NASA-CR-158960

Final Report

The Determination of Measures of Software Reliability

(NASA-CR-158960) THE DETERMINATION OF MEASURES OF SOFTWARE RELIABILITY Final Report (Aerospace Corp., El Segundo, Calif.) 118 p HC A06/MF A01 CSCL 09B N79-15674
Unclas
G3/61 42154

F. D. MAXWELL

THE AEROSPACE CORPORATION
El Segundo, Calif. 90245

Prepared for
NASA Langley Research Center
Hampton, Virginia
under Contract NAS1-14392

National Aeronautics and
Space Administration

Scientific and Technical
Information Office
1978

Report No.
NASA-CR-158960
ATR-79(7590)-1

THE DETERMINATION OF MEASURES OF SOFTWARE RELIABILITY

Prepared by

F. D. Maxwell

December 1978

Advanced Programs Division
THE AEROSPACE CORPORATION
El Segundo, Calif. 90245

Prepared for

NASA Langley Research Center
Hampton, Virginia

Contract No. NAS1-14392

1. Report No. NASA-CR-158960	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle The Determination of Measures of Software Development		5. Report Date December 1978
		6. Performing Organization Code
7. Author(s) F. D. Maxwell B. C. Corn		8. Performing Organization Report No.
		10. Work Unit No.
9. Performing Organization Name and Address The Aerospace Corporation El Segundo, California		11. Contract or Grant No. NASA1-14392
		13. Type of Report and Period Covered Final, August 1978
12. Sponsoring Agency Name and Address NASA Langley Research Center Hampton, Virginia		14. Sponsoring Agency Code
15. Supplementary Notes		

16. Abstract

Measurement of software reliability was carried out during the development of data base software for a multi-sensor tracking system. The failure ratio and failure rate were found to be consistent measures. Trend lines could be established from these measurements that provide good visualization of the progress on the job as a whole as well as on individual modules. Over one-half of the observed failures were due to factors associated with the individual run submission rather than with the code proper.

Possible application of these findings for line management, project managers, functional management, and regulatory agencies is discussed. Steps for simplifying the measurement process and for use of these data in predicting operational software reliability are outlined.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Software reliability Reliability measurement Software failure ratio Software failure rate Software reliability trend Error types	Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
Unclassified	Unclassified	123	

* For sale by the National Technical Information Service, Springfield, Virginia 22161

Report No.
ATR-79(7590)-1

THE DETERMINATION OF MEASURES OF SOFTWARE RELIABILITY

PREPARED BY:

F. D. Maxwell, Manager
Statistical Design and Analysis Section
Electronics and Optics Division

APPROVED BY:

B. C. Corn

B. C. Corn, Director
Digital Processing Office
Development Directorate

ACKNOWLEDGEMENTS

The work reported here was performed by The Aerospace Corporation under Contract No. NAS1-14392 with the NASA Langley Research Center under the technical guidance of Mr. G.E. Migneault. It utilized data collected from a software development project sponsored jointly by the USAF Rome Air Development Center, with Mr. Frank Sliwa as the Project Engineer, and the Metric Integrated Processing System (MIPS) team at the USAF Space and Missile Test Center under the direction of Mr. J. A. Salazar. The contractor for the development of the pertinent MIPS segments is Federal Electric Corporation.

Acknowledgement is given to Dr. E. Pate who contributed to the computer program implementation of the statistical analysis. Much valuable assistance was received from Mr. Sam W. Beddingfield and Ms. Ruth Pervorse in the preparation of this manuscript.

PRECEDING PAGE BLANK NOT FILMED

SUMMARY

The objective of this study is to establish the feasibility, cost, and benefits of software reliability measurement in a specific environment, and to formulate from this study recommendations for more general applications. The ultimate goal of the entire program is the determination of software failure rate parameters analogous to hardware failure rate or wear-out rate parameters.

During the first phase of this effort, data collected on categories of errors encountered during a software development was analyzed to determine if consistent measures could be derived to use as valid indicators of reliability. The failure ratio (number of failed runs F observed in N total runs in a given elapsed time) and the failure rate (number of failed runs F observed in t seconds of CPU time in a given elapsed calendar time) demonstrated qualification as such measures.

The principal effort since the last report has been to apply rigorous statistical analyses and non-parametric methods to the existing data base. Linear and non-linear orthogonal polynomial regression analyses confirmed the validity of the failure rate and ratio as potential measures of reliability. A high positive statistical correlation was shown between failure severity and error category, failure severity and error count,

and error category with error count. In addition, a preliminary investigation into reliability forecasting showed the ensemble averages of both the failure rate and ratio are stationary and statistically significant.

The failure rate and ratio measures appear to remain valid indicators when subjected to the parametric and non-parametric analyses described in this report. The preliminary attempt at forecasting was statistically valid but, this, of course, needs to be validated by real-world observations. Problems encountered in data collection due to lack of direct control over the process highlighted the need for formalizing this critical portion of any future effort even if the cost increases. Operational avionics systems should provide a superior source of failure data since the recording of such information is routinely performed as a part of aircraft maintenance by personnel other than the software staff.

CONTENTS

LIST OF FIGURES AND TABLES

LIST OF FIGURES AND TABLES

LIST OF FIGURES AND TABLES

1. INTRODUCTION

This report summarizes work performed at The Aerospace Corporation on a software reliability measurement study [illegible] th Langley Research Center, National Aeronautics and Space Administration, under Contract NAS1-14392. The specific objective of the study is to establish the feasibility, cost, and benefits of such measurement in a specific environment, and to formulate from this study recommendations for more general applications of software reliability measurement directed towards the goal of the determining of software failure rate parameters analogous to hardware failure rate parameters. A collateral objective is the identification of any other factors possibly contributing to software reliability that might be observed during the course of the data collection and analysis effort.

This study was initiated in April, 1976. The work accomplished between April, 1976 and April, 1977 was reported in September, 1977 in NASA Contractor Report 14[illegible]205[3]. This report covers the work performed between April, 1977 and June, 1978.

Data analyzed in this study came from two sources: (1) Project ASTROS (Advanced Systematic Techniques for Reliable Operational Software)[3], a joint effort of the Space and Missile Test Center (SAMTEC) and the Rome Air Development

Center (RADC), both organizations within the Air Force Systems Command; and (2) the NASA Viking Program at the Jet Propulsion Laboratory.

For the purpose of this study we have defined reliable software as follows:

> It is software that is correct (capable of execution and yielding correct results) and that meets other user requirements such as timing and interfacing with the environment.

This concept is consistent with an earlier statement, "Software possesses reliability to the extent that it can be expected to perform its intended functions satisfactorily."[1] There is justifiable concern about attempting to base measurement on "intended functions", but more restrictive formulations tend to prevent recognition of reliability problems arising from poorly drawn specifications. A need exists to evaluate software reliability against formally specified, as well as against more loosely defined or implied requirements.

For reliability measurement, the software is operated over a period of time; segments of the operation are scored as failure or success by the qualitative criteria cited above; and, from these scores, an indicator of measured reliability is generated.

The principal indicators derived from he data are the failure ratio and the failure rate. The failure ratio, U, is defined as

$$U = F/N \tag{1}$$

where F is the number of failures observed in N runs in a given calendar period, usually one month. The failure rate, u, is defined as

$$u = f/t \tag{2}$$

where f is the number of failures observed during the total CPU time, t seconds, accumulated over a given calendar period, again usually one month. These failure metrics, and particularly their complement, the reliability metrics,

$$R = 1 - U = S/N \tag{3}$$

where S stands for the number of successes and

$$MTBF = t/f \tag{4}$$

are analogous to commonly used hardware reliability expressions. The relation of these metrics to those used by other researchers in software reliability is described elsewhere.[2]

The failure ratio and the failure rate are obtainable from records usually maintained in the development of critical software; they are consistent in time and among modules for the specific program studied; and they are potentially useful for management and research purposes.

The use of the failure ratio, i.e., the ratio of failed runs to total runs in a given period of time, as a measure of software reliability is one of the innovations introduced in this study. Previous investigators had simply reported the number of failures per calendar interval. To the extent that the number of runs per month (or other interval) is not uniform, these measures will yield different results. For most purposes, the measure that will be preferred is the one that has the smallest variability. In the earlier report on this study it was shown that the failure ratio affords a more stable measure of reliability.

In the course of the study it was observed that many runs ended in failure due to improper data setups, job control cards, or other factors not directly associated with the code developed. By counting as failures only those runs in which the cause of the failure resided in the program proper, we generated the program failure ratio.

Both the total failure ratio and the program failure ratio exhibit a general trend with time. By the use of regression, trend lines can be generated for the development period and/or for the most recent intervals to provide indicators of progress or lack of progress. The generation and use of these trend lines is discussed in the previous report.[3]

The principal effort since the last report has been to verify the validity of these measures by more rigorous statistical analyses and to determine if meaningful correlations could be observed between variables existing in the data base. Linear and non-linear orthogonal polynomial regression analyses corroborate the effective use of the failure rate and ratio as measures of reliability. A high positive correlation was shown between failure severity and error category, failure severity and error count, and error category with error count. In addition, a preliminary investigation into reliability forecasting showed that the ensemble averages of both the failure rate and ratio are stationary and the confidence limits were defined.

The failure rate and ratio measures appeared to remain valid indicators when subjected to the parametric and non-parametric analyses described in this report. The methods for analyses that were developed may be generalized to a broad class of problems; however, the specific results should only be generalized to comparable data bases.

Problems encountered in data collection due to lack of direct control over the process highlight the need for formalizing this critical portion of any future effort. Operational avionics systems should provide a superior source of failure data since the recording of such information is routinely performed as a part of aircraft maintenance by personnel other than the software staff.

During this study, a search of the literature for generalized models of software reliability was conducted. The bibliography resulting from this search is contained in Appendix D.

2. BACKGROUND

As noted earlier the data for this study came from two sources: (1) Project ASTROS at the Space and Missile Test Center; and (2) NASA's Viking program at the Jet Propulsion Laboratory. These data bases are briefly described in the following sections.

2.1 Project ASTROS Data

The ASTROS data that was analyzed in this report was collected during the development of the Launch Support Data Base (LSDB), a portion of the Metric Integrated Processing System (MIPS). MIPS provides the primary metric (i.e., positional) data processing for test or trajectory measurement activities on missiles, aircraft, and satellites. MIPS includes control, real-time, and non-real-time segments. LSDB is a non-real-time segment that includes data management functions, coordinate transformations, and other scientific calculations supporting track generation from multiple sources. It is run prior to launch operations without real-time constraints.

The design of LSDB was started in September, 1975. The software failure data was collected during development of the LSDB from the initial coding through the in-house test phases prior to acceptance by the government. During the development, the number of lines of code continually increased as runs were being made, and the effect of these changes on the reliability measurements is discussed later in this report. During program development there was no unusual pressure to control reliability for current runs, but there was adherence to normal standards for reliable software.

LSDB was developed as part of a demonstration program on structured programming techniques. Personnel appeared to be motivated by their participation in such a demonstration, and the data collection efforts and management attention may have constituted confounding human factors that affected both the data and the measurements.

The MIPS system specification required a modular program structure, hierarchical program design, and execution ordered programming. In addition to these overall requirements, the decision was made to create a highly disciplined programming environment for portions of the non-real-time segment that include the LSDB. This environment included the following:

a. top-down development
b. structured code
c. program support library
d. chief programmer teams
e. structured walk-throughs.

The data accumulated for the evaluation phase provided a unique opportunity to conduct software reliability measurements during program development.

The LSDB program is composed of five major components (here referred to as "modules") consisting of approximately 40 independent subroutines (referred to as "utilities"). The modules, linked with controls, are illustrated in Figure 2-1. The entire LSDB Program comprises approximately 25,000 lines of FORTRAN source code, of which the modules account for about 18,000 lines. Of the total, approximately 40 percent of the module code consists of comments. Most of the LSDB code was written in structured FORTRAN, translated into ANSI FORTRAN by means of the S-FORTRAN precompiler, and then compiled on an IBM 360/65 computer. Small segments were written in the IBM assembly language (BAL). Originally, five programmers were assigned to LSDB. After a few months, the participation was reduced to a staff of three plus a programmer-librarian.

SAMTEC Data Documentation

For every run made on LSDB, a run analysis report form was completed that listed the date, the module name, CPU time for the run, and coded information on the number of changes and run steps as shown in Appendix B. The run was scored as a success or failure by the development group. If a run was identified as a failure, additional information, contained in the failure analysis report, was provided identifying the type

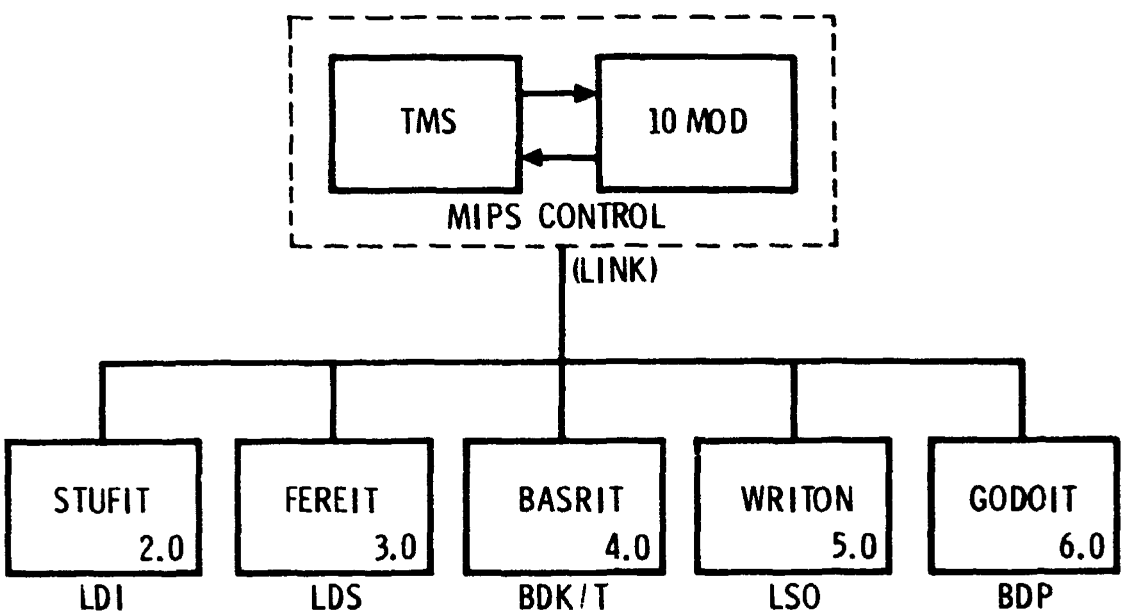

Figure 2-1. MIPLSD Visual Table of Contents

and cause of failure. This form was also prepared by the program development personnel. This form is the second exhibit in Appendix B.

It was not known a priori what factors in the programming and computer system environment might affect software reliability. For that reason, the stipulated requirements for the software product (here LSDB) as well as a description of the general environment was included as part of the record of this Software Reliability Measurement Study. Forms for reporting this background information are reproduced in Appendix B. The primary use intended for this information is for future comparative evaluation of the reliability measurements on LSDB with those from other sources. It is hoped that analytic information about the effects of programming, test, and management techniques can be gained from such comparisons.

Data were received from SAMTEC through June of 1977 when the developing contractor's contractual obligation to collect the data ended.

2.2 Viking Data

In order to establish an additional source of data, the cooperation of the Jet Propulsion Laboratory staff responsible for Viking ground data processing was solicited and received. This system was fully operational with limited development effort to correct errors and to make enhancements. Data were received from April, 1977 through September,1977 in the form of status reports and IBM computer operating system tapes. The June tape was unreadable and the September tape was not received.

No source of data equivalent to the SAMTEC Run Analysis form was available from JPL. However, it was possible from the information contained in the error discrepancy reporting system (VISA's) to determine which errors were actually software-caused and to perform some failure rate and ratio calculations.

3. DESCRIPTIVE AND COMPARATIVE ANALYSES OF LSDB PROGRAM MODULES

The raw data collected during the LSDB development at SAMTEC was examined during this phase of the study to determine if other measures than failure rate or ratio could be derived. The analyses were done to provide insight into the detailed analyses that might be possible, or that should be performed. Variables such as the number of runs by module, types of runs, number of statement changes, number of lines of code, types of errors and types of statements such as assignment, logic and control were computed and compared. The results are given in the following paragraphs.

3.1 Project ASTROS Data

The total number of available records of runs available from Project ASTROS is 2,718. The sample selected is 2,700 (1,389 for 1976 and 1310 for 1977) of which 514 were unsuccessful. With the exception of 41 runs, all efforts indicated on the forms were in the category of program development. The distribution of program activities in the 2700 run sample is given in Figure 3-1 and indicates a dominant mode of compile and run. The distribution of the number of statement changes is given in Figure 3-2. The severity of failure in 490 cases was local job failure only; four other cases were reported as miscellaneous and one was reported as real time. The error category distribution is given in Figure 3-3; the dominant modes were logic errors (97) and operation errors (115). Single errors were detected in 419 of the failures; however, this measure is questionable for the actual number of errors since the sequence of detection of errors in sequential runs is unknown.

The distribution of the number of runs by Module is given in Figure 3-4. The BDP was the least used Module (185 runs); the LSO was the most used (492 runs). During 1976, the LDG Module had the longest runs (CPU=300 sec.); all modules, except LSD, had at least one run of 199 sec. CPU time. The 1977 pattern of module use showed an increase for BDT, LSD and LSO. LSD showed the longest run of 312 CPU. The distribution of the percent of total CPU time by module is given in Figure 3-5.

The percent of successful runs by module is tabulated in Table 3-1. The average success rate for all modules improved from 77.1% in 1976 to 85.0% in 1977.

The source code for the entire Metric Integrated Processing System (MIPS) was obtained from the contractor and a SNOBOL program (see Appendix A) was written to categorize the LSDB program in terms of statement type per module. This was done to assess the correspondence between error rates and program complexity as reflected by statement type. The

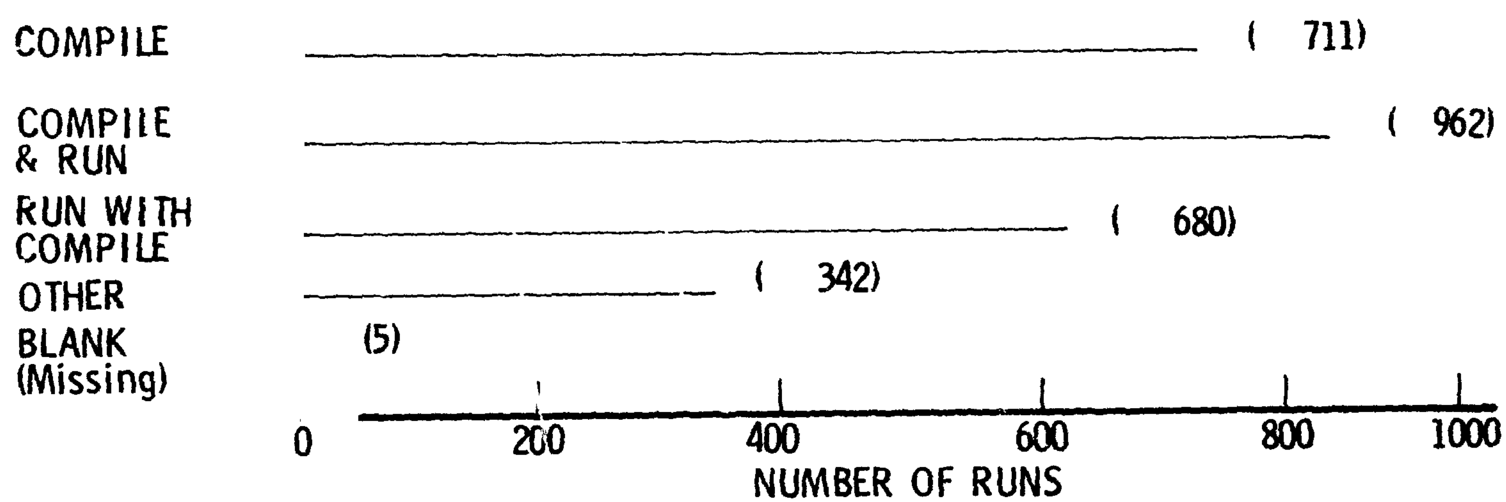

Figure 3-1. Distribution of Program Activities in 2700 Run Sample

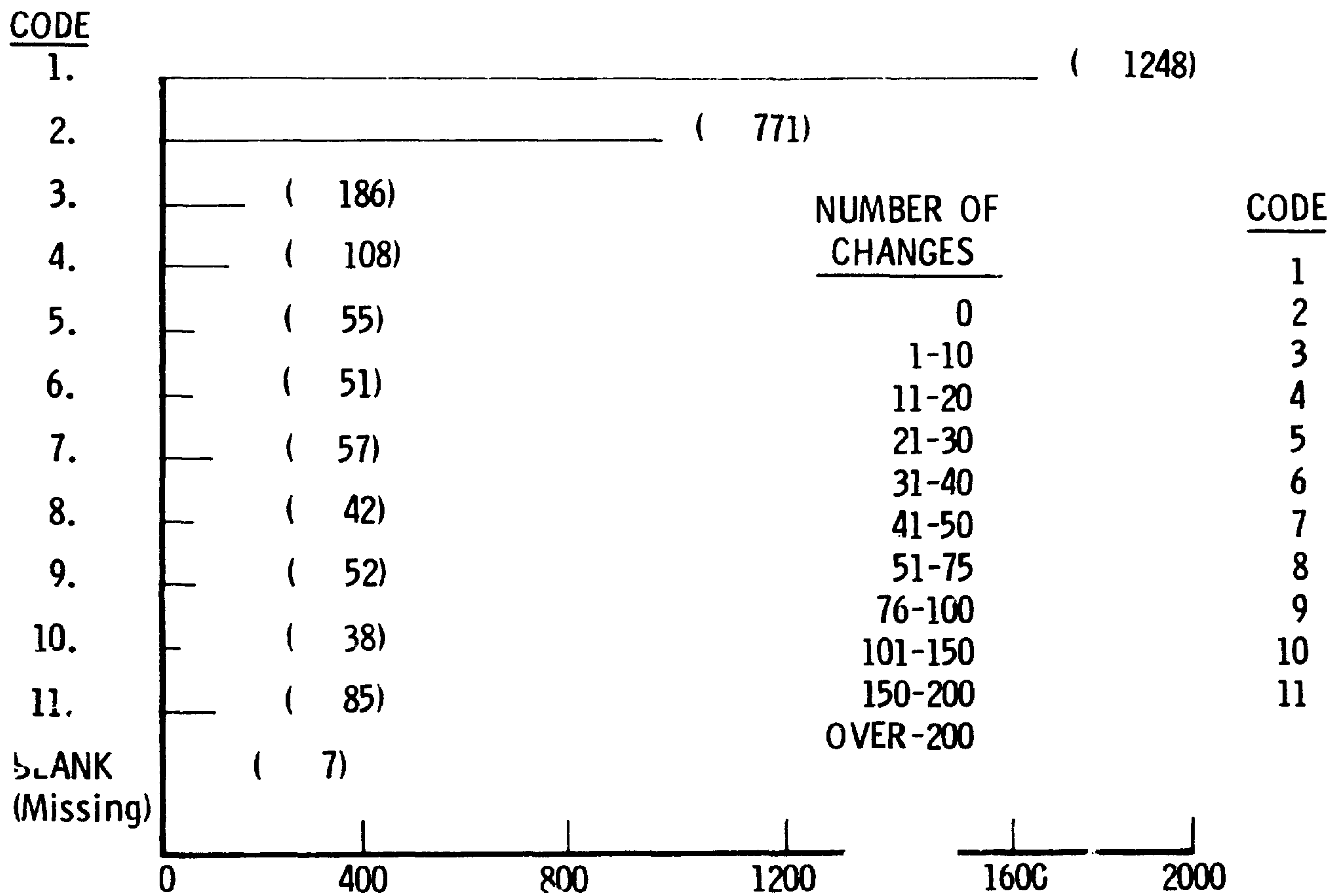

Figure 3-2. Number of Statement Changes in 2700 Run Sample

ERROR CODE	ABSOLUTE FREQ	RELATIVE FREQ (PCT)	ADJUSTED FREQ (PCT)
COMPUTATION	5	0.2	1.0
LOGIC	97	3.6	19.2
DATA INPUTS	17	0.6	3.4
DATA HANDLING	12	0.4	2.4
DATA OUTPUTS	3	0.1	0.6
ARRAY	1	0	0.2
DATA BASE	4	0.1	0.8
OPERATION	115	4.3	22.8
EXECUTION	41	1.5	8.1
OTHER	87	3.2	17.2
JCL	51	1.9	10.1
KEYPUNCH	72	2.7	14.3
NO ERROR	2195	81.3	--
TOTAL	2700	100.0	100.0

NO ERROR (2195)

Figure 3-3. Types of Errors Encountered in 2700 Run Sample

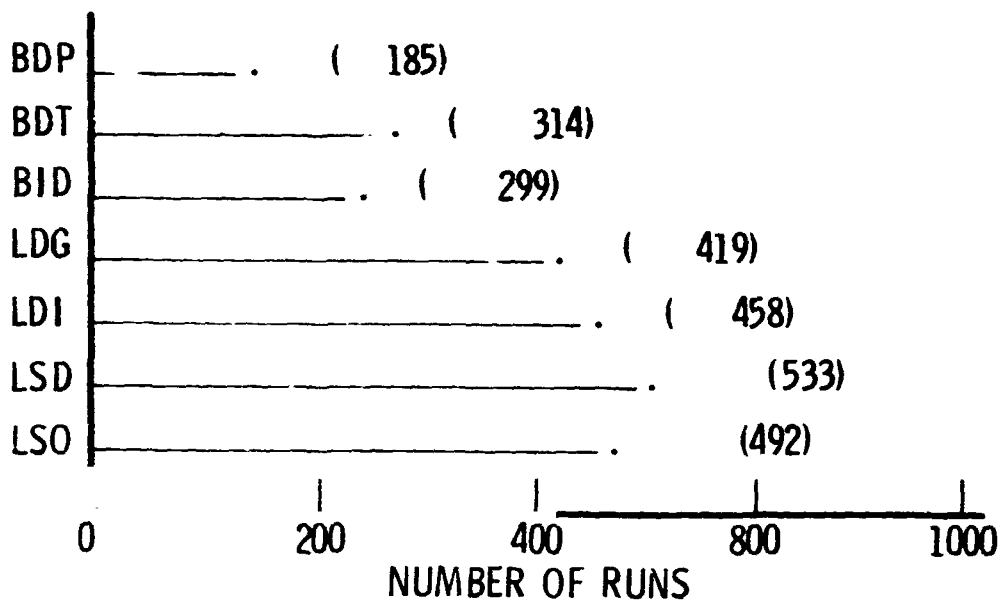

Figure 3-4. Number of Runs by Module in 2700 Run Sample

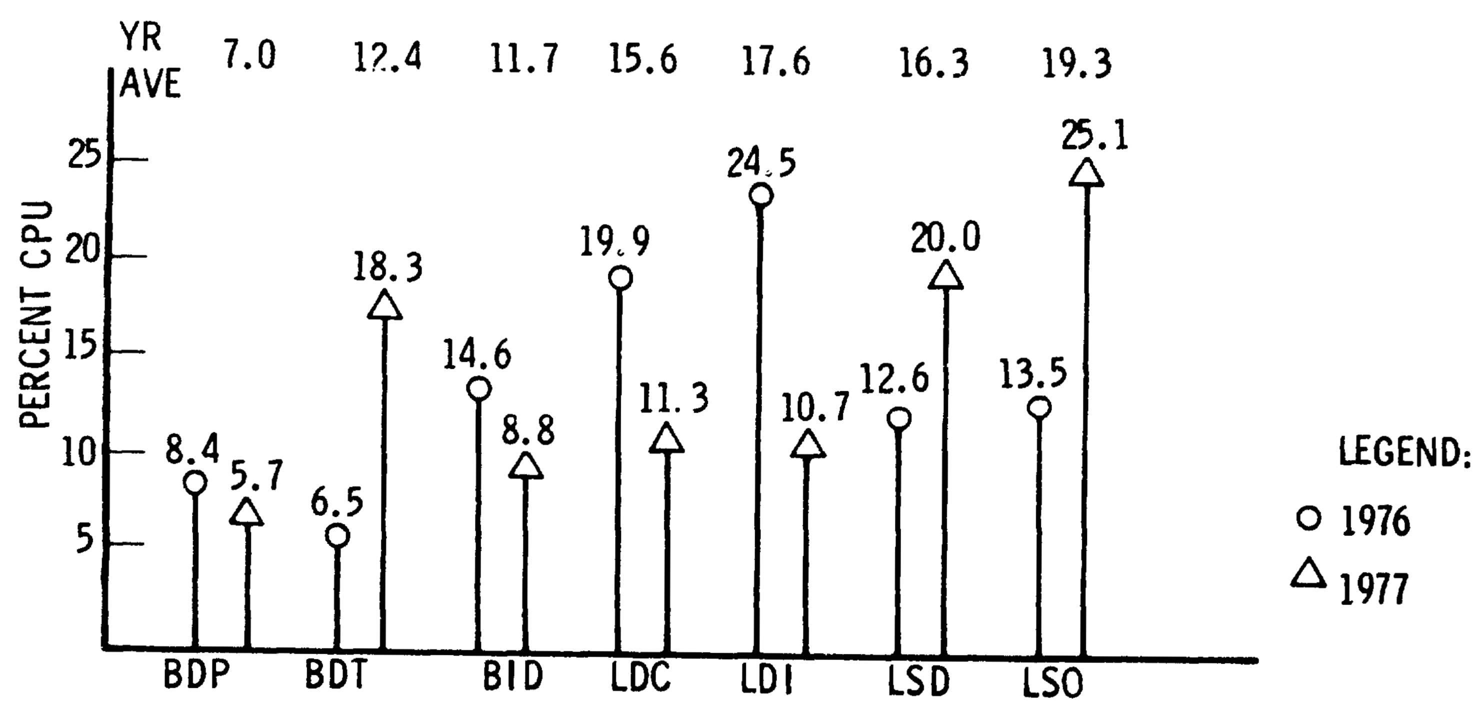

Figure 3-5. Distribution of CPU Time

Table 3-1. Success/Failure of Runs by Module

1976 DATA

	BDP	BDT	BID	LDG	LDI	LSD	LSO	TOTAL
SUCCESS COUNT	84	63	162	199	252	191	120	1071
% OF TOTAL SUCCESS	7.8	5.9	15.2	18.6	23.5	17.8	11.2	
% OF TOTAL MODULE RUNS	74.3	74.1	85.3	74.0	78.3	81.3	68.6	
% OF TOTAL RUNS	6.0	4.5	11.7	14.3	18.1	13.8	8.6	
FAILURE COUNT	29	22	28	70	70	44	55	348
% OF TOTAL FAILURES	9.1	6.9	8.8	22.0	22.0	13.8	17.3	
% OF TOTAL MODULE RUNS	25.7	25.9	14.7	26.0	21.7	18.7	31.4	
% OF TOTAL RUNS	2.1	1.6	2.0	5.0	5.0	3.2	4.0	
TOTAL	113	85	190	269	322	235	175	1389
MODULE % OF TOTAL	8.1	6.1	13.7	19.4	23.2	16.9	12.6	

6 DEGREES OF FREEDOM. SIGNIFICANCE - .0036

****

1977 DATA

	BDP	BDT	BID	LDG	LDI	LSD	LSO	TOTAL
SUCCESS CO'NT	65	190	91	129	107	271	261	1114
% OF TOTAL SUCCESS	5.8	17.1	8.2	11.6	9.6	24.3	23.4	
% OF TOTAL MODULE RUNS	90.3	83.0	83.5	86.6	78.7	90.9	82.3	
% OF TOTAL RUNS	5.0	14.5	6.9	9.8	8.2	20.7	19.9	
FAILURE COUNT	7	39	18	20	29	27	56	196
% OF TOTAL FAILURES	3.6	19.9	9.2	10.2	14.8	13.8	28.6	
% OF TOTAL MODULE RUNS	9.7	17.0	16.5	13.4	21.3	9.1	17.7	
% OF TOTAL RUNS	.5	3.0	1.4	1.5	2.2	2.1	4.3	
TOTAL	72	229	109	149	136	298	317	1310
MODULE % OF TOTAL	5.5	17.5	8.3	11.4	10.4	22.7	24.2	

Table 3-2. LSDB Statement Types

	BDP	BDT	LDG	LDI	LSO	Totals	Order
Equivalence	13	36	45	8	30	132	6
Call	58	67	76	28	182	411	3
If	148	107	109	143	304	811	1
Execute	87	45	80	102	182	496	2
Else	71	35	31	58	102	297	4
Do For	39	17	34	35	82	207	5
Undo	22	25	14	28	35	124	7
Cycle	1	2	0	0	2	5	9
Do Case	2	3	5	10	7	27	8
Do Until	0	0	0	4	0	4	10
Exit	563	331	460	340	1009		
Exit Order	2	5	3	4	1		
Program Failure Rate	.0028	.0031	.0017	.0027	.0014		
Order of Failure Rate	2	1	4	3	5		
CPU SECONDS	177	307	392	443	479		
No. of Statements	1979	1521	4837	3395	6294		

assumption was made that invocation of an external routine (subroutine call), logical decision and branching, and looping were statements of greater complexity than assignment. The distribution of statement types between the various modules of LSDB is tabulated in Table 3-2.

The results of the tabular analysis are shown in Table 3.2. The results indicate no clear pattern or relationship between variables or statement type, use and failure ratio.

3.2 Viking Data

The Viking data exhibited failure characteristics that are similar to the ASTROS data in a number of ways. For example, Table 3-3 shows that the source of failure could be attributed to the program in only 28% of the total. During the final month of data acquisition, the program errors constituted only 16% of the total. Most of the error sources were not explicitly identified.

The data for the monthly distribution of CPU time, number of runs, failure ratio and failure rate are given in Table 3-4. There was no apparent significant decrease in the recorded failure ratio or failure rate prior to the fourth month for data acquisition. However, at the end of the six month interval, both the failure rate and failure ratio were reduced to approximately one-half the beginning levels. Figure 3-6 is a plot of the failure ratio for software only as well as

TABLE 3-3

VIKING FAILURE SOURCE DISTRIBUTION

	PROGRAM	MECHANICAL	JCL/KEYPUNCH	OTHER	TOTAL
APRIL	19	0	0	35	54
MAY	10	3	1	36	50
JUNE	7	3	0	24	34
JULY	7	3	1	18	29
AUGUST	6	0	1	16	23
SEPTEMBER	6	4	1	27	38
TOTAL	55	13	4	156	228

TABLE 3-4

VIKING FAILURES

MONTH 1977	VISAs	RUNS	CPU 10^3 sec	FAILURE RATIO	FAILURE RATE* PER 10^3 sec
APRIL	54	1531	670	0.353	0.081
MAY	50	1327	422	0.0378	0.118
JUNE	34	NO DATA - BAD TAPE			
JULY	29	1075	408	0.0270	0.071
AUGUST	23	1409	550	0.0163	0.042
SEPTEMBER	38	NO TAPE DATA			

*Interpreting each VISA as a failure.

the composite of all VISAs. The number of data points does not provide an adequate date base for more detailed analyses. The results do indicate a possible trend in which the failure ratio for software alone declines at a lower rate than the composite. The total number of recorded program failure did not change significantly during the last four months of data acquisition; however, a significant increase in the failure incidence in some part of the system caused an increase to a level greater than the third month. The data are adequate to permit interpretation of this change. The trend, prior to that time, indicated that the total program was approaching a limiting level that would be asymptotic to the program failure rate.

The failure ratio and failure rate for the operating system are recorded in Table 3-5. Both improved by an approximately factor of three over the test interval. The final ratio was 0.01.

Table 3-5. Viking Operating System Recorded Failures

Month 1977	Software Failures	Runs	CPU 10^3 sec	Failure Ratio	Failure Rate,* per 10^3 sec
April	19	1531	670	0.012	0.0283
May	10	1327	422	0.007	0.0235
June	7	No data - bad tape			
July	7	1075	408	0.006	0.0171
August	6	1409	550	0.004	0.0109

*Interpreting each VISA as a failure.

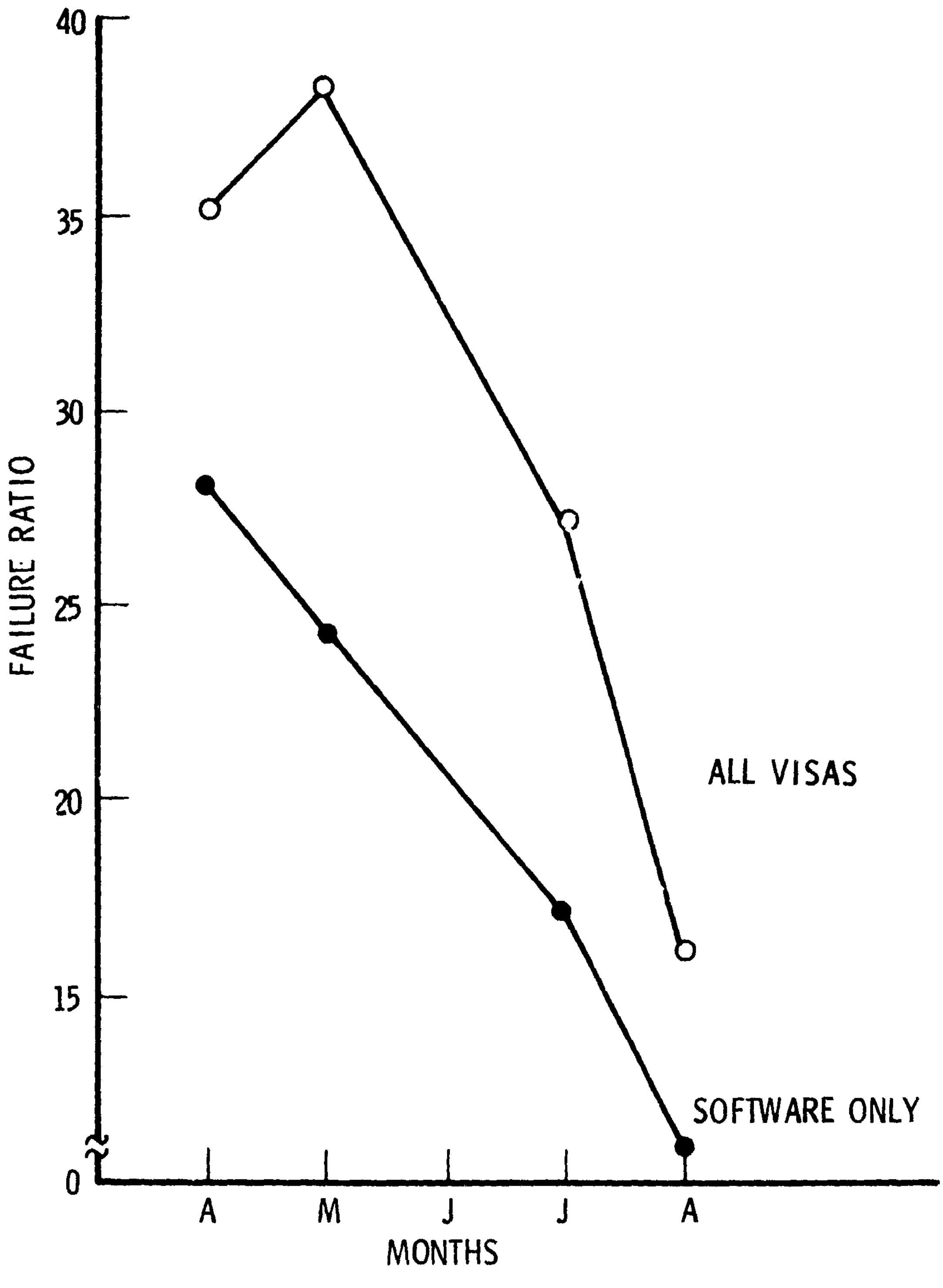

Figure 3-6. Viking Failure Ratio

3.3 Effects of Schedules

Scheduled reviews have an apparent effect on the rate of failure. Figure 3-7 reveals the effects of scheduldinq of LSDB activities as reported by VAFB. The notation (n) refers the reader to a point on Figure 3-7.

1) The high points: In-house testing of the module LDI started in early 1976 (1) . In April the testing was reduced in order to reevaluate the testing. In April to May period the testing was resumed. (3) represents the final testing of LDI and the testing of LDG. (5) represents the testing of modules BDP, BDT, BID, LSD, and (7) represents the testing of LSO.
   The low points: (2) was a period in which the documentation for the PDR (Preliminary Design Review) was produced. Points (4) Sept 1 and (6) December 15 were the times of the first and second CDR. (Critical Design Reviews).

Reviewing the above data it is clear that, at least in the gross sense, the number (ratio) of failures occurring in a module vs time is strongly a function of managerial action.

Telling the team what to test and when to test it influences the maxima and minima values of the curve. However, the magnitude of the maxima is a dependent function of the number of errors in the code (although how many are discovered is again a function of the testing procedure).

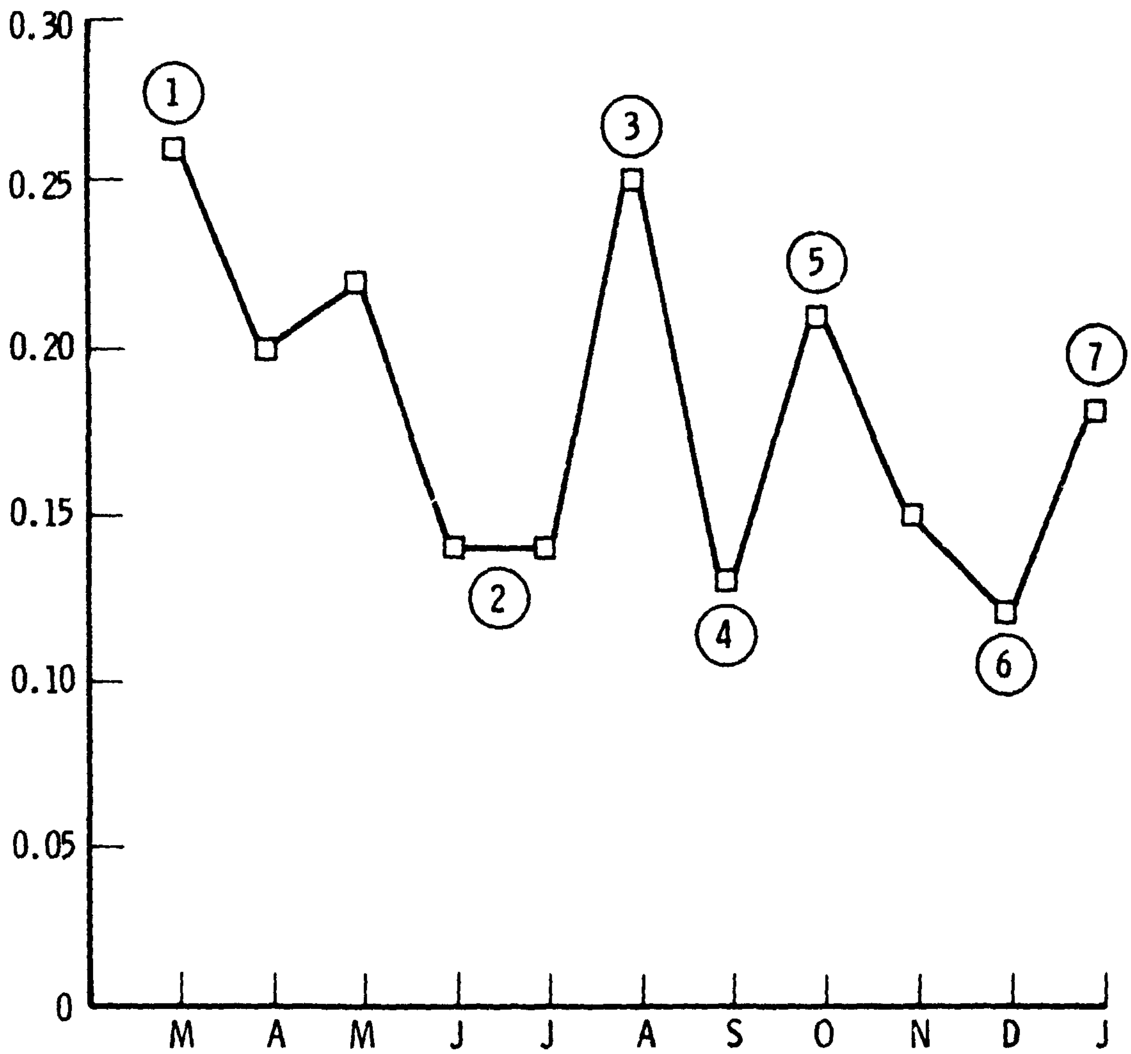

Figure 3-7. LSDB Total Failure Ratio

4. FURTHER ANALYSES ON THE EXISTING DATA BASE

During the first phase of this study, failure rate and ratio measurements were plotted and simple linear regression analyses performed.[3] To gain the maximum benefit from the data collection process, it was decided to subject the existing data base to more rigorous analyses both to verify the validity of possible measurements of reliability and to determine meaningful data that should be collected for future studies.

Since the data were acquired temporally, general time series analyses are possible for most parameters. The methods include linear and non-linear regression on time, autocorrelation, limited spectral analyses and stochastic forecasting. The results of the latter three methods are deferred to the next section.

Where the measurements were not adequate for parametric analyses, non-parametric analyses were performed. These methods include tests for normality, goodness-of-fit to theoretical distributions (deterministic models), correlation of variables or parameters, and tests for similarity and difference of variables.

An additional and important method for analysis is the between module comparison for internal validity of characterization and homogeneity. The comparison for external validity was reported previously.[3]

4.1 Regression Analysis

Nonlinear regression analyses were performed on the failure rate and failure ratio measured parameters. The method used, in most cases, was orthogonal polynomial regression. This method is somewhat more complex than simple least squares regression. However, the precision versus complexity trade off of parameter estimates, and adaptability for assessing improvement achieved by adding coefficients for higher order terms justify the complexity. The method involves the computation of a set of coefficients for each data point and remapping of the orthogonal polynomials back into a fundamental regression equation.

4.1.1 Composite Module Regression

Nonlinear (second order) regression was applied so as to observe the asymptotic behavior of the data. The results of the regression for composite modules over 16 months are given in Figures 4-1 and 4-2 for the rate and ratio respectively. The results normalized by the number of statements are given in Figures 4-3 and 4-4. The average failure rate decreased from an initial value of approximately 1% to a value of approximately 0.1% at the end of the observation of program development. The failure ratio of failed to total runs dropped from 15% to 5% (approximately) during the observed development interval.

As expected, the second order composite regression, for data stratified by week, produces the same general range as linear regression for failure rate and failure ratio estimates at the extremes. However, the more accurate fit reveals that the trend is toward an increase in failure rate and failure ratio from the initial value and a subsequent decrease with time. The regression demonstrated by the normalization of statement changes is shown in Figure 4-5.

4.1.2 Module Comparison

Comparisons of the trend in failure rate and failure ratio normalized by the number of statement changes shown in Figures 4-6 through 4-10. Consistency in the decrease of the failure rate and failure ratio of all modules both at the beginning and at the end of the observation period was observed.

It may be seen that the general forms of the regression curves are reverted J's, inverted U's, with some of the inverted J's having no significant up-turn.

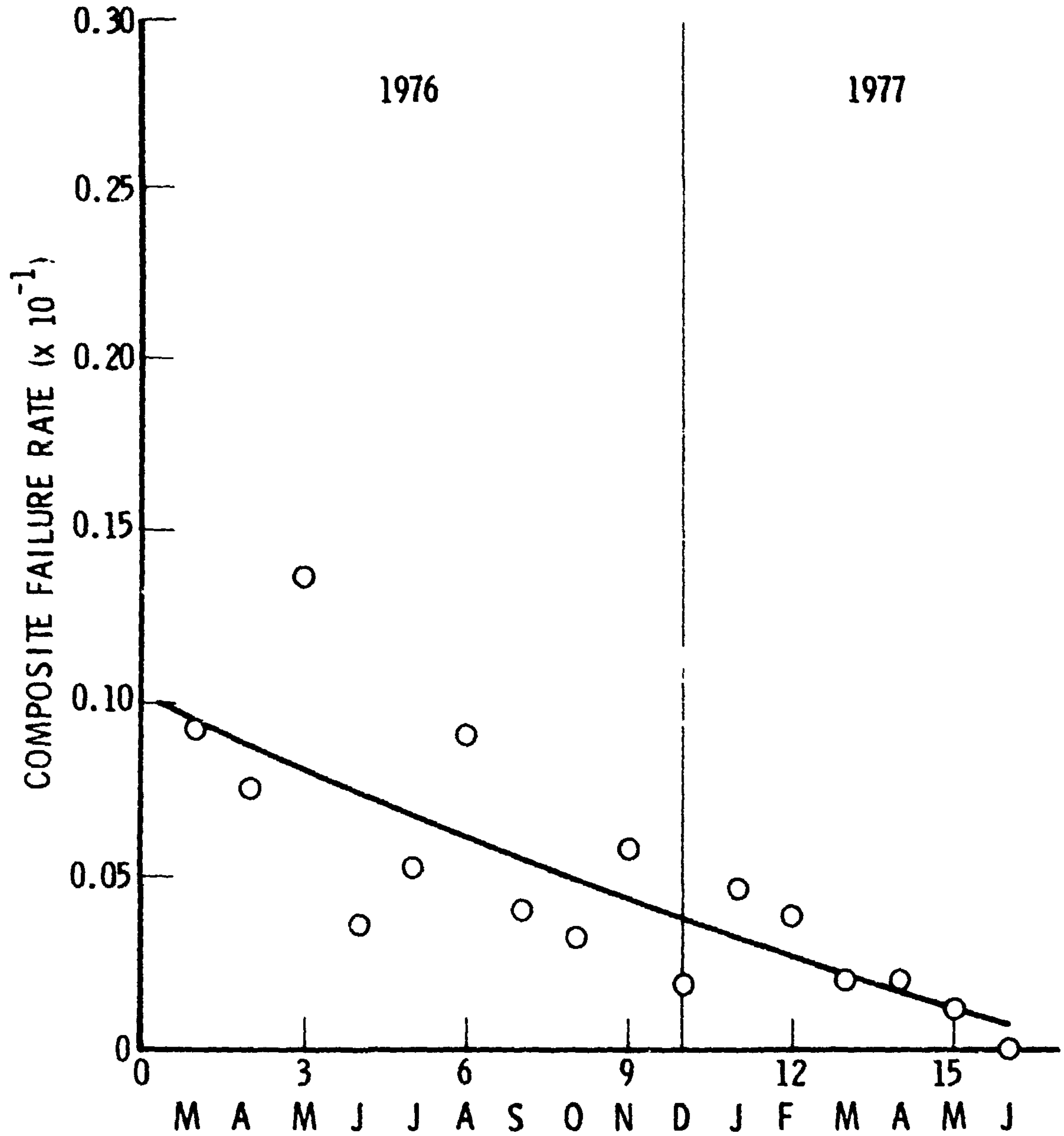

Figure 4-1. Composite Failure Rate

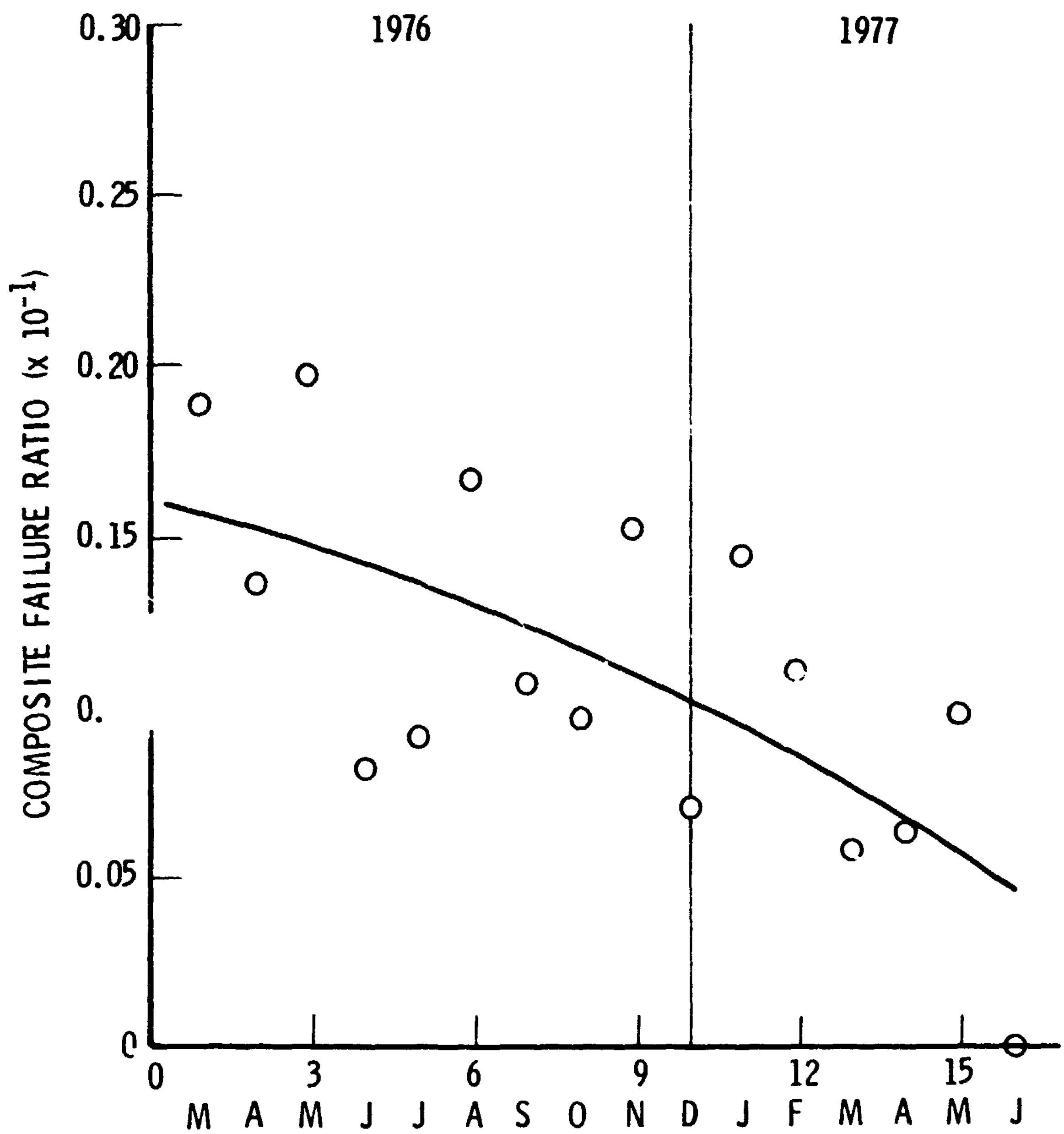

Figure 4-2. Composite Failure Ratio

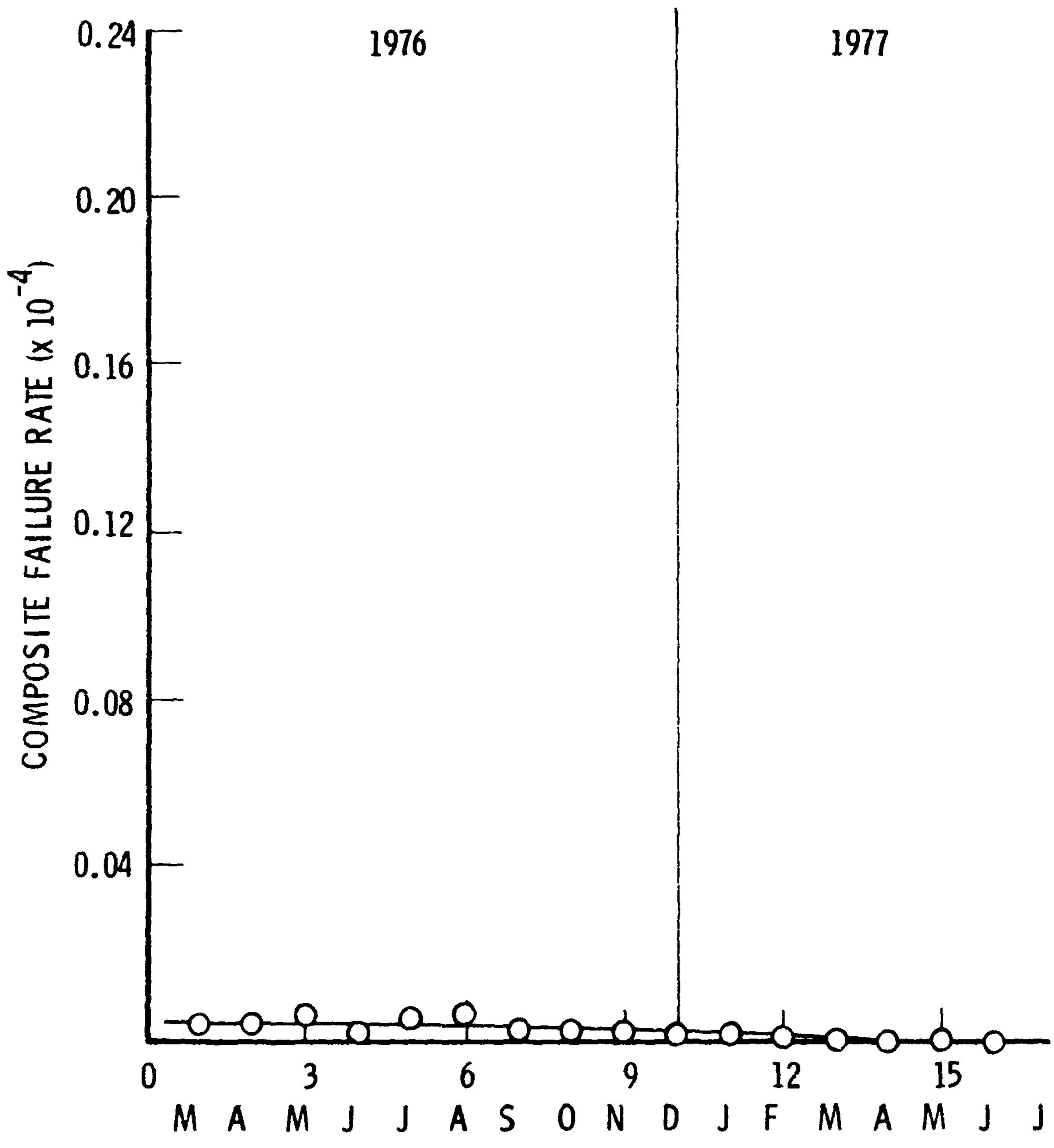

Figure 4-3. Failure Rate Normalized by Number of Statements

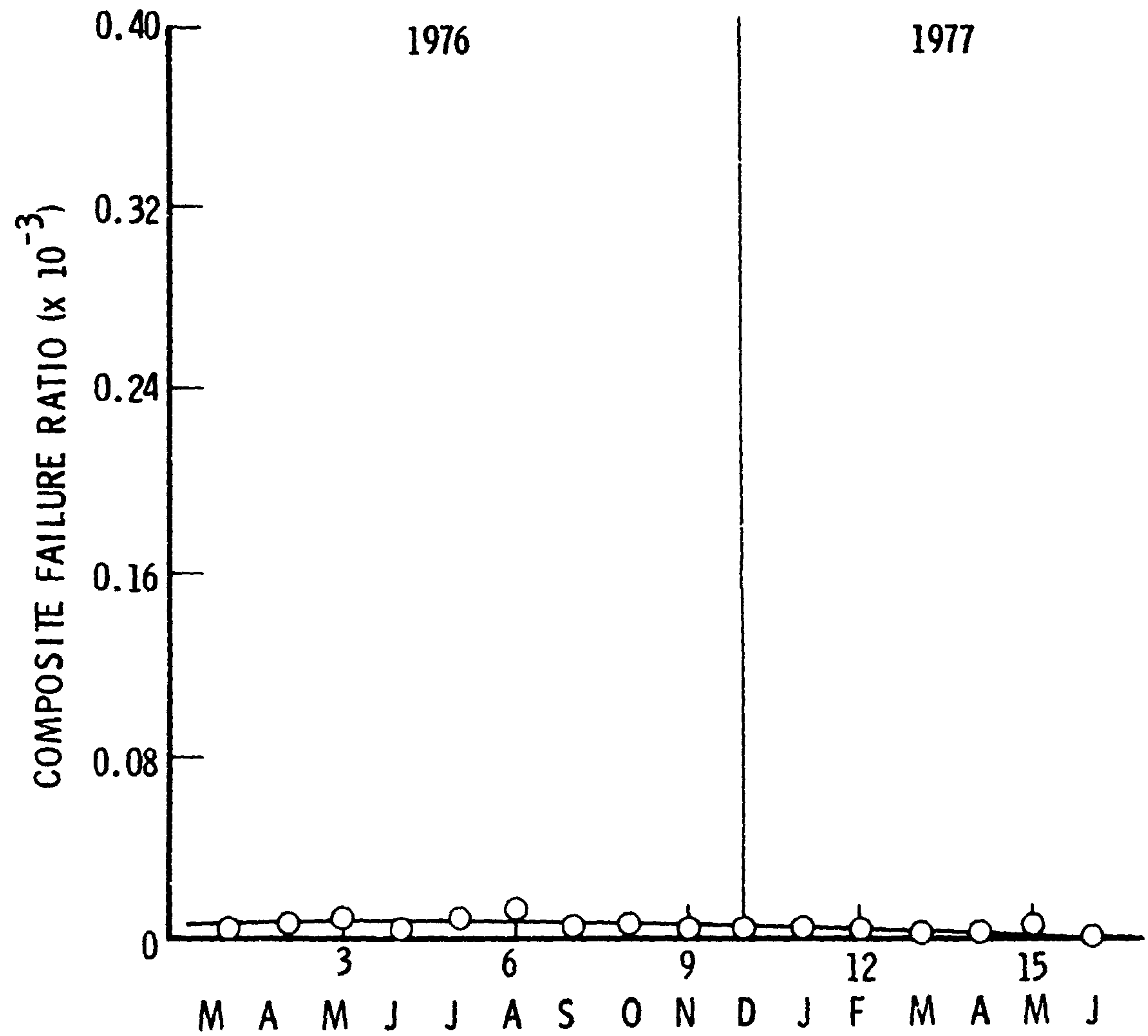

Figure 4-4. Failure Ratio Normalized by Number of Statements

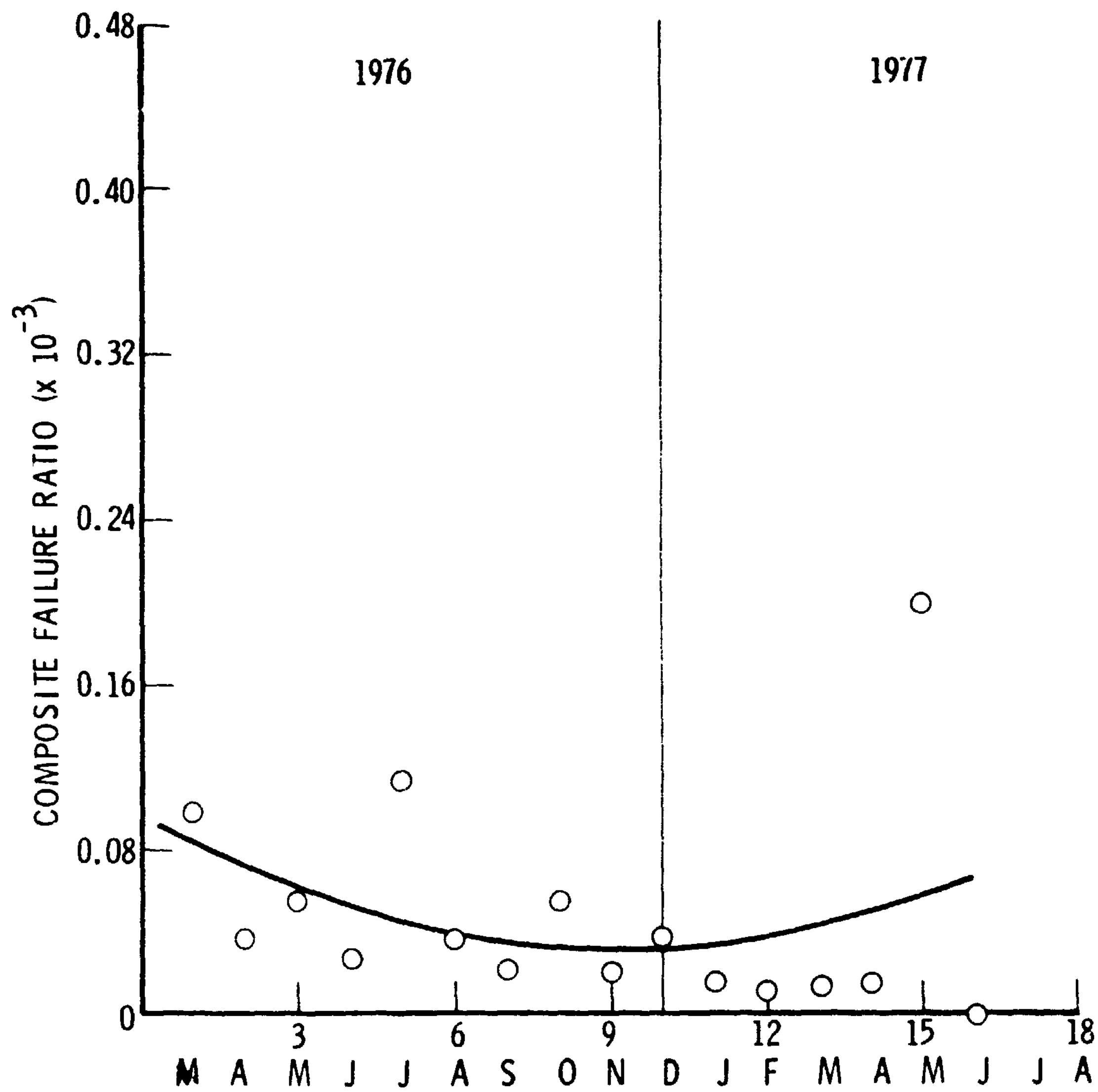

Figure 4-5. Failure Ratio Normalized by Number of Changes

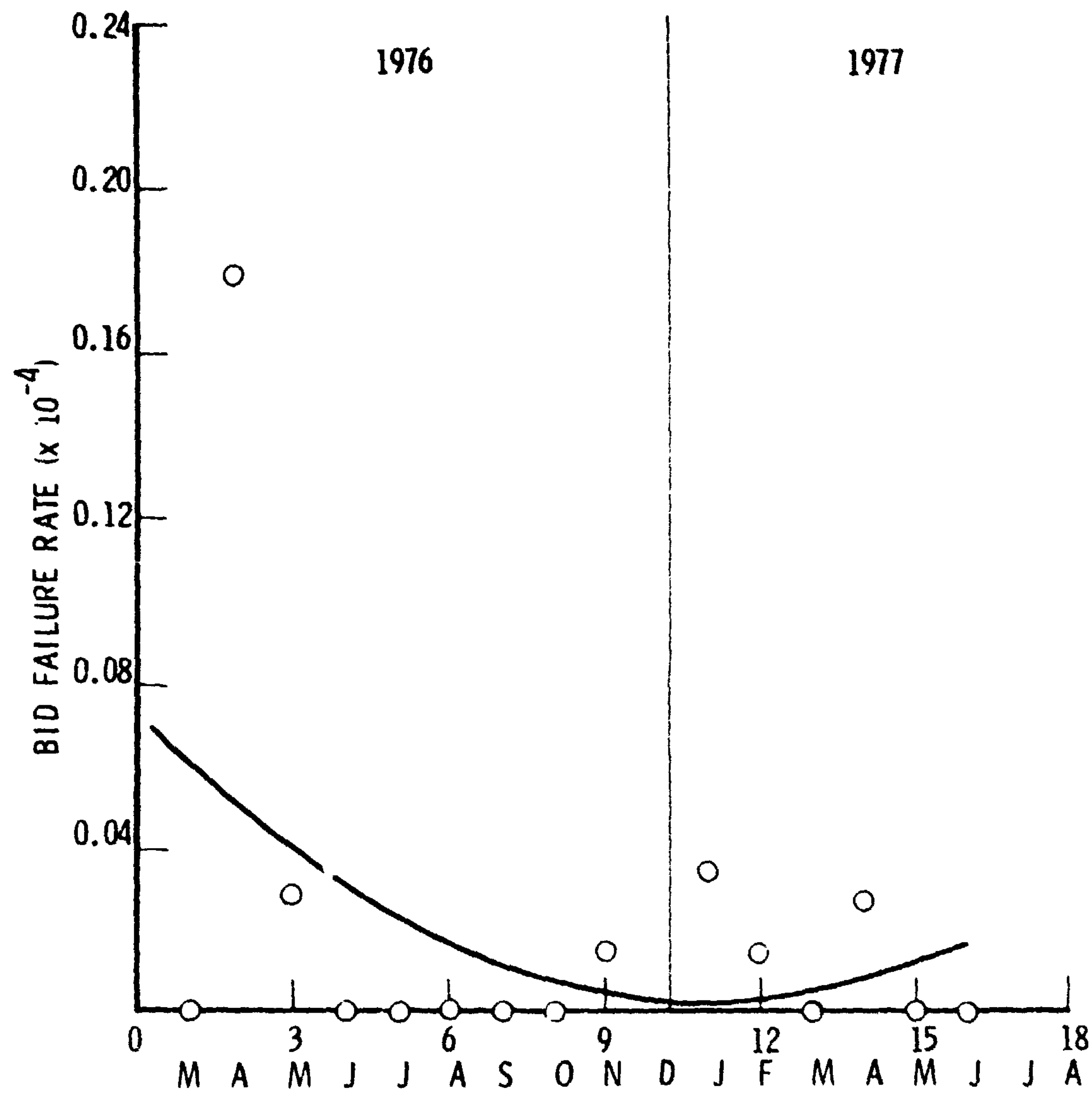

Figure 4-6. Failure Rate Normalized by Number of Changes

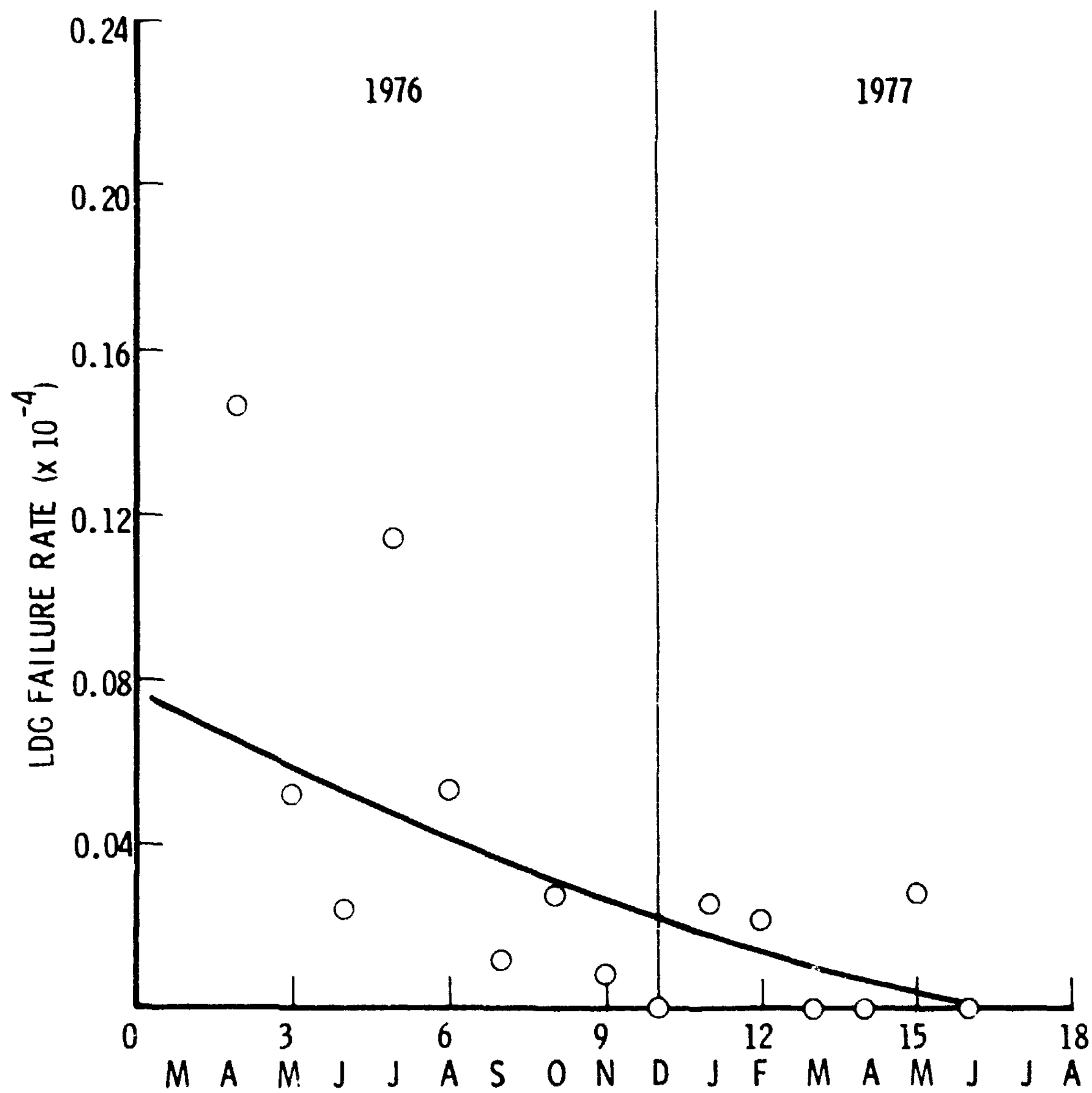

Figure 4-7. LDG Failure Rate Normalized by Number of Changes

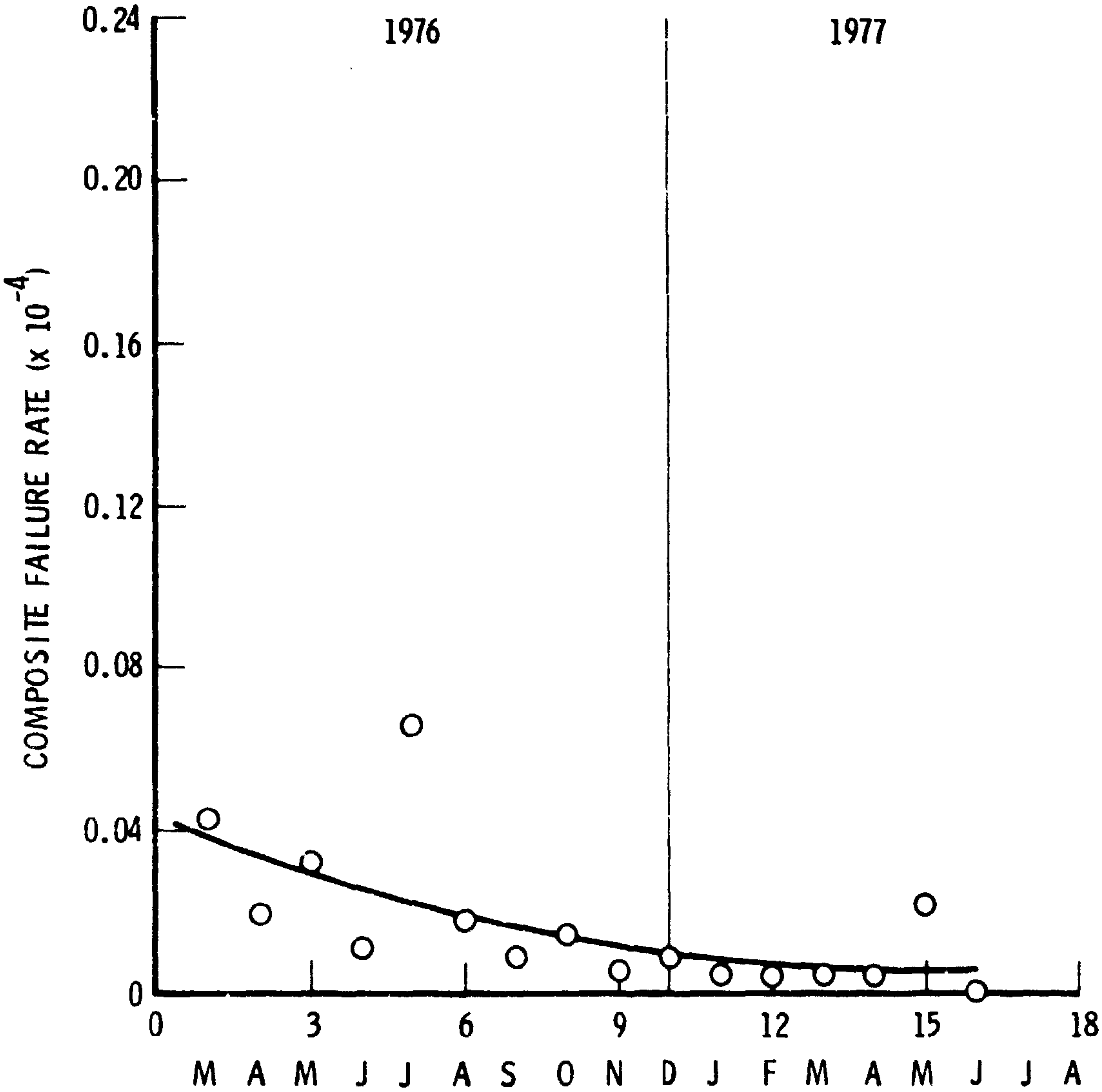

Figure 4-8. Composite Failure Rate Normalized by Number of Changes

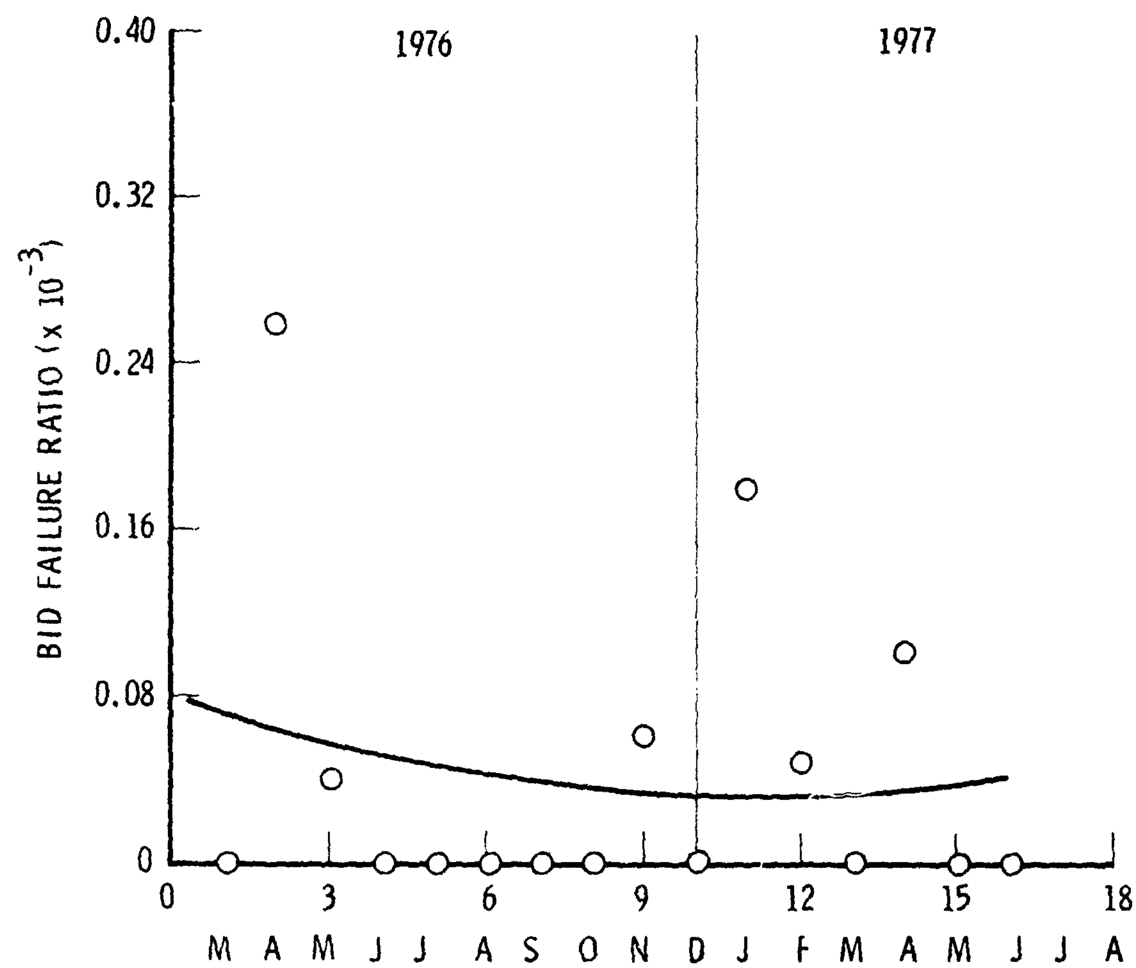

Figure 4-9. BID Failure Ratio Normalized by Number of Changes

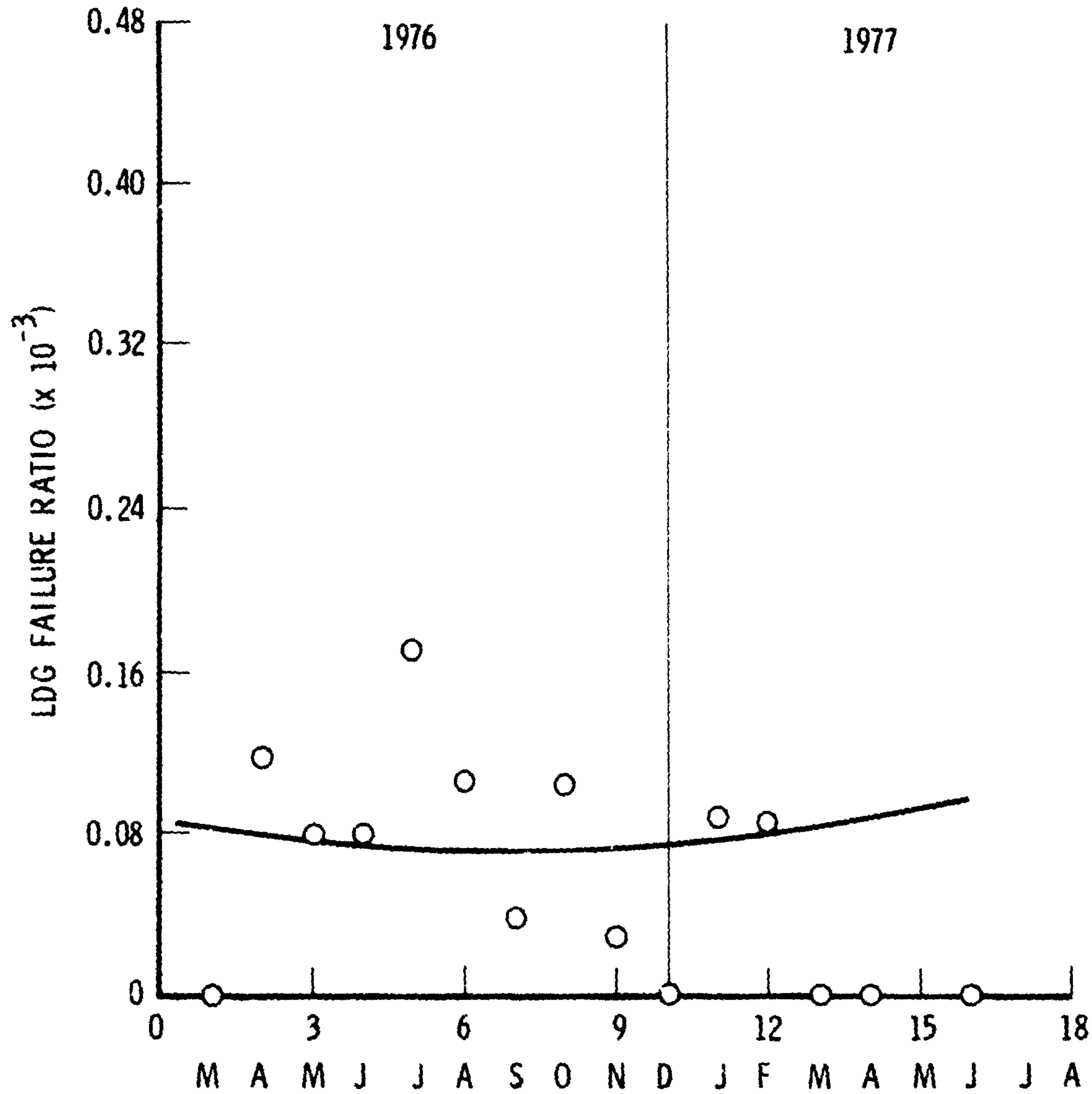

Figure 4-10. LDG Failure Ratio Normalized by Number of Changes

This characteristic, of course, is the equivalent quadratic form for the first three terms of the negative exponential given by

$$e^{-X} = 1 - X + \frac{X^2}{2!} + \epsilon(X)$$

$$\text{for } X \geq 1 \quad \text{and} \quad \epsilon(X) \leq \frac{X^3}{3!}$$

4.2 Non-parametric Analyses

Before any statistical tests are performed, the data must be examined for level of measurement and distribution. For data having measurement precision sufficient for parametric tests, the sample distributions of undetermined form must be checked to determine if there is sufficient goodness-of-fit to established theoretical distributions. For validity, such tests require that underlying assumptions be met. Failing either criteria, the data must be analyzed using non-parametric techniques. Transformations are legitimate only if the data can be transformed and the inverse transform of the results can be mapped back into the original domain of the data for consistent interpretation.

Tests for goodness-of-fit to a normal distribution were made on the number of statement changes, CPU time and failure severity. The results of the Kolmolgorov-Smirnov tests are given in Table 4-1 for the successful runs and Table 4-2 for the runs in which program errors were detected. The results indicate that none of these variates can be assumed to have come from a normally distributed population with any reasonable confidence.

A test for goodness-of-fit to a Poisson distribution was made on the CPU time by runs distribution . The results indicated the probability of the sample distribution having come from a Poisson distributed population was less than 0.00001. The same results were observed for success/failure of runs. Therefore, Poisson models appear to be inappropriate models for the variables.

Tests were also made to determine the probability that the number of reported statement changes for successful and unsuccessful runs could have come from the same population. The results of the tests indicated the probability to be less than one chance in 100,000.

Another test was made to check the corroboration of work category (program modification) similarity for successful runs with runs having failures. The results of a Kolmogorov-Smirnov test indicated a probability greater than 0.9999 that the work categories were from the same population.

TABLE 4-1

K-S TESTS FOR NORMALITY OF VARIABLES

(2136 SUCCESSFUL RUNS)

VARIABLE	MEAN	STD DEV.	MAXIMUM ABSOLUTE DIFFERENCE	2-TAILED TEST $P(H_0)$
NUMBER OF STATEMENT CHANGES	5	5	- 0.34	0.0000+
CPU TIME (Sec)	28.39	46.41	0.26	0.000+

TABLE 4-2

K-S TESTS FOR NORMALITY OF VARIABLES

(514 RUNS WITH DETECTED ERRORS)

VARIABLE	MEAN	STD. DEV.	MAXIMUM ABSOLUTE DIFFERENCE	2-TAILED TEST $P(H_0)$
NUMBER OF STATEMENT CHANGES	15	1.5	- 0.29	0.0000+
CPU TIME (Sec)	33.29	88.28	0.35	0.0000+
FAILURE SEVERITY	2.9	0.61	0.52	0.0000+

Kolmogorov-Smirnov tests were performed on variables which were measurable at the appropriate level and for which a sufficient number of runs were recorded. The tests were made on the distributions for the number of statement changes in successful runs compared to unsuccessful. The outcome is that which might be expected intuitively. Specifically, the probability that the number of changes was similar in both cases was less than 0.001 which is stronger than might be expected. In contrast, the work categories for successful runs or unsuccessful runs are indistinguishable. The probability of them being from the same population is 0.999.

TABLE 4-3

NONPARAMETRIC CORRELATION OF VARIABLES

VARIABLES COMPARED (ORDERED)	CORRELATION COEFFICIENT	SIGNIFICANCE LEVEL
F ure severity with erιor category	0.978	0.001
Failure severity with error count	0.917	0.001
Error category with	0.903	0.001
CPU time with number of statement changes	0.248	0.001
Work category (Program Mod.) with Program Activity	0.128	0.001
CPU time with Program Activity	-0.461	0.001
Program Activity with number of statement changes	-0.3570	0.001
CPU time with error category	-0.153	0.001
CPU time with error count	-0.147	0.001
CPU time with failure severity	-0.138	0.001
All other Parameter Comparisons	0.11	

4.3 Non-Parametric Results

The relationships between variables were examined using distribution-free (non-parametric) methods. The methods included Spearman's non-parametric correlation analysis, Kolmogorov-Smirnov tests for similarity (independence) and Chi-square tests for comparability.

4.3.1 Non-parametric Correlation

Table 4-3 presents the results of the non-parametric correlation analyses performed on variables that were measured at the appropriate levels. It may be observed that the highest positive correlation (on a scale from -1 to +1) is (0.978) between the failure severity with the error category . This high value for correlation should be interpreted as being a measure of the concentration of failures for local job failure only. In contrast, the error category distribution is quite broad and multimodal. The second highest correlation is also attributable to the concentration of failure severity into one category. A similar effect was observed for correlation of error count (number of errors) with any other variable. As it should be, the correlation of error count with the category was high (0.903).

The next group of correlation coefficients are not as impressive but perhaps provide more insight into relationships that are not as intuitively obvious. The CPU time was correlated with a number of variables. The CPU time is distributed over 177 categories with a general distribution of the highest percentages in the first 12 categories; for the next 12 categories the CPU time dropped to approximately one-third the average for the first 12 and continued as a long tailing-off for the remaining categories. The general form is that of a negative exponential, which of itself is not significant. However, in terms of potential inference rather than form, the characteristic is similar to a Chi-square distribution with three degrees-of-freedom. This may or may not be due to chance, but if it is significant, future studies might be directed toward the decomposition of the CPU time dependency upon a small number (4) of factors. It should also be cautioned that apparent variables may not be independent but interactive instead. In any event, the data as recorded does not permit factor analysis, and therefore, the non-parametric correlation of CPU time with other variables was computed as given, in Table 4-2.

The variable found to have most significant positive correlation with CPU time was the number of statement changes (0.248). The coefficient is not high in absolute value but it is relatively high compared to other variables. The two relatively high negative correlations are due to the ar trary ordering of the program activity measured variable which is comprised of combinations of compile/run activities. The correct interpretation of the results should be that there is relatively high correlation of program activity with CPU time and the number of statement changes, respectively. The other correlations are of lesser magnitude; the proper interpretation is as given by the sign in the table.

5. Reliability Forecasts

Any sequence of tests or experiments must eventually be concluded. The key question is when to stop. There are a number of answers to the question that are premised upon given criteria or values. In either case, the future reliability must be addressed. For example, the criteria could be the maximum deviation of a sample from a deterministic estimate, or the maximum mean-square-error between samples at a given confidence level. Another answer could be to stop when a measure of failure converges, or when the forecast converges to some value or has a well defined trend that passes through zero. The forecasts for the failure rates and failure ratios were computed for the composite (ensemble average) of the five modules and the individual modules.

The method for forecasting is based on work first published by G.U. Yule and refined by Box and Jenkins.[7] It is a stochastic method that does not depend upon the assumptions required for a deterministic and stationary model. The autoregressive integrated moving average (ARIMA) method is somewhat a misnomer in that the "integration" evolved from a hardware application concept which makes use of a nonstationary summation filter.

The data plots (as stratified by month) and forecasts for nine months beyond the 16 month test period are given in

Figures 5-1 and 5-2 for composite failure rate and composite failure ratio respectively. The 95% confidence bands for forecasts are indicated. Where the lower band goes below zero it is omitted. It may be seen from Figure 5-1 that the forecast for the composite failure rate converges to approximately 0.02. Figure 5-2 reveals that the forecast for the composite ratio trends toward zero after remaining at approximately 0.025 for three months.

Figures 5-3 and 5-4 present additional normalized failure ratio forecast examples. In Figure 5-3 the forecast of BDP failure ratio as normalized by the number of statements predicts that the trend would approach zero asymptotically in approximately six months following the end of the recorded tests. The trend declined from the initial stochastic estimate of approximately 0.02 per 10^3 statements. The upturn toward the end of the test period is attributable to the number of changes to the program module. Figure 5-4 reveals a similar forecast trend for the BDT module except the zero asymptote is predicted for eight months after the end of the recorded data interval. The absence of an upturn is apparently due to fewer program changes. The LDG failure ratio as presented in Figure 6-5 is relatively flat (on the average) for the first 10 months and begins a downward trend in January of 1977 toward zero in June of 1977. The forecast predicts that the normalized failure ratio of the LDG Module should have converged toward zero by the beginning of 1978, providing the type of perturbations introduced after the end of the data acquisition period were not significantly different from the perturbations encountered during the 15 months in which data were collected. The LDI and LSD failure ratios are given in Figures 5-6 and 5-7 respectively and are quite similar to LDG as previously discussed.

The failure rate characteristics with forecasts are given in Figures 5-8 through 5-12. The BDP module exhibits failure rate characteristics in Figure 5-8 that are quite similar to the BDP failure ratio. However, for the BDT module the correspondence between the failure rate, as presented in Figure 5-9, and the failure ratio is not as good. The best forecast estimate based on all past BDT module measurements produces divergence from zero. However, this may be influenced by the wide divergence of failure rate at the beginning of the test period. If only the last 10 months of the test data were used the forecast would likely converge. This module obviously had rather severe problems initially. The best estimate for the failure rate of the LDG module is almost linear and the forecast indicates earlier convergence of the failure rate approaching zero sooner than the rate for BDT module and closer to the same time as the BDP module. The LDI and LSO failure rate characteristics, as exhibited in Figures 5-11 and 5-12, are not significantly different, as a function of time, than the failure ratio forecasts.

The essence of this analysis is that the ensemble average of both the failure rate and failure ratio are stationary and provide a basis for forecasting the program reliability. Individual module forecasts may not be as well behaved. Future study should provide an opportunity to test verify these methods for forecasting.

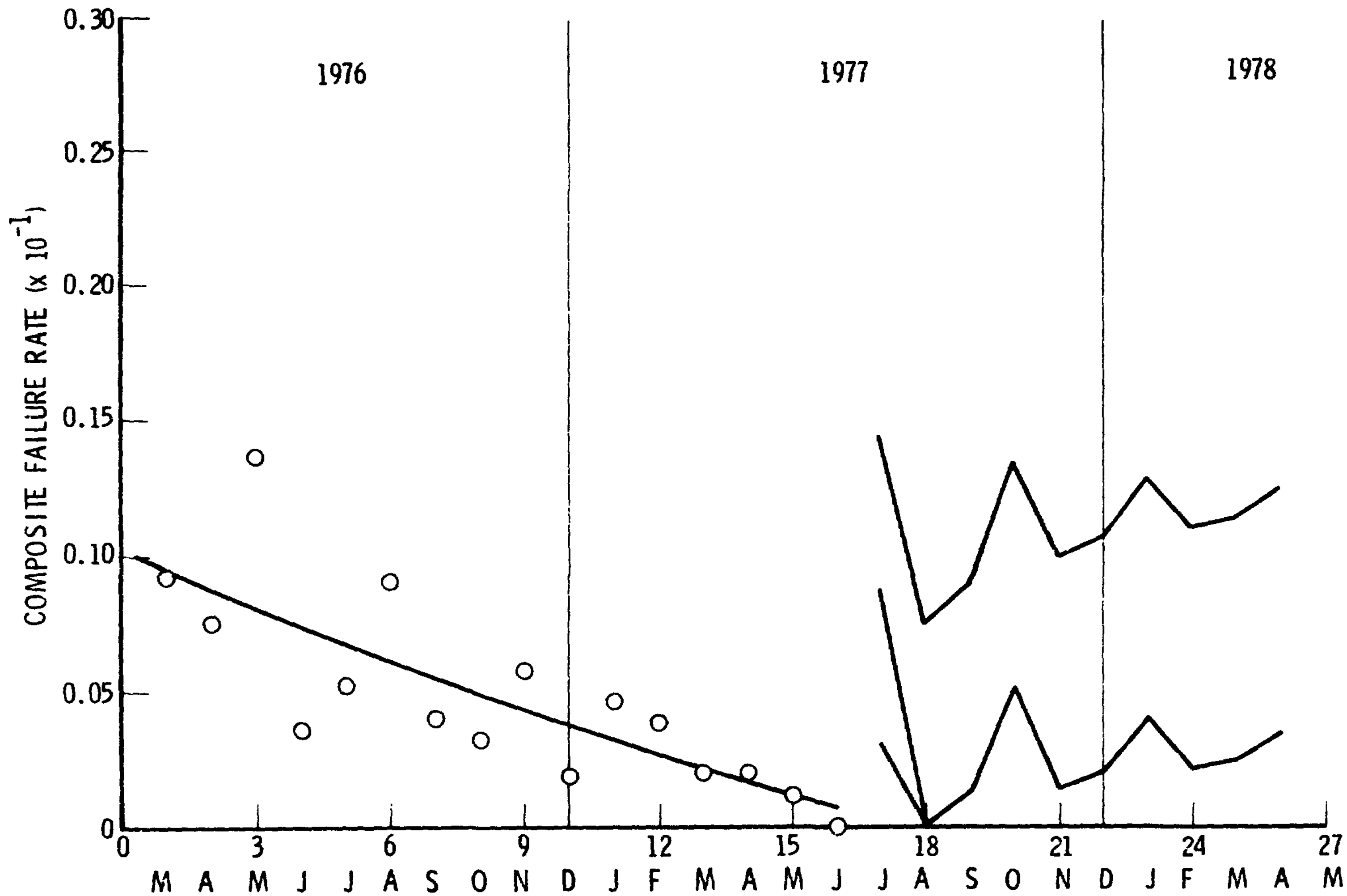

Figure 5-1. Composite Failure Rate

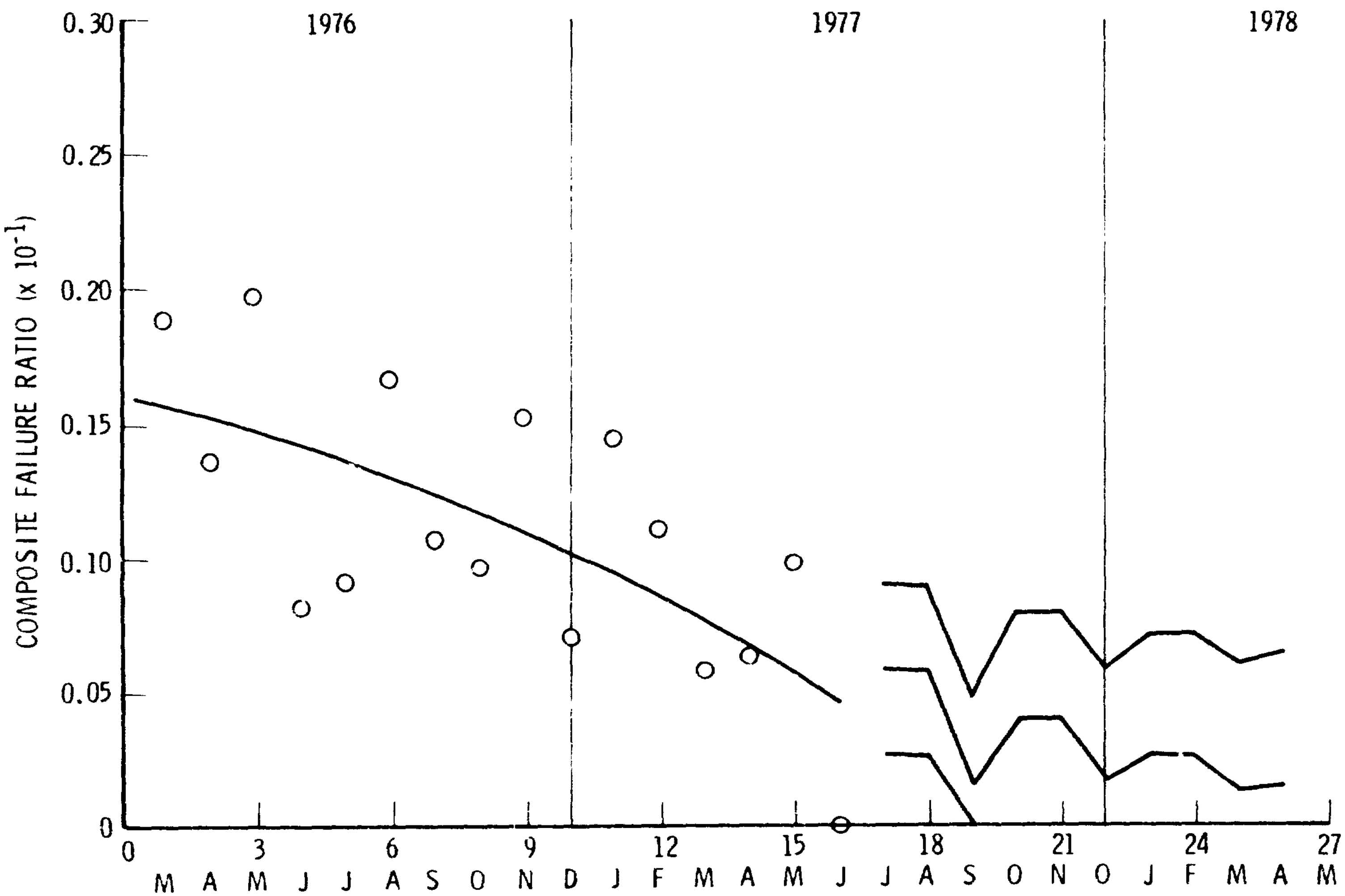

Figure 5-2. Composite Failure Ratio

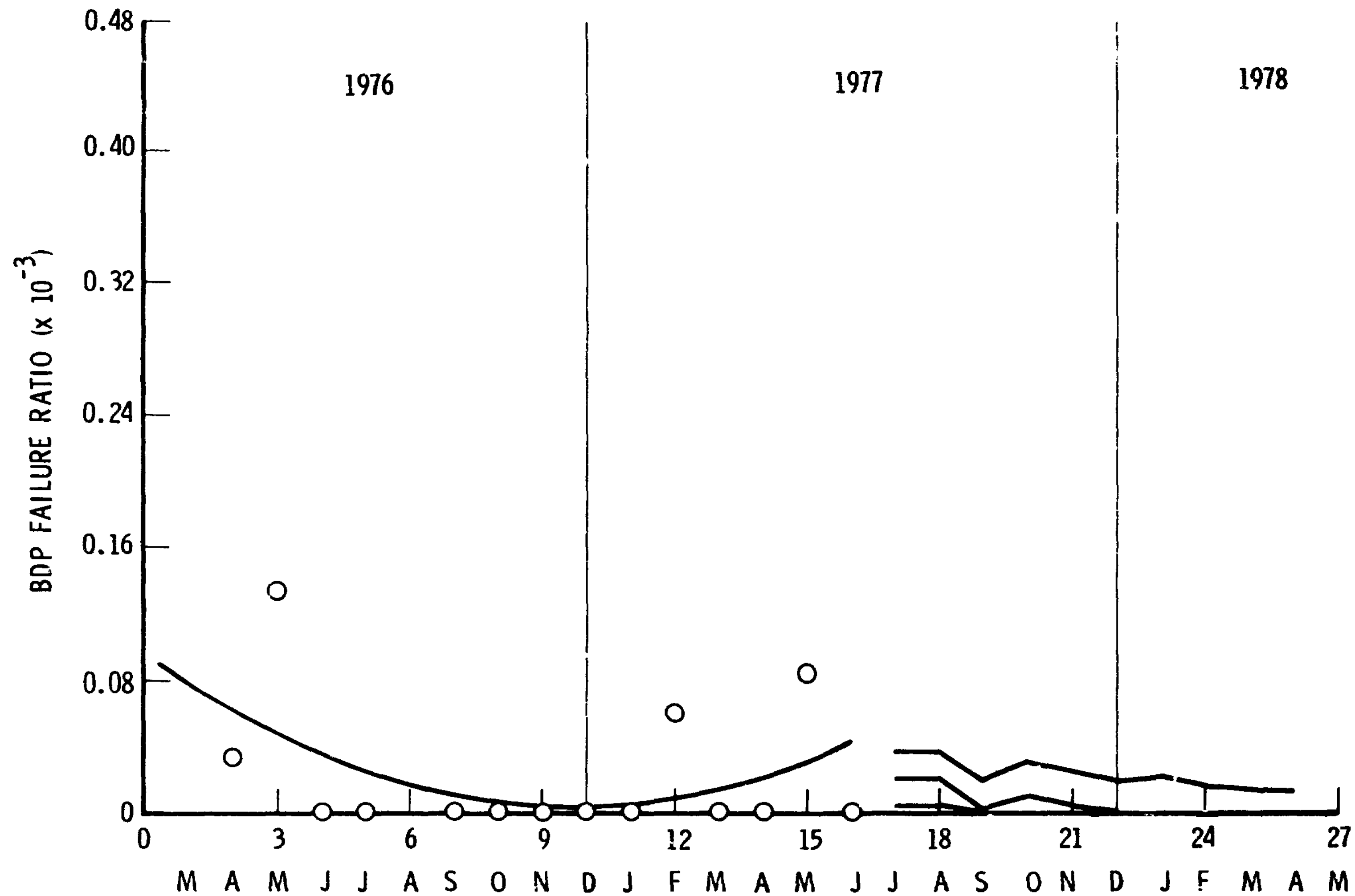

Figure 5-3. BDP Failure Ratio -- Normalized by Number of Statements

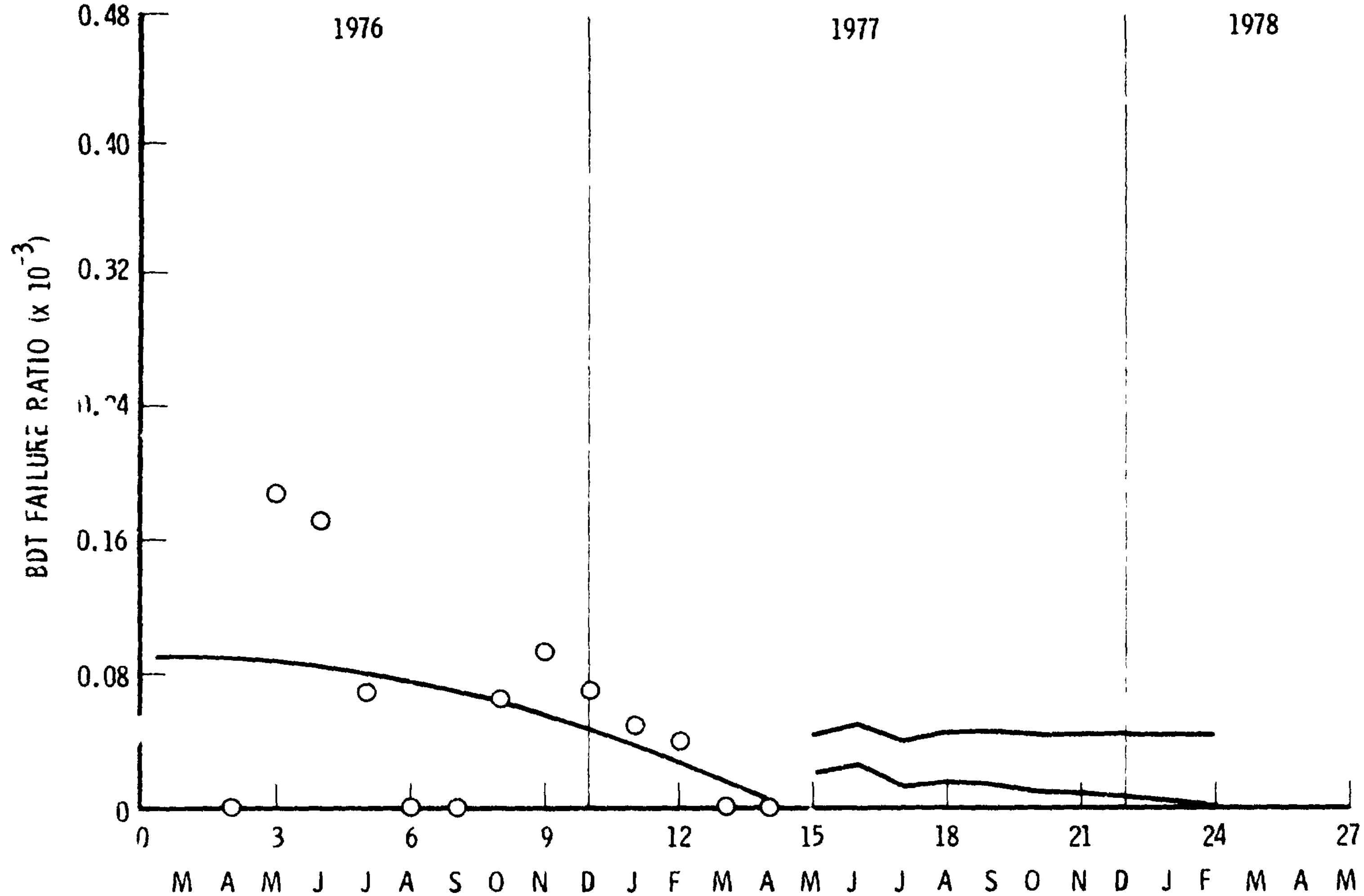

Figure 5-4. BDT Failure Ratio - Normalized by Number of Statements

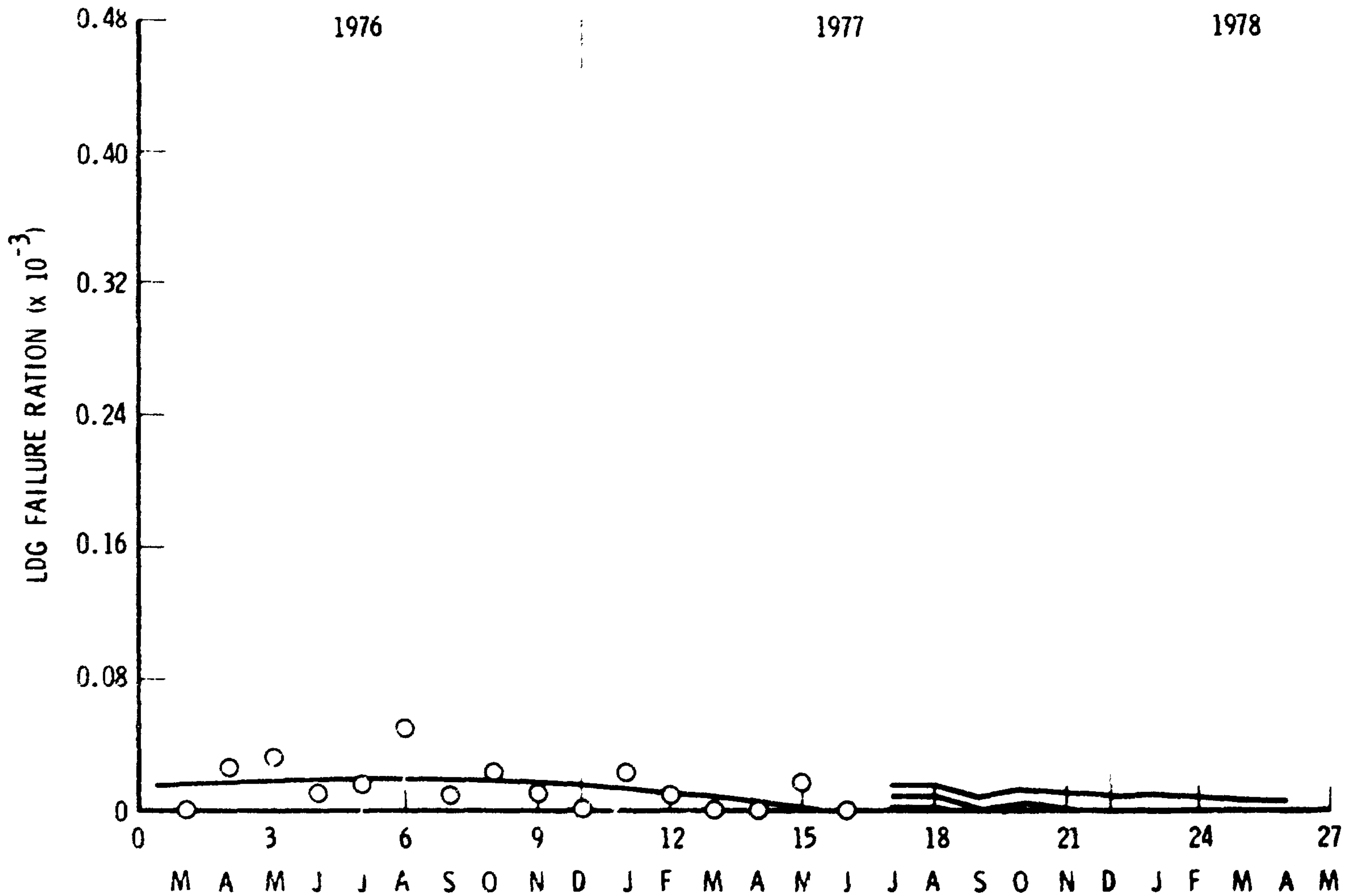

Figure 5-5. LDG Failure Ratio – Normalized by Number of Statements

Figure 5-6. LDI Failure Ratio – Normalized by Number of Statements

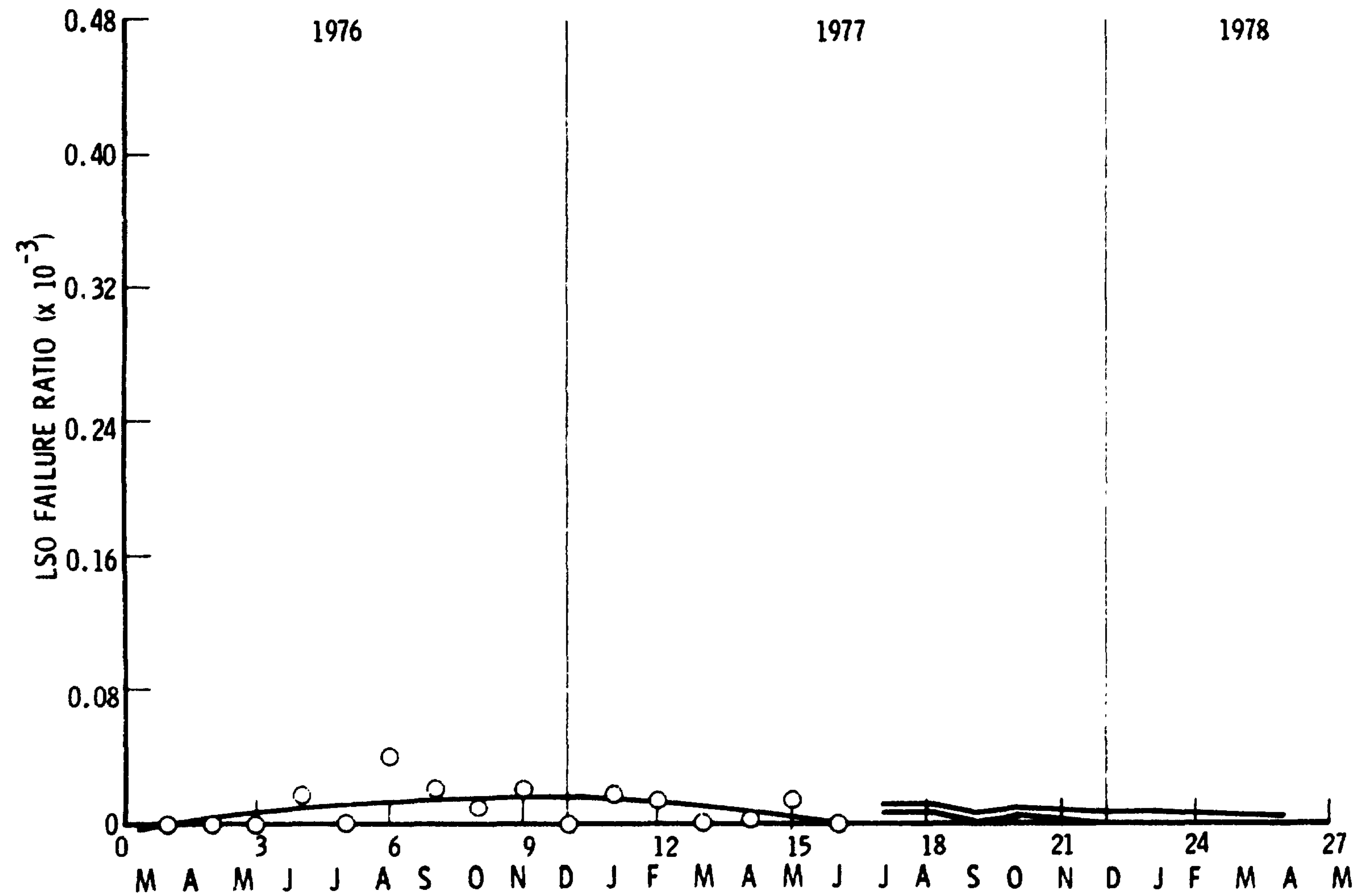

Figure 5-7. LSO Failure Ratio - Normalized by Number of Statements

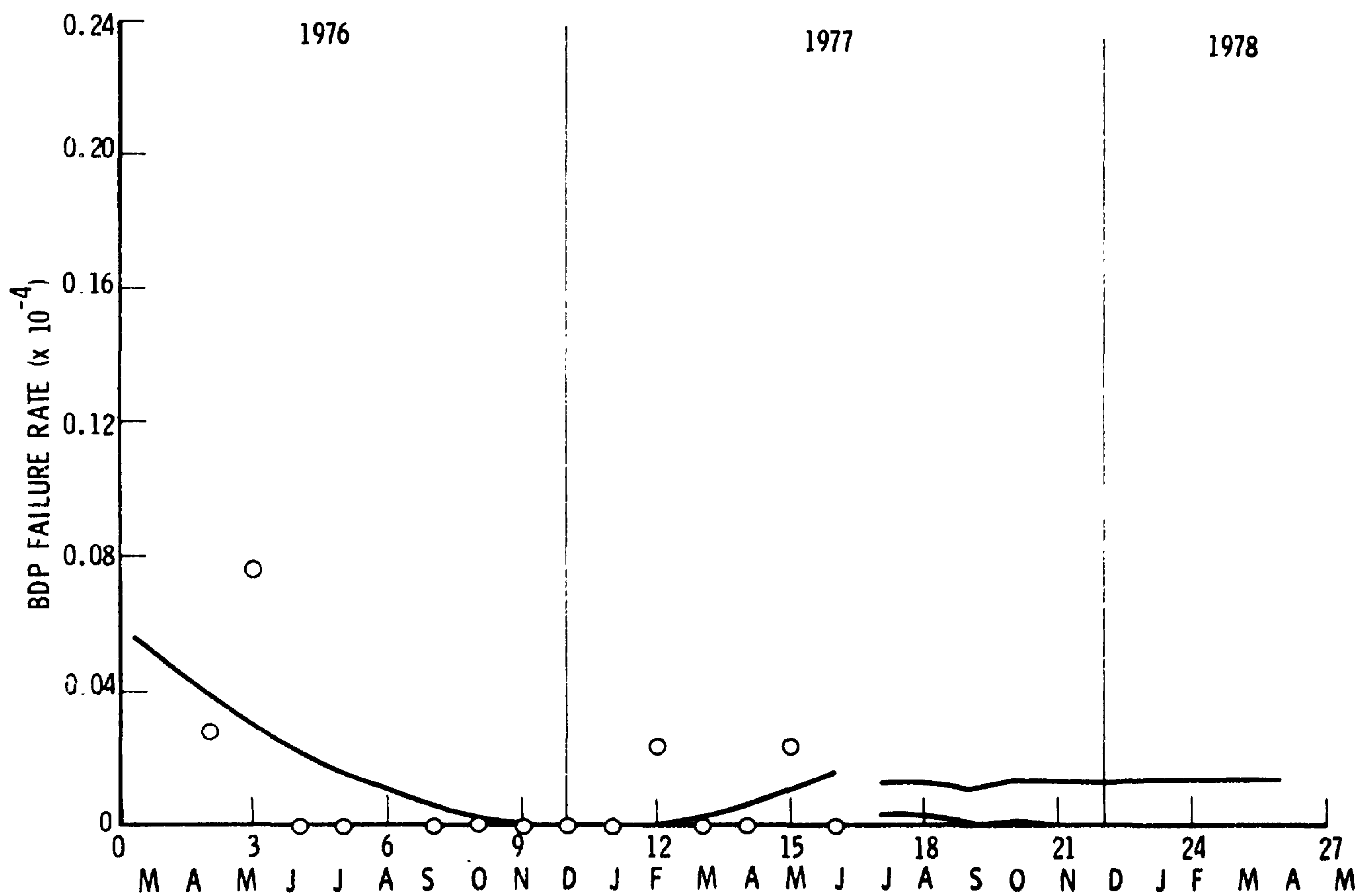

Figure 5-8. BDP Failure Rate – Normalized by Number of Statements

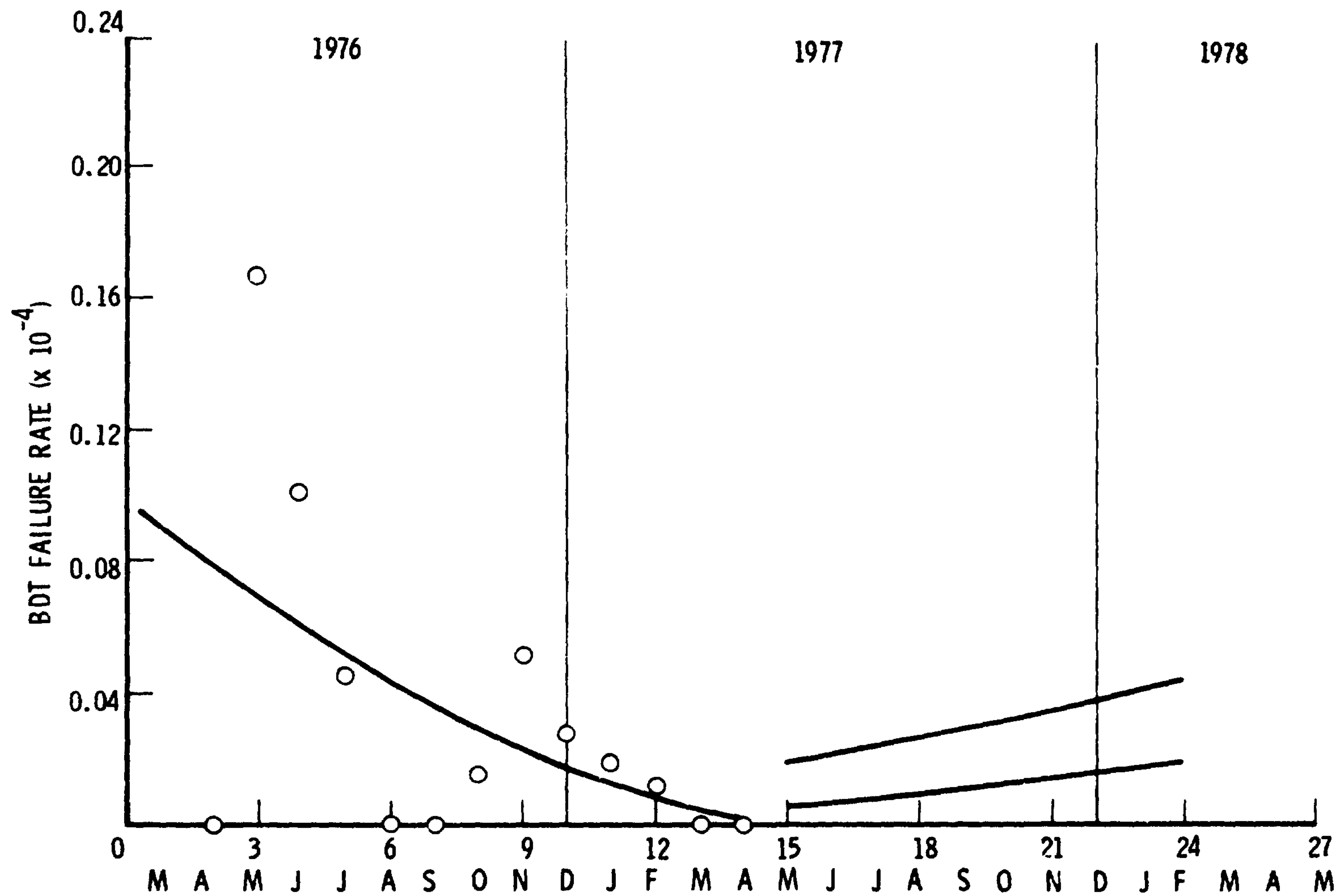

Figure 5-9. BDT Failure Rate - Normalized by Number of Statements

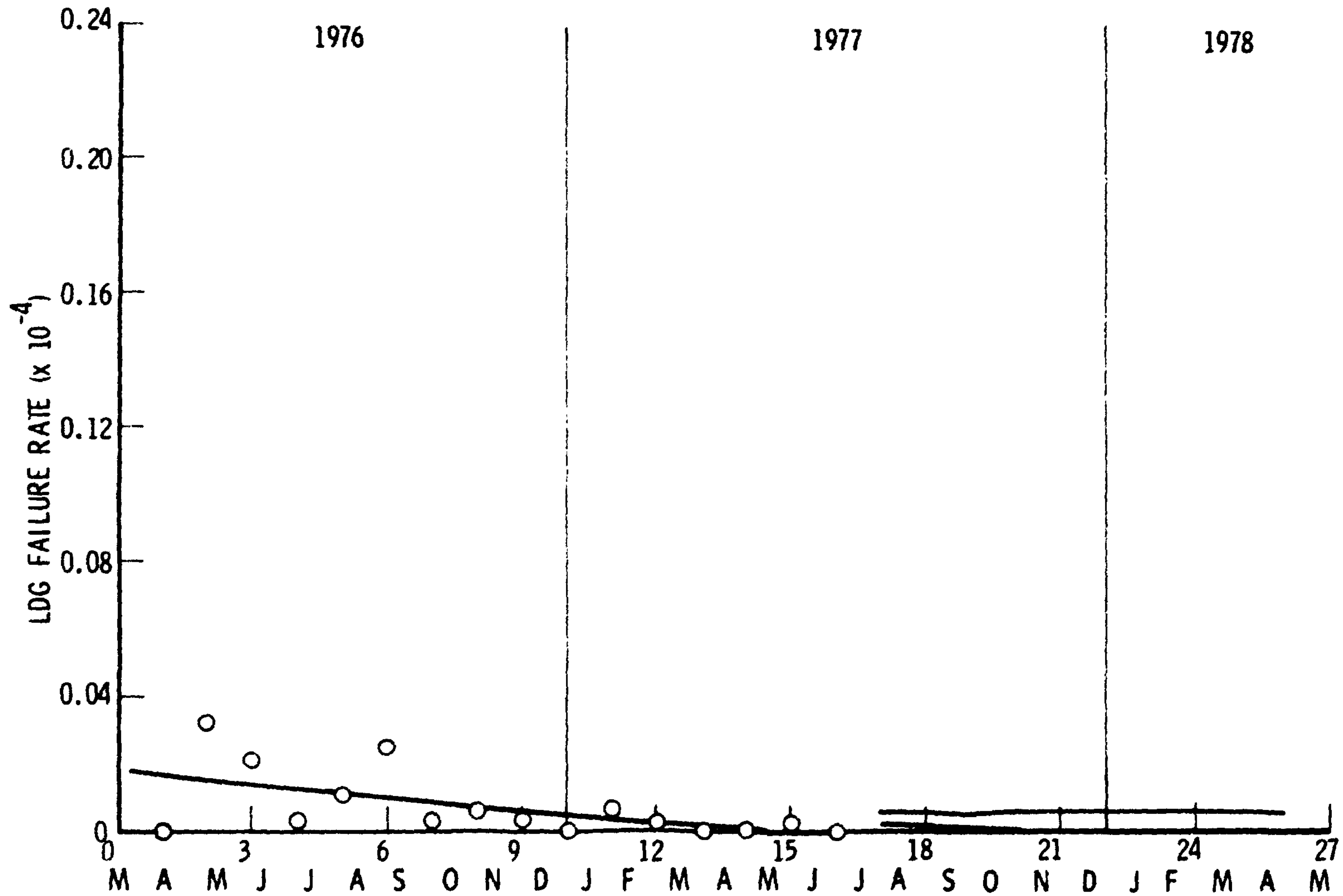

Figure 5-10. LDG Failure Rate - Normalized by Number of Statements

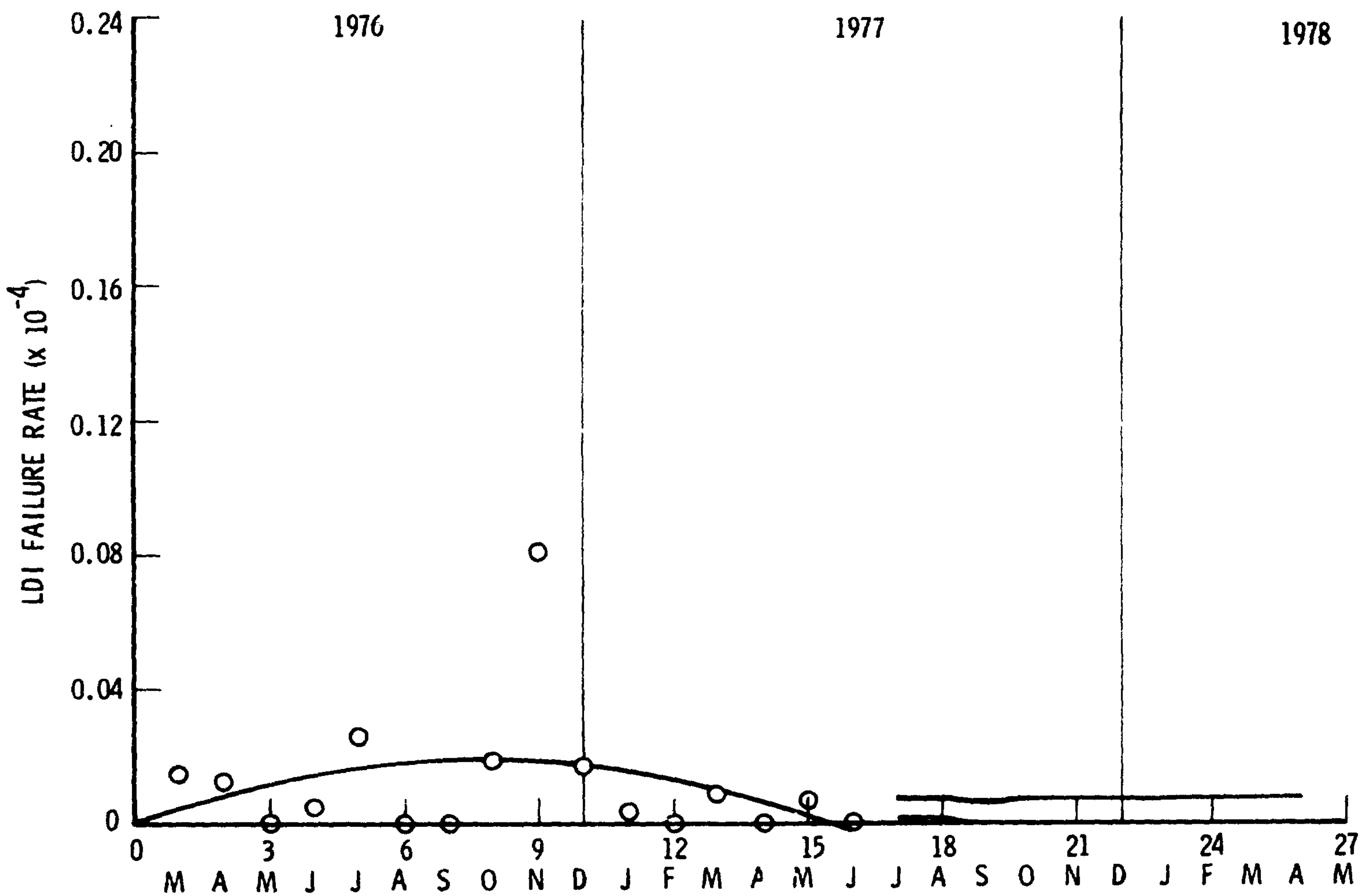

Figure 5-11. LDI Failure Rate — Normalized by Number of Statements

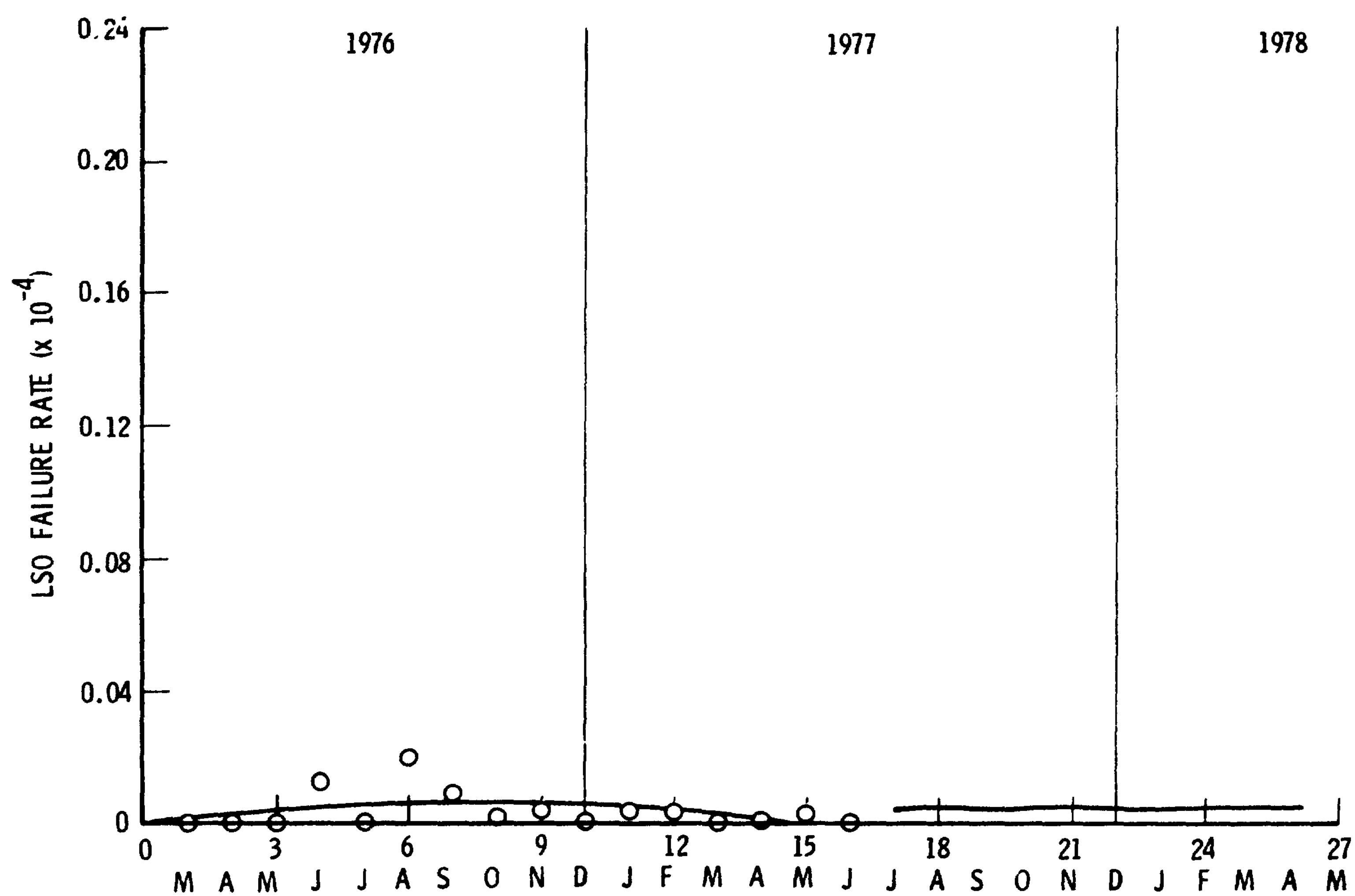

Figure 5-12. LSO Failure Rate — Normalized by Number of Statements

6. SIGNIFICANT FINDINGS

Specific findings from this study and potential applications include the following:

1. Meaningful measurement of software reliability during development is feasible. These measurements should be useful to line management as a systematic method for assessing the progress of software reliability and identifying and comparing sources.

2. Data acquisition for measurement of software reliability requires a deliberately distinct effort. The data normally recorded for systems records are not adequate for software reliability measurements. All personnel involved should be fully aware of this limitation.

3. Most of the failures during development were not due to coding errors but, rather, were caused by associated data processing procedures. Such an outcome suggests that management might be able to enhance program reliability during development by establishing standards for data handling and program operation in general. Time, effort, and costs should be reduced if appropriate procedures are implemented and conscientiously followed.

4. The failure processes are not accurately described by deterministic methods; stochastic processes are apparent. Therefore, simplistic generalized models should be closely scrutinized before being employed. A generalized method may be adapted to modelling of a specific case or set of data. However, the converse is not legitimate. Specifically, changing coefficients and exponents (of a deterministic model) that are derived from a single set of data does not produce a "generalized" model of anything.

5. Scheduling or other management actions appear to have a significant affect on the rate of occurrence of failure during development. Such interactions are apparent contributors to widely varying excursions in failure events. Line management, project management and functional (software development) management should be alert individually to the potential for such induced problems.

6. The natural outcome of some of the measurements produced data that were stratified into a limited number of categories. The analysis of such data must be restricted to theoretically sound and verified methods. Non-parametric (distribution free) methods should be used where appropriate and inverse transformations of results (as well as transformations of data) cannot be validated. Pretest of data acquisition procedures and instruments is strongly recommended.

7. Stochastic methods may be used at the end of a given time interval for estimating future reliability. This capability leads to criteria for definition of when to stop development testing. Examples are a forecast trend that is asymptotic to an acceptable level of error; or is stationary about zero. This should provide both management and researchers with a basic tool for comparison and assessment of programs for meeting future reliability goals, comparative reliability and comparison of the benefits of continued testing against incurred costs of time and effort.

PRECEDING PAGE BLANK NOT FILMED

7. CONCLUSIONS AND RECOMMENDATIONS

Data collected during the development of a software system needed for ground based launch support at the Air Force Space and Missile Test Center, Vandenberg Air Force Base, California, and from the operational Viking ground data processing system at the Jet Propulsion Laboratory, Pasadena, California was analyzed to determine if any valid measures of software reliability could be made that might have utility when applied to operational avionics systems to predict their reliability.

The failure rate (number of failures divided by CPU seconds for the calendar interval) and the failure ratio (number of failures divided by the total number of runs for the calendar interval) emerged as valid measures. They were subjected to linear, and to nonlinear orthogonal polynomial, regression analyses which confirmed their validity as indicators of system stability.

The composite failure rate and ratio data were also used to forecast the reliability of the system for nine months following the seventeen month test period for which data existed. The forecast predicted that the failure rate would converge to 0.002 and the ratio would converge to near zero after an initial three months at 0.025. This forecast could not be validated against real-world experience since the data collection process had ceased after the seventeen month period. This lack of corroborating data emphasizes the criticality of defining the scope of the data collection process at the outset to insure the availability of necessary data.

The raw data plots of failure rate and ratio exhibited both high and low points. Project staff at SAMTEC was queried as to any events that might have caused these and it was learned that the high points were all directly related to the start of intensive periods of testing and the lows to relative inactivity due to program review preparation. The concerned project manager should note from this that other than pure software problems can impact apparent progress.

The techniques of measurement discussed in this report appear promising as indicators of reliability. It is recommended that they be applied to operational avionics systems with a recorded history of failures to accomplish the further step of establishing an effective measure of software reliability analogous to hardware mean time to failure. Careful attention to data collection should be paid to insure the quality and continuity of the data base, including separation of actual software changes. The establishment and analysis of this data base would be a major contribution towards the goal of system certifiability.

APPENDIX A

REFERENCES

1. JOHNSON, J. P. "Software Reliability Measurement Study" SAMSO-TR-75-279, Aerospace Corporation, El Segundo, Ca. 8 December 1975.

2. HECHT, H. "Measurement, Estimation and Prediction of Software Reliability" NASA CR-145135, National Aeronautics and Space Administration, Washington, D. C., January 1977.

3. HECHT, H., STURM, W.A., and TRATTNER, S., "Reliability Measurement During Software Development," NASA-CR-145205, National Aeronautics and Space Administration, Langley Research Cer . Hampton, Va., September 1977.

4. HALSTEAD, M. H., Element of Software Science, Elseiver Publishing Co., New York, 1977.

5. KOPETZ, H., "On the Connections Between Range of Variab and Control Structure Testing" International Conference on Reliable Software, Los Angeles, 1975.

6. KENDALL, M.G., and STUART, A., The Advanced Theory of Statistics, 2d ed., 3 Vols., Hafner Publishing Co., New York 1968.

7. BOX, G. E. P., and JENKINS, G., Time Series Analysis Forecasting and Control, Holden-Day, San Francisco, 1970.

APPENDIX B

COMPUTER PROGRAMS

SNOBOL4 (VERSION 3.7, JUL. 10, 1971)

PROPRIETARY COMPUTER SYSTEMS INC

```
        &STLIMIT = 500000                                            1
        &ANCHOR = 1                                         00000020 2
                                                            00000030 3
        INPUT("DISK',1,80)                                  00000040 4
                                                            00000050 5
        DEFINE('INITMOD()')                                 00000060 6
        DEFINE('PRTMOD()')                                  00000070 7
                                                            00000080 8
        PAT1 = "C     ***** F"                              00000090 9
        PAT2 = "*"                                          00000100 10
        PAT3 = 'C'                                          00000110 11
        PAT4 = "EJECT"                                      00000120 12
        PAT5 = LEN(5) . P1  LEN(1) . P2 LEN(66) . P3  LEN(8) . P4  00000130 13
        PAT6 = "IMPLICIT" | "REAL" | "INTEGER" | "LOGICAL"           14
        PAT7 = "COMMON"                                     00000150 15
        PAT8 = "EQUIVALENCE"                                00000160 16
        PAT9 = "DIMENSION"                                  00000170 17
        PAT10 = "DATA"                                      00000180 18
        PAT11 = "CALL"                                      00000190 19
*     PATTERN FOR SKIPPING ONE OR MORE BLANKS
        PAT12 = NULL | SPAN(" ")                                     20
*     PATTERN FOR IF STATEMENTS
        PAT13 = "IF" PAT12 "("                                       21
*     PATTERN FOR EXECUTE STATEMENTS
        PAT14 = PAT12 "EXECUTE" PAT12 "("                            22
*     PATTERN FOR STOP STATEMENTS
        PAT15 = "STOP"                                               23
*     PATTERN FOR ELSE STATEMENTS
        PAT16 = "ELSE"                                               24
*     PATTERN FOR ALL END STATEMENTS
        PAT17 = "END"                                                25
*     PATTERN FOR LETTERS
        PAT18 = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"                         26
*     PATTERN FOR DIGITS
        PAT19 = "0123456789"                                         27
*     PATTEN FOR ALFHANUMERICS
        PAT20 = PAT18 PAT19                                          28
*     PATTERN FOR IDENTIFIERS
        PAT21 = PAT12 ANY(PAT18) (NULL | SPAN(PAT20))                29
*     PATTERN FOR <IDENTIFIER> = TYPE ASSIGNMENT STATEMENTS
        PAT22 = PAT21 PAT12 "="                                      30
                                                            00000210 31
*     PATTERN FOR <DO FOR> STATEMENTS
        PAT23 = "DO" PAT12 "FOR"                                     32
*     PATTERN FOR <UNDO> STATEMENTS
        PAT24 = "UNDO"                                               33
*     PATTERN FOR PROCEDURE STATEMENTS
        PAT25 = "PROCEDURE"                                          34
*     PATTERN FOR <CYCLE> STATEMENTS
        PAT26 = "CYCLE"                                              35
*     PATTERN FOR <DO CASE> STATEMENT
        PAT27 = "DO" PAT12 "CASE"                                    36
*     PATTERN FOR <CASE> STATEMENTS
        PAT28 = "CASE"                                               37
*     PATTERN FOR <WRITE> STATEMENTS
```

```
      PAT29 = "WRITE"                                                    38
*   PATTERN FOR <FORMAT  STATEMENTS
      PAT30 = "FORMAT"                                                   39
*   PATTERN FOR <IDENTIFIER> ( <ARB> ) = TYPE ASSIGNMENT STATEMENT
      PAT31 = PAT21 PAT12 "(" BREAK(")") ( ")" | ")" ) PAT12 "="         40
*   PATTERN FOR <SUBROUTINE> STATEMENTS
      PAT32 = "SUBROUTINE"                                               41
*   PATTERN FOR <RETURN> STATEMENTS
      PAT33 = "RETURN"                                                   42
*   PATTERN FOR BLOCK DATA
      PAT34 = "BLOCK"                                                    43
*   PATTERN FOR CSECT
      PAT35 = PAT12 "CSECT"                                              44
*   PATTERN FOR <DO UNTIL>
      PAT36 = "DO UNTIL"                                                 45
*   PATTERN FOR <NAMELIST>
      PAT37 = "NAMELIST"                                                 46
*   PATTERN FOR <REWIND>
      PAT38 = "REWIND"                                                   47
*   PATTERN FOR <READ>
      PAT39 = "READ"                                                     48
*   PATTERN FOR <DO LABEL>
      PAT40 = "DO LABEL"                                                 49
*   PATTERN FOR <LABEL>
      PAT41 = "LABEL"                                                    50
*   PATTERN FOR <EXIT>
      PAT42 = "EXIT"                                                     51
*   PATTERN FOR <DO WHILE>
      PAT43 = "DO WHILE"                                                 52
      BUF = DISK                                                00000220 53
      INITMOD()                                                 00000230 54
READLOOP  BUF = DISK   :F(XIT)                                  00000240 55
                                                                00000250 56
      BUF  PAT2   :S(SKIP)                                      00000260 57
      BUF  PAT1   :S(NEWMODULE)                                 00000270 58
      BUF  PAT3   :S(COMMENT)                                   00000280 59
      BUF  PAT4   :S(READLOOP)                                  00000290 60
      BUF  PAT5                                                 0[illegible]0 61
      P2 " "   :F(READLOOP)                                     00000310 62
      P3 PAT17   :S(ENDS )                                               63
      P3 PAT13   :S(IFST)                                                64
      P3 PAT11   :S(CALLST)                                     00000370 65
      P3 PAT14   :S(EXS )                                                66
      P3 PAT6  :S(TYPEST)                                       00000320 67
      P3 PAT16   :S(ELST)                                                68
      P3 PAT8  :S(EQUIVST)                                      00000340 69
      P3 PAT7  :S(CONNST)                                       00000330 70
      P3 PAT29   :S(NOWST)                                               71
      P3 PAT30   :S(NOFST)                                               72
      P3 PAT9  :S(DIMST)                                        00000350 73
      P3 PAT10   :S(DATST)                                      00000360 74
      P3 PAT15   :S(STPST)                                               75
      P3 PAT23   :S(NODOST)                                              76
      P3 PAT24   :S(NOUNST)                                              77
      P3 PAT25   :S(NOPROCST)                                            78
      P3 PAT26   :S(NOCYLST)                                             79
      P3 PAT27   :S(NODOCST)                                             80
      P3 PAT28   :S(NOCST)                                               81
      P3 PAT32   :S(SUBST)                                               82
      P3 PAT33   :S(RTNST)                                               83
```

```
      P3 PAT34  :S(BLKST)
      P3 PAT35  :S(READLOOP)
      P3 PAT36  :S(DOUNT)
      P3 PAT37  :S(NAMLST)
      P3 PAT38  :S(REWST)
      P3 PAT39  :S(READST)
      P3 PAT40  :S(DOLABST)
      P3 PAT41  :S(LABST)
      P3 PAT42  :S(EXITST)
      P3 PAT43  :S(DOWNST)
      P3 PAT22  :S(ASGST)
      P3 PAT31  :S(ASGST)

      NOUNREC = NOUNREC + 1
      OUTPUT = BUF  :(READLOOP)


SKIP  BUF = DISK  :F(XIT)
      BUF PAT1  :S(NEWMODULE)F(SKIP)

NEWMODULE  PRTMOD()
      INITMOD()  :(READLOOP)

COMMENT  NOCOM = NOCOM + 1  :(READLOOP)

TYPEST NOTYPE = NOTYPE + 1  :(READLOOP)

COMNST  NOCOMN = NOCOMN + 1  :(READLOOP)

EQUIVST  NOEQU = NOEQU + 1  :(READLOOP)

DIMST  NODIM = NODIM + 1  :(READLOOP)

DATST  NODAT = NODAT + 1  :(READLOOP)

CALLST  NOCALL = NOCALL + 1  :(READLOOP)

IFST  NOIF = NOIF + 1  :(READLOOP)

EXST   NOEXE = NOEXE + 1  :(READLOOP)

STPST  NOSTOP = NOSTOP + 1  :(READLOOP)

ELST  NOELSE = NOELSE + 1  :(READLOOP)

ENDST  NOEND = NOEND + 1  :(READLOOP)

ASGST   NOASG =NOASG + 1  :(READLOOP)

NODOST  NODOS = NODOS + 1  :(READLOOP)

NOUNST  NOUNS = NOUNS + 1  :(READLOOP)

NOPROCST  NOPROCS = NOPROCS + 1  :(READLOOP)

NOCYLST  NOCYLS = NOCYLS + 1  :(READLOOP)

NODOCST  NODCS = NODCS + 1  :(READLOOP)

NOCST  NOCASE = NOCASE + 1  :(READLOOP)
```

```
NOWST  NOWRT = NOWRT + 1  :(READLOOP)

NOFST  NOFMT = NOFMT + 1  :(READLOOP)

SUBST  NOSBS = NOSBS + 1  :(READLOOP)

RTNST  NORTNS = NORTNS + 1  :(READLOOP)

BLKST  BLKSK = BLKSK + 1  :(READLOOP)

DOWNT   DOWNTK = DOWNTK + 1  :(READLOOP)

NAMLST   NLK = NLK + 1  :(READLOOP)

REWST   REWK = REWK + 1  :(READLOOP)

READST   REEDK = REEDK + 1  :(READLOOP)

DOLABST   DLK = DLK + 1  :(READLOOP)

EXITST  EXK = EXK + 1  :(READLOOP)

DOWNST   DWK = DWK + 1  :(READLOOP)

LABST   LK = LK + 1  :(READLOOP)

XIT  PRTMOD()
     OUTPUT =
     OUTPUT =
     OUTPUT =
     OUTPUT = "END-OF-JOB"  :(END)

INITMOD  OUTPUT =
      OUTPUT =
      OUTPUT =
      OUTPUT = BUF
      OUTPUT =
      OUTPUT =
     NOCOM = 0
     NOTYPE = 0
     NOCOMN = 0
     NOEQU = 0
     NODIM = 0
     NODAT = 0
     NOCALL = 0
     NOIF = 0
     NOEXE = 0
     NOSTOP = 0
     NOELSE = 0
     NOEND = 0
     NOASS = 0
     NODOS = 0
     NOUNS = 0
     NOPROCS = 0
     NOCYLS = 0
     NODCS = 0
     NOCASE = 0
     NOWRT = 0
     NOFMT = 0
```

```
        NOSBS = 0
        NORTNS = 0
        BLKSK = 0
        DOUNTK = 0
        NLK = 0
        REWK = 0
        REEDK = 0
        DLK = 0
        EXK = 0
        DWK = 0
        LK = 0
        NOUNREC = 0   :(RETURN)

PRTMOD  OUTPUT =
        OUTPUT =
        OUTPUT =
        OUTPUT = "NUMBER OF COMMENT CARDS IS ..."  NOCOM
        OUTPUT = "NUMBER OF TYPE CARDS IS..." NOTYPE
        OUTPUT = "NUMBER OF COMMON CARDS IS..." NOCOMN
        OUTPUT = "NUMBER OF EQUIVALENCE CARDS IS..." NOEQU
        OUTPUT = "NUMBER OF DIMENSION CARDS IS ..." NODIM
        OUTPUT = "NUMBER OF DATA CARDS IS ..." NODAT
        OUTPUT = "NUMBER OF CALL STATEMENTS IS ..." NOCALL
        OUTPUT = "NUMBER OF IF STATEMENTS IS ..." NOIF
        OUTPUT = "NUMBER OF EXECUTE STATEMENTS IS..." NOEXE
        OUTPUT = "NUMBER OF STOP STATEMENTS IS ..." NOSTOP
        OUTPUT = "NUMBER OF ELSE STATEMENTS IS..." NOELSE
        OUTPUT = "NUMBER OF END STATEMENTS IS..." NOEND
        OUTPUT = "NUMBER OF ASSIGNMENT STATEMENTS IS..." NOASG
        OUTPUT = "NUMBER OF <DO FOR> STATEMENTS IS..." NODOS
        OUTPUT = "NUMBER OF <UNDO> STATEMENTS IS..." NOUNS
        OUTPUT = "NUMBER OF PROCEDURE STATEMENTS IS..." NOPROCS
        OUTPUT = "NUMBER OF <CYCLE> STATEMENTS IS..." NOCYLS
        OUTPUT = "NUMBER OF UNRECOGNIZED STATEMENTS IS..." NOUNREC
        OUTPUT = "NUMBER OF <DO CASE> STATEMENTS IS..." NODCS
        OUTPUT = "NUMBER OF <CASE> STATEMENTS IS..." NOCASE
        OUTPUT = "NUMBER OF WRITE STATEMENTS IS ..." NOWRT
        OUTPUT = "NUMBER OF FORMAT STATEMENTS IS..." NOFMT
        OUTPUT = "NUMBER OF <SUBROUTINE> STATEMENTS IS..." NOSBS
        OUTPUT = "NUMBER OF <RETURN> STATEMENTS IS..." NORTNS
        OUTPUT = "NUMBER OF BLOCK DATA STATEMENTS IS..." BLKSK
        OUTPUT = "NUMBER OF <DO UNTIL> STATEMENTS IS..." DOUNTK
        OUTPUT = "NUMBER OF NAMELIST STATEMENTS IS..." NLK
        OUTPUT = "NUMBER OF REWIND STATEMENTS IS..." REWK
        OUTPUT = "NUMBER OF READ STATEMENTS IS..." REEDK
        OUTPUT = "NUMBER OF <DO LABEL> STATEMENTS IS..." DLK
        OUTPUT = "NUMBER OF EXIT STATEMENTS IS..." EXK
        OUTPUT = "NUMBER OF LABEL STATEMENTS IS..." LK
        OUTPUT = "NUMBER OF <DO WHILE> STATEMENTS IS..." DWK
        OUTPUT =     :(RETURN)
END

NO ERRORS DETECTED IN SOURCE PROGRAM
```

```
C     ***** FUNCTION BASRIT *****                                   00000013

NUMBER OF COMMENT CARDS IS ...617
NUMBER OF TYPE CARDS IS...43
NUMBER OF COMMON CARDS IS...34
NUMBER OF EQUIVALENCE CARDS IS...36
NUMBER OF DIMENSION CARDS IS ...4
NUMBER OF DATA CARDS IS ...4
NUMBER OF CALL STATEMENTS IS ...67
NUMBER OF IF STATEMENTS IS ...107
NUMBER OF EXECUTE STATEMENTS IS...45
NUMBER OF STOP STATEMENTS IS ...5
NUMBER OF ELSE STATEMENTS IS...35
NUMBER OF END STATEMENTS IS...115
NUMBER OF ASSIGNMENT STATEMENTS IS...0
NUMBER OF <DO FOR> STATEMENTS IS ..17
NUMBER OF <UNDO> STATEMENTS IS...25
NUMBER OF PROCEDURE STATEMENTS IS...19
NUMBER OF <CYCLE> STATEMENTS IS...2
NUMBER OF UNRECOGNIZED STATEMENTS IS...0
NUMBER OF <DO CASE> STATEMENTS IS...3
NUMBER OF <CASE> STATEMENTS IS...16
NUMBER OF WRITE STATEMENTS IS ...16
NUMBER OF FORMAT STATEMENTS IS...16
NUMBER OF <SUBROUTINE> STATEMENTS IS...2
NUMBER OF <RETURN> STATEMENTS IS...2
NUMBER OF BLOCK DATA STATEMENTS IS...0
NUMBER OF <DO UNTIL> STATEMENTS IS...0
NUMBER OF NAMELIST STATEMENTS IS...0
NUMBER OF REWIND STATEMENTS IS...0
NUMBER OF READ STATEMENTS IS...0
NUMBER OF <DO LABEL> STATEMENTS IS...0
NUMBER OF EXIT STATEMENTS IS...331
NUMBER OF LABEL STATEMENTS IS...0
NUMBER OF <DO WHILE> STATEMENTS IS...0

C     ***** FUNCTION FEREIT *****                                   00000010

NUMBER OF COMMENT CARDS IS ...957
NUMBER OF TYPE CARDS IS...57
NUMBER OF COMMON CARDS IS...29
NUMBER OF EQUIVALENCE CARDS IS...45
NUMBER OF DIMENSION CARDS IS ...4
NUMBER OF DATA CARDS IS ...5
NUMBER OF CALL STATEMENTS IS ...76
```

```
NUMBER OF IF STATEMENTS IS ...109
NUMBER OF EXECUTE STATEMENTS IS...80
NUMBER OF STOP STATEMENTS IS ...5
NUMBER OF ELSE STATEMENTS IS...31
NUMBER OF END STATEMENTS IS...146
NUMBER OF ASSIGNMENT STATEMENTS IS...0
NUMBER OF <DO FOR> STATEMENTS IS...34
NUMBER OF <UNDO> STATEMENTS IS...14
NUMBER OF PROCEDURE STATEMENTS IS...33
NUMBER OF <CYCLE> STATEMENTS IS...0
NUMBER OF UNRECOGNIZED STATEMENTS IS...0
NUMBER OF <DO CASE> STATEMENTS IS...5
NUMBER OF <CASE> STATEMENTS IS...21
NUMBER OF WRITE STATEMENTS IS ...21
NUMBER OF FORMAT STATEMENTS IS...21
NUMBER OF <SUBROUTINE> STATEMENTS IS...4
NUMBER OF <RETURN> STATEMENTS IS...5
NUMBER OF BLOCK DATA STATEMENTS IS...0
NUMBER OF <DO UNTIL> STATEMENTS IS...0
NUMBER OF NAMELIST STATEMENTS IS...0
NUMBER OF REWIND STATEMENTS IS...0
NUMBER OF READ STATEMENTS IS...0
NUMBER OF <DO LABEL> STATEMENTS IS...0
NUMBER OF EXIT STATEMENTS IS...460
NUMBER OF LABEL STATEMENTS IS...0
NUMBER OF <DO WHILE> STATEMENTS IS...0

C     ***** FUNCTION GCDOIT *****                                    00001200

NUMBER OF COMMENT CARDS IS ...652
NUMBER OF TYPE CARDS IS...40
NUMBER OF COMMON CARDS IS...32
NUMBER OF EQUIVALENCE CARDS IS...13
NUMBER OF DIMENSION CARDS IS ...4
NUMBER OF DATA CARDS IS ...15
NUMBER OF CALL STATEMENTS IS ...58
NUMBER OF IF STATEMENTS IS ...148
NUMBER OF EXECUTE STATEMENTS IS...87
NUMBER OF STOP STATEMENTS IS ...8
NUMBER OF ELSE STATEMENTS IS...71
NUMBER OF END STATEMENTS IS...205
NUMBER OF ASSIGNMENT STATEMENTS IS...0
NUMBER OF <DO FOR> STATEMENTS IS...39
NUMBER OF <UNDO> STATEMENTS IS...22
NUMBER OF PROCEDURE STATEMENTS IS...39
NUMBER OF <CYCLE> STATEMENTS IS...1
NUMBER OF UNRECOGNIZED STATEMENTS IS...0
NUMBER OF <DO CASE> STATEMENTS IS...2
NUMBER OF <CASE> STATEMENTS IS...11
NUMBER OF WRITE STATEMENTS IS ...119
NUMBER OF FORMAT STATEMENTS IS...119
NUMBER OF <SUBROUTINE> STATEMENTS IS...4
NUMBER OF <RETURN> STATEMENTS IS...5
```

```
NUMBER OF COMMENT CARDS IS ...3679
NUMBER OF TYPE CARDS IS...202
NUMBER OF COMMON CARDS IS...168
NUMBER OF EQUIVALENCE CARDS IS...30
NUMBER OF DIMENSION CARDS IS ...9
NUMBER OF DATA CARDS IS ...20
NUMBER OF CALL STATEMENTS IS ...182
NUMBER OF IF STATEMENTS IS ...304
NUMBER OF EXECUTE STATEMENTS IS...182
NUMBER OF STOP STATEMENTS IS ...8
NUMBER OF ELSE STATEMENTS IS...102
NUMBER OF END STATEMENTS IS...338
NUMBER OF ASSIGNMENT STATEMENTS IS...0
NUMBER OF <DO FOR> STATEMENTS IS...82
NUMBER OF <UNDO> STATEMENTS IS...35
NUMBER OF PROCEDURE STATEMENTS IS...64
NUMBER OF <CYCLE> STATEMENTS IS...2
NUMBER OF UNRECOGNIZED STATEMENTS IS...0
NUMBER OF <DO CASE> STATEMENTS IS...7
NUMBER OF <CASE> STATEMENTS IS...61
NUMBER OF WRITE STATEMENTS IS ...218
NUMBER OF FORMAT STATEMENTS IS...193
NUMBER OF <SUBROUTINE> STATEMENTS IS...9
NUMBER OF <RETURN> STATEMENTS IS...10
NUMBER OF BLOCK DATA STATEMENTS IS...1
NUMBER OF <DO UNTIL> STATEMENTS IS...0
NUMBER OF NAMELIST STATEMENTS IS...0
NUMBER OF REWIND STATEMENTS IS...0
NUMBER OF READ STATEMENTS IS...0
NUMBER OF <DO LABEL> STATEMENTS IS...0
NUMBER OF EXIT STATEMENTS IS...1009
NUMBER OF LABEL STATEMENTS IS...0
NUMBER OF <DO WHILE> STATEMENTS IS...0


END-OF-JOB
```

```
NUMBER OF BLOCK DATA STATEMENTS IS...1
NUMBER OF <DO UNTIL> STATEMENTS IS...0
NUMBER OF NAMELIST STATEMENTS IS...0
NUMBER OF REWIND STATEMENTS IS...0
NUMBER OF READ STATEMENTS IS...0
NUMBER OF <DO LABEL> STATEMENTS IS...0
NUMBER OF EXIT STATEMENTS IS...563
NUMBER OF LABEL STATEMENTS IS...0
NUMBER OF <DO WHILE> STATEMENTS IS...0


C     ***** FUNCTION STUFIT *****                                  00000010


NUMBER OF COMMENT CARDS IS ...452
NUMBER OF TYPE CARDS IS...26
NUMBER OF COMMON CARDS IS...34
NUMBER OF EQUIVALENCE CARDS IS...8
NUMBER OF DIMENSION CARDS IS ...2
NUMBER OF DATA CARDS IS ...17
NUMBER OF CALL STATEMENTS IS ...28
NUMBER OF IF STATEMENTS IS ...143
NUMBER OF EXECUTE STATEMENTS IS...102
NUMBER OF STOP STATEMENTS IS ...6
NUMBER OF ELSE STATEMENTS IS...58
NUMBER OF END STATEMENTS IS...197
NUMBER OF ASSIGNMENT STATEMENTS IS...0
NUMBER OF <DO FOR> STATEMENTS IS...35
NUMBER OF <UNDO> STATEMENTS IS...28
NUMBER OF PROCEDURE STATEMENTS IS...30
NUMBER OF <CYCLE> STATEMENTS IS...0
NUMBER OF UNRECOGNIZED STATEMENTS IS...0
NUMBER OF <DO CASE> STATEMENTS IS...10
NUMBER OF <CASE> STATEMENTS IS...59
NUMBER OF WRITE STATEMENTS IS ...36
NUMBER OF FORMAT STATEMENTS I`...35
NUMBER OF <SUBROUTINE> STATEMENTS IS...3
NUMBER OF <RETURN> STATEMENTS IS...5
NUMBER OF BLOCK DATA STATEMENTS IS...0
NUMBER OF <DO UNTIL> STATEMENTS IS...4
NUMBER OF NAMELIST STATEMENTS IS...1
NUMBER OF REWIND STATEMENTS IS...2
NUMBER OF READ STATEMENTS IS...3
NUMBER OF <DO LABEL> STATEMENTS IS...2
NUMBER OF EXIT STATEMENTS IS...340
NUMBER OF LABEL STATEMENTS IS...2
NUMBER OF <DO WHILE> STATEMENTS IS...0


C     ***** FUNCTION WRITON *****                                  00000015
```

APPENDIX C

DATA ACQUISITION FORMS

COMPUTER PROGRAM RUN ANALYSIS REPORT INSTRUCTIONS

To be filled out by programming librarian or responsible programmer after each computer run. If the run was unsucessful (SYNTAX errors, abort, calculation error, loop, etc.), the supplemental form <u>*COMPUTER PROGRAM FAILURE ANALYSIS REPORT*</u> *should also be complete. This form will yield error statistic data and computer run time data.*

1. Use program mnemonic.
2. This time is start time of computer execution from the computer printout.
3. If answer is no, complete <u>COMPUTER PROGRAM FAILURE ANALYSIS REPORT</u>.
4. This can be gotten from the computer printout.
5. Check the appropriate box.
6. Check the appropriate box.
7. Check the appropriate box.
8. Check the appropriate box.

SYSTEM________________________ DATE________________________

COMPUTER PROGRAM RUN ANALYSIS REPORT.

1. Computer Program Component ID ______________________________

2. Run Date: __ __ Day __ __ Mon __ __ Yr __ __ Hr __ __ Min

3. Successful Run? ________________

4. CPU Time: __ __ Min __ __. __ __ __ Sec

5. Category of Work:

   a. Program Development [ ]

   b. Program Modification:

      (1) Implementation of Additional Requirement [ ]

      (2) Implementation of Hardware Change [ ]

      (3) Memory/Time Optimization Enhancement [ ]

      (4) Error Correction [ ]

      (5) Design Modification [ ]

   c. Program Conversion [ ]

   d. Other ________________________ [ ]

6. CPCI/CPC Status

   a. CPC Test and Eval [ ] c. Full Integ. Test [ ]

   b. Partial Integ. Test [ ] d. Production Program [ ]

   e. Other ________________ [ ]

7. Program Activity

   a. Compilation [ ] c. Run with no compile [ ]

   b. Compile and run [ ] d. Other __________ [ ]

8. Number of Source Statements Changed/Deleted Inserted

a.	None	[ ]	e.	31-40	[ ]	i.	101-150	[ ]
b.	1-10	[ ]	f.	41-50	[ ]	j.	151-200	[ ]
c.	11-20	[ ]	g.	51-75	[ ]	k.	Over 200	[ ]
d.	21-30	[ ]	h.	76-100	[ ]			

Contact ________________

COMPUTER PROGRAM FAILURE ANALYSIS REPORT
INSTRUCTIONS

To be filled out by the responsible developer for each unsuccessful run. The failure information should be available on the program printout or from the computer operator. The error data can be derived from an analysis of the program output. (It is possible that a failure can be caused by more than one error, list them all).

1. Use program mnemonic.

2. This time is start time of computer execution from the computer printout.

3. Check box which most nearly describes the failure indication. If other is checked, briefly describe failure.

4. The count under the error category means number of errors not number of erroneous statements.

   A. Examples of COMPUTATIONAL ERRORS include: (1) Incorrect operand in equation, (2) Incorrect use of parenthesis, (3) Sign convention error (4) Units or data conversion error, (5) Computation produces an over/under flow, (6) Incorrect equation used, (7) Precision lost due to mixed mode, (8) Missing computations, (9) Rounding or truncation error and loop.

   B. Examples of LOGIC ERRORS include: (1) Incorrect operand in logical expression (2) Logic activities out of sequence, (3) Wrong variable being checked, (4) Missing logic or condition tests, (5) Too many/too few statements in loop, (6) Loop iterated incorrect number of times (including endless loop).

   C. Examples of DATA INPUT ERRORS include: (1) Invalid input read from correct data file, (2) Input read from incorrect data file, (3) Incorrect input format, (4) Incorrect format statement referenced, (5) EOF encountered prematurely, (6) EOF missing.

   D. Examples of DATA HANDLING ERRORS include: (1) Data file not rewound before reading, (2) Data initialization not done, (3) Data initialization done improperly, (4) Variable used as a flag or index not set properly, (5) Variable referred to by wrong name, (6) Variable type is incorrect, (7) Data packing/unpacking error, (8) Sort error, (9) Subscripting error.

   E. Examples of DATA OUTPUT ERRORS include: (1) Data written on wrong file, (2) Data written using wrong format statement, (3) Data written in the wrong format, (4) Data written with wrong carriage control, (5) Incomplete or missing output, (6) Output field size to small, (7) Line count and page eject problems.

   F. Examples of INTERFACE ERRORS include: (1) Wrong subroutine called, (2) Call to subroutine made in wrong place, (3) Subroutine arguments not consistent in type, units, order, etc. (4) Subroutine called is nonexistent.

   G. Examples of ARRAY PROCESSING ERRORS include: (1) Array not properly dimensioned, (2) Array referenced out of bounds, (3) Array being referenced at incorrect location, (4) Array pointers not incremented properly.

   H. Examples of DATA BASE ERRORS include: (1) Data should have been initialized in data base but wasn't, (2) Data initialized to incorrect value in data base, (3) Data base units are incorrect.

   I. Examples of OPERATION ERRROS include: (1) Operating system error, (2) Hardware error, (3) Operator error, (4) Test execution error.

   J. Examples of PROGRAM EXECUTION ERRORS include: (1) Time limit exceeded, (2) Core storage limit exceeded, (3) Output line limit exceeded, (4) Compilation error.

   K. Examples of DOCUMENTATION ERRORS include: (1) User manual error, (2) Interface spec error, (3) Design spec error, (4) Requirements spec error.

   L. Briefly describe the error(s).

SYSTEM ______________________ DATE ______________________

COMPUTER PROGRAM FAILURE ANALYSIS REPORT

1. Computer Program Component ID ______________________

2. Run Date: ___ Day ___ Mon ___ Yr ___ Hr ___ Min

3. Severity of Failure

A.	Caused Complete System to Crash	[ ]
B.	Caused A Dependent Job to Fail	[ ]
C.	Local Job Failure Only	[ ]
D.	Real Time Failure	[ ]
E.	Other ______________	[ ]

4. Error Category

			Count
A.	Computational Error	[ ]	______
B.	Logic Error	[ ]	______
C.	Data Input Error	[ ]	______
D.	Data Handling Error	[ ]	______
E.	Data Output Error	[ ]	______
F.	Interface Error	[ ]	______
G.	Array Processing Error	[ ]	______
H.	Data Base Error	[ ]	______
I.	Operation Error	[ ]	______
J.	Program Execution Error	[ ]	______
K.	Documentation Error	[ ]	______
L.	Other ______________	[ ]	______

Contact ______________________

APPENDIX D

BIBLIOGRAPHY

Barlow, Richard and Scheuer, Ernest, "Reliability Growth During a Development Testing Program," Technometrics, Feb 1966, Vol. 8, No. 1, p. 53.

Abstract: The problem of estimating reliability of a system undergoing development testing is examined. It is assumed that the test program is conducted in K stages and that similar items are tested within each stage. In addition, it is assumed that the probability of an inherent failure, q_o, remains constant throughout the test program while the probability of an assignable cause failure in the i-th stage, q_i, does not increase with i. The number of inherent failures, of assignable cause failures, and of successes is recorded in each stage. Maximum likelihood estimates of q_o, $q_i (i = 1, 2, \cdots, K)$ and a conservative confidence bound for the reliability in the K-th stage are obtained. Numerical examples to illustrate the methods are given.

Belady, L.A.; Lehman, N. M.; "An Introduction to Growth Dynamics", Statistical Computer Performance Evaluation, Freiberger (ed.) Academic Press, New York, 1972, 503-511.

Belady, L. A.; Lehman, M. M.: "On the Macro-Dynamics of Programming and Other Systems", in preparation Fall 1971, disposition unknown on 27 March 1973.

Belady, L. A.: Lehman, M. M.: "Programming Systems Growth Dynamics", IBM Research Division RC3546, September 1971.

Berkovitz, Shimshon: "The Calculation of Availability of Systems with Arbitrary Structure and Success Criteria". MTR 2314, The MITRE Corporation, Bedford, Massachusetts, 17 May 1972.

Abstract: A recursive scheme for computing system availability from component element availabilities is developed and displayed. It requires only the knowledge of all minimal sets of components needed for successful system operation. It does not rely on any series, parallel or bridge structure but is applicable to the most general redundant systems. The scheme can be truncated at any level of recursion to yield good approximations. Included is a discussion of how the level of truncation depends on the number of system components, their availabilities and the desired accuracy.

Boehm, Barry W.: "Software and Its Impact: A Quantitative Assessment", Datamation, May, 1973, 48-59.

Brown, J. R. and Lipow, M., "Testing for Software Reliability," Proceedings, 1975 International Conference on Reliable Software, IEEE Catalog No. 75 CH0940-7 CSR.

Abstract: This paper presents a formulation of a novel methodology for evaluation of testing in support of operational reliability assessment and prediction. The methodology features an incremental evaluation of the representativeness of a set of development and validation test cases together with definition of additional test cases to enhance those qualities. If test cases are derived in typical fashion (i.e., to find and remove bugs, to investigate software performance under off-nominal conditions, to exercise structural elements and functional capabilities of the software, and to demonstrate satisfaction of software requirements), then the complete set of test cases is not necessarily representative of anticipated operational usage. The paper reports on initial research into formulation of valid measures of testing representativeness.

Buckley, F. J., "Software Testing - A Report From the Field, " Proc. 1973 IEEE Symposium Computer Software Reliability, Brooklyn Polytechnic Institute, April 1973, pp 102-106.

Chandy, K. M.; Ramamoorthy, C. V.; Cowan, A.: "A Framework for Hardware-Software Tradeoffs in the Design of Fault-Tolerant Computers", AFIPS Conference Proceedings, Volume 41, Part I, 1972, 53-63.

Abstract: Our approach to reliability rests on a framework of four indices called the Hardware Reliability Efficiency index (HRE), the Software Reliability Efficiency index (SRE), the Real-Time Criticality index (RTC) of a system, and the inclusion factor. For a given method of achieving reliability HRE and SRE are measures of the increase in reliability of the system per unit of expenditure. For the same amount of expenditure, a method with a high HRE (or SRE) gives better reliability than a method with low HRE (or SRE). In this paper we shall discuss ways of computing the fficiency indices for several different reliability methods. The real-time criticality index is a measure of the penalty incurred for a late completion of the system mission. Thus an air-traffic control system would have a high RTC compared to other systems. The inclusion factor (defined later) is a dimensionless number; if the inclusion factor for a given method is less than one, then that method should not be used in the system. The inclusion factor is a function of the method being considered and of system objectives. Thus a given technique may be optimally included in the design of one system and excluded from another.

Coutinho, John de S.: "Software Reliability Growth", Record, IEEE Symposium on Computer Software Reliability, 1973.

Craig, G. R., Hetrick W. L., Lipow, M., Thayer, T. A. et al, Software Reliability Study", TRW Systems Group, Interim Technical Report, RADC-TR-74-250, Oct 74 (Under RADC Contract F30602-74-C-0036, Software Reliability Study). (AD787784/8GI)
Abstract: The study of software error types, techniques for locating them, and recommendations for improvement of reliability are discussed. Interim results from a study of errors encountered in three large software packages are presented. Data collection and analysis schemes are summarized for subject data sets; and plans for data collection on a fourth software project are outlined. Finally, a survey of present software reliability models and a summary of TRW work in this area are given.

Davenport, Wilbur B., Jr.: Probability and Random Processes, McGraw-Hill Book Company, New York, 1970.
"Dept. of Defense, Military Standardization Handbook--Reliability Prediction of Electronic Equipment, MIL-HDBK-217B (Sept 1974).

Dickson, J. C., Hesse, J. L., Kientz, A. C., and Shooman, M.L., Quantitative Analysis of Software Reliability."
Abstract: Although reliability engineering has matured as a discipline in the past decade under the pressure of increasing user requirements, little formal thought has been given to a systems reliability approach encompassing both hardware and software aspects. The preponderance of existing literature has concentrated on formal verification techniques for existing firmware or software, leaving the reliability analysis realm essentially untouched. Since many applications such as space programs, airline reservation systems, and military weapon systems require high reliability assurance prior to release to the user, the purpose of this paper is to suggest a methodology suitable for use in system reliability studies. The prediction model which is developed is based on error correction rates, and is applied to the time profile of these rates for several classes of data.

Proceedings, 1972 Annual Reliability and Maintainability Symposium, IEEE Catalog No. 72 CH0577-7 R, pp. 148-157.

Ellingson, O. E. "Computer Program and Change Control," Proc. 1973 IEEE Symposium Computer Software Reliability, Brooklyn Polytechnic Institute, April 1973, pp. 82-89.

Endres, A. "An Analysis of Errors and Their Causes in System Programs," Proceedings, 1975 International Conference on Reliable Software, IEEE Catalog No. 75 CH0940-7CSR.

Feller, W.: An Introduction to Probability Theory and Applications, Wiley, New York, 1957.

Floyd, R. W., "Assigning Meanings to Programs," Proc. Symp. in App. Math., Vol. 19, American Math Society, Providence, R.I., 1967, pp. 19-32.

Funami, Y. and Halstead, M.H., "A Software Physics Analysis of Akiyama's Debugging Data," Proceedings of the MRI Symposium on Computer Software Engineering, Polytechnic Institute of New York (April 1976).

Abstract: It is probably obvious that B, the number of errors that a programmer might make in implementing any given algorithm in any given programming language, depends upon the total number of opportunities for making an error. Until recently, it has also been equally obvious that there was little reason to expect that such a basic quantity even existed, and even less reason to suspect that it could be measured. Recent discoveries in an area of Natural Science called Algorithm Dynamics or Software Physics (6, 8, 9, 10, 14), however, include a simple hypothesis which relates a set of measurable parameters of a program to the total number of elementary mental discriminations required to generate that program. A few experiments on Programmer Productivity (5, 7, 11, 13) have suggested that the hypothesis successfully accounts for the combined effects of program volume and program difficulty.

None of the reported studies have specifically addressed the application to program bugs. Yet if the hypothesis is in reasonable agreement with reality it yields the total number of elementary mental discriminations required in writing a program, and this must also be the total number of possibilities for making an erroneous discrimination. In the following sections we will reproduce the hypothesis and apply it to an independent set of data presented to the Lubjana Conference by Akiyama in 1971.

Girard, E. and Rault, J.D., "A Programming Technique for Software Reliability," Proc. 1973 IEEE Symp. Computer Software Reliability, Brooklyn Poly. Inst., April 1973, pp. 44-50.

Abstract: The overall feature of software products and traditional programming techniques are first reviewed. Then we propose and describe a two-step programming technique which, among other advantages, allows one to enhance software testing and reliability. It is shown how this technique can be included in two different program testing schemes (probabilistic and deterministic) and used to assess quantitatively program reliability.

Gnedenko, B. V., Belyayev Y. K., and Solovyev, A. D., Mathematical Methods of Reliability Theory, Academic Press, New York, 1969.

Goodenough, J. B., and Gerhart, S.L., "Toward a Theory of Test Data Selection," Proceedings, 1975 International Conference on, Reliable Software, IEEE Catalog No. 75 CH0940-7CSR.

Abstract: This paper examines the theoretical and practical role of testing in software development. We prove a fundamental theorem showing that properly structured tests are capable of demonstrating the absence of errors in a program. The theorem's proof hinges on our definition of test reliability and validity, but its practical utility hinges on being able to show when a test is actually reliable. We explain what makes tests unreliable (for example, we show by example why testing all program statements, predicates, or paths is not usually sufficient to insure test reliability), and we outline a possible approach to developing reliable tests. We also show how the analysis required to define reliable tests can help in checking a program's design and specifications as well as in preventing and detecting implementation errors.

Green, T. F., and Schneidewind, Howard, G. T., and Pariseau, R. J. "Program Structure Complexity and Error Characteristics," Proceedings of the Symposium on Computer Software Engineering XXIV, MRI Symposia Series, Polytechnic Press, Brooklyn, NY (1976).

The ability to detech and correct errors in a computer program is governed to a great extent by the structure of the program. Structure is important in two ways: (1) errors are more difficult to find in complex structures; and (2) more errors are generated initially during programming with complex structures. A method of characterizing structure is to represent the program logic in the form of a directed graph, where nodes and area represent decision instructions and straight line coding, respectively. This representation can be analyzed in terms of the following measures; probability of reaching an arc with an input; test coverage achieved with N inputs; numbers of nodes and arcs; and ration of actual to maximum number of arcs. Since program structure is most meaningful when related to the distribution of possible errors in the program, the ability to detect errors in various structures is studied. This is accomplished by employing an error detection simulation model. The relationships which are analyzed are error detection and test coverage as a function of program structure and number of inputs. These functions would be used in the design of software to avoid structure which are difficult to test and during testing for allocating resources to tests in accordance with structure and error detection characteristics.

As expected, it was found that the ability to detect errors decreases with increasing complexity. This was caused by program coverage decreasing with increasing complexity. An interesting aspect of the results is the asymptotic nature of the functions, which demonstrates the difficulty of finding additional errors after a critical value of coverage has been achieved, where the critical value of coverage is relatively low in complex structures.

Haines, Andrew L.: "Some Contributions to the Theory of Restricted Classes of Distributions with Applications to Reliability", M73-35, The MITRE Corporation, Washington Operations, May 1973.

Haney, F. M.: "Module Connection Analysis - A Tool for Scheduling Software Debugging Activities", AFIPS Conference Proceedings, Volume 41, Part I, 1972, 173-179.

Hecht, H., Measurement, Estimation, and Prediction of Software Reliability, NASA CR-145135, National Aeronautics and Space Administration, Washington, DC (January 1977). Also in Software Engineering Techniques, Infotech International Ltd., Maindenhead, Berkshire, England, (1977), Vol. 2, p. 209-244.

IEEE: Record, 1973 IEEE Symposium on Computer Software Reliability, New York City, April 30 - May 2 1973, No. 73 CHO 0741-9 CSR.

Itoh, D., and Izutani, T., "FADEBUG-I, a New Tool for Program Debugging," Record 1973 IEEE Symposium on Computer Software Reliability, IEEE Catalog No. 73 CHO741-9 CSR.

Jaynes, Edwin T.: "Prior Probabilities", IEEE Trans. or Systems Science and Cybernetics, Vol. SSC-4, No. 3, September, 1968, 227-241.

Abstract: In decision theory, mathematical analysis shows that once the sampling distribution, loss function, and sample are specified, the only remaining basis for a choice among different admissible decisions lies in the prior probabilities. Therefore, the logical foundations of decision theory cannot be put in fully satisfactory form until the old problem of arbitrariness (sometimes called "subjectiveness") in assigning prior probabilities is resolved.

The principle of maximum entropy represents one step in this direction. Its use is illustrated, and a correspondence property between maximum-entropy probabilities and frequencies is demonstrated. The consistency of this

principle with the principles of conventional "direct probability" analysis is illustrated by showing that many known results may be derived by either method. However, an ambiguity remains in setting up a prior on a continuous parameter space because the results lack invariance under a change of parameters; thus a further principle is needed.

It is shown that in many problems, including some of the most important in practice, this ambiguity can be removed by applying methods of group theoretical reasoning which have long been used in theoretetical physics. By finding the group of transformations on the parameter space which convert the problem into an equivalent one, a basic desideratum of consistency can be stated in the form of functional equations which impose conditions on, and in some cases fully determine, an "invariant measure" on the parameter space. The method is illustrated for the case of location and scale parameters, rate constants, and in Bernoulli trials with unknown probability of success.

In realistic problems, both the transformation group analysis and the principle of maximum entropy are needed to determine the prior. The distributions thus found are uniquely determined by the prior information, independently of the choice of parameters. In a certain class of problems, therefore, the prior distributions may now be claimed to be fully as "objective" as the sampling distributions.

Jelinski, Z.: Moranda, P.: "Applications of a Probability-Based Model To a Code Experiment", Record, IEEE Symposium on Computer Software Reliability, 1973, 78-80.

Jelinski, Z.: Moranda, P.: "Software Reliability Research", Statistical Computer Performance Evaluation, Freiberger (Ed.), Academic Press, New York, 1972.

Abstract: Software reliability study was initiated by Advanced Information Systems subdivision of McDonnell Douglas Astronautics Company, Huntington Beach, California, to conduct research into the nature of the software reliability problem including definitions, contributing factors and means for control.

Discrepancy reports which originated during the development of two large-scale real-time systems form two separate primary data sources for the reliability study. A mathematical model, descriptively entitled the De-Eutrophication Process, was developed to describe the time pattern of the occurrence of discrepancies (errors). This model has been employed to estimate the initial (or residual) error content in a software package as well as to estimate the time between discrepancies at any phase of its development. Means of predicting mission success on the basis of errors which occur during testing are described.

Problems in categorizing software anomalies are described and the special area of the genesis of discrepancies during the integration of modules is discussed. Management techniques which should reduce the number of software anomalies are described.

Jelinski, Z.; Moranda, P.: "Applications of a Probability-Based Model to a Code Reading Experiment", Record, IEEE Symposium on Computer Software Reliability, 1973.

Johnson, J. P., Software Reliability Measurement Study, SAMSO-TR-75-279, Aerospace Corporation, El Segunda, CA (8 December 1975).

Abstract: The report contains plans for a complete software reliability measurement program using both manual and automatic data entry. The program is to be run in conjunction with SAMTEC at Vandenberg AFB in an effort to establish measurement and evaluation criteria for the advanced systematic techniques for reliable operational software (ASTROS)) project. An integral part of that project is the implementation and evaluation of structured programming techniques.

Included in the report are all forms necessary to describe the software development environment, the hierarchy and size of programming modules, and to capture any significant events that will affect programming and test while they are in progress. Forms and instructions for their use for manual data collections are included, as are descriptions of items that could be collected automatically.

Keezer E. I., "Practical Experiences in Establishing Software Quality Assurance," Proc. 1973 IEEE Symp. Computer Software Reliability, Brooklyn Poly. Inst., April 1973, pp. 132-135.

King, J. C., A Program Verifier, Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, 1969.

Abstract: This research is a first step toward developing a "verifying compiler." Such a compiler, as well as doing the standard translation of a program to a machine executable form, attempts to prove that the program is correct. In order to do this a program must be annotated with propositions in a mathematical notation which define the "correct: relations among the program variables. The verifying compiler then checks for consistency between the program and its propositions.

The thesis presents the theoretical basis of the method and then describes a prototype verifier in detail. This verifier, running on an IBM 300, operates on programs written in a simple programming language for integer arithmetic. Many programs have been automatically verified by this program.

These include a simple sort program, a program which examines a number for the property 'prime,' and a rather subtle program which raises an integer to an integeral power.

The formal analysis of a program produces "verification conditions: which must be proven to be theorems over integers. The verifier proves these theorems by using powerful formula simplification routines and specialized techniques for integer expressions. Ideas for improving this verifier and for building one which will operate on a more complicated programming language are presented.

Knuth, Donald E.: "An Empirical Study of FORTRAN Programs", CSD Report CS-186, Stanford University, 1970.

Abstract: A sample of programs, written in FORTRAN by a wide variety of people for a wide variety of applications, was chosen "at random" in an attempt to discover quantitatively "what programmers really do." Statistical results of this survey are presented here, together with some of their apparent implications for future work in compiler design. The principal conclusion which may be drawn is the importance of a program "profile," namely a table of frequency counts which record how often each statement is performed in a typical run; there are strong indications that profile-keeping should become a standard practice in all computer systems, for casual users as well as system programmers. This paper is the report of a three month study undertaken by the author and about a dozen students and representatives of the software industry during the summer 1970. It is hoped that a reader who studies this report will obtain a fairly clear conception of how FORTRAN is being used, and what compilers can do about it.

LaPadula, Leonard J., "Engineering of Quality Software Systems, Vol VIII - Software Reliability Modeling and Measurement Techniques", MITRE Corp., RADC-TR-74-325, Vol VIII, Final Technical Report (Jan 1 - Jun 30, 1973), Jan 1975 (Under RADC contract F19628-C-73-0001, Software Reliability and Timeliness). (AD A007773).

Abstract: This report presents an overview of the technological background common to the six tasks of project 522A, a part of MITRE Project 5220, The Advanced Systems Technology Program, under the direction of the Rome Air Development Center, United States Air Force. Besides discussing general background, this volume provides an introduction to each of the other seven volumes of the entire report.

LaPadula, L. J.; Clapp, J. A.: Engineering of Quality Software Systems", MTR-2648 Volume I, The MITRE Corporation, Bedford, Massachusetts, June 1973.

Lipow, M. Estimation of Software Package Residual Errors, TRW-SS-72-09, TRW Systems Group, Redondo Beach, CA (Nov 1972).

Lipow, M., "Maximum Likelihood Estimation of Parameters of a Software Time-To-Failure Distribution", TRW Systems Group, TRW Report No. 2260.1.9-73D-15(Rev 1), Jun 1973.

Lipow, M., "Some Variations of a Model for Software Time-To-Failure", TRW Systems Group, Correspondence ML-74-2260.1.9-21, Aug 1974.

Liskov, B. H.: "Guidelines for the Design and Implementation of Reliable Software Systems", MTR-2345, The MITRE Corporation, Bedford, Massachusetts, 14 April 1972.

Abstract: This document describes experimental guidelines governing the production of reliable software systems. Both programming and management guidelines are proposed. The programming guidelines are intended to enable programmers to cope with a complex system effectively. The management guidelines describe an organization of personnel intended to enhance the effect of the programming guidelines.

Littlewood, B. and Verrall, J. L. "A Bayesian Reliability Growth Model for Computer Software," Journal of the Royal Statistical Society, Series C, Applied Statistics, 1973.

Lloyd D. and Lipow, H. Reliability: Management Methods, and Mathematics, Prentice-Hall, Englewood Cliffs, New Jersey, 1964.

London, R. L. "Certification of the Algorithm Treesort," Comm. ACM. Vol. 13, No. 6, 1970, pp. 371-373.

Mac Williams, W., "Reliability of Large Real-Time Control Software Systems," Proc. 1973 'EEE Symp. Computer Software Reliability, Brooklyn Poly. Inst. April 1973, pp. 1-6.

Abstract: This paper is written from the point of view of the design of today's large and complex real-time computer-based control systems using multi-processor computers.

The software reliability is not under control as a design tool in anything like the hardware sense. In fact, it is not to clear how to define software reliability in a precise way and to measure it. What can we learn about software reliability by examining hardware reliability theory?

This paper may be viewed in terms of three levels of definition of software reliability: a) an overall or high-level definition, b) an intermediate-level definition (which might be termed a system designer's definition), and c) a low-level, measurement, or nitty-gritty definition.

Merritt, M. J. et al., Characteristics of Software Quality,"Report 25201-6001-RU-00, TRW Systems, Redondo Beach, CA (December 1973).

Miller, I. and Freund, J., Probability and Statistics for Engineers, Prentice-Hall, Englewood Cliffs, New Jersey, 1965.

MIL-STD-483, Configuration Management Practices . Sy. .ms, Equipment, Munitions and Computer Programs.

MIL-STD-490, Specification Practices.

Mills, H. D., "On the Statistical Validation of Computer Programs," IBM Report FSC-72-6015, July 1970.

MIPS (Metric Integrated Processing System) Performance and Design Requirements, System Segment Specification, MIPS-1023-3117,C6, Data Processing Directorate, Federal Electric Corporation, Vandenberg Air Force Base, Ca, Contract No. F04701-72-C-0203 (29 November 1976).

Moranda, P., "Probability-Based Models for the Failures During Burn-In Phase", Joint National Meeting ORSA/TIMS, Las Vegas, NV, Nov. 1975.

Moranda, P. B. and Jelinski, Z., "Software Reliability Research", Conference on Statistical Methods for the Evaluation of Computer Systems Performance, Providence, R.I., Nov 1971.

Moranda, P.B. and Jelinski, Z., "Final Report on Software Reliability Study". McDonnell Douglas Astronautics Company, MDC Report No. 63921, Dec 1972.

Munck, R. G.: "Discussion of Session VII, Software Reliability", Statistical Computer Performance Evaluation, Freiberger (ed.), Academic Press, New York, 1972, 513-514.

Nelson, E. C. A Statistical Basis for Software Reliability Assessment, TRW-SS-73-03, TRW Systems Group, Redondo Beach, CA (1973).

Abstract: A mathematical definition of the reliability of a computer program is developed from the mathematical definitions of a program and program execution given in Blum's mathematical theory of the semantics of programming languages. The reliability so defined is measurable and it is related to the structural properties of computer programs using concepts borrowed from the PACE system of automated software test tools.

Ogdin, Jerry L.: "Improving Software Reliability," Datamation, (January, 1973), 49-52.

Pierce, William H.: Failure-Tolerant Computer Design, Academic Press, New York, 1965.

Richards, F. R., "Computer Software; Testing, Reliability, Models, and Quality Assurance", Naval Postgraduate School, Monterey, CA. July 1974.

Rubey, R. J. "Quantitative Aspects of Software Validation," Proceedings, 1975 International Conference on Reliable Software, IEEE Catalog No. 75 CH0940-7 CSR.

Abstract: This paper discusses the need for quantitative descriptions of software errors and methods for gathering such data. The software development cycle is reviewed and the frequency of the errors that are detected during software development and independent validation are compared. Data obtained from validation efforts are presented, indicating the number of errors in 10 categories and three severity levels; the inferences that can be drawn from this data are discussed. Data describing the effectiveness of validation tools and techniques as a function of time are presented and discussed. The software validation cost is contrasted with the software development cost. The applications of better quantitative software error data are summarized.

Rudner, Beulah "Design of a Seeding/Tagging Reliability Test," in Summary of Technical Progress, Software Modling Studies, RADC-TR-76-143, Rome Air Development Center (May 1976).

Schick, G. J.; Wolverton, R. W.: "Assessment of Software Reliability", MDAC Paper WD 1872, McDonnell Douglas Corporation, August 1972.

Abstract: This paper discusses the problems in achieving reliability of large-scale software systems. Comparative studies of a contemporary U.S. Air Force software project, a NASA software project, and a commercial real-time software project are described. Software development and test management procedures which lead to software reliability are analyzed. The underlying premise advnaced is that software reliability must be designed into the system from the beginning using a systems approach. The paper describes the systems approach to software reliability which requires (1) understanding of the total software development and test life cycle, (2) identification of conventional and extended conventional test techniques for precision validation testing of applications programs, and (3) allocation of resources in a cost-and performance-effective manner, in advance, over the entire development period. The paper focuses on the testing approach, test planning and integration, deficiency reporting and control, and data collection and analysis.

Schneidewind, N. "A Methodology for Software Reliability Prediction and Quality Control," Naval Postgraduate School Technical Report NPS55SS72111A, November 1972.

Abstract: The increase in importance of software in command control and other complex systems requires increased attention to the problems of software reliability and quality control. This paper reports on initial attempts to develop a methodology for Naval Tactical Data System software reliability and presents the results of several statistical analyses which were performed in order to obtain an appreciation for the statistical characteristics of software reliability data. An approach to analyzing software reliability problems is outlined and a methodology for reliability prediction and quality control is presented. Characteristics of software reliability statistical distributions are reported.

Schneidewind, N. F. "An Approach to Software Reliability Prediction and Quality Control," <u>Fall Joint Computer Conference</u>, 1972, pp. 837-847.

Abstract: The increase in imporoance of software in command and control and other complex systems has not been accompanied by commensurate progress in the development of analytical techniques for the measurement of software quality and the prediction of software reliability. This paper presents a rationale for implementing software reliability programs; defines software reliability; and describes some of the problems of performing software reliability analysis. A software reliability program is outlined and a methodology for reliability prediction and quality control is presented. The results of initial efforts to develop a software reliability methodology at the Naval Electronics Laboratory Center are reported.

Shooman, M. L. and Natarajan, S. "Effect of Manpower Deployment and Error Generation on Software Reliability." Proceedings of the Symposium on Computer Software Engineering XXIV, MRI Symposia Series, Polytechnic Press, Brooklyn, NY (1976).

Shooman, Martin L.: "Operational Testing and Software Reliability Estimation During Program Development", Record, IEEE Symposium on Computer Software Reliability, 1973.

Abstract: This paper discusses some quantitative models which can be used to measure, manage, and predict the level of perfection (freedom from bugs) of software during the development and test stages. The measures used are the reliability function, R(t), and the mean time between software

failures, MTTF, both of which improve as more resources (time, man-hours, computer-hours) are expended on the program. The methodology described is most applicable to the last (but extensive) phase of software development generally called test and integration.

In order to calculate the MTTF and R(t) one needs test data on the system, or since we wish to predict, on a preliminary version of the system. The obvious choice is the succession of updated versions of the software produced during system integration. It is proposed that the functional software test program (system exerciser) written to test all large software systems be used to generate this data. The only additional efforts required over a normal test program to obtain the necessary data are: (a) careful post-analysis of test results to segregate hardware, software, and operator errors, and (b) running of the functional test occasionally during the entire system integration phase rather than just at the end.

A plot of the MTTF versus time yields a growth curve. Once several points on the curve have been established the future behavior (during test and integration and immediately after program release) can be predicted by extrapolation. Unless a technique well suited to the physical problem is used, extrapolation can be very misleading. A much better technique, requiring fewer data points for the same prediction accuracy, is to postulate an underlying model for error removal and use the test data to estimate the model constants. The error model used in this paper is based on previous work relating R(t) and MTTF to debugging data. The number of errors remaining in a software program is probabilistically modeled in terms of the number of errors corrected, the program size and the initial number of errors. An additional assumption is made that the software failure rate (crash rate) is proportional to the number of remaining errors. This allows one to write an expression for the software reliability and the mean time to software failure. To evaluate the two constants in the model, it is necessary to collect test data of the type previously described at a minimum of two separate points in the test and integration phase. If data is taken at more than two points the additional data sets may be used to study the consistency of the parameters and validate or suggest changes in the basic model. If the model is validated and the paired parameter estimates are consistent, then the data at the several test points can be used for a pooled estimate.

Shooman, Martin L.: "Probabilistic Models for Software Reliability Prediction", Statistical Computer Performance Evaluation, Freiberger (Ed.), Academic Press, New York, 1972.

Abstract: With the advent of large sophisticated hardware-software systems developed in the 1960s, the problem of computer system reliability has emerged. The reliability of computer hardware can be modeled in much the same way as other devices using conventional reliability theory; however, computer software errors require a different approach. This paper discusses a newly developed probabilistyic model for predicting software reliability. The model constants are calculated from error data collected from similar previous programs. The calculations result in a decreasing probability of no software errors versus operating time (reliability function). The rate at which reliability decreases is a function of the man-months of debugging time. Similarly, the mean time between operational software errors (MTBF) is obtained. The MTBF increases slowly and then more rapidly as the debugging effort (man-months) increases. The model permits estimation of software reliability before any code is written and allows later updating to inprove the accuracy of the parameters when integration or operational tests begin.

Shooman, M. L., "Software Reliability: Measurement and Models", 1975 Annual Reliability and Maintainability Symposium, Washington, DC, Jan 28-30, 1975.

Abstract: With the advent of large sophisticated hardware-software systems developed in the 1960s, the problem of computer system reliability has emerged. The reliability of computer hardware can be modeled in much the same way as other devices using conventional reliability theory; however, computer software errors require a different approach.

The paper begins by describing the types and causes of software errors and provides working definitions of software errors and software reliability. Some of the basid data on frequency of occurrence of errors is then discussed. The paper then summarizes and references some of the software reliability models which have been proposed and concentrates on one developed by the author.

This newly developed probabilistic model predicts reliability based on the initial number of errors in a program, the number removed, and the number remaining is the program. The model constants are calculated from operational test data on the software performance.

The calculations result in a decreasing probability of no software errors versus operating time (reliability function). The rate at which the reliability decreases is a function of the man-months of debugging time. Similarly, the mean time to occurrence of operational software errors (MTTF) is obtained. The MTTF increases slowly and then more rapidly as the debugging effort (man-months) increases. The model permits estimation of software reliability before any code is written and allows later updating to improve the accuracy of the prediction when integration or operational tests begin.

Shooman, M., et al, "Summary of Technical Progress Software Modeling Studies", Polytechnic Institute of New York, RADC-TR-75-245, Interim Report, Jun 1975 (Under RADC Contract F30602-75-C-0294) (AD A018 G18).

Abstract: During the period of time of 1 October 1974 to 30 June 1975, Polytechnic Institute of New York conducted research uinder RADC contract F30602-74-C-0294 in the area of software reliability. This report presents the progress of this research. Subjects of investigation were Markov models for the prediction of software availability, theortetical models for software testing, automatic programming, automatic testing of programs and collection of error data, estimation of the initial number of program errors, program complexity and hierarchies of computable functions.

Research into the use of Markov models for prediction of software availability has been completed and a report RADC-TR-75-169, "Computer Software Reliability: Many-State Markov Modeling Techniques," has been published covering this topic. This technique involves using a statistical model to predict the future performance of software using past performance data.

Theoretical models have been studied concerning software testing fot use in determining the minimum number of tests that are necessary to verify that a program has been completely tested. This involves determining the paths that are contained in the program and the number of tests necessary to test each path.

The seeding and tagging approach for estimating the number of software errors in a program has been investigated and experiments have been planned to verify this approach. This method of estimating the initial error content of a program involves several people debugging the same program. The total number of errors are then statistically determined using the number of errors found by each person that are contained in common with a "tagged" set of errors.

The possibility of reducing chances for program errors by matching the power of the programming language to the complexity of the problem being solved is being addressed by the investigation of hierarchies of computable functions defined bvy substitution and recursion. This research relates to the extension of basic automata theory to set up degrees of difficulty in computation and to adapt the schemata provided by recursive function theory to programming in higher level languages with more useful data types.

Sukert, A. N. "A Software Reliability Modeling Study" Rome Air Development Center (ISIS) Griffis Air Force Base, NY RADC-TR-76-247 Aug 1976 (AD A030437).

Thayer, T. A., et al., Software Reliability Study, Final Technical Report, 76-2260.1.9-5, TRW Defense and Space Systems Group, One Space Park, Redondo Beach, CA, Contract No. F30602-74-C-0036 (19 March 1976).
Abstract: A study of software errors is presented. Techniques for categorizing errors according to type, identifying their source, and detecting them are discussed. Various techniques used in analyzing empirical error data collected from four large software systems are discussed and results of analysis are presented. Use of results to indicate improvements in the error prevention and detection processes through use of tools and techniques is also discussed.
A survey of software reliability models is included, and recent work on TRW's Mathematical Theory of Software Reliability (MTSR) is presented.
Finally, lessons learned in conjunction with collecting software data are outlined, with recommendations for improving the data collection process.

Thompson, W. and Walsh, D. "Reliability and Confidence Limits for Computer Software," General Research Corporation Report.

Trauboth, H., "Guidelines for Documentation of Scientific Software Systems," Proc. 1973 IEEE Symp. Comput : Software Reliability, Brooklyn Poly. Inst., April 1973, pp. 124-131.

Tribus, Myron; Pitts, Gary: "The Widget roglem Revisited", IEEE Trans. on Systems Science and Cybernetics, Volume SSC-4, No. 3, September 1968, 241-248.
Abstract: The Jaynes "widget problem" is reviewed as an example of an application of the principle of maximum entropy in the making of decisions. The exact solution yields an unusual probability distribution. The problem illustrates why some kinds of decisions can be made intuitively and accurately, but would be difficult to rationalize without the principle of maximum entropy.

Trivedi, A. K. and Shooman, M., "Computer Software Reliability: Many-State Markov Modeling Techniques", Polytechnic Inst. of New York, RADC-TR-75-169, Interim Report, Jul 1975 (Under RADC contract F30602-74-C0294, Software Modeling Studies). (AD A0014824).
Abstract: Many-state Markov models have been developed for the purpose of providing quantitative reliability criteria for computer software. The software system under consideration is assumed to be large, so that statistical

deductions become meaningful, and is assumed to initially contain an unknown number of bugs. The basic models provide estimates and predictions for a quantifier that represents the state of debugging of the system and which is generally the most probable number of software errors that will have been corrected at a given time in the operation of this software system based upon preliminary modeling of the error occurrence rate and the error correction rate. The models also provide predictions for the availability and for the reliability of the system. The differential equations corresponding to the basic many-state Markov models are solved for verification and demonstrative purposes.

Manufacturer's data have been obtained on this performance of system software for a medium-sized software operating system. These data have been analyzed to obtain frequency distributions of the random variables representing the time to close software error reports. The data are then used for application of the basic many-state Markov model. A general discussion of error data collection is undertaken in some detail, and suggestions are made for possible improvements in software error data documentation practices.

Various extensions and modifications of the basic many-state Markov models are discussed. The classes of the so called many-state Markov G-Models and H-Models are developed to handle, respectively the case of arbitrary degress of system degradation and the case of various categories of system "down" states. The solutions and results of some of these cases are presented. Finally, the computational efficiency and tradeoffs involved in the solutions of the many-state Markov models are disucssed.

Tsichritzis, D. and Ballard, A. "Software Reliability," <u>INFOR</u>, Vol. 11, No. 2, June 1973, pp. 113-124.

Abstract: Our approach assumes that there is increasing interest in both practical and theoretical aspects of the reliability of computer software, and this paper reviews many aspects of software design and production which affect reliability. For the most part, the topics are discussed relative to simple examples, and with reference to the previous work of others; however, a new approach to formally proving system correctiness is presented. The system can be represented at any instance of time by its state. The progress of the system is represented by a state history. Any property can therefore be described as a relation between states. The correctness proof is an induction with respect to the sequence of such states followed during execution. The paper also covers, in review, program design, protection, programming style, testing and other topics.

Wagoner, W. L., "The Final Report on a Software Reliability Measurement Study", The Aerospace Corp., Report No TOR-0074(4112)-1, Aug 15, 1973.

Abstract: This report presents the final results of a Software Reliability Measurement Study performed by the author. The objectives of the study were as follows:

1. To establish a rudimentary definition of software reliability.

2. To identify parameters affecting software failure rates (e.g., program size, difficulty, programmer experience, schedule, etc).

3. To determine the critical parameters required for a software reliability model, including the distribution of software errors as a function of time.

The report includes:

1. A definition of terms relative to software reliability.

2. A section discussing software error detection rates and parameters which affect this process.

3. A summary of existing models and a comparison with a model proposed by the author.

4. An annotated list of ereferences on software reliability.

Weiss, H., "Estimation of Reliability Growth in a Comples Systems with Poisson Failure," Operations Research, Vol. 4, 1956, pp. 532-545.

Welker, E. L. and Lipow, M. "Estimating the Exponential Failure Rate from Data with No Failure Event," Proceedings 1974 Annual Reliability and Maintainability Symposium, IEEE Catalog No. 74CH0820-1RQC (January 1974).

Wolverton, R. W. and Schick, G. J., Assessment of Software Reliability, TRW-SS-72-04, TRW Systems Group, Redondo Beach, CA (1972). (Identical with paper in Proc. of the 11th Annual Meeting of the German Operations Research Society, Hamburg, GERMANY, Sept 1972.)

Zelen, Marvin, (ed.): Statistical Theory of Reliability, Proceedings of an Advanced Seminar Conducted by the Mathematics Research Center, US Army, at the University of Wisconsin, Madison, May 8-10, 1962, University of Wisconsin Press, 1963.