

AD-A165 231

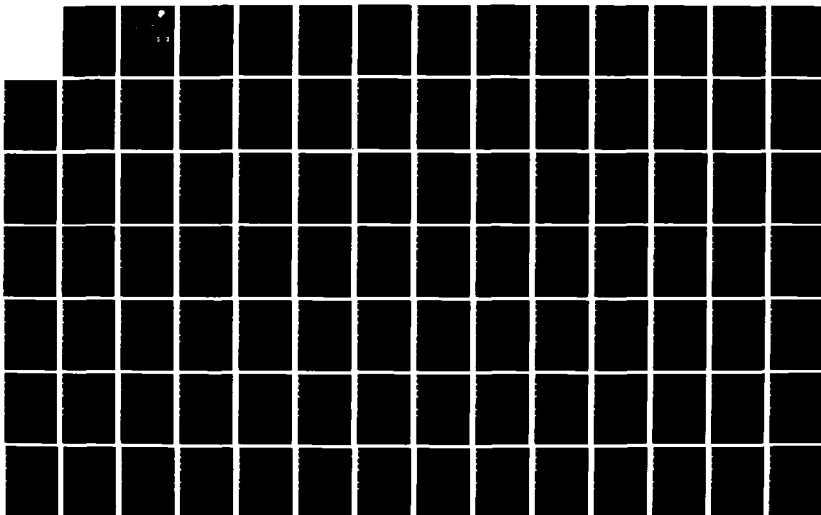
IMPACT OF HARDWARE/SOFTWARE FAULTS ON SYSTEM
RELIABILITY VOLUME 1 STUDY R. (U) MARTIN MARIETTA
AEROSPACE ORLANDO FL E C SOISTMAN ET AL DEC 85
OR-18173 RADC-TR-85-228-VOL-1

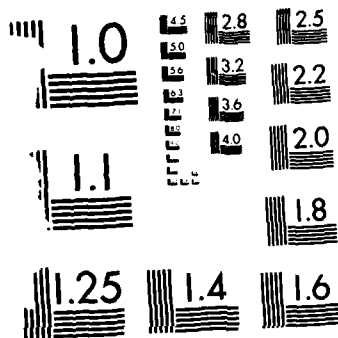
1/3

UNCLASSIFIED

F/G 9/2

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

12



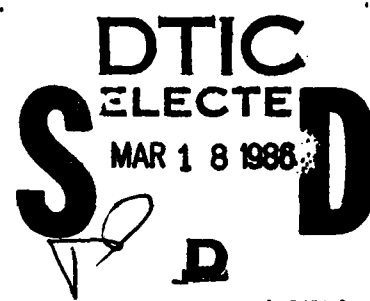
RADC-TR-85-228, Vol I (of two)
Final Technical Report
December 1985

AD-A165 231

IMPACT OF HARDWARE/SOFTWARE FAULTS ON SYSTEM RELIABILITY Study Results

Martin Marietta Orlando Aerospace

Edward C. Soistman and Katherine B. Ragsdale



APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DTIC FILE COPY

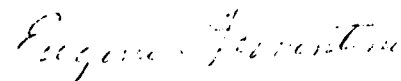
ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

86 3 18 190

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-85-228, Vol I (of two) has been reviewed and is approved for publication.

APPROVED:



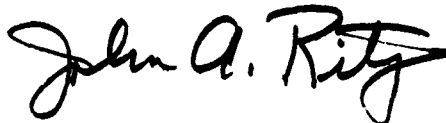
EUGENE FIORENTINO
Project Engineer

APPROVED:



W. S. TUTHILL, COLONEL, USAF
Chief, Reliability & Compatibility Division

FOR THE COMMANDER:



JOHN A. RITZ
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (RBET) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

AD-A165 231

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS N/A		
2a. SECURITY CLASSIFICATION AUTHORITY N/A			3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited.		
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE N/A					
4. PERFORMING ORGANIZATION REPORT NUMBER(S) OR 18,173			5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-85-228, Vol I (of two)		
6a. NAME OF PERFORMING ORGANIZATION Martin Marietta Orlando Aerospace		6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (RBET)		
6c. ADDRESS (City, State, and ZIP Code) P.O. Box 5837 Orlando FL 32855			7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700		
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Rome Air Development Center		8b. OFFICE SYMBOL (If applicable) RBET	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-83-C-0050		
8c. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700			10. SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO 62702F	PROJECT NO. 2338	TASK NO. 02
					WORK UNIT ACCESSION NO. 96
11. TITLE (Include Security Classification) IMPACT OF HARDWARE/SOFTWARE FAULTS ON SYSTEM RELIABILITY Study Results					
12. PERSONAL AUTHOR(S) Edward C. Soistman and Katherine B. Ragsdale					
13a. TYPE OF REPORT Final		13b. TIME COVERED FROM Mar 83 TO Jan 85		14. DATE OF REPORT (Year, Month, Day) December 1985	
				15. PAGE COUNT 206	
16. SUPPLEMENTARY NOTATION N/A					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP			
14	04		Reliability		
09	02		Software Quality		
			Hardware/Software Reliability Prediction		
19. ABSTRACT (Continue on reverse if necessary and identify by block number)					
The objective of this study was to develop techniques, for predicting total system reliability, which include the combined effects of software and hardware. Since hardware reliability techniques are much further developed, the study emphasized methods of characterizing software reliability. The software reliability prediction methodology contained in the report is compatible with hardware reliability techniques and definitions and is applicable during early development so that the predictions can influence the design and development process. The software reliability prediction techniques use both software product and development process characteristics to develop estimates of the reliability of the various software components which comprise the system. The software component reliabilities are combined via a Markov model to obtain estimates of software system reliability. Estimates of the					
20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED		
22a. NAME OF RESPONSIBLE INDIVIDUAL Eugene Fiorentino			22b. TELEPHONE (Include Area Code) (315) 330-3476		22c. OFFICE SYMBOL RADC (RBET)

UNCLASSIFIED

execution frequencies of the various software components, as a function of the mission profile, are required by the methodology.

Procedures for application of the techniques are provided and are intended for use by a reliability engineer having a basic knowledge of software engineering practices. The techniques offer a rudimentary framework for predicting total system (HW & SW) reliability. Validation and refinement of the techniques using software development and field reliability performance data remains to be accomplished.

Item 17. COSATI CODES (Continued)

<u>Field</u>	<u>Group</u>
18	03

UNCLASSIFIED

EXECUTIVE SUMMARY

This study was performed to investigate the impact of hardware and software faults on system reliability and to develop a prediction methodology that is applicable early in the development life cycle when it is still possible to influence the development process and the reliability of the resultant system. Since hardware reliability prediction is relatively well understood and practiced, the study focused on software.

Existing software reliability prediction models rely heavily on fault removal rates experienced when computer software is exercised, either during test or operational usage. Although these models provide meaningful results, they are not directly applicable during early development, but rather after the development process is nearly complete. The methodology presented here is based on parameters that are known early in the development process. As the product evolves, the methodology may be applied iteratively using progressively more accurate input parameters.

This approach is distinct from others in that it focuses on the process used to develop software, as well as the unique characteristics of the product itself:

- 1 Software is the realization of an algorithm that satisfies a given set of performance and functional requirements. Even before the software exists, its requirements are known. The methodology uses this information in the form of inherent characteristics of the software.
- 2 Software is the result of a logical process in the collective minds of its developers. Unlike hardware, software has no physical parts to fail. Software faults, therefore, are the result of errors made during the development process. The methodology incorporates error avoidance and detection information in the form of characteristics of the development process.
- 3 Software cannot fail unless it is used. Likewise, software components cannot fail except when they are being executed. Logical paths are nonexistent with respect to reliability when they are not being executed. Conversely, heavily traveled portions of the software tend to dominate its performance. The methodology uses a Markov technique to determine expected software performance based on mission-derived path probabilities.

A related Rome Air Development Center (RADC) study [103] is currently being performed by Software Applications International Corporation (SAIC). Whereas the objectives of the Martin Marietta study placed emphasis on reliability prediction at the beginning of the development life cycle, the SAIC study is oriented toward the end of the development phase and the beginning of the system integration phase. While this study concentrates on the PROCESS that will be used to develop the product, the SAIC study concentrates on the PRODUCT that results from the process. Although close liaison between studies was maintained to ensure that the studies didn't diverge or overlap, the methodologies and conclusions drawn were independently derived. Interestingly, the methodologies are complementary.

The development life cycle of any system involves the translation of ideas or requirements into a physical product by means of some engineering and/or manufacturing process. In the beginning, the product does not exist, but its functional requirements and planned development process are known. Reliability prediction, therefore, must be accomplished using these known parameters. At the end, the product, good or bad, exists. At this time, the emphasis can, and should, focus on the measurable attributes of the product itself. Both Martin Marietta and SAIC address error density within a software product as a prime determinant of its reliability. The Martin Marietta study focuses on predicting error density based on inherent characteristics of the product and the error avoidance and detection techniques that influence it. The SAIC study has defined characteristic measures of the developed software that can be used to measure its error density.

Other models exist for predicting software reliability based on fault removal rates experienced during integration testing and operational usage. The method presented herein does not conflict or compete with them either. Again, the differences of approach are due to the development phase during which the prediction technique is to be applied. There are no fault removal rates to record until the system is built and operated.

The overall methodology requires the prediction of:

- 1 Individual component (software module) success probabilities
- 2 Expected path probabilities for specific operational missions.

A method is described to predict component success probabilities based on the inherent characteristics of the component and the process that will be used to develop it. Likewise, a means of using the mission scenario to determine path probabilities is presented. Finally, the overall software system reliability is predicted by combining the component success and path probabilities mathematically.

- 1 At the beginning of the development life cycle, prediction is made based on the development process as described herein.
- 2 As components are developed, attributes of the product become measurable, and the techniques being defined by SAIC can be readily used to predict component probabilities of success. In addition, some of the existing reliability models can also be applied to measure the reliability of the components. The same methodology is used, except that one of the primary parameters becomes measurable and provides more accurate inputs to the prediction.
- 3 As the system is integrated and tested in a real or simulated operational environment, the path probabilities become measurable. Again, the methodology is still applicable but has more precise parameters.

While the study has addressed many aspects of the prediction problem, it also revealed several areas where more detailed research and data collection is needed:

- 1 Demonstration of the methodology was accomplished using error avoidance and detection probabilities statistically derived from the two surveys conducted during the study. No adequate and complete historical data base could be found to quantitatively describe the effectiveness of the error detection techniques so highly regarded (qualitatively) in the industry. Although nearly everyone agrees that a code walkthrough is a good error detection technique, available data is insufficient to determine how good. There is a tremendous void of good historical process data.
- 2 Extremely high execution rates of computer programs allow programs to complete thousands of executions in a short period of time. If every execution is considered to be an independent trial, the software reliability for that short period (thousands of executions) would be extremely low unless the single execution reliability was infinitesimally close to unity. Fortunately, a logic path that works correctly one time will always work correctly if it is executed with identical input and state variables. Of course, even a simple computer program is exposed to a nearly infinite number of input and state variable combinations. More detailed analysis of input domain partitioning is needed before we can fully understand and evaluate the effects of continuous cycling of software.
- 3 There is a school of thought that says that software reliability is dominated by rarely used paths. The rationale is that, in an operational environment, input combinations are encountered which cause path traversals that contain errors which have never before been exercised during test. Although the methodology does not currently utilize path probabilities within individual software components, it can be easily modified as our knowledge of rarely used paths increases.

Accession For	
NTIS	CRA&I ☒
DTIC	TAB ☐
Unannounced ☐	
Justification:	
By	
Distribution: /	
Availability Codes	
Dist	Availability or Special
A-1	



TABLE OF CONTENTS

1.0	INTRODUCTION	1
1.1	Overview and Objectives	1
1.2	Approach	2
1.3	Results	4
2.0	LITERATURE SEARCH	7
2.1	Overview and Objectives	7
2.2	Approach	7
2.3	Results	7
2.3.1	Reliability Modeling Approaches	7
2.3.2	Hardware Reliability Factors	9
2.3.3	Software Reliability Factors	12
2.4	Conclusions	16
3.0	CHARACTERIZATION TECHNIQUES FOR RELIABILITY PREDICTION	17
3.1	Overview and Objectives	17
3.2	Approach	17
3.3	Results	17
3.3.1	System Reliability	17
3.3.2	Hardware Subsystem Reliability	22
3.3.3	Software Subsystem Reliability	23
3.4	Conclusions	23
4.0	TWO-PASS SURVEY	25
4.1	Overview and Objectives	25
4.2	Approach	25
4.3	Results	25
4.3.1	Pilot Survey	25
4.3.2	Full-Scale Survey	27
4.4	Conclusions	29
5.0	RELIABILITY PREDICTION METHODOLOGY	31
5.1	Overview and Objectives	31
5.2	Approach	31
5.3	Results	31
5.3.1	System Reliability	32
5.3.2	Mission Reliability	34

5.3.3	Hardware Reliability	37
5.3.4	Software Reliability	38
5.4	Conclusions	52
6.0	DATA COLLECTION	55
6.1	Overview and Objective	55
6.2	Approach	55
6.3	Results	55
6.3.1	Failure Rate	55
6.3.2	Latent Defects	56
6.3.3	Fault Assessment	56
6.3.4	Enhancements	56
6.3.5	Developer versus User	56
6.3.6	Military versus Commercial	57
6.4	Conclusions and Recommendations	57
6.4.1	Record All Software Problems	57
6.4.2	Identify the Source	58
6.4.3	Distinguish Hardware from Software	58
6.4.4	Consistent Data Collection	58
7.0	CONCLUSIONS AND RECOMMENDATIONS	59
APPENDIX A	- GLOSSARY	A-1
APPENDIX B	- LITERATURE REFERENCE LIST	B-1
APPENDIX C	- PILOT SURVEY FORM AND INSTRUCTIONS	C-1
APPENDIX D	- PILOT SURVEY RESULTS	D-1
APPENDIX E	- FULL-SCALE SURVEY FORM AND INSTRUCTIONS	E-1
APPENDIX F	- FULL-SCALE SURVEY RESULTS	F-1

LIST OF FIGURES

Figure 1.	System Reliability Methodology	3
Figure 2.	Classic Approach - New Ingredient	32
Figure 3a.	Single Mission System	33
Figure 3b.	Multi-phased System	33
Figure 4.	Typical Mission Reliability Prediction Approach	37
Figure 5.	Hardware Reliability	38
Figure 6.	Software Reliability Prediction Methodology	39
Figure 7.	Software Decomposition Process and Terminology	41
Figure 8.	Relationship of R(I), R(C), A and D	47

LIST OF TABLES

Table 1. Software Reliability Models	8
Table 2. Factors for Part Failure Rate Models Discrete Semiconductors and Capacitors	10
Table 3. Software Characteristics	18
Table D-1. Twenty Highest Ranked Software Attributes/Factors	D-3
Table D-2. Statistical Results of Pilot Survey	D-4
Table F-1. Profile of Participants	F-3
Table F-2. Ranking of Inherent Characteristics	F-4
Table F-3. Statistical Results of Survey	F-5

1.0 INTRODUCTION

1.1 Overview and Objectives

System reliability characterization and prediction techniques, which incorporate the combined effects of both hardware and software components and which reflect frequency of total system failures during mission or operating time, are needed by U.S. Air Force systems planners and procurement offices. Most of the techniques currently available for predicting the software contributions to overall system reliability rely on knowledge of historical failure-versus-time data to project future performance. Although such methods can be effectively applied during the testing or operational phase of a system life cycle, they are of limited value during early development when alternate design and testing strategies can be evaluated. The methodology developed during this study presents a mechanism for predicting software reliability based on the inherent characteristics of the software and the process used to develop it. Combined with existing prediction techniques for hardware reliability, this technique will enable systems planners and developers to assess overall system reliability at a time when it can still be influenced.

A system can be defined as a collection of all equipment, facilities, procedures, and personnel that together accomplish a specific mission or task. Computers are an integral part of most modern systems and must be included in any system analysis process. While computer reliability can be determined by classical hardware reliability prediction techniques, the software embedded within it presents a new challenge to analysts who must account for its effects on the system.

Virtually every system in both current and planned military inventory has made or will make extensive use of computer technology. In nearly all instances, the computer and its embedded software are not only part of the system, they are essential to the performance of required operational missions. Military systems can be distinguished from many other computerized systems by considering some of their special characteristics:

- 1 Operational missions involve wartime activities that cannot be thoroughly exercised in their true operational environment.
- 2 Overall military tactical and strategic plans and decisions require accurate estimates of the probability of success of individual systems.
- 3 Each system must be available when needed and must operate correctly from the time it is activated until it completes its mission.

In light of the special considerations listed above, it is essential that the prediction methodology encompass the entire system. That is, all system components must be accounted for within the technique. Overall systems reliability is, therefore, defined in terms of its missions and the

specific functions it is required to perform. The following definition is assumed throughout this report:

SYSTEM RELIABILITY - The probability that the required system will perform its intended functions for prescribed mission(s) and time period(s) in the specified operating environment.

Therefore, the objectives of the study were to develop characterization techniques for predicting total system reliability and to develop a practical methodology for use by system reliability engineers early in the development cycle to evaluate various design approaches and alternatives against system quantitative reliability requirements. Recognizing the inherent differences between hardware and software, the methodology was envisioned as consisting of separable, but analogous, prediction techniques for each and a means for combining the results into a single prediction.

Figure 1 illustrates these objectives. Specific attributes affecting system reliability were to be characterized and used to evaluate hardware and/or software reliability. The results would then be combined into a single reliability prediction for the entire system.

As illustrated in Figure 1, the combined system reliability model assumes that separate methods may be applied to hardware and software. Specifically, the method assumes that physical components (hardware) and logical components (software) may be evaluated independently and then combined as equally essential system components. That is, the system reliability definition given earlier can be expressed as a function of system hardware reliability and system software reliability:

SYSTEM HARDWARE RELIABILITY - The probability that the required hardware will perform its intended functions for prescribed mission(s) and time period(s) in the specified operating environment, without causing system outage or failure.

SYSTEM SOFTWARE RELIABILITY - The probability that required software will perform its intended functions for prescribed mission(s) and time period(s) in the specified operating environment, without causing system outage or failure.

Since hardware reliability techniques have been well developed and applied, the study primarily concentrated on the creation of a software reliability prediction methodology. The approach taken was to concentrate on the inherent characteristics of the software and its development process. Hardware aspects of the problem were investigated only for situations where software is interfaced.

1.2 Approach

The approach taken during this effort involved the performance of five interrelated subtasks: a literature search, the characterization of reliability factors, a two-pass survey, the development of a system

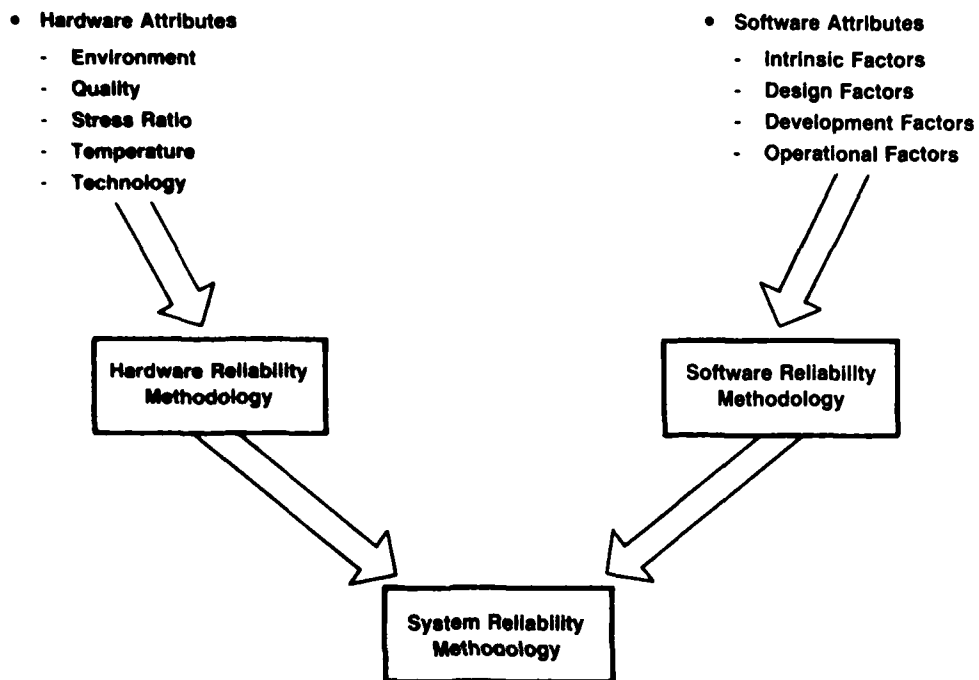


Figure 1. System Reliability Methodology

prediction methodology, and the validation of this method via comparison with data collected from existing systems.

The literature search was used to quickly assess the state of the art with respect to techniques presently available and to identify specific software factors and characteristics that have a direct or indirect effect on system reliability. During the literature search, use was made of several in-house and external data bases by a team of Martin Marietta engineers, representing the Reliability, Systems, Software, and Product Assurance Departments.

During and following the literature search, the many factors that were identified had to be evaluated for their applicability to the planned prediction methodology. Factors were categorized from several different perspectives to determine those which had a significant impact on reliability and those that had little or no impact.

The surveys were used to solicit and summarize the subjective opinions of expert practitioners and to use the results in the development of the methodology. A pilot survey was created and distributed based on the results of the literature search, and its results were used to identify and isolate the most significant software factors and development techniques. A second survey based on the results of the pilot and the requirements of the envisioned methodology was distributed to a much wider audience and was used to determine the quantitative values needed by the methodology.

Methodology development was to be accomplished by the identification, modification and application of existing techniques where possible and by the creation of new techniques where necessary. The intent was to produce a software reliability prediction technique that was compatible with hardware prediction methods and that could be readily combined with hardware reliability estimates.

Concurrent with development of the methodology, real data from operational military systems was collected. The data was to be used to validate the methodology by comparing the actual reliability measured for those systems to the reliability predicted by the methodology.

1.3 Results

More than 500 abstracts and 100 documents were reviewed during the literature search. An on-line data base was constructed and used as a reference throughout the study. Most of the existing software models were reviewed, and many software attributes were identified and considered for inclusion in the method.

In the process of reviewing the factors identified during the literature search, it was found that software attributes could be viewed from various perspectives. Many of the factors were, in reality, attributes of an existing software product (lines of code, number of branches, etc.). Although such factors can be used for prediction purposes, quantitative

relationships between the given attributes (e.g., number of branches) and reliability were found to be non-existent or extremely difficult to quantify using available data. Other factors were found to be characteristics of the software, related to quality considerations rather than reliability. Such factors as readability and flexibility are desirable quality characteristics, but do not directly affect the ability of the software to perform its intended functions.

A pilot survey was conducted to provide a ranking of the relative impact of each factor on reliability. All of the factors which were identified during the literature search, including those that were felt to be unrelated to reliability prediction, were included. The survey was distributed to thirty hand-picked software experts who were asked to rank them. Results were used to eliminate many factors from further consideration. They were also used as the basis for a full-scale survey, which was distributed to more than 300 software practitioners. In the full-scale survey, respondents were asked to quantitatively evaluate each of the factors listed with respect to its influence on software reliability. Approximately 100 persons responded; the results were compiled and incorporated into the methodology.

The literature search also revealed that, although many models exist, nearly all are concerned with the software reliability measurement and prediction by extrapolation. Furthermore, most published works, concerning software development factors and techniques, address the qualitative rather than quantitative aspects of software. Development of the methodology, therefore, required that a new technique be devised. Wherever possible, the methodology uses approaches and techniques developed by other researchers. The basic features of the methodology are:

- 1 Applicable during early design/development
- 2 Applicable throughout the development cycle
- 3 Yields quantitative reliability predictions
- 4 Utility as a design and process evaluation tool
- 5 Uses the operational mission scenario as a basis for prediction
- 6 Compatible with MIL-HDBK-217D techniques and reliability definitions.

A complete validation of the methodology could not be precisely accomplished with existing reliability data. Although extensive amounts of real data was collected during the effort, the completeness of the data was found to be insufficient for making comparisons with the prediction results. The methodology relates inherent functional characteristics and development strategies to predicted performance. Historical data bases tend to include only failure histories, not detailed developmental characteristics. Current and recently completed projects do not yet have an established data base of failure information. The methodology was, therefore, validated by qualitative comparisons to an actual system and through use of several detailed examples.

2.0 LITERATURE SEARCH

2.1 Overview and Objectives

The primary objectives of the literature search were to identify and classify hardware and software reliability factors and to identify existing models and evaluate their applicability with respect to overall study objectives.

2.2 Approach

The four major document sources included in the literature search data base were the Rome Air Development Center customer files, National Technical Information Services, Martin Marietta (Orlando Aerospace) Technical Information Center, and internal technical files. More than 500 abstracts were screened for the literature search.

Five team members were assigned different areas of responsibility in the literature search and a set of articles to be covered. As material was reviewed, it was entered into a data base with the reviewer's annotations. The data base constructed during this phase was subsequently used as a reference and source of information throughout the performance of the study.

Over 100 of the screened documents were reviewed in depth for data pertinent to the study. These were abstracted and commented upon; the results were entered in the study data base. Appendix B contains the title, author, date, source, and abstract of each document reviewed.

2.3 Results

2.3.1 Reliability Modeling Approaches

In the course of the literature search, software reliability models of three different categories were examined: measurement models, estimation models, and prediction models. One of the primary findings of the literature search was that the vast majority of software reliability effort has historically been expended on creating methods for measuring current software reliability and extrapolating that data to estimate the future reliability of existing software packages.

Two measurement models, the Hecht Measurement model and the Nelson model, as well as 19 estimation models were examined. The majority of estimation models are modifications of either the Moranda model or the Markov model. References to three prediction models were found: the Motley and Brooks model; the McCall, Richards, and Walters model; and the Halstead model. Very little was found concerning the prediction of a software package reliability where no historical failure/debugging and run-time data exist from which to estimate and extrapolate. The reliability models reviewed during the literature search are listed in Table 1.

TABLE 1. Software Reliability Models.

Reliability Measurement Models

- Hecht measurement model
- Nelson model

Reliability Estimation Models

- Reliability growth model (La Padjla)
- Mills model
- Rudner model
- Jelinski-Moranda de-eutrophication model
- Jelinski-Moranda geometric de-eutrophication model
- Jelinski-Moranda geometric poisson model
- Schick-Wolverton model
- Modified Schick-Wolverton model
- Shooman exponential model
- Weibull (Wagoner) Model
- Goel-Okumoto Bayesian model
- Littlewood-Verrall Bayesian model
- Shooman-Natarajan model
- Shooman-Trivedy Markov model
- Shooman Micromodel
- Littlewood Markov model
- Littlewood semi-Markov model
- Moranda a priori model
- Hecht estimation model
- MUSA model

Reliability Prediction Models

- Motley and Brooks model
- McCall, Richards, and Walters model
- Halstead model

The consensus of most software reliability experts is that it is necessary to estimate the number of flaws remaining in a software package to extrapolate or predict its reliability at a given point in the future.

Also evident is that the vast majority of experts in this field do not believe that software has a constant failure rate analagous to the failure rate of hardware represented by the bottom of the bathtub curve. The opinion is held that the software failure rate continues to decline asymptotically toward zero.

2.3.2 Hardware Reliability Factors

Hardware reliability prediction techniques have been developed and refined over the last 25 years. Current hardware techniques have been proven to be very effective as reliability design tools and for evaluating alternative design approaches against quantitative reliability requirements. The factors which impact hardware reliability are well known to reliability engineers.

Hardware reliability is directly dependent on many factors, but the major categories are:

- 1 ENVIRONMENT - Stresses imposed by the environment in which the hardware is to function
- 2 QUALITY - Measure of the relative goodness of a given part in comparison to similar parts
- 3 STRESS RATIO - Ratio of applied electrical stress to the rated stress (voltage for capacitors, power for resistors and transistors, and current for diodes)
- 4 TEMPERATURE - Ambient temperature that the part is exposed to while in an operational mode
- 5 TECHNOLOGY - Generic part type.

The most widely used vehicle for predicting electronic hardware reliability is MIL-HDBK-217D. Some attributes may affect a given part type considerably more than another part type. A separate failure rate model is supplied for each generic part-type to account for these differences. Table 2 illustrates the extensive characterization of the PRODUCT which is required to predict hardware reliability. In contrast, software reliability can be seen to be much more PROCESS dependent. Although software factors such as complexity, number of interfaces and size might indicate various degrees of inherent error proneness, it is the process which ultimately drives the number of errors which remain in software. Other than the stresses imposed by the input domain, software is obviously not subject to the degradation effects of environment and aging.

TABLE 2. Factors for Part Failure Rate Models
Discrete Semiconductors and Capacitors.

Factor	Description
<u>Common Factors</u> - Used in all or many part categories	
E	Environment - Accounts for influence of environmental factors related to application categories
Q	Quality - Accounts for effects of different quality levels
<u>Discrete Semiconductors</u>	
A	Application - Accounts for effect of application in terms of circuit function
R	Rating - Accounts for effect of maximum power or current rating
C	Complexity - Accounts for effect of multiple devices in a single package
S2	Voltage stress - Adjusts model for a second electrical stress (application voltage) in addition to wattage
F	Frequency and peak operating power factor
T	Temperature - Accounts for effects of temperature
M	Matching networks - Accounts for effects of type of matching networks

TABLE 2. Factors for Part Failure Rate Models
Discrete Semiconductors and Capacitors (Cont).

Factor	Description
<u>Capacitors</u>	
SR	Series resistance - Adjusts model for the effect of series resistance in circuit application of some electrolytic capacitors
CV	Capacitance value - Adjusts model for effect of capacitance related to case size
C	Construction factor - accounts for effects of hermetic and nonhermetic seals on CL and CLR capacitors
CF	Configuration factor - Accounts for effects of fixed and variable constructions on CG capacitors

2.3.3 Software Reliability Factors

The literature search revealed an intense interest within the industry to identify software characteristics which impact software reliability. In general, the factors identified during the literature search could be categorized according to their pertinence to a particular phase of the overall software life cycle. Specifically, the natural categorization suggested by the literature search was primarily chronological:

- o Software Product Development - These factors are introduced during the development phase. They are essentially design and code techniques.
- o Software Product Functional Performance - These factors measure the software's response to the requirements specification.
- o Software Initial Operation - These factors relate to the usability of the software product.
- o Software Product Revision - Most software is expanded or otherwise altered during its useful life. These factors influence the reliability of the modified product.
- o Software Transition - These factors address the characteristics of software that allow it to adjust to changing hardware environments.

The remainder of this section describes the software characteristics identified during the literature search. Although they are presented as categorized above, it should be noted that several re-categorizations were performed before the list could be used as factors within the methodology. First, the factors were re-categorized into inherent, design/development and application characteristics (see Section 3.0). Next they were prioritized in accordance with their relative influences on system reliability (Section 4.3.1). Finally, the most significant factors were re-categorized again (Section 4.3.2), this time to separate the inherent characteristics of the software PRODUCT from the error avoidance and error detection characteristics of the software development PROCESS.

Software Product Development

Software reliability will only be achieved if it is a requirement that is contractually specified and subsequently designed and coded into the software product during the development phase of the life cycle.

Virtually all sources studied during the literature search concluded that reliable software is achievable only by the application of a sound engineering approach. Although different authors favor different techniques and methods for developing reliable software, most converge on factors that:

- 1 Facilitate early recognition of problems
- 2 Increase the probability of fault detection
- 3 Simplify fault isolation
- 4 Isolate functional and logical activities.

The characteristics or factors most commonly addressed recognize and support the current state-of-the-art software development techniques. In the following list, complexity is the single most-important factor relating to software reliability, while all others are agreed-upon methods of reducing it:

- 1 Complexity - This is a negative factor that tends to make computer programs incomprehensible. It includes much more than other terms such as "readability" or "modularity." Complexity can be introduced into requirements, design, code, and even test activities. To avoid, detect, and correct faults prior to the operational phase when reliability will be needed, the software must be reviewed, discussed, inspected, and tested by human beings of limited capacity to recognize very complex interrelationships. Complexity seriously impedes and precludes these activities.
- 2 Top-down Functional Decomposition - In one way or another, most authors suggested functional decomposition or functional threading to minimize complexity.
- 3 Modularization - The segmentation of computer programs into single-purpose, single-entry, single-exit modules was the most popular technique discussed in the literature.
- 4 Hierarchical Design - Most authors acknowledged the significant reduction in complexity that can be achieved by enforcing a hierarchical structure of module segment calling and controlling relationships.
- 5 Structured Approaches - The advantages of structured techniques are well known. Their impact on software complexity is clear, but as of now, unquantifiable.

Software Product Functional Performance

The factors discussed in this section relate to the degree to which a software product meets its stated performance requirements. Results of the literature search indicate a consensus that the following software factors are significant:

- 1 Correctness - Ability of the computer program to accurately perform all of the functions required by the specifications.

- 2 Validity - Ability of the computer program to provide the performance, functions, and interfaces that are sufficient for beneficial application in the intended user environment. The distinction between this and the definition of "correctness" should be noted. Whereas correctness checks for accomplishment of the specified objectives, validity pertains to specifications as well as resulting software.
- 3 Generality - Ability of the computer program to perform its intended functions over a wide range of usage modes and inputs, even when a range is not directly specified as a requirement.
- 4 Testability - Characteristic of a computer program that allows its functional requirements to be logically separated to allow step-by-step testing of each aspect of the program.
- 5 Efficiency - Measure of the use of high-performance algorithms and conservative use of resources to minimize the cost of computer program operation.

Software Initial Operation

These factors influence the performance of the software during the operational phase. In this section, the phrase "initial operation" is used to isolate those characteristics of software that pertain only to the original computer program that was specified and developed. It does not include those aspects that relate to the software's ability to be maintained. Those aspects will be covered elsewhere. The following factors were identified in the literature as being pertinent to the initial operation of a computer program:

- 1 Usability - Characteristic of software that is indicative of its responsiveness to human factors considerations. It is a measure of how well it has used natural and convenient techniques for human operation.
- 2 Resilience - Sometimes referred to as ROBUSTNESS or in hardware terminology, SENSITIVITY; this is a measure of a computer program's ability to perform in a reasonable manner, despite violations of assumed usage and input conventions.
- 3 Fault Tolerance - Ability of the computer program to perform correctly, despite the presence of error conditions.

Software Product Revision

Software reliability is concerned with the usage of computer programs over some operational life in a specified operational environment. The existence of totally unmodified software in the operational environment is

a rarity if, in fact, any exist. The literature search revealed several pertinent factors:

- 1 Clarity - Ability of the computer program to be easily understood. It is a measure, not only of the computer program itself, but also of its supporting documentation.
- 2 Readability - Measure of how well a skilled programmer, not the original creator, can understand the program and correlate it to the original and new requirements.
- 3 Maintainability - Catch-all term used to summarize all the features of a computer program that allow it to be easily altered or expanded. It considers all of the other factors shown in this list.
- 4 Modifiability - Measurement that includes consideration of the extent to which likely candidates for change are isolated from the rest of a computer program. As an example, the isolation of input and output routines that are hardware- or human-dependent would increase the program's modifiability since these are very vulnerable areas to post-delivery modifications.
- 5 Flexibility - Measure of the computer program's design, which allows it to perform or to be easily modified to perform functions beyond the scope of its original requirements.

Software Transition

Software transition is the process of changing the environment of a software product either by installing it in another system, in another application, or in another software product. Similar to hardware, thoroughly tested software can be regarded as a standard part that can be incorporated into another application.

As software is better understood by nonsoftware personnel, it is becoming evident that many standard hardware techniques and methods can be applied to its usage. One idea that seems to be gaining in popularity is that of the development and use of general purpose software that has been identified as having high reliability. This idea is similar to the usage of Hi-Rel parts in the development of new hardware systems. Some of the software factors discovered during the literature search that seem to indicate the degree of software transition reliability are listed below:

- 1 Portability - Characteristic of computer software that allows it to be used in a computer environment different from the one for which it was originally designed. Use of standard high-level languages is one of the ways to increase portability.

- 2 Reusability - Measure of the extent to which a computer program can be used in a different application from the one for which it was developed.
- 3 Interoperability - Measure of the ease by which a computer program can be made to interface with other computer programs.

2.4 Conclusions

The literature search, while providing extensive background information on research already accomplished, revealed that no existing model can be directly applied to the problem of reliability prediction prior to development. It was concluded that the methodology required would have to be developed during this study and would require the usage of parameters available to the reliability engineer at the beginning of the development cycle.

It was also concluded that the natural (chronological) categorization evidenced by the literature search did not adequately separate software product characteristics from software development process characteristics. Many of the factors identified are, in fact, measures or qualities of the resulting software product, not determinants of its reliability.

3.0 CHARACTERIZATION TECHNIQUES FOR RELIABILITY PREDICTION

3.1 Overview and Objectives

Concurrent with the literature search discussed in Section 2.0, a preliminary methodology was developed. The objective of this phase of the study was to reevaluate the factors and characteristics investigated during the literature search and to characterize them in a manner consistent with the methodology.

3.2 Approach

The approach used during this phase was primarily analytical. First, the overall system and mission reliability characteristics were investigated to establish the rationale and assumptions of the model. Secondly, the software characteristics identified during the literature search were re-evaluated in light of the anticipated methodology which recognizes the significant relationship between the reliability of the software product and the process which creates it. The factors already presented in Section 2.3.3 were simultaneously purged, expanded, and re-categorized (Table 3) to facilitate their use in the model.

3.3 Results

Reliability prediction of total systems, which includes the effects of embedded software, must simultaneously consider the effects of both hardware and software. The analysis performed during this phase verified that, although they are closely related, hardware and software can be evaluated independently for purposes of reliability prediction. Furthermore, the analysis concluded that mission reliability prediction should be the primary goal of the methodology.

3.3.1 System Reliability

System reliability, as previously defined, is the probability that the required system will perform its intended functions for prescribed missions and time periods in the specified operating environment. Before describing the details of the approach, it is necessary to eliminate the ambiguities of the definition itself. Furthermore, it is necessary to constrain the problem definition to a more solvable and practical one.

Mission Reliability

First and foremost, it should be recognized that although a given system may have various operational missions, the motivation for performing a reliability prediction is to obtain a quantitative measure of the likelihood of success of a specific mission. Although the prediction and analysis of availability, maintainability, and supportability require consideration of all possible uses of a given system, the methodology presented herein is directed toward the prediction of system reliability for a

TABLE 3. Software Characteristics.

Operational Requirements

- Predominantly control
- Predominantly computational
- Predominantly input/output
- Predominantly real-time
- Predominantly interactive

Environmental Requirements

- Number of hardware interfaces
- Number of software interfaces
- Number of human interfaces

Size Considerations

- Number of functions performed
- Overall program size
- Number of compilation units
- Maximum size per unit

Complexity Considerations

- Number of entries and exits
- Number of control variables
- Use of single-function modules
- Number of modules
- Maximum module size
- Hierarchical control between modules
- Logical coupling between modules
- Data coupling between modules

Organizational Considerations

- Separate design and coding
- Independent test organization
- Independent quality assurance
- Independent configuration control
- Independent verification/validation
- Programming team structure
- Educational level of team members
- Experience level of team members

TABLE 3. Software Characteristics (Cont).

Methods Used

- Definition/enforcement of standards
- Use of High Order Language (HOL)
- Formal reviews (PDR, CDR, etc.)
- Frequent walkthroughs
- Top-down and structured approaches
- Unit development folders
- Software development library
- Formal change and error reporting
- Progress and status reporting

Design Approach

- Modular construction
- Structured design
- Structured code

Tools Used

- Flow charts
- Structure charts
- Decision tables
- HIPO charts

Documentation

- System requirements specification
- Software requirements specification
- Interface design specification
- Software design specification
- Source listings
- Test plans, procedures and reports
- S/W development plan
- S/W quality assurance plan
- S/W configuration management plan
- Requirements traceability matrix
- Version description document
- Software discrepancy reports

Duty Cycle

- Constant mission usage
- Periodic mission usage
- Infrequent mission usage

TABLE 3. Software Characteristics (Cont).

Environment

- Variability of hardware
- Training level of operators
- Variability of input data
- Variability of outputs
- Degree of human interaction

Non-operational Usage

- Training exercises
- Periodic self test
- Built-in diagnostics

Modification/Error Correction

- Performed in the field
- Performed at depot
- Performed at factory

Qualitative Characteristics

- Correctness
- Validity
- Generality
- Testability
- Efficiency/economy
- Resilience (Robustness)
- Usability
- Fault tolerance
- Clarity
- Readability
- Maintainability
- Modifiability
- Flexibility
- Portability
- Reusability
- Interoperability

specific mission. While the prediction of multimission system reliability is not specifically addressed, it should be noted that the determination of single-mission system reliabilities is the primary and most critical element of multimission predictions. Where such predictions are required, the method presented could be applied to each defined mission; these results are combined mathematically using classical statistical methods.

System Definition

Like most things in nature, knowledge of a system can be obtained through a detailed understanding of its composition. Specifically, knowledge of component characteristics that comprise the system and their interactions with each other is equivalent to knowledge of the system.

In the case of computerized systems, the initial decomposition consists of those components that may be categorized as hardware, software, or human. Of these, software is the least understood quantitatively. Hence, the methodology developed concentrates on the contributions of software to the overall reliability of the system. It is assumed that the prediction and measurement techniques currently being applied to hardware systems are adequate for the description of the hardware contributions to system reliability. Furthermore, it is assumed that human participation in the mission accomplishment is limited to controlling the hardware/software components by providing input data or reacting to outputs as necessary. As such, the human component can be considered as part of the operational environment, not part of the system. This does not impose a limitation on the methodology, but rather is a logical way of representing an automated system. Therefore, the methodology considers a system to be comprised of hardware and software components.

Interactions of System Components

Knowledge of the interaction of system components with one another is critical to the understanding of the system itself. There are two major categories of interaction that must be addressed. The first involves information exchange that provides necessary inputs to a component. The second involves behavior modification, whereby one component influences the manner in which another accomplishes its allocated requirements. The distinction can be made by considering that the first category allows one component to alter the environment of the other by presenting new conditions for it to operate upon, while the second category allows a component to actually alter the other component. This is purely an abstraction and does not represent the physical modification of system components, but instead, a logical modification of the system definition. For example, the exchange of commands and/or data between the hardware and software components of a system fall entirely in the first category, even when the exchange is entirely incorrect. The ability of a component to perform its intended functions despite erroneous inputs is usually referred to as its robustness, and is somewhat measurable. On the other hand, fault-tolerance implies that a fault has occurred and that the system will self-adjust to meet its

requirements. That is, the system is automatically altered to a new configuration.

Although considerable knowledge is known about active and dormant hardware redundancy and mathematical techniques have been devised for measuring its effects, this knowledge is not directly transferable to software components. Identical hardware components can be redundantly configured to increase reliability, but identical software components will react identically to the same environment. Redundant software must be different software, and the automatic replacement of a portion of software when it fails is tantamount to a redefinition of the system. The situation where the occurrence of a hardware failure causes a software compensation (or vice versa) adds a degree of complexity, which is currently beyond the state of the art of practical measurement. Extensive work is currently under way with respect to fault tolerance, both on a subcomponent as well as a system level. There are many respected system designers and analysts who feel that fault tolerance is the ultimate answer to system reliability problems. The methodology does not directly incorporate software fault tolerance considerations but does support such analyses. First, the Markov technique allows the analyst to incorporate redundant software into the overall model in the same manner as the other software components. The path probability of the redundant software is simply the probability that the primary software component will fail. Secondly, when the model is exercised, sensitivity information, in the form of relative utilization, is produced for each software component. The analyst can, therefore, use the model to identify candidates for application of fault tolerance techniques.

Conclusions Concerning System Reliability

An automated system can be described by two major subsystems, its hardware and its software, and that for a given system configuration, the interaction between these components is limited to information exchange. We can, therefore, analyze them independently by considering the hardware interface as part of the software's operating environment and vice versa. By restricting our analysis to mission-critical hardware and software, we can clearly see that for a particular mission, a combined hardware/software system can be represented by the serial configuration of a hardware and software component and that the system reliability is simply the product of the two component reliabilities as determined from their respective configurations and environments.

3.3.2 Hardware Subsystem Reliability

Both the literature search and the preceding analysis confirmed that hardware and software reliability prediction could be accomplished independently. Therefore, it was decided that the methodology would concentrate exclusively on software reliability prediction. System hardware reliability prediction is adequately accomplished via MIL-HDBK-217D.

3.3.3 Software Subsystem Reliability

Unlike hardware, software failures are not physical but rather logical in nature. Software does not degrade with age, nor does it degrade due to the physical stresses usually considered in hardware reliability prediction. The reliability of the software is the direct result of the process used to produce it. If the development process allows errors to be made and go undetected, the software product will eventually fail. During this phase of the study, the data collected during the literature search was expanded and re-categorized in order to isolate those characteristics which affected the quality, albeit reliability, of the development process. The overall intent was to eventually identify intrinsic or inherent factors which influence the difficulty of producing software and to identify development and design techniques which may be applied to increase the likelihood of avoiding and detecting errors.

Table 3 presented the expanded list of software factors which was created. The list was constructed specifically for use as a tool for further data collecting and as a format for the Delphi survey performed during the next phase of the study. Since one of the objectives of the survey was to rank the effects of individual factors, they were presented in subgroups as shown in the table. The subgroup titles are self-explanatory. They were chosen as potential categories from which to determine pi factors for use in the model. Although this categorization scheme was, once again, revised as a result of the first survey, it formed the basis for the final form used within the methodology.

Section 4.0 of this report discusses the survey in detail. The survey forms, instructions, and results are included in the appendices. Section 5.0 presents the prediction methodology, and the manner in which the software factors are incorporated.

3.4 Conclusions

The primary conclusion from this phase of the study was that software reliability prediction could be accomplished in a manner similar to classical hardware techniques. That is, the overall software reliability prediction can be accomplished by decomposing the software into its component parts, predicting the reliability of each component, and mathematically combining the individual predictions. It was also concluded that, whereas hardware reliability prediction is primarily dependent on PRODUCT characteristics and physical construction, software reliability prediction is heavily influenced by PROCESS characteristics and mission-dependent path probabilities. It was also determined that too many factors were being considered and that the pilot survey should be used to identify those factors most critical to the methodology.

4.0 TWO-PASS SURVEY

4.1 Overview and Objectives

As a result of the literature search and the subsequent characterization phase, an extensive list of software factors and attributes was compiled. The surveys were conducted to identify the most significant factors and to quantify their effects on reliability. The preferred approach was to quantify the factors based on historical software error data. It was found, however, that it was not possible to derive the required quantitative information from currently available data. Specialized data collection and experimentation efforts are required to fully quantify the factor effects.

4.2 Approach

The approach taken was to divide the survey into two phases. First, a pilot survey was accomplished to identify the factors most significant to reliability prediction. Second, a full-scale survey was conducted to quantify the effects of those factors. Since the phases of the survey were performed sequentially, the remainder of this section is outlined to separate and highlight the objectives, approaches and results of each of the surveys.

4.3 Results

4.3.1 Pilot Survey

Objectives

The literature search produced an abundance of characteristics and development techniques that relate to software reliability (Table 3). To learn more about these factors, a pilot survey was conducted. The objective of the survey was to obtain a qualitative ranking of the software factors, characteristics, and techniques in terms of their impact on software reliability. An implied objective was the elimination, for further consideration, of those factors perceived as having little or no impact.

Approach

A survey form was devised using the factors and attributes identified during the literature search. They were categorized into the groups already described in Section 3.0. The survey form was actually a matrix with the factors forming the rows of the matrix. Eight columns were included. The first was titled "System Reliability Impact," and the other seven were labeled to correspond to the major software error categories:

- Specification Errors
- Design Errors

- Coding Errors
- Software Interface Errors
- Hardware Interface Errors
- Human Interface Errors
- Capacity Problems.

Participants were asked to rate the degree of correlation between each of the software factors listed in the rows of the matrix and each impact area listed in the columns. The four possible ratings were H (high), M (medium), L (low), and O (zero) correlation. It is important to note that the participants were asked simply to indicate a correlation level and not a positive or negative relationship. The goal was to identify those factors that have a significant effect on reliability -- not to judge whether the effect was good or bad. The pilot survey form and its instructions are included in Appendix C.

Results

The survey was distributed to 30 hand-picked experts who were selected for their known interest and expertise in software development. Twenty-three responded. Some confusion resulted from the decision to design the form without provisions to distinguish positive and negative effects. Many participants were uneasy with assigning the same rating to a factor that they felt had a high positive influence on reliability and a factor that they felt had a high negative influence. Fortunately, since the distribution was very limited, it was possible to clarify the intent through personal contact.

The responses were recorded in a computerized data base and converted into numerical values of nine, five, one, and zero for high, medium, low and no correlation, respectively. Averages were computed for each factor, each group of factors, and each error category. They were then ranked numerically by averages within each category and by overall average.

Appendix D presents the results of the analysis performed. Of the factors themselves, interface and requirement specifications were rated as having the greatest impact on system reliability. Many of the participants remarked on the form that complete and unambiguous specifications have a strong, positive effect on reliability while incomplete or incorrect ones have an equally strong, negative effect. Frequent walkthroughs were rated as the most influential technique, followed closely by the definition and use of predefined standards and conventions. Curiously, the use of standards rated slightly higher than structured approaches and formal reviews. The inherent factor having the highest rating was real-time applications. Since it is generally felt that software required to operate in real time tends to be more complex than other applications, this result was not surprising.

The ranking of the results also indicated that many of the factors were not felt to be significant to reliability prediction. Most were

obviously unrelated but included in the survey for completeness and objectivity. Factors such as economy, readability, flexibility and most of the other -ilities were rated very low. Although each is significant, they are attributes of the finished software product, not determinants of its reliability. Some other lowly rated factors were not so obvious. The level of training and experience of the operational users was not considered significant. The educational level of the software developers was rated much lower than their experience level.

As a result of the pilot survey, many factors were eliminated from further analysis and those that remained were rearranged into the categories that would be utilized in the methodology. Specifically, it was noted that software factors could best be categorized into groups of inherent and developmental characteristics. The developmental characteristics, in turn, could best be described in relation to their ability to avoid errors or to detect (and correct) errors. This categorization was reinforced by the results of the pilot survey.

4.3.2 Full-Scale Survey

Objectives

The pilot survey revealed the characteristics of software that have a significant impact on reliability. It was necessary, however, to quantify these characteristics and factors in some way; therefore, a second, full-scale survey was conducted. The objectives of the full-scale survey were to obtain a quantitative ranking of the software factors and to determine values for the parameters in the methodology.

Approach

Using the results of the pilot survey, the survey form was redesigned to include only the significant software factors and to reflect the structure of the methodology being developed. The inherent characteristics and development techniques of software were combined with development phases and error types to produce the new questionnaire.

The survey was sent to approximately 350 persons who are involved with software and/or reliability. To avoid ambiguity, detailed instructions and definitions of all terms were included with the survey.

The full-scale survey form and its instructions, consisting of five pages, each different, are included in Appendix E. The first sheet listed several groups of inherent characteristics of software. Participants were asked to rank the groups in order of importance to system reliability. The individual characteristics were also ranked within the groups.

The same inherent characteristics were listed again on Sheets 2 and 3. There were four columns on Sheet 2, representing four phases of development. The entry for each row and column combination represented the relative

quantity of errors introduced by that particular factor during that particular development phase. The four columns on Sheet 3 represented four types of errors. The entries on that sheet indicated the percentage of each type of error present due to each characteristic.

The column headings on Sheets 4 and 5 were the same four error types from Sheet 3. Whereas Sheet 3 was used to quantify the distribution of error types caused by inherent characteristics, Sheet 4 listed development techniques and was used to quantify the effectiveness of each technique in avoiding each error type. Sheet 5 was similarly used to quantify the effectiveness of error detection mechanisms. The intent was to quantify the effectiveness of the software development process. This intent was deliberately hidden from the participants to avoid individual bias and pre-supposition. In fact, the participant was asked to treat each sheet independently from the others.

Unlike the pilot survey, this pass included a cover sheet to record the general educational and experience level of the respondent. In addition, the respondent was asked to identify his primary area of interest and general qualifications. The respondent was also invited to comment on the adequacy and relevancy of the survey itself.

Results

Approximately 100 responses were received. Each response was recorded in a computerized data base and combined statistically with all other responses. Analysis of the data included on the cover sheet indicated that the participants were highly qualified software professionals. Seventy-seven percent work in the software industry, 18 percent work for government agencies, and the rest are associated with universities. Seventy-three percent of those responding have at least 10 years experience in software development, and more than half have a graduate degree. The detailed profiles are presented in Appendix F as are all other results of the survey.

The rankings on Sheet 1 confirmed the pilot survey results that described the relative importance of the inherent characteristics of software. Modules that involve predominantly real-time implementation are ranked as most critical to software reliability. Control and interactive modules were ranked second in importance. The type of software seen as having the least effect on reliability is a predominantly computational application.

According to the survey, the number of interfaces is more critical to reliability than the type of interface (hardware, software, or human). The complexity of the operations performed by a module was deemed more important than the quantity of operations.

The most effective error avoidance mechanism, according to the survey is the rigid control of requirements and design specifications. Structured design approaches were likewise rated to be highly effective. Since the pilot survey had been used to eliminate less important factors, nearly all

avoidance mechanisms listed were rated as being effective. Surprisingly, the lowest ranked avoidance mechanisms involved configuration control.

The results indicated that frequent walkthroughs are the most effective means of detecting errors, followed closely by informal unit level testing. Quality audits were ranked considerably less effective. Most formal reviews were rated equally effective with the Critical Design Review being identified as having a slight edge on error detection capability. All of the testing procedures were rated about the same.

4.4 Conclusions

The surveys performed were invaluable to the development of the software reliability prediction methodology. The pilot survey served to effectively isolate those factors which truly affect reliability; the full-scale survey enabled a preliminary quantification of the effects of those factors.

Statistical analysis of the survey data provided initial approximations of the quantitative aspects of the software factors used in the prediction methodology. As was indicated earlier, the participants in the survey were highly qualified professionals. In the absence of precise and extensive measured data, their collective opinions were used as quantitative estimates of the effects of software factors on its reliability.

The results of the full-scale survey were, therefore, used as the basis for the calculation of inherent reliability and the developmental factors used within the methodology. For purposes of using the methodology, best, worst, and nominal values are presented for each factor. These values were statistically derived from the survey results.

5.0 RELIABILITY PREDICTION METHODOLOGY

5.1 Overview and Objectives

System reliability prediction is essential to planners and designers of military systems. Unfortunately, current methods of evaluation either ignore or simplistically incorporate the software contribution to system reliability. Those methods that do account for software use test results as a means of mathematically predicting its reliability. In the event that the software contribution causes system reliability to degrade below acceptable levels, testing must be continued to effect improvements.

The objective of this reliability prediction methodology is to provide a method for predicting system reliability, including both hardware and software effects in the beginning and throughout development. Proper use of the methodology will enable both the procuring agency and the developer to predict operational reliability at a time when it is still possible to affect it.

5.2 Approach

Hardware reliability prediction is adequately described by MIL-HDBK-217D and its supporting documentation; therefore, the methodology described here concentrates on the prediction of the system's software component. Essentially, the prediction of software module reliabilities is based on inherent factors of their operational mission and size. They are adjusted by developmental factors estimated from the design, development, and testing methods that will be employed during the development process. These component reliabilities are then mathematically combined via a Markov technique, which accounts for the duty cycling effects of logic paths that are being randomly exercised in accordance with functional mission requirements. After a single software reliability figure is obtained, the combined reliability is computed as though hardware and software are series components of the overall system.

5.3 Results

Performing a reliability analysis of a system prior to its development generally involves successive system decompositions. This continues until a component level is achieved, where reliability information is available, followed by mathematical reconstruction of the system to determine overall system reliability. Although software must be decomposed and evaluated differently from hardware, the methodology is similar. System decomposition is performed in accordance with currently used techniques. When the decomposition reaches a subsystem that requires embedded software, the software must be identified as an entity and treated as a subsystem component. While the rest of the hardware components are being evaluated via classical methods, the software portion can be evaluated using the methodology described here. Resulting information can then be used in the mathematical reconstruction of the system reliability prediction. This approach

allows the independent calculation of hardware and software subsystem reliabilities, with a minimal divergence from classical reliability prediction techniques. It also isolates the methodology used for software reliability prediction so that it can be easily modified, expanded, or replaced as new methods become available.

The remainder of this section describes those procedures that must be followed to predict hardware and software system reliability. The well-known and practiced methods of MIL-STD-785B are implemented for the entire system. However, software is considered a separate entity for which reliability must be specified, allocated, predicted and assessed. This is illustrated in Figure 2.

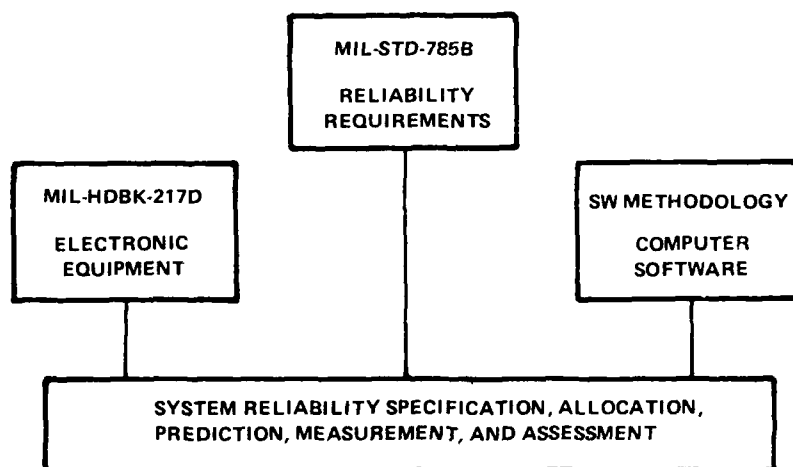


Figure 2. Classic Approach - New Ingredient

5.3.1 System Reliability

System reliability is the probability that the system will perform its intended function for the prescribed mission and time period in the specified operating environment. At the highest level of decomposition, it is necessary to define the specific mission that the system is required to perform. As a minimum, most systems can be considered to be either in an operating mode or a nonoperating mode. Many systems operate in a multimission or multiphased scenario. Since the ability of a system to perform its intended function is conditional, depending upon its availability at the time when the function is required, nonoperating failure mechanisms cannot

be ignored. Multiphased mission scenarios involve the requirement to perform functionally distinct operations in accordance with some recurring sequence and/or frequency. Conversely, multimission systems may successfully perform a given mission despite a failed state in a logically parallel mission mode. This latter example redefines the system and is not considered in this analysis.

Figure 3a depicts a single mission system in both operating and nonoperating modes. The ratio of time spent in each mode is a critical parameter of the overall system reliability analysis. For example, if the system described were an emergency power backup system, its nonoperating time would far exceed its operating time. Conversely, if it were a surveillance radar, its nonoperating time would probably be limited to the minimum required for periodic maintenance or refurbishment.

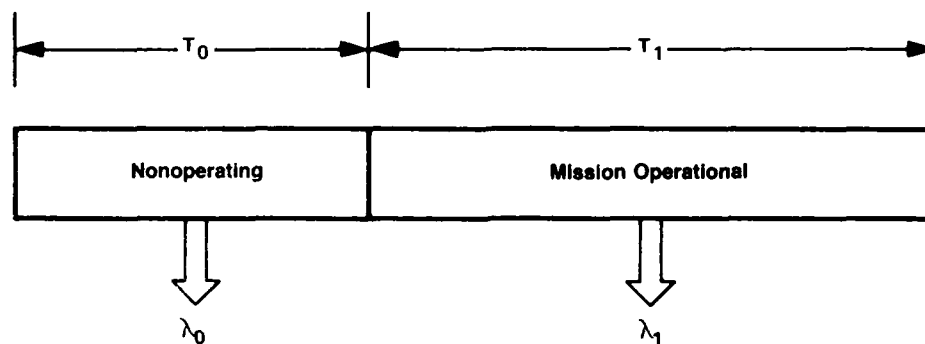


Figure 3a. Single Mission System

Figure 3b depicts a multiphased system that is either nonoperating or performing one of its required missions. Since all of the missions are required for overall system success, they must be logically serialized in reliability calculations.

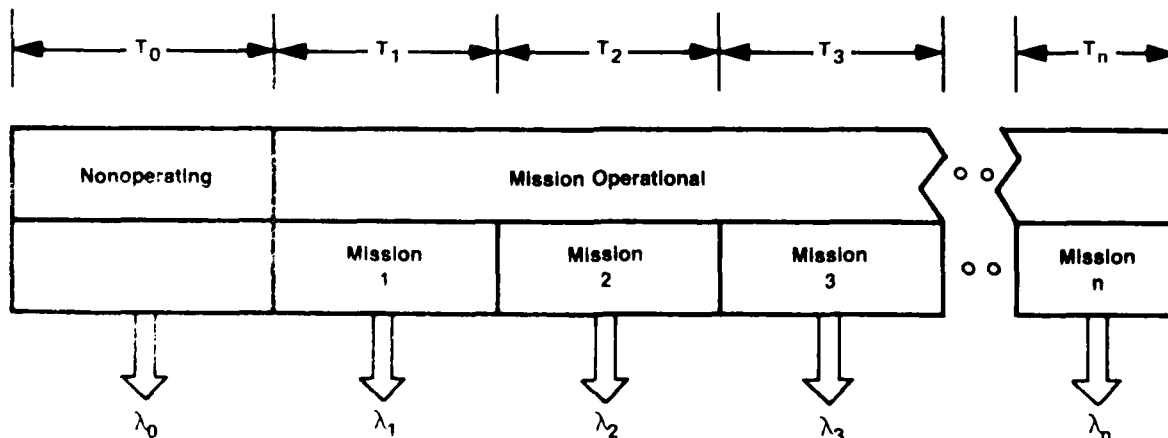


Figure 3b. Multi-phased System

In both examples, it is necessary to determine individual mode reliabilities and prorate their contribution to system reliability in accordance with the expected time to be allocated to each mode. Specifically:

$$\text{System Failure Rate} = \frac{\sum_{i=0}^{i=n} (\text{mission time } i) \times (\text{mission failure rate } i)}{\sum_{i=0}^{i=n} (\text{mission time } i)} \quad (1)$$

where:

i=0 for nonoperating mode

i=1 for mission 1

o

o

o

i=n for mission n.

5.3.2 Mission Reliability

From the previous discussion, it is clear that determining overall system reliability is critically dependent on a proper evaluation of mission reliability for each intended mission of the system. In fact, reliability requirements for many systems are specified in terms of specific mission reliabilities. This is particularly evident in military systems that have drastically different peacetime and wartime missions. Although the definition of system reliability does not change with the mission type, it is quite likely that the motivation for requiring reliability information does. Reliability information is required for tactical and strategic planning, and mission success probability is the primary measurement criterion. Reliability information is also required for logistics and maintenance planning. However, availability becomes the primary goal. In any case, mission reliability must be determined, whether it is to be a stand-alone prediction or an intermediate calculation needed to determine overall system reliability or operational readiness.

The need for a clear definition of specific mission functional requirements cannot be overemphasized. In the case of hardware reliability predictions, distinct missions could very likely have considerably different stress and environmental influences acting upon the components, thereby drastically affecting their individual reliabilities. Furthermore, the functional requirements of a particular mission might require the inclusion of components or subsystems not required in another mission. Based on the physical construction of the system, certain components might, therefore, be in a nonoperating state while others are operational. The resulting reliability block diagram could be considerably altered.

Hardware duty cycling effects are generally considered in reliability predictions at the highest levels only. Individual components within a circuit are usually considered to be all operational or all dormant. Since each component is essential to the continuity of the circuit, duty cycling effects are considered negligible and consequently ignored.

Duty cycling effects, however, are probably the most critical determinant of software reliability. Software cannot fail unless it is being used. A logic path does not exist during a time period (albeit microseconds) when it is not being used. The functional flow of control through a computer program is, in fact, its reliability block diagram. This represents the first and most critical deviation from classical reliability techniques, which is necessitated by the inclusion of software within a system and software reliability calculations within a reliability analysis. It follows that software reliability is a function of individual component reliabilities and their path probabilities. The method for determining each will be described later in this report.

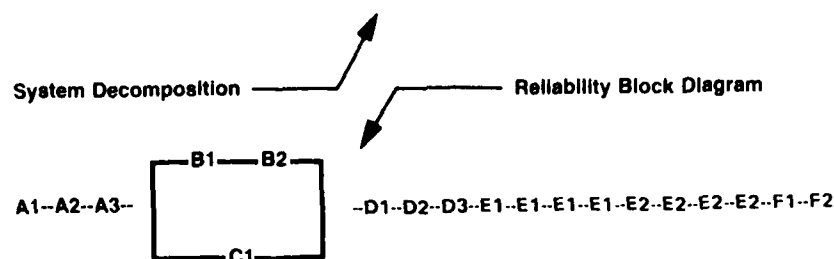
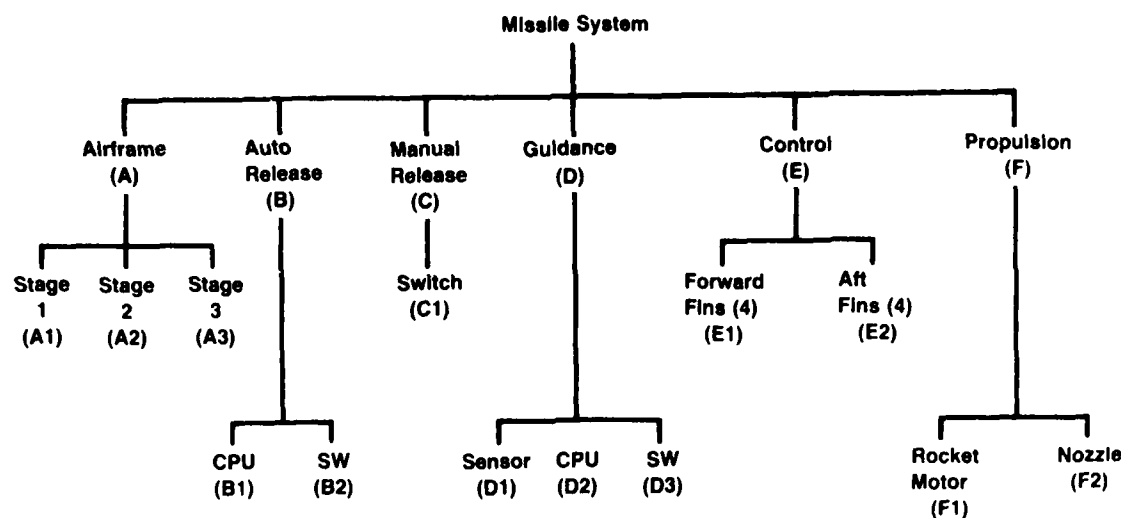
Mission decomposition is performed as described by classical reliability modeling techniques:

- 1 Define the subsystems required for mission success, and translate this into a mission success diagram.
- 2 For each subsystem, continue the decomposition process until a level is reached where a reliability prediction can be made.

After subsystem reliability predictions (including software) have been made, determination of mission reliability is the mathematical reverse of the decomposition procedure. Figure 4 depicts a typical mission reliability prediction analysis. It shows a missile system comprised of six major components, two of which contain embedded software. Although the figure only shows two levels of decomposition, it is obvious that the process can be continued as far as necessary to arrive at a level where subsystem or component reliability can be predicted.

Two things should be emphasized at this point. First, the presence of software in the reliability block diagram does not alter the normal techniques used for system decomposition, nor does it affect the mathematical reconstruction of the system reliability prediction. Only a technique for decomposing component parts and evaluating their individual and collective reliabilities is needed.

The second point is a major one. As shown later, the software reliability prediction methodology relies very heavily on the functional cycling of the individual software modules. It is, therefore, strongly mission dependent. To combine software and hardware reliability calculations, it is essential that the units of measurement be consistent. As shown later, the software model will produce reliability calculations for a specific mission and mission time. Unlike hardware considerations, which assume that the components have arrived at a level of constant failure rate, the software failure rate can be considered constant only for a prescribed mission scenario. Separate analysis must be performed for each mission considered in the analysis.



Missile Reliability

$$\begin{aligned}
 R &= R_a \times (R_b + R_c - R_b R_c) \times R_d \times R_e \times R_f \\
 \text{where: } R_a &= R_{a1} \times R_{a2} \times R_{a3} \\
 R_b &= R_{b1} \times R_{b2} \\
 R_c &= R_{c1} \\
 R_d &= R_{d1} \times R_{d2} \times R_{d3} \\
 R_e &= R_{e1} \times R_{e1} \times R_{e1} \times R_{e1} \times R_{e2} \times R_{e2} \times R_{e2} \times R_{e2} \\
 R_f &= R_{f1} \times R_{f2}
 \end{aligned}$$

Figure 4. Typical Mission Reliability Prediction Approach

5.3.3 Hardware Reliability

Hardware reliability prediction techniques are well established and generally well understood and practiced. The electronic ratings and reliability evaluation methods presented in MIL-HDBK-217D are assumed to be completely applicable to the hardware portion of the combined hardware/software methodology. Generally, hardware components have been categorized into device types, each of which has a descriptive model for determination of its reliability.

Each specific component in a device category has a base failure rate that represents its design characteristics. Multipliers, or pi factors are applied to the base rate as specified in the device model to arrive at a component failure rate adjusted by its developmental and operational characteristics (Figure 5).

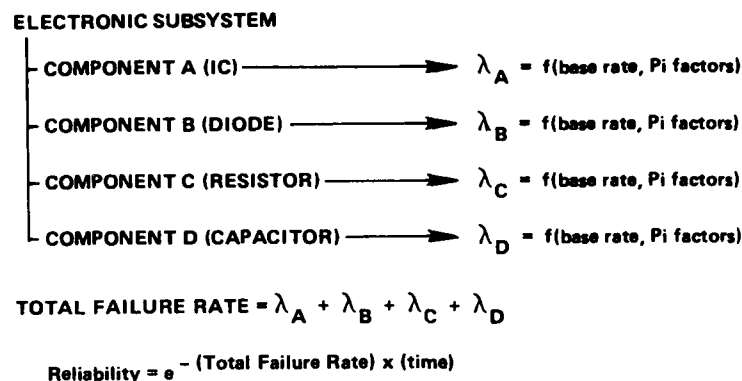


Figure 5. Hardware Reliability

5.3.4 Software Reliability

The methodology developed for predicting software reliability is organized to maximize the procedural similarities to hardware techniques. In hardware calculations, it is essential to identify the component parts, determine their respective reliabilities using base failures rates and system/application specific multipliers, and mathematically combine them in accordance with the reliability block diagram. The software methodology uses a parallel sequence of events: 1) the software subsystem is decomposed into its components, 2) for each component, a reliability prediction is calculated based on its inherent characteristics and developmental factors, and 3) the component reliabilities are mathematically recombined into a single prediction for the overall subsystem. Figure 6 summarizes the overall flow of the methodology. Using standard software engineering tools, the following sequence of events is accomplished.

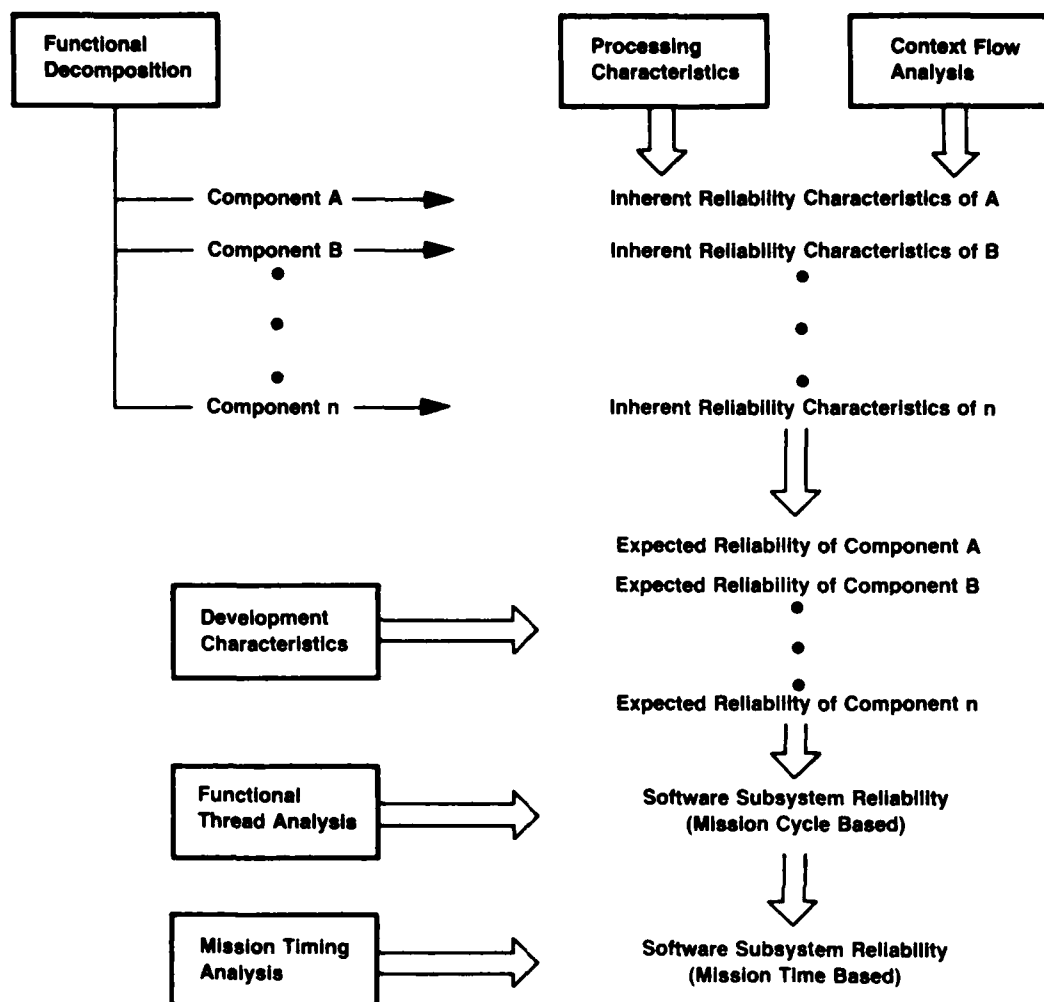


Figure 6. Software Reliability Prediction Methodology

- 1 A functional decomposition is performed to identify those physical components (programs, modules, routines, etc.) that comprise the software subsystem.
- 2 Analysis of the processing characteristics of each component and an analysis of the overall context (control and data information) flow between components is performed to identify their inherent characteristics. Each component is classified in accordance with its characteristics (e.g., real-time, single-function, extensive interfaces, etc.).
- 3 Characteristics of the intended development process are identified and categorized by the nature of their contributions to the process. As discussed earlier, all methods and techniques are

categorized as being either error avoidance mechanisms (e.g., use of a high order language, use of structured techniques, etc.) or error detection mechanisms (e.g., frequent walkthroughs, quality audits, etc.).

- 4 Individual software component reliabilities are computed.
- 5 A mission functional thread analysis is performed to determine the duty cycling effects caused by the specific mission profile for which reliability prediction is needed.
- 6 Overall software reliability for the specific mission is computed using a Markov technique.
- 7 Overall system reliability for the specific mission is computed as the product of hardware and software reliabilities.

All of the above data can be made available during the early stages of a software development. It is possible to perform all of the above analyses based on proposal or preliminary design information. Therefore, it is also possible to predict software reliability at a time when it is still possible to alter development plans to enhance it. The early application of the methodology also provides a basis for comparing the cost effectiveness of alternate approaches to increasing reliability. Of course, as the project progresses, more detailed design information and more precise timing estimates become available to the analyst. Periodic reapplication of the methodology can be used to give assurance that reliability requirements will be met or to alert management to avoid potential shortfalls. Actual measurements of component reliability can also be made during development, and the system reliability can be estimated prior to software integration.

The remainder of this section discusses the specific implementation of the methodology. This volume presents the rationale and derivation of the model. Volume II includes detailed examples as well as the list of factors derived from the surveys conducted.

Functional Decomposition

Decomposition of computer software is accomplished much like hardware. Whereas, hardware subsystems can be segmented wherever a connection has been or will be made, software can be broken anywhere in the sequence of commands that it executes. In both cases, however, it is illogical to disconnect components, except at the physical (or logical) boundaries of complete subunits. System hardware might be decomposed into black boxes; the boxes decomposed into printed circuit boards; the boards decomposed into circuits; and finally, the circuits decomposed into their respective electrical components. It is essential that every phase of the process yields complete subunits. Computer programs are similarly decomposed.

As one of the largest consumers of computer software, the Department of Defense has taken a leadership role in the development of software standards and methods. Figure 7 illustrates the generally accepted terminology associated with software decomposition. It lists the terminology

COMPUTER SOFTWARE CONFIGURATION ITEM (CSCI) - software aggregate which is designated by the procuring agency for configuration control

└--COMPUTER SOFTWARE COMPONENT (CSC) - a functional or logically distinct part of a computer software configuration item

└--UNIT - the lowest level logical entity specified in the detailed design which completely describes a non-divisible function in sufficient detail to allow implementing code to be produced and tested independently of other units

└--MODULE - the lowest physical entity specified in the detailed design which may be assembled or compiled alone

└--INSTRUCTION - a single line of code which may correspond to a single action of the computer or may be automatically translated into a series of single actions of the computer

└--OPERATION - the action to be performed by the computer

└--OPERANDS - the symbolic or absolute addresses of the computer memory where the data to be processed reside.

Figure 7. Software Decomposition Process and Terminology

specified in the proposed military standard DoD-STD-SDS and has been extended to the lowest possible level. At the highest level, software is defined as a configuration item, one of which is defined by and for the procuring agency. It has considerable contractual significance, but no logical or functional characteristics. At the other end of the spectrum, the level of detail is so specific that prediction is not possible until after implementation.

Although the software prediction methodology is not affected by the level to which the software is decomposed, it is practical to define its applicability as ranging from the CSC level through the module level. Generally, CSC level decomposition is possible during the requirements definition phase of software development; unit decomposition is possible during preliminary design; and module decomposition is possible during the detailed design phase. At each milestone, the software reliability prediction methodology can be reapplied with greater accuracy. For generality, the term "software component" is used to include all levels of software decomposition between the CSC and module level.

Inherent Reliability Characteristics

Just as hardware components can be classified into component categories such as resistors, capacitors, and diodes, software can be categorized into characteristic groups. In the case of software, however, the distinction between groups is based on logical composition rather than physical makeup. A direct relationship between the complexity of a computer program and its reliability is intuitively expected. Likewise, it is intuitive that the complexity of the software is related to its intended application (the programmatic complexity of the design is considered later). In other words, even before it is designed or implemented, real time software is expected to be more error prone than batch software where timing is not a critical consideration. Similarly, historical evidence shows that programs with a large amount of interface requirements experience higher failure rates than those that contain minimal interfaces.

To determine the inherent reliability characteristic of a given module, the analyst must identify those operational characteristics that best describe the module to be developed. Error distribution estimates for the more predominant operational characteristics of software modules have been derived from the surveys described earlier. At present, they accurately represent the combined opinions of software experts. Hopefully, the recently increased emphasis on software reliability and quality measurement will promote and facilitate the collection of detailed statistical information that is similar to that information for the hardware reliability engineer in MIL-HDBK-217D. Currently, the methodology is more urgently needed than the data precision.

It should be noted that this phase of analysis addresses the inherent error characteristics of the various components which must be combined with the planned development process characteristics to determine module reliability. The calculation of the module characteristic error distribution

is simply an average of the effects of its inherent characteristics. If we define $C(j)$ as the percentage of errors of type j to be expected in the module being evaluated, it follows that:

$$C(j) = \sum_{i=1}^{i=N} \frac{c(j,i)}{N} \quad (2)$$

where: $c(j,i)$ is the percentage of errors of type j caused by inherent characteristic i , (listed in Volume II)

and N is the number of inherent characteristics applicable.

Development Characteristics

As described earlier, software development characteristics can be categorized in terms of their contributions to error avoidance or error detection. Virtually any activity during the development life cycle can be evaluated in terms of these two characteristics. The software reliability prediction methodology uses measures of error avoidance and error detection effectiveness which are based on the planned technical and managerial techniques and methods used to develop the software. As in the case of inherent reliability characteristics, the data base used for these predictions was constructed from the results of the survey performed during the study.

It is generally accepted that certain development techniques are good and will make the software better. For example, it is generally agreed that structured approaches are good and will have a positive influence on the quality and reliability of the product. Quantification of the effects is typically attempted after the development is complete, and the results are rarely applicable to new projects. While the approach described herein has a limitation due to the unavailability of detailed historical records, the method is directly applicable to any software development venture.

Error avoidance effectiveness is calculated as the probability of not introducing an error given the opportunity for making the error. Mathematically, it is computed as unity minus the probability that a hypothetical error will not be avoided by any of the techniques employed. If we define $A(j)$ as the probability of avoiding errors of type j , it follows that:

$$A(j) = 1.00 - \prod_{i=1}^{i=N} (1.00 - a(j,i)) \quad (3)$$

where: $a(j,i)$ is the probability of error type j being avoided by the application of technique i , (listed in Volume II)

and N is the number of techniques employed.

Error detection effectiveness is similarly calculated as the probability that an existing error will be discovered and corrected. Again, it is mathematically computed as unity minus the probability that a hypothetical error will not be detected by any of the techniques employed. If we define $D(j)$ as the effectiveness of detecting errors of type j , it follows that:

$$D(j) = 1.00 - \prod_{i=1}^{i=N} (1.00 - d(j,i)) \quad (4)$$

where: $d(j,i)$ is the probability of error type j being detected by the application of technique i , (listed in Volume II)

and N is the number of techniques employed.

Expected Component Reliability

The expected reliability of a software component is a function of the inherent characteristics of the component and the characteristics of the development process used to produce it. Unfortunately, there is insufficient historical data available to isolate, with any degree of confidence, the casual relationship between a specific characteristic or development technique and the resulting effect on component reliability. The term "reliability" is used here to connote the probability that the software component will perform its intended functions correctly the next time it is executed. That is, component reliability is defined as the probability of success in a single trial. If it were possible to extensively exercise the component in a controlled experiment, its reliability could be approximated by the ratio of its successful executions to its total executions:

$$R_c = P_r (\text{success}) = \frac{\text{Number of Successful Executions}}{\text{Number of Executions}} \quad (5)$$

Unfortunately, the component does not exist at the beginning of the development effort, so it is not possible to determine the success probability as a simple relative frequency of measured events. The logical recourse is to theoretically derive the probability function by computing the ratio of all possible successful executions to the number of possible executions. It should be noted that the number of possible executions is extremely large since every variation of the inputs to the component and every variation of the computer memory creates a new combination of circumstances in which the component must operate.

We can redefine the probability of success in terms of the number of variations as:

$$R_c = P_r (\text{success}) = \frac{\text{Number of Possible Successful Variations}}{\text{Total Number of Possible Variations}} \quad (6)$$

In the above expression, the number of variations is an extremely large number representing all combinations of inputs, memory states and functional requirements to be accomplished.

Although the component itself does not yet exist, its functional requirements do. Likewise, the process or mechanism exists to convert those requirements into a software component. The software engineer is required to create logic which will correctly perform every possible variation. Those that he properly implements will successfully execute; those that he fails to implement correctly will not. The ratio can be expressed as:

$$R_c = P_r (\text{success}) = \frac{\text{Number of Variations Correctly Implemented}}{\text{Total Number of Variations Implemented}}. \quad (7)$$

This is obviously a measure of the probability of successful implementation well as the probability of successful execution.

AT SOME LEVEL OF DECOMPOSITION, THE RELIABILITY OF THE PRODUCT IS EQUAL TO THE RELIABILITY OF THE PROCESS WHICH PRODUCED IT.

It follows that developmental techniques which increase the reliability of the process will also increase the reliability of the product by reducing its error content. That is, component reliability can be predicted based on an inherent probability of successful implementation enhanced by the application of the error avoidance and detection techniques discussed in the Development Characteristics section. Specifically, software component reliability can be calculated as:

$$R_c = R_i + E(1-R_i) \quad (8)$$

where: R_c is the component reliability (probability of success)

R_i is the inherent reliability of either the process or the component

and E is the enhancement factor achieved by the application of error avoidance and detection techniques.

a. Inherent Component Reliability - R_i

There are several approaches to determine inherent software component reliability, each of which has both advantages and disadvantages. Any measure which presents a ratio of successful implementations to total implementations may be used. Some of the more obvious measures are discussed below:

- 1 Assume that R_i is equal to zero. This causes equation (8) to reduce simply to the enhancement factor. At first glance, this approach appears to be a gross simplification. It essentially says that unless an effort is made to avoid and/or detect errors, the software component will not work. We feel that this is the most theoretically sound approach. However, it assumes that the developer has absolutely no knowledge of the product he is responsible to develop. Even a casual knowledge of what he's supposed to do can be considered an error avoidance technique. This assumption carries with it the additional assumption that the checklist of avoidance and detection techniques is exhaustive. It does, however, define a lower bound on the reliability prediction when the developmental characteristics are known.
- 2 Assume that the inherent UNreliability is proportional to measured fault densities of existing software which has similar characteristics. This approach has the advantage of data availability. Although there is a fairly wide range of measured values of faults per line of code, there is sufficient historical data for an analyst to make a sound engineering determination of the best figure to use. Care must be taken, however, to distinguish how and when the data used was collected. Many organizations do not begin counting faults until the software is tested in the overall system while others begin recording failure data as soon as individual components have completed unit test. The prediction methodology assumes that R_i includes consideration of all errors made, not just the ones recorded subsequent to integration testing. This method should produce an upper bound on the reliability prediction due to the fact that the actual number of faults in a software product cannot be less than the number recorded.
- 3 Assume that the inherent UNreliability is proportional to a fault density which has been interpolated from the range of historically recorded fault densities. The interpolation could be based on the same characteristics already discussed in the Inherent Reliability Characteristics section. Although such a scheme has not yet been formulated, it is the opinion of the author that one could be created and that it would provide the most unbiased measure.

b. Reliability Enhancement Factor - E

Figure 8 illustrates the relationship between inherent reliability, avoidance effectiveness and detection effectiveness. The figure introduces some terminology not previously described:

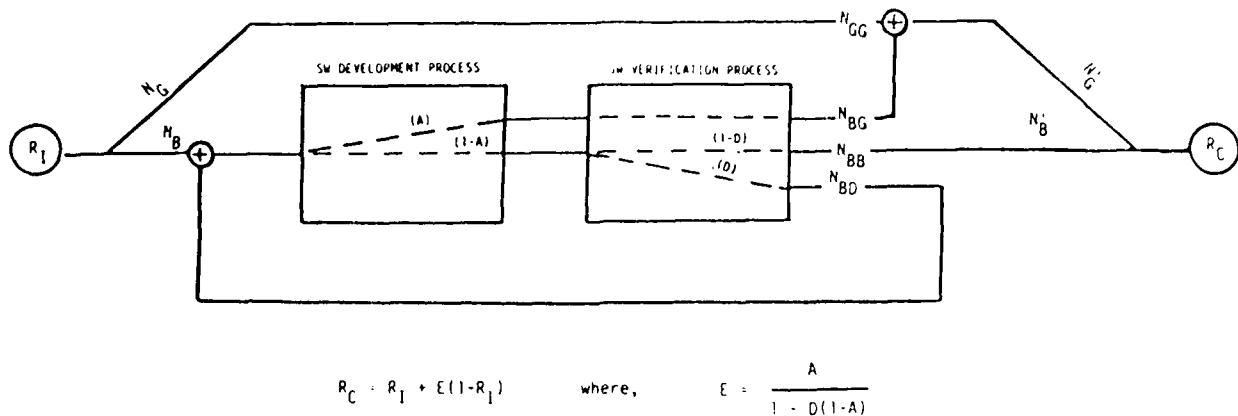


Figure 8. Relationship of R(I), R(C), A and D

- R_i Inherent reliability.
- N Total possible variations implemented (reference equation (6)).
- NG Total variations inherently implemented correctly. These are the variations that would have been properly implemented without process enhancement.
- NB Total variations inherently implemented incorrectly. These are candidates to be avoided or detected.
- I_i Number of variations being worked on during the i 'th iteration. These include the original errors to be eliminated plus reworks of errors discovered on the previous iteration.
- NGG Number of variations which "pass thru" the avoidance/detection filters because they are already correctly implemented.
- NBG_i Number of previously incorrect implementations which were successfully avoided on the current iteration.
- NBB_i Number of previously incorrect implementations which were neither avoided nor detected on the current iteration.
- NBD_i Number of previously incorrect implementations which were successfully detected and returned for rework.
- A, D These are the error avoidance and detection factors described in the Development Characteristics section.

The process depicted represents a typical software development operation. As a result of the inherent characteristics of the software to be developed, errors will be made. The development team will attempt to avoid making those errors by the application of software engineering techniques. Recognizing that they will probably not avoid all errors, tests and other detection techniques are implemented to locate and rework the faults. Avoided errors will exit the process as corrected implemented variations. Detected errors will be reworked by the process until they either are avoided or escape the detection mechanisms. Eventually, all N variations exit the process. Since the enhancement factor is an improvement factor, it is defined as:

$$E = \frac{NBG}{NBG + NBB} = \frac{\text{Number of Corrected Bad Implementations}}{\text{Total Number of Bad Implementations}}. \quad (9)$$

The derivation that follows, verifies that the enhancement factor is independent of the initial number of bad implementations and is simply a function of the process characteristics:

$$E = \frac{A}{1 - D(1-A)} \quad (10)$$

where: A and D are the error avoidance and detection factors described in Section 5.3.4.

(1) Preliminary Calculations

The following relationships can be determined directly from Figure 8. The number of initial inputs to the process is equal to the initial number of errors expected due to inherent characteristics. On subsequent iterations, the number of inputs is equal to the number of reworks necessitated by the previous iteration:

$$I_0 = NB \quad \text{and} \quad I_i = NBD_{i-1}. \quad (11)$$

The number of avoided errors on any iteration is equal to the number of inputs processed multiplied by the probability of avoiding errors:

$$NBG_i = I_i(A). \quad (12)$$

The number of errors detected on any iteration is equal to the number of errors NOT avoided multiplied by the probability of detecting them:

$$NBD_i = I_i(1-A)(D). \quad (13)$$

The number of errors not detected on any iteration is equal to the number of errors NOT avoided multiplied by the probability of NOT detecting them:

$$NBB_i = I_i(1-A)(1-D). \quad (14)$$

Combining equations (11) and (13) yields the feedback relationship which causes the process to continue until all of the original inputs have exited the process:

$$I_i = I_0 [D(1-A)]^i. \quad (15)$$

(2) Derivation

The components of equation (9) can now be expressed in terms of the initial inputs to the process and the avoidance/detection mechanisms used during the process.

The number of corrected and uncorrected bad implementations which exit the process are equal to the infinite sum of the number which exit on individual iterations. That is:

$$NBG = \sum_{i=0}^{\infty} NBG_i \text{ and } NBB = \sum_{i=0}^{\infty} NBB_i. \quad (16)$$

Substitution of equations (13) and (14) into (15) yields:

$$NBG = \sum_{i=0}^{\infty} I_i (A) \text{ and } NBB = \sum_{i=0}^{\infty} I_i (1-A)(1-D). \quad (17)$$

Since the multipliers are independent of i , they can be factored out of the summations:

$$NBG = (A) \sum_{i=0}^{\infty} I_i \text{ and } NBB = (1-A)(1-D) \sum_{i=0}^{\infty} I_i. \quad (18)$$

It follows that equation (9) becomes:

$$E = \frac{(A) \sum_{i=0}^{\infty} I_i}{(A) \sum_{i=0}^{\infty} I_i + (1-A)(1-D) \sum_{i=0}^{\infty} I_i}. \quad (19)$$

The summations cancel resulting in:

$$E = \frac{A}{A + (1-A)(1-D)} \quad (20)$$

or:

$$E = \frac{A}{1 - D(1-A)} \quad (21)$$

Path Analysis

After the reliability of each software component has been estimated, it is possible to predict overall software subsystem reliability. This is accomplished by using the Markov process as suggested by Cheung [36]. The approach is based on the fact that individual software components contribute to the overall reliability when, and only when, they are executed. It has already been shown that individual component reliabilities can be predicted. Fortunately, it is also possible to predict their operational usage. As was mentioned earlier, preliminary design activities not only identify the required software functional components, but also show their relationships to each other. This may be expressed in the form of functional flow diagrams, decision tables, hierarchical structure charts, or in the case of interrupt driven system, timing and frequency requirements for each component.

The flow of control between software program components can be considered a Markov process if we assume that component reliabilities are independent. If a given software program has n components, it is necessary to know the reliability of each component and the probability of going from one component to another. The component reliabilities are in the diagonal matrix R , and the path probabilities are in the matrix P ; i.e.,

$$R = \begin{bmatrix} R_1 & 0 & 0 & . & . & . & 0 \\ 0 & R_2 & 0 & . & . & . & 0 \\ 0 & 0 & R_3 & . & . & . & 0 \\ . & . & . & . & . & . & . \\ 0 & 0 & 0 & . & . & . & R_n \end{bmatrix} \quad P = \begin{bmatrix} P_{11} & P_{12} & . & . & . & P_{1n} \\ P_{21} & P_{22} & . & . & . & P_{2n} \\ P_{31} & P_{32} & . & . & . & P_{3n} \\ . & . & . & . & . & . \\ P_{n1} & P_{n2} & . & . & . & P_{nn} \end{bmatrix} \quad (22)$$

where R_i is the reliability of component i , and P_{ij} is the probability that control is passed from component i to component j .

The matrix Q is the product of matrices R and P . The ij 'th entry represents the joint probability that component i will execute correctly (R_i) AND pass control to component j (P_{ij}).

$$Q = \begin{bmatrix} R_1 * P_{11} & R_1 * P_{12} & . & . & . & R_1 * P_{1n} \\ R_2 * P_{21} & R_2 * P_{22} & . & . & . & R_2 * P_{2n} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ R_n * P_{n1} & R_n * P_{n2} & . & . & . & R_n * P_{nn} \end{bmatrix} \quad (23)$$

By considering each component to be a state and by defining two additional states, C for correct program termination and F for failed termination of the program, a Markov chain can be constructed with $n+2$ states. The transition matrix T is formed by adding two rows and two columns to the matrix Q . An additional two rows and columns are for the states C and F . The matrix T is defined as follows:

$$T = \begin{array}{c} \begin{matrix} & 1 & 2 & . & . & . & n & C & F \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ . \\ n \\ C \\ F \end{matrix} \begin{bmatrix} R_1 * P_{11} & R_1 * P_{12} & . & . & . & R_1 * P_{1n} & 0 & 1-R_1 \\ R_2 * P_{21} & R_2 * P_{22} & . & . & . & R_2 * P_{2n} & 0 & 1-R_2 \\ . & . & . & . & . & . & . & . \\ R_n * P_{n1} & R_n * P_{n2} & . & . & . & R_n * P_{nn} & R_n & 1-R_n \\ 0 & 0 & . & . & . & 0 & 1 & 0 \\ 0 & 0 & . & . & . & 0 & 0 & 1 \end{bmatrix} \end{array} \quad (24)$$

The ij 'th entry of T is the probability of going from state i to state j in one step. The ij 'th entry of T^2 is the probability of going from state i to state j in two steps. The ij 'th entry of T^3 is the probability of going from i to j in three steps. The reliability of the software is the probability of going from state 1 to state C in x steps or less, as x approaches infinity. Thus, to compute this reliability, calculate

$$\sum_{i=1}^{i=x} T^i, \text{ as } x \text{ approaches infinity.}$$

There is a simpler way to compute the reliability of a program using the matrix Q . Let

$$S = I + Q + Q^2 + Q^3 + Q^4 + \dots \quad (25)$$

where I is the identity matrix. Note that

$$\begin{aligned} (I-Q) * (I + Q + Q^2 + Q^3 + Q^4 + \dots) &= I + Q + Q^2 + Q^3 + \dots \\ &\quad - Q - Q^2 - Q^3 - \dots \quad (26) \\ &= I \end{aligned}$$

and so,

$$I + Q + Q^2 + Q^3 + Q^4 + \dots = (I - Q)^{-1}. \quad (27)$$

It follows that

$$S = (I - Q)^{-1}. \quad (28)$$

Letting S_{1n} be the entry in the first row and n 'th column of S , the reliability of the program for a single cycle, R_c , is given by

$$R_c = S_{1n} \times R_n. \quad (29)$$

Several important points should be made here. First, since the calculation of software reliability was computed on operational path probabilities, the prediction made is applicable only for the scenario or specific mission described by these probabilities. In the likely event that the system being analyzed has a variety of missions (e.g., peacetime, standby, war-time, etc.), the reliability of each must be independently calculated by adjusting the path probabilities matrix and reaccomplishing the Markov analysis. A second point is equally significant, but tends to simplify the overall prediction. The numerical value of the software reliability, is independent of time. It was computed as the function of a specific mission or operational scenario and can be assumed to be constant for that mission. The final step in predicting a combined hardware/software system reliability value is calculated by multiplying the hardware reliability value, determined by classical methods and the software reliability value, calculated by the methods described herein.

5.4 Conclusions

The methodology described in this report was developed to allow the independent calculation of subsystem reliabilities for hardware and software components of an overall system and the combination of these calculations into a single reliability prediction. For the hardware subsystem, the classical methods of MIL-HDBK-217D are used without modification. For

the software subsystem, a new approach is developed, which combines software knowledge with influences caused by the development methodologies employed. Knowledge, or inherent reliability, is based on historical reliability measurements of software developed for similar applications. The influence or pi factors are determined by the manner in which the software will be or is being developed.

This approach allows the system planners to predict system reliability before development begins. This is accomplished by identifying the inherent reliabilities of software components, combining them into a single prediction of software reliability, and incorporating this prediction into the system reliability model. Development methodologies can then be chosen to increase overall reliability, and cost tradeoffs can be performed to determine the most cost-effective means of achieving a required reliability. In the event that desired system reliability cannot be achieved within budgetary constraints, information obtained from the methodology will be available to support a go or no-go decision at a time prior to extensive investments in the development.

By using this methodology in combination with other reliability measurement techniques, the original estimates of inherent component reliabilities can be refined as those components become testable. The method can, therefore, be used at various stages of the development process to periodically reevaluate the system reliability predictions and to measure effectiveness of the development methods employed.

In addition to being recursive and reiterative, the method is designed for expandability and flexibility. There are no constraints on the number or type of pi factors that may be incorporated into a particular prediction. As the state of the art of software development advances, it will be an easy task to incorporate the effects of additional technical and managerial approaches.

Many software development experts foresee a future environment of off-the-shelf or standard component software products where a developer will create a software subsystem by logically combining existing (high reliability) software packages. If this environment is realized, the inherent component reliabilities essential to the method presented here will be readily available and highly accurate. Since these components of the software subsystem will be developed, they will not be influenced by developing pi factors and will be easily inserted into the calculations.

Many individual factors that are used by the reliability prediction methodology will require refinement as the state of the art advances. The method itself, however, is sound, practical, flexible, and will form the basis for meaningful prediction, measurement, and estimation of operational system reliability.

6.0 DATA COLLECTION

6.1 Overview and Objective

The objective of this phase of the effort was to collect and analyze data from existing fielded systems and compare the results measured with those that would have been predicted by the methodology.

6.2 Approach

The approach taken was to initiate a broad spectrum data collection effort at the beginning of the study; to determine what data was needed as the methodology developed; and finally, to calculate from the data those parameters that directly relate to the inputs and predictions produced by the method.

Data collection involved both Martin Marietta sources as well as external, public domain data bases. Within Orlando Aerospace, data were collected from three major projects: two missile systems, and one command and control application. Requests were sent to our sister divisions in Baltimore and Denver for data on two more large-scale Defense projects. Three major data bases were purchased from the Data and Analysis Center for Software (DACS).

6.3 Results

With one exception, the data collection effort was very disappointing. The data available was either incomplete or inconclusive. Originally, it was thought that sufficient data could be gathered within our own organization. In fact, the best source located was data gathered from our Assault Breaker project. But even the Assault Breaker data is insufficient for validating the methodology due to the lack of detailed operational data. In Volume II of this report, the Assault Breaker program is explained in detail, and the methodology is demonstrated on real data. Unfortunately, Assault Breaker performance is too good. Although the methodology predicts a very high reliability, the software has performed even better. At best, the demonstration establishes credibility of the prediction technique.

Essentially, the data collected was unusable for one or more of the reasons discussed in the following paragraphs.

6.3.1 Failure Rate

When the study was originally proposed, it was anticipated that a critical parameter of the methodology would be some sort of software failure rate. So many models have been developed based on the removal of errors over time, it was anticipated that our methodology would have a similar dependency. Chronological failure histories of some of our internal projects might provide such information. For this reason, such data was collected early in the study and the DACS Software Reliability Dataset, compiled by John Musa, was purchased. As the methodology evolved, it became clear that software reliability is more a function of the missions that it performs than the amount of time it runs successfully between failures. This is not meant to imply that such information is not meaningful.

Reliability measurement and estimation are essential to evaluation of operational systems. The methodology developed as part of this study, however, is oriented toward prediction at a time when no failure histories are available. Data needed to adequately validate the methodology would consist of repetitious executions of a given mission scenario within a variety of input domains. When a failure is recorded, the fault could not be repaired. We are not aware that such an experiment has been accomplished.

6.3.2 Latent Defects

Many individuals and organizations still do not accept the notion of software reliability. Because software does not exhibit the physical characteristics of hardware devices, many professionals devoutly subscribe to the deterministic view of software; it is either 100 percent or zero percent reliable. In reality, this is a true assessment. However, it is also reality that software failures occur statistically in a manner similar to hardware failures. Despite the similarity, software faults are not typically recorded as failures for reliability measurement purposes. They are treated as latent defects, fixed, and forgotten.

6.3.3 Fault Assessment

Most systems that have been operational long enough to have established reliability data available were fielded before software was recognized as a separable entity of the system. In many cases, software requirements are specified as computer requirements, and faults uncovered during operation are assessed against the computer, not against the software. Investigation into specific problem descriptions associated with operational systems revealed such problem descriptions as: "the computer went down" and "the computer miscalculated the coordinates." The failures were either discounted when the computer was reinitialized or were improperly charged against the computer, just as though a resistor had failed. A proper evaluation is possible only if the original problem reports are available. It is virtually impossible to use data base information or summaries to determine software performance separate from computer performance.

6.3.4 Enhancements

When hardware fails, something that had been working stops. When software fails, the fault was present all along. When hardware fails, it is fixed by returning it to its previous working state. When software fails, it is returned to a better state. By this rationale, many software changes are not recorded as fault corrections, but rather as enhancements. Again, detailed analysis of individual change notices reveals which changes were, in fact, enhancements and which ones were not. However, review of large scale data bases is inconclusive.

6.3.5 Developer versus User

When a company such as Martin Marietta develops and delivers a defense system, their data collection terminates. Although extensive records are maintained prior to delivery, very little data is returned after delivery. Data bases available from developers, therefore, terminate at the point

where the system became operational. Even though data concerning the inherent factors of the software are available and the error avoidance and detection mechanisms used during development are known, the resulting performance data are not. Conversely, those who are charged with maintaining a fielded system have performance data, but they do not have the developmental information needed to validate a prediction methodology. A particularly frustrating experience is to have an abundance of operational data and an abundance of developmental data for different systems.

6.3.6 Military versus Commercial

The most extensive software performance data bases available were compiled for commercial systems. Literally, thousands of software failures have been recorded against millions of hours of operation. Reliability of a military system must be measured against its operational mission. As defined earlier, system software reliability is the probability of performing an intended mission without causing system outage or failure. To record a software failure would, therefore, require a system outage or failure. For the type of software developed for weapon systems, an outage and a failure are the same. A missile cannot be reinitialized while in flight. Whereas, most commercial applications can continue to be used with an acceptable failure rate, most military systems require a very high probability of working correctly every time. Defense systems data bases, therefore, tend to contain extensive data on mission scenarios rather than to meet their primary purpose (e.g., data on training exercises rather than on actual flights).

6.4 Conclusions and Recommendations

Although the data collected during this study were inconclusive as a method validation tool, analysis indicated some significant shortcomings about the way we handle software data throughout its life cycle. The recommendations in the following paragraphs are offered.

6.4.1 Record All Software Problems

We must instill in data collectors and recorders the importance of properly identifying problem sources. In most cases, the personnel charged with operating a system are not qualified to make a determination of the cause of a problem. When a system defect is recorded, it must eventually be charged against some aspect of the system. When a hardware failure cannot be replicated, the failure is recorded as being unverified or transient. This is a logical and well-accepted practice. However, when software fails, a fault exists. When the system is reinitialized or rerun, we are, in most cases, changing the environment (or input domain) in which the software was operating when it failed. Sooner or later, the problem will reoccur. If it is to be corrected, it is essential that the state of the system during the problem be recorded as accurately as possible. If the system's state can be re-established, the fault will remanifest itself. All software errors and suspected software errors must be documented, even if a simple restart makes it seem to disappear.

6.4.2 Identify the Source

When a software fault is isolated, it is very important for analysis to reveal the exact cause of the problem. There is no such thing as an enhancement resulting from a system failure. Software is not just the code that's embedded in the computer. Software is also the design logic that resulted in the code, and it is the requirement that resulted in the design. If it is decided to enhance the software to extend its capabilities, then the original capabilities were not properly stated and the software had a requirements deficiency. Virtually everyone in the software industry acknowledges the fact that incomplete, ambiguous, and constantly changing requirements are the heaviest contributors to software costs and performance problems. However, we deny ourselves the data needed to correct, or at least minimize, the impact by labeling requirement and design changes as enhancements.

6.4.3 Distinguish Hardware From Software

The physical portion of a computer is a hardware component of a system. The logical operation of that computer is a software component of the system. In most cases, operations personnel cannot distinguish between a computer hardware failure and a computer software failure. The symptoms in many cases are the same. Recent emphasis on automatic fault isolation and built-in-test is encouraging. As mentioned earlier, it is essential that software faults be identified as such.

6.4.4 Consistent Data Collection

Procuring agencies interface with both the developer of a system and the eventual user of the system. They alone can establish and influence a meaningful data collection effort on both sides of the delivery milestone. The developer must be encouraged or paid to record significant facts concerning the development process. Software problem reports generated during development are typically devised and used by the developer. Since the procuring agency is not generally concerned with resolved problems, these reports are not usually identified as deliverable data items. Likewise, the techniques used by the contractor during development are usually not documented in a uniform or consistent fashion. They are hardly ever documented in a deliverable data item. When the system is fielded, the operational user begins collecting performance data without the benefit of knowing how the software was developed. When he does identify a problem, he may be able to fix it. However, he cannot determine why it happened or what could have been done to avoid it.

The Software Technology for Adaptable, Reliable Systems (STARS) program has recently generated a series of Data Item Descriptions that provide a uniform and consistent format for recording software facts during and subsequent to development. Although, usage of the forms devised will increase the cost of projects using them, the data that will be collected should prove invaluable to the development and evaluation of new software methods and techniques.

7.0 CONCLUSIONS AND RECOMMENDATIONS

The methodology derived during this effort should be applicable to virtually any operational system which employs critical, embedded computers and software. The method is both mathematically sound and computationally feasible at any level of detail required. Like all reliability models, the accuracy of the predictions computed is highly dependent on the parameters used during the calculations. The factors used in hardware reliability predictions have been derived and measured over an extensive period of time. Until recently, very little quantitative information about software factors has been available. The true value of the methodology will not be totally realized until more precise measurement of software factors and characteristics is possible.

With the increased dependency of military systems on embedded software, the industry is becoming increasingly more interested in understanding, measuring and controlling the software development process and product. The trend toward high-level languages, particularly Ada, the trend toward standard computer architectures, and the enforcement of software design and development standards will facilitate the derivation and use of additional measures of software factors which can be used directly by the methodology.

The SAIC study [103] mentioned earlier will be concluded soon. The measures defined therein appear to be both meaningful and compatible with the methodology developed during this effort. An effort is needed to directly validate the feasibility, accuracy and cost effectiveness of both the SAIC measures and the Martin Marietta methodology. It is strongly recommended that consideration be given to the definition of a follow-on effort to validate the study results on an actual military system development contract. The effort should be accomplished concurrently with the system being developed. Data concerning the development process itself should be captured and used in the prediction methodology. The SAIC measurements should be accomplished at various stages during the system development and detected faults should be recorded and analyzed when they happen, not after it is too late to determine their root causes. Such an effort would produce a wealth of data not currently available and would form an intelligence base usable by Air Force planners and civilian contractors to determine the most cost effective means of developing high reliability systems.

The method developed during this effort, much like the corresponding hardware methodology described in MIL-HDBK-217D, is applied by defining the individual parts (modules) which make up the system and by defining the interactions between them. Although the method can be applied manually, it lends itself quite well to computerization. Furthermore, one of its intended uses is as a tool for performing tradeoff analyses of various design approaches and techniques. It is suggested that consideration be given to the construction of a user-friendly, conversational and flexible computer program which would lead a reliability engineer through the method. Such a program could be designed around the current method using current parameters and factors. It could be designed for expansion and/or modification as the techniques and metrics evolve. Furthermore, it could be extremely valuable as a tool to begin the accumulation and refinement of statistical data bases needed for accurate software reliability measurement and prediction.

The Ada programming language will very shortly be the source language for all DoD-embedded, mission-critical software. It would be most appropriate to initiate data collection concerning the effects of Ada on software and system reliability. A study of the language itself and an expansion of the combined system reliability prediction methodology to accommodate it would be a valuable asset to Air Force System planners and project officers as they begin and carry out the transition to Ada.

APPENDIX A

GLOSSARY

ACCEPTANCE CRITERIA -- The criteria a software product must meet to successfully complete a test phase or meet delivery requirements.

ACCEPTANCE TESTING -- Formal testing conducted to determine whether a system satisfies its acceptance criteria and to enable the customer to determine whether to accept the system. See also QUALIFICATION TESTING.

AUTOMATED DESIGN TOOL -- A software tool which aids in the synthesis, analysis, modeling or documentation of a software design. Examples include simulators, analytic aids, design representation processors and documentation generators.

AVAILABILITY -- The probability that computer software is capable of functioning in accordance with requirements at any time. This probability is often measured with respect to total need time.

BATCH PROCESSING -- A technique by which items are coded and collected into groups for processing.

CDR -- See CRITICAL DESIGN REVIEW.

CERTIFICATION -- The process of confirming that a system is operationally effective and capable of satisfying mission requirements under realistic operating conditions.

CHANGE REQUEST -- See SOFTWARE CHANGE REQUEST.

CLARITY -- The ability of the computer program to be easily understood. It is a measure not only of the computer program itself, but also of its supporting documentation.

COMPATIBLE HARDWARE/SOFTWARE PREDICTION MODEL -- Suitable interpretation of hardware and software mathematical relationships for combined computations so as to make feasible prediction of the System Reliability.

COMPLEXITY -- The degree of complication of a system or system component, determined by such factors as the number and intricacy of interfaces, the number and intricacy of conditional branches, the degree of nesting, the types of data structures, and other system characteristics.

CONFIGURATION CONTROL -- The systematic evaluation, coordination, approval or disapproval, and implementation of all approved changes in the configuration of a configuration item after formal establishment of its approved technical documentation.

CONFIGURATION CONTROL BOARD -- The authority responsible for evaluating and approving or disapproving proposed engineering changes, and ensuring implementation of the approved changes.

CONFIGURATION ITEM -- An aggregation of hardware/computer software, or any of its discrete portions, that satisfy an end-use function and are designated by the Government for configuration management.

CONFIGURATION MANAGEMENT -- A discipline applying technical and administrative direction and surveillance to identify and document the functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specification and other related contract requirements.

CONTROL VARIABLES -- Dynamic program data which affects or controls processes within other modules or subprograms.

CORRECTNESS -- The ability of the computer program to perform exactly and correctly all of the functions required by the specifications.

COUPLING -- See DATA COUPLING and LOGICAL COUPLING

CRITICAL DESIGN REVIEW (CDR) -- A formal technical design review conducted to ensure that the detailed design satisfies the requirements correctly and completely. It is conducted after completion of the detailed design but prior to coding. It establishes the design baseline.

DATA COUPLING -- An inter-relationship between or among program modules in which data items are shared without formal parameter passing.

DECISION TABLE -- A table of all conditions that are to be considered in the description of a problem, together with the actions to be taken. The two are linked by "decision rules" which tie each combination of conditions with a corresponding combination of actions.

DESIGN FACTORS -- Factors which can be characterized as reliability design tools or methodologies (e.g., top-down design, modularity, structured programming, etc.).

DETAILED DESIGN SPECIFICATION -- This document provides complete programming design sufficiently detailed for a programmer to code from with minimal additional direction.

DEVELOPMENT FACTORS -- Factors which can be characterized as being part of the software reliability engineering development process (e.g., test-debug-fix, use of developmental aids or standards, quality assurance measures, etc.).

DUTY CYCLE -- A measure of the need time of a computer program or portion of the program with respect to total system time.

ECONOMY -- See EFFICIENCY.

EFFICIENCY -- A measure of the use of high-performance algorithms and conservation of use of resources to minimize the cost of computer operation. Sometimes referred to as **ECONOMY**.

EMBEDDED SOFTWARE -- An interactive assembly of computer programs and computer data that is integral to a major system whose primary function is not data processing.

ENGINEERING CHANGE NOTICE -- A document used to process changes to baseline documents and which includes both notice of an engineering change to a configuration item and the supporting documentation by which the change is described.

FAILURE -- See **SOFTWARE FAILURE**.

FAULT -- A software defect that causes program operation to fail to perform program requirements.

FAULT AVOIDANCE -- The act of eliminating the mechanisms which cause erroneous software to be created and/or the application of mechanisms which encourage or support correct software creation. It relates to the elimination of errors before they occur.

FAULT CORRECTION -- The act of removing, avoiding or otherwise negating the effects of a detected fault. Can be accomplished automatically by the software or by alteration of the software.

FAULT DETECTION -- The recognition of the presence of a software defect either by its external manifestations or by an inspection of the software itself. Implied in the definition is the ability to locate and/or isolate the defect itself.

FAULT TOLERANCE -- The ability of the computer program to perform correctly despite the presence of error conditions.

FCA -- See **FUNCTIONAL CONFIGURATION AUDIT**.

FLEXIBILITY -- A measure of the extent to which the computer program's design allows it to perform or to be easily modified to perform functions beyond the scope of its original requirements.

FLOW CHART -- A symbolic representation of the functional flow and an abbreviated description of the inputs, processing, outputs and flow of control of a computer program or portion of the program.

FORMAL QUALIFICATION TESTING (FQT) -- Testing conducted prior to Functional Configuration Audit to demonstrate CPCI compliance with all applicable software specifications.

FQT -- See **FORMAL QUALIFICATION TESTING**

FUNCTIONAL CONFIGURATION AUDIT (FCA) -- Audit to verify that the actual performance of the configuration items complies with the B-5 development specifications.

FUNCTIONAL DECOMPOSITION -- A method of designing a system by breaking it down into its components in such a way that the components correspond directly to system functions and subfunctions.

FUNCTIONAL DESIGN SPECIFICATION -- This document establishes the functional design of the software at the computer program level; provides sufficient design information to accomplish the goals of the Preliminary Design Review.

GENERALITY -- The ability of the computer program to perform its intended functions over a wide range of usage modes and inputs, even when not directly specified as a requirement.

HARDWARE RELIABILITY -- The probability that the required hardware in a system will operate failure free in a specified environment for the prescribed missions and time periods.

HIERARCHICAL CONTROL -- A sequence of control which consists of multiple levels of decomposition, general to specific.

HIERARCHICAL DESIGN -- A design which consists of multiple levels of decomposition, general to specific.

HIERARCHICAL INPUT-PROCESSING-OUTPUT (HIPO) CHARTS -- A document which consists of diagrams illustrating the functional flow of inputs, processing and outputs on multiple levels of decomposition, general to specific.

HIGH ORDER LANGUAGE -- A programming language which provides compression of computer instructions such that one program statement represents many machine language instructions. It is nonproblem specific and is used by programmers to communicate with the computer.

HIPO CHARTS -- See HIERARCHICAL INPUT-PROCESSING-OUTPUT CHARTS

INDEPENDENT VERIFICATION AND VALIDATION (IV&V) -- Verification and Validation of a software product is performed by an organization that is both technically and managerially separate from the organization responsible for developing the product.

INTERFACE DESIGN SPECIFICATION -- This is an optional document which is required whenever the system contains two or more computers that must communicate with each other. It provides a detailed logical description of all data units, messages, control signals and communication conventions between the digital processors.

INTEROPERABILITY -- A measure of the ease by which a computer program can be made to interface with other computer programs.

INTRINSIC FACTORS -- Factors which can be termed as inherent characteristics or attributes of the software (e.g., language, complexity, size, etc.).

IV&V -- See INDEPENDENT VERIFICATION AND VALIDATION.

LOGICAL COUPLING -- This is the relationship which exists between program modules due to the passage of control variables from one module to the other. Whereas data variables provide parameters to and from a module, control variables affect the logical operation of the module itself. See also DATA COUPLING.

MAINTAINABILITY -- A characteristic of design and installation which is expressed as the probability that an item will be retained in or restored to a specified condition within a given period of time, when maintenance is performed in accordance with prescribed procedures and resources; i.e., a measure of the extent to which the computer program can be easily altered or expanded to satisfy new requirements or to correct deficiencies.

MODIFIABILITY -- The characteristics of being easy to modify; one aspect of maintainability. This implies controlled change in which some parts or aspects remain the same while others are altered in such a way that a desired new result is obtained. This measurement includes consideration of the extent to which likely candidates for change are isolated from the rest of the computer program.

MODULAR CONSTRUCTION -- An organization of the functions of the computer program into a set of discrete program modules.

MODULAR DECOMPOSITION -- See FUNCTIONAL DECOMPOSITION.

MODULARITY -- The extent to which the computer program is segmented into single-purpose, single-entry, single-exit modules.

MODULE -- A discrete identifiable set of computer instructions usually handled as a unit by assembler, compiler, linkage editor, loading routine, or other type of routine or subroutine. It is frequently defined as the lowest stand alone, testable set of instructions.

MODULE COUPLING -- See DATA COUPLING and LOGICAL COUPLING

OPERATING SYSTEM -- Software that controls the execution of programs. An operating system may provide services such as resources allocation scheduling, input/output control, and data management. Although operating systems are predominantly software, partial or complete hardware implementations are possible. An operating system provides support in a single spot rather than forcing each program to be concerned with controlling hardware. See also SYSTEM SOFTWARE.

OPERATIONAL FACTORS -- Factors which can be derived and characterized from system requirement and specification documents (e.g., operational/mission scenarios, inputs-outputs functions, performance criteria, etc.).

PCA -- See PRELIMINARY CONFIGURATION AUDIT.

PDL -- See PROGRAM DESIGN LANGUAGE.

PDR -- See PRELIMINARY DESIGN REVIEW.

PHYSICAL CONFIGURATION AUDIT (PCA) -- A formal examination of the as-built version of a configuration item against its technical documentation to ensure the adequacy, completeness, and accuracy of the technical design documentation.

PQT -- See PRELIMINARY QUALIFICATION TESTING.

PORTABILITY -- The characteristic of computer software which allows it to be used in a computer environment different from the one for which it was originally designed.

PRELIMINARY DESIGN REVIEW (PDR) -- A formal technical review of the basic design approach. It is held after the completion of preliminary design efforts but prior to the start of detailed design. See also SYSTEM DESIGN REVIEW and CRITICAL DESIGN REVIEW.

PRELIMINARY QUALIFICATION TESTING (PQT) -- An incremental process which provides visibility and control of the computer program development during the time period between the Critical Design Review (CDR) and Formal Qualification Testing (FQT); conducted for those functions critical to the CPCI.

PROGRAM -- All the software that can physically interrelate as an entity. It is also the name given to the highest level function in a hierarchical design.

PROGRAM DESIGN LANGUAGE (PDL) -- A language with special constructs and, sometimes, verification protocols used to develop, analyze, and document a design.

PROGRAM SPECIFICATION LANGUAGE (PSL) -- A language used to specify the design, requirements, behavior, or other characteristics of a system or system component.

PROGRAM SIZE -- A measurement of size usually expressed as the number of lines of code, the number of computer instructions or the number of bytes of code.

PROGRAM SUPPORT ENVIRONMENT -- An integrated collection of tools accessed via a single command language to provide programming support capabilities throughout the software life cycle. The environment typically contains tools for designing, editing, compiling, loading, testing, configuration management and project management.

PSEUDO CODE -- A combination of programming language and natural language used for computer program design.

PSL -- See PROGRAM SPECIFICATION LANGUAGE.

QUALIFICATION TESTING -- Formal testing, usually conducted by the developer for the customer, to demonstrate that the software meets its specified requirements. See also ACCEPTANCE TESTING, PRELIMINARY QUALIFICATION TESTING, AND FORMAL QUALIFICATION TESTING.

QUALITY ASSURANCE -- A planned and systematic pattern of all actions necessary to provide adequate confidence that the item or product conforms to established technical requirements.

READABILITY -- A measure of how well a skilled programmer who was not the original creator of the computer program can understand the program and correlate it to the original and to new requirements.

REAL-TIME PROCESSING -- The processing of information or data in a manner sufficiently rapid that the results of the processing are available in time to influence the process being monitored or controlled.

REPAIRABILITY -- The extent to which a change to correct a deficiency can be localized, so as to have minimal influence on other program modules, logic paths or documentation.

REQUIREMENTS SPECIFICATION -- A specification that sets forth the requirements for a system or a system component; for example, a software configuration item. Typically included are functional requirements, performance requirements, interface requirements, design requirements and development standards.

REQUIREMENTS TRACEABILITY MATRIX -- A set of tables that provides traceability of software requirements from the system specification to the individual item requirements specifications, to the design specification which implements the requirements, and to the software plans and procedures that verify that requirements have been fully implemented.

RESILIENCE -- A measure of the computer program's ability to perform in a reasonable manner despite violations of the assumed usage and input conventions. Also referred to as ROBUSTNESS.

REUSABILITY -- A measure of the extent to which the computer program can be used in an application different from the one for which it was developed.

REVIEWS -- See specific entries, e.g., SDR, PDR, CDR.

ROBUSTNESS -- See RESILIENCE.

SDR -- See SYSTEM DESIGN REVIEW.

SELF-TEST CAPABILITY -- The extent to which the computer program can be easily and thoroughly tested by internal procedures.

SOFTWARE CHANGE REQUEST — A document used to describe and process proposed changes to baseline software and its associated documentation.

SOFTWARE CONFIGURATION MANAGEMENT PLAN -- This document describes the contractor's organization for configuration management, the procedures that will be used to implement tailored contractual requirements, and the persons/groups responsible for each particular phase of configuration management.

SOFTWARE CORRECTIVE MODIFICATION -- The periodic updating of the software to preclude system failure when processing potential data sets.

SOFTWARE DESIGN SPECIFICATION — This document describes the assignment of each of the software requirements to a specific functional software module, the functional interface for each module, the data base utilized by each module, and the design implementation which has been built into the operational software.

SOFTWARE DEVELOPMENT LIBRARY -- The libraries, library procedures and automated aids used to maintain control of the software baseline by providing a consistent, systematic and orderly method for organizing, maintaining and controlling a project's computer program elements and documentation during the development phase.

SOFTWARE DEVELOPMENT PLAN — This document presents the comprehensive plan for the project's software development activities by describing the software development organization, the software design and testing approach, the programs and documentation that will be produced, software milestones and schedules, and the allocation of development resources.

SOFTWARE FAILURE -- The inability to perform an intended logical operation in the presence of the specified data/environment due to a fault in the software.

SOFTWARE MAINTAINABILITY MODEL -- A mathematical model that may be derived from prior experience in correcting software faults that predicts frequency of faults of various categories, and may include suitable parameters to accommodate results of timeline analysis of software corrective and preventative maintenance, determination of mean-time-to-restore (MTTR) as well as maximum restore time for the required percentile of the timeline data, and determination of optimum performance of software corrective and preventative modification tasks, including frequency and duration.

SOFTWARE PROBLEM REPORT — A report of a program deficiency identified during software qualification, test, system integration and test, or system operation, which appears to be software related.

SOFTWARE QUALITY ASSURANCE PLAN — This document is produced during the software planning phase and describes the procedures that will be used to implement Software Quality Assurance control, and the persons/groups responsible for each phase of Software Quality Assurance.

SOFTWARE RELIABILITY -- The probability that the required software of a system will perform its intended functions for the prescribed missions and time periods in the specified operating environment without causing system outage or failure.

SOFTWARE RELIABILITY PREDICTION MODEL -- A mathematical model that could include appropriate parameters such as code complexity, branching numerics, structured/modular format utilization, execution rate, timing restrictions, and data complexity, predictability and variability, as may be verified by test data.

SOFTWARE REQUIREMENTS REVIEW (SRR) -- A review to achieve formal agreement between the customer and the developer that the software requirements specifications are complete and accurate.

SOFTWARE REQUIREMENTS SPECIFICATION -- This document establishes the requirements for the performance, design, test and qualification of the computer program. See also REQUIREMENTS SPECIFICATION.

SOFTWARE SUPPORT LIBRARY -- A software library containing computer readable and human readable information relevant to a software development effort.

SOURCE LISTING -- A document that displays tabulated data identifying the sequential appearance of instructions as they appear on the computer program media.

SRR -- See SOFTWARE REQUIREMENTS REVIEW.

SPECIFICATION CHANGE NOTICE -- A formal notification of a change in the specification.

STEPWISE REFINEMENT -- The process whereby steps are taken in the following order: (1) the total concept is formulated, (2) the functional specification is designed, (3) the functional specification is refined at each intermediate step where intermediate steps include code or processes required by the previous step, and (4) final refinements are made to completely define the problem.

STRUCTURE CHART -- A design and documentation technique used in structured programming to show the purpose and relationships of the various modules in a program.

STRUCTURED APPROACH -- An approach to software design which consists of using stepwise refinement to formulate and define a problem. See also STRUCTURED DESIGN.

STRUCTURED CODE -- Code that has been generated with a limited number of well-defined control structures using stepwise refinement. See also STRUCTURED PROGRAMMING.

STRUCTURED DESIGN -- A disciplined approach to software design which adheres to a specified set of rules based on principles such as top-down design, stepwise refinement and data flow analysis.

STRUCTURED PROGRAMMING -- A computer program constructed of a basic set of control structures, each one having one entry point and one exit. The set of control structures typically include: sequence of two or more instructions, conditional selection of one of two or more instructions or sequence of instructions, and repetition of an instruction or a sequence of instructions.

SUBPROGRAM -- A computer program that can be part of another program.

SYSTEM AVAILABILITY -- The probability or proportion of operational time that the hardware and software is in the required operable and committable state at a time when the mission is required with a specified data environment.

SYSTEM CAPABILITY -- The probability that the hardware and software can achieve the required mission objectives given the operational conditions, including data environment, during the mission.

SYSTEM DEPENDABILITY -- The probability that the hardware and software will perform successfully during one or more required sequences of a mission, given the hardware and software status at the start of the mission (availability).

SYSTEM DESIGN REVIEW (SDR) -- A formal review conducted to evaluate the optimization, traceability, correlation, completeness and risk of the allocation of system requirements among system components including software.

SYSTEM EFFECTIVENESS -- The measure of the degree to which the hardware and software achieve the mission requirements in the operational environment as evidenced in system availability, dependability and capability.

SYSTEM EFFECTIVENESS MODEL -- A mathematical model encompassing both hardware and software for a prior prediction, a pre-operational test evaluation or an operational demonstration of the deliverable system effectiveness. The model should encompass the foregoing defined parameters and include a practical means of computation and analysis. Implementation of the model is generally demonstrated with data from other programs or data assumed from requirements, prior to application in the current program.

SYSTEM INTEGRATION TESTING -- The process of testing an integrated hardware and software system to verify that the system meets its specified requirements.

SYSTEM REQUIREMENTS SPECIFICATION -- This document states the technical and mission requirements for a system as an entity, allocates requirements to functional areas, and defines the interfaces between or among the functional areas.

TEST ORGANIZATION -- A group responsible for preparing test plans and procedures, executing the test procedures, and analyzing the test results in order to verify that the system performed its intended functions. This group is also responsible for documenting problems detected during testing and verifying by retest that corrections to the software work properly.

TEST PLANS AND PROCEDURES -- Documents which set forth how the system or configuration item will be qualified, describing on the system level how it will be demonstrated that each performance requirement stated in the System Functional Requirements Specification has been met. Similarly at the configuration item level, these documents describe the method of qualifying a configuration item against functional requirements stated in its Software Functional Requirements Specification.

TEST READINESS REVIEW (TRR) -- A review conducted prior to each test to ensure adequacy of the documentation and to establish a configuration baseline.

TESTABILITY -- The extent to which a software product assists in the establishment of verification criteria and supports evaluation of its performance.

TOP-DOWN APPROACH -- An approach which identifies major functions to be accomplished, then proceeds from there to an identification of the lesser functions that derive from the major ones.

TOP-DOWN DESIGN -- An ordering to the sequence of decisions which are made in the decomposition of a software system by beginning with a simple description of the entire process (top level). Through a succession of refinements of what has been defined at each level, lower levels are specified.

TRR -- See TEST READINESS REVIEW

UNIT DEVELOPMENT FOLDER -- This is a software development notebook that is used to collect and organize software requirements and products for a unit of software as they are produced. It contains all requirements documentation, flow charts, discrepancy reports and test results.

UNIT LEVEL TESTING -- Testing to verify program unit logic, computational adequacy, data handling capability, interfaces and design extremes, and to execute and verify every branch.

USABILITY -- The characteristic of software which is indicative of its responsiveness to human factors considerations. It is a measure of how well the software has utilized natural and convenient techniques for human operation.

VALIDATION -- The evaluation, integration and test activities carried out at the system level to ensure that the system satisfies the performance and design criteria in the system specification.

VALIDITY -- The ability of the computer program to provide the performance, functions and interfaces that are sufficient for beneficial application in the intended user environment. Validity pertains to the specifications as well as the resulting software.

VERIFICATION -- The interactive process of determining whether the product of each step of the configuration item development process fulfills all of the requirements levied by the previous step.

VERSION DESCRIPTION DOCUMENT -- This document provides a file that indicates the exact information on the software media and accompanies a configuration item to provide pertinent data regarding the computer program.

WALKTHROUGH -- A process by which a team of programming personnel do an in-depth review of a program or portion of a program by inspection to detect errors and improve program reliability.

APPENDIX B

LITERATURE REFERENCE LIST

ANNOTATED BIBLIOGRAPHY

- 1) TITLE: SOFTWARE RELIABILITY MODELLING AND ESTIMATION TECHNIQUES
AUTHOR: AMRIT L. GOEL, SYRACUSE UNIVERSITY
DOC DATE: 82/10
SOURCE: RADC
ABSTRACT: This report presents the results of the software reliability modelling and estimation research pursued under contract F30602-78-C-0351 during the period October 1978 - October 1981. Two new models of very general applicability are introduced and the necessary mathematical and practical details are developed in this report. A new methodology for determining when to stop testing and start using software is described and developed. Finally a new model for analyzing the operational performance of a combined hardware-software system is reported even though it was not a part of the original research plan.

- 2) TITLE: COMBINED HW/SW RELIABILITY MODELS
AUTHOR: HUGHES CO., L. JAMES, J. BOWEN, J. MCDANIEL
DOC DATE: 82/4
SOURCE: RADC-TR-82-68
ABSTRACT: A general methodology is developed for combining hardware and software reliability. Based on this general methodology, a baseline combined HW/SW reliability was developed incorporating and unifying the SW reliability theory of Jelinski-Moranda, Geol-Okumoto with traditional HW reliability theory. The baseline model is computerized and includes various HW/SW failure and repair characteristics, allowance for imperfect SW debugs and modes of HW/SW interaction. Finally a HW/SW tradeoff procedure is developed using a combined HW/SW availability measure. Examples are provided to illustrate the general theory and tradeoff procedure.

ANNOTATED BIBLIOGRAPHY

- 3) TITLE: EXECUTIVE SUMMARY OF COMMUNICATIONS PROCESSOR OPERATING SYSTEM STUDY
- AUTHOR: JULIAN GITLIN
- DOC DATE: 80/11
- SOURCE: RADC-TR-80-316
- ABSTRACT: This report is an executive summary prepared by the RADC program manager for the communications processing operating system program accomplished by Plessey Fairfield and Data Industries for RADC under contract F30602-76-C-0456. The CPOS final report consists of nine volumes which include the major technical areas of concern in designing a secure, accountable and reliable operating system that would control the hardware/software resources of an integrated switching node for the defense communications system in the 1990's. The CPOS final report consists of nine volumes:
1. Communications Switch Operating System Study Requirements Analysis
 2. Software Reliability Study
 3. Security Considerations Study
 4. Operating System Survey
 5. Candidate Selection
 6. Implementation Methods Study
 7. Verification and Validation
 8. Design Specification
 9. Experimentation
- Although preparation of reliable software depends on an understanding of the type of errors that occur and their causes, fault analysis or failure mode analysis often are not applicable because software errors are less systematic than hardware errors. The use of a system-wide design procedure and a program support library have been recommended to offset errors induced by the individual skills and thought processes of the programming staff. No single reliability modelling approach has been able to handle the probabilistic estimates of errors remaining in a program. Use of a comprehensive error detection log and several models serve as management indicators of program reliability, but will not serve as formal acceptance criteria. Failure detection and recovery are necessary elements of program design, but error correction is not recommended. Implementation methods recommended include (1) imposition of DoD standards and guidelines, (2) PERT management methodology, (3) modular design, (4) top down design, (5) HOL, (6) structured programming, (7) top down programming, (8) automated debugging tools, and (9) formal design methodology.

ANNOTATED BIBLIOGRAPHY

- 4) TITLE: SSD SYSTEMS EFFECTIVENESS SOFTWARE RELIABILITY STANDARD
AUTHOR: LOCKHEED/W. HANSEN
DOC DATE: 80/5/1
SOURCE: RADC/LOCKHEED MISSILES AND SPACE COMPANY, INC.
ABSTRACT: This standard provides computer software reliability development engineering practices oriented toward modern development techniques for use in project planning and procurement. It is designed to compliment the "SSD Standards and Practices for Software Engineering" manual for those projects with specific quantitative reliability requirements or goals. The overall intent of this reliability standard is to assure that reliability requirements will be achieved with a minimum of cost/design/technical risk. In addition, justifiable functionally equivalent approaches or solutions will be considered. All reliability procedures and techniques herein are intended to allow equivalent substitutions when approved.
- 5) TITLE: COMMUNICATIONS PROCESSOR OPERATING SYSTEM TASK-2 RELIABILITY CONSIDERATIONS
AUTHOR: PLESSEY FAIRFIELD/DATA INDUSTRIES, R. WAXMAN, R. DOMITZ, GOLDBERG
DOC DATE: 80/6
SOURCE: RADC-TR-80-187, VOL II OF NINE
ABSTRACT: This document, a part of the series whose executive summary is covered in reference 0003, attempts to define and quantify the concept of software reliability. It covers (1) software errors, (2) software reliability from the viewpoint of programming techniques, (3) operational techniques for assuming software with a given level of reliability, and (5) integrity of a system that undergoes catastrophic failure.

ANNOTATED BIBLIOGRAPHY

- 6) TITLE: SOFTWARE RELIABILITY: REPETITIVE RUN EXPERIMENTATION AND MODELING
AUTHOR: BOEING/P. NAGEL, J. SKRIVAN
DOC DATE: 82/2
SOURCE: RADC/NASA CR-165836
ABSTRACT: Boeing conducted a controlled software-development experiment in support of software-reliability estimation and modelling. Two programmers individually designed and coded three programs each from three specifications. These programs were executed in repetitive run sampling, where failure data was recorded on each of a series of program states. The data was used to verify that interfailure times are exponentially distributed, to obtain estimates of the failure rates of individual errors and to demonstrate how widely the rates vary. This latter fact invalidates many of the popular software reliability models now in use. It was observed that the log failure rate of interfailure time was nearly linear as a function of the number of errors corrected. Cox's proportional hazards model is proposed as a new model. Estimates for the unknown parameters were obtained for all programs. A tentative physical predictor was proposed based on Halstead's information criteria N which might be used in forecasting model parameters.
- 7) TITLE: PROCEEDINGS SEMINAR ON IMPROVING AVAILABILITY OF HW-SW SYSTEMS
AUTHOR: LOS ANGELES CHAPTERS OF COMPUTER SOCIETY, IEEE ; RELIABILITY SOCIETY
DOC DATE: 82/11/13
SOURCE: RADC
ABSTRACT: The rapid development in Computer Technology of the past two decades has brought with it an urgent need to provide advanced methods of fulfilling the potential of these hardware-software systems in the user environment. However, the differences of skills and working environments between hardware and software designers has frequently clouded the understanding, responsibility and needed contribution of each discipline's role.

ANNOTATED BIBLIOGRAPHY

- 8) TITLE: SOFTWARE RELIABILITY STUDY
AUTHOR: TRW/T. THAYER
DOC DATE: 76/3/19
SOURCE: RADC
ABSTRACT: A study of software errors presented. Techniques for categorizing errors according to type, identifying their source and detecting them are discussed. Various techniques used in analyzing empirical error data collected from four large software systems are discussed and results of analysis are presented. Use of results to indicate improvements in the error prevention and detection processes through use of tool and techniques is also discussed. A survey of software reliability models is included, and recent work on TRW's Mathematical Theory of Software Reliability (MTSR) is presented. Finally, lessons learned in conjunction with collecting software data are outlined, with recommendations for improving the data collection.
- 9) TITLE: PERFORMANCE ENGINEERING OF SOFTWARE: A CASE STUDY
AUTHOR: C. SMITH/DUKE UNIV., J. BROWNE/UNIV. OF TEXAS AT AUSTIN
DOC DATE: 82
SOURCE: RADC
ABSTRACT: This paper summarizes the concepts of performance engineering in large software systems and illustrates the application of performance engineering techniques to the early design phase of a large database system.
Performance engineering is a methodology for evaluating the performance of software systems throughout their life cycles. The case study given here demonstrates that it is possible to predict resource usage patterns of complex software systems even in early design phases of the system, although detailed predictions of resource usage are not likely to be validated. The results presented here show the leverage of considering performance implications in the early design phases of a software project.

ANNOTATED BIBLIOGRAPHY

- 10) TITLE: INCREASING INFORMATION SYSTEMS PRODUCTIVITY
AUTHOR: C. SMITH/DUKE UNIVERSITY
DOC DATE: 81/5
SOURCE: RADC
ABSTRACT: Performance engineering is defined as a practical discipline for use through the life cycle to increase information system productivity. Productivity is increased through maintenance of output with the same number of personnel, in a shorter time, and with a broader customer base attraction. Considerations in performance improvement include: (1) Product use, (2) Design Definition and Alternatives, (3) Design Update Definition and Alternatives, (4) Host Computer, and (5) Operational Constraints. These are evaluated in a "best case" environment in early stages of software development, with more detailed average performance analysis following. The goal is to continually improve design. Factors critical to success of a performance engineering project are: (1) Management Commitment, (2) Schedule adjustment, (3) Credibility of results, (4) Timely results, (5) Justification of recommendations, (6) Optimistic vs realistic analysis, and (7) Cooperative effort. A schedule that allows performance engineering, based upon realistic use scenarios, will build a higher quality product.
- 11) TITLE: SOFTWARE RELIABILITY - BIBLIOGRAPHY
AUTHOR: BALBIR S. DHILLON
DOC DATE: 81/9
SOURCE: MMC
ABSTRACT: This paper presents a brief introduction and an extensive bibliography of topics in software reliability and related areas.
- 12) TITLE: A COMPATIBLE HARDWARE/SOFTWARE RELIABILITY PREDICTION MODEL
AUTHOR: X. CASTILLO, T. SMITH
DOC DATE: 81/7/22
SOURCE: CARNEGIE-MELLON UNIVERSITY
ABSTRACT: A new modeling methodology to characterize failure processes in Time-Sharing systems due to hardware transients and software errors is presented. The basic assumption made is that the instantaneous failure rate can be approximated by a deterministic function of time plus a zero-mean stationary Gaussian process, both depending on the usage of the resource considered. The probability density function of the time to failure obtained has a decreasing hazard function. Implications of this methodology are discussed.

ANNOTATED BIBLIOGRAPHY

- 13) TITLE: MANAGEMENT OVERVIEW OF SYSTEM TECHNICAL SUPPORT PLAN FOR THE FIRE-FINDER SYSTEM SUPPORT CENTER
AUTHOR: L. HESELTON/SEMCOR, INC
DOC DATE: 80/8/8
SOURCE: NTIS AD-A095555
ABSTRACT: This System Technical Support Plan outlines the way to correct software problems on counter-mortar and counter-artillery radars. It was found that training field repairmen in system software was not a cost effective way to correct software problems. A system support center was established to resolve software faults or problems instead. They support field personnel in determining if a problem is caused by a hardware or software fault, and correct software faults. The support center determines: 1) fix, 2) testing to be performed, and 3) releases.
- 14) TITLE: FAULT DETECTION EFFICIENCY MEASUREMENT VIA HW FAULT SIMULATION
AUTHOR: C. TIMOC/TIMOC INTERNATIONAL CO.
DOC DATE: 80/3
SOURCE: NTIS
ABSTRACT: The overall objective of this program is to provide the means for testing so as to assure nearly fault-free operation. A more specific objective is to measure the stuck fault detection efficiency of the test vectors developed by JPL/Hughes for the MIL-M-38510/470 NASA.
A hardware stuck fault simulator for the 1802 microprocessor was implemented and the stuck fault detection efficiency of the test vectors developed by JPL/Hughes for the MIL-M-38510/470 NASA were measured in three phases as follows:
Phase 1. Build a breadboard system to perform the fault-free function of the 1802.
Phase 2. Add fault simulation capabilities to the fault-free breadboard.
Phase 3. Measure the stuck fault detection efficiency of the test vectors.
A total of 874 faults were injected into the combinatorial and sequential parts of the RCA 1802 microprocessor and it was found that 39 stuck faults were not detected. Therefore, the measured stuck fault detection efficiency of the MIL-M-38510/470 NASA is 95%. Since the 39 undetected faults can create catastrophic errors in equipment designed for high reliability applications, it is recommended that the MIL-M-38510/470 NASA be enhanced with additional test vectors so as to achieve 100% stuck fault detection efficiency.

ANNOTATED BIBLIOGRAPHY

- 15) TITLE: FAULT-TOLERANT SOFTWARE FOR SPACECRAFT APPLICATIONS
AUTHOR: H. HECHT/AEROSPACE CORP.
DOC DATE: 75/12/10
SOURCE: NTIS AD A022068
ABSTRACT: Fault-tolerant computers have been developed for applications that require a very high degree of hardware reliability, and it is frequently asked whether similar techniques can be brought to bear on software for critical applications, e.g., ascent guidance software on launch vehicles, launch-control software for ground computers, and control and command software. The principal techniques employed in hardware fault tolerance are seen to be applicable also through software fault tolerance: error detection, protective redundancy, and rollback provisions. Of course, they need to be implemented in a specific manner; particularly the redundancy must be provided by a different code than that used for the primary modules.
The recovery block (proposed by Randell), with the addition of a watchdog timer, has been implemented in a number of skeleton routines and has been found quite suitable in connection with the established structure for spaceborne software.
A reliability model is proposed that shows a very considerable reduction in failure probability even when the fault-tolerance provisions themselves are far from perfect. It is therefore believed that the time is quite ripe to undertake serious studies of fault-tolerant software for space applications.
- 16) TITLE: THE COST OF SOFTWARE FAULT TOLERANCE
AUTHOR: G. MIGNEAULT/NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
DOC DATE: 82/9
SOURCE: NTIS
ABSTRACT: This paper proposed the use of software fault tolerance techniques as a means of reducing software costs in avionics and as a means of addressing the issue of system unreliability due to faults in software. A model is developed to provide a view of the relationships among cost, redundancy, and reliability which suggests strategies for software development and maintenance which are not conventional.
Observations are made about the problem of escalating budget for software and about the nature of some of the causes of increasing software cost. Attention is paid to schemes for using dissimilar redundancy in software to obtain a high level of reliability.

ANNOTATED BIBLIOGRAPHY

- 17) TITLE: EXCEPTION HANDLING AND SOFTWARE FAULT TOLERANCE
AUTHOR: F. CRISTIAN
DOC DATE: 82
SOURCE: NTIS
ABSTRACT: Some basic concepts underlying the issue of fault tolerant software design are investigated. Relying on these concepts a unified point of view on programmed exception handling based on backward recovery is constructed. The cause-effect relationship between software design fault and failure occurrences is explored and a class of faults for which exception handling can provide effective fault tolerance is characterized. It also shows that there exists a second class of design faults which cannot be tolerated by using default exception handling. The role that software verification methods can play in avoiding the production of such faults is discussed.
- 18) TITLE: PRODUCTION OF RELIABLE FLIGHT CRUCIAL SOFTWARE: VALIDATION METHODS RESEARCH FOR FAULT TOLERANT AVIONICS AND CONTROL SYSTEMS SUB-WORKING GROUP MEETING
AUTHOR: J. DUNHAM, J. KNIGHT/RESEARCH TRIANGLE INST.
DOC DATE: 82/5
SOURCE: NTIS
ABSTRACT: The state of the art in the production of crucial software for flight control applications was addressed. The association between reliability metrics and software is considered. Thirteen software development projects are discussed. A short term need for research in the areas of tool development and software fault tolerance was indicated. For the long term, research in format verification or proof methods was recommended. Formal specification and software reliability modeling, were recommended as topics for both short and long term research.

ANNOTATED BIBLIOGRAPHY

- 19) TITLE: AN OVERVIEW OF RELIABLE COMPUTER SYSTEM DESIGN
AUTHOR: J. MC DONALD/ROYAL SIGNALS AND RADAR ESTABLISHMENT
DOC DATE: 79/11
SOURCE: NTIS
ABSTRACT: This paper was produced to support a series of lectures on reliable computer system design given at a NATO ASI summer school on multiple processor computers. The paper was intended to be fairly self-contained, but it does lack a description of a practical fault tolerant system. The paper presents an overview of reliable computer system design. It attempts to provide a pragmatic guide to redundancy and recovery, but does not give a very thorough discussion of either the theory or philosophy of reliable systems. The paper introduces and defines the basic concepts of reliability, and describes the basic mechanisms for achieving fault tolerance. It compares the attributes of multi-processor and multi-computer systems from the point of view of reliability. It describes in some detail techniques for achieving tolerance to both hardware and software faults. The paper concludes by outlining some of the major unsolved problems of reliable system design.
- 20) TITLE: ANALYSIS OF FAULT DETECTION, CORRECTION, AND PREVENTION IN INDUSTRIAL COMPUTER SYSTEMS
AUTHOR: E. SCHAFFER, T. WILLIAMS/PURDUE UNIV.
DOC DATE: 77/9
SOURCE: NTIS
ABSTRACT: This research is concerned with three fault-tolerant computer methods for meeting reliability requirements: (1) Hardware redundancy is defined as any circuitry in the system not necessary for normal computer operation should no faults occur; (2) Software redundancy is defined as additional program instructions present solely to handle faults; and (3) Time redundancy is defined as any retrial of instructions. In order to provide an understanding of the fault-tolerant methods under study today, examples of their uses and limitations are presented. Hardware aspects of coding and modular redundancy are discussed. Discussions of software include means of protection, detection, and correction of software faults through software as well software methods to handle hardware errors. These methods include diagnostics as well as executive recovery techniques and retrials of instructions through time redundancy. Present day computer capabilities also are presented. Finally, duplex & triplex fault-tolerant industrial computer systems are discussed that may be built from conventional computers with little or no need for expensive additional hardware.

ANNOTATED BIBLIOGRAPHY

- 21) TITLE: EFFECTS OF FAILURE ON PERFORMANCE OF GRACEFULLY DEGRADABLE SYSTEMS
AUTHOR: J. LOSQ/STANFORD UNIV.
DOC DATE: 76/12
SOURCE: NTIS AD-A049849
ABSTRACT: The recent development of multiprocessor systems that offer resistance to faults by gracefully degrading after a failure opens vast new ranges of applications for fault tolerance and high reliability. The paper presents a general model for the evaluation of such systems. It takes into account the internal structure of the hardware, the characteristics of the various detection mechanisms, the unreliability of the software and even the type of applications these systems are used for. It provides many measures of the systems' performance such as: availability, meantime between crashes, average processing power and proportion of time spent in degraded mode. System optimization gives the best values for the number of processors, memories, ..., and shows the trade-offs between hardware and software fault-detection mechanisms. The model is illustrated by a concrete example.
- 22) TITLE: SOFTWARE QUALITY ASSURANCE
AUTHOR: ED SOISTMAN
DOC DATE: JUNE 1979
SOURCE: UNIVERSITY OF CENTRAL FLORIDA
ABSTRACT: The problems associated with software development and use are investigated from a management point of view. Having identified the critical aspects of effective software management, an approach is suggested for the creation and implementation of a software quality assurance program. Particular attention is focused on the concept of Life Cycle Procurement as currently utilized by the Department of Defense.
- 23) TITLE: ENGINEERING RELIABILITY - NEW TECHNIQUES AND APPLICATIONS
AUTHOR: B. DHILLON
DOC DATE:
SOURCE: MMC
ABSTRACT: This article stresses that most of the work in the area of software reliability can be divided into the following three categories.
1. Writing correct programs to begin with.
2. Testing the programs to take out bugs.
3. Modeling of software in an attempt to predict its reliability and possibly study the impact of related parameters.
The following models were discussed :
Shooman, Markov and Jelinski-Moranda.

ANNOTATED BIBLIOGRAPHY

- 24) TITLE: THE SOFTWARE DEVELOPMENT NOTEBOOK - A PROVEN TECHNIQUE
AUTHOR: JOHN MCKISSICK JR AND ROBERT A. PRICE
DOC DATE: 1979
SOURCE: PROCEEDINGS 1979 ANNUAL RELIABILITY AND MAINTAINABILITY SYMPOSIUM
ABSTRACT: The continuing need for improved computer software demands improved software development techniques such as the Software Development Notebook. The organization, content, use and audit of Software Development Notebooks are documented in this paper. Experience and results from the application of this technique are also presented.
- 25) TITLE: QUANTITATIVE SOFTWARE RELIABILITY MODELS - DATA PARAMETERS
AUTHOR: DOROTHY SWEARINGEN AND JOHN DONAHOO
DOC DATE: OCTOBER 1979
SOURCE: WORKSHOP ON QUANTITATIVE SOFTWARE MODELS, NY, OCT. 1979.
ABSTRACT: This paper summarizes the results of a study to identify data requirements for software reliability modelling. Brief descriptions of the models and the data needed to exercise the models are presented. The paper concludes with a list of recommendations for future data collection.
- 26) TITLE: MODULARITY IS NOT A MATTER OF SIZE
AUTHOR: ROBERT H. DUNN AND RICHARD S. ULLMAN
DOC DATE: 1979
SOURCE: PROCEEDINGS 1979 ANNUAL RELIABILITY AND MAINTAINABILITY SYMPOSIUM
ABSTRACT: Division of a computer program into a number of smaller programs designated as modules is a universally accepted practice among software engineers. A modular architecture offers the following advantages:
- Parallel Development
- Reduced program size and costs
- Understandability
- Reliability
- Testing

ANNOTATED BIBLIOGRAPHY

- 27) TITLE: AN ANALYSIS OF ERRORS AND THEIR CAUSES IN SYSTEMS PROGRAMS
AUTHOR: ALBERT ENDRES, IBM GERMANY, BOEBLINGEN, GERMANY
DOC DATE: APRIL 1975
SOURCE: INTERNATIONAL CONFERENCE ON RELIABLE SOFTWARE, APRIL 1975
ABSTRACT: Program errors detected during internal testing of the operating system DOS/VS form the basis for an investigation of error distributions in system programs. Using a classification of errors according to various attributes, conclusions can be drawn concerning the possible causes of these errors. The information thus obtained is applied in a discussion of the most effective methods for detection and prevention of errors.
- 28) TITLE: METRICS FOR EVALUATION OF QUANTITATIVE SOFTWARE MODELLING DATABASES
AUTHOR: JON MARTENS, IIT RESEARCH INSTITUTE, ROME, NY
DOC DATE: OCTOBER 1979
SOURCE: WORKSHOP ON QUANTITATIVE SOFTWARE MODELS, NY, OCT. 1979
ABSTRACT: Two metrics for evaluating software modelling databases are developed in this paper. The integration metric measures the degree of data element sharing between datasets; the coverage metric measures the datasets' ability to fulfill a model's requirements.
- 29) TITLE: SOFTWARE QUALITY METRICS FOR LIFE-CYCLE COST-REDUCTION
AUTHOR: GENE F. WALTERS AND JAMES A. MCCALL
DOC DATE: AUGUST 1979
SOURCE: IEEE TRANSACTIONS ON RELIABILITY VOL R-28 #3, AUGUST 1979
ABSTRACT: This paper identifies factors or characteristics of which reliability is one, which comprise the quality of computer software. It then discusses their impact over the life of a software product and describes a methodology for specifying them quantitatively, including them in system design, and measuring them during development. The methodology is still experimental, but is rapidly evolving toward application in all types of software. The paper emphasizes those factors of software quality which have the greatest importance at the later stages of a software products life.

ANNOTATED BIBLIOGRAPHY

- 30) TITLE: SPECIAL SERIES ON SYSTEM INTEGRATION
AUTHOR: ELECTRONIC DESIGN AND SYSTEMS & SOFTWARE
DOC DATE: 83/4/14
SOURCE: MMC
ABSTRACT: Tools have become available to catch errors as soon as possible in the software development cycle. These include the familiar syntax and data type checkers, as well as automatic change and configuration control bookkeepers. Specification language checkers are even available to reject errors from the start. The article covers some of these programs, and describes a reliability predictor that uses rate of error discovery statistics to compute MTBF. The rate of closure from the observed to desired has been validated on 20 programming projects and is described as suitable for use in scheduling program delivery to the customer. Ada reliability aspects are also mentioned.
- 31) TITLE: A GENERAL SOFTWARE RELIABILITY MODEL FOR PERFORMANCE PREDICTION
AUTHOR: J. SHANTHIKUMAR/SYRACUSE UNIV.
DOC DATE: 81/3/2
SOURCE: MICROELECTRON RELIABILITY VOL. 21
ABSTRACT: In this paper we give a general Markov process formulation for a software reliability model and present expressions for software performance measures. We discuss a general model and derive the maximum likelihood estimates for the required parameters of this model. The generality of this model is demonstrated by showing that the Jelinski-Moranda model and the Non-Homogeneous Poisson Process (NHPP) model are both very special cases of our model. In this process we also correct some errors in a previous paper of the NHPP model.

ANNOTATED BIBLIOGRAPHY

- 32) TITLE: SOFTWARE RELIABILITY - STATUS AND PERSPECTIVES
AUTHOR: C. RAMAMOORTHY, F. BASTANI
DOC DATE: 81/12/21
SOURCE: IEEE TRANSACTIONS ON SOFTWARE ENGINEERING 7/82
ABSTRACT: It is essential to assess the reliability of digital computer systems used for critical real-time control applications (e.g, nuclear power plant safety control systems). This involves the assessment of the design correctness of the combined hardware/software system as well as the reliability of the hardware. In this paper we survey methods of determining the design correctness of systems as applied to computer programs. Automated program proving techniques are still not practical for realistic programs. Manual proofs are lengthy, tedious, and error-prone. Software reliability provides a measure of confidence in the operational correctness of the software. Since the early 1970's several software reliability models have been proposed. We classify and discuss these models using the concepts of residual error size and the testing process used. We also discuss methods of estimating the correctness of the program and the adequacy of the set of test cases used. These methods are directly applicable to assessing the design correctness of the total integrated hardware/software system which ultimately could include large complex distributed systems.
- 33) TITLE: LIKELIHOOD FUNCTION OF A DEBUGGING MODEL FOR COMPUTER SOFTWARE RELIABILITY
AUTHOR: B. LITTLEWOOD, J. VERRALL/CITY UNIVERSITY, LONDON
DOC DATE: 82/6/02
SOURCE: IEEE TRANSACTIONS ON RELIABILITY , VOL R-30, NO. 2, 6/81
ABSTRACT: A simple model for software reliability growth, originally suggested by Jelinski & Moranda, has been widely used but suffers from difficulties associated with parameter estimation. It is shown that a major reason for obtaining nonsensical results from the model is its application to data sets which exhibit decreasing reliability. Presented is a simple, necessary and sufficient condition for the maximum likelihood estimates to be finite and suggest that this condition be tested prior to using the model.

ANNOTATED BIBLIOGRAPHY

- 34) TITLE: A SUMMARY OF THE DISCUSSION ON "AN ANALYSIS OF COMPETING SOFTWARE RELIABILITY MODELS"
- AUTHOR: AMRIT L. GOEL
- DOC DATE: 80/9
- SOURCE: IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 9/80
- ABSTRACT: In March 1978, Schick and Wolverton published a paper in the IEEE Transactions On Software Engineering. Moranda (later) criticized several aspects of this paper. His critique was reviewed by Littlewood and rebutted by Schick and Wolverton. The purpose of this note is to summarize and comment on the main points raised.
- 35) TITLE: THEORIES OF SOFTWARE RELIABILITY: HOW GOOD ARE THEY AND HOW CAN THEY BE IMPROVED
- AUTHOR: B. LITTLEWOOD
- DOC DATE: 79/3/23
- SOURCE: IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 9/80
- ABSTRACT: An examination of the assumptions used in early bug-counting models of software reliability shows them to be deficient. Suggestions are made to improve modeling assumptions and examples are given of mathematical implementations. Model verification via real-life data is discussed and minimum requirements are presented. An example shows how these requirements may be satisfied in practice. It is suggested that current theories are only the first step along what threatens to be a long road.

AD-A165 231

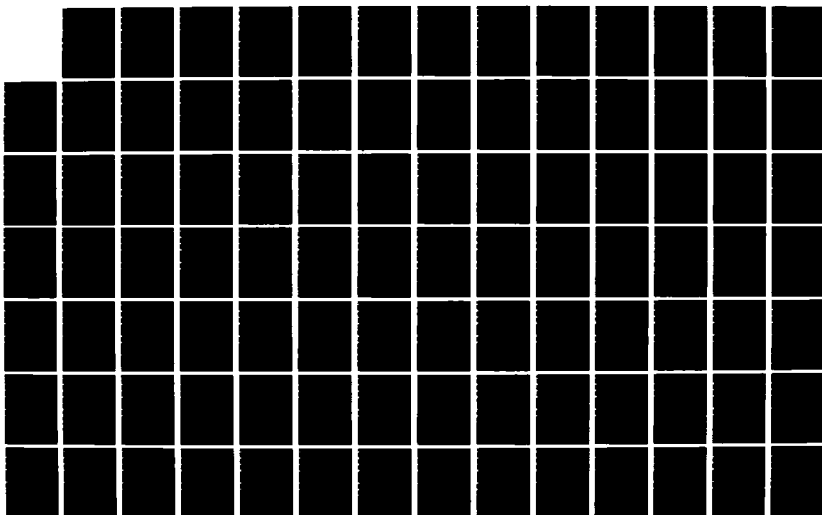
IMPACT OF HARDWARE/SOFTWARE FAULTS ON SYSTEM
RELIABILITY VOLUME 1 STUDY R. (U) MARTIN MARIETTA
AEROSPACE ORLANDO FL E C SOISTMAN ET AL. DEC 85
OR-18173 RADC-TR-85-228-VOL-1

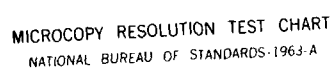
2/3

UNCLASSIFIED

F/G 9/2

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

ANNOTATED BIBLIOGRAPHY

- 36) TITLE: A USER-ORIENTED SOFTWARE RELIABILITY MODEL
AUTHOR: R. CHEUNG
DOC DATE: 79/8/13
SOURCE: IEEE TRANSACTIONS ON SOFTWARE ENGINEERING
ABSTRACT: A user oriented reliability model has been developed to measure the Reliability that a system provides to a user community. It has been observed that in many systems, especially software systems, reliable service can be provided to a user when it is known that errors exist, provided that the service requested does not utilize the defective parts. The reliability of service therefore, depends both on the reliability of the components and the probabilistic distribution of the utilization of the components to provide the service. In this paper, a user-oriented software reliability figure of merit is defined to measure the reliability of a software system with respect to a user environment. The effects of the user profile, which summarizes the characteristics of the users of a system, on system reliability are discussed. A simple Markov model is formulated to determine the reliability of a software system based on the reliability of each individual module and the measured intermodular transition probabilities as the user profile. Sensitivity analysis techniques are developed to determine modules most critical to system reliability. The applications of this model to develop cost-effective testing strategies and to determine the expected penalty cost of failures are also discussed. Some future refinements and extensions of the model are presented.
- 37) TITLE: MODELS FOR HARDWARE-SOFTWARE SYSTEM OPERATIONAL-PERFORMANCE EVALUATION
AUTHOR: A. GOEL, J. SOENJOTO/SYRACUSE UNIVERSITY
DOC DATE: 81/5/18
SOURCE: IEEE TRANSACTIONS ON RELIABILITY, 8/81
ABSTRACT: Stochastic models for hardware-software systems are developed and used to study their performance as a function of hardware-software failure and maintenance rates. Expressions are derived for the distribution of time to a specified number of software errors, system occupancy probabilities, system reliability, availability, and average availability. The behavior of these measures is investigated via numerical examples.

ANNOTATED BIBLIOGRAPHY

- 38) TITLE: STOCHASTIC RELIABILITY-GROWTH: A MODEL FOR FAULT-REMOVAL IN
COMPUTER PROGRAMS AND HARDWARE-DESIGNS
AUTHOR: B. LITTLEWOOD/CITY UNIVERSITY, LONDON
DOC DATE: 81/1/12
SOURCE: IEEE TRANSACTIONS ON RELIABILITY, 10/81
ABSTRACT: An assumption commonly made in early models of software reliability is that the failure rate of a program is a constant multiple of the (unknown) number of faults remaining. This implies that all faults contribute the same amount to the failure rate of the program. The assumption is challenged and an alternative proposed. The suggested model results in earlier fault-fixes having a greater effect than later ones (the faults which make the greatest contribution to the overall failure rate tend to show themselves earlier, and so are fixed earlier), and the DFR property between fault fixes (assurance about programs increases during periods of failure-free operation, as well as at fault fixes). The model is tractable and allows a total execution time to achieve a target reliability, and total number of fault fixes to target reliability, are obtained. The model might also apply to hardware reliability growth resulting from the elimination of design errors.
- 39) TITLE: AN ERROR DETECTION MODEL FOR APPLICATION DURING SOFTWARE
DEVELOPMENT
AUTHOR: P. MORANDA/McDONALD DOUGLAS ASTRONAUTICS COMPANY
DOC DATE: 81/1/29
SOURCE: IEEE TRANSACTIONS ON RELIABILITY, 10/81
ABSTRACT: A variation of the Jelinski/Moranda model is described. The main feature of this new model is that the expanding size of a developing program is accommodated, so that the quality of a program can be estimated by analyzing an initial segment of the written code. Two parameters are estimated. The data are: a) time separations between error detections, b) the number of errors per written instruction, c) the failure rate (or finding rate) of a single error, and d) a time record of the number of instructions under test. This model permits predictions of MTTF and error content of software.

ANNOTATED BIBLIOGRAPHY

- 40) TITLE: RELIABILITY AND MAINTAINABILITY OF ELECTRONIC SYSTEMS --
CHAPTER 6 SOFTWARE RELIABILITY AND MAINTAINABILITY
AUTHOR: J. ARSENAULT
DOC DATE: 79/1
SOURCE: A. FLOWERS
ABSTRACT: This article expounds on attributes which may be employed in the generation of reliable and maintainable software including: modularity, emphasis on system design not coding, top down approach and structured programming.
- 41) TITLE: IEEE RELIABILITY SOCIETY NEWSLETTER
AUTHOR: IEEE/SUSAN EAMES, EDITOR
DOC DATE: 83/4
SOURCE: MMC-E. GRIFFIN
ABSTRACT: This document consists of chapter meeting notes and a status summary for 1982.
- 42) TITLE: HARDWARE/SOFTWARE FMECA
AUTHOR: FRED M. HALL; EVALUATION RESEARCH CORP.; ARLINGTON
RAYMOND A. PAUL; SURFACE WEAPONS CENTER; DAHLGREN
WENDY E. SNOW; EVALUATION RESEARCH CORP.; ARLINGTON
DOC DATE: JANUARY 1983
SOURCE: 1983 PROCEEDINGS ANNUAL RELIABILITY AND MAINTAINABILITY SYMPOSIUM
ABSTRACT: (AUTHORS) This paper describes procedures which can be used to determine reliability REQUIREMENTS for both hardware and software elements of a system which incorporates an embedded computer subsystem. The analysis technique, Hardware/Software Failure Modes, Effects and Criticality Analysis (FMECA) may be utilized to ENSURE reliable software throughout the software development cycle; including requirements definition, coding, checkout and software test ... H/S FMECA provides a method for examining software errors at the highest functional levels, then progressively tracking errors into lower level functions
- 43) TITLE: GROUND LAUNCHED ASSAULT BREAKER FLIGHT PROGRAM PROBLEM REPORT FILE
AUTHOR: E.L. GRIFFIN
DOC DATE: AUGUST 1982
SOURCE: VARIOUS MARTIN MARIETTA DEVELOPMENTAL TEAM PERSONNEL
ABSTRACT: This log contains the complete set of 166 software problem reports written against the 14K Ground Launched Assault Breaker Missile Flight Program. The problem reports reflect hardware, software, and combination hardware/software problems for a very complex real-time program that navigates to a target based on airborne target update data and dispenses submunitions on a tank.

ANNOTATED BIBLIOGRAPHY

- 44) TITLE: CAN SOFTWARE BENEFIT FROM HARDWARE EXPERIENCE
AUTHOR: HERBERT HECHT
DOC DATE: 1975
SOURCE: PROCEEDINGS 1975 ANNUAL RELIABILITY AND MAINTAINABILITY SYMPOSIUM
ABSTRACT: (AUTHOR) The question posed in the title is answered with a qualified "yes". While software requires a completely different attack on failure mechanisms than used for hardware, there is considerable commonality in reliability organization and procedures, and some in the systems reliability area. The present lack of a generally accepted metric for software reliability impedes the information transfer from the hardware field. Steps towards a quantitative measure for software reliability are outlined.
- 45) TITLE: ON SOFTWARE COMPLEXITY
AUTHOR: L. A. BELADY
DOC DATE: OCTOBER 1979
SOURCE: WORKSHOP ON QUANTITATIVE SOFTWARE MODELS, N.Y., OCT. 1979
ABSTRACT: In 1974 there was very little known work on complexity in general and even less on the complexity of programming. At that time Professor Beilner and myself shared an office at Imperial College and according to our mutual interest, discussed problems of "complex" programs and of large scale programming....We found a sizeable body of work on computational complexity (and after 5 year's work (ecs)) ... we learned that complexity can be perceived in at least two different ways: sometimes it appears as a measure of uncertainty or surprise and sometimes it is deterministic and is defined as a count or magnitude. ... The objective of this paper is to give a brief and a little bit organized survey of this "complex" repertoire of approaches.
- 46) TITLE: IN SEARCH OF SOFTWARE COMPLEXITY
AUTHOR: DR. BILL CURTIS, GENERAL ELECTRIC, ARLINGTON VA
DOC DATE: OCTOBER 1979
SOURCE: WORKSHOP ON QUANTITATIVE SOFTWARE MODELS N.Y., OCT. 1979
ABSTRACT: This paper is a summary of a tutorial given on the state-of-the-art in software complexity research... Different types of complexity metrics are reviewed along with a framework for studying large compilations of metrics. Ultimately, a definition is posed on interactional terms which encompasses diverse areas within the field.

ANNOTATED BIBLIOGRAPHY

- 47) TITLE: AN ANTI-COMPLEXITY EXPERIMENT
AUTHOR: L. A. BELADY
DOC DATE: OCTOBER 1979
SOURCE: WORKSHOP ON QUANTITATIVE SOFTWARE MODELS, N.Y., OCT. 1979
ABSTRACT: ...[I]n the early 70's systematic studies of large programs uncovered an established the evolutionary nature of software, namely that programs are not static objects but undergo continuous modification to cope with the everchanging environment...[C]omplexity, a term which in this context informally describes the difficulty of predictably manipulating software.
- 48) TITLE: SOFTWARE FAULT TOLERANCE: WHY IT'S NECESSARY AND A METHODOLOGY
AUTHOR: MYRON HECHT, SOHAR INCORPORATED, LA JOLLA, CA
DOC DATE: 13 NOV 1982
SOURCE: PROCEEDINGS, SEMINAR ON IMPROVING AVAILABILITY OF HARDWARE-SOFTWARE SYSTEMS, IEEE COMPUTER SOCIETY, LOS ANGELES
ABSTRACT: As real time control requirements grow more complex and progressively more demands are placed on the computer system, software related failures will become increasingly important contributors to the total number of system failures. This trend is already evident in the error data collected from several highly reliable computer systems. Unfortunately, the complete prevention of faults in complex software is beyond present technical capabilities. Under these circumstances, critical applications must provide for the toleration of software faults (in addition to hardware faults). The recovery block concept presents a widely applicable means for implementing fault tolerance within a program.

ANNOTATED BIBLIOGRAPHY

- 49) TITLE: CARE III FINAL REPORT
AUTHOR: L. BRYANT, Bryant, L. GUCCIONE, J. STIFFLER
DOC DATE: 79/11
SOURCE: NTIS/N80-15423
ABSTRACT: This report describes the work done during the first phase of a two-phase effort to develop a computer program to aid in assessing the reliability of fault-tolerant avionics systems. The overall effort consists of five major tasks: 1) Establish the basic requirements that must be satisfied if the program is to achieve its overall objective. 2) Define a general program structure consistent with these requirements. 3) Develop and program a mathematical model relating the reliability of a fault-tolerant system to the (not necessarily time-independent) failure rates and coverage factors characterizing its various elements. 4) Develop and program a mathematical model for evaluating the coverage (probability of successful recovery) associated with any given fault as a function of the type and location of the fault, the applicable fault detection and isolation mechanism, and the number and status of prior faults. 5) Develop and program a procedure whereby a user of these models can accurately and conveniently specify the configuration of the system to be evaluated and the constraints influencing its ability to recover from faults.
- The first three of these tasks were completed during Phase One; the resulting requirements, program structure, and reliability model are discussed in detail in Volume I of this report, along with the tradeoffs and sample reliability assessments made in arriving at the approach finally taken. The Computer Program Requirements Document is contained in Volume II. This latter volume also includes several appendices containing computer print-outs and other ancillary material supporting the conclusions presented in Volume I.

ANNOTATED BIBLIOGRAPHY

- 50) TITLE: FACTORS IN SOFTWARE QUALITY
AUTHOR: JIM A. MCCALL, PAUL K. RICHARDS, GENE F. WALTERS
DOC DATE: NOVEMBER 1977
SOURCE: RADC TR-77-369, VOLUMES I, II & III
ABSTRACT: An hierarchial definition of factors affecting software quality was compiled after an extensive literature search. The definition covers the complete range of software development and is broken down into non-oriented and software-oriented characteristics. For the lowest level of the software-oriented factors, metrics were developed that would be independent of the programming language. These measurable criteria were collected and validated using actual Air Force data bases. A handbook was generated that will be useful to the Air Force managers for specifying overall quality of the software system.
- 51) TITLE: SYSTEM HARDWARE AND SOFTWARE RELIABILITY ANALYSIS
AUTHOR: WILLIAM E. THOMPSON; COLUMBIA RESEARCH CORPORATION; ARLINGTON
DOC DATE: 1983
SOURCE: 1983 IEEE PROCEEDINGS OF THE ANNUAL RELIABILITY AND MAINTAINABILITY SYMPOSIUM
ABSTRACT: This paper presents a practical model and procedure for analyzing combined hardware and software reliability of embedded computer systems. The emphasis is on combined hardware and software reliability qualification testing.
- 52) TITLE: MIL-S-XXXXX: SOFTWARE RELIABILITY AND MAINTAINABILITY SPECIFICATION
AUTHOR: ELECTRONIC SYSTEMS DIVISION, HANSCOM FIELD, MASS
DOC DATE: OCTOBER 1983
SOURCE: RADC/RBET
ABSTRACT: This specification requires the establishment and implementation of a Software Reliability and Maintainability (R&M) Program by the contractor. The purpose of this program is to assure that software developed, acquired or otherwise provided under this contract complies with the requirements of the contract. It is intended that the program be effectively tailored and economically planned and developed in consonance with, or an extension of the contractor's other reliability, maintainability, quality assurance, administrative and technical programs. When referenced,....this specification shall apply to the acquisition of software either acquired alone or as part of a system. [It]..shall also apply to all deliverable design, test, maintenance and support software developed under this contract. For purposes of this specification, the term software includes firmware.

ANNOTATED BIBLIOGRAPHY

- 53) TITLE: SOFTWARE RELIABILITY ESTIMATION: A REALIZATION OF
COMPETING RISK
AUTHOR: WAY KUO
DOC DATE: SPRING 1983
SOURCE: MICROELECTRONICS AND RELIABILITY JOURNAL
ABSTRACT: A software reliability model presented here assumes a time-
dependent failure rate and that debugging can remove as well
as add faults with a non-zero probability. This paper pro-
poses a compound-distribution software reliability model. We
make three assumptions.
1) The software originally contains $X(0)$ bugs where $X(0)$
is a constant to be determined.
2) Each time interval between failures has a nonconstant
occurrence rate.
3) When a software fault is fixed, an additional error can
be introduced, the probability of correcting or intro-
ducing errors follows a discrete distribution.
- 54) TITLE: ESTIMATING SOFTWARE DEVELOPMENT
AUTHOR: E. B. DALY
DOC DATE: JULY 1979
SOURCE: IEEE
ABSTRACT: Programming effort can be estimated and evaluated in terms of
instructions generated per hour. Although this methodology
of judging software effort has been shown to be very effective,
it must be utilized with great care since rates vary dramati-
cally among software jobs having different design objectives,
different program complexity and different resources.
- 55) TITLE: STATISTICAL PREDICTION OF PROGRAMMING ERRORS
AUTHOR: R. W. MOTLEY AND W. D. BROOKS
DOC DATE: 30 NOV 1976
SOURCE: RADC-TR-77-175
ABSTRACT: The need for developing new tools and techniques for pro-
ducing more reliable low cost software...has led to attempts
to analyze the nature and types of software errors in order
to be able to accurately predict the reliability of the soft-
ware product....The report focuses on the analysis, using
multiple linear regression techniques, of software error data
and related structural, complexity, and programmer-related
variables extracted from two large DoD command and control
software projects totalling over 250,000 lines of higher order
language source code. This eight month study focused on the
statistical prediction of programming errors using a wide
range of program structure/complexity variables and selected
programmer variables as predictors.

ANNOTATED BIBLIOGRAPHY

- 56) TITLE: DESIGN AND CODE INSPECTIONS TO REDUCE ERRORS IN PROGRAM DEVELOPMENT
AUTHOR: M. E. FAGAN
DOC DATE: MARCH 1976
SOURCE: REPRINT FROM IBM SYSTEMS JOURNAL, VOL. 15, NO. 3, 1976
ABSTRACT: Substantial net improvements in programming quality and productivity have been obtained through the use of formal inspections of design and of code. Improvements are made possible by a systematic and efficient design and code verification process, with well-defined roles for inspection participants. The manner in which inspection data is categorized and made suitable for process analysis is an important factor in attaining the improvements. It is shown that by using inspection results, a mechanism for initial error reduction followed by ever-improving error rates can be achieved.
- 57) TITLE: MIL-STD-1644(TD), TRAINER SYSTEM SOFTWARE ENGINEERING REQUIREMENTS
AUTHOR: NAVAL TRAINING EQUIPMENT CENTER, ORLANDO, FL 32813
DOC DATE: 15 JAN 1982
SOURCE:
ABSTRACT: The purpose of this standard is to establish uniform requirements for the development and documentation of trainer system software. These requirements shall apply to trainer system software (including firmware and microcode) which is developed either alone or as a portion of a trainer system or subsystem development. Only unmodified computer vendor commercial software will be exempt from the development and documentation requirements of this standard.
- 58) TITLE: SOFTWARE CONSIDERATIONS IN AIRBORNE SYSTEMS AND EQUIPMENT CERTIFICATION
AUTHOR: SPECIAL COMMITTEE OF RTCA
DOC DATE: NOVEMBER 1981
SOURCE: RADIO TECHNICAL COMMISSION FOR AERONAUTICS (RTCA)
ABSTRACT:

ANNOTATED BIBLIOGRAPHY

- 59) TITLE: THE MEASUREMENT AND MANAGEMENT OF SOFTWARE RELIABILITY
AUTHOR: JOHN D. MUSA
DOC DATE: SEPTEMBER 1980
SOURCE: PROCEEDINGS OF THE IEEE
ABSTRACT: The theme of this paper is the field of software reliability measurement and its applications. Needs for and potential uses of software reliability measurement are discussed. Software reliability and hardware reliability are compared, and some basic software reliability concepts are outlined. A brief summary of the major steps in the history and evolution of the field is presented. Two of the leading software reliability models are described in some detail. The topics of combinations of software (and hardware) components and availability are discussed briefly. The paper concludes with an analysis of the current state of the art and a description of further research needs.
- 60) TITLE: VALIDITY OF EXECUTION-TIME THEORY OF SOFTWARE RELIABILITY
AUTHOR: JOHN D. MUSA
DOC DATE: AUGUST 1979
SOURCE: IEEE TRANSACTIONS ON RELIABILITY
ABSTRACT: This paper investigates the validity of the execution-time theory of software reliability. The theory is outlined, along with appropriate background, definitions, assumptions, and mathematical relationships. Both the execution time and calendar time component are described. The important assumptions are discussed. Actual data are used to test the validity of most of the assumptions. Model and actual behavior are compared. The development projects and operational computation center software from which the data have been obtained are characterized to give the reader some basis for judging the breadth of applicability of the concepts.
- 61) TITLE: OPERATIONS RESEARCH AND RELIABILITY
AUTHOR: DANIEL GROUCHKO (EDITOR)
DOC DATE: JUNE/JULY 1969
SOURCE: PROCEEDINGS OF A NATO CONFERENCE
ABSTRACT:

ANNOTATED BIBLIOGRAPHY

- 62) TITLE: RELIABILITY MODEL DEMONSTRATION STUDY, VOLUMES I & II
AUTHOR: J. E. ANGUS, J. B. BOWEN, S. J. VANDENBERG(HUGHES AIRCRAFT)
DOC DATE: AUGUST 1983
SOURCE: ROME AIR DEVELOPMENT CENTER
ABSTRACT: This report contains the results of a study to determine the use and applicability to Air Force software acquisition managers of six quantitative software reliability models to a major command, control, communications, and intelligence CCCI system. The scope of the study included the collection of software error data from an ongoing CCCI project, fitting six software reliability models to the data, analyzing the predictions provided by the models, and developing conclusions, recommendations, and guidelines for software acquisition managers pertaining to the use and applicability of the models.
- 63) TITLE: COMMENTS ON HIGHLY RELIABLE SOFTWARE FOR AVIONICS APPLICATIONS
AUTHOR: JACOB T. SCHWARTZ
DOC DATE: SEPTEMBER 23, 1981
SOURCE: INSTITUTE FOR COMPUTER APPLICATION IN SCIENCE AND ENGINEERING
ABSTRACT: The differences between hardware and software reliability for digital systems are discussed in the context of applications where a failure may result in the loss of human life. In particular, it is argued that techniques for guaranteeing reliability for hardware are not necessarily appropriate for software. The potential of a variety of approaches for assuring software reliability is discussed.

ANNOTATED BIBLIOGRAPHY

- 64) TITLE: FITTING AN EXPONENTIAL SOFTWARE MODEL TO FIELD FAILURE DATA
AUTHOR: R. W. SCHMIDT
DOC DATE: MAY 1982
SOURCE: OFFICE OF NAVAL RESEARCH
ABSTRACT: The quantitative prediction and measurement of software reliability is of vital importance in the development of high quality cost effective software. Many software reliability models have been postulated in the literature, however few have been applied to field data. A model based upon the assumption that the failure rate of the software is proportional to the number of residual software errors leads to a constant failure rate and an exponential reliability function. The model contains two constants: the proportionality constant K and the initial (total) number of errors E_t . The constants K and E_t can be estimated during early design by comparison of the present project with historical data. During the integration test phase, a more accurate determination of the model parameters can be obtained by using simulator test data as if it were operational failure data. The simulator data is collected at two different points in the integration test phase and the two parameters can be determined from moment estimator formulas. The more powerful maximum likelihood method can also be employed to obtain point and interval estimates. It is also possible to use least squares methods to obtain parameter estimates which is the simplest method and provides insight into the analysis of the data.
This report utilizes a set of software development and field data taken by John D. Musa as a vehicle to study the ease of calculation and the correspondence of the three methods of parameter estimation. The sensitivity of the reliability predictions to parameter changes are studied and compared with field results.
The results show if data is carefully collected, software reliability models are practical and yield useful results. These can serve as one measure to help in choosing among competitive designs and as a gauge of when to terminate the integration test phase.
- 65) TITLE: IEEE COMPUTER SOCIETY SOFTWARE RELIABILITY MEASUREMENT WORKING GROUP
AUTHOR:
DOC DATE: AUGUST 1982
SOURCE:
ABSTRACT:

ANNOTATED BIBLIOGRAPHY

- 66) TITLE: SOFTWARE ACQUISITION MANAGEMENT GUIDEBOOK: SOFTWARE QUALITY ASSURANCE
AUTHOR: GEORGE NEIL AND HARVEY I. GOLD
DOC DATE: AUGUST 1977
SOURCE: ELECTRONICS SYSTEMS DIVISION
ABSTRACT: This report is one of a series of Software Acquisition Management Guidebooks which provide information and guidance for ESD Program Office personnel who are charged with planning and managing the acquisition of command, control, and communications system software procured under Air Force 800 series regulations and related software acquisition management concepts. It provides guidance for establishing and implementing a software quality assurance program which it discusses in terms of Program Office quality assurance requirements (as defined by AFR 74-1 and ESDM 74-1), contractor quality assurance requirements as defined by MIL-S-52779(AD), and software quality assurance at ESD. Special attention is given to: (1) the relationship of quality assurance to other acquisition management disciplines; (2) the integration of quality assurance requirements into the system; (4) monitoring the implementation of quality assurance requirements; and (5) common problems and proposed solutions.
- 67) TITLE: JOINT LOGISTICS COMMANDERS (JLC)
AUTHOR: MAJ. LARRY A. FRY; AIR FORCE; ANDREWS AFB
DOC DATE: 1981
SOURCE: 1981 PROCEEDINGS ANNUAL RELIABILITY AND MAINTAINABILITY SYMPOSIUM
ABSTRACT: In April 1979, the Computer Software Management (CSM) Subgroup, working under the auspices of the Joint Logistics Commanders (JLC) Joint Policy Coordinating Group for Computer Resource Management (JPCG-CRM), conducted a workshop to review current DOD policy, procedures and standards in the area of software management. Panels were organized to review the areas of software acquisition and development standards, software documentation, standards for software quality and software acceptance criteria. The findings and recommendations from this workshop are summarized in this paper.

ANNOTATED BIBLIOGRAPHY

- 68) TITLE: EVALUATING AUTOMATABLE MEASURES OF SOFTWARE DEVELOPMENT
AUTHOR: VICTOR R. BASILI AND ROBERT W. REITER, JR
DOC DATE: 1979
SOURCE: IEEE
ABSTRACT: There is a need for distinguishing a set of useful automatable measures of the software development process and product. Measures are considered useful if they are sensitive to externally observable differences in development environments and their relative values correspond to some intuition regarding these characteristic differences. Such measures could provide an objective quantitative foundation for constructing quality assurance standards and for calibrating mathematical models of software reliability and resource estimation. This paper presents a set of automatable measures that were implemented, evaluated in a controlled experiment, and found to satisfy these usefulness criteria. The measures include computer job steps, program changes, program size, and cyclomatic complexity.
- 69) TITLE: A CRITIQUE OF THE JELINSKI-MORANDA MODEL FOR SOFTWARE RELIABILITY
AUTHOR: BEV LITTLEWOOD, THE GEORGE WASHINGTON UNIVERSITY; WASHINGTON D.C. AND THE CITY UNIVERSITY; LONDON
DOC DATE: 1981
SOURCE: 1981 PROCEEDINGS ANNUAL RELIABILITY AND MAINTAINABILITY SYMPOSIUM
ABSTRACT: The paper discusses some problems associated with an early model for software reliability growth during debugging, first proposed by Jelinski and Moranda. It is suggested that the assumption of all faults contributing equally to the overall failure rate is overly naive and can be improved. Necessary and sufficient conditions are given for ML estimates of the model parameters to be finite. It is shown that there is always a finite probability that $N = \infty$. Since other authors have shown from simulated data that ML estimates can be misleading even when finite, it is important that goodness-of-fit tests are performed. Such a test on one set of data shows the model to perform badly; an alternative model due to Littlewood and Verrall is better.

ANNOTATED BIBLIOGRAPHY

- 70) TITLE: THE MANY FACETS OF QUANTITATIVE ASSESSMENT OF SOFTWARE RELIABILITY
AUTHOR: J.-C. RAULT, IRIA - INSTITUT DE RECHERCHE d'INFORMATIQUE ET d'AUTOMATIQUE DOMAINE DE VOLUCEAU , FRANCE
DOC DATE: 1979
SOURCE: IEEE
ABSTRACT: After recalling the prevailing techniques and procedures for attaining reliable and high quality software (qualitative, preventive, fault-avoidance, fault-tolerance), approaches to quantitative assessing software reliability are considered. Two main types of approach are distinguished and surveyed: extension of hardware-based techniques (application of conventional reliability theory, extension of hardware test efficiency measurement to software) and techniques specific to software (correlating structural, textual and behavioral complexity measures to some measure of reliability). Scope and domains of application of these approaches are delineated. Then, a comprehensive scheme for assessing software reliability that attempts to reconcile the various reliability models surveyed is proposed. As a conclusion, one discusses a possible incorporation of the proposed scheme into current programming stations with a view to extending their capabilities in both software project management and software quality assurance.
- 71) TITLE: TESTING FOR SOFTWARE RELIABILITY
AUTHOR: J. R. BROWN AND M. LIPOW; TRW SYSTEMS
DOC DATE:
SOURCE:
ABSTRACT: This paper presents a formulation of a novel methodology for evaluation of testing in support of operational reliability assessment and prediction. The methodology features an incremental evaluation of the representativeness of a set of development and validation test cases together with definition of additional test cases to enhance those qualities. Several techniques which permit specification of expected operational usage are described. An experimental application of the techniques to a small program is provided as an illustration of the proposed use of the methodology for operational software reliability estimation.

ANNOTATED BIBLIOGRAPHY

- 72) TITLE: DESIGN OF SELF-CHECKING SOFTWARE
AUTHOR: S. S. YAU AND R. C. CHEUNG
DOC DATE:
SOURCE: PROCEEDINGS OF INTERNATIONAL CONFERENCE ON RELIABLE SOFTWARE
ABSTRACT: This paper discussed different techniques for constructing a piece of self-checking software for systems where ultra-reliability is required. Self-checking software can be designed to detect software errors, to locate and to stop the propagation of software errors and to verify the integrity of the system.
- 73) TITLE: FAULT-TOLERANT SOFTWARE
AUTHOR: HERBERT HECHT, SENIOR MEMBER IEEE; SOHAR INC., LOS ANGELES
DOC DATE: AUGUST 1979
SOURCE: IEEE TRANSACTIONS ON RELIABILITY
ABSTRACT: Limitations in the current capabilities for verifying programs by formal proof or by exhaustive testing have led to the investigation of fault-tolerance techniques for applications where the consequence of failure is particularly severe. Two current approaches, N-version programming and the recovery block, are described. A critical feature in the latter is the acceptance test, and a number of useful techniques for constructing these are presented. A system reliability model for the recovery block is introduced, and conclusion derived from this model that affect the design of fault-tolerant software are discussed.
- 74) TITLE: ORGANIZING FOR SUCCESSFUL SOFTWARE DEVELOPMENT
AUTHOR: EDMUND B. DALY
DOC DATE: DECEMBER 1979
SOURCE: DATAMATION
ABSTRACT: Software development requires competent technologists, competent managers and an effective organization structure. A good organization structure is meaningless without a well-defined design methodology and without effective management practices. The organization structure brings together technologists and management, but the structure must work within the culture of the organization.

ANNOTATED BIBLIOGRAPHY

- 75) TITLE: MODIFIED MUSA THEORETIC SOFTWARE RELIABILITY
AUTHOR: H. B. CHENOWETH, PH.D.; WESTINGHOUSE ELECTRIC CORP.
DOC DATE: 1981
SOURCE: 1981 PROCEEDINGS ANNUAL RELIABILITY AND MAINTAINABILITY SYMPOSIUM
ABSTRACT: Presented in this paper is an analysis of the underlying assumptions and constraints relative to the Musa software execution time reliability model. The theory was examined to validate the underlying basic assumptions with the view of adding greater generality. The resulting modifications in the model theoretic analysis technique and parameter definition that result from a particular subclass of programs are explained in terms of their contribution to the predicted MTTF and the expected time to complete testing.
- 76) TITLE: REPORT ON THE DELIBERATIONS OF THE SOFTWARE RELIABILITY WORKING GROUP
AUTHOR: M. LIPOW, CHAIRMAN; TRW SYSTEMS AND ENERGY GROUP, REDONDO BEACH, CA
DOC DATE: 1979
SOURCE: IEEE
ABSTRACT: Recommendations that software and hardware reliability be considered in a systems context were presented to the Working Group. Similarities and differences of hardware and software characteristics as elements of reliability models and as they shape terminology were suggested as the main topics to be discussed.
The Working Group's primary recommendations were that a new IEEE Task Group be established to formulate and support software reliability data collection methodology, that the IEEE Terminology Task Group be supported, and that several existing software reliability models are usable, and should be applied.
- 77) TITLE: A SOFTWARE EVALUATION: RESULTS AND RECOMMENDATIONS
AUTHOR: JAN M. HOWELL; U.S. AIR FORCE; EDWARDS AFB
DOC DATE: 1983
SOURCE: 1983 PROCEEDINGS ANNUAL RELIABILITY AND MAINTAINABILITY SYMPOSIUM
ABSTRACT: This paper discusses a software quality evaluation conducted on a major U.S. Air Force avionics system. The originally planned effort was limited to a hardware evaluation but was expanded to include software when the impact of software became apparent. The paper presents some results obtained, discusses pattern software problems encountered and suggests ways to avoid those repetitive difficulties.

ANNOTATED BIBLIOGRAPHY

- 78) TITLE: IMPLEMENTATION AND MEASURABLE OUTPUT OF SOFTWARE QUALITY ASSURANCE
AUTHOR: P. CASTIGLIONE, W. THOMPSON; GENERAL ELECTRIC CO., BINGHAMTON
DOC DATE: 1983
SOURCE: 1983 PROCEEDINGS ANNUAL RELIABILITY AND MAINTAINABILITY SYMPOSIUM
ABSTRACT: This paper describes the Software Quality Assurance (SQA) controls and activities for all elements of the software design, development, and manufacturing process used at General Electric Aerospace Control Systems Department.
- 79) TITLE: WORKING GROUP ON SOFTWARE COST
AUTHOR: C. E. WALSTON, IBM, FEDERAL SYSTEMS DIVISION, BETHESDA, MD
DOC DATE: 1979
SOURCE: IEEE
ABSTRACT: This paper summarizes and highlights the discussions that took place in the working group on software cost. A number of issues were identified relating to the state of the art in software cost estimation and to its implications for developing and using software cost models.
- 80) TITLE: SOFTWARE SAFETY: A DEFINITION AND SOME PRELIMINARY THOUGHTS
AUTHOR: NANCY G. LEVESON; DEPT. OF INFORMATION AND COMPUTER SCIENCE UNIVERSITY OF CALIFORNIA; IRVINE, CA 92717
DOC DATE: APRIL 1981
SOURCE: HUGHES AIRCRAFT CO.
ABSTRACT: Software safety is the subject of a research project in its initial stages at the University of California Irvine. This research deals with critical real-time software where the cost of an error is high, e.g. human life. In this paper software techniques having a bearing on safety are described and evaluated. Initial definitions of software safety concepts are presented along with some preliminary thoughts and research questions.

ANNOTATED BIBLIOGRAPHY

- 81) TITLE: THE CURRENT STATE OF SOFTWARE RELIABILITY MODELING
AUTHOR: V. VERMURI; DEPT. OF COMPUTER SCIENCE, STATE UNIVERSITY OF NY
BINGHAMTON, NY 13901
DOC DATE: 1979
SOURCE: IEEE
ABSTRACT: The development of reliable software is a complex process. This complexity stems from its large size, strong human element in its design and development and the uncertainties associated with its operational environment. The problem of building normative models to characterize software development process, in order to predict its reliability, cost, utility, etc., is therefore an inherently complex process. The difficulties are further exacerbated by the lack of standardized practices. Furthermore, the range of validity of most of the error frequency models is confined to the immediate environment within which they were developed. Their sensitivity to parameters and test data remains to be tested. It appears, therefore, that the time is ripe for the development of an integrated approach which brings together the concepts such as complexity, reliability, utility, and cost. Within such a framework, it is possible to find a proper niche for the current crop of operational, developmental and maintenance models. Tools and techniques for such an integrated approach are available in the body of knowledge variously known as general systems theory or the theory of modeling of complex systems.

ANNOTATED BIBLIOGRAPHY

- 82) TITLE: EVOLUTION OF QUALITY/ RELIABILITY DUE TO LITIGATION
AUTHOR: RICHARD M. JACOBS, PE, CONSULTANT SERVICES INSTITUTE, INC.,
LIVINGSTON; JOHN MIHALSKY, ED. D., NEW JERSEY INSTITUTE OF
TECHNOLOGY, NEWARK
DOC DATE: 1983
SOURCE: 1983 PROCEEDINGS ANNUAL RELIABILITY AND MAINTAINABILITY
SYMPOSIUM
ABSTRACT: During the past twelve to fifteen years, manufacturers and
sellers of products have been bombarded with litigation in-
volving their product. During this period some management
groups have recognized that it takes twenty to twenty-five
times more in sales to pay for the cost of a litigation or
claim. This does not count the roughly \$10,000 of internal
costs that companies have been experiencing to service their
own litigation staff.
Included in the internal costs of retrieving information are
the following items:
failure reports, inspection records, tests records,
drawings, specifications, operating procedures, organ-
izational charts, patents, every engineering change issued
for the product, every manufacturing change, all purchase
orders, receiving tickets, stockroom records, service
records for the item involved, internal memoranda, engi-
neering note books, log books, calibration records, per-
sonnel records (for those who are associated with the
item involved).
The Quality Control and Reliability Specialist in every phase
of the operation is directly involved in the recording of the
data contained in the documents listed above and in some re-
spect becomes associated with the accuracy and the repeat-
ability of these data. In addition to the litigation re-
trieval costs, there are now laws passed by various Federal
and State Legislatures to which the companies must comply.
Evidence of this compliance most frequently originates or
passes through the hands of the Quality and Reliability
Specialist.

ANNOTATED BIBLIOGRAPHY ---

- 83) TITLE: PITFALLS OF SOFTWARE QUALITY ASSURANCE MANAGEMENT
AUTHOR: EDDIE F. THOMAS; GENERAL DYNAMICS FORT WORTH DIVISION
DOC DATE: 1983
SOURCE: 1983 PROCEEDINGS ANNUAL RELIABILITY AND MAINTAINABILITY SYMPOSIUM
ABSTRACT: This paper will address the pitfalls associated with establishing and managing a Computer Software Assurance Program (CSAP). The paper will be based on the experience gained with the CSAP Program on the General Dynamics F-16 Multi-national Staged Improvement Program (MSIP). The basic question addressed is how a CSAP can be structured so that the pitfalls can be minimized or avoided and how the productivity can be improved by directing the CSAP effort to not only satisfy the requirements of MIL-S-52779A but at the same time to help (rather than hinder) the software development effort.
- 84) TITLE: PROGRAM CONTROL COMPLEXITY AND PRODUCTIVITY
AUTHOR: J. E. GAFFNEY, JR, IBM, FEDERAL SYSTEMS DIVISION, MANASSAS, VIRGINIA 22110
DOC DATE: 1979
SOURCE: IEEE
ABSTRACT: This paper describes two simple measures of Program Control Complexity and indicates their relationship to Programmer Productivity. This work is based upon a limited amount of real program data and is related to the earlier work of McCabe and Chen.
- 85) TITLE: AN INDEX OF COMPLEXITY FOR STRUCTURED PROGRAMS
AUTHOR: IRENE L. STORM AND STANLEY PREISER, UNIVAC AND POLYTECHNIC INSTITUTE OF NEW YORK
DOC DATE: 1979
SOURCE: IEEE
ABSTRACT: An index of complexity for structured programs is introduced. It provides an "a prior" measure of program complexity, thus signaling the programmer when the program "probably" exceeds the limit of "easy" comprehension. In general, the index of complexity for structured programs is less than the index of complexity for unstructured programs.

ANNOTATED BIBLIOGRAPHY

- 86) TITLE: SWEDISH HARDWARE/SOFTWARE RELIABILITY
AUTHOR: LEIF KRISTIANSEN; ERICSSON DEFENSE AND SPACE SYSTEMS; MOELNDAL
DOC DATE: 1983
SOURCE: 1983 PROCEEDINGS ANNUAL RELIABILITY AND MAINTAINABILITY SYMPOSIUM
ABSTRACT: This paper presents the application of reliability in a defense industry. Section 1 describes significant events in the reliability area at our company. Section 2 presents Reliability Prediction, Maintainability Analysis and Reliability Growth used by our company. The last section describes a method for objective control of software production.
COMMENT:
- 87) TITLE: HARDWARE/SOFTWARE AVAILABILITY FOR A PHONE SYSTEM
AUTHOR: RICHARD PISKAR; ISKRA TELEMATIKA; KRANJ
DOC DATE: 1983
SOURCE: 1983 PROCEEDINGS ANNUAL RELIABILITY AND MAINTAINABILITY SYMPOSIUM
ABSTRACT: The paper predicts the availability of a digital telephone decentralized system composed of computer controlled switching modules and duplicated intermodule PCM switches. There are known hardware failure rates of modules and PCM switches and estimated software failure rates. The system is repairable with different repair rates for modules and PCM switches. The failure and repair rates are constant, and the distributions of times to faults are negative exponential. The problem is to determine the availability of the system. Markov process model with discrete states and continuous time is used to determine hardware and software availability. In steady state the probabilities of each state are derived from a system of simultaneous equations. Combining the probabilities of the states with the number of subscribers or lines affected, the analytical expressions for hardware and software availabilities of the system are derived. The analytical expression of general decentralized and distributed system availability and average times to fault in dedicated parts of the system are the results. The hardware/software model is offering the analytical relation between reliability and productivity, or creation of economic value during production and use of the system.

ANNOTATED BIBLIOGRAPHY

- 88) TITLE: COMBINED HARDWARE AND SOFTWARE AVAILABILITY
AUTHOR: ROBERT D. HAYNES, ARINC RESEARCH CORP., ANNAPOLIS;
WILLIAM E. THOMPSON, COLUMBIA RESEARCH CORP., ARLINGTON
DOC DATE: 1981
SOURCE: 1981 PROCEEDINGS ANNUAL RELIABILITY AND MAINTAINABILITY SYMPOSIUM
ABSTRACT: This paper presents a Bayesian availability model for combined hardware and software systems. Such systems are sometimes called embedded computer systems. The system model presented assumes that each embedded computer system malfunction can be related to one of three sources: (1) hardware, (2) software, or (3) an unknown source. The procedures presented in this paper can also serve as the basis for system specifications, warranty provisions, or other contractual agreements related to combined hardware and software system availability.
- 89) TITLE: HARDWARE-SOFTWARE AVAILABILITY: A COST BASED TRADE-OFF STUDY
AUTHOR: AMRIT L. GOEL , SYRACUSE UNIVERSITY, SYRACUSE;
JOPIE B. SOENJOTO, CENTRAL BUREAU OF STATISTICS, JAKARTA
DOC DATE: 1983
SOURCE: 1983 PROCEEDINGS ANNUAL RELIABILITY AND MAINTAINABILITY SYMPOSIUM
ABSTRACT: Software has become a major source of system malfunctions and a prime contributor to the overall cost of maintaining large commercial and weapons systems. This paper addresses the problem of assessing the reliability and availability of such systems. A cost model is developed to study the trade-off between the hardware and software subsystems for cases where such trade-offs are permissible. The basic approach follows the model developed by Goel and Soenjoto

ANNOTATED BIBLIOGRAPHY

- 90) TITLE: RELIABILITY OF SHUTTLE MISSION CONTROL CENTER SOFTWARE
AUTHOR: MARTIN L. SHOOMAN, POLYTECHNIC INSTITUTE OF NEW YORK, BROOKLYN
GEORGE RICHESON, NATA, HOUSTON
DOC DATE: 1983
SOURCE: 1983 PROCEEDINGS ANNUAL RELIABILITY AND MAINTAINABILITY SYMPOSIUM
ABSTRACT: This paper presents the results of a study made of the reliability of software for the Space Shuttle Mission Control Center Data Processing Complex. The ground based software, which is approximately 1.2 million lines of source code, was used to simulate the mission prior to flight, for use in flight controller and astronaut training. During the course of the simulation, all discrepancies from correct behavior were reported, and subsequently diagnosed as due to hardware, software, or operator errors. The model predictions are compared with the performance of this software during the first shuttle flight.
- 91) TITLE: A GUIDEBOOK FOR SOFTWARE RELIABILITY ASSESSMENT
AUTHOR: AMRIT L. GOEL, SYRACUSE UNIVERSITY
DOC DATE: AUGUST 1983
SOURCE: ROME AIR DEVELOPMENT CENTER, GRIFFISS AFB, NY
ABSTRACT: The purpose of this guidebook is to provide state-of-the-art information about the selection and use of existing software reliability models. Towards this objective, we have presented a brief summary of the available models backed by a detailed discussion of most of the models in the appendices. One of the difficulties in choosing a model is to find a match between the testing environment and a class of models. To help a user in this process, we have presented a detailed discussion of most of the assumptions that characterize the various software reliability models. The process of developing a model has been explained in detail and illustrated via numerical examples.

ANNOTATED BIBLIOGRAPHY

- 92) TITLE: COMPLEXITY MEASURES IN AN EVOLVING LARGE SYSTEM
AUTHOR: G. BENYON-TINKER, DEPT. OF COMPUTING AND CONTROL, IMPERIAL COLLEGE, LONDON
DOC DATE:
SOURCE:
ABSTRACT: One large (and functionally complicated) program has been studied in an attempt to find a measure of complexity which reflects the effort required to understand it. Examination of the program showed that a major part of the effort was related to the way in which the component procedures interacted functionally via the calling hierarchy. This suggested a new way to measure the complexity. Each of the 13 released versions of the program was analysed to determine the structure of the procedure calling hierarchy. The evolution of the complexity measure obtained indicated a modest increase in complexity of understanding which was consistent with other evidence. Procedures also interacted strongly through sharing access to global variables, and a new approach to classifying the complexity of such interactions is proposed.
- 93) TITLE: DORMANCY AND POWER ON-OFF CYCLING EFFECTS ON ELECTRONIC EQUIPMENT AND PART RELIABILITY
AUTHOR: J. A. BAUER, D. F. COTTRELL, T. R. GAGNIER, E. W. KIMBALL, et al, MARTIN MARIETTA AEROSPACE, ORLANDO, FL
DOC DATE: AUGUST 1973
SOURCE: ROME AIR DEVELOPMENT CENTER, GRIFFISS AFB, NY
ABSTRACT: Martin Marietta Aerospace has conducted two 12-month programs. The first was to collect, study, and analyze reliability information and data on dormant military electronic equipment and parts and to develop current dormant failure rates, factors, and prediction techniques. The second was to collect, study, and analyze reliability information and data on military electronic systems subjected to power on-off cycling, to correlate failure incidence with power on-off cycling, and to quantify power on-off cycling affects with respect to the dormancy and operating states.

ANNOTATED BIBLIOGRAPHY

- 94) TITLE: ASSESSMENT OF SOFTWARE RELIABILITY
AUTHOR: G. J. SCHICK AND R. W. WOLVERTON, TRWSYSTEMS ENGINEERING AND
INTEGRATION DIVISION, LOS ANGELES AND REDONDO BEACH
DOC DATE: SEPTEMBER 1972
SOURCE:
ABSTRACT: This paper discusses methods for and problems in achieving
reliability of large-scale software systems. Comparative
studies were made of a U.S. Air Force software project, a
NASA software project, and a commercial software project.
Software development and test management procedures which
lead to software reliability are analyzed. The underlying
premise is that software reliability must be designed into
the system from the outset using a systems approach. The
systems approach to achieving software reliability requires
(1) understanding of the total software development and test
life cycle, (2) identification of conventional and extended
conventional test techniques for precision validation testing
of applications programs, and (3) allocation of resources in
a cost-performance-effective manner, in advance, over the
entire development period.
- 95) TITLE: A DETERMINISTIC MODEL TO PREDICT "ERROR-FREE" STATUS OF COMPLEX
SOFTWARE DEVELOPMENT
AUTHOR: IRWIN NATHAN, SR MEMBER IEEE, XEROX CORPORATION
DOC DATE: 1979
SOURCE: IEEE
ABSTRACT: A top-down model for evaluating software "reliability" is
proposed and tested against seven case histories. The model
is shown to accurately predict when the software can be ex
pected to be reasonably "error-free". The model is based
upon the work of 19th century actuary by the name of Benjamin
Gompertz. The model was then used in a forecasting mode for
an on-going software evaluation.

ANNOTATED BIBLIOGRAPHY

- 96) TITLE: SOFTWARE RELIABILITY
AUTHOR: MARTIN L. SHOOMAN, POLYTECHNIC INSTITUTE OF NEW YORK;
MYRON LIPOW, TRW, INC., REDONDO BEACH, CA
DOC DATE:
SOURCE:
ABSTRACT: A. Basics
 . Terminology
 . Hardware - Software comparisons
 . Reliability as a quality attribute

B. Control of Software Errors
 . Sources of errors
 . Prevention and detection of errors
C. Reliability Models and Measurements
 . Complexity models
 . Empirical, structural, and statistical models
D. Applications and Case Histories
E. Questions and Discussion
- 97) TITLE: A WORKABLE SOFTWARE QUALITY/RELIABILITY PLAN
AUTHOR: ROBERT H., DUNN AND RICHARD S. ULLMAN
DOC DATE: 1978
SOURCE: 1978 PROCEEDINGS ANNUAL RELIABILITY AND MAINTAINABILITY SYMPOSIUM
ABSTRACT: The twin problems of the reliability and maintainability of embedded software are viewed as susceptible to solution by the disciplines of built-in quality assurance. Several facets to such an approach are presented. Emphasis is placed on the use of appropriate techniques and tools, with audits seen as the principal device for assuring a well-planned and orderly executed development cycle. The thrust of the paper is toward pragmatic solutions. Thus, the array of techniques and tools described is restricted to those that are readily available and have previously met with success in the development of embedded software.

ANNOTATED BIBLIOGRAPHY

- 98) TITLE: QUANTITATIVE SOFTWARE COMPLEXITY MODELS: A PANEL SUMMARY
AUTHOR: VICTOR R. BASILI, DEPT OF COMPUTER SCIENCE, UNIVERSITY OF MARYLAND
DOC DATE: 1979
SOURCE: IEEE
ABSTRACT: Several participants at the conference formed a panel on software complexity measures. The following topics were discussed:
 . Defining A Software Complexity Measure
 . Developing A Software Measure
 . Using A Software Measure
 . The Effect of Software Metrics
- 99) TITLE: AIRBORNE SYSTEMS SOFTWARE ACQUISITION ENGINEERING GUIDEBOOK FOR QUALITY ASSURANCE.
AUTHOR: M. LIPOW, TRW DEFENSE AND SPACE SYSTEMS GROUP
DOC DATE: NOVEMBER 1977
SOURCE: AERONAUTICAL SYSTEMS DIVISION, WRIGHT PATTERSON AFB, OHIO
ABSTRACT: This report is one of a series of guidebooks which provide guidance for ASD and SAMSO Program Office and engineering personnel in the acquisition management and engineering of Airborne Systems software procured under Air Force 800 series regulations. It provides information that will help personnel plan, specify, and monitor quality assurance activities in connection with the acquisition of Computer Program Configuration Items (CPCI's) for Airborne Systems.
- 100) TITLE: ELEMENTS OF SOFTWARE SCIENCE
AUTHOR: MAURICE H. HALSTEAD
DOC DATE: 1977
SOURCE:
ABSTRACT: This book contains the first systematic summarization of a branch of experimental and theoretical science dealing with the human preparation of computer programs and other types of written material. Application of the classical methods of the natural sciences demonstrates that even such relatively intangible objects as written abstracts and computer programs are governed by natural laws, both in their preparation and in their ultimate form.
The work underlying each chapter of this monograph is firmly based on the methods and principles of classical experimental science. Even so, the results in this area, or more specifically, the concept that significant quantitative results are attainable in such an area, are sufficiently counter-intuitive as to appear almost weird.

ANNOTATED BIBLIOGRAPHY

- 101) TITLE: WHEN AND HOW TO USE A SOFTWARE RELIABILITY MODEL
AUTHOR: AMRIT L. GOEL, VICTOR R. BASILI, AND PETER M. VALDES
DOC DATE: DECEMBER 1982
SOURCE: GODDARD SPACE FLIGHT CENTER
ABSTRACT: Many analytical models were proposed during the last decade for software reliability assessment. These models served a useful purpose in identifying the need for an objective approach to determining the quality of a software system as it goes through various stages of development. However, by and large, these models have not been as widely and convincingly used as was expected.
In this paper we attempt to identify the causes of this state of affairs and suggest some remedial actions. For example, we feel that very often the models are used without a clear understanding of their underlying assumptions and limitations. Also, there seems to be some misunderstanding about the interpretations of model inputs and outputs. To overcome some of these difficulties, we provide a classification of the available models and suggest which types of models are applicable in a given phase of the software development cycle.
- 102) TITLE: GUIDE FOR MANAGING NONDELIVERABLE COMPUTER RESOURCES
AUTHOR: MAJ. GEORGE W. TREVER, AFCMD/EPER, KIRKLAND AFB
DOC DATE: 24 JAN 1984
SOURCE: RECEIVED THROUGH NSIA
ABSTRACT: (OBJECTIVES) This guide was prepared to assist managers structure and understand policies and procedures in the management of nondeliverable computer resources (NDCR). It is the intent of this guide to capture the basic management methods and lessons learned attributed to deliverable embedded computer systems and translate them into management terms appropriate for NDCR.
- 103) TITLE: METHODOLOGY FOR SOFTWARE AND SYSTEM RELIABILITY PREDICTION
PHASE II INTERIM REPORT
AUTHOR: J. McCALL, et al.
DOC DATE: MARCH 1985
SOURCE: RADC F30602-83-C-0118
ABSTRACT: The purpose of this report is to describe the interim results of a research and development effort to develop a methodology for predicting and estimating software reliability. This report represents interim findings during Phase II of the project. This effort was performed under Contract Number F30602-83-C-0118 for the U.S. Air Force Rome Air Development Center (RADC).

APPENDIX C
PILOT SURVEY FORM & INSTRUCTIONS

LEGEND: A. SW Specification Errors
 B. SW Design Errors
 C. SW Coding Errors
 D. SW Interface Problems
 E. HW Interface Problems
 F. Human Interface Problems
 G. Capacity Problems

RATING: (blank) No Opinion
 0 No Correlation
 L Low Correlation
 M Medium Correlation
 H High Correlation

FACTOR/CHARACTERISTIC/TECHNIQUE	Sys.Rel Impact	A	B	C	D	E	F	G
OPERATIONAL REQUIREMENTS								
Predominantly Control								
Predominantly Computational								
Predominantly Input/Output								
Predominantly Real-Time								
Predominantly Interactive								
ENVIRONMENTAL REQUIREMENTS								
Number of Hardware Interfaces								
Number of Software Interfaces								
Number of Human Interfaces								
SIZE CONSIDERATIONS								
Number of Functions Performed								
Overall Program Size								
Number of Compilation Units								
Maximum size per unit								
COMPLEXITY CONSIDERATIONS								
Number of Entries and Exits								
Number of Control Variables								
Use of Single-Function Modules								
Number of Modules								
Maximum Module Size								
Hierarchical Control between Module								
Logical Coupling between Modules								
Data Coupling between Modules								

LEGEND: A. SW Specification Errors
 B. SW Design Errors
 C. SW Coding Errors
 D. SW Interface Problems
 E. HW Interface Problems
 F. Human Interface Problems
 G. Capacity Problems

RATING: (blank) No Opinion
 0 No Correlation
 L Low Correlation
 M Medium Correlation
 H High Correlation

FACTOR/CHARACTERISTIC/TECHNIQUE	Sys.Rel Impact	A	B	C	D	E	F	G
ORGANIZATIONAL CONSIDERATIONS								
Separate Design and Coding								
Independent Test Organization								
Independent Quality Assurance								
Independent Configuration Control								
Independent Verification/Validation								
Programming Team Structure								
Educational Level of Team Members								
Experience Level of Team Members								
METHODS USED								
Definition/Enforcement of Standards								
Use of High Order Language (HOL)								
Formal Reviews (PDR,CDR,etc)								
Frequent Walkthroughs								
Top-Down & Structured Approaches								
Unit Development Folders								
Software Development Library								
Formal Change & Error Reporting								
Progress & Status Reporting								
DESIGN APPROACH								
Modular Construction								
Structured Design								
Structured Code								

LEGEND: A. SW Specification Errors
 B. SW Design Errors
 C. SW Coding Errors
 D. SW Interface Problems
 E. HW Interface Problems
 F. Human Interface Problems
 G. Capacity Problems

RATING: (blank) No Opinion
 0 No Correlation
 L Low Correlation
 M Medium Correlation
 H High Correlation

FACTOR/CHARACTERISTIC/TECHNIQUE	Sys.Rel Impact	A	B	C	D	E	F	G
TOOLS USED								
Flow Charts								
Structure Charts								
Decision Tables								
HIPO Charts								
DOCUMENTATION								
System Requirements Spec								
Software Requirements Spec								
Interface Design Spec								
Software Design Spec								
Source Listings								
Test Plans, Procedures & Reports								
S/W Development Plan								
S/W Quality Assurance Plan								
S/W Configuration Management Plan								
Requirements Traceability Matrix								
Version Description Document								
Software Discrepancy Reports								

LEGEND: A. SW Specification Errors
 B. SW Design Errors
 C. SW Coding Errors
 D. SW Interface Problems
 E. HW Interface Problems
 F. Human Interface Problems
 G. Capacity Problems

RATING: (blank) No Opinion
 0 No Correlation
 L Low Correlation
 M Medium Correlation
 H High Correlation

FACTOR/CHARACTERISTIC/TECHNIQUE	Sys.Rel Impact	A	B	C	D	E	F	G
DUTY CYCLE								
Constant Mission Usage								
Periodic Mission Usage								
Infrequent Mission Usage								
ENVIRONMENT								
Variability of Hardware								
Training Level of Operators								
Variability of Input Data								
Variability of Outputs								
Degree of Human Interaction								
NON-OPERATIONAL USAGE								
Training Exercises								
Periodic Self Test								
Built-in Diagnostics								
MODIFICATION / ERROR CORRECTION								
Performed in the Field								
Performed at Depot								
Performed at Factory								

LEGEND: A. SW Specification Errors
 B. SW Design Errors
 C. SW Coding Errors
 D. SW Interface Problems
 E. HW Interface Problems
 F. Human Interface Problems
 G. Capacity Problems

RATING: (blank) No Opinion
 0 No Correlation
 L Low Correlation
 M Medium Correlation
 H High Correlation

FACTOR/CHARACTERISTIC/TECHNIQUE	Sys. Rel Impact	A	B	C	D	E	F	G
QUALITATIVE CHARACTERISTICS								
Correctness								
Validity								
Generality								
Testability								
Efficiency / Economy								
Resilience (Robustness)								
Useability								
Fault Tolerance								
Clarity								
Readability								
Maintainability								
Modifiability								
Flexibility								
Portability								
Reusability								
Interoperability								
OTHER (List as necessary)								

* * * I N S T R U C T I O N S * * *

The ultimate objective of this effort is to develop characterization techniques for predicting TOTAL SYSTEM RELIABILITY BASED ON MISSION OR OPERATING TIME. The techniques must include the combined effects of BOTH HARDWARE AND SOFTWARE as well as the ENVIRONMENT in which the system operates. The purpose of this initial survey is to identify those factors and characteristics which affect the SOFTWARE component of the overall system reliability. Specifically, the survey addresses:

SYSTEM SOFTWARE RELIABILITY - The probability that the required software will perform its intended functions for the prescribed mission(s) and time period(s) in the specified operating environment, without causing system outage or failure.

It should be noted that the above definition does not specifically address "qualities" of the software other than its ability to perform as specified without causing overall system outage. For example, it is possible for very inefficient software to be very reliable. The intent of the survey is to correlate qualitative and quantitative factors. THE RATING THAT YOU ARE ASKED TO PROVIDE SHOULD BE BASED ON THE FACTOR'S RELATIONSHIP TO RELIABILITY, NOT ON ITS IMPORTANCE TO NON-OPERATIONAL QUALITIES.

Horizontally, the survey includes an overall category and several groupings of categories which are typically used to record error content of computer programs. In the case of the error categories, considerable amount of QUANTITATIVE data is available from past projects. Quantifying the overall system reliability is the goal of this study.

Vertically, the survey includes a mixed list of intrinsic factors, design and development methodologies, techniques and operational characteristics which are generally well covered QUALITATIVELY in the literature.

You are asked to relate the rows and the columns by marking those blocks where you feel there is a cause/effect relationship indicating the degree of correlation by using the codes provided. It should be noted that some of the vertical entries such as "structured design", are generally assumed to have a positive effect on reliability by reducing design errors. Others such as "predominately real-time" are thought to decrease reliability by increasing complexity. In this survey, you are not asked to analyze the type of correlation, but simply indicate the degree of correlation.

Although this survey will not provide quantitative effects of the qualitative factors, it is expected that it will indicate a strong relationship between specific factors and specific error types which in turn, can be related to total error content of a program. This total error content can then be combined with the operational "duty cycling" of the program to yield a quantitative prediction of its reliability.

Definitions which are unique or special to this survey are presented herein. A general glossary of software terms and definitions is included separately.

SPECIAL TERMINOLOGY

In order to make this survey as concise as possible and to insure consistent interpretation of the terminology used, the following definitions are presented:

SYSTEM SOFTWARE RELIABILITY - The probability that the required software will perform its intended functions for the prescribed mission(s) and time period(s) in the specified operating environment, without causing system outage or failure.

SW SPECIFICATION ERRORS - These include all errors resulting from missing, incorrect or misunderstood requirements. Include in this category any problems associated with the communication of requirements between the contractor and the developer.

SW DESIGN ERRORS - These are errors which occur in the logical implementation of the required function. Include in this category all errors where a required function is not included, not invoked, not checked, not complete or does not produce correct results due to logical construction.

SW CODING ERRORS - These are computational or calculation errors. Included are: errors of omission such as uninitialized variables; mathematical errors such as incorrect expressions, conversion and truncation errors; and programming errors such as improper use of indices, variables and overlays. Since, this survey is addressed to operational reliability, DO NOT INCLUDE SYNTAX ERRORS that would be eliminated prior to operation.

SW INTERFACE PROBLEMS - This category addresses all errors which occur between software components of the system such as when one program unit fails to call, calls in the wrong sequence, or otherwise improperly calls another program unit. It also includes all errors resulting from the improper sharing or passing of data and/or control variables between program units. Examples include: passing wrong arguments, mismatched scale factors, missing arguments, etc.

HW INTERFACE PROBLEMS - This category includes all errors which result in loss of data or untimely exchange of data between system hardware and embedded software. Examples include situations where buffers become saturated or computation cycles exceed their timing allocations. Also included are errors caused by improper data exchange between system hardware and embedded software. Examples include: missing data, incorrect data, mismatched scales, etc.

HUMAN INTERFACE PROBLEMS - This category includes all operator errors that are not corrected or compensated for by the computer software. Examples include the acceptance of improper commands and data and the rejection of proper inputs. Also included are output errors which result in human interface problems such as missing output, incorrect output, ambiguous output, etc.

CAPACITY PROBLEMS - This category includes all errors where the system performs all of its operational tasks but not within its required timing constraints or only performs them on a subset of its intended input domain, i.e., there are situations where it doesn't work due to the quantity or content of its tasks. For example, buffers that become saturated during periods of high activity.

APPENDIX D
PILOT SURVEY RESULTS

Thirty surveys were distributed and 23 were returned. Each row of the survey represented a software factor or characteristic which affects system reliability. The first column of the survey was entitled System Reliability Impact, and the remaining columns represented the following error categories:

- A: SW Specification Errors
- B: SW Design Errors
- C: SW Coding Errors
- D: SW Interface Problems
- E: HW Interface Problems
- F: Human Interface Problems
- G: Capacity Problems

The participants were asked to assign a rating to each block indicating the degree of correlation between the factors and the error categories. If a participant had no opinion, the block was left blank.

Each of the four possible ratings was given a numerical value as shown below.

<u>Rating</u>	<u>Numerical Value</u>
H (High Correlation)	9.0
M (Medium Correlation)	5.0
L (Low Correlation)	1.0
O (No Correlation)	0.0

A blank meant that the respondee had no opinion on that particular item and so blanks were ignored in the analysis. Using the numerical values for the ratings, overall averages were calculated for each block of the survey; i.e., for each row and column combination. Individual averages for each row were computed as well as overall row averages. Although numerical averages were computed for the qualitative and other factors listed on the last page of the survey, they were not included in the overall rankings.

Table D-1 shows the top twenty software factors ranked in decreasing order of overall row average. It is obvious that specifications, application types and design methodologies were determined to be the leading influences on system reliability. Table D-2 gives the overall average, the number of responses and the standard deviation for each block of the survey.

TABLE D-1. TWENTY HIGHEST RANKED SOFTWARE ATTRIBUTES / FACTORS

<u>Row Average</u>	<u>Attribute / Factor</u>
7.91	Interface Design Spec
7.62	Software Requirements Spec
7.41	System Requirements Spec
7.40	Frequent Walkthroughs
7.14	Software Design Spec
6.83	Definition/Enforcement of Standards
6.46	Predominantly Real-Time
6.40	Experience Level of Team Members
6.38	Predominantly Control
6.31	Top-Down & Structured Approaches
6.24	Test Plans, Procedures & Reports
6.21	Predominantly Interactive
6.13	Formal Reviews (PDR,CDR,etc)
6.12	Requirements Traceability Matrix
6.10	Modular Construction
6.05	Performed in the Field
5.98	Structured Design
5.91	Constant Mission Usage
5.87	Number of Software Interfaces
5.83	S/W Development Plan

TABLE D-2. STATISTICAL RESULTS OF PILOT SURVEY

SW FACTOR/CHARACTERISTIC		Sys. Rel. Impact							
			A	B	C	D	E	F	G
Predominantly Control	Average	6.9	7.7	7.0	5.0	7.0	5.9	6.3	4.7
	No. Resp.	19.	18.	18.	15.	14.	15.	12.	14.
	Std. Dev.	2.8	2.4	3.1	3.7	3.0	3.9	3.1	3.9
Predominantly Computational	Average	4.1	5.9	6.9	5.6	4.6	3.3	1.7	3.3
	No. Resp.	18.	16.	17.	13.	12.	12.	13.	13.
	Std. Dev.	3.0	3.5	2.5	3.6	3.3	3.8	2.5	3.6
Predominantly Input/Output	Average	5.2	7.0	6.3	5.3	6.5	6.7	3.6	4.8
	No. Resp.	18.	16.	15.	14.	15.	16.	13.	13.
	Std. Dev.	3.2	3.3	3.3	3.3	3.1	2.7	3.2	3.1
Predominantly Real-Time	Average	7.0	6.3	6.9	5.9	6.7	6.2	4.9	7.1
	No. Resp.	20.	15.	15.	13.	14.	15.	11.	15.
	Std. Dev.	2.8	3.4	3.0	3.7	3.0	3.5	3.7	2.6
Predominantly Interactive	Average	6.1	7.5	6.8	5.3	6.1	5.1	7.9	4.2
	No. Resp.	18.	13.	13.	13.	15.	14.	18.	12.
	Std. Dev.	3.0	2.6	3.1	3.4	3.2	3.9	2.7	3.1
Number of Hardware Interfaces	Average	6.5	4.9	5.4	3.2	5.5	8.6	3.5	4.2
	No. Resp.	21.	14.	16.	14.	13.	19.	15.	14.
	Std. Dev.	3.0	3.6	3.0	3.1	3.7	1.3	3.1	3.4
Number of Software Interfaces	Average	7.7	7.4	7.4	5.0	8.6	3.0	2.6	3.3
	No. Resp.	21.	15.	15.	14.	19.	14.	14.	14.
	Std. Dev.	2.3	2.5	2.5	3.1	1.3	3.3	2.2	3.3
Number of Human Interfaces	Average	6.3	5.5	5.5	3.2	3.8	3.2	7.8	2.9
	No. Resp.	19.	15.	15.	14.	14.	13.	17.	13.
	Std. Dev.	3.0	3.3	3.3	3.1	3.0	2.9	2.7	2.9
Number of Functions Performed	Average	6.9	6.9	6.7	5.0	7.0	3.7	3.4	4.9
	No. Resp.	21.	17.	19.	16.	16.	15.	15.	15.
	Std. Dev.	2.4	2.9	2.8	3.3	2.1	3.5	3.2	2.9
Overall Program Size	Average	5.9	5.2	6.1	5.6	5.5	2.3	2.0	6.6
	No. Resp.	20.	15.	18.	16.	15.	14.	14.	15.
	Std. Dev.	3.3	3.6	2.8	2.9	3.8	3.3	2.8	2.0
Number of Compilation Units	Average	4.9	4.5	4.6	4.7	5.9	1.6	1.6	2.9
	No. Resp.	18.	15.	18.	16.	16.	14.	14.	14.
	Std. Dev.	3.2	3.9	3.1	3.5	3.3	2.7	2.7	3.7

SW FACTOR/CHARACTERISTIC		Sys. Rel. Impact	A	B	C	D	E	F	G
Maximum size per unit	Average	5.9	4.5	5.1	5.5	5.2	2.9	2.1	3.8
	No. Resp.	17.	14.	15.	15.	13.	14.	13.	13.
	Std. Dev.	3.4	3.9	3.4	3.1	3.3	3.4	2.9	3.6
Number of Entries and Exits	Average	7.7	5.0	7.3	6.7	8.4	3.1	2.9	2.2
	No. Resp.	18.	15.	19.	19.	19.	15.	14.	13.
	Std. Dev.	2.4	4.2	2.6	2.8	1.5	3.6	3.7	3.3
Number of Control Variables	Average	6.8	5.6	5.6	6.5	7.8	2.8	3.3	2.7
	No. Resp.	18.	14.	17.	16.	17.	14.	13.	14.
	Std. Dev.	2.5	3.9	3.0	2.9	1.9	3.2	3.2	2.9
Use of Single-Function Modules	Average	6.8	5.1	7.1	5.6	6.6	2.6	2.4	1.7
	No. Resp.	18.	15.	17.	17.	15.	13.	13.	13.
	Std. Dev.	3.1	3.7	3.0	3.0	2.9	2.8	3.4	3.3
Number of Modules	Average	3.5	4.8	6.8	4.6	6.4	2.4	1.9	2.9
	No. Resp.	17.	13.	17.	14.	14.	14.	13.	14.
	Std. Dev.	3.2	3.9	3.0	3.4	3.4	3.3	2.7	3.0
Maximum Module Size	Average	5.7	4.9	6.5	6.2	4.6	2.2	2.1	3.4
	No. Resp.	18.	13.	18.	17.	14.	14.	13.	14.
	Std. Dev.	2.8	4.6	2.9	3.1	3.5	3.4	3.4	3.2
Hierarchical Control Between Modules	Average	5.7	4.2	6.6	5.0	7.0	2.0	2.8	1.2
	No. Resp.	18.	13.	15.	13.	16.	13.	13.	13.
	Std. Dev.	3.1	3.5	2.9	3.3	3.3	3.0	3.4	2.4
Logical Coupling Between Modules	Average	6.5	4.4	7.1	5.9	7.4	2.2	2.3	2.3
	No. Resp.	16.	13.	15.	14.	17.	13.	13.	13.
	Std. Dev.	3.2	4.2	3.0	3.2	2.5	2.9	2.8	3.5
Data Coupling between Modules	Average	6.1	4.4	7.8	6.9	7.2	2.8	2.5	1.7
	No. Resp.	18.	14.	16.	15.	18.	15.	13.	13.
	Std. Dev.	2.7	4.3	2.4	3.0	2.5	3.3	3.0	2.8
Separate Design and Coding	Average	3.8	3.6	6.3	5.8	5.3	3.8	3.3	2.0
	No. Resp.	17.	12.	18.	16.	14.	12.	12.	10.
	Std. Dev.	3.5	3.2	3.1	3.3	2.9	3.6	2.8	2.1
Independent Test Organization	Average	6.7	5.8	6.5	6.0	5.5	5.3	5.8	3.0
	No. Resp.	19.	13.	15.	19.	15.	12.	13.	11.
	Std. Dev.	3.1	3.9	3.4	3.3	3.4	3.3	3.5	2.9
Independent Quality Assurance	Average	7.0	6.2	5.7	5.9	4.9	4.8	6.1	2.2
	No. Resp.	18.	13.	16.	16.	14.	12.	14.	11.
	Std. Dev.	2.8	3.6	3.4	3.5	3.6	3.6	3.4	2.9

SW FACTOR/CHARACTERISTIC		Sys. Rel. Impact	A	B	C	D	E	F	G
Independent Configuration Control	Average	5.7	5.5	5.5	4.4	4.6	4.5	4.5	1.6
	No. Resp.	17.	13.	14.	15.	14.	11.	13.	10.
	Std. Dev.	2.9	3.8	3.6	3.7	3.5	3.4	4.0	1.8
Independent Verification and Validation	Average	6.1	6.3	6.6	6.3	5.7	5.2	5.8	2.3
	No. Resp.	18.	12.	15.	15.	15.	12.	13.	10.
	Std. Dev.	3.3	3.3	3.3	3.6	3.6	3.4	3.5	3.0
Programming Team Structure	Average	5.4	3.6	5.2	5.7	4.1	2.2	3.3	2.0
	No. Resp.	18.	14.	17.	17.	15.	11.	12.	10.
	Std. Dev.	3.0	3.9	3.0	2.9	2.5	2.3	2.8	2.1
Educational Level of Team Members	Average	4.5	4.3	5.2	3.8	4.9	3.8	4.5	2.4
	No. Resp.	17.	14.	17.	16.	14.	12.	13.	10.
	Std. Dev.	2.5	3.3	3.3	3.2	2.8	3.2	3.3	2.3
Experience Level of Team Members	Average	7.9	6.1	7.4	6.0	6.6	5.3	5.8	4.9
	No. Resp.	18.	16.	17.	16.	14.	12.	13.	10.
	Std. Dev.	1.8	3.7	2.8	3.4	2.8	2.8	3.1	3.9
Definition/Enforcement of Standards	Average	7.3	6.8	7.1	7.4	7.8	7.2	6.5	3.1
	No. Resp.	19.	13.	17.	17.	14.	13.	13.	10.
	Std. Dev.	2.4	3.5	2.5	2.5	2.7	2.6	3.5	3.1
Use of High Order Language (HOL)	Average	6.9	3.1	3.5	8.4	5.8	4.0	2.2	4.4
	No. Resp.	19.	14.	15.	19.	15.	13.	13.	12.
	Std. Dev.	2.8	4.1	3.7	2.0	3.8	3.8	2.8	3.5
Formal Reviews (PDR,CDR,etc)	Average	5.4	7.8	6.9	5.1	6.4	7.1	6.6	3.4
	No. Resp.	21.	17.	19.	16.	16.	13.	14.	12.
	Std. Dev.	3.1	2.4	2.8	4.0	3.5	3.3	3.7	3.4
Frequent Walkthroughs	Average	7.5	7.7	8.2	7.6	8.0	7.5	7.5	4.3
	No. Resp.	19.	15.	19.	18.	17.	13.	14.	12.
	Std. Dev.	2.4	2.5	2.1	3.2	2.4	2.0	2.7	4.0
Top-Down & Structured Approaches	Average	6.8	6.8	7.4	7.1	7.9	5.2	5.2	2.6
	No. Resp.	20.	16.	18.	17.	15.	13.	13.	12.
	Std. Dev.	2.7	2.9	2.8	2.9	2.4	3.3	3.7	3.5
Unit Development Folders	Average	6.2	4.2	6.9	7.0	6.3	4.9	5.0	2.8
	No. Resp.	17.	15.	16.	16.	15.	13.	13.	13.
	Std. Dev.	2.4	3.6	2.7	2.5	2.5	3.4	3.3	3.0
Software Development Library	Average	6.5	5.2	6.1	7.2	6.2	3.9	3.6	2.3
	No. Resp.	16.	12.	14.	16.	13.	12.	11.	12.
	Std. Dev.	3.2	4.2	3.0	3.4	3.6	3.6	3.9	2.5

SW FACTOR/CHARACTERISTIC		Sys. Rel. Impact	A	B	C	D	E	F	G
Formal Change & Error Reporting	Average	6.5	5.5	5.7	5.9	5.5	4.6	4.7	2.1
	No. Resp.	16.	13.	15.	16.	13.	12.	13.	13.
	Std. Dev.	2.0	3.8	3.2	3.5	2.9	2.8	3.0	2.9
Progress & Status Reporting	Average	4.7	3.9	5.0	4.7	4.6	3.9	4.6	1.5
	No. Resp.	15.	13.	16.	16.	11.	11.	11.	12.
	Std. Dev.	3.6	3.9	3.9	3.8	3.3	3.1	3.3	2.2
Modular Construction	Average	8.4	4.2	8.1	7.4	8.2	4.3	4.1	1.8
	No. Resp.	19.	14.	17.	15.	15.	12.	13.	12.
	Std. Dev.	2.0	4.0	2.7	2.9	2.2	3.0	3.6	2.0
Structured Design	Average	8.0	3.9	7.9	7.1	7.3	4.9	3.6	2.3
	No. Resp.	20.	15.	19.	15.	14.	11.	11.	12.
	Std. Dev.	2.2	4.1	2.6	2.6	2.6	3.2	3.4	2.9
Structured Code	Average	7.7	3.2	4.8	8.1	7.3	4.5	3.4	2.2
	No. Resp.	19.	14.	15.	18.	14.	12.	12.	13.
	Std. Dev.	2.3	4.0	3.9	2.2	3.0	3.8	3.4	2.8
Flow Charts	Average	4.6	4.1	5.9	6.1	6.8	4.6	3.2	2.4
	No. Resp.	18.	14.	13.	15.	13.	11.	12.	11.
	Std. Dev.	3.3	4.1	2.4	2.8	2.6	2.8	2.9	3.1
Structure Charts	Average	6.0	5.6	7.0	5.7	6.7	3.8	3.5	2.0
	No. Resp.	16.	11.	14.	12.	12.	10.	11.	11.
	Std. Dev.	3.1	3.6	2.1	3.3	2.7	3.3	3.8	3.0
Decision Tables	Average	5.6	5.7	6.7	5.7	6.2	3.5	2.9	2.2
	No. Resp.	13.	10.	12.	12.	10.	11.	10.	10.
	Std. Dev.	2.2	3.3	2.7	3.3	3.3	2.8	2.9	3.1
HIPO Charts	Average	5.5	6.0	7.3	5.7	6.1	4.2	3.1	2.4
	No. Resp.	16.	11.	14.	12.	11.	10.	11.	11.
	Std. Dev.	2.5	2.8	2.1	3.3	3.1	3.2	3.4	3.6
System Requirement Spec	Average	7.7	8.8	7.9	5.6	7.7	8.0	7.4	5.5
	No. Resp.	19.	18.	15.	13.	12.	12.	13.	13.
	Std. Dev.	2.3	0.9	2.4	3.6	2.6	2.5	2.8	3.8
Software Requirement Spec	Average	7.8	9.0	8.5	6.9	8.7	5.9	7.4	5.6
	No. Resp.	17.	17.	16.	15.	14.	12.	13.	11.
	Std. Dev.	2.4	0.0	1.4	3.3	1.1	3.6	2.8	3.6
Interface Design Spec	Average	8.3	7.7	8.8	7.0	8.7	9.0	8.0	4.8
	No. Resp.	18.	15.	17.	14.	15.	13.	13.	11.
	Std. Dev.	1.5	2.5	1.0	3.4	1.0	0.0	2.6	3.8

SW FACTOR/CHARACTERISTIC		Sys. Rel. Impact	A	B	C	D	E	F	G
Software Design Spec	Average	7.4	7.5	8.3	7.1	7.6	6.3	6.4	5.8
	No. Resp.	18.	15.	18.	15.	15.	12.	13.	12.
	Std. Dev.	2.9	3.2	2.3	3.1	2.7	2.8	3.4	3.7
Source Listings	Average	4.2	3.8	5.8	7.4	5.5	3.0	3.8	2.2
	No. Resp.	17.	13.	13.	17.	13.	11.	12.	11.
	Std. Dev.	3.3	4.0	3.9	2.8	3.3	2.9	4.1	2.9
Test Plans, Procedures & Reports	Average	7.3	6.4	6.7	6.3	6.4	5.6	5.4	5.2
	No. Resp.	19.	14.	14.	18.	14.	13.	14.	13.
	Std. Dev.	2.0	3.5	3.0	3.1	2.5	3.2	3.3	3.2
S/W Development Plan	Average	6.5	6.3	6.8	6.8	6.7	4.3	5.2	2.7
	No. Resp.	21.	15.	16.	18.	14.	12.	13.	12.
	Std. Dev.	3.0	3.3	2.5	2.5	2.6	3.4	3.7	3.4
S/W Quality Assurance Plan	Average	5.4	5.4	6.3	5.8	5.5	3.9	5.1	2.5
	No. Resp.	18.	15.	15.	16.	15.	13.	15.	13.
	Std. Dev.	3.1	3.9	3.3	3.3	3.3	3.1	3.7	3.8
S/W Configuration Management Plan	Average	5.8	4.6	5.2	5.2	4.9	3.0	3.8	1.9
	No. Resp.	16.	14.	13.	15.	13.	11.	12.	12.
	Std. Dev.	2.6	3.5	3.6	3.6	3.4	2.9	3.2	2.9
Requirements Traceability Matrix	Average	7.2	8.3	7.5	4.9	6.3	4.6	5.5	3.0
	No. Resp.	18.	17.	16.	13.	12.	12.	12.	13.
	Std. Dev.	2.5	1.6	2.0	3.4	2.0	3.3	3.6	3.9
Version Description Document	Average	3.6	4.8	4.3	4.8	3.8	2.8	3.5	1.6
	No. Resp.	15.	13.	12.	13.	13.	12.	12.	12.
	Std. Dev.	2.6	3.8	3.4	3.8	3.0	2.9	3.7	2.7
Software Discrepancy Reports	Average	6.3	5.2	6.3	7.1	5.6	4.7	4.9	3.5
	No. Resp.	18.	13.	14.	15.	13.	13.	13.	13.
	Std. Dev.	3.1	4.0	3.3	2.6	3.6	3.4	3.8	4.0
Constant Mission Usage	Average	6.7	6.3	6.3	5.6	6.1	7.3	5.3	2.8
	No. Resp.	18.	12.	12.	12.	11.	12.	12.	10.
	Std. Dev.	3.3	3.7	3.1	3.5	3.6	2.7	3.3	3.6
Periodic Mission Usage	Average	4.9	4.8	4.5	3.5	4.6	5.3	3.7	1.9
	No. Resp.	16.	11.	11.	11.	10.	12.	11.	10.
	Std. Dev.	2.6	3.4	3.4	2.2	2.3	2.7	2.8	2.9
Infrequent Mission Usage	Average	4.7	3.8	4.3	3.2	3.4	4.7	3.5	1.5
	No. Resp.	18.	12.	12.	12.	10.	12.	12.	10.
	Std. Dev.	3.9	4.0	3.8	3.3	3.4	3.6	3.7	2.7

SW FACTOR/CHARACTERISTIC		Sys. Rel. Impact	A	B	C	D	E	F	G
Variability of Hardware	Average	5.9	4.8	4.4	2.4	2.8	8.0	4.6	1.4
	No. Resp.	17.	9.0	10.	10.	9.0	12.	10.	7.0
	Std. Dev.	2.7	4.3	3.7	3.0	2.1	1.8	3.0	1.6
Training Level of Operators	Average	5.4	2.5	3.4	2.3	1.6	2.9	6.3	1.1
	No. Resp.	18.	11.	10.	11.	10.	9.0	12.	8.0
	Std. Dev.	2.7	3.8	3.2	3.2	2.4	3.2	3.3	1.6
Variability of Input Data	Average	6.3	4.1	5.6	4.5	6.0	3.8	5.7	3.1
	No. Resp.	18.	12.	11.	12.	12.	11.	11.	9.0
	Std. Dev.	2.7	3.6	3.6	3.8	2.5	3.7	3.0	3.6
Variability of Outputs	Average	4.1	3.8	4.9	4.5	4.9	3.8	4.6	0.9
	No. Resp.	16.	12.	10.	12.	11.	10.	11.	8.0
	Std. Dev.	3.5	3.7	3.9	3.8	3.7	3.3	2.8	0.4
Degree of Human Interaction	Average	7.0	3.6	4.0	2.8	3.3	4.6	7.8	0.6
	No. Resp.	18.	11.	10.	12.	10.	9.0	16.	8.0
	Std. Dev.	2.5	3.4	3.4	2.9	2.2	2.4	1.9	0.5
Training Exercises	Average	3.8	2.1	2.2	2.5	2.1	2.4	3.8	0.3
	No. Resp.	17.	11.	10.	11.	10.	11.	12.	9.0
	Std. Dev.	2.4	3.4	3.1	3.0	3.1	3.6	3.6	0.5
Periodic Self Test	Average	5.9	4.3	4.9	3.3	3.6	3.4	2.1	0.6
	No. Resp.	16.	10.	9.0	10.	9.0	11.	9.0	9.0
	Std. Dev.	2.9	3.9	3.6	3.5	3.6	3.4	2.2	0.5
Built-in Diagnostics	Average	7.3	4.0	6.0	5.3	5.3	4.8	4.4	1.4
	No. Resp.	19.	11.	11.	12.	10.	12.	10.	9.0
	Std. Dev.	2.6	3.3	3.3	3.7	4.1	3.6	4.2	2.9
Performed in the Field	Average	6.4	6.4	6.5	5.8	6.3	6.7	6.2	3.3
	No. Resp.	16.	8.0	8.0	10.	9.0	9.0	9.0	7.0
	Std. Dev.	3.7	3.9	3.0	3.2	2.8	3.2	3.7	3.1
Performed at Depot	Average	6.1	4.9	4.4	5.0	4.9	5.4	4.9	1.7
	No. Resp.	14.	8.0	7.0	8.0	8.0	8.0	8.0	6.0
	Std. Dev.	3.0	3.9	3.6	3.0	3.9	4.1	3.9	1.6
Performed at Factory	Average	5.5	5.4	4.0	4.6	4.0	4.4	4.9	2.1
	No. Resp.	15.	8.0	8.0	10.	9.0	9.0	9.0	7.0
	Std. Dev.	4.1	4.1	3.5	3.5	4.0	3.8	4.1	2.0
Correctness	Average	8.5	9.0	8.7	8.3	8.6	7.8	6.5	2.5
	No. Resp.	17.	13.	12.	11.	10.	10.	10.	8.0
	Std. Dev.	1.3	0.0	1.2	1.6	1.3	2.7	3.6	4.0

SW FACTOR/CHARACTERISTIC		Sys. Rel. Impact	A	B	C	D	E	F	G
Validity	Average	8.4	7.8	8.6	8.6	9.0	8.0	6.4	2.1
	No. Resp.	14.	11.	11.	9.0	8.0	8.0	8.0	7.0
	Std. Dev.	1.5	2.9	1.2	1.3	0.0	2.8	3.9	3.5
Generality	Average	3.7	6.2	7.0	7.0	5.6	6.7	4.4	1.6
	No. Resp.	14.	10.	8.0	8.0	7.0	7.0	8.0	8.0
	Std. Dev.	3.5	3.3	3.0	2.1	3.6	3.1	3.5	2.1
Testability	Average	7.7	6.0	6.1	6.7	7.2	5.8	5.9	1.8
	No. Resp.	19.	11.	11.	12.	9.0	10.	8.0	8.0
	Std. Dev.	2.3	3.8	3.1	3.2	2.9	3.2	3.8	2.1
Efficiency / Economy	Average	3.3	4.8	5.3	6.3	5.8	4.9	3.4	4.9
	No. Resp.	14.	11.	10.	12.	9.0	9.0	9.0	8.0
	Std. Dev.	3.6	4.2	4.1	3.7	3.5	3.6	3.1	3.9
Resilience (Robustness)	Average	7.1	8.0	8.1	7.0	7.3	5.6	5.4	1.1
	No. Resp.	13.	8.0	9.0	8.0	7.0	7.0	7.0	7.0
	Std. Dev.	2.8	2.8	2.7	3.0	3.1	3.6	3.8	1.8
Useability	Average	5.7	5.7	7.0	6.3	6.9	5.6	6.4	2.4
	No. Resp.	14.	9.0	10.	9.0	8.0	7.0	8.0	7.0
	Std. Dev.	3.8	4.2	3.4	2.8	3.3	2.8	3.2	2.4
Fault Tolerance	Average	8.3	8.2	7.9	6.6	7.5	5.4	6.2	2.3
	No. Resp.	17.	10.	11.	10.	8.0	9.0	9.0	8.0
	Std. Dev.	1.6	2.5	1.9	2.8	3.0	3.1	3.7	2.3
Clarity	Average	7.1	6.9	7.3	7.5	7.7	5.9	7.3	2.1
	No. Resp.	15.	12.	12.	11.	9.0	8.0	10.	8.0
	Std. Dev.	2.1	2.9	2.7	2.7	2.8	3.8	3.1	2.4
Readability	Average	5.5	5.6	6.8	7.3	7.2	6.0	6.3	1.6
	No. Resp.	15.	11.	11.	12.	9.0	8.0	9.0	8.0
	Std. Dev.	2.6	4.1	3.7	3.2	3.5	3.5	3.5	2.1
Maintainability	Average	8.1	5.6	8.0	8.0	7.7	5.4	7.1	3.3
	No. Resp.	17.	11.	12.	12.	9.0	9.0	9.0	8.0
	Std. Dev.	2.2	3.6	1.8	1.8	2.8	3.1	3.2	3.2
Modifiability	Average	7.1	6.0	7.2	6.8	7.0	5.0	6.9	3.0
	No. Resp.	15.	10.	11.	11.	8.0	7.0	8.0	7.0
	Std. Dev.	3.3	3.7	2.8	2.8	3.0	2.3	3.3	3.4
Flexibility	Average	6.4	5.2	7.8	7.0	7.5	5.0	5.9	3.6
	No. Resp.	14.	10.	10.	10.	8.0	7.0	8.0	7.0
	Std. Dev.	3.4	4.2	2.7	2.8	3.0	3.3	3.8	4.1

SW FACTOR/CHARACTERISTIC		Sys. Rel. Impact	A	B	C	D	E	F	G
Portability	Average	3.4	5.3	4.9	6.6	5.5	4.9	3.4	3.9
	No. Resp.	13.	10.	10.	10.	8.0	9.0	8.0	8.0
	Std. Dev.	3.2	3.7	3.4	3.4	4.0	3.6	3.1	3.7
Reusability	Average	3.6	5.8	5.3	5.9	6.1	4.5	4.3	3.7
	No. Resp.	12.	9.0	9.0	9.0	7.0	8.0	7.0	7.0
	Std. Dev.	3.2	3.5	3.3	2.7	3.0	2.6	3.0	3.2
Interoperability	Average	4.6	6.5	5.9	5.4	6.7	5.5	4.9	3.6
	No. Resp.	14.	10.	8.0	8.0	7.0	8.0	8.0	7.0
	Std. Dev.	3.4	3.6	3.8	3.5	3.1	3.3	3.2	4.1

APPENDIX E
FULL SCALE SURVEY FORM & INSTRUCTIONS

SYSTEM RELIABILITY SURVEY

Conducted By:

Ed Soistman (MP-306)
Martin Marietta Aerospace
Orlando Division
P.O. Box 5837
Orlando, FL 32855
(305) 356-7062

Completed By:

Name: _____

Conducted For:

Rome Air Development Center
Mr. Gene Fiorentino (RADC/RBET)
Griffiss AFB, New York 13441

Address: _____

Please check the block which MOST closely identifies your primary involvement with systems involving software:

_____ Systems User
_____ Systems Procurement
_____ Systems Validation
_____ Systems Operations
_____ Systems Research

_____ Systems Definition
_____ Software Design
_____ Systems Development
_____ Systems Management
_____ Training / Education

OTHER: _____

How many years of experience do you have in systems or software activities? _____

What is your highest level educational degree? _____

What single discipline best describes your education/expertise: _____

Comments concerning this survey: _____

Please rank the major categories in order of importance to system reliability using the values 1 thru 5 with "1" being the assigned to the most important category. Use a similar ranking within each of the categories to indicate each requirement's likelihood to introduce errors. As before, use a value of "1" to indicate the most likely.

	* RANK *	* SUB- * RANK *
OPERATIONAL APPLICATION TYPE		
Predominantly Control		
Predominantly Real Time		
Predominantly Input/Output		
Predominantly Interactive		
Predominantly Computational		
MISSION VARIABILITY		
Many Distinct Operational Missions		
Several Variations of Operational Missions		
Single Operational Mission		
FUNCTIONAL COMPLEXITY		
Many Operations Required - Highly Complex		
Many Operations Required - Relatively Simple		
Few Operations Required - Highly Complex		
Few Operations Required - Relatively Simple		
SYSTEM INTERACTION		
Extensive Hardware Interface Requirements		
Minimal Hardware Interface Requirements		
Extensive Software Interface Requirements		
Minimal Software Interface Requirements		
Extensive Human Interface Requirements		
Minimal Human Interface Requirements		
INPUT DOMAIN VARIABILITY		
Wide Range of Error-Prone Inputs		
Wide Range of Error-Free Inputs		
Narrow Range of Error-Prone Inputs		
Narrow Range of Error-Free Inputs		

Use the following codes to indicate the relative quantity of errors introduced during each of the phases shown:

(blank) for NO OPINION
 L for a LOW level of errors introduced
 M for a MODERATE level of errors introduced
 H for a HIGH level of errors introduced

	***** Phase When Introduced *****			
	Req. Defn.	Prelim. Design	Detailed Design	Code
OPERATIONAL APPLICATION TYPE				
Predominantly Control				
Predominantly Real Time				
Predominantly Input/Output				
Predominantly Interactive				
Predominantly Computational				
MISSION VARIABILITY				
Many Distinct Operational Missions				
Several Variations of Operational Missions				
Single Operational Mission				
FUNCTIONAL COMPLEXITY				
Many Operations Required - Highly Complex				
Many Operations Required - Relatively Simple				
Few Operations Required - Highly Complex				
Few Operations Required - Relatively Simple				
SYSTEM INTERACTION				
Extensive Hardware Interface Requirements				
Minimal Hardware Interface Requirements				
Extensive Software Interface Requirements				
Minimal Software Interface Requirements				
Extensive Human Interface Requirements				
Minimal Human Interface Requirements				
INPUT DOMAIN VARIABILITY				
Wide Range of Error-Prone Inputs				
Wide Range of Error-Free Inputs				
Narrow Range of Error-Prone Inputs				
Narrow Range of Error-Free Inputs				

Use the following numeric codes to indicate the percentage of inherent errors which fall into each category. The sum of each row should equal 10 which represents 100% of the errors induced.

0	0%	of Errors Present
1	10%	of Errors Present
:	:	:
9	90%	of Errors Present
10	100%	of Errors Present

	***** General Error Category *****			
	Logic	Inter- face	I/O	Comp.
OPERATIONAL APPLICATION TYPE				
Predominantly Control				
Predominantly Real Time				
Predominantly Input/Output				
Predominantly Interactive				
Predominantly Computational				
MISSION VARIABILITY				
Many Distinct Operational Missions				
Several Variations of Operational Missions				
Single Operational Mission				
FUNCTIONAL REQUIREMENTS				
Many Functions Required - Highly Interrelated				
Many Functions Required - Relatively Independent				
Few Functions Required - Highly Interrelated				
Few Functions Required - Relatively Independent				
SYSTEM INTERACTION				
Extensive Hardware Interface Requirements				
Minimal Hardware Interface Requirements				
Extensive Software Interface Requirements				
Minimal Software Interface Requirements				
Extensive Human Interface Requirements				
Minimal Human Interface Requirements				
INPUT DOMAIN VARIABILITY				
Wide Range of Error-Prone Inputs				
Wide Range of Error-Free Inputs				
Narrow Range of Error-Prone Inputs				
Narrow Range of Error-Free Inputs				

Use the following numeric codes to indicate the relative percentage of errors that might be avoided by use of the listed mechanism: NOTE: THE SUM DOES NOT HAVE TO BE 100%.

(blank) NO OPINION
 0 0% Error Avoidance
 1 10% Error Avoidance
 : : :
 9 90% Error Avoidance
 10 100% Error Avoidance

***** General Error Category *****

Logic Inter- I/O Comp.
 face

ORGANIZATIONAL CONSIDERATIONS

Independent Quality Assurance Organization
 Independent Test Organization
 Independent Verification and Validation (IV&V)
 Use of a Software Support Library
 Use of a Software Configuration Control Board

DOCUMENTATION

Thorough and Enforced Software Development Plan
 Rigidly Controlled System Requirements Spec
 Rigidly Controlled Interface Design Spec
 Rigidly Controlled Software Requirements Spec
 Rigidly Controlled Software Functional Design Spec
 Rigidly Controlled Software Detailed Design Spec

METHODS EMPLOYED

Requirements Traceability Matrix
 Structured Analysis Tools
 Program Specification Language (PSL)
 Program Design Language (PDL)
 High Order Language (HOL)
 Hierarchical, Top-Down Design
 Structured Design
 Single Function Modularization
 Structured Code
 Use of Automatic Measurement Tools
 Use of Automatic Test Tools

Use the following numeric codes to indicate the relative percentage of errors that might be detected by use of the listed mechanism: NOTE: THE SUM DOES NOT HAVE TO BE 100%.

(blank) NO OPINION
 0 0% Error Detection
 1 10% Error Detection
 : : :
 9 90% Error Detection
 10 100% Error Detection

***** General Error Category *****

Logi- Inter- I/O Comp.
 face

INTERNAL REVIEWS

Frequent Peer Walkthroughs
 Infrequent Peer Walkthroughs
 Frequent Progress Reviews
 Infrequent Progress Reviews
 Frequent Quality Audits
 Infrequent Quality Audits

ERROR HANDLING

Use of Software Problem Reports Prior to PQT
 Use of Software Problem Reports Subsequent to PQT
 Use of Software Problem Reports Subsequent to PQT
 Use of Specification Change Notices (SCN's)
 Use of Engineering Change Notices (ECN's)

FORMAL REVIEWS

Software Requirements Review (SRR)
 Preliminary Design Review (PDR)
 Critical Design Review (CDR)
 Test Readiness Review (TRR)
 Functional Configuration Audit (FCA)
 Physical Configuration Audit (PCA)

TESTS AND DEMONSTRATIONS

Informal Unit-Level Testing
 Preliminary Qualification Testing (PQT)
 Formal Qualification Testing (FQT)
 Software Integration Testing
 System Integration Testing
 Operational Field Testing

INSTRUCTIONS / GLOSSARY

FOR

SYSTEM RELIABILITY SURVEY

This instruction packet is provided to explain and clarify the enclosed survey form. It should be used as a reference whenever the terms or instructions used in the survey need further clarification.

There is a separate section for each of the five sheets of the survey for ease of reference. Each section contains an expanded set of instructions, definitions of each of the terms used as column headings and definitions for each of the row entries.

Although most definitions apply to more than one survey sheet, the definitions have been repeated for ease of use.

```
*****
*
*              SPECIAL NOTE FOR SHEETS 1, 2 AND 3              *
*
*****
*
*   In any large software product, it is extremely difficult to precisely
*   identify inherent characteristics of the overall product due to the
*   vast number and diversity of functional requirements that must be sat-
*   isfied.
*
*   Your responses to the survey questions should be based on the assump-
*   tion that any software system can be decomposed into functionally dis-
*   crete sets of requirements which, in turn, can be better correlated to
*   the entries on the survey.
*
*   That is, without pre-supposing a modular design, we can consider that
*   the requirements themselves can be "modularized" so that each "module"
*   can be evaluated on its own merit.
*
*   Your responses on sheets 1, 2 and 3 should, therefore, be oriented
*   towards these "modules" rather than the overall computer program.
*
*****
```

I N S T R U C T I O N S - S H E E T 1

Sheet 1 itemizes certain INHERENT factors which exist in a software product when it is originally conceived and defined. They are the operational requirements which the system is expected to perform. It is felt that the requirements for certain capabilities influence the complexity and "error-proneness" of the product independent of the development methodologies used. In reality, these factors influence the development and testing methodologies.

Column entries on Sheet 1 are relative rankings of importance. In the first column, you are asked to rank the five major categories against one another. In the second column, you are asked to rank the subcategories within each group.

Row entries on Sheet 1 are categories and subcategories of inherent factors which have been singled out for purposes of this survey.

R O W D E F I N I T I O N S - S H E E T 1

OPERATIONAL APPLICATION TYPE -- All responses to this survey should be oriented toward characteristics of individual modules. The purpose of each module can usually be used to determine its PREDOMINANT application type. For example if the purpose of the module is to issue commands to hardware components we would say the the module is of the "predominantly control" type even though it includes computational commands.

CONTROL -- The action of initiating, sequencing, terminating or otherwise influencing the operation of system components external to the software.

REAL-TIME -- The processing of information or data in a manner sufficiently rapid that the results of the processing are available in time to influence the process being monitored or controlled.

INPUT/OUTPUT -- The process of accepting and delivering data to and from system components external to the software. For purposes of this survey input/output should be limited to file input and report output activities as opposed to the control type described above or the interactive type described below.

INTERACTIVE -- A method of conversational input/output wherein the software produces an output which invokes a responsive input or receives an input which requires a responsive output.

COMPUTATIONAL -- The process wherein internally available data is combined, rearranged and/or otherwise manipulated to alter its state. For example a module whose purpose is to convert measurements from one dimension to another should be regarded as being computational.

MISSION VARIABILITY -- In most large scale software applications, a variety of "missions" or modes of operation are supported. For example, software requirements for embedded software in a missile system may involve distinct modes of operation such as "pre-flight", "boost" and "ballistic" activities. Some modules will perform the same activities regardless of the mission type, while others will have distinctly different characteristics depending on the mission mode. MANY and SEVERAL operational missions are relative terms that may be interpreted at the discretion of the reader.

FUNCTIONAL COMPLEXITY - In order to meet its intended purpose, a module may be required to perform more than one specific task. The purpose of these entries on the survey is to accommodate the fact that some functions are relatively easy to design and code whereas others can require extensive and highly complex logic. The adjectives used are relative and may be interpreted by the reader.

SYSTEM INTERACTION -- This category is a refinement of earlier categories. Interface requirements are as previously defined. EXTENSIVE and MINIMAL are relative terms that may be interpreted by the reader. Remember that you are evaluating individual functional requirements, not the overall software.

INPUT DOMAIN VARIABILITY -- This category is a refinement of earlier categories. Here, the interest is not in the quantity of inputs required, but rather the domain from which it comes. For example, a function which requires "yes" or "no" answers to many questions would have a "NARROW RANGE" of values (yes or no). On the other hand, a single input of an angle measurement might have a domain of -180.0000 to +180.0000 degrees. This one would be considered to have a "WIDE RANGE" of inputs.

ERROR-PRONE/ERROR-FREE -- These adjectives are used to distinguish the effects on module reliability caused by the SOURCE of data inputs. A device which contains self-checking features to insure that its inputs to the computer are correct would be considered "error-free". On the other hand, other input devices, such as human operators, may be considered to be "error-prone". Despite the high degree of subjectivity in rating this category, you are asked to rate the effects introduced by such situations.

I N S T R U C T I O N S - S H E E T 2

Column entries on Sheet 2 represent phases of the life cycle wherein errors are likely to be introduced.

Row entries on Sheet 2 are the same INHERENT FACTORS that are used on Sheet 1. their definitions are repeated below.

You are asked to indicate the relative number of errors that are introduced in each phase due to each characteristic. Please remember that we are analyzing functions (modules), not the overall software product. DO NOT "read in" any particular methodology.

C O L U M N D E F I N I T I O N S - S H E E T 2

REQUIREMENTS DEFINITION PHASE - This is the period of time during which the requirements for the software product, such as functional and performance characteristics are defined and documented.

PRELIMINARY DESIGN PHASE - During this phase the software architecture is defined as a result of analysis of the requirements and consideration of possible design alternatives. Typical activities include the definition and structuring of computer programs, components and data, definition of interfaces and preparation of timing and sizing estimates.

DETAILED DESIGN PHASE - During this phase, the preliminary design is refined and expanded to contain more detailed descriptions of the processing logic, data structures and input/output requirements. The level of detail must be sufficient for implementation.

CODING OR IMPLEMENTATION PHASE - The software product is created and debugged during this phase. The detailed design is implemented via a computer "language" which may range from pure binary coded instructions to a very high order procedural language. For purposes of this survey, testing of individual software components are considered to be included in this phase.

ROW DEFINITIONS - SHEET 2

OPERATIONAL APPLICATION TYPE — All responses to this survey should be oriented toward characteristics of individual modules. The purpose of each module can usually be used to determine its **PREDOMINANT** application type. For example if the purpose of the module is to issue commands to hardware components we would say the the module is of the "predominantly control" type even though it includes computational commands.

CONTROL — The action of initiating, sequencing, terminating or otherwise influencing the operation of system components external to the software.

REAL-TIME — The processing of information or data in a manner sufficiently rapid that the results of the processing are available in time to influence the process being monitored or controlled.

INPUT/OUTPUT — The process of accepting and delivering data to and from system components external to the software. For purposes of this survey input/output should be limited to file input and report output activities as opposed to the control type described above or the interactive type described below.

INTERACTIVE — A method of conversational input/output wherein the software produces an output which invokes a responsive input or receives an input which requires a responsive output.

COMPUTATIONAL — The process wherein internally available data is combined, rearranged and/or otherwise manipulated to alter its state. For example a module whose purpose is to convert measurements from one dimension to another should be regarded as being computational.

MISSION VARIABILITY — In most large scale software applications, a variety of "missions" or modes of operation are supported. For example, software requirements for embedded software in a missile system may involve distinct modes of operation such as "pre-flight", "boost" and "ballistic" activities. Some modules will perform the same activities regardless of the mission type, while others will have distinctly different characteristics depending on the mission mode. **MANY** and **SEVERAL** operational missions are relative terms that may be interpreted at the discretion of the reader.

FUNCTIONAL COMPLEXITY - In order to meet its intended purpose, a module may be required to perform more than one specific task. The purpose of these entries on the survey is to accommodate the fact that some functions are relatively easy to design and code whereas others can require extensive and highly complex logic. The adjectives used are relative and may be interpreted by the reader.

SYSTEM INTERACTION -- This category is a refinement of earlier categories. Interface requirements are as previously defined. EXTENSIVE and MINIMAL are relative terms that may be interpreted by the reader. Remember that you are evaluating individual functional requirements, not the overall software.

INPUT DOMAIN VARIABILITY — This category is a refinement of earlier categories. Here, the interest is not in the quantity of inputs required, but rather the domain from which it comes. For example, a function which requires "yes" or "no" answers to many questions would have a "NARROW RANGE" of values (yes or no). On the other hand, a single input of an angle measurement might have a domain of -180.0000 to +180.0000 degrees. This one would be considered to have a "WIDE RANGE" of inputs.

ERROR-PRONE/ERROR-FREE — These adjectives are used to distinguish the effects on module reliability caused by the SOURCE of data inputs. A device which contains self-checking features to insure that its inputs to the computer are correct would be considered "error-free". On the other hand, other input devices, such as human operators, may be considered to be "error-prone". Despite the high degree of subjectivity in rating this category, you are asked to rate the effects introduced by such situations.

I N S T R U C T I O N S - S H E E T 3

Column entries on Sheet 3 are general error categories and are defined below.

Row entries on Sheet 3 are the same as those on Sheets 1 and 2 and are repeated below.

You are asked to distribute whatever errors may occur into the four categories listed. On this sheet, the quantity of errors is irrelevant. The question being asked is, "Assuming that the row entry causes errors, what percent falls into each category?"

C O L U M N D E F I N I T I O N S - S H E E T 3

LOGICAL ERRORS - This category includes all instances where a particular function is missing, incorrect or inadequate due to inadequate requirements definition, design errors or omissions or implementation errors.

INTERFACE ERRORS - This category includes all instances where a required function is not implemented properly due to improper communication between system components. All possible interfaces are included in the grouping:

- Software/Software: Includes errors which occur between software components of the system such as when one program unit fails to call, calls in the wrong sequence, or otherwise improperly calls another program unit. Also included are all errors resulting from the improper sharing or passing of data and/or control variables between program units.
- Software/Hardware: Includes all errors which result in loss of data or untimely exchange of data between system hardware and embedded software. Included are situations where buffers become saturated or computation cycles exceed their timing allocations. Also included are errors caused by improper data exchange between system hardware and embedded software.
- Software/Human: SEE INPUT/OUTPUT ERRORS

INPUT/OUTPUT ERRORS - This category includes all instances where a required function is not properly accomplished due to the manner in which input or output is implemented. For purposes of this survey, include in this category all software/human interfaces. For example, on input the software may either accept improper commands or reject proper ones. On output, the software may generate erroneous or ambiguous messages to the operator. Since the survey is oriented toward mission critical, embedded software, I/O between the hardware and other software components is considered in the "Interface Error" category.

COMPUTATIONAL ERRORS - These are calculation errors. Included are: errors of omission such as uninitialized variables; mathematical errors such as incorrect expressions, conversion and truncation errors; and programming errors such as improper use of indices, variables and overlays. DO NOT INCLUDE SYNTAX ERRORS that would be eliminated prior to operation.

ROW DEFINITIONS - SHEET 3

OPERATIONAL APPLICATION TYPE — All responses to this survey should be oriented toward characteristics of individual modules. The purpose of each module can usually be used to determine its **PREDOMINANT** application type. For example if the purpose of the module is to issue commands to hardware components we would say that the module is of the "predominantly control" type even though it includes computational commands.

CONTROL — The action of initiating, sequencing, terminating or otherwise influencing the operation of system components external to the software.

REAL-TIME — The processing of information or data in a manner sufficiently rapid that the results of the processing are available in time to influence the process being monitored or controlled.

INPUT/OUTPUT — The process of accepting and delivering data to and from system components external to the software. For purposes of this survey input/output should be limited to file input and report output activities as opposed to the control type described above or the interactive type described below.

INTERACTIVE — A method of conversational input/output wherein the software produces an output which invokes a responsive input or receives an input which requires a responsive output.

COMPUTATIONAL — The process wherein internally available data is combined, rearranged and/or otherwise manipulated to alter its state. For example a module whose purpose is to convert measurements from one dimension to another should be regarded as being computational.

MISSION VARIABILITY — In most large scale software applications, a variety of "missions" or modes of operation are supported. For example, software requirements for embedded software in a missile system may involve distinct modes of operation such as "pre-flight", "boost" and "ballistic" activities. Some modules will perform the same activities regardless of the mission type, while others will have distinctly different characteristics depending on the mission mode. **MANY** and **SEVERAL** operational missions are relative terms that may be interpreted at the discretion of the reader.

FUNCTIONAL COMPLEXITY - In order to meet its intended purpose, a module may be required to perform more than one specific task. The purpose of these entries on the survey is to accommodate the fact that some functions are relatively easy to design and code whereas others can require extensive and highly complex logic. The adjectives used are relative and may be interpreted by the reader.

SYSTEM INTERACTION — This category is a refinement of earlier categories.

Interface requirements are as previously defined. EXTENSIVE and MINIMAL are relative terms that may be interpreted by the reader. Remember that you are evaluating individual functional requirements, not the overall software.

INPUT DOMAIN VARIABILITY — This category is a refinement of earlier categories.

Here, the interest is not in the quantity of inputs required, but rather the domain from which it comes. For example, a function which requires "yes" or "no" answers to many questions would have a "NARROW RANGE" of values (yes or no). On the other hand, a single input of an angle measurement might have a domain of -180.0000 to +180.0000 degrees. This one would be considered to have a "WIDE RANGE" of inputs.

ERROR-PRONE/ERROR-FREE — These adjectives are used to distinguish the effects

on module reliability caused by the SOURCE of data inputs. A device which contains self-checking features to insure that its inputs to the computer are correct would be considered "error-free". On the other hand, other input devices, such as human operators, may be considered to be "error-prone". Despite the high degree of subjectivity in rating this category, you are asked to rate the effects introduced by such situations.

INSTRUCTIONS - SHEET 4

Column entries on Sheet 4 are the same general error categories used on Sheet 3 and their definitions are repeated below.

Row entries on Sheet 4 "error AVOIDANCE mechanisms" which are effective in minimizing or reducing errors.

You are asked to evaluate the effectiveness of each mechanism by indicating what percentage of each error type can be avoided by the use of that mechanism.

COLUMN DEFINITIONS - SHEET 4

LOGICAL ERRORS - This category includes all instances where a particular function is missing, incorrect or inadequate due to inadequate requirements definition, design errors or omissions or implementation errors.

INTERFACE ERRORS - This category includes all instances where a required function is not implemented properly due to improper communication between system components. All possible interfaces are included in the grouping:

- Software/Software: Includes errors which occur between software components of the system such as when one program unit fails to call, calls in the wrong sequence, or otherwise improperly calls another program unit. Also included are all errors resulting from the improper sharing or passing of data and/or control variables between program units.
- Software/Hardware: Includes all errors which result in loss of data or untimely exchange of data between system hardware and embedded software. Included are situations where buffers become saturated or computation cycles exceed their timing allocations. Also included are errors caused by improper data exchange between system hardware and embedded software.
- Software/Human: SEE INPUT/OUTPUT ERRORS

INPUT/OUTPUT ERRORS - This category includes all instances where a required function is not properly accomplished due to the manner in which input or output is implemented. For purposes of this survey, include in this category all software/human interfaces. For example, on input the software may either accept improper commands or reject proper ones. On output, the software may generate erroneous or ambiguous messages to the operator. Since the survey is oriented toward mission critical, embedded software, I/O between the hardware and other software components is considered in the "Interface Error" category.

COMPUTATIONAL ERRORS - These are calculation errors. Included are: errors of omission such as uninitialized variables; mathematical errors such as incorrect expressions, conversion and truncation errors; and programming errors such as improper use of indices, variables and overlays. DO NOT INCLUDE SYNTAX ERRORS that would be eliminated prior to operation.

ROW DEFINITIONS - SHEET 4

QUALITY ASSURANCE ORGANIZATION -- A group responsible for the planned and systematic review of the software development process and its products to provide adequate confidence that the item or product conforms to established technical requirements.

TEST ORGANIZATION -- A group responsible for preparing test plans and procedures, executing the test procedures, and analyzing the test results in order to verify that the system performed its intended functions. This group is also responsible for documenting problems detected during testing and verifying by retest that corrections to the software work properly.

INDEPENDENT VERIFICATION AND VALIDATION (IV&V) -- Verification and validation of a software product performed by an organization that is both technically and managerially separate from the organization responsible for developing the product.

SOFTWARE SUPPORT LIBRARY -- A software library containing computer readable and human readable information relevant to a software development effort.

CONFIGURATION CONTROL BOARD -- The authority responsible for evaluating and approving or disapproving proposed engineering changes, and ensuring implementation of the approved changes.

SOFTWARE DEVELOPMENT PLAN -- This document presents the comprehensive plan for the project's software development activities by describing the software development organization, the software design and testing approach, the programs and documentation that will be produced, software milestones and schedules, and the allocation of development resources.

SYSTEM REQUIREMENTS SPECIFICATION -- This document states the technical and mission requirements for a system as an entity, allocates requirements to functional areas, and defines the interfaces between or among the functional areas.

INTERFACE DESIGN SPECIFICATION -- This is an optional document which is required whenever the system contains two or more computers that must communicate with each other. It provides a detailed logical description of all data units, messages, control signals and communication conventions between the digital processors.

SOFTWARE REQUIREMENTS SPECIFICATION -- This document establishes the requirements for the performance, design, test and qualification of the computer program.

SOFTWARE FUNCTIONAL DESIGN SPECIFICATION -- This document establishes the functional design of the software at the computer program level. It provides sufficient design information to accomplish the goals of the Preliminary Design Review.

SOFTWARE DETAILED DESIGN SPECIFICATION -- This document provides complete programming design sufficiently detailed for a programmer to code from with minimal additional direction.

REQUIREMENTS TRACEABILITY MATRIX -- A set of tables which provides traceability of software requirements from the system specification to the individual item requirements specifications, to the design specification which implements the requirements, and to the software plans and procedures that verify that requirements have been fully implemented.

STRUCTURED ANALYSIS TOOLS -- These define a systematic method of using function networks and other tools to develop an analysis-phase model of a system. Typical tools include Data Flow Diagrams, Data Dictionaries and structured English.

PROGRAM SPECIFICATION LANGUAGE (PSL) -- A language used to specify the requirements, design, behavior, or other characteristics of a system or system component.

PROGRAM DESIGN LANGUAGE (PDL) -- A language with special constructs and, sometimes, verification protocols used to develop, analyze, and document a design.

HIGH ORDER LANGUAGE (HOL) -- A programming language which provides compression of computer instructions such that one program statement represents many machine language instructions. It is non-problem specific and is used by programmers to communicate with the computer.

HIERARCHICAL DESIGN -- A design which consists of multiple levels of decomposition, general to specific. It is a structured approach with the additional restriction that program control is accomplished hierarchially. That is, program modules may only invoke other modules which are subordinate to them.

TOP-DOWN DESIGN -- An ordering to the sequence of decisions which are made in the decomposition of a software system, by beginning with a simple description of the entire process (top level). Through a succession of refinements of what has been defined at each level, lower levels are specified.

STRUCTURED DESIGN -- A disciplined approach to software design which adheres to a specified set of rules based on principles such as top-down design, modularization, stepwise refinement, etc.

SINGLE FUNCTION MODULARIZATION -- An organization of the functions of the computer program into a set of discrete program modules each of which is designed to perform a single function.

STRUCTURED CODE -- Code that has been generated with a limited number of well-defined control structures using stepwise refinement.

AUTOMATIC MEASUREMENT TOOLS — This category includes all computer programs which evaluate other computer programs. They may be used to verify compliance with coding standards, to measure progress, or to provide a measure of complexity. They may be applied to any or all phases of the development cycle.

AUTOMATIC TEST TOOLS — This category includes all computer programs that automatically devise and/or execute tests on other computer programs by analysis of the path logic and variable domains of the software being tested and construction of test data sets which will exercise all logical paths under all or extreme input conditions.

I N S T R U C T I O N S - S H E E T 5

Column entries on Sheet 5 are the same general error categories used on Sheets 3 and 4 and their definitions are defined below.

Row entries on Sheet 5 are "error DETECTION mechanisms" which are effective in detecting or recognizing errors.

You are asked to evaluate the effectiveness of each mechanism by indicating what percentage of each error type can be detected (and corrected) by the use of that mechanism. Ignore the possibility that the mechanism itself may introduce errors.

C O L U M N D E F I N I T I O N S - S H E E T 5

LOGICAL ERRORS - This category includes all instances where a particular function is missing, incorrect or inadequate due to inadequate requirements definition, design errors or omissions or implementation errors.

INTERFACE ERRORS - This category includes all instances where a required function is not implemented properly due to improper communication between system components. All possible interfaces are included in the grouping:

- Software/Software: Includes errors which occur between software components of the system such as when one program unit fails to call, calls in the wrong sequence, or otherwise improperly calls another program unit. Also included are all errors resulting from the improper sharing or passing of data and/or control variables between program units.
- Software/Hardware: Includes all errors which result in loss of data or untimely exchange of data between system hardware and embedded software. Included are situations where buffers become saturated or computation cycles exceed their timing allocations. Also included are errors caused by improper data exchange between system hardware and embedded software.
- Software/Human: SEE INPUT/OUTPUT ERRORS

INPUT/OUTPUT ERRORS - This category includes all instances where a required function is not properly accomplished due to the manner in which input or output is implemented. For purposes of this survey, include in this category all software/human interfaces. For example, on input the software may either accept improper commands or reject proper ones. On output, the software may generate erroneous or ambiguous messages to the operator. Since the survey is oriented toward mission critical, embedded software, I/O between the hardware and other software components is considered in the "Interface Error" category.

COMPUTATIONAL ERRORS - These are calculation errors. Included are: errors of omission such as uninitialized variables; mathematical errors such as incorrect expressions, conversion and truncation errors; and programming errors such as improper use of indices, variables and overlays. DO NOT INCLUDE SYNTAX ERRORS that would be eliminated prior to operation.

ROW DEFINITIONS - SHEET 5

FREQUENT/INFREQUENT -- These are relative terms that may be interpreted by the reader. In general, however, it is preferred that "frequent" be used to describe activities that occur on a regular, scheduled basis (e.g., weekly). "Infrequent" carries the connotation that the activity is less rigidly planned and accomplished (e.g., whenever a problem is suspected).

WALKTHROUGH -- A review process in which an analyst, designer or programmer leads one or more peers through a segment of the software product which he or she has developed.

PROGRESS REVIEW -- For purposes of this survey, a progress review is a periodic report given to an individual's supervisor to provide an assessment of the state of completion of a software product. This is in contrast to a walkthrough which is conducted among peers and is primarily technical in nature.

QUALITY AUDIT -- For purposes of this survey, a quality audit is an announced or unannounced inspection of a software product or process. For example, an audit may consist of an inspection of a portion of a programmer's code to verify compliance with programming standards.

PQT and FQT -- See Below

SOFTWARE PROBLEM REPORT -- A report of a program deficiency identified during software qualification, test, system integration and test, or system operation, which appears to be software related.

SPECIFICATION CHANGE NOTICE -- A formal notification of a change in the specification.

ENGINEERING CHANGE NOTICE -- A document used to process changes to baseline documents and which includes both notice of an engineering change to a configuration item and the supporting documentation by which the change is described.

SOFTWARE REQUIREMENTS REVIEW (SRR) -- A review to achieve formal agreement between the customer and the developer that the software requirements specifications are complete and accurate.

PRELIMINARY DESIGN REVIEW (PDR) -- A formal technical review of the basic design approach. It is held after the completion of preliminary design efforts but prior to the start of detailed design. See also **SYSTEM DESIGN REVIEW** and **CRITICAL DESIGN REVIEW**.

CRITICAL DESIGN REVIEW (CDR) -- A formal technical design review conducted to ensure that the detailed design correctly and completely satisfies the requirements. It is conducted after completion of the detailed design but prior to coding. It establishes the design baseline.

TEST READINESS REVIEW (TRR) -- A review conducted prior to each test to ensure adequacy of the documentation and to establish a configuration baseline.

FUNCTIONAL CONFIGURATION AUDIT (FCA) -- Audit to verify that the actual performance of the configuration items complies with the B-5 development specifications.

PHYSICAL CONFIGURATION AUDIT (PCA) -- A formal examination of the as-built version of a configuration item against its technical documentation to ensure the adequacy, completeness, and accuracy of the technical design documentation.

UNIT LEVEL TESTING -- Testing to verify program unit logic, computational adequacy, data handling capability, interfaces and design extremes, and to execute and verify every branch.

PRELIMINARY QUALIFICATION TESTING (PQT) -- An incremental testing process which provides visibility and control of the computer program development during the time period between the Critical Design Review (CDR) and Formal Qualification Testing (FQT); conducted for those functions critical to the CPCI.

FORMAL QUALIFICATION TESTING (FQT) -- Testing conducted prior to Functional Configuration Audit to demonstrate CPCI compliance with all applicable software specifications.

SOFTWARE INTEGRATION TESTING -- Tests of the overall computer program used to verify proper module interfaces with respect to sequencing, timing, and data compatibility.

SYSTEM INTEGRATION TESTING -- The process of testing an integrated hardware and software system to verify that the system meets its specified requirements.

OPERATIONAL FIELD TESTING -- Tests performed by operational personnel in the operational environment. These can be the same tests performed earlier during FQT.

APPENDIX F
FULL SCALE SURVEY RESULTS

Three hundred and fifty surveys were distributed and ninety were returned. Table F-1 shows a profile of the qualifications of the people who responded to the survey. The occupations identified, the education level, and the years experience show that we were quite successful in tapping genuine expert opinions. The results presented herein represent the statistical properties of the entire sample. Several test cases were conducted to see if there was a noticeable difference in the responses of particular groups of individuals (e.g., government versus industry), but none was apparent.

Table F-2 shows the results of sheet 1 of the survey where participants were asked to rank the importance of various factors and categories. No single category was universally recognized as the most (or least) important. The results, however, are indicative of the wide spectrum of software applications and the subsequent divergence of concerns among developers. That is, the primary concerns of software developers and users are directly related to the type of software with which they are associated. It also indicates that a reliability prediction methodology must either be "customized" for each application or must include provisions for all possible influences.

Table F-3 presents the statistical results of the survey. Specifically, the following statistics are shown:

- o AVERAGE - This is the numerical average of all the responses received. It was computed individually for each row-column entry by summing the non-blank entries and dividing by the number of non-blank entries.
- o BOUNDS - These define the 90% confidence interval based on the computer standard deviation of the mean response.
- o STANDARD DEVIATION - This is the standard deviation of the mean, based on the computed sample standard deviation and the sample size of each individual row-column entry.
- o NUMBER OF RESPONSES - This is the number of non-blank responses to each individual entry.

TABLE F-1. PROFILE OF PARTICIPANTS

TYPE OF WORK	NUMBER OF PARTICIPANTS	PERCENTAGE
University	4	4.5
Martin Marietta	15	17.0
Other Contractor	53	60.2
Government	16	18.2

NUMBER OF YEARS EXPERIENCE

Unknown	4	4.5
1 - 5	10	11.4
6 - 10	11	12.5
11 - 15	25	28.4
16 - 20	18	20.5
21 - 25	11	12.5
26 - 30	8	9.1
31 - 35	1	1.1
36 - 40	0	0.0
41 - 45	0	0.0

AREA OF INVOLVEMENT

Systems User	1	1.1
Systems Procurement	1	1.1
Systems Validation	18	20.5
Systems Operations	1	1.1
Systems/SW Research	9	10.2
Systems Definition	0	0.0
Systems/SW Des/Dev	36	40.9
Systems Management	8	9.1
Training/Education	2	2.3
Other	12	13.6

LEVEL OF EDUCATION

Unknown	3	3.4
High School	2	2.3
Associate's Degree	2	2.3
Bachelor's Degree	34	38.6
Master's Degree	34	38.6
Doctoral Degree	13	14.8

TABLE F-2 RANKING OF INHERENT CHARACTERISTICS

OPERATIONAL APPLICATION TYPE	3.5
Predominantly Control	2.7
Predominantly Real Time	1.7
Predominantly Input/Output	3.3
Predominantly Interactive	2.8
Predominantly Computational	4.5
MISSION VARIABILITY	3.1
Many Distinct Operational Missions	1.4
Several Variations of Operational Missions	1.6
Single Operational Mission	2.9
FUNCTIONAL COMPLEXITY	2.3
Many Operations Required - Highly Complex	1.0
Many Operations Required - Relatively Simple	2.8
Few Operations Required - Highly Complex	2.2
Few Operations Required - Relatively Simple	4.0
SYSTEM INTERACTION	2.3
Extensive Hardware Interface Requirements	1.9
Minimal Hardware Interface Requirements	4.6
Extensive Software Interface Requirements	2.0
Minimal Software Interface Requirements	4.8
Extensive Human Interface Requirements	2.4
Minimal Human Interface Requirements	5.1
INPUT DOMAIN VARIABILITY	3.8
Wide Range of Error-Prone Inputs	1.1
Wide Range of Error-Free Inputs	2.8
Narrow Range of Error-Prone Inputs	2.3
Narrow Range of Error-Free Inputs	3.8

TABLE F-3. STATISTICAL RESULTS OF SURVEY

<u>INHERENT FACTORS</u>	<u>REQ DEF</u>	<u>PRELIM DESIGN</u>	<u>DETAIL DESIGN</u>	<u>CODE</u>
PREDOMINANTLY CONTROL				
Average	5.9	4.7	4.7	3.1
Upper Bound	6.5	5.2	5.2	3.6
Lower Bound	5.3	4.2	4.3	2.6
Standard Deviation	0.4	0.3	0.3	0.3
Number of Responses	79	80	79	78
PREDOMINANTLY REAL TIME				
Average	6.1	5.8	6.3	4.4
Upper Bound	6.7	6.4	6.7	5.0
Lower Bound	5.5	5.3	5.8	3.9
Standard Deviation	0.3	0.3	0.3	0.4
Number of Responses	82	82	80	79
PREDOMINANTLY INPUT/OUTPUT				
Average	5.3	4.4	4.3	3.0
Upper Bound	5.9	4.9	4.7	3.5
Lower Bound	4.7	3.9	3.9	2.5
Standard Deviation	0.4	0.3	0.3	0.3
Number of Responses	80	80	80	76
PREDOMINANTLY INTERACTIVE				
Average	6.2	5.1	4.8	3.6
Upper Bound	6.8	5.6	5.3	4.1
Lower Bound	5.6	4.5	4.3	3.0
Standard Deviation	0.4	0.3	0.3	0.3
Number of Responses	78	78	78	78
PREDOMINANTLY COMPUTATIONAL				
Average	3.7	3.3	3.5	3.0
Upper Bound	4.2	3.8	3.9	3.6
Lower Bound	3.1	2.8	3.0	2.5
Standard Deviation	0.3	0.3	0.3	0.3
Number of Responses	79	78	80	80

INHERENT FACTORS	REQ DEF	PRELIM DESIGN	DETAIL DESIGN	CODE
------------------	------------	------------------	------------------	------

MANY DISTINCT OPERATIONAL MISSIONS

Average	7.4	6.4	5.6	3.5
Upper Bound	7.9	6.9	6.1	4.1
Lower Bound	7.0	5.9	5.1	3.0
Standard Deviation	0.3	0.3	0.3	0.3
Number of Responses	82	79	78	77

SEVERAL VARIATIONS OF OPERATIONAL MISSIONS

Average	6.2	5.4	4.5	2.9
Upper Bound	6.7	5.8	5.0	3.4
Lower Bound	5.6	4.9	4.1	2.4
Standard Deviation	0.3	0.3	0.3	0.3
Number of Responses	82	79	77	77

SINGLE OPERATIONAL MISSION

Average	3.3	2.8	2.5	1.7
Upper Bound	3.9	3.2	2.9	2.0
Lower Bound	2.8	2.4	2.1	1.3
Standard Deviation	0.3	0.3	0.3	0.2
Number of Responses	82	78	78	77

MANY OPERATIONS REQUIRED - HIGHLY COMPLEX

Average	7.7	7.1	6.5	4.4
Upper Bound	8.2	7.5	6.9	5.0
Lower Bound	7.2	6.7	6.0	3.8
Standard Deviation	0.3	0.2	0.3	0.3
Number of Responses	86	84	84	81

MANY OPERATIONS REQUIRED - RELATIVELY SIMPLE

Average	5.1	4.6	3.8	2.5
Upper Bound	5.6	5.0	4.2	2.9
Lower Bound	4.5	4.1	3.3	2.0
Standard Deviation	0.3	0.3	0.3	0.3
Number of Responses	80	81	80	80

INHERENT FACTORS	REQ DEF	PRELIM DESIGN	DETAIL DESIGN	CODE
------------------	------------	------------------	------------------	------

FEW OPERATIONS REQUIRED - HIGHLY COMPLEX

Average	6.0	5.6	5.2	3.2
Upper Bound	6.6	6.1	5.7	3.7
Lower Bound	5.5	5.1	4.8	2.7
Standard Deviation	0.3	0.3	0.3	0.3
Number of Responses	83	81	81	79

FEW OPERATIONS REQUIRED - RELATIVELY SIMPLE

Average	2.9	2.5	2.2	1.8
Upper Bound	3.3	2.9	2.6	2.1
Lower Bound	2.4	2.1	1.8	1.4
Standard Deviation	0.3	0.2	0.2	0.2
Number of Responses	79	81	82	80

EXTENSIVE HARDWARE INTERFACE REQUIREMENTS

Average	6.9	6.3	5.9	4.2
Upper Bound	7.5	6.8	6.3	4.8
Lower Bound	6.3	5.9	5.5	3.6
Standard Deviation	0.3	0.3	0.3	0.4
Number of Responses	80	80	79	74

MINIMAL HARDWARE INTERFACE REQUIREMENTS

Average	3.6	3.0	2.5	1.9
Upper Bound	4.1	3.5	2.9	2.3
Lower Bound	3.1	2.6	2.0	1.5
Standard Deviation	0.3	0.3	0.3	0.2
Number of Responses	74	75	76	73

EXTENSIVE SOFTWARE INTERFACE REQUIREMENTS

Average	7.0	7.0	6.5	4.5
Upper Bound	7.5	7.4	6.9	5.1
Lower Bound	6.5	6.5	6.1	3.9
Standard Deviation	0.3	0.3	0.2	0.4
Number of Responses	82	81	81	77

INHERENT FACTORS	REQ DEF	PRELIM DESIGN	DETAIL DESIGN	CODE
------------------	------------	------------------	------------------	------

MINIMAL SOFTWARE INTERFACE REQUIREMENTS

Average	3.3	2.9	2.4	1.8
Upper Bound	3.8	3.3	2.8	2.2
Lower Bound	2.8	2.5	2.0	1.5
Standard Deviation	0.3	0.3	0.3	0.2
Number of Responses	75	77	77	74

EXTENSIVE HUMAN INTERFACE REQUIREMENTS

Average	7.1	6.7	6.0	4.1
Upper Bound	7.7	7.2	6.4	4.6
Lower Bound	6.6	6.2	5.6	3.5
Standard Deviation	0.3	0.3	0.3	0.3
Number of Responses	82	79	80	76

MINIMAL HUMAN INTERFACE REQUIREMENTS

Average	3.4	2.9	2.6	1.8
Upper Bound	3.9	3.4	3.1	2.1
Lower Bound	2.9	2.4	2.2	1.4
Standard Deviation	0.3	0.3	0.3	0.2
Number of Responses	75	76	77	74

WIDE RANGE OF ERROR-PRONE INPUTS

Average	6.5	6.4	6.1	4.7
Upper Bound	7.1	6.9	6.6	5.4
Lower Bound	5.9	5.9	5.6	4.1
Standard Deviation	0.3	0.3	0.3	0.4
Number of Responses	79	79	80	77

WIDE RANGE OF ERROR-FREE INPUTS

Average	4.2	4.1	3.4	2.0
Upper Bound	4.7	4.6	3.9	2.4
Lower Bound	3.7	3.6	2.9	1.6
Standard Deviation	0.3	0.3	0.3	0.2
Number of Responses	78	77	75	74

<u>INHERENT FACTORS</u>	<u>REQ DEF</u>	<u>PRELIM DESIGN</u>	<u>DETAIL DESIGN</u>	<u>CODE</u>
-------------------------	--------------------	--------------------------	--------------------------	-------------

NARROW RANGE OF ERROR-PRONE INPUTS

Average	4.8	4.6	3.9	3.0
Upper Bound	5.4	5.1	4.5	3.6
Lower Bound	4.3	4.2	3.4	2.5
Standard Deviation	0.3	0.3	0.3	0.3
Number of Responses	76	76	75	73

NARROW RANGE OF ERROR-FREE INPUTS

Average	2.7	2.4	2.1	1.6
Upper Bound	3.2	2.9	2.5	1.9
Lower Bound	2.2	2.0	1.7	1.2
Standard Deviation	0.3	0.3	0.2	0.2
Number of Responses	76	75	76	72

INHERENT FACTORSLOGICINTI/OCOMP

PREDOMINANTLY CONTROL

Average	37.2	29.9	19.2	13.8
Upper Bound	40.9	32.8	21.4	15.8
Lower Bound	33.5	27.0	16.9	11.8
Standard Deviation	2.3	1.8	1.4	1.2
Number of Responses	75	75	75	75

PREDOMINANTLY REAL TIME

Average	30.8	30.8	20.3	16.6
Upper Bound	34.1	33.6	22.9	19.4
Lower Bound	27.6	28.0	17.8	13.9
Standard Deviation	2.0	1.7	1.5	1.7
Number of Responses	76	76	76	76

PREDOMINANTLY INPUT/OUTPUT

Average	20.5	25.6	42.3	10.5
Upper Bound	22.9	28.1	45.6	12.2
Lower Bound	18.0	23.2	39.0	8.7
Standard Deviation	1.5	1.5	2.0	1.1
Number of Responses	75	75	75	75

PREDOMINANTLY INTERACTIVE

Average	27.0	34.2	26.4	12.1
Upper Bound	30.1	37.1	29.5	14.1
Lower Bound	24.0	31.3	23.3	10.2
Standard Deviation	1.8	1.8	1.9	1.2
Number of Responses	75	75	75	75

PREDOMINANTLY COMPUTATIONAL

Average	27.7	16.9	13.3	41.1
Upper Bound	30.8	19.2	15.0	44.9
Lower Bound	24.6	14.7	11.7	37.2
Standard Deviation	1.9	1.4	1.0	2.3
Number of Responses	75	75	75	75

<u>INHERENT FACTORS</u>	<u>LOGIC</u>	<u>INT</u>	<u>I/O</u>	<u>COMP</u>
-------------------------	--------------	------------	------------	-------------

MANY DISTINCT OPERATIONAL MISSIONS

Average	38.5	29.9	17.5	14.1
Upper Bound	41.7	32.2	19.1	16.2
Lower Bound	35.3	27.6	15.8	12.0
Standard Deviation	1.9	1.4	1.0	1.3
Number of Responses	74	74	74	74

SEVERAL VARIATIONS OF OPERATIONAL MISSIONS

Average	36.2	29.8	17.8	16.2
Upper Bound	39.3	32.3	19.5	18.7
Lower Bound	33.0	27.3	16.2	13.8
Standard Deviation	1.9	1.5	1.0	1.5
Number of Responses	74	74	74	74

SINGLE OPERATIONAL MISSION

Average	32.3	25.1	19.4	21.2
Upper Bound	35.5	27.4	21.3	24.3
Lower Bound	29.0	22.7	17.4	18.1
Standard Deviation	2.0	1.4	1.2	1.9
Number of Responses	74	74	74	74

MANY OPERATIONS REQUIRED - HIGHLY INTERRELATED

Average	38.5	31.6	15.1	14.9
Upper Bound	41.7	34.3	16.7	17.1
Lower Bound	35.3	28.8	13.4	12.7
Standard Deviation	2.0	1.7	1.0	1.3
Number of Responses	78	78	78	78

MANY OPERATIONS REQUIRED - RELATIVELY INDEPENDENT

Average	37.2	23.6	18.7	19.1
Upper Bound	40.6	25.9	20.8	21.6
Lower Bound	33.9	21.2	16.7	16.7
Standard Deviation	2.0	1.5	1.3	1.5
Number of Responses	75	75	75	75

INHERENT FACTORSLOGICINTI/OCOMP

FEW OPERATIONS REQUIRED - HIGHLY INTERRELATED

Average	34.6	31.6	16.8	17.2
Upper Bound	37.5	34.4	18.4	19.5
Lower Bound	31.7	28.7	15.1	14.8
Standard Deviation	1.8	1.7	1.0	1.4
Number of Responses	76	76	76	75

FEW OPERATIONS REQUIRED - RELATIVELY INDEPENDENT

Average	34.3	24.0	18.2	21.8
Upper Bound	37.4	26.1	19.8	24.5
Lower Bound	31.2	21.9	16.7	19.1
Standard Deviation	1.9	1.3	1.0	1.6
Number of Responses	76	76	76	76

EXTENSIVE HARDWARE INTERFACE REQUIREMENTS

Average	26.2	42.5	20.9	10.4
Upper Bound	29.2	45.6	23.3	11.8
Lower Bound	23.2	39.4	18.5	8.9
Standard Deviation	1.8	1.9	1.5	0.9
Number of Responses	76	76	76	76

MINIMAL HARDWARE INTERFACE REQUIREMENTS

Average	31.1	28.4	21.4	17.5
Upper Bound	34.0	31.5	23.2	19.6
Lower Bound	28.3	25.4	19.6	15.3
Standard Deviation	1.7	1.9	1.1	1.3
Number of Responses	72	72	72	72

EXTENSIVE SOFTWARE INTERFACE REQUIREMENTS

Average	28.4	41.7	17.5	11.3
Upper Bound	31.4	45.0	19.9	13.0
Lower Bound	25.4	38.3	15.1	9.6
Standard Deviation	1.8	2.0	1.5	1.0
Number of Responses	78	78	78	78

INHERENT FACTORS	LOGIC	INT	I/O	COMP
------------------	-------	-----	-----	------

MINIMAL SOFTWARE INTERFACE REQUIREMENTS

Average	30.7	27.5	21.5	18.9
Upper Bound	33.3	30.9	23.3	21.4
Lower Bound	28.2	24.2	19.7	16.3
Standard Deviation	1.6	2.0	1.1	1.5
Number of Responses	72	72	72	72

EXTENSIVE HUMAN INTERFACE REQUIREMENTS

Average	27.0	34.1	26.7	11.2
Upper Bound	30.0	37.6	30.0	12.9
Lower Bound	24.0	30.5	23.4	9.4
Standard Deviation	1.8	2.2	2.0	1.1
Number of Responses	75	75	75	75

MINIMAL HUMAN INTERFACE REQUIREMENTS

Average	32.2	26.9	21.3	17.8
Upper Bound	34.9	29.5	23.3	20.1
Lower Bound	29.4	24.4	19.2	15.6
Standard Deviation	1.7	1.6	1.2	1.4
Number of Responses	72	72	72	72

WIDE RANGE OF ERROR-PRONE INPUTS

Average	30.3	23.6	28.3	17.8
Upper Bound	34.4	26.8	32.2	21.0
Lower Bound	26.3	20.3	24.4	14.6
Standard Deviation	2.5	2.0	2.4	2.0
Number of Responses	73	73	73	73

SYS42::OPA0:,OPERATOR 15:07:36.73
 SYSTEM COMING DOWN AT 6:00PM FOR BACKUPS
 WIDE RANGE OF ERROR-FREE INPUTS

Average	30.4	25.3	24.7	19.2
Upper Bound	33.4	28.0	27.2	21.8
Lower Bound	27.4	22.5	22.3	16.5
Standard Deviation	1.8	1.7	1.5	1.6
Number of Responses	70	70	70	70

INHERENT FACTORSLOGICINTI/OCOMP

NARROW RANGE OF ERROR-PRONE INPUTS

Average	31.3	23.6	25.9	18.0
Upper Bound	34.8	26.6	28.9	20.4
Lower Bound	27.9	20.6	23.0	15.6
Standard Deviation	2.1	1.8	1.8	1.5
Number of Responses	72	72	72	72

NARROW RANGE OF ERROR-FREE INPUTS

Average	31.1	23.7	24.8	20.0
Upper Bound	34.2	26.0	27.5	22.1
Lower Bound	28.1	21.5	22.1	17.9
Standard Deviation	1.9	1.4	1.6	1.3
Number of Responses	70	70	70	70

<u>AVOIDANCE MECHANISMS</u>	<u>LOGIC</u>	<u>INT</u>	<u>I/O</u>	<u>COMP</u>
-----------------------------	--------------	------------	------------	-------------

INDEPENDENT QUALITY ASSURANCE ORGANIZATION

Average	35.4	34.7	30.8	30.3
Upper Bound	41.2	40.0	36.3	36.1
Lower Bound	29.6	29.4	25.3	24.5
Standard Deviation	3.5	3.2	3.3	3.6
Number of Responses	70	69	66	67

INDEPENDENT TEST ORGANIZATION

Average	30.4	33.8	34.9	35.4
Upper Bound	35.6	39.1	40.4	40.6
Lower Bound	25.1	28.5	29.5	30.1
Standard Deviation	3.2	3.2	3.3	3.2
Number of Responses	69	68	68	70

INDEPENDENT VERIFICATION AND VALIDATION (IV&V)

Average	36.9	33.9	35.2	37.8
Upper Bound	42.2	39.0	40.7	43.4
Lower Bound	31.5	28.8	29.6	32.2
Standard Deviation	3.2	3.1	3.3	3.4
Number of Responses	69	68	66	67

USE OF A SOFTWARE SUPPORT LIBRARY

Average	22.1	24.3	22.8	24.3
Upper Bound	26.3	28.7	27.0	28.8
Lower Bound	17.9	20.0	18.6	19.7
Standard Deviation	2.5	2.6	2.6	2.8
Number of Responses	67	68	66	69

USE OF A SOFTWARE CONFIGURATION CONTROL BOARD

Average	22.5	32.1	22.3	16.9
Upper Bound	27.0	37.1	26.6	20.8
Lower Bound	18.1	27.1	18.0	13.0
Standard Deviation	2.7	3.1	2.6	2.4
Number of Responses	67	71	67	66

AVOIDANCE MECHANISMSLOGICINTI/OCOMP

THOROUGH AND ENFORCED SOFTWARE DEVELOPMENT PLAN

Average	32.8	34.8	33.2	32.2
Upper Bound	38.0	39.8	38.3	37.5
Lower Bound	27.7	29.8	28.2	26.9
Standard Deviation	3.1	3.0	3.1	3.2
Number of Responses	72	72	71	71

RIGIDLY CONTROLLED SYSTEM REQUIREMENTS SPEC

Average	35.4	38.3	32.3	30.2
Upper Bound	40.3	43.1	36.9	35.2
Lower Bound	30.5	33.4	27.6	25.3
Standard Deviation	3.0	3.0	2.8	3.0
Number of Responses	75	73	72	69

RIGIDLY CONTROLLED INTERFACE DESIGN SPEC

Average	30.2	53.4	37.0	23.1
Upper Bound	35.1	58.3	42.1	27.8
Lower Bound	25.3	48.6	32.0	18.3
Standard Deviation	3.0	2.9	3.1	2.9
Number of Responses	70	77	72	68

RIGIDLY CONTROLLED SOFTWARE REQUIREMENTS SPEC

Average	40.2	39.5	36.6	34.0
Upper Bound	45.1	44.3	41.4	38.6
Lower Bound	35.3	34.7	31.8	29.5
Standard Deviation	3.0	2.9	2.9	2.8
Number of Responses	75	75	71	72

RIGIDLY CONTROLLED SOFTWARE FUNCTIONAL DESIGN SPEC

Average	36.3	38.8	34.5	34.4
Upper Bound	40.9	43.6	39.3	39.2
Lower Bound	31.7	34.0	29.7	29.7
Standard Deviation	2.8	2.9	2.9	2.9
Number of Responses	74	74	72	71

AVOIDANCE MECHANISMSLOGICINTI/OCOMP

RIGIDLY CONTROLLED SOFTWARE DETAILED DESIGN SPEC

Average	38.3	39.2	39.0	40.3
Upper Bound	43.2	44.4	44.1	45.4
Lower Bound	33.5	34.1	33.9	35.3
Standard Deviation	2.9	3.1	3.1	3.1
Number of Responses	75	73	70	73

REQUIREMENTS TRACEABILITY MATRIX

Average	35.5	31.9	26.2	22.9
Upper Bound	41.1	36.6	31.1	27.6
Lower Bound	29.8	27.2	21.4	18.1
Standard Deviation	3.4	2.8	3.0	2.9
Number of Responses	64	68	65	63

STRUCTURED ANALYSIS TOOLS

SYS42::OPAO:,OPERATOR 15:08:16.11
PLEASE DONOT SUBMIT ANY VERY LONG JOBS

Average	33.7	30.6	28.3	25.8
Upper Bound	38.8	35.2	33.1	30.5
Lower Bound	28.6	26.0	23.4	21.2
Standard Deviation	3.1	2.8	2.9	2.8
Number of Responses	65	66	64	65

PROGRAM SPECIFICATION LANGUAGE (PSL)

Average	31.8	29.0	25.2	24.8
Upper Bound	36.8	33.9	29.7	29.8
Lower Bound	26.9	24.2	20.8	19.9
Standard Deviation	3.0	2.9	2.7	3.0
Number of Responses	65	63	63	62

PROGRAM DESIGN LANGUAGE (PDL)

Average	35.7	31.3	27.9	28.6
Upper Bound	40.3	35.8	32.5	33.4
Lower Bound	31.0	26.7	23.3	23.9
Standard Deviation	2.8	2.8	2.8	2.9
Number of Responses	72	68	66	66

AVOIDANCE MECHANISMS

LOGIC

INT

I/O

COMP

HIGH ORDER LANGUAGE (HOL)

Average	32.6	29.4	30.2	34.9
Upper Bound	37.4	34.1	35.1	39.7
Lower Bound	27.9	24.8	25.3	30.2
Standard Deviation	2.9	2.8	3.0	2.9
Number of Responses	70	63	64	72

HIERARCHICAL, TOP-DOWN DESIGN

Average	38.3	36.7	31.5	26.5
Upper Bound	43.0	41.3	36.3	31.2
Lower Bound	33.6	32.2	26.8	21.9
Standard Deviation	2.9	2.7	2.9	2.8
Number of Responses	67	72	66	66

STRUCTURED DESIGN

Average	38.1	36.0	32.3	28.5
Upper Bound	42.7	40.8	37.2	33.1
Lower Bound	33.5	31.2	27.4	23.8
Standard Deviation	2.8	2.9	3.0	2.9
Number of Responses	71	70	65	68

SINGLE FUNCTION MODULARIZATION

Average	37.5	32.6	30.2	30.8
Upper Bound	42.5	37.7	35.5	35.8
Lower Bound	32.5	27.4	25.0	25.8
Standard Deviation	3.1	3.1	3.2	3.0
Number of Responses	68	70	65	67

STRUCTURED CODE

Average	37.1	30.5	28.0	33.7
Upper Bound	41.9	35.1	33.0	38.7
Lower Bound	32.3	26.0	23.1	28.6
Standard Deviation	2.9	2.7	3.0	3.1
Number of Responses	70	65	64	68

AVOIDANCE MECHANISMS

LOGIC

INT

I/O

COMP

USE OF AUTOMATIC MEASUREMENT TOOLS

Average	23.9	22.0	22.8	24.5
Upper Bound	28.3	26.4	27.2	29.4
Lower Bound	19.4	17.6	18.4	19.5
Standard Deviation	2.7	2.7	2.7	3.0
Number of Responses	57	56	56	56

USE OF AUTOMATIC TEST TOOLS

Average	30.1	28.2	28.8	30.8
Upper Bound	35.0	32.9	33.5	35.8
Lower Bound	25.1	23.4	24.0	25.7
Standard Deviation	3.0	2.9	2.9	3.1
Number of Responses	64	63	64	64

DETECTION MECHANISMS

LOGIC

INT

I/O

COMP

FREQUENT PEER WALKTHROUGHS

Average	45.2	40.5	37.9	39.3
Upper Bound	49.9	45.4	42.9	44.3
Lower Bound	40.5	35.5	32.8	34.3
Standard Deviation	2.9	3.0	3.1	3.0
Number of Responses	80	75	75	77

INFREQUENT PEER WALKTHROUGHS

Average	25.1	22.5	19.5	20.3
Upper Bound	28.7	26.4	23.0	23.7
Lower Bound	21.5	18.7	16.0	16.8
Standard Deviation	2.2	2.3	2.1	2.1
Number of Responses	72	70	69	70

FREQUENT PROGRESS REVIEWS

Average	24.5	22.0	21.5	19.8
Upper Bound	29.1	26.4	25.9	23.9
Lower Bound	19.8	17.7	17.0	15.6
Standard Deviation	2.8	2.6	2.7	2.5
Number of Responses	68	68	67	68

INFREQUENT PROGRESS REVIEWS

Average	11.0	11.0	10.2	8.8
Upper Bound	13.3	13.5	12.5	10.8
Lower Bound	8.8	8.6	7.8	6.7
Standard Deviation	1.4	1.5	1.5	1.3
Number of Responses	62	63	63	61

FREQUENT QUALITY AUDITS

Average	28.8	26.9	26.7	24.5
Upper Bound	33.4	31.5	31.3	29.0
Lower Bound	24.3	22.2	22.1	20.1
Standard Deviation	2.8	2.8	2.8	2.7
Number of Responses	71	69	68	70

DETECTION MECHANISMSLOGICINTI/OCOMP

INFREQUENT QUALITY AUDITS

Average	13.8	13.2	12.0	11.8
Upper Bound	16.7	16.0	14.5	14.4
Lower Bound	10.9	10.4	9.5	9.3
Standard Deviation	1.8	1.7	1.5	1.5
Number of Responses	66	65	65	65

USE OF SOFTWARE PROBLEM REPORTS PRIOR PQT

Average	33.2	33.2	29.7	31.5
Upper Bound	38.4	38.3	34.8	37.0
Lower Bound	28.1	28.1	24.5	26.0
Standard Deviation	3.1	3.1	3.1	3.3
Number of Responses	66	66	65	65

USE OF SOFTWARE PROBLEM REPORTS SUBSEQUENT TO PQT

Average	26.5	27.8	25.2	25.8
Upper Bound	30.9	32.5	29.7	30.3
Lower Bound	22.1	23.2	20.8	21.2
Standard Deviation	2.7	2.8	2.7	2.8
Number of Responses	66	65	64	65

USE OF SOFTWARE PROBLEM REPORTS SUBSEQUENT TO FQT

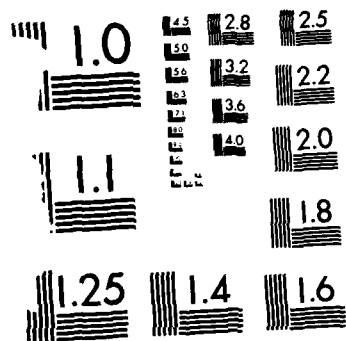
Average	24.5	28.4	23.7	24.6
Upper Bound	29.2	33.8	28.3	29.4
Lower Bound	19.9	23.0	19.0	19.8
Standard Deviation	2.8	3.3	2.8	2.9
Number of Responses	62	62	60	61

USE OF SPECIFICATION CHANGE NOTICES (SCN's)

Average	19.3	22.1	18.9	17.1
Upper Bound	23.5	26.5	22.8	21.1
Lower Bound	15.1	17.8	15.0	13.1
Standard Deviation	2.5	2.6	2.4	2.4
Number of Responses	62	65	64	62

AD-A165 231 IMPACT OF HARDWARE/SOFTWARE FAULTS ON SYSTEM 3/3
RELIABILITY VOLUME 1 STUDY R. (U) MARTIN MARIETTA
AEROSPACE ORLANDO FL E C SOISTMAN ET AL DEC 85
UNCLASSIFIED OR-18173 RADC-TR-85-228-VOL-1 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

DETECTION MECHANISMSLOGICINTI/OCOMP

USE OF ENGINEERING CHANGE NOTICES (ECN's)

Average	20.1	23.1	19.8	18.2
Upper Bound	24.5	27.7	24.2	22.5
Lower Bound	15.7	18.4	15.4	13.9
Standard Deviation	2.7	2.8	2.7	2.6
Number of Responses	61 .	64	62	61

SOFTWARE REQUIREMENTS REVIEW (SRR)

Average	31.4	33.1	28.0	23.4
Upper Bound	36.3	37.6	32.7	27.9
Lower Bound	26.5	28.6	23.4	18.8
Standard Deviation	3.0	2.7	2.8	2.8
Number of Responses	73	72	69	67

PRELIMINARY DESIGN REVIEW (PDR)

Average	28.6	32.1	27.9	22.7
Upper Bound	32.8	36.1	32.0	26.8
Lower Bound	24.5	28.0	23.8	18.5
Standard Deviation	2.5	2.5	2.5	2.5
Number of Responses	73	73	70	67

CRITICAL DESIGN REVIEW (CDR)

Average	32.3	32.8	30.5	26.2
Upper Bound	36.6	36.8	34.8	30.4
Lower Bound	28.1	28.7	26.2	22.0
Standard Deviation	2.6	2.5	2.6	2.6
Number of Responses	75	74	71	70

TEST READINESS REVIEW (TRR)

Average	20.1	23.4	21.9	20.3
Upper Bound	24.0	27.5	25.9	24.4
Lower Bound	16.3	19.3	17.9	16.2
Standard Deviation	2.4	2.5	2.4	2.5
Number of Responses	68	71	68	68

<u>DETECTION MECHANISMS</u>	<u>LOGIC</u>	<u>INT</u>	<u>I/O</u>	<u>COMP</u>
-----------------------------	--------------	------------	------------	-------------

FUNCTIONAL CONFIGURATION AUDIT (FCA)

Average	18.2	21.8	20.8	18.3
Upper Bound	22.4	26.1	25.3	22.5
Lower Bound	14.0	17.5	16.4	14.1
Standard Deviation	2.6	2.6	2.7	2.6
Number of Responses	68	71	68	67

PHYSICAL CONFIGURATION AUDIT (PCA)

Average	14.9	17.4	17.7	14.5
Upper Bound	18.7	21.0	21.4	18.2
Lower Bound	11.2	13.8	14.1	10.9
Standard Deviation	2.3	2.2	2.2	2.2
Number of Responses	67	69	66	64

INFORMAL UNIT-LEVEL TESTING

Average	43.4	31.8	34.5	45.6
Upper Bound	48.1	37.1	39.7	50.5
Lower Bound	38.6	26.6	29.3	40.7
Standard Deviation	2.9	3.2	3.2	3.0
Number of Responses	77	73	74	76

PRELIMINARY QUALIFICATION TESTING (PQT)

Average	36.1	34.1	31.8	32.1
Upper Bound	40.7	38.5	36.4	36.9
Lower Bound	31.4	29.7	27.3	27.2
Standard Deviation	2.8	2.7	2.8	2.9
Number of Responses	70	74	73	70

FORMAL QUALIFICATION TESTING (FQT)

Average	31.4	30.0	28.9	30.2
Upper Bound	36.0	34.4	33.7	35.2
Lower Bound	26.7	25.6	24.2	25.1
Standard Deviation	2.8	2.7	2.9	3.1
Number of Responses	74	76	75	72

DETECTION MECHANISMSLOGICINTI/OCOMP

SOFTWARE INTEGRATION TESTING

Average	34.9	45.0	38.2	33.4
Upper Bound	39.5	49.5	42.8	38.3
Lower Bound	30.4	40.5	33.7	28.4
Standard Deviation	2.8	2.7	2.8	3.0
Number of Responses	75	77	77	73

SYSTEM INTEGRATION TESTING

Average	33.0	41.3	36.7	31.2
Upper Bound	37.7	45.9	41.5	36.4
Lower Bound	28.2	36.8	32.0	26.0
Standard Deviation	2.9	2.8	2.9	3.2
Number of Responses	77	79	79	74

OPERATIONAL FIELD TESTING

Average	32.7	35.2	36.3	31.2
Upper Bound	36.0	38.4	39.5	34.6
Lower Bound	29.4	32.0	33.1	27.8
Standard Deviation	3.3	3.2	3.2	3.4
Number of Responses	75	78	77	75

*MISSION
of
Rome Air Development Center*

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C³I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C³I systems. The areas of technical competence include communications, command and control, battle management, information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic, maintainability, and compatibility.

DTIC

FILMED

4-86

END