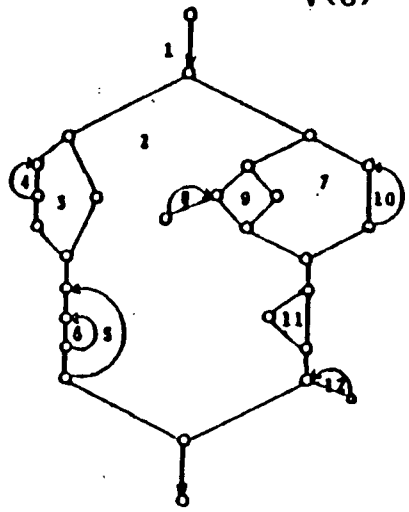


v(G) = 2

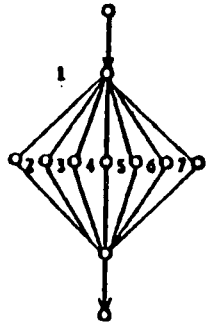
v(G) = 3



v(G) = 4



v(G) = 12



v(G) = 7

GENERAL ELECTRIC COMPANY

SPACE DIVISION

SCHNEIDEWIND AND HOFFMANN'S DATA (1979)

INFORMATION SYSTEMS PROGRAMS

SOFTWARE MANAGEMENT RESEARCH

PREDICTOR	NUMBER OF PROCEDURES	CORRELATIONS		
		# OF ERRORS	FIND TIME	FIX TIME
CYCLOMATIC NUMBER	31	.78	.67	.72
REACHABILITY	20	.77	.90	.66
SOURCE STATEMENTS	20	.59	.59	.51

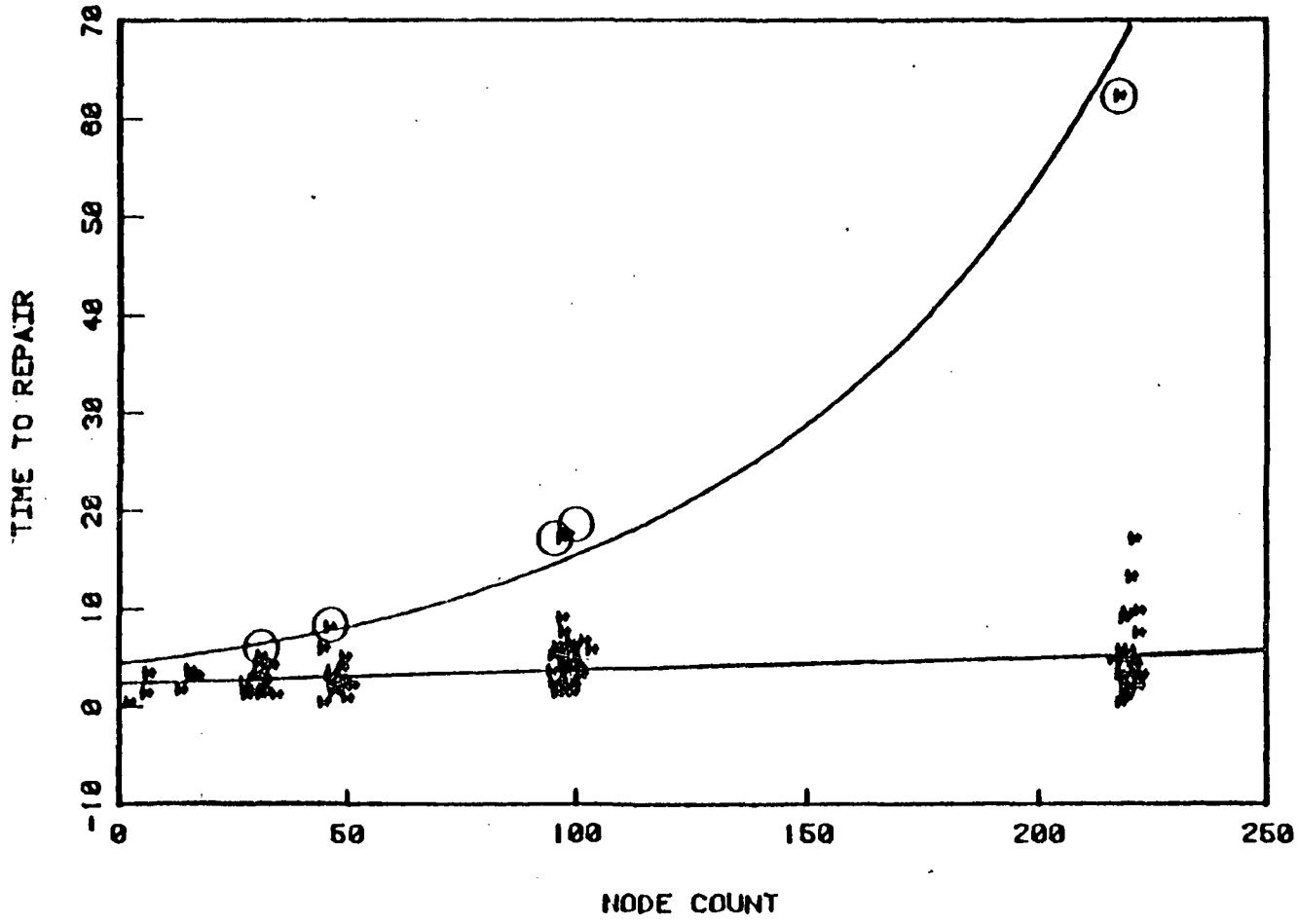
GENERAL ELECTRIC COMPANY

SPACE DIVISION

FEUER AND FOWLKES DATA (1979)

INFORMATION SYSTEMS PROGRAMS

SOFTWARE MANAGEMENT RESEARCH



GENERAL ELECTRIC COMPANY

SPACE DIVISION

MAURICE H. HALSTEAD

ELEMENTS OF SOFTWARE SCIENCE (1977)

INFORMATION SYSTEMS PROGRAMS

SOFTWARE MANAGEMENT RESEARCH

EQUATION:

$$E = \frac{\eta_1 N_2 (N_1 + N_2) \text{Log}_2 (\eta_1 + \eta_2)}{2\eta_2}$$

WHERE,

η_1 = # OF UNIQUE OPERATORS

η_2 = # OF UNIQUE OPERANDS

N_1 = F OF OPERATORS

N_2 = F OF OPERANDS

DESCRIPTION:

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. THESE QUANTITIES CAN BE USED TO CALCULATE THE NUMBER OF MENTAL COMPARISONS REQUIRED TO GENERATE A PROGRAM. HALSTEAD'S EFFORT METRIC, E, EXPRESSES THE COMPLEXITY OF COMPUTER SOFTWARE IN PSYCHOLOGICAL TERMS.

GENERAL ELECTRIC COMPANY

SPACE DIVISION

HALSTEAD'S MEASURE OF DELIVERED BUGS

INFORMATION SYSTEMS PROGRAMS

SOFTWARE MANAGEMENT RESEARCH

$$B = V/E_{CRIT}$$

$$= \frac{V_\lambda}{13,824}$$

WHERE,

V = VOLUME

E_{CRIT} = THE MEAN NUMBER OF ELEMENTARY DISCRIMINATIONS BETWEEN POTENTIAL ERRORS IN PROGRAMMING

λ = LEVEL OF THE IMPLEMENTATION LANGUAGE

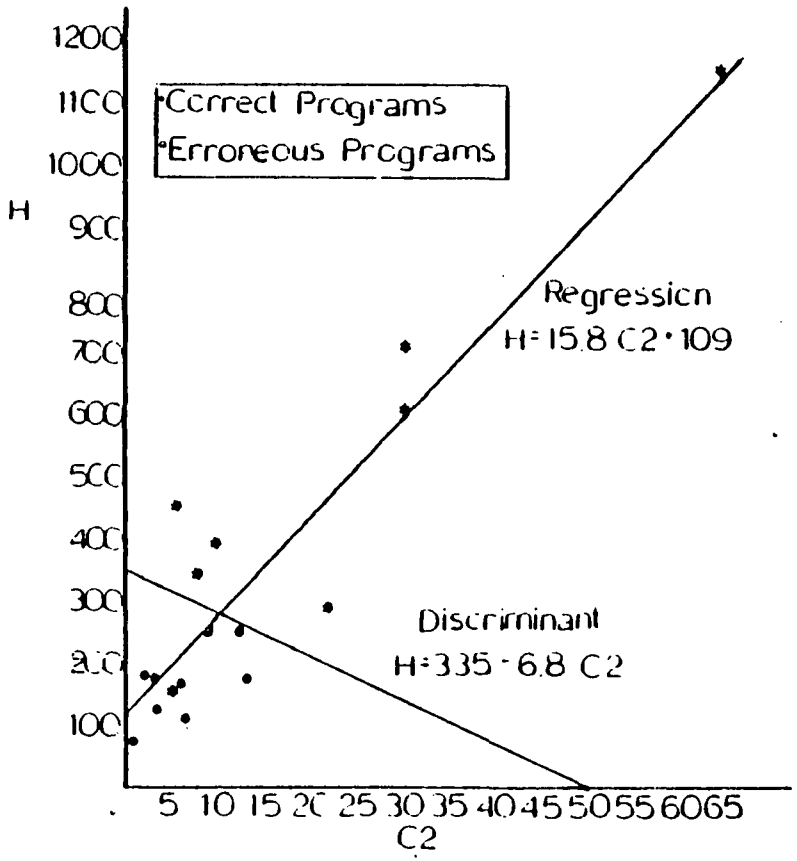
GENERAL ELECTRIC COMPANY

SPACE DIVISION

BELL AND SULLIVAN'S DATA (1974)

INFORMATION SYSTEMS PROGRAMS

SOFTWARE MANAGEMENT RESEARCH



GENERAL ELECTRIC COMPANY

SPACE DIVISION

CORNELL AND HALSTEAD'S DATA (1976)

INFORMATION SYSTEMS PROGRAMS

SOFTWARE MANAGEMENT RESEARCH

MILLIONS OF MENTAL DISCRIMINATIONS	NUMBER OF ERRORS	
	ACTUAL	PREDICTED
170.3	102	102
15.3	18	20
322.6	146	156
28.2	26	30
100.2	71	71
65.5	37	54
6.5	16	11
58.5	50	50
135.9	80	88
903.0	546	582

R = .99

GENERAL ELECTRIC COMPANY

SPACE DIVISION

FITZSIMMONS AND LOVE'S DATA (1978)

INFORMATION SYSTEMS PROGRAMS

SOFTWARE MANAGEMENT RESEARCH

SUBSYSTEM	NUMBER OF MODULES	RANGE OF STMTS PER MODULE	TOTAL STMTS	CORRELATION OF E WITH ERRORS
COMMAND EXECUTIVE	47	70-7100	53,920	.81
DATABASE MANAGER	42	10-6050	64,910	.75
REPORT GENERATOR	51	50-3700	47,450	.75
TOTAL	140	10-7100	166,280	.77

GENERAL ELECTRIC COMPANY

SPACE DIVISION

CURTIS, SHEPPARD, & MILLIMAN'S DATA (1979)

INFORMATION SYSTEMS PROGRAMS

SOFTWARE MANAGEMENT RESEARCH

	E	V(G)	LENGTH
INTERRELATIONSHIPS			
V(G)	.76***		
LENGTH	.56***	.90***	
TIME TO FIND BUG:			
TOTAL PROGRAM	.75***	.65***	.52***
SUBROUTINE	.66***	.63***	.67***

NOTE: N = 27

*** $P \leq .001$

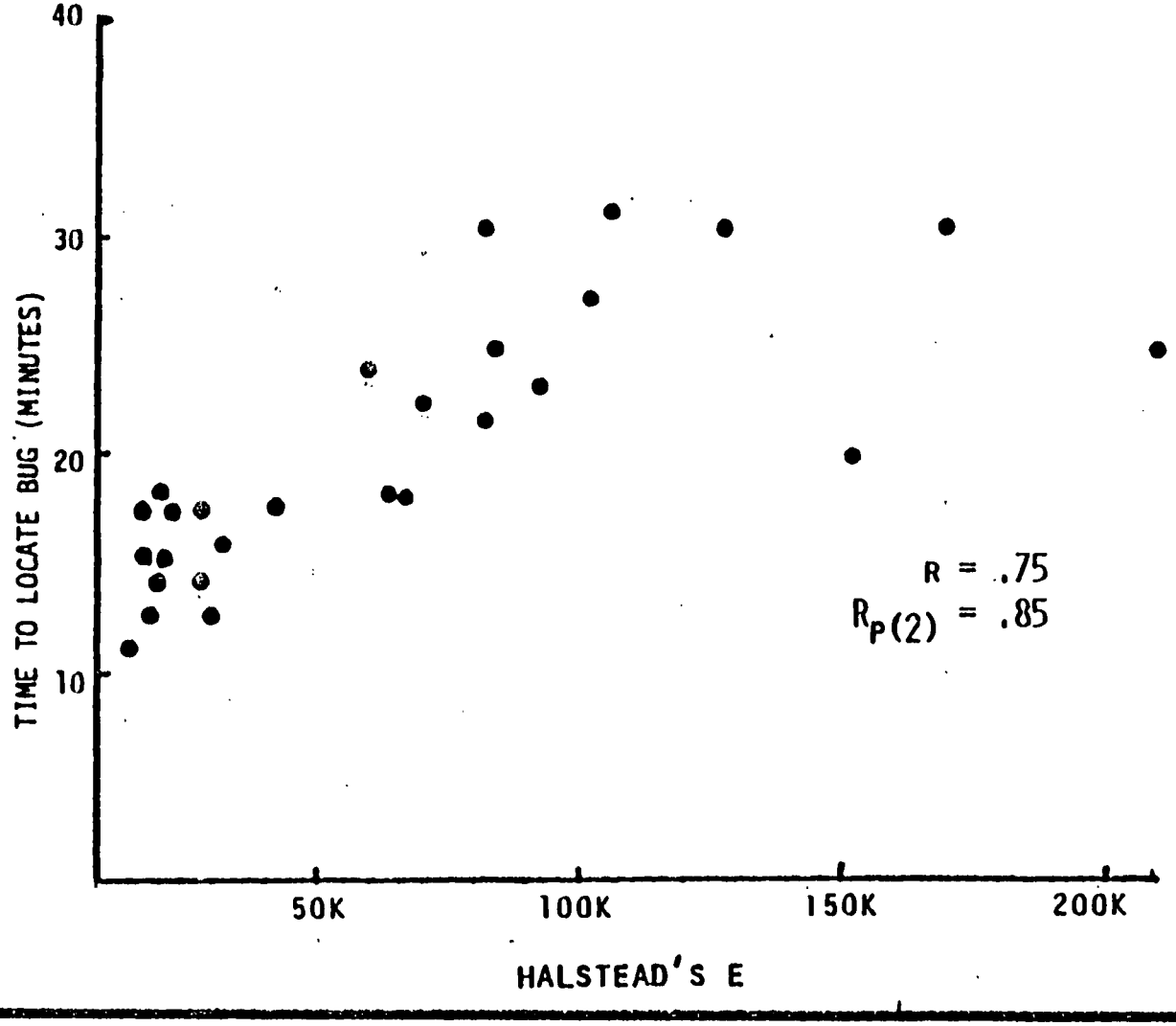
GENERAL ELECTRIC COMPANY

SPACE DIVISION

SCATTERPLOT OF HALSTEAD'S E WITH DEBUGGING TIME

INFORMATION SYSTEMS PROGRAMS

SOFTWARE MANAGEMENT RESEARCH



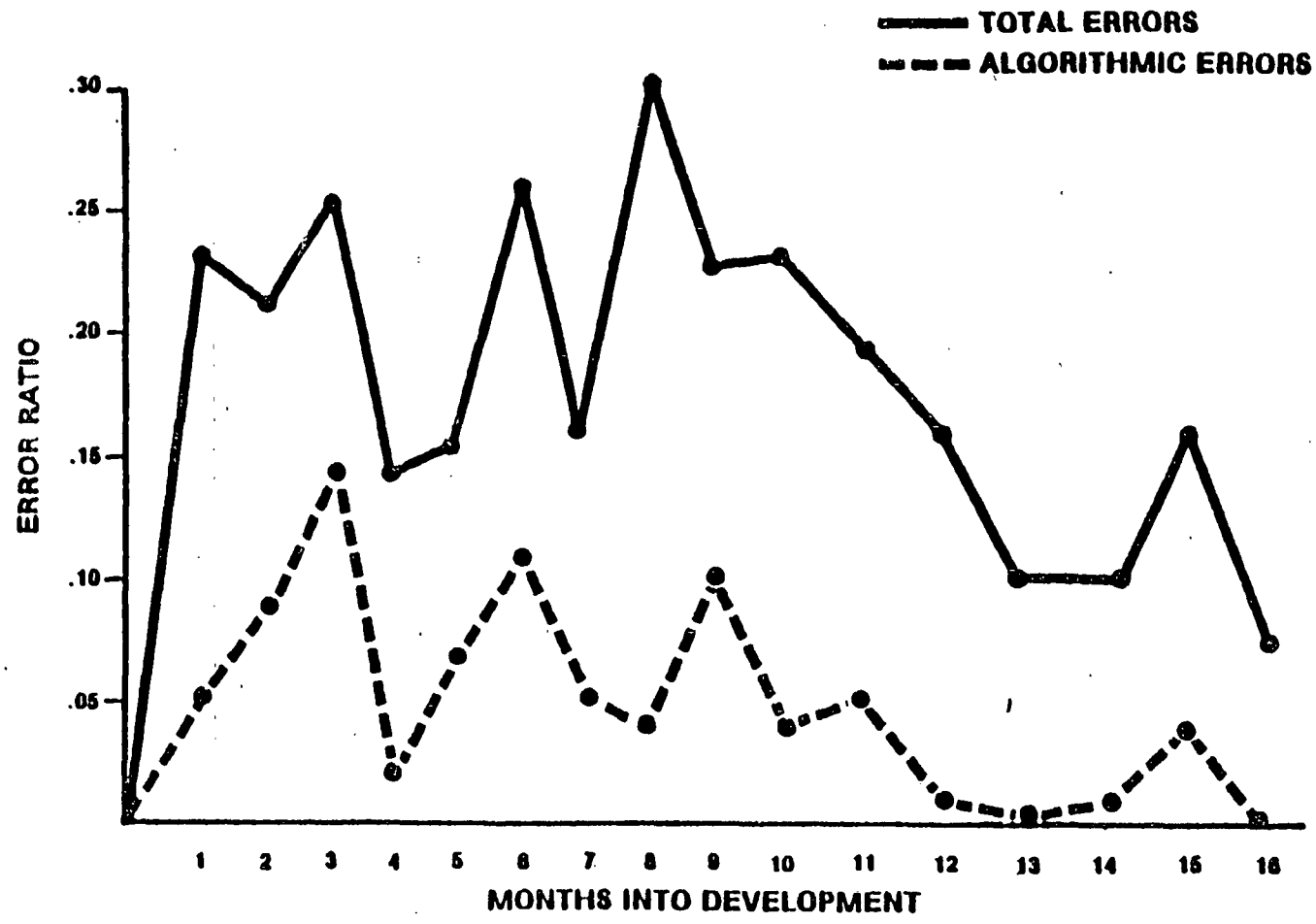
GENERAL ELECTRIC COMPANY

SPACE DIVISION

CURTIS AND MILLIMAN'S DATA (1979)
ERROR RATIO BY MONTH

INFORMATION SYSTEMS PROGRAMS

SOFTWARE MANAGEMENT RESEARCH



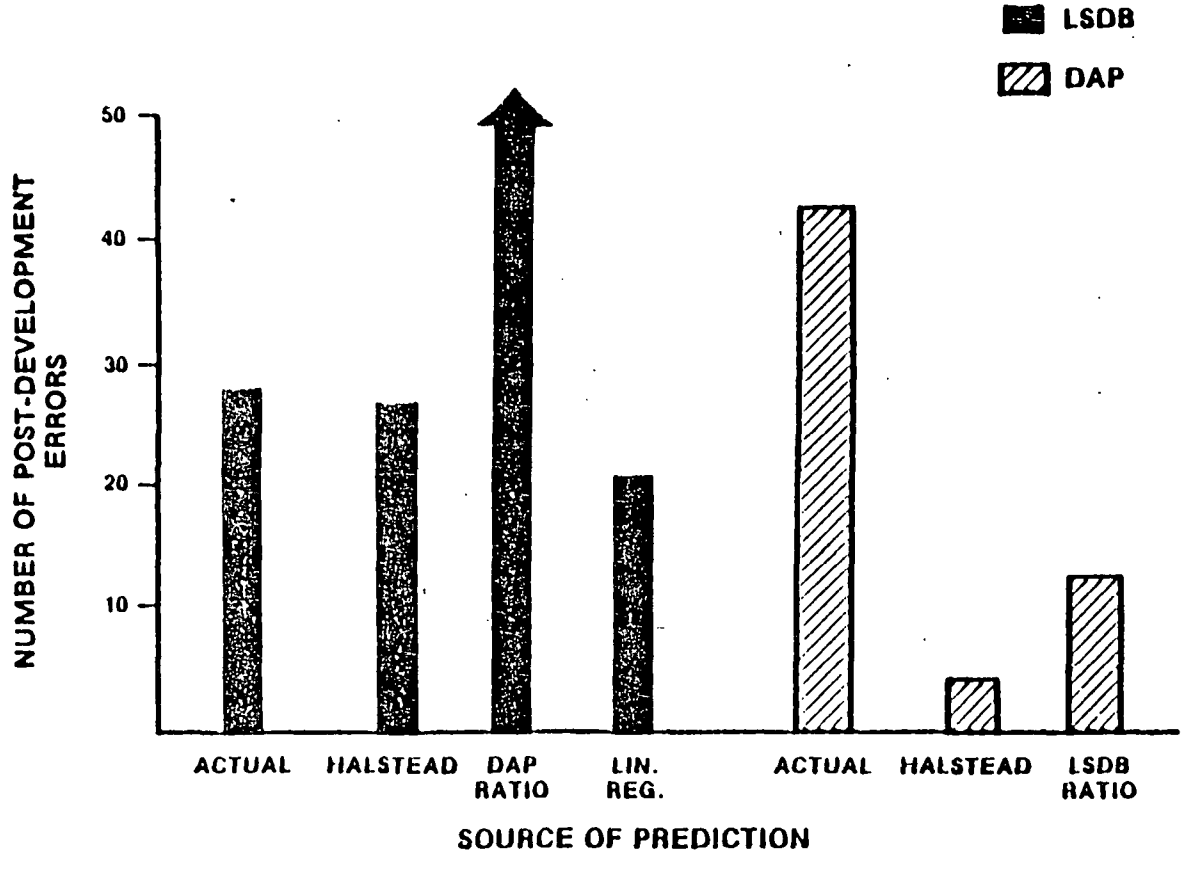
GENERAL ELECTRIC COMPANY

SPACE DIVISION

PREDICTION OF POST-DEVELOPMENT ERRORS

INFORMATION SYSTEMS PROGRAMS

SOFTWARE MANAGEMENT RESEARCH



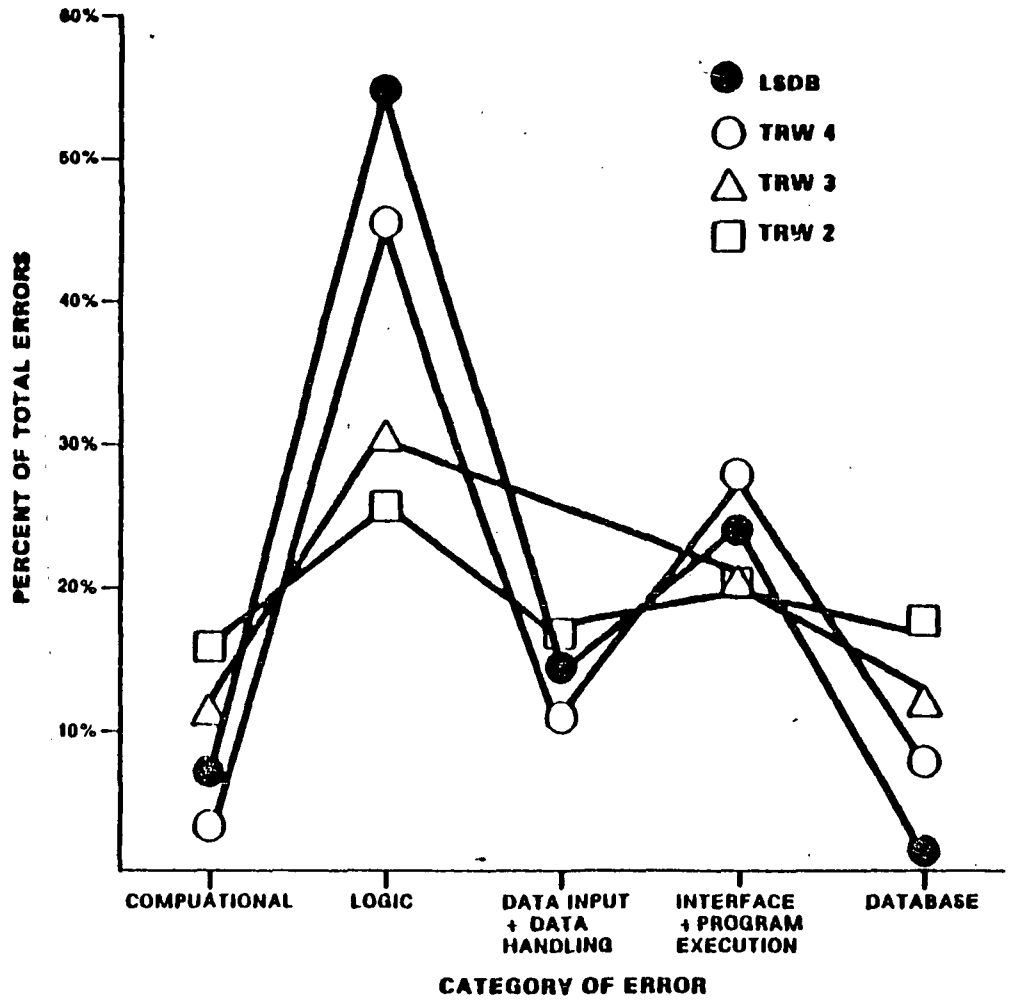
GENERAL ELECTRIC COMPANY

SPACE DIVISION

COMPARISON OF ERROR DISTRIBUTIONS

INFORMATION SYSTEMS PROGRAMS

SOFTWARE MANAGEMENT RESEARCH



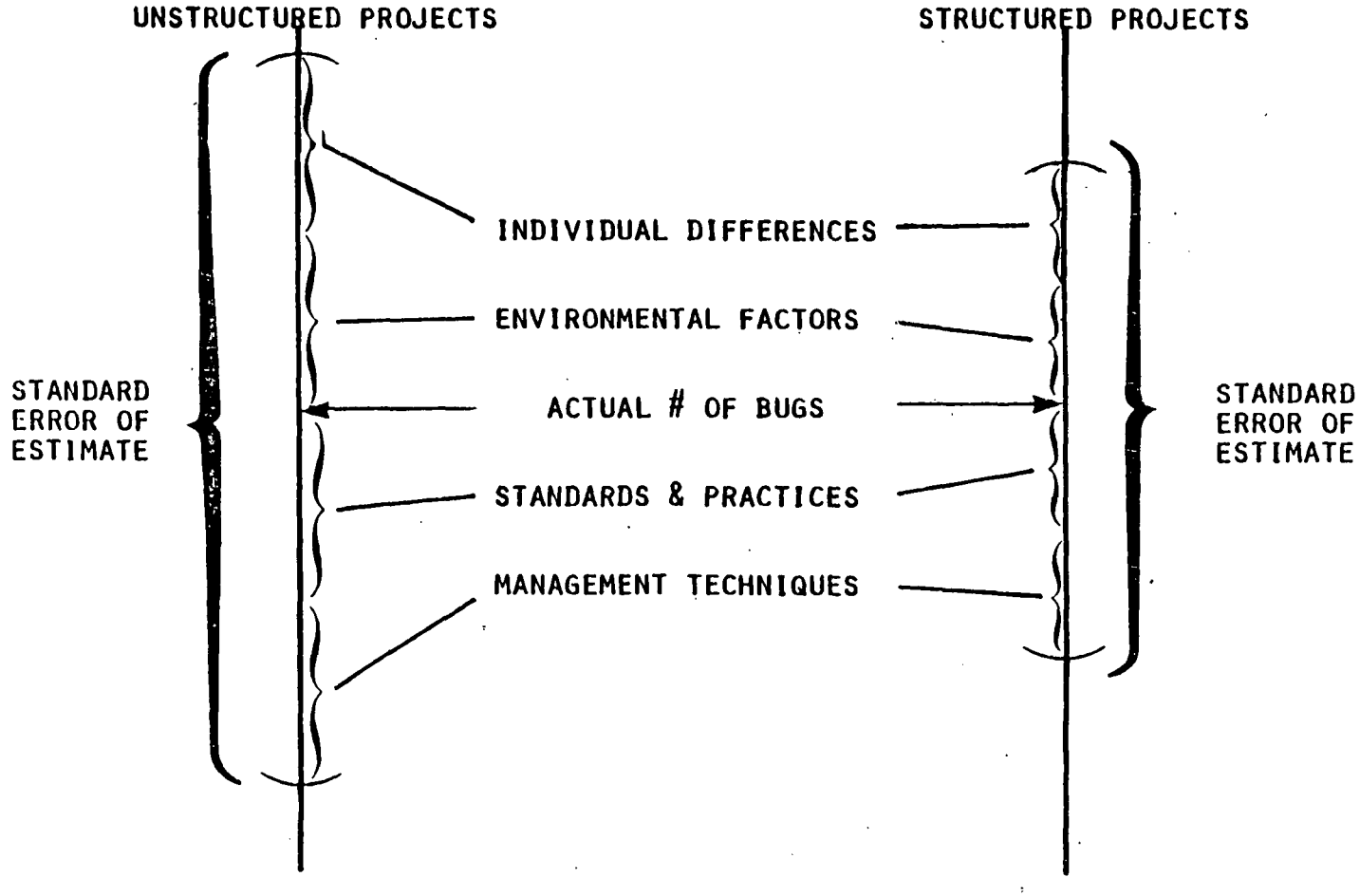
GENERAL ELECTRIC COMPANY

SPACE DIVISION

FACTORS INFLUENCING THE ACCURACY OF PREDICTION

INFORMATION SYSTEMS PROGRAMS

SOFTWARE MANAGEMENT RESEARCH



GENERAL ELECTRIC COMPANY

SPACE DIVISION

CONCLUSIONS

INFORMATION SYSTEMS PROGRAMS

SOFTWARE MANAGEMENT RESEARCH

- MEASURES OF SOFTWARE CHARACTERISTICS CAN BE USED TO PREDICT THE NUMBER OF ERRORS IN A PORTION OF CODE AND THE EFFORT REQUIRED TO FIND AND CORRECT THEM
- DIFFERENT PREDICTIVE PLOTS WILL BE OBSERVED FOR DIFFERENT CLASSES OF ERRORS
- THERE ARE OPTIMAL LEVELS IN THE CODE FOR CALCULATING METRICS
- THE PREDICTION OF SOFTWARE RELIABILITY AND MAINTENANCE REQUIREMENTS CAN BEGIN EARLY IN THE SOFTWARE DEVELOPMENT CYCLE, AND IMPROVEMENTS CAN BE MONITORED