

N 78-10487

NASA Contractor Report 145205

Final Report

Reliability Measurement During Software Development

H. Hecht, W. A. Sturm,
and S. Trattner

The Aerospace Corporation
El Segundo, California

Prepared for
Langley Research Center
under Contract NAS1-14392

NASA

National Aeronautics
and Space Administration

Scientific and Technical
Information Office

1977

REPRODUCIBLE COPY
ORIGINAL CASTLE COPY

Report No.
NASA-CR-145205
[ATR-77(7590)-2]

RELIABILITY MEASUREMENT
DURING SOFTWARE DEVELOPMENT

Prepared by
H. Hecht, W. A. Sturm, and S. Trattner

September 1977

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

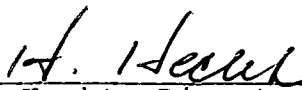
Prepared for
NASA Langley Research Center
Hampton, Virginia

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NASA-CR-145205	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Reliability Measurement During Software Development		5. TYPE OF REPORT & PERIOD COVERED Final Apr 1976-Apr 1977
		6. PERFORMING ORG. REPORT NUMBER ATR-77(7590)-2
7. AUTHOR(s) H. Hecht W. A. Sturm S. Trattner		8. CONTRACT OR GRANT NUMBER(s) NAS1-14392
9. PERFORMING ORGANIZATION NAME AND ADDRESS The Aerospace Corporation El Segundo, California		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS NASA Langley Research Center Hampton, Virginia		12. REPORT DATE September 1977
		13. NUMBER OF PAGES 96
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Software reliability Error types Reliability measurement Software reliability trend Software failure ratio Software failure rate		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) 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		


RELIABILITY MEASUREMENT
DURING SOFTWARE DEVELOPMENT

ATR-77(7590)-2

Prepared by



H. Hecht, Director
Digital Processing Office
Advanced Programs Division
Development Operations



G. W. Anderson, Group Director
Development Group Directorate
Advanced Programs Division
Development Operations

**Page
Intentionally
Left Blank**

9869

ACKNOWLEDGMENTS

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, and Mr. R. R. Hall of that organization was responsible for implementation of the data collection.

Much valuable assistance was received from Ms. Bonnie J. Schmidt both in the maintenance of the software reliability data base and in the preparation of this manuscript.

**Page
Intentionally
Left Blank**

CONTENTS

ACKNOWLEDGMENTS.....	vii
SUMMARY.....	1
I. INTRODUCTION.....	3
II. OVERVIEW.....	9
III. THE COMPUTER PROGRAM AND ITS ENVIRONMENT.....	17
IV. MEASUREMENTS AND TIME TRENDS.....	21
V. VARIATIONS BETWEEN MODULES.....	39
VI. CAUSES OF FAILURE.....	45
VII. FINDINGS AND POTENTIAL APPLICATIONS.....	53
VIII. WHERE DO WE GO FROM HERE?.....	57
REFERENCES.....	59
APPENDIX A - BACKGROUND DATA FORMS.....	A-1
APPENDIX B - ASTROS, ADVANCED SYSTEMATIC TECHNIQUES FOR RELIABLE OPERATIONAL SOFTWARE: ANOTHER LOOK.....	B-1

**Page
Intentionally
Left Blank**

0069
FIGURES

1.	Total and Program Failure Ratios for All.....	12
2.	Total Failures and Total Failure Ratios for All.....	14
3a.	Computer Program Run Analysis Report.....	22
3b.	Computer Program Run Analysis Report Instructions.....	23
4a.	Computer Program Failure Analysis Report.....	24
4b.	Computer Program Failure Analysis Report Instructions...	25
5.	Comparison of Hardware and Software Failure Ratios.....	29
6.	Total Failure Ratio for All.....	31
7.	Program Failure Ratio for Utilities.....	33
8.	Total Runs and Total Changes for Utilities.....	34
9.	Program Failure Ratio for LDG.....	37
10.	Program Failure Rate for LDG.....	38
A-1	General Project Summary.....	A-2
A-2	Program Schedule.....	A-4
A-3	Management Methodology Summary.....	A-5
A-4	Design and Processor Summary.....	A-7
A-5	Personnel Profile.....	A-8
A-6	Personnel Profile Summary.....	A-9
A-7	Testing Summary.....	A-11
A-8	Top-Level HIPO Chart.....	A-12

**Page
Intentionally
Left Blank**

TABLES

1.	Background Data Reporting Forms.....	20
2.	Major Module Reliability Summary.....	40
3.	Data for the First 100 Runs on Each Module.....	44
4.	Failure Ratios by Error Class for All.....	47
5.	Failure Ratios by Error Class for Modules.....	48
6.	Failure Ratios by Error Class for Utilities.....	49
7.	Relative Frequency of Errors (in percent).....	51

SUMMARY

Measurement of software reliability was carried out during the development of database software for a multi-sensor tracking system. Every run made during this project was scored as success or failure, and supporting data were collected on forms for further analysis. The failure ratio (number of failures per calendar interval divided by total number of runs) and failure rate (number of failures divided by CPU time for the interval) were found to be consistent measures, on a month-to-month basis as well as from module to module, and therefore considered valid indicators of reliability in this environment. 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.

**Page
Intentionally
Left Blank**

I. INTRODUCTION

Software reliability is essential to some of the nation's most widely publicized technical undertakings, e.g., the Space Transportation System and the control of nuclear power. The dramatic advancements now taking place in microelectronics and memory technologies, accompanied by significant reductions in computer hardware costs, will lead to even greater use of the programmable digital computer in the foreseeable future and hence increased dependence on software. The speed and flexibility of the computer make it, in principle, an ideal choice as a multi-variable controller in real-time systems ranging in complexity and physical scope from sophisticated aircraft flight controls to unmanned mass transportation systems. In these and other instances where the system being controlled interacts with the public at large the safety of increasingly large numbers of people will depend on the reliability of the software. Hence, software reliability--what it is, how to measure it, how to estimate and/or predict it, and how to achieve it--is a subject of more than just academic interest. This report addresses the first three of these aspects of software reliability with emphasis given to reliability measurement.

For the purposes 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 (and particularly 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. Typically, the software will not be modified during the period of measurement, and the developed reliability numeric is applicable to the measurement period and then-existing software configuration only.

Estimation of software reliability is performed by taking reliability measurements (as above) on an existing program and modifying the result to represent the reliability in a different operating environment. Estimation requires some quantifiable

relationship between the measurement environment and the environment for which the estimate is to be valid.

Prediction of software reliability is any statement about the reliability of a computer program that is not based on measurement taken on that particular program. While this terse definition may permit predictions based on casting of dice or even less respectable methods (which, according to rumors, are sometimes utilized for that purpose), prediction is normally based on comparison of program length, complexity, and environmental requirements with those of a program for which measurements exist.

The ability to measure and predict software reliability is required for the proper development and application of critical computer programs, but suitable measures for this purpose are not yet in general use. This study has identified two quantitative indices, the failure ratio and the failure rate, that promise to fill this need. They 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.

Although the motivation for this study is the measurement and improvement of software reliability in an operating environment,

the results reported here for the development phase are significant for three reasons.

- a. The cost of "fixing" software is low during this early stage and increases considerably after it is documented, formally accepted, etc. To the extent that reliability measurement during software development can point to problem areas, it will permit corrective effort at the most effective level.
- b. The level and the trend of reliability during development may be an indicator of reliability during formal test and operational phases. The validity of this assumption can only be established by following this (and possibly other software products) through further stages of the life cycle, and it is intended to do so.
- c. The data analysis and data presentation techniques developed here can be carried through test and operation. Trying out new data collection techniques might be very objectionable in an operational environment.

Following this introduction, an overview has been provided that outlines where and how the primary data were obtained, describes the general methods of analysis, and reviews a sample chart. The body of the report first describes the computer program on which these measurements were taken and the programming and data collection environments. The next section describes the measurement process, introduces the general analytical procedures, and presents findings on time trends of software reliability. This is followed by a section that summarizes data on individual modules of the computer program and discusses differences between modules in the measured quantities.

Causes of failures are then discussed, summaries on these are presented, and a comparison of causes of failures found here with those reported earlier is made. The next-to-last section presents a summary of the findings during this phase of the study and suggests applications of what has been learned so far. Readers who are confronted with immediate problems in software reliability measurement may want to start their perusal of this report with that material (Section 7). Recommendations for further work in this field are contained in the final section.

**Page
Intentionally
Left Blank**

II. OVERVIEW

The motivation for the study was the growing concern with the reliability of software in critical applications, and particularly in future air transport where the safety of aircraft and passengers demand flawless performance of software programs.

The data analyzed in this report were collected during the development of the Launch Support Data Base (LSDB), a portion of the Metric Integrated Processing System (MIPS). LSDB 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 data covers the development of the LSDB from coding through the in-house test phases prior to acceptance by the government. During the initial part of this period the code constantly expanded as runs were being made, and the effect of this on the reliability measurements is discussed later in this paper. During program development there was no unusual pressure to control reliability for current runs, but the aim was to produce software that would be reliable in operation.

LSDB was developed as part of a demonstration program on structured programming techniques. Personnel were motivated by their participation in a demonstration, and the data collection efforts and management attention may constitute further factors

for making some of the data presented here not completely representative. Arrangements have been made to collect similar data on another MIPS component where programming techniques are not prescribed. Assessment of unique factors may be possible by comparing reliability measurements from these two efforts. Further information on MIPS and LSDB is presented in Section 3.

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 (compile through execute). The run was scored as a success or failure by the development group. If it was identified as a failure, additional information, contained in the failure analysis report, was provided which identified the type of failure and the cause. This form, too, was prepared by the program development personnel. The principal metrics developed from the information in these forms are the failure ratio and the failure rate. The former is the ratio of failed runs to total runs over a given period, usually one month. The failure rate is the ratio of failed runs to the total CPU time accumulated over a given period, again usually one month. Both of these measures were evaluated for LSDB as a whole and for individual modules. Some differences in the information furnished by these measures are discussed in Sections 4 and 5.

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 for LSDB. 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.

Typical measures obtained in the study are depicted in Figure 1. The solid line represents the monthly failure ratios for the entire LSDB software as reported from March 1976 through January 1977. The dashed line is the program failure ratio as determined above for the same reporting period.

While the plots in Figure 1 seem to depict a "noisy" process it appears to be possible to make some observations regarding the behavior of the software reliability which, in turn, lead to conclusions regarding the utility of the measures employed. For example, in no month did the total failure ratio exceed 0.26 nor did it drop below 0.13. Therefore, it appears to be a reasonably stable and meaningful measure of the software development. Both the total failure ratio and the program failure ratio exhibit a general trend with time. By the use of linear regression, trend lines can be generated for the entire development period and/or for the most recent, say six-month intervals, to provide indicators of progress (or a lack thereof). The generation and use of these trend lines is discussed in Section 4. It should

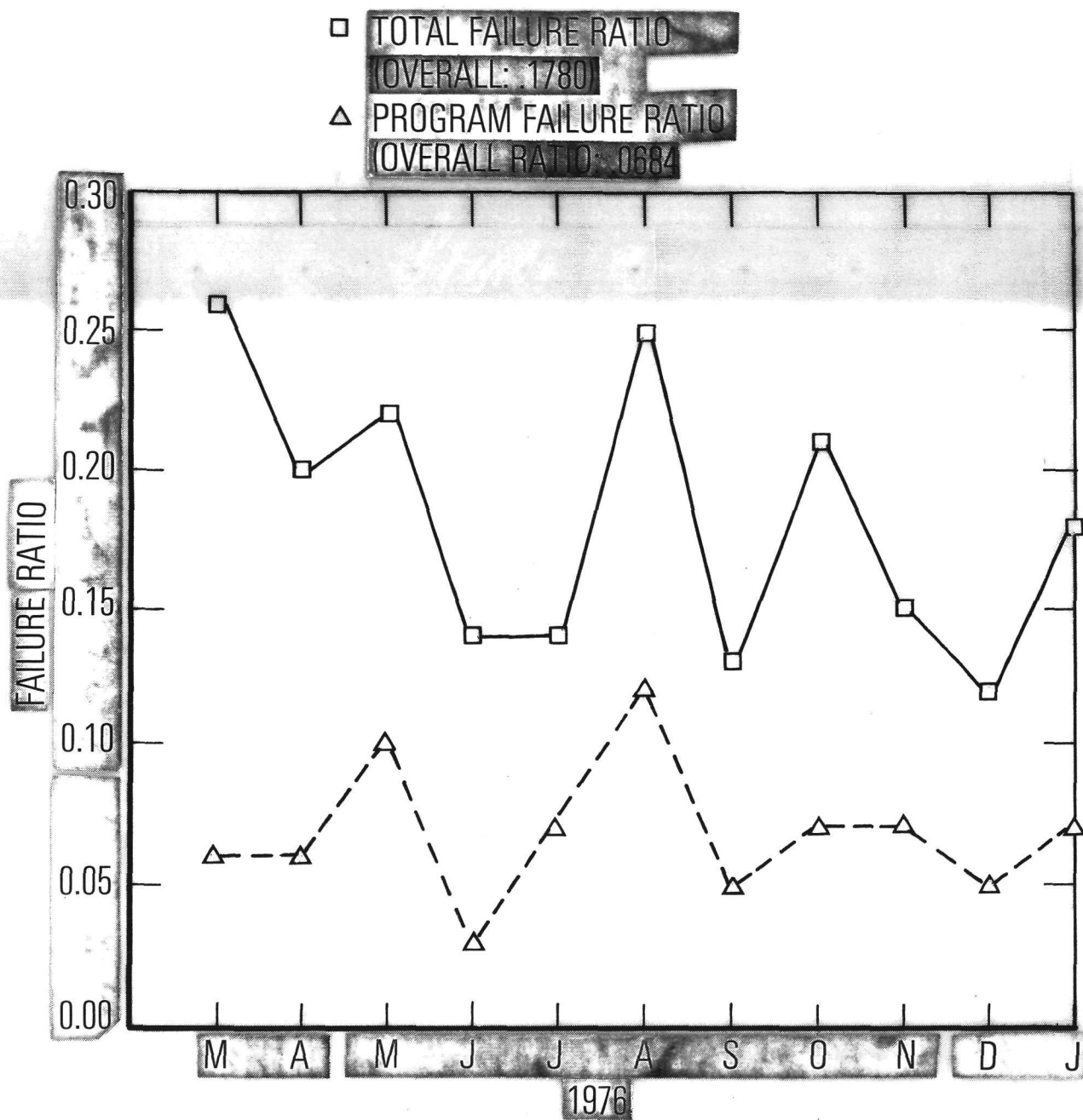


Figure 1. Total and Program Failure Ratios for All

also be noted that there is a large difference between the total failure ratio and the program failure ratio. This is discussed in Section 6; in Section 5 it is shown that the ratio of total-to-program failures is nearly uniform among the modules of LSDB. Finally, it was observed that the peaking of the failure ratios, discernible in August and October 1976 and January 1977, coincides with the time that major design reviews of the programs were taking place; it is plausible to believe that these significant deviations in the central tendency of the failure data are in some way correlated with these program events.

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 this connection, a comparison of the total failure experience on LSDB when reported by failure ratio and by failures per month is presented in Figure 2. The more stable measure of reliability furnished by the failure ratio is quite obvious. It should also be noted that this plot represents the entire program activity on LSDB which, because it was constrained by manpower and computer

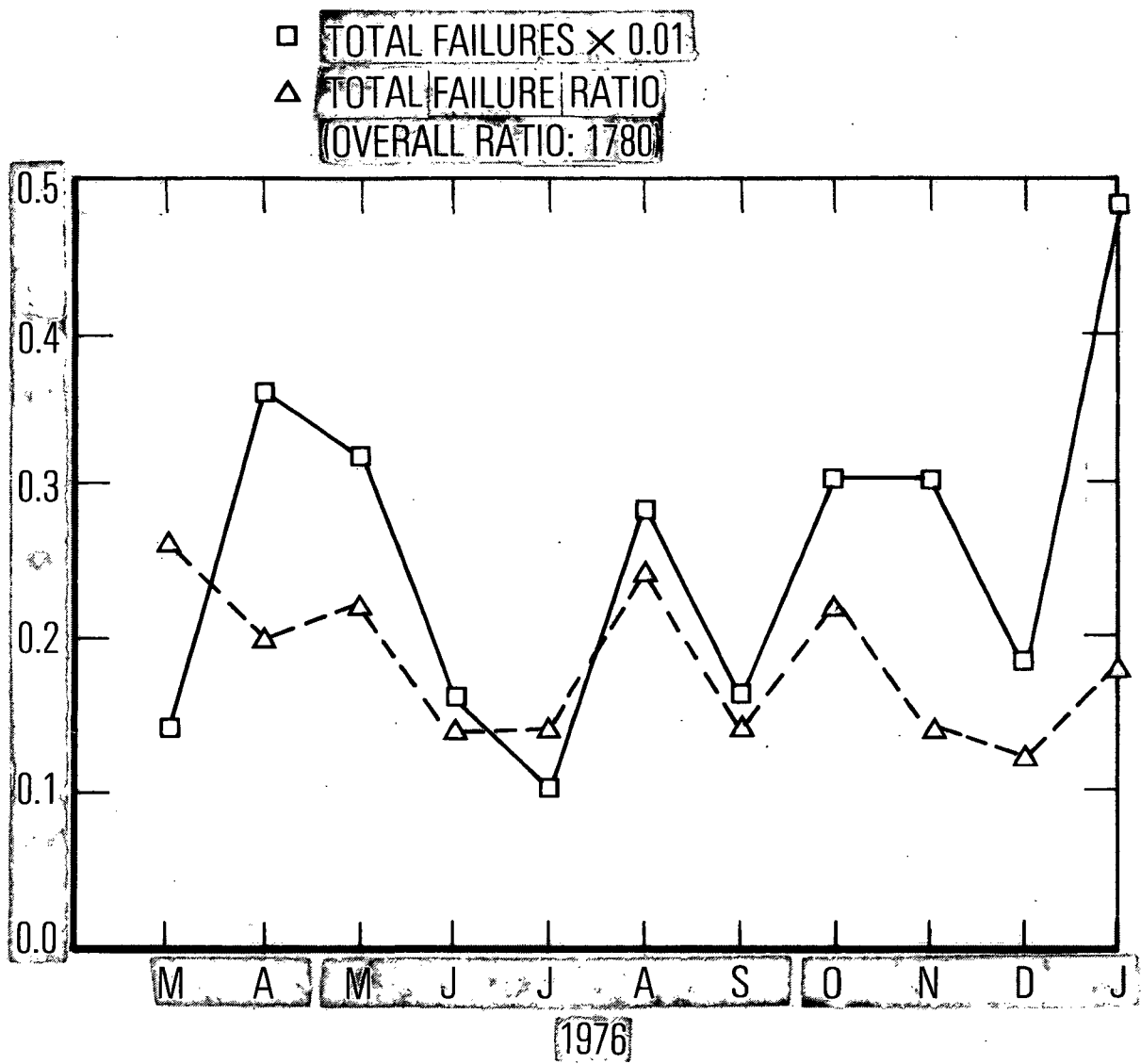


Figure 2. Total Failures and Total Failure Ratios for All

9069

time availability tended to yield a more uniform number of runs per month than the activity on an individual module. Because failure ratio yields a more stable index, its use seems well justified for use in future software reliability measurements.

In total this study has established that a meaningful quantitative measure of software reliability can be generated, and data collection and data analysis methods to support this measurement are available. The consistency of the failure ratio data from month to month shows that quantity to be useful for measurement and estimation, at least within the development phase. The consistency observed here also makes it desirable to use the failure ratio as a possible predictor of software reliability in later stages of the life cycle as well. Specific applications of the results of the study are discussed in Section 7.

**Page
Intentionally
Left Blank**

III. THE COMPUTER PROGRAM AND ITS ENVIRONMENT

The host project for the Software Reliability Measurement Study is the Metric Integrated Processing System (MIPS) that is being developed for the U. S. Air Force Space and Missile Test and Evaluation Center (SAMTEC) at Vandenberg Air Force Base, California. MIPS provides the primary metric (i.e., positional) data processing for test or trajectory measurement activities on missiles, aircraft, and satellites. The constituents of MIPS are the control segment, the real-time segment, and the non-real-time segment. The system specification² requires a modular program structure, hierarchical program design, and execution order programming. In addition to these overall requirements it was decided to demonstrate the value of a highly disciplined programming environment on portions of the non-real-time segment including the program from which the data for this study were obtained. The additional requirements imposed on the software in this study included the following:

- a. Top-down development
- b. Structured code
- c. Program support library
- d. Chief programmer teams
- e. Structured walk-throughs.

The effort of implementing and evaluating these techniques is termed ASTROS (Advanced Systematic Techniques for Reliable Operational Software). Further background on ASTROS is presented

in a technical paper by J. A. Salazar and R. R. Hall. Because its initial distribution was quite limited it is reproduced here as Appendix B. The data accumulated for the evaluation phase of ASTROS provided a unique opportunity to conduct software reliability measurement during program development.

The measurements reported here were taken on the Launch Support Data Base (LSDB), a major component of the non-real-time segment of MIPS. The LSDB Program is broken down into five major components (which will be referred to as "modules") and approximately 40 independent subroutines (which will be referred to as "utilities"). LSDB includes data base management functions and scientific calculations, e.g, to translate local line-of-sight data into a common earth-centered coordinate system. The entire LSDB Program comprises approximately 25,000 lines of source code, of which the modules account for about 18,000. 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 for use on an IBM 360/65 computer. Small segments were written in the IBM assembly language (BAL). Originally, five programmers were assigned to LSDB but after a few months this was reduced to a staff of three plus a programmer-librarian. The design was started in September 1975.

The coding period for which measurements are presented here comprised 11 months from March 1976 through January 1977.

It is intended to develop another non-real-time component of MIPS in a non-structured programming environment during 1977. This should permit comparisons of reliability measurements and related development data between the two software components.

It is at present not known what factors in the programming and computer system environment affect software reliability. For this reason it was decided that a listing of requirements levied against the software product (here LSDB) as well as a description of the general environment should be a part of the record of this Software Reliability Measurement Study. Forms for reporting this background information had been developed earlier³. The forms utilized for reporting background data on LSDB are listed in Table 1; samples of the forms are reproduced in Appendix A. The primary use of this information is future comparative evaluation of the reliability measurements on LSDB with those from other sources. It is hoped that quantitative information about the effects of programming, test, and management techniques can be gained from such comparisons.

TABLE 1

BACKGROUND DATA REPORTING FORMS

General Project Summary
Management Methodology Summary
Design and Processor Summary
Personnel Profiles
Testing Summary (to be filled out during project test)
HIPO Charts (Hierarchy plus Input, Processing, Output)

IV. MEASUREMENTS AND TIME TRENDS

Data for this study was provided by the programming team by means of reports and data tapes.

Two classes of reports were utilized: those providing background data were mentioned in the preceding section; the other class of reports provide data on each run. The run report forms are illustrated in Figures 3 and 4. The run analysis form (Figure 3) was prepared for every run, while the failure analysis form was prepared only for those runs that were not successful. The run reports were prepared daily by the program librarian. The determination that a run had failed was made by the development group. The data obtained from the run reports were entered into a computer data base, and statistical summaries and plots were generated by a series of APL programs. The principal data tapes utilized were those from the System Management Function (SMF) of the IBM computer operating system. Since the SMF tapes duplicated most of the information on the run analysis form, the tapes were used primarily as a check on the completeness of the data collection effort(1). It was found that data collection was carried out very conscientiously.

(1) For future applications of software reliability measurement it may be possible to eliminate the run analysis form and derive the data from a system tape such as SMF.

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 ☐
 - b. Partial Integ. Test ☐
 - c. Full Integ. Test ☐
 - d. Production Program ☐
 - e. Other _____ ☐
7. Program Activity
 - a. Compilation ☐
 - b. Compile and run ☐
 - c. Run with no compile ☐
 - d. Other _____ ☐
8. Number of Source Statements Changed/Deleted Inserted
 - a. None ☐
 - b. 1-10 ☐
 - c. 11-20 ☐
 - d. 21-30 ☐
 - e. 31-40 ☐
 - f. 41-50 ☐
 - g. 51-75 ☐
 - h. 76-100 ☐
 - i. 101-150 ☐
 - j. 151-200 ☐
 - k. Over 200 ☐

Contact _____

Figure 3a

COMPUTER PROGRAM RUN ANALYSIS REPORT INSTRUCTIONS

To be filled out by programming librarian or responsible programmer after each computer run. If the run was unsuccessful (SYNTAX errors, abort, calculation error, loop, etc.), the supplemental form COMPUTER PROGRAM FAILURE ANALYSIS REPORT 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 COMPUTER PROGRAM FAILURE ANALYSIS REPORT.
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.

Figure 3b

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 _____

Figure 4a

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 too 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 ERRORS 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).

The principal measurements derived from the data are the failure ratio and the failure rate. The failure ratio is defined as

$$U = F/N \quad (1)$$

where F is the number of failures observed in N runs. The failure rate is defined as

$$u = F/t \quad (2)$$

where F is the number of failures observed during t seconds of CPU time. These failure metrics, and particularly their complement, the reliability metrics,

$$R = 1 - U = S/N \quad (3)$$

where S stands for the number of successes

$$\text{and} \quad \text{MTBF} = t/F \quad (4)$$

are analogous to commonly used hardware reliability expressions. The relation of these metrics to those used by other researchers in software reliability is described in Ref. 4.

During the development phase the dominant computer usage is frequently for compilation and related activities. Under these circumstances, computer time used during a run is approximately

proportional to the length of the code submitted(2). This relationship has significant effects on the failure rate data presented and will be commented on further below.

Failure of a computer run can be due to three major causes: (1) hardware faults or errors by the computer operator, e.g., in mounting the wrong tapes; (2) faulty interactions, specific to that run, with the data base or with the operating system (e.g., caused by control cards); and (3) errors in the code itself. The first category has been termed "Operation Error", and, although run analysis and failure analysis forms were completed for these cases, they were not considered in the failure ratio or failure rate measures (i.e., they affected neither numerator nor denominator of these fractions). Failures due to (2) and (3) are reported as "Total Failures", while the term "Program Failures" pertains exclusively to those in the third category.

A typical graphical representation of these quantities has already been presented in the overview (Figure 1). The central tendency of total and program failure ratio has been commented on there, and the constancy of these ratios over a period of time has been interpreted as making them useful for reliability measurement.

(2) During test and operation the CPU time is dominated by the execute step. This is less directly bound to length of code because of time spent in loops, I/O, etc.

Another way of evaluating the plots of software failure ratios is to compare them to equivalent plots for hardware. For this purpose, reliability data on a guidance set were obtained, and the failure ratio was defined as the number of sets failing in a given month divided by the number of sets in the field (approximately constant over the period). A comparison of such a hardware failure ratio with the total software failure ratio for LSDB is shown in Figure 5. A significant conclusion from this figure is that the deviations about the average value are about the same for both plots. The hardware failure data was reported in a logistic planning document and had been used for predicting the quantity of spares and number of repair crews required. This suggests that the month-to-month variability of the software data should not present a major obstacle to their serving for useful reliability forecasting. Also, the similarity of the two failure ratio plots permits one to speculate in a general way about combining hardware and software failure ratios. (The two specific plots shown here pertain to completely different equipments and environments; it is not appropriate to combine them.)

In addition to the similarities, the two plots in Figure 5 also exhibit a significant difference: the software failure ratio shows a decreasing trend with time over the interval shown whereas the long-term trend for the hardware failure ratio is a

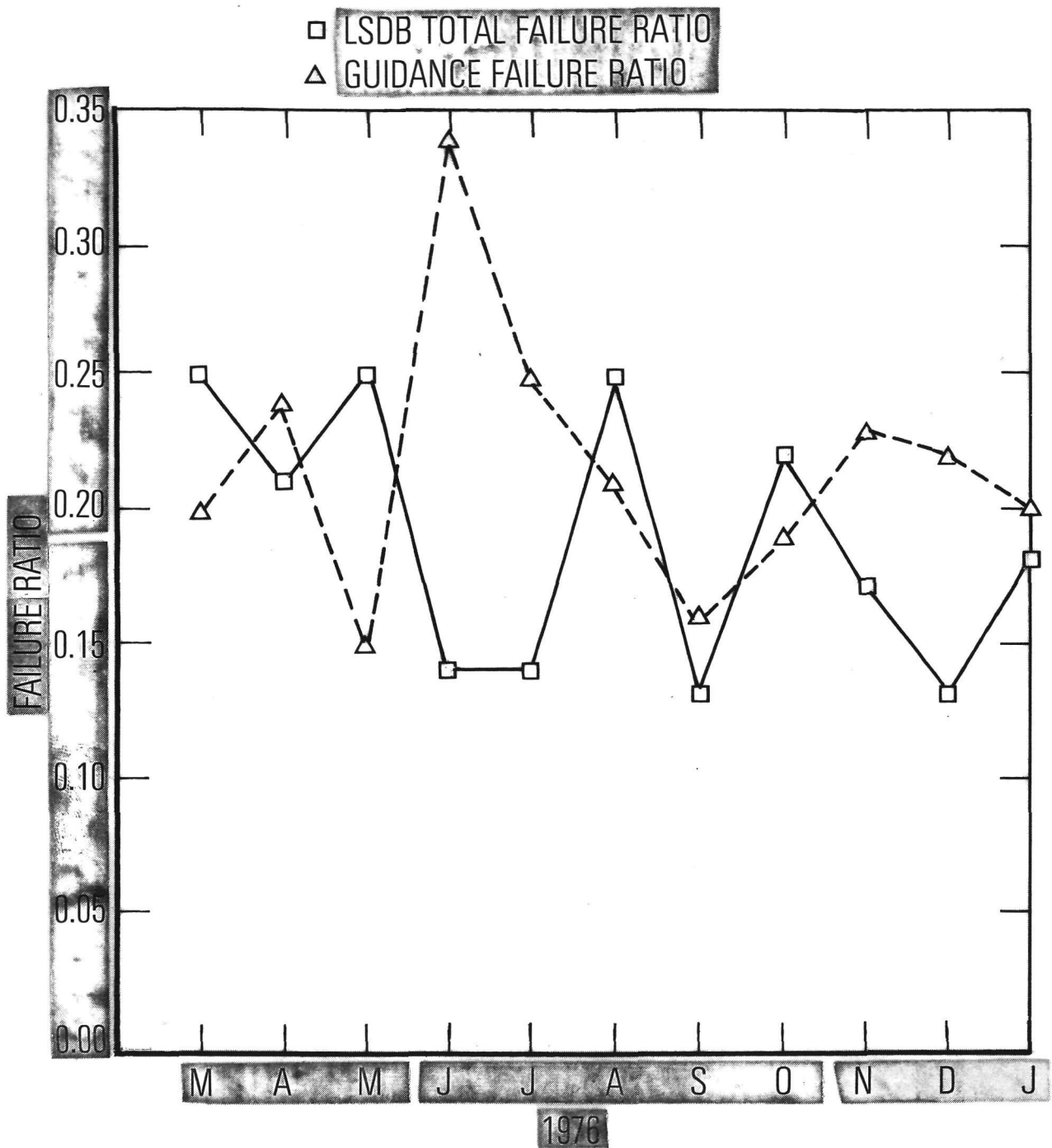
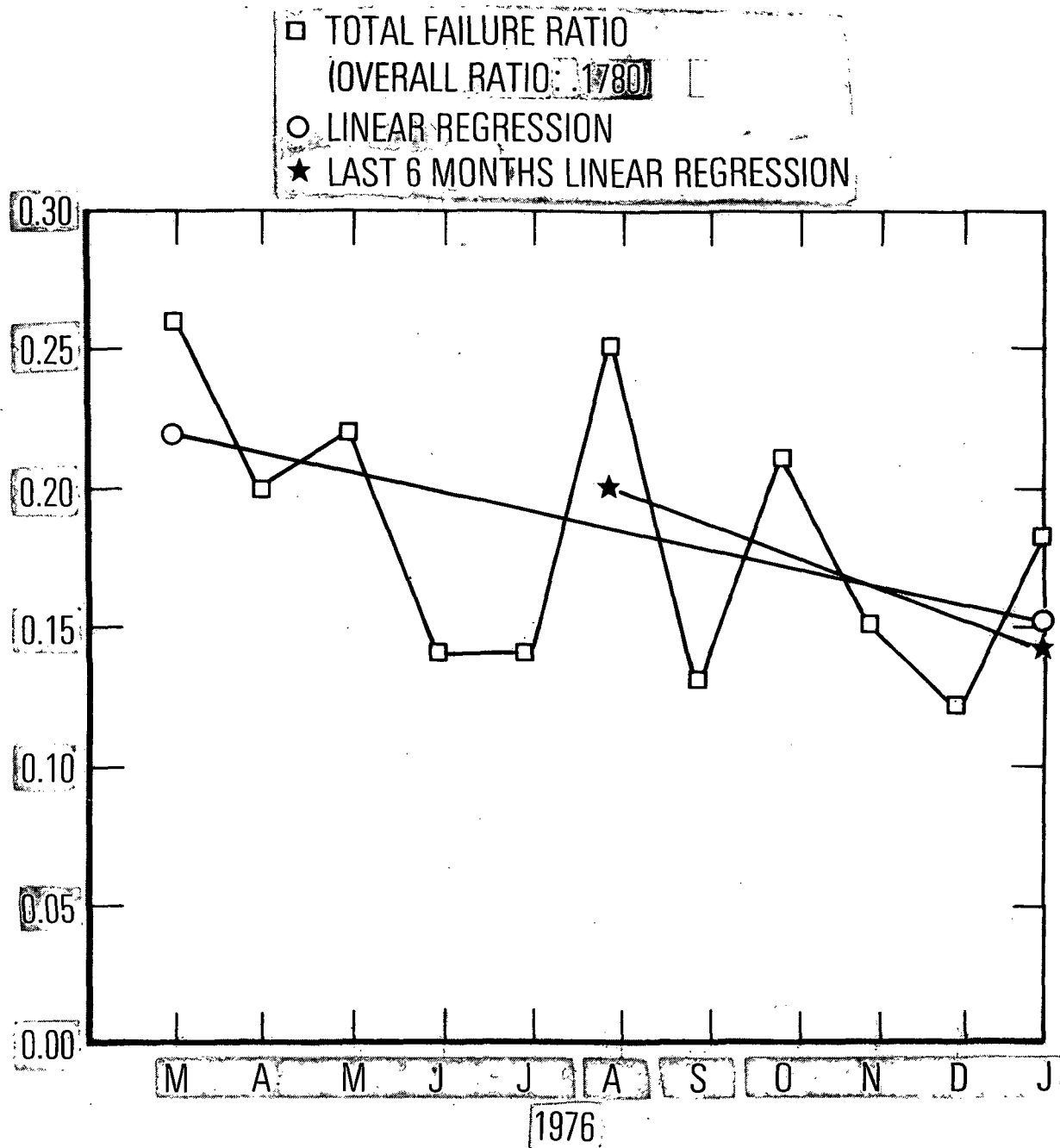


Figure 5. Comparison of Hardware and Software Failure Ratios

constant independent of time. A decrease in the failure ratio is of course what one expects to see as the software reliability is improved during development.

Formal reporting of the trend with time is a potential tool for demonstrating progress to management and contracting agencies. For this reason, plots with trend lines (obtained by linear regression) were developed as shown in Figure 6. Trends for both the entire interval and just for the last six months seemed desirable as measures of progress. To avoid overly complex presentations, it was therefore decided to separate the plots for total and program failure ratios. Note that the trend lines in Figure 6, particularly the one for the last six months' period, indeed show a desirable decrease in the failure ratio.

Below the plot is printed the 90-percent confidence interval for the slope of the regression lines. This shows that there is only five percent probability that the overall trend is less negative than -0.0069 ($-0.0073 + 0.0004$), or that the six-month trend is less negative than -0.0110 ($-0.0123 + 0.0013$). The use of the t-distribution permits calculation of other confidence intervals. E.g., there is only 0.5 percent probability that the six-month slope is less negative than -0.010 . Either the point estimate of the slope or an upper confidence limit may serve as a useful management tool for control of reliability during software development. Experience with several other development projects



○ CONF INT: $-.0073 \pm .0004$
 ★ CONF INT: $-.0123 \pm .0013$

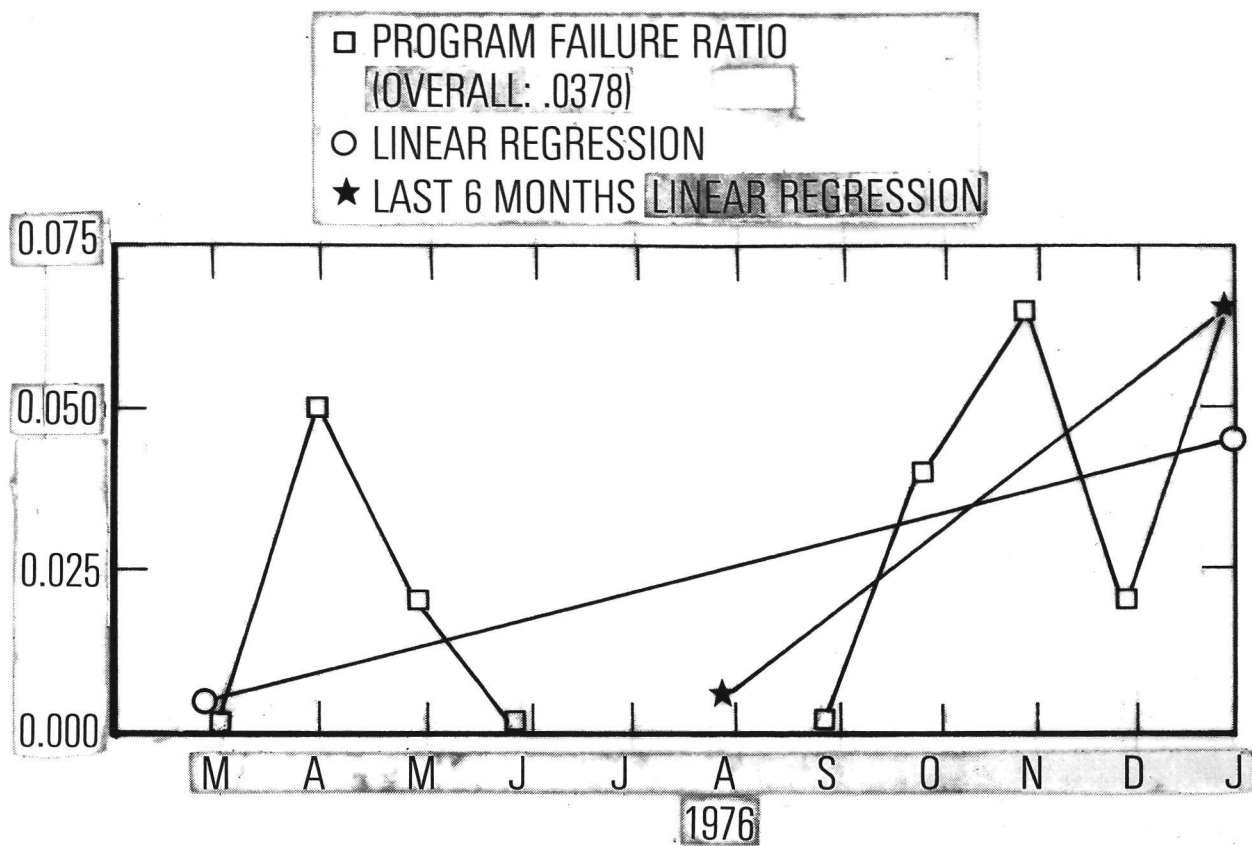
Figure 6. Total Failure Ratio for All

is required to establish suitable time periods for the regression and the statistical limits that should be employed.

In evaluating the trend lines it must be kept in mind that this is a top-down program development, in which modules are constantly being fleshed out as they are being tested so that the actual amount of code increases for a considerable time while runs are being made. It is therefore not surprising to see a more sharply decreasing trend for the last six months' period when the overall size of the code has been reasonably stable.

Figures 1 and 6 have demonstrated the central tendency of failure ratio measurements, deviations about the expected value due to random events, and more pronounced deviations when the project life cycle exposes the code to a higher stress level. It has also been shown that, despite these deviations, a trend in failure ratio can be clearly depicted by linear regression lines.

Failure ratio plots can also highlight unusual events that affect individual elements of the computer program as shown in Figure 7. The utilities have already been introduced as a collection of small subroutines within the LSDB Program. The initial three months of coding showed a normal failure ratio history, and after that the utilities seemed to be exceptionally failure free through September. Runs continued to be made on these subroutines during these months as shown in Figure 8.



○ CONF INT: .0041 +/− .0001
 ★ CONF INT: .0120 +/− .0003

Figure 7. Program Failure Ratio for Utilities

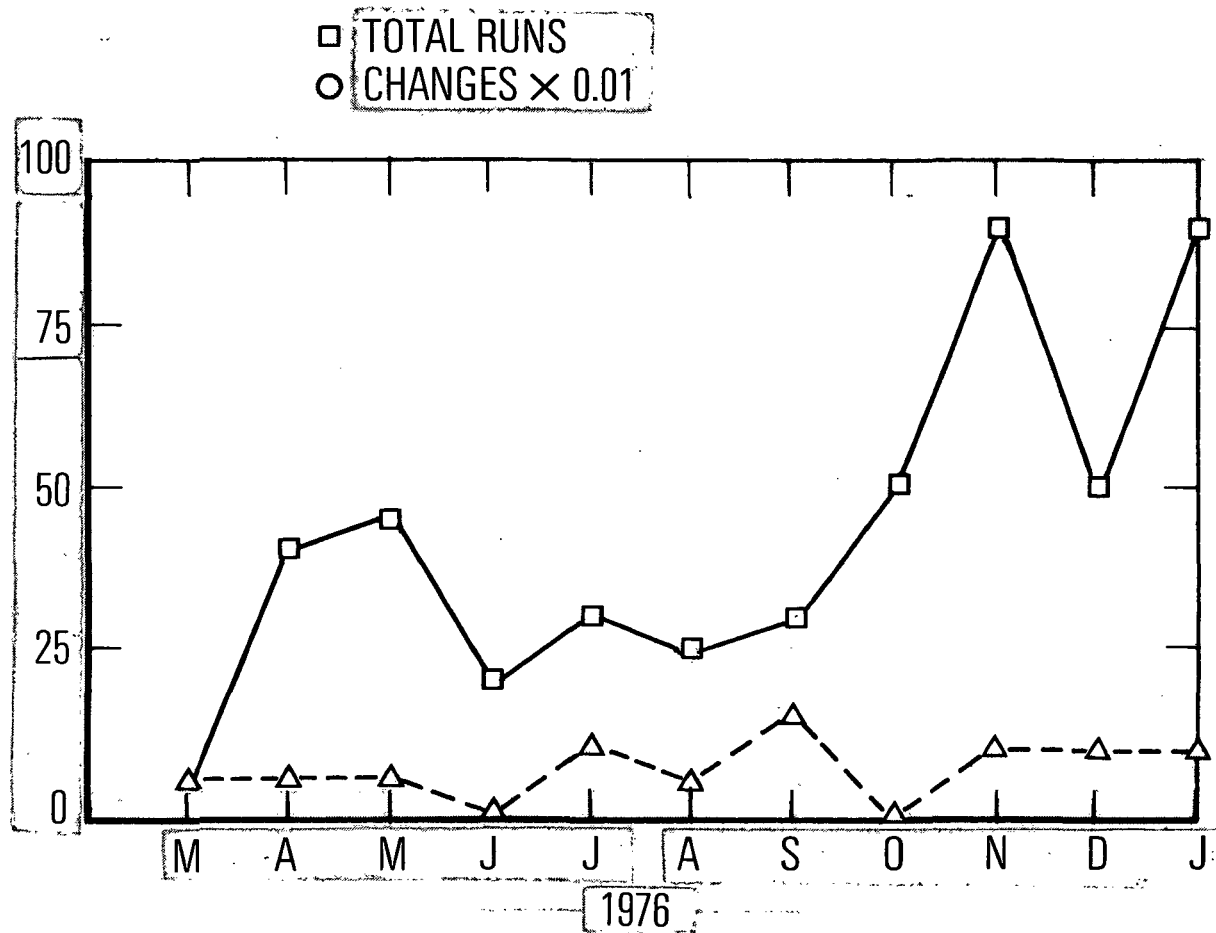


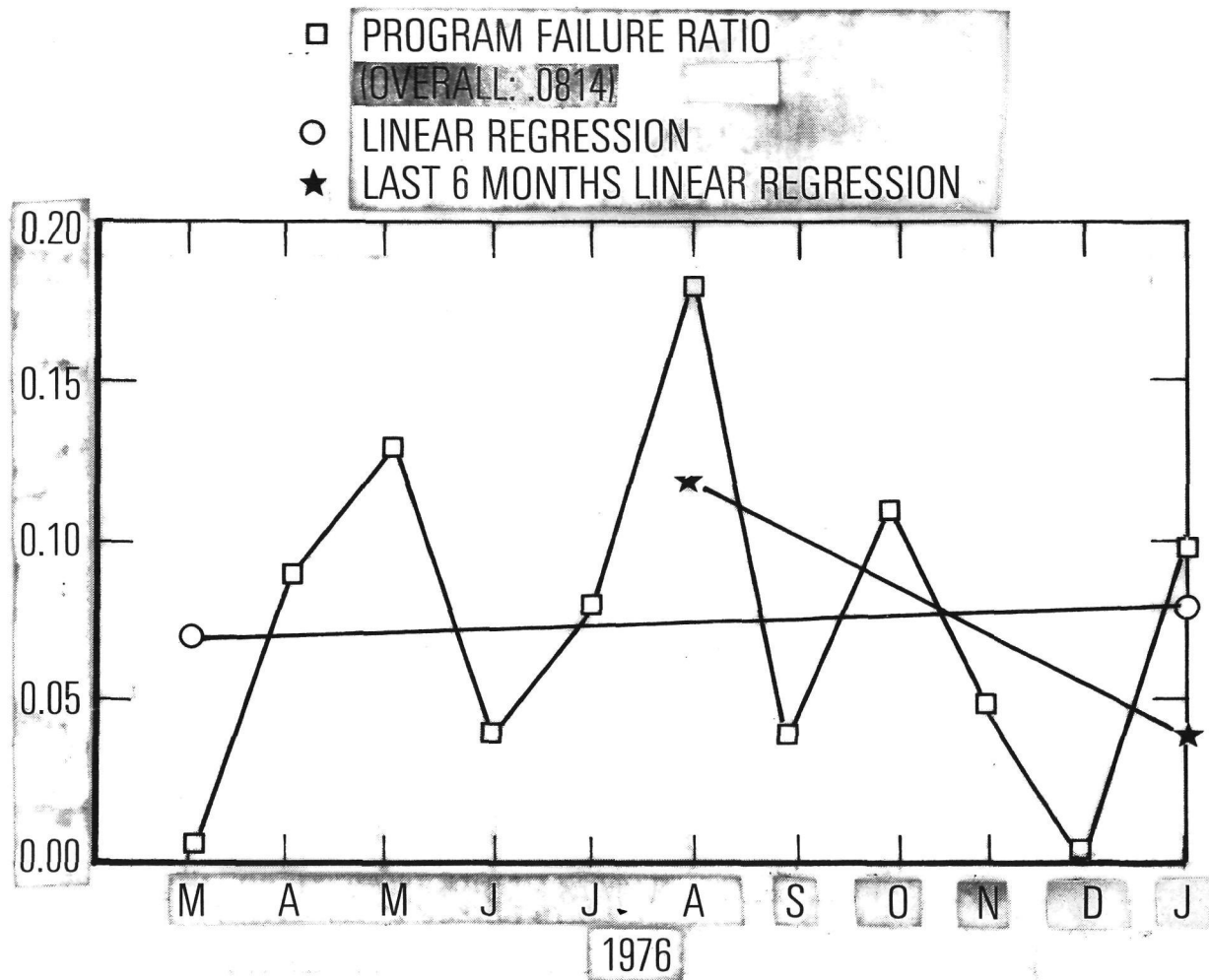
Figure 8. Total Runs and Total Changes for Utilities

Beginning in October 1976, Figure 7 shows a sharp increase in the failure ratio, and fairly high failure ratios have persisted through January 1977. As can be seen in Figure 8, the number of changes increased above the previous level in the summer of 1976, and a high level of program runs and change activity has continued ever since. On inquiry to the developers it was found that a major change in the utilities format was required by the concurrent development of another MIPS segment that uses LSDB output. Also, the addition of new coding during LSDB development put additional stress on the utilities. Events of this nature are, of course, not uncommon in any major software development. The ability of the failure ratio plots to highlight the effects of changes and other occurrences on reliability has considerable potential as a management tool.

That runs made early during the coding period involved fewer instructions than those made after the module was completed(3) can be seen by comparing plots of failure ratio and failure rate for a given module. It will be recalled that failure rate is based on a measure of computer time that is a function of the length of the source code. Thus, the overall failure ratio trend

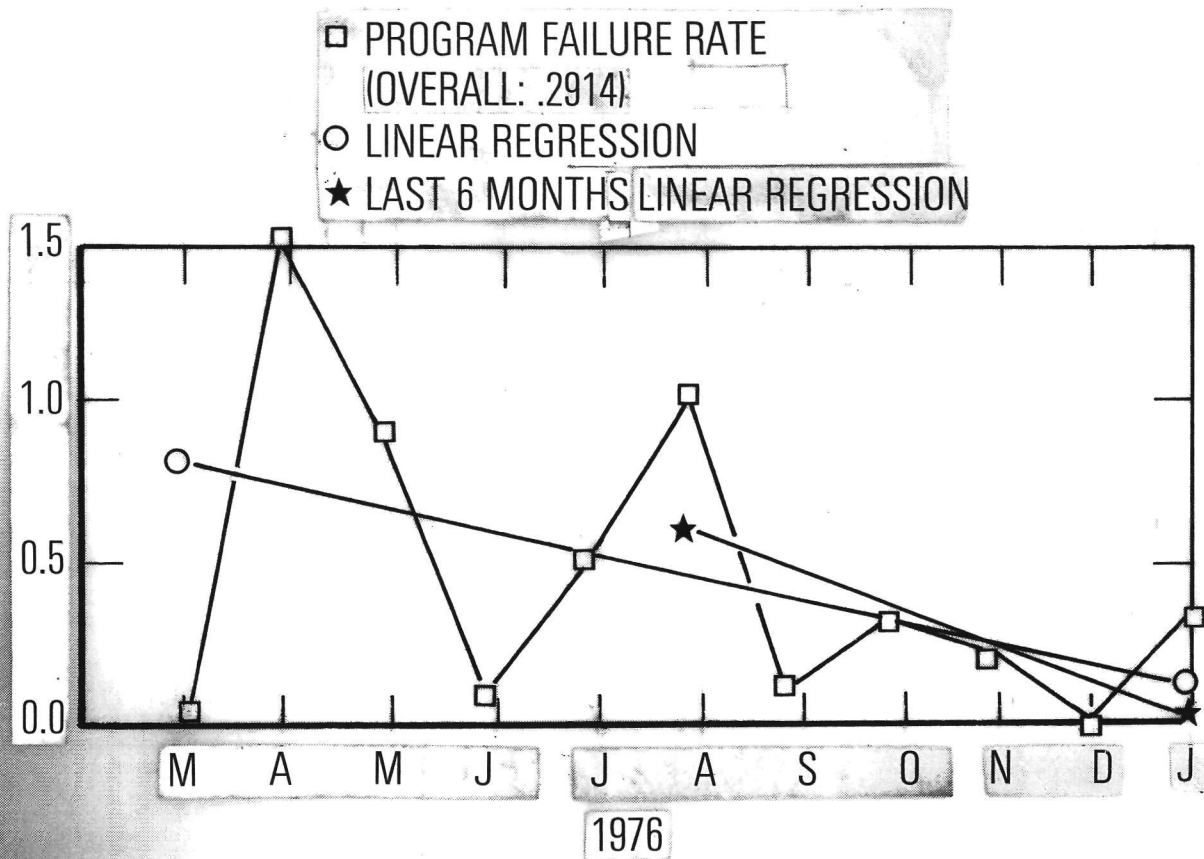
(3) Each module consists of a number of elements (50 to 100 statements long). Initially some of these elements were "dummied" and then replaced by full code as the details were defined.

line shown for the LDG module in Figure 9 would indicate that no improvement in failure ratio has been observed for this module. On the other hand, the failure rate plot in Figure 10 shows a steady trend of improved software reliability throughout this period. For the latest six months, both figures indicate that a sizeable reduction in the failure experience has taken place. Under conditions where the size of the code is expected to change materially during development it is therefore concluded that reliability measurement based on failure rate (where computer time is dominated by compilation) may provide a more representative index of reliability.



○ CONF INT: $-.0001 \pm .0006$
 CONF INT: $.0167 \pm .0019$

Figure 9. Program Failure Ratio for LDG



CONF INT: $-.0627 \pm .0380$
 CONF INT: $-.1065 \pm .0480$

Figure 10. Program Failure Rate for LDG

V. VARIATIONS BETWEEN MODULES

Failure ratios and failure rates for the major modules in LSDB are summarized in Table 2. The mean and standard deviation for the module values of these quantities are shown at the bottom of the table. The current number of source statements and number of runs accomplished to date for each module are also shown.

The first observation is that within each failure ratio category there is a remarkable consistency. The highest failure ratio in no case exceeds twice the value of the lowest failure ratio for that column. (In hardware reliability tests for different lots of a given device type it is not uncommon to see two-to-one variations.)

At first glance it would appear that the program failure ratio, yielding a smaller standard deviation, furnishes the most cohesive measure. It must, however, be considered that the mean for the program ratio is considerably lower than the mean for the total failure ratio. Under these conditions a more meaningful index of variability is furnished by the ratio of standard deviation to the mean, which is also called the coefficient of variability. This is listed in the last row of the table. By this criterion total failure ratio furnishes a slightly more stable index. The ratio of program to total failures is also fairly constant between modules except for the last module, BDT,

TABLE 2

MAJOR MODULE RELIABILITY SUMMARY

<u>Module</u>	<u>No. of Source Statemts</u>	<u>No. of Runs</u>	<u>Failure Ratio</u>		<u>Ratio P/T</u>	<u>Failure Rate</u>		<u>Ratio P/T</u>	<u>Total Fail Ratio per 1000 Statemts</u>
			<u>Total</u>	<u>Progr</u>		<u>Total</u>	<u>Progr</u>		
LSO	6294	239	0.265	0.082	0.31	0.66	0.21	0.32	0.042
LDG	4837	283	0.174	0.081	0.46	0.62	0.29	0.47	0.036
LDI	3395	380	0.185	0.074	0.40	0.75	0.30	0.40	0.054
BDP	1979	127	0.178	0.076	0.43	0.77	0.33	0.43	0.090
BDT	1521	108	0.190	0.120	0.63	1.00	0.63	0.63	0.125
Total	18026	1137							
Mean	Raw		0.20	0.087		0.76	0.35		0.069
Std Dev Between Modules			0.038	0.019		0.15	0.16		0.037
Coefficient of Variability			0.19	0.22		0.20	0.45		0.54

for which program failures constitute a significantly higher percentage. That non-program failures constitute the major fraction of failures during development is an important finding of this study that is commented on in Section 6.

The variability for total failure rate is about the same as that for total failure ratio, as measured by the coefficient of variability and also as can be determined from the total range of the data. For program failure rate a considerably greater variability exists (e.g., range is three to one), this being largely attributable to the high program failure rate for BDT. The ratio of program-to-total-failure rate is almost identical to that for the failure ratios. This is at first surprising, since many errors that cause failures only in the program category lead to abort early in execution (e.g., data reference errors). It must be remembered, however, that the predominant use of CPU time is for compilation and that this step can be completed without dependence on data-related control statements.

That failure rate normalizes for program size (compared to failure ratio) can be seen particularly by the relative ranking of the LSO module, the largest one in this group. It has the highest total failure ratio and the second highest program failure ratio, but ranks second lowest in total failure rate and lowest in program failure rate. Also, the relative ranking of BDP and LDI is reversed between failure ratios and failure rates,

as would be expected due to the larger number of statements in LDI.

A direct normalization with respect to number of statements for the total failure ratio is shown in the last column. The variability in this column is greater than for the non-normalized cases, as can be seen by both the coefficient of variability and the total range (here greater than three to one). Also, one is led to the surprising conclusion that normalized failure ratio decreases with program size. A number of explanations can be offered for this: the exposure to non-code-related errors is the same for each module regardless of size; the division of the program into modules is governed by concepts of equal complexity of computation and not by estimates of lines of code; or, finally, that small modules receive less attention. None of these hypotheses completely explains the observations.

The somewhat anomalous data for the BDT module in this comparison led to inquiries of the developers for a possible explanation. Lack of attention to this module or assignment of it to inexperienced personnel were clearly ruled out by their findings. It was, however, noted that this module contained the most difficult algorithms, including many coordinate conversions, and that accounted in their opinion for the observed characteristics.

The listing in Table 2 shows that the larger modules had been subjected to more runs than the small ones, and it was suspected that this might introduce a bias in lowering the failure ratio and failure rate of the large modules. For this reason, the summary of results for exactly 100 runs on each module was prepared as shown in Table 3. This shows only very modest changes in the failure ratio and slightly larger ones in the failure rate from the overall data in Table 2. The total failure ratio is seen to be only slightly higher for the large modules than for the small ones, while program failure ratio seems to be almost independent of module size. Failure rate, on either total or program basis, also shows no clear trend that would indicate that module size has a major effect. Further internal analysis on LSDB (e.g., in quantifying module complexity) and studies of the effect of module size from other development environments should be undertaken.

TABLE 3
DATA FOR THE FIRST 100 RUNS ON EACH MODULE

<u>Module</u>	Failure Ratio		Failure Rate	
	<u>Total</u>	<u>Program</u>	<u>Total</u>	<u>Program</u>
LSO	0.28	0.08	1.03	0.29
LDG	0.28	0.08	1.39	0.40
LDI	0.18	0.08	0.76	0.34
BDP	0.20	0.09	0.88	0.39
BDT	0.19	0.12	1.00	0.63

VI. CAUSES OF FAILURES

A gross classification of causes of failures was made when separate plots for total and program failures were presented (e.g., in Figure 1). The primary reason for this segregation is that failures due to the program itself would be expected to be carried into the operational environment, while failures due to improper setup of a data deck or improper use of job control language affect only the immediate execution of the run then in progress. This does not imply that the reliability of a software product should be characterized solely by the program failure ratio or program failure rate. Well-designed programs are easy to set up (and thus cause few errors in job control) and are robust with regard to at least some of the commonly encountered difficulties in interacting with the data base. However, this major division was fairly easy to make, and it shed light on some of the processes that cause software failures. It is quite obvious from Tables 2 and 3 that failures other than those in the program itself caused most of the difficulties during this development phase. This is in itself a significant conclusion but one that does not come as a surprise to anyone familiar with the development of scientific programs against a complex data base. The programmer is trained to devote his efforts to the control structure, algorithms, and logic of the problem, and

details of data setup and job control receive comparatively less attention. There is also, of course, the feeling that errors in these latter categories can easily be remedied by resubmitting the run. If reduction of failure frequency during the development phase is to be accomplished, considerable educational effort in this area is obviously required.

A more detailed breakdown by error categories obtained from the failure analysis reports is shown in Tables 4, 5, and 6(4). As originally entered on the failure analysis sheets, the "Other" category was the largest one. More detailed examination showed many of these errors to be due to either keypunch or Job Control Language (JCL). When these were established as separate cases (on the basis of explanations entered at the time the sheets were filled out) a more uniform distribution of error classifications emerged. As shown in Table 4, on an overall basis (for the entire time period) logic errors were the single most significant classification. This was followed by a fairly close grouping of JCL, residual other, and keypunch errors. Data input and program execution errors ranked further down, and all other categories contributed only very minor amounts. Due to the early stage in

(4) Program errors comprise class codes A, B, F, G, and J in these tables. The descriptions of the class codes on the tables are necessarily brief. A complete explanation is provided on Figure 4b.

TABLE 4

FAILURE RATIOS BY ERROR CLASS FOR ALL

	OVERALL	MAR	APR	MAY	JUN	1976				NOV	DEC	1977 JAN	
						JUL	AUG	SEP	OCT				
A:	.003	.	.	.014	.009	.	.	.017	.	.	.062	.046	.053
B:	.051	.	.051	.041	.017	.071	.115	.035	.049
C:	.011	.053	.017	.075048	.	.
D:	.008	.	.	.007
E:	.001	.	.006
F:
G:	.001	.	.006
H:	.003	.	.011	.	.009004
I:	.012	.053	.011	.041	.009	.	.009	.	.021011
J:
K:007	.034	.	.009	.052	.014	.	.031	.020	.068
L:	.030	.053	.017	.055	.017	.	.062	.009	.007	.	.009	.020	.019
M:	.022	.035	.017	.007	.043	.057	.035	.017	.132	.	.009	.040	.019
N:	.039	.035	.063	.007

ERROR CLASS CODES

- A. COMPUTATIONAL
- B. LOGIC
- C. DATA INPUT
- D. DATA HANDLING
- E. DATA OUTPUT
- F. INTERFACE
- G. ARRAY PROCESSING
- H. DATA BASE
- I. OPERATIONS
- J. PROGRAM EXECUTION
- K. DOCUMENTATION
- L. OTHER
- M. KEYPUNCH
- N. JCL

TABLE 5

FAILURE RATIOS BY ERROR CLASS FOR MODULES

	OVERALL	1976												1977	
		MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	JAN			
A:	.005	.	.	.020	.011	.	.	.024047		
E B:	.059	.	.053	.051	.021	.119	.148	.047	.057	.067	.059	.	.047		
R C:	.014	.060	.008	.111		
R D:		
O E:	.001	.	.008		
R F:		
G:	.001	.	.008		
H:	.004	.	.015	.	.011006		
L J:	.018	.060	.015	.061	.011	.	.011	.	.034018		
A K:		
S L:	.034	.060	.015	.	.042	.	.	.059	.023	.042	.010	.	.082		
S M:	.025	.040	.015	.051	.011	.	.080	.012	.011	.008	.030	.	.023		
N:	.036	.	.075	.010	.053	.048	.023	.	.092	.017	.050	.	.018		

ERROR CLASS CODES

- A. COMPUTATIONAL
- B. LOGIC
- C. DATA INPUT
- D. DATA HANDLING
- E. DATA OUTPUT
- F. INTERFACE
- G. ARRAY PROCESSING
- H. DATA BASE
- I. OPERATIONS
- J. PROGRAM EXECUTION
- K. DOCUMENTATION
- L. OTHER
- M. KEYPUNCH
- N. JCL

TABLE 6

FAILURE RATIOS BY ERROR CLASS FOR UTILITIES

	OVERALL	MAR	APR	MAY	JUN	1976				NOV	DEC	1977 JAN
						JUL	AUG	SEP	OCT			
A)	.035	.	.048	.021035	.056	.020	.064
B)	.004	.	.048
C)	.024	.	.	.021102	.	.
D)
E)
F)
G)
H)
I)
J)
K)
L)	.024	.	.024	.021	.	.	.040	.033	.	.019	.040	.043
M)	.014	.	.024	.064	.045009	.	.011
N)	.045	.286	.024	.	.	.071	.080	.067	.193	.	.020	.021

ERROR CLASS CODES

A. COMPUTATIONAL

B. LOGIC

C. DATA INPUT

D. DATA HANDLING

E. DATA OUTPUT

F. INTERFACE

G. ARRAY PROCESSING

H. DATA BASE

I. OPERATIONS

J. PROGRAM EXECUTION

K. DOCUMENTATION

L. OTHER

M. KEYPUNCH

N. JCL

the life cycle when these data were taken, interface errors and documentation errors were not encountered at all. Distribution of the errors over the 11-month interval showed only minor trends, most noticeably in that input errors ceased to be encountered after the first three months. Among the major error types, however, the relative standing remained almost unchanged throughout the development period. Comparison of Table 5 and 6 also shows only minor differences between the modules and the utilities. JCL errors were encountered in greater frequency among the latter, which is not surprising since the utilities interact more closely with the overall control structure of the computer system.

For some error categories, a comparison of relative frequency with errors found in an earlier study⁵ could be made. For this purpose the data input and data output categories were combined into an I/O error classification, interface and execution errors were combined into a single category, and data base and global variable errors (the latter reported only in the comparison study) were also combined. The resulting relative frequency of errors is shown in Table 7(5). The columns headed

(5) The relative frequencies are computed for only those errors for which comparable classifications could be established. Some LSDB and some TRW errors were not considered in computing the percentages.

TRW2, TRW3, and TRW4 relate respectively to Projects 2, 3, and 4 reported in Ref. 5. In all cases, logic errors comprised the highest percentage, although this seems to be exceptionally high for the LSDB population. At the other end of the scale, computational errors were the least significant or second least significant in all cases. The overall conclusion is that the relative frequency of errors encountered during this study is roughly comparable with that found in earlier experience.

TABLE 7
RELATIVE FREQUENCY OF ERRORS (in percent)

<u>Error Category</u>	<u>Source</u>			
	<u>LSDB</u>	<u>TRW2</u>	<u>TRW3</u>	<u>TRW4</u>
Computational	4.15	17.23	11.49	2.37
Logic	66.33	27.23	30.51	47.46
I/O Error	10.68	16.60	23.41	12.2034
Interface/Execution	15.58	20.11	20.22	25.76
Database/Global Vars	3.27	18.83	14.36	12.20

**Page
Intentionally
Left Blank**

VII. FINDINGS AND POTENTIAL APPLICATIONS

Significant findings of this study are that software reliability measurement during the development phase is possible, that the forms utilized here captured significant data, and that they could be conscientiously filled out by the development personnel. Of considerable immediate interest is the stability of the failure ratio and rate quantities, both as a function of time and between modules. This implies promise for use of these quantities in estimation and prediction of software reliability in the operational phase.

The setting up of a computer data base for accumulating the software measurement data and the summary routines generated as part of that effort provided great flexibility in data presentation, and the resulting plots illustrate the significant processes at work. The inclusion of trend lines, based on linear regression, further helped in emphasizing the essential information content of the data base. The use of moving trend lines, covering say the most recent three to six months data, as a management reporting tool seems to have obvious promise.

The fact that almost two-thirds of the failures reported were not due to errors in the coding proper is an important finding of this study. It leads one to believe that more education in the mechanics of run setups, data referencing, and

other interfacing with the computer system would be very desirable in minimizing errors during the development phase, speeding up the development process, and increasing programmer productivity.

As the month-to-month variation in failure ratios and failure rates indicate, software development is not a completely deterministic process. Random factors and not-so-random factors can cause appreciable deviations from the smooth convergence to a zero-failure-rate situation.

These findings suggest that software reliability measurement as described here may produce data for the following applications:

1. For the Line Management the failure ratio or failure rate may identify reliability problems that may reside in program modules or in organizational units or individuals. The trend lines described in Section 4 of this report may be particularly useful for this purpose. The analysis of error types may identify the critical program development phase (analysis, design, coding, etc.).

2. For the Project Manager, experience with software reliability measurement over a number of projects can guide cost and schedule allocation between various steps of the development process, particularly between analysis and test. Such measurements may also be significant in evaluating whether budget

or schedule constraints will permit the attainment of a given reliability goal for the computing function. Failure rate measurements may be particularly appropriate in hardware/software tradeoffs.

3. For functional management (the director of software development or an equivalent staff function), reliability measurement can be used for the evaluation of development tools and procedures. In test of critical software cost can be staggering, and software reliability measurement is expected to be particularly useful in identifying good test practices and tools so as to reduce these costs. It may even help with that most baffling of all questions: when to stop testing. Also, consistent implementation of software reliability measurement for several projects should yield a reliability growth model that permits early indication whether reliability goals will be met.

4. For regulatory agencies failure rate measurements may be a key element in determining that safety or availability criteria have been met. In mosts cases the criteria will be aimed at the computing function as a whole (or at a higher system) rather than at software. The compatibility of the software failure rate with conventional hardware reliability indices will be particularly useful.

**Page
Intentionally
Left Blank**

VIII. WHERE DO WE GO FROM HERE?

The valuable baseline that has been established on the LSDB Program during the development phase makes it very desirable to continue the process of reliability measurement on the LSDB program into the operational test and user phases. The absolute level of failure ratios and failure rates that were seen during the development can be translated into meaningful quantities only if they are related to similar measures in the actual usage environment. It is also proposed to accumulate similar measurements on another non-real-time program that will be developed under MIPS in a non-structured programming environment. Although no two programs are exactly alike, a comparison would obviously furnish some insight into the value of the structured program techniques. It is further intended by use of a much simplified data collection technique, relying primarily on data accumulated in the operating system, to conduct software reliability measurement on a major operational ground computer system supporting a NASA spacecraft.

All told the data recorded here, together with what is expected to be learned from the scheduled activities in the near future, form a good basis for performing software reliability measurement during the development of critical programs both as a tool for the management process and as a forecasting device for

the operational reliability of the resulting systems. We also hope to have accumulated a data base here that can be used as a starting point for further research.

REFERENCES

1. M. J. Merritt, et al., Characteristics of Software Quality," Report 25201-6001-RU-00, TRW Systems, Redondo Beach, CA (December 1973).
2. 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).
3. J. P. Johnson, Software Reliability Measurement Study, SAMSO-TR-75-279, Aerospace Corporation, El Segundo, CA (8 December 1975).
4. H. Hecht, 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., Maidenhead, Berkshire, England, (1977), Vol. 2, p. 209-244.
5. T. A. Thayer, 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).

APPENDIX A

BACKGROUND DATA FORMS

One unique feature of this Software Reliability Measurement Study is the great amount of information collected which describes the environment under which the Launch Support Data Base (LSDB) software is being developed. This background data is of particular value for future comparisons of the reliability parameters obtained here with those from other sources. The background data is contained in five report forms: the General Project Summary, the Management Methodology Summary, the Design and Processor Summary, the Programmer/Systems Analyst Questionnaire, and the System Development Log.

The General Project Summary (Figure A-1) provides an overview of the LSDB software development efforts. LSDB is a level-of-effort contract providing non-real-time software (part of Prelaunch) for Project MIPS. It is being developed on an IBM 360/65 computer with the following configuration: 768K bytes of memory and 2 million bytes of Large Capacity Storage (LCS).

The function of LSDB is to prepare and update the parameters required to support missile and spacecraft launches at SAMTEC. The total development time for LSDB is about 21 months (64 man-months), with a scheduled completion date of July 1977. Following the July date (completion of Development Test and Engineering) the LSDB software will be subjected to configuration audits (functional and physical) and then integrated with the MIPS Missile Flight Control software through December 1977. On

SYSTEM <u>LSDB</u>		GENERAL PROJECT SUMMARY		DATE <u>1/21/76</u>
•	Type of Contract	<u>Work Request 070 - Level of Effort</u>		
•	Target Computer System(s)	<u>IBM 360/65</u>	<u>768K bytes memory</u>	<u>2 million bytes LCS, tapes, disks, CRTs, card reader, card punch</u>
•	Project Description	<u>Launch support data base preparation (LSDB) is a CPCI under the non-real-time segment (NRT) of Project MIPS. It is being developed using structured programming techniques as part of an operational system. LSDB prepares and updates all parameters needed by the real-time segment (RTS) of Project MIPS and the back-up information display system (BIDS).</u>		
•	Project Start Date	<u>10/75</u>	<u>Estimated Project End Date 7/77</u>	
•	Project Duration	<u>21 months</u>		
•	Total Cost (Actual or Estimated)			
•	Estimate Number of Project Personnel	<u>1 manager, 1 support, 1 design analysis, 5 programming, 1 test</u>		
•	Estimate of Number of Modules	<u>5 + 25 PMs (subroutines)</u>		
•	Estimate Number of Pages of Documentation	<u>100 reqts, 100 specs, 50 test, 50 user manuals</u>		
•	Estimate Total Number of Instructions	<u>4000 FORTRAN + 1000 BAL (assembly language)</u>		
•	Estimate Number of Different Input Formats	<u>19 data base disk files</u>		
•	Estimate Number of Different Output Formats	<u>4 reports (printer) 2 tape</u>		
•	Estimate Total Number of Man/Months	<u>16 mgmt, 16 support, 8 test, 8 design & analysis, 64 programming</u>		
•	Estimate Total Amount of Computer Time	<u>360 hours</u>		
•	Person Filling out Form	<u>LG</u>		

Figure A-1

1 January 1978 the LSDB software is scheduled to become operational.

A total of six full-time personnel were involved in the LSDB development when the report forms were completed, but this was subsequently reduced to four. LSDB consists of five modules (6) (LDG, LDI, LSD, BDP, and BDT) and was originally estimated to require 4000 lines of FORTRAN code and 1000 lines of assembler code. About 360 hours of computer time have been estimated to complete the job. The General Project Summary is supplemented by the Program Schedule (Figure A-2).

The Management Methodology Summary (Figure A-3) describes the rules under which the development must be done and the outputs from the project. The development effort is being done in compliance with MIL-STD-483 (Configuration Management Practices for Systems, Equipment, Munitions, and Computer Programs) and modified by the Implementation Plan for Advanced Programming Techniques on the MIPS Project (TD-75-1392). This modification has been included to provide structured programming techniques (Top-Down Design, Chief Programmer, Librarian, Top-Down Test, HIPO, Structured Code, Structured Walk-Through) to be used extensively through the LSDB development. The Preliminary Design Review (PDR) of the software was scheduled for June 1976, with the Critical Design Review (CDR) being September 1976.

(6) There were originally six modules, but BDT and BDR were combined into BDT.

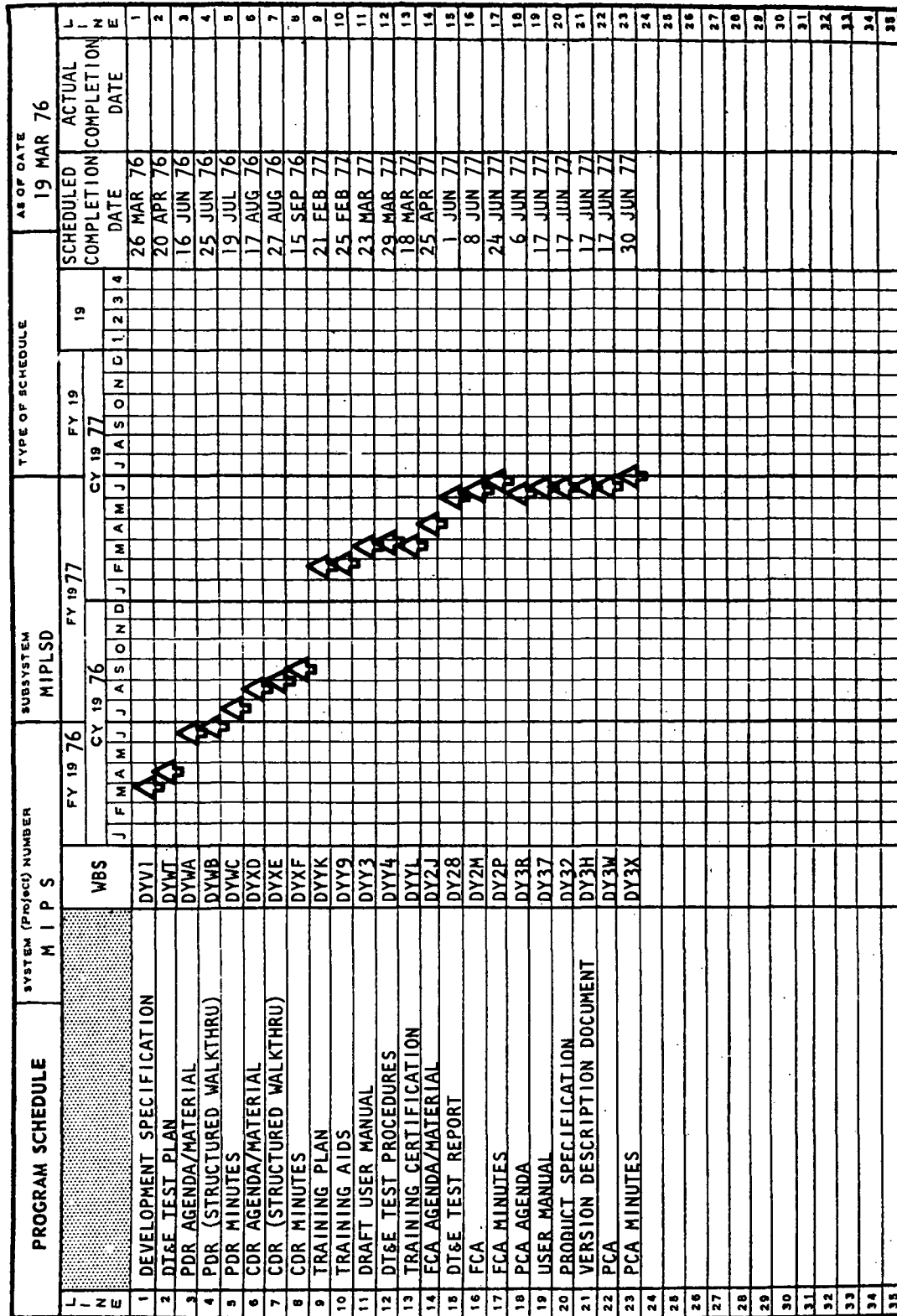


Figure A-2. Program Schedule

SYSTEM LSDB

MANAGEMENT METHODOLOGY SUMMARY

DATE 1/21/76

- Management Procedure/Tools Used TD-75-1392 (Implementation Plan for Advanced Programming Techniques on the MIPS Project)
- List Reports Generated Including all Specifications and/or Requirements _____

(1) <u>Development Spec</u>	Supplied to AF	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No
(2) <u>Product Spec</u>	Supplied to AF	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No
(3) <u>Test Plan</u>	Supplied to AF	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No
(4) <u>Test Procedures</u>	Supplied to AF	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No
(5) <u>Final Report</u>	Supplied to AF	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No
- Review Schedule PDR (structured walkthru) June 1976
(LSDB) CDR (structured walkthru) Sept 1976
- Test Plan Required ☒ Yes ☐ No Used ☒ Yes ☐ No
- Configuration Management Required ☒ Yes ☐ No Used ☒ Yes ☐ No
- AF Regulations, Manuals, and Military Standards Required MIL-STD-483 modified by TD-75-1392
- Software Deliverable LSDB CPCI
- Facilities and Procedures for Data Gathering _____
- Person Filling out Form LG

However, both dates were extended three to six months, and the CDR was divided into five sections (one per module).

The output from LSDB consists of a Development Specification, a Product Specification, a Test Plan, Test Procedures, and a Final Report. All documents are a part of the Contract Data Requirements List (CDRL).

The Design and Processor Summary (Figure A-4) primarily provides information on the IBM 360/65 computer. The software is being developed on the same computer that will be used operationally. Development jobs are submitted in both over-the-counter batch mode with a 24-hour turnaround time and by remote terminal mode with a two-hour turnaround time. The operating system being used is OS/MVT Version 21.7 with the optimized FORTRAN H compiler and the Level F assembler. About 90 percent of the total code has been written in a High Order Language (FORTRAN and S-FORTRAN), with the remainder in Assembler Language.

A separate Personnel Profile exists for each programmer/analyst/librarian associated with LSDB development, and a sample is shown in Figure A-5. A summary of all Personnel Profiles is shown in Figure A-6. The typical programmer had eight years programming and analysis experience, a college degree with two years postgraduate training, has worked on similar projects, and is familiar with a wide variety of programming languages.

The Librarian's function is to submit jobs for the users, maintain statistics on the system, and generally to relieve the programmers of many of the administrative tasks associated with

SYSTEM MIPS/LSDB

DATE 28 Jun 76

DESIGN AND PROCESSOR SUMMARY

1. Target Computer(s) 360/65
Target Computer Same as Development Computer Yes
2. Processing Environment _____

3. Configuration: On Line _____ Batch X Remote Batch X
4. Operating Systems(s) Version 21.7 IBM OS
5. Compiler Version(s) H FORTRAN
6. Assembler(s) 1
7. Est. Percent: HOL 90 % Assembler 10 %
8. Automated Software Tools Used:
S-FORTRAN
Librarian (Attach Vendors Users Manuals)

9. Design Standards _____
10. Programming Standards TD-15 (ASTROS Plan)
11. Programming Techniques Employed:
Top Down Design X HIPO X
Chief Programmer X Structured Code X (mostly)
Librarian X Structured Walk Thru X
Top Down Test X Other _____
12. List Existing Programs/CPC's to be Used MIPLIB subroutine

13. Est. Turn around Time (MRS): Batch 24 Remote Batch 2

Contact LG

Figure A-4

TESTING SUMMARY INSTRUCTIONS

To be filled out by chief programmer or member of independent test group when project is ready for system test. This form identifies testing and requirements documents. It also identifies testing approach, tools, procedures, etc.

1. Reference all requirements and specification documents which were the guidelines for system development.
2. Reference all test plans/procedures the system is being tested against.
3. Briefly describe the overall testing philosophy including debug, computer program test, and integration.
4. Briefly describe or reference procedure documentation for verifying coding standard adherence.
5. List formal audits and tentative dates.
6. Reference management procedures used as guideline for testing.
7. Reference quality assurance documents which describe QA involvement in testing.
8. Estimate person-hours to be expended in testing, include programmers, testers, QA, support and management.

SYSTEM LSDB

DATE 21 Jan 76

PERSONNEL PROFILE

1. Name or ID. JMM
2. Project Assignment (Job Title) Programmer
3. Education Level: HS 4 YRS College 6 YRS
Degree(s) BS - Computer Science
BS - Applied Mathematics
MS - Computer Science
4. Special Computer Training Courses:
 - a. Structure Design Date 12-16 Jan 76
 - b. _____ Date _____
 - c. _____ Date _____
5. Target Language(s) S-FORTRAN, FORTRAN-G, BAL
6. Years of Experience as:

Operator/Technician _____	Analyst _____
Programmer <u>x</u>	Other _____
7. Years of experience on:

Target Computer(s) <u>5</u>	Target Language <u>5</u>
Operating System <u>5</u>	Similar Projects <u>2</u>
8. List Other Programming Languages ALGOL, BASIC, BOBOL, PL/I,
SNOBOL, TYDAC
9. List Other Computers IBM 1620, 360/40, 360/65, 370, 7094,
NOVA 2000

Figure A-5

PERSONNEL PROFILE SUMMARY

Identification	A	B	C	D	E	F	G	H	I	J
LSDB Participation	Full	Full	2/3	Full	1/3	1/4	<1/4	1/4	1/4	1/4
Highest Degree	MS	BS	BA	1 yr college	MS	MBA	BS	BS	MS	BA
Languages	1, 2, 3	1, 2, 3	1	--	1, 2	2	*	1	*	1
Yrs experience as programmer or analyst	2	19	9	Librarian	5	15	1	<1	15	8
Yrs experience on target computer	5	5	*	*	3	1	1	2	*	2
Yrs experience on similar projects	2	15	*	*	0	6	*	*	18	*
No. of other languages	6	7	*	*	5	5	1	5	3	1

*Information not available

Languages

- 1 S-FORTRAN
- 2 FORTRAN
- 3 BAL (IBM Assembly Language)

Figure A-6

program development. In this instance, the librarian has not had a programming or engineering background and has one year of college.

The Testing Summary is shown in Figure A-7. A significant feature of the test philosophy is that at least three distinct groups in the developing organization performed the tests over the development cycle.

An example of a HIPO chart is reproduced in Figure A-8. It shows the function of the major modules within LSDB. The BDR module referred to in that figure was during the coding phase combined with BDP.

SYSTEM MIPS/LSDB

DATE _____

TESTING SUMMARY

1. Requirements and Specification Documents MIPS LSDB Dev Spec
(MIPLSD-1364-3119) MIPS Non-Real-Time Segment Spec (MIPNRT-1264-3117)
2. Test Plans/Procedures MIPS, LSDB Test Plan (MIPLSD-1364-3706, Vol. I)
MIPS LSDB Test Procedures (MIPLSD-1364-3706, Vol. II)
3. Testing Philosophy (1) Structured Programming Team performs detail Testing and debug prior to delivery to test group; (2) test group performs preliminary informal DT&E testing per test procedures to validate CPC and CPCI programs and test procedures; (3) program control formally performs DT&E; (4) fully integrated testing is performed with other CAI.
4. Method Employed to Audit Coding Standard Adherence Implementation Plan for Advanced Programming Techniques on the MIPS Project (TD-75-1392-A)
5. Formal Audits and Dates
 - a. FCA - 8 Jun 77 Date _____
 - b. FPCA - 17 Jun 77 Date _____
 - c. _____ Date _____
6. Internal Management Procedures for Control of Testing MIPS LSDB Test Plan, MIPS LSDB Test Procedures
7. Quality Assurance Procedures Para. 4 to MIPS LSDB Dev Spec

Contact DCC

Figure A-7

TECHNOLOGY CRITIQUE INSTRUCTIONS

To be filled out by all project personnel as they leave the project or when the project is complete. This form will provide a subjective evaluation of the technologies employed by the people who actually used them.

1. Names are optional.
2. Person's assignment description or job title, if meaningful.
3. List time assigned to project in months.
4. Check techniques/tools used on project, briefly describe others.
5. Honest evaluation of your feelings.
6. Be candid, help us define a workable policy.
7. Once again be candid, let's make the training useful.

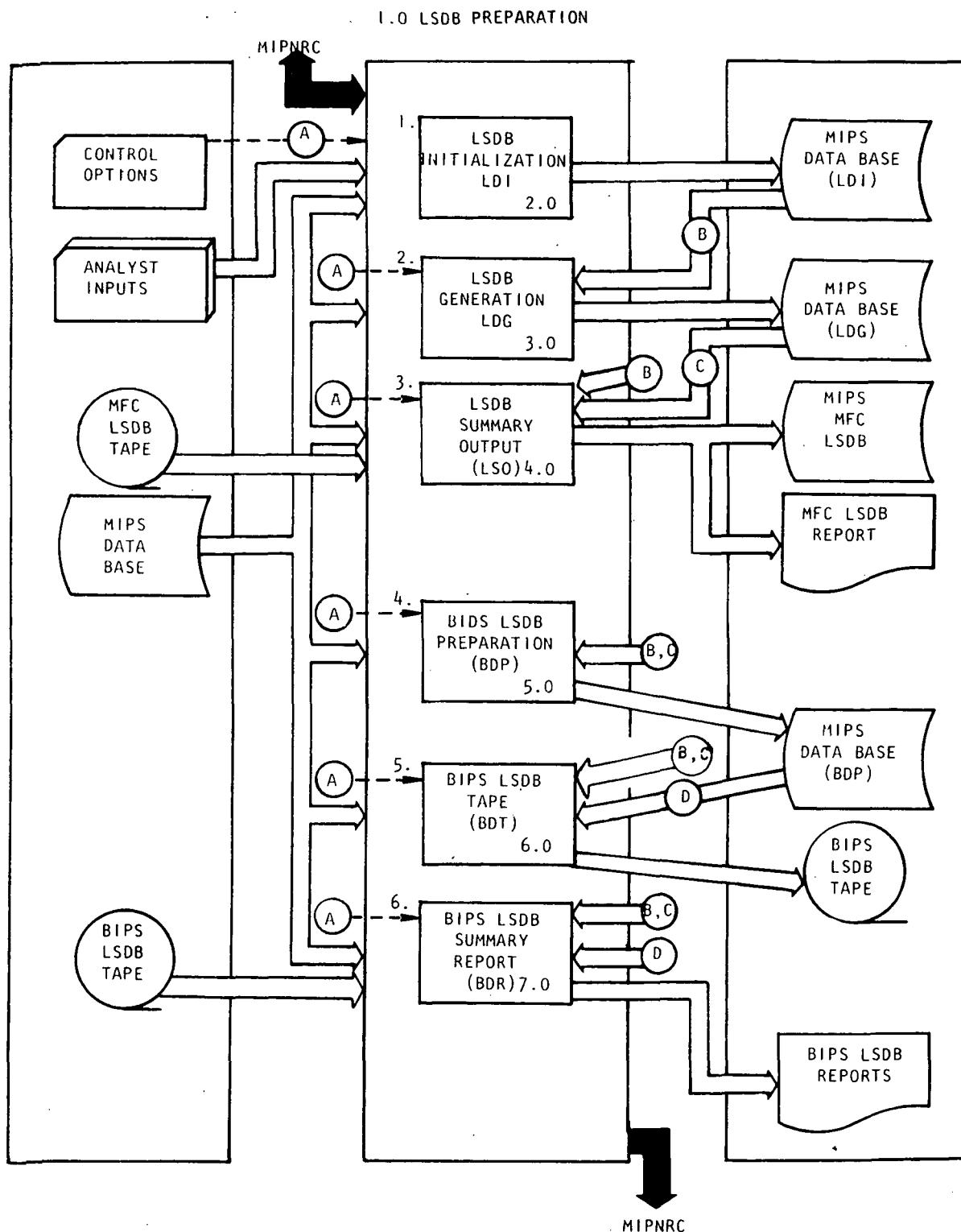


Figure A-8. Top-Level HIPO Chart

APPENDIX B

ASTROS

Advanced Systematic Techniques

for

Reliable Operational Software:

Another Look

By

J. A. Salazar, SAMTEC

R. R. Hall, Federal Electric Corporation

March 1977

Space and Missile Test Center
Vandenberg Air Force Base, California

ABSTRACT

The Advanced Systematic Techniques for Reliable Operational Software (ASTROS) project is a joint Space and Missile Test Center (SAMTEC) and Rome Air Development Center (RADC) effort to validate the claimed benefits from the application of modern programming practices in an Air Force operational environment.

The ASTROS project was briefed to the Range Commander's Council in April 1976 at Patuxent, Md. by Lt. Col. Everett A. Lyons, III from a paper co-authored by him and Mr. Robert R. Hall. At that time SAMTEC had just undertaken the measurement of the target program, the Launch Support Data Base (LSDB) configuration item, for this technology investigation.

This paper relates the progress of the ASTROS project in the last year. Special emphasis is given on the management problems encountered so that any agency which is contemplating a "structured programming" project can take advantage of SAMTEC's experiences and recommendations.

ASTROS: Advanced Systematic Techniques for Reliable Operational Software: Another Look

1.0 Introduction

The purpose of this paper is to detail the progress made in the joint SAMTEC-RADC investigation into the applicability of modern programming practices. A little background will be given into the beginning of the ASTROS project, but the paper will focus upon the progress of the Launch Support Data Base (LSDB) measurement activity. The emphasis will be in the management problems encountered and what steps SAMTEC has taken to overcome them. It is hoped that other organizations involved in structured programming activities (or those about to become involved) can get some value from the SAMTEC experience.

There is a genuine need for actual and reliable statistics on the value of structured programming techniques in an operational environment. A project needs to be thoroughly documented so that the tools and techniques used are understood. The quantities measured, their method of evaluation, and the variables affecting the statistics must be described. Equally important is a description of the "normal" operational environment and the comparative statistics gathered from its production. The management planning, involvement, and support must be defined. The costs incurred in implementation, procurement of software tools, training, continuing support and overhead to existing operations must be recognized and documented.

We at SAMTEC are concerned with the rising costs and have been interested in implementing advanced programming techniques in an effort to improve programmer productivity and increase software reliability.

The Rome Air Development Center (RADC) shares our concern and also recognizes the paucity of objective measurement data on the application of structured programming. At the confluence of these objectives the ASTROS project was born. SAMTEC is implementing contemporary technology to selected projects and objectively measuring them. Since the measurement is objective, infeasibility is an acceptable outcome.

1.1 Purpose of ASTROS

The objective of the ASTROS project, simply stated, was to investigate and validate the various structured programming concepts and tools in the SAMTEC operational environment. This investigation has been done with the goal of improving programmer productivity and developing more reliable software. More precisely, ASTROS applied structured programming techniques to selected programming projects in order to validate the hypothesis (generally accepted throughout the industry) that these techniques will yield (1) lower cost per line of code, (2) higher quality code (less errors), (3) more easily maintainable code, and (4) more realistic schedules. Initially, these investigations have been confined to a single processor, IBM 360/65, and to a single higher level language, FORTRAN.

A secondary, but important, objective has been to provide RADC with productivity measurements within the Air Force operational environment in order to objectively evaluate the benefits derived in applying these techniques. Also, we need to trade-off the extra costs for support and system overhead needed to operate within a structured environment against any productivity and reliability gains.

The objectives suggested certain methods of approach to best provide an environment for objective evaluation of the data gathered in order to provide validation of the above hypothesis. In short, "How do we prove structured programming works without stacking the deck?" and "How do we measure without imposing distortion on the results by the measuring process?" The basic approach was to keep the cards on the table, i.e. (1) declare methodology, measurement controls, evaluation criteria, expectations, etc. prior to gathering any measurements and (2) minimize non-contributory measuring.

To achieve the objectives stated above, the ASTROS project concentrated on three areas: (1) investigation and validation of structured programming tools and concepts, (2) management aspects of structured programming, and (3) measurement.

2.0 Background.

A little departure here to acquaint the reader with a little background of the ASTROS project.

2.1 Beginnings.

In December 1974, SAMTEC invited representatives from RADC, the Air Staff, SAMSO and PMR; as well as members of all the software development companies at the Western Test Range; to a conference at Vandenberg AFB. The purpose of the conference was to acquaint the attendees with modern programming practices, often given the generic term "structured programming". After a day of lectures and presentations, the attendees were split into working groups to devise an implementation plan for applying these modern programming practices in the SAMTEC operational environment. The interim reports of these working groups were integrated and compiled into an Implementation Plan approved and published in February 1975.

2.2 Implementation Steps.

The Implementation Plan detailed a series of phased steps necessary in order to place SAMTEC into a stance to undertake the development of a target project utilizing the modern programming practices and to measure the results. It is certainly pleasing to report that each of the steps was carried out on schedule. A brief description of each of the steps is described below.

2.2.1 Formation of a Team.

The Implementation Team was modeled after a Chief Programmer Team. The Project Leader was the primary decision maker and reviewer of the rest of the team's output. The Librarians served as a funnel for all information flow into and out of the team. Selected technical personnel were added to the team as their functions were required.

2.2.2 Set Up a Technology Library.

A specialized "structured programming" library was installed. The primary purpose of this library was to provide the team with all available technology so that they could become familiar with modern programming practices and learn from other implementers experiences. A secondary purpose was to provide a technology center for "state-of-the-art" thinking in modern software engineering circles. As time has gone on the technology center has grown in importance and size. There are over fifty books and over 500 magazine articles, technical reports, etc. in the library. Requests for information are received from all over the country.

2.2.3 Define, Roles, Functions and Procedures for Involved Personnel.

It was important to SAMTEC that, prior to embarking upon a development project, the roles of the personnel involved would be defined. The way the team would function, how a structured programming team would operate in an unstructured environment, and what deviations and waivers would be needed from established Military Standards were all addressed. Structured programming standards and conventions were established. The duties of each of the team members was defined. The contents of the Systems Development Library was detailed. All of this was done in general terms before the selection of a particular development project was made.

2.2.4 Selection of Support Software.

The ASTROS Implementation Plan identified the selection and procurement of a Program Support Library (PSL) as the pacing item in the implementation scheme. It also stated it was key to a chief programmer team operation. The IBM 370/65 computer and the FORTRAN language were identified as the target computer and target language. The Applied Data Research (ADR) LIBRARIAN was selected as the PSL with the most readily available functions on the IBM 360/65. Since FORTRAN was the selected language a pre-compiler was necessary in order to program using the structured code constructs. A rigorous validation process yielded Caine, Farber and Gordon's S-FORTRAN pre-compiler as the best available at the time. Other automated tools such as program design language generators, code auditors, automated test case generators, code analyzers and restructuring programs were all considered. It was decided that no other tools would be necessary for this first investigation.

2.2.5 Definition of the Management Process.

A Structured Programming Life Cycle was devised in order to show how a structured programming project could be implemented into the standard Military Program Life Cycle. The approach here was to show an evolutionary, rather than revolutionary, process. The reviews and audits were described. Special documentation necessary in a structured environment was detailed. The "structured walk-through" procedures were established.

2.2.6 Definition of Measurement Forms and Procedures.

A great deal of energy went into deciding what to measure. The tendency to count something just because it was countable was strenuously avoided; we intended to minimize the impact of the measuring process upon the developer. Also, great pains were taken to assure the developers that we are measuring the technology and not the people. How well this is believed will influence the objectivity and usefulness of the data.

This goal of obtaining meaningful, quantitative data in order to perform an objective analysis without impacting the software developer's schedules may be unrealistic. First of all, the developers know they are being measured and the "Hawthorne effect" cannot be ignored. Second, much of the information required has to be provided by the developers. In order to minimize the impact of the measurement process, a large part of the data being gathered has been automatically captured by placing "hooks" into the LIBRARIAN software and into the IBM Accounting Software. Also, manual forms have been devised; most of them are completed on a "one-shot" basis. All forms are short and concise with explicit instructions on the back. Although the forms have been designed for this project, they could have applicability in other software development activities.

The Personnel Profile is filled out once by everybody when they are assigned to the project. The General Contract/Project Summary and the Management Methodology Summary are filled out by the Project Lead/Chief Programmer once at the start of the project. The Design and Processor Summary and the Testing Summary are completed prior to the Critical Design Review (CDR), i.e. the last formal review before start of coding. The Systems Development Log is a multi-purpose form completed on an irregular basis whenever a significant event transpires which could have an impact on design and/or schedule. It is also completed when a document or software increment is delivered, when a review or audit has taken place, or when a piece of software passes a testing phase. The Computer Program Run Analysis Report is filled out for each job submittal. Most of the information on this form is available on the program listing. If a run is unsuccessful, the Computer Program Failure Analysis Report is completed. Much work went into generically categorizing the many possible errors to yield a workable set; error examples are included on the back. As each person leaves the project, either because it is complete, they have a new assignment, or they are leaving; they are asked to complete the Technology Critique. The General Project Wrap-Up Report is filled out by the Project Lead/Chief Programmer at the completion of the project.

A Weekly Module Report will be generated automatically from the librarian software, the ADR LIBRARIAN. This report will keep track of module size, number of updates, number of runs, etc. A unique feature of this report, to be used as a means for management visibility and control, is the reporting facility. The report can be generated a) periodically (weekly) b) on demand or c) on exception.

2.2.7 Selection of Target Projects.

The ASTROS Implementation Plan established the constraint on the target project that all software produced had to be something that

would have to be developed anyway. This ruled out a pure experiment or a parallel development. We centered upon the MIPS (Metric Integrated Processing Systems) for candidates for measurement and comparison. The MIPS project holds appeal for application of advanced techniques for such reasons as:

1. It is a large system being developed incrementally
2. The environment is defined and controlled (processor, system, data base, reporting structure, etc.)
3. It is well managed and highly visible
4. The IBM 360/65 has more automated tools available than any other SAMTEC processor
5. The end-product is useful with a long projected life
6. All of the Non-Real Time Software is being developed using a common language, FORTRAN
7. Its increments are non-trivial, i.e. complex algorithms, large data bases and interfaces, and significant manpower requirements (10-15 man-years per increment).

Three Non-Real Time Increments were selected. One, the Data Analysis Processor (DAP), is being measured and developed using traditional programming techniques. Another, the Launch Support Data Base Generation (LSDB) is being measured and developed using such advanced techniques as top-down design, HIPO, structured code, program support library, chief programmer teams, and structured walk-throughs. The third increment, History Tape Generation (HTG), is being developed using the advanced techniques but will not be measured.

2.2.8 Training of Involved Personnel.

The ASTROS team defined six different training courses for specialized application of the various structured programming techniques. Vandenberg personnel were provided the necessary training required in order to implement advanced programming techniques.

1. Overview.

This course was a general survey of what, why, who and how of structured programming. Each of the tools and concepts, along with the management approach of the ASTROS team, was briefly highlighted. This was a two hour presentation.

2. Structured Design.

This course included discussions of top down design, Program Design Languages, HIPO, Threads and top down test. The course augmented technical discussion with in-class problem solving using the team approach and structured walk throughs. This course was a 40 hour course, taught 4 hours a day for two weeks.

3. Structured Code.

The theory of structured constructs, their history, and mathematical proofs were discussed briefly. The bulk of the course was an instruction in the usage of Caine, Farber, Gordon S-FORTRAN pre-compiler with emphasis on coding and review of actual problems. This course was 20 hours in length, taught 4 hours a day for one week.

4. ADR LIBRARIAN.

This course concentrated on the optional features of the ADR LIBRARIAN and how to best utilize them. It included a discussion and working example of use of the Systems Management Facilities (SMF). This was a 20 hour course taught in 5-four sessions.

5. Measurement Reporting.

This course discussed the measurement reporting forms and explain how to fill them out. Emphasis was given to the error classifications and to the meanings of each classification. Ways of automating the collection function by using the ADR Librarian was discussed. This 4 hours, taught in two hour segments with follow-up OJT and monitoring during the developmental phase.

6. Management of Structured Project.

This course addressed the systems management aspects of structured programming. MIL-STD's were discussed as to their applicability, deviations were defined. Measurement reporting and management controls available in the ADR LIBRARIAN were discussed. A structured programming life cycle was presented and discussed. Chief programmer team organization and structured walk throughs were highlighted. This was a 20 hour course taught in 5-four sessions.

2.2.9 Exercise Concepts and Measure Results.

After the above activities were completed, SAMTEC was ready to procede with the technology investigation. A joint SAMTEC/RADC

Memorandum of Agreement was signed in February 1975 and the development and the measurement of the Launch Support Data Base Generation (LSDB) Computer Program Configuration Item (CPCI) was initiated. It was at this point that the ASTROS project was presented to the Range Commanders Council in April 1975. Many interesting things have happened since, leading to some enlightening discoveries. The story of the last year comprises the rest of this paper.

3.0 LSDB Measurement Activity

The Launch Support Data Base CPCI has been continuously measured and monitored since February 1976. The project is scheduled for completion in June 1977. This paragraph will provide a brief description of the project, explain the advanced programming techniques being applied to the project, detail what measurements are being gathered, discuss the development progress and the management problems encountered, and take a look at the present status.

3.1 Description of the Project

The LSDB CPCI is a non-real time increment of the Metric Integrated Processing System (MIPS). It is being developed on the IBM 360/65 using a Remote Job Entry (RJE) terminal as the primary submittal technique and operates in a batch mode. The program computes the data base parameters used by the Missile Flight Control (MFC) CPCI in its real-time range safety calculations. LSDB is comprised of six major Computer Programming Components (CPC's) each with many sub-modules, subroutines and procedures. The error and productivity measurements are being collected on the CPC level. There are four full time members of the development team; the chief programmer, two programmers and a programming librarian. The integration testing is being conducted by an independent testing group within the MIPS organization.

3.2 Advanced Techniques Applied

Below is a short description of the advanced programming techniques being applied by the LSDB development team. These techniques and concepts include a top down development approach, structured code, Hierarchy plus Input-Process-Output (HIPO), a Chief Programmer Team and Structured Walk-Throughs.

3.2.1 Top Down Development Approach

Top-Down Design approach is used in the LSDB development. Each subsystem is designed from the control sections down to the lowest level sub-routine prior to the start of code. There is one HIPO per subsystem which is expanded until the lowest level of detail is reached.

Top-Down test requires the highest level unit or units of a system or subsystem to be coded and tested first. Since that unit will normally invoke lower level units, dummy code must be substituted temporarily for the lower level units. The required dummy units (program stubs) may be generalized, prestored on disk, and included automatically by the linkage editor during a test run (as in the case of a CALL sequence). Although the program stubs do not normally perform any meaningful computations, they can output a message for debugging purposes each time they are executed. Thus, it is possible to exercise and check the processing paths in the highest level unit before initiating implementation of the lower level units which it invokes. The lower level units are built and tested in the same manner, using stubs for programs which they invoke. This procedure is repeated, substituting actual program units for the dummy units at successively lower levels until the entire system has been integrated and tested.

The important point is that program units at each level are full integrated and tested with their predecessors before coding begins at the next lower level.

The use of dummy calling routines or test drivers is prohibited unless a Request for Deviation/Waiver has been submitted and approved.

A routine is not be implemented until the calling routine has been implemented. Data definition statements are coded and the actual data records are generated before exercising any segment which references them. Top-Down Implementation does not imply that the implementation must proceed down the hierarchy in parallel. Some branches intentionally will be developed earlier than other branches. All code will be produced by members of the team and reviewed by at least one member. If a logic change is required during code generation or review, the HIPO is updated and appropriate entries will be made in the development work book. If the logic changes are made to a controlled portion of the HIPO, (e.g., a HIPO in the Part I), a formal review is performed prior to proceeding. In general, HIPO's are updated continuously throughout project duration to reflect the latest design/documentation of programs being developed. The CPC/CPCI test procedures are updated continuously during the implementation period.

3.2.2 Structured Code

The five constructs illustrated below are adhered to wherever possible. The Caine, Farber & Gordon pre-compiler S-FORTRAN shall be used for all programs except Processing Modules (PM's). Processing Modules are MIPS special purpose programs designed to input and output data.

3.2.3 Hierarchy plus Input-Process-Output (HIPO).

HIPO is the advanced programming technique used to support documentation. HIPO is used as both a documentation technique and as an aid in the support of top-down design. HIPO documentation standards are contained in the IBM document "HIPO - A Design Aid and Documentation Technique". This document is used as a guide. The concepts are used without all the drafting requirements.

3.2.4 Chief Programmer Team.

A chief programmer team is used in the development of the LSDB CPCI. The team consists of the chief programmer and back-up programmer, programmers, and a librarian.

Responsibilities of the team are:

- a. Chief Programmer is responsible to the non-real time supervisor for the development of the programming system. This person shall carry technical responsibility for the project including management coordination; production of the critical core of the programming system in detailed code, direct specification of all other codes required for system implementation, and review of the integration of that code. Individual responsibilities are identified below:
 1. Function as first level team manager.
 2. Approve all HIPO's.
 3. Conduct structured walk-throughs for Preliminary Design Review (PDR) and Critical Design Review (CDR).
 4. Chair the working walk-throughs with the team.
 5. Approve all code.
 6. Review and approve all code checkout runs and certify the software ready for CPC/CPCI test.
 7. Be responsible for preparation of:
 - (a) Test Plan.
 - (b) Training Plan.
 - (c) Users Manuals.

- (d) Classroom training of users.
 - (e) Test Procedures.
 - (f) Development Specification; Product Specifications.
8. Generate entries for the appropriate documentation sections of the LSDB Development Library.
 9. Be responsible for Measurements.
- b. Back-Up Programmer supports the chief programmer at a detailed task level so that to assume the chief programmer's responsibility temporarily or permanently. This person, though primarily a programmer, may be called upon to explore alternate design approaches, independent test planning, or other special tasks but will serve normally as an active participant in technical design, internal supervision, and external management functions.
- c. Programmers responsibilities are, as a minimum:
1. Code assigned routines.
 2. Review code.
 3. Generate/gather subsystem test data.
 4. Generate/gather unit test data.
 5. Code necessary stubs.
 6. Review development specifications and test plans.
 7. Write Users Manuals.
 8. Write training plans.
 9. Train users.
 10. Generate entries for the appropriate document sections of the LSDB development library.
- d. Librarian assembles, compiles, and linkage-edits programs and submits test-runs as requested by project programmers. The librarian has responsibility for the project-critical task of maintaining the library. The librarian also performs the following functions:

1. Submit All Computer Runs. The programming team members must submit all computer jobs and/or keypunch forms to the librarian. The librarian punches the cards, verifies all JCL, or adds the required JCL and submits the job. The librarian also corrects all obvious JCL, FORTRAN, and Assembler errors.
2. Catalog and Bind All Computer Runs. All computer jobs are cataloged and kept in a binder. The catalog contains the job name and data, filed in chronological order.
3. Maintain the Development Data Base. The librarian is responsible for running a variety of programs in support of the developer's data base. This requires performing the following functions:
 - (a) Adding new or replacing functions
 - (b) Adding new or replacing subroutines
 - (c) Allocating or deleting data sets
 - (d) Maintaining source in a LIBRARIAN data set
 - (e) Executing all the appropriate utilities to maintain data base integrity such as DBEXAM, DBDELETE, DBSAVE, DBCOPY, DBMOVE, FDR, COMPRESS, VTOCLIST etc.
4. Maintain the LSDB Development Library. The librarian maintains all the document sections of the LSDB development library.
 - o Add new or replacement functions.
 - o Add new or replacement subroutines.
 - o Allocate or delete data sets.
 - o Maintain source in a LIBRARIAN data set.
 - o Configuration control of all source not maintained by ADR Librarian must be provided.
5. Maintain the Subsystem Work Book. The librarian maintains a developers subsystem work book.

6. Team Support. The librarian performs keypunching, typing, editing and drafting for the team when these normal services are not available from the support groups.

7. Official Project Recorder. The librarian compiles and issues all walk through and meeting minutes.

3.2.5 Structured Walk-Throughs.

A structured walk-through is conducted on all CPC's of the LSDB CPCI. There will be two types of structured walk-throughs, a working walk-through and a formal walk-through, each of which is described below. Not only can these reviews help the developer to find errors in his work earlier in the development cycle but reviewers have an opportunity to learn new approaches and techniques and can be kept informed of the characteristics of work related to their areas of responsibility.

The working walk-through is a structured walk-through with the attendees composed of the LSDB structured programming team. Non-team members shall attend at the chief programmer's discretion, e.g., a member of Systems Engineering staff shall be invited to attend a working walk-through of the CPC/CPCI Test Procedures.

At the formal structured walk-throughs for the PDR & CDR, attendees from other groups are invited by the developer. Since it is difficult to conduct a structured walk-through for a large group of people, PDR and CDR attendance is limited by the Non-Real-Time Branch Supervisor. While the entire Chief Programming team may be present, only the chief programmer or his designate will conduct the walk-through. Action items, using standard MIPS Action Item Forms, shall be assigned rather than attempting to resolve problems at the walk-through. Reasons for the Chief Programmer Team's approach shall be presented when several reasonable alternatives were considered.

The basic characteristics of the structured walk-through are:

- a. It is arranged and scheduled by the developer of the work product being reviewed.
- b. The Chief Programmer selects the list of reviewers but the Non-Real-Time branch supervisor may review the list to ensure that the review team will be able to provide an adequate review. External participants may include:
 1. Management.
 2. Developers of other parts of the MIPS system.
 3. Supporting users and hardware engineers.

4. Testers responsible for component and system testing.
 5. Designers of the system to ensure compatibility and continuity of design.
 6. Individuals responsible for documenting the function being reviewed.
- c. All of the developer's products are reviewed during at least one working walk-through.
 - d. The reviewers are given the review materials at least four to six days prior to the walk-through and are expected to review them and come to the session with a list of questions.
 - e. The walk-through is structured in the sense that all attendees know what it is to be accomplished and what role they are to play. The emphasis during the walk-through is on problem identification rather than problem solution.
 - f. The chief programmer chairs the session and the librarian records all errors, discrepancies, exposures, and inconsistencies uncovered during the walk-through. MIPS Action Item forms are used to record all actions to be taken and responses thereto.
 - g. A typical walk-through is scheduled to last for a specified period of time, usually one or two hours. If the session's objectives have not been met at the end of that period, another walk-through is scheduled for the next convenient time.
 - h. The following actions occur at a walk-through. First, the reviewers are requested to comment on the completeness, accuracy, and general quality of the work product. Major concerns are expressed and identified as areas for potential follow-up. The product author gives a brief tutorial overview of the work product. The product author "walks" the reviewers through the work product in a step-by-step fashion which follows the logic of the function under investigation. The product author takes the reviewers through the material in enough detail to satisfy the major concerns expressed earlier in the meeting.
 - i. Immediately after the meeting, the librarian distributes copies of the handwritten action items to all the attendees. The chief programmer, supported by management, ensures that the action items are successfully resolved, and that the reviewers are notified of the actions taken or of the corrections made.

3.3 Measurements.

As mentioned above the LSDB project is being measured and closely monitored during its entire development. The manual measurement forms described in paragraph 2.2.6 are being augmented by automated measurement gathering facilities available in the ADR LIBRARIAN and the IBM Systems Management Facility (SMF) of the Operating System (OS). Of particular note is the great bulk of archival data being gathered in support of the measurements. For instance, every computer listing ever generated since the project inception has been saved. The LSDB Structured Programming Library contains Standards and Conventions, the Development Work Book, the Test Work Book, the Monitor Work Book, Development Documents, Structured Programming Documents, and the LSDB Support Documents. These items are discussed in more detail in paragraph 3.3.3 below.

3.3.1 Manual Forms.

The manual forms were devised so that most of them are completed on a "one-shot" basis. All forms are short and concise with explicit instructions on the back. Although the forms have been designed for this project, they could have applicability in other software development activities.

The Personnel Profile is filled out once by everybody when they are assigned to the project. The General Contract/Project Summary and the Management Methodology Summary are filled out by the Project Lead/Chief Programmer once at the start of the project. The Design and Processor Summary and the Testing Summary are completed prior to the Critical Design Review (CDR), i.e. the last formal review before start of coding. The Systems Development Log is a multi-purpose form completed on an irregular basis whenever a significant event transpires which could have an impact on design and/or schedule. It is also completed when a document or software increment is delivered, when a review or audit has taken place, or when a piece of software passes a testing phase. The Computer Program Run Analysis Report is filled out for each job submittal. Most of the information on this form is available on the program listing. If a run is unsuccessful, the Computer Program Failure Analysis Report is completed. Much work went into generically categorizing the many possible errors to yield a workable set; error examples are included on the back. As each person leaves the project, either because it is complete, they have a new assignment, or they are leaving; they are asked to complete the Technology Critique. The General Project Wrap-Up Report is filled out by the Project Lead/Chief Programmer at the completion of the project.

Particular attention should be given to the Computer Program Run Analysis form and the Computer Program Failure Analysis form. The first form is filled out for every job submittal. The second form is completed when any job failure occurs. The bulk of the software reliability information will be analysed by using these two forms. The

Software Development Log is useful in reconstructing the project scenario including document deliveries, reviews and audits, and major design decisions.

3.3.2 Automated Measurements.

In order to take much of the burden of the project control and measurement gathering off from the LSDB development team, automated measures are gathered from the ADR LIBRARIAN and the IBM OS Systems Management Facility.

A Weekly Module Report will be generated automatically from the librarian software, the ADR LIBRARIAN. This report will keep track of module size, number of updates, number of runs, etc. A unique feature of this report, to be used as a means for management visibility and control, is the reporting facility. The report can be generated a) periodically (weekly) b) on demand or c) on exception.

The program accounting information is gathered from SMF. The compile times, pre-compile times, linkage edit times and run times are identified and accumulated on a module-by-module basis. This information is also available on a CPCI level. Other information such as turn-around time and I/O peripheral times are also captured.

3.3.3 Archival Data.

As mentioned above, every computer listing generated by the project has been retained as part of the back-up archival information. In addition, the LSDB Structured Programming Library contains much valuable information to completely describes the processing environment in which this development effort is being accomplished. The information contained in this library is detailed below.

1. Standards and Conventions. The Standards and Conventions section contains the following items.
 - a. HIPO Standards
 - b. Structured Code Standards
 - c. Top Down Test Standards
 - d. Library Maintenance Procedures
 - e. Data Base Maintenance Procedures
 - f. Submittal Procedures
- (1) Development to Production Library

- (2) Documentation including Computer Listings
 - g. Job Submittal Procedures
 - h. File Description Procedures
 - i. Procedures for updating the development work book
 - j. Procedures for updating the monitor work book
 - k. Procedures for updating the test work book
 - l. ADR LIBRARIAN procedures
 - m. Code and message standards
- 2. Work Books. There are several work books which are kept current during the development process. These include
 - a. Development Work Book
 - (1) Decision logs/reason for design approach.
 - (2) Notes from all walk-throughs.
 - (3) HIPO/code problems that were uncovered.
 - (4) Results of AET testing.
 - (5) Deviation/waiver log.
 - (6) Memorandums.
 - b. Test Work Book
 - (1) Test plan guidance.
 - (2) Test report guidance.
 - (3) LSDB test plan.
 - (4) Top-down testing approach for MIPLSD
 - c. Monitor Work Book
 - (1) General contract/project summary.
 - (2) Management methodology summary.
 - (3) Design and processor summary.

- (4) Testing summary.
 - (5) Technology critique.
 - (6) Personnel Profile.
 - (7) System Development log.
 - (8) Computer program failure analysis report.
 - (9) General project wrap-up report.
 - (10) Notes.
 - (11) Computer run analysis reports contained in several volumes in chronological order.
3. Development Documents. Documents which have been prepared during the development process are also retained. These include
- a. LSDB development specifications.
 - b. LSDB test plan.
 - c. LSDB test procedures.
 - d. CDR material.
 - e. Product specifications.
 - f. Action items.
 - g. Draft documents.
4. Structured Programming Documents. The following documents are kept as reference documents
- a. Structured programming courses handouts.
 - b. S-FORTRAN.
 - c. IBM JCL/FORTRAN.
 - d. IBM Structured Programming test/workbook.
5. LSDB Support Documents. The following documents are retained as interface reference material for the LSDB developers.
- a. VPAR 34 Vol. I.
 - b. VPAR 34 Vol. II.

- c. MFC CDR Material.
- d. MFC development specifications.
- e. System segment specifications.

3.4 Project Progress.

The LSDB development project has progressed on, or near, schedule during its whole lifetime. All major milestones have been met. This is despite the fact that the walk-throughs have uncovered several design deficiencies which have been corrected without impacting the schedule. Ordinarily many of these deficiencies would not be uncovered until demonstration testing (DT&E) or beyond. The user involvement in the walk-throughs has led to some design modifications due to requirement changes. In a "traditional" development life cycle these modifications most certainly would not have been made until after the software had become operational. Then the cost of making these modifications would probably have been significantly higher.

3.5 Project Problems.

Over a year of planning went into the preparation of this technology investigation. Other implementer's experiences were viewed in light of the SAMTEC operational environment. Theoretically everything was addressed and all contingencies planned for. Yet we have had some problems. The importance of this paper seems to be in the fact that despite careful planning, rigorous application, and close monitoring, certain unforeseen difficulties did arise. The following paragraphs the most problems and what our management approach was to solving them.

3.5.1 Organizing a Library.

The joint SAMTEC/RADC Memorandum of Agreement stated that SAMTEC would collect measurements and deliver them to RADC. Although the manual measurement forms and the automated processes were established prior to the start of the LSDB development project, the archival (or support) data had not been addressed. The measurements were decided upon, but until the project was underway the necessity for retaining archival data was not realized. What data should be kept, what form should it be preserved, and how was it to be delivered were all things that were addressed after the project was underway. The burden of establishing the LSDB development library fell on the project monitor and, primarily, upon the programming librarian. Establishing what should go into each of the work books, establishing procedures for maintaining the library, and establishing the responsibilities of all the involved personnel was an arduous task. The day-by-day development activities progressed while much of this data had to be recovered. It was a long, difficult process. Everything is in place now, but the task would have been much easier had these items, procedures, responsibilities, etc. been established in advance.

3.5.2 Single "Structured" Project in an Unstructured Environment.

The choice of a MIPS CPCI was a good one for the reasons detailed in paragraph 2.2.7. However, there are some inherent problems in taking a piece of a system under development and change the development ground rules for that piece. Many of these difficulties were anticipated and the LSDB Implementation Plan specifically addressed organizational structure and responsibilities, walk-through procedures, and MIL-STD deviations required in reviews and documentation. What was not anticipated were difficulties uncovered by the technology itself. The top down development method allows for design to progress hierarchically down through the structure establishing the interfaces as the detail expanded. In a system where the data base had previously been established, the increments, or CPCI's defined, and their interfaces described, much of the top down design work has been done. This places design constraints at the CPCI level; the data interfaces suggest the module interfaces rather than the other way around. Another difficulty involves team operations. In theory placing a librarian onto a development team relieves the technical people of the administrative and clerical tasks and allows management to utilize their talents in a more effective manner. This has not proved true in our technology investigation for a very peculiar reason. The librarian has effectively relieved the developers of the clerical functions which once took so much of their time. However, since they are working on a target project which was using different techniques in comparison with the other development efforts, management had to assign other efforts to fill the void. This suggests better planning of manpower utilization is needed when using this technology. The technical people will become more productive only when their given more to do.

3.5.3 Impact of Measurements on the Development.

Once again, we foresaw that there would be an impact made by the measurement process. We strived to minimize this impact with the careful development of the manual forms and by augmenting the measurement gathering by automated means. We also placed more personnel on the project to absorb the measurement burden. This took care of the physical impact but didn't take care of the psychological impact. People do not like to be watched while they're working. Despite the minimal effect the measurements have upon their day-to-day activities, there is certainly an antagonism in knowing they are being measured at all. We are witnessing a sort of "inverse - Hawthorne effect". Continued assurances that we are measuring the technology and not the people has helped keep the measurement process flowing.

3.5.4 Communication in Walk-Throughs.

Both the developers and users at SAMTEC have been used to communicating at Preliminary Design Reviews (PDR's) or Critical Design

Reviews (CDR's). Both of these reviews are rather formal, involve a lot of people, and usually a lot of material. Because of the bulk of information, a whole CPCI, these reviews seldom get into design detail. In going to a structured walk-through format, SAMTEC established explicit rules as to the attendance, format, and materials to be covered. This was a new procedure for user and developer alike. There was a definite learning curve involved before these parties were comfortable with format. Now this communication seems to flow easily but at first many small procedural modifications were required.

3.5.5 User-Involvement.

There are only two things wrong with user involvement - there's not enough or there's too much. The idea behind the structured walk-throughs was to get the user involved in the day-by-day development process so that he could better see if his requirements are being met. It is felt that design deficiencies and requirement changes could be identified earlier in the life cycle they could be modified with minimal impact upon the schedule. This has certainly proved true in the LSDB development. However, the old statement "a little learning is a dangerous thing" is certainly applicable. As the user has become more and more knowledgeable of the design detail, he has often yielded to the human tendency to dictate that detail. He also has the tendency to want slight requirement changes or "nice-to-have" additions without being aware of the impact of these changes on the system as a whole. This is a problem that is gradually improving as the user's learning goes up and with the developer's adherence to a workable design.

3.5.6 Use of HIPO's as a Communicational and Design Technique.

Probably SAMTEC's biggest management problem has been with the use of HIPOs. Theoretically HIPOs allow the user to see how his requirements have been satisfied by presenting the design in his own words. In practice, learning a new graphical technique, after years of trying to get comfortable with flow charts, was a very confusing and frustrating experience. Once again there is a great learning curve. The developer has to know how to use HIPOs in his design so that they truly communicate his approach. The user needs to understand the graphics and the hierarchy so that he can understand the data flow. Perhaps a Program Design Language (PDL) is the answer. PDL has proven very effective when presented in SAMTEC's training courses. By using PDL in the "Process" portion of a HIPO, an effective means of bridging the "Input" to the "Output" was seen. Also the PDL can be done at any level of detail necessary to communicate the detail required.

3.6 Present Status.

The LSDB project is nearing completion. The CPCI is currently undergoing testing by the independent MIPS test group. The measurement forms and archival data are being microfiched in preparation for delivery to RADDC. The developers are placing more comments in code, documenting, and writing their views of the project and the technology.

Acknowledgement.

This project and this paper would not have been possible without the continued support and encouragement of many people. It would be impossible to name them all, but we would like to single out some people who have made significant contributions to this effort. Mr. Frank Sliwa of RADC has given his enthusiasm, confidence and friendship to all of us during our working relationship. We are indebted to Dr. Herbert Hecht of Aerospace Corporation for his continued support and for permission to use some of his visuals in the presentation. Lt. Col. Everett A. Lyons III (ret.) was instrumental in getting the ASTROS project underway and providing keen management insight. Dr. John P. Johnson and Donald Reifer of Aerospace Corporation have lent their technical knowledge and guidance. David Wulftange of Federal Electric Corporation has performed a yeoman service as the LSDB project monitor. John Johnson of Federal Electric Corporation who has given his support as MIPS Project Manager. And a very special thanks to the LSDB development team - Tom Hull, Chief Programmer; Sue Wagoner, Programming Librarian; John McMillan and John Malengo.

