13:07:34 OCA PAD INITIATION - PROJECT HEADER INFORMATION 03/15/89

Active

Project #: E-21-T14 Cost share #: E-21-321 Rev #: 0
Center # : R6583-T14 Center shr #: F6583-T14 OCA file #: 128
Work type : RES
Contract#: F30602-88-D-0025-0014 Mod #: Document : TO
Prime #: Contract entity: GTRC

Subprojects ? : N
Main project #:

Project unit: EE Unit code: 02.010.118
Project director(s):
PARIS D T EE (404)894-2902

Sponsor/division names: AIR FORCE / GRIFFISS AFB, NY
Sponsor/division codes: 104 / 023

Award period: 890217 to 890731 (performance) 890830 (reports)

Sponsor amount	New this change	Total to date
Contract value	49,989.00	49,989.00
Funded	49,989.00	49,989.00
Cost sharing amount		5,555.00

Does subcontracting plan apply ?: Y

Title: SYSTEM HARDWARE/SOFTWARE RELIABILITY AND ASSESSMENT HANDBOOK

PROJECT ADMINISTRATION DATA

OCA contact: Brian J. Lindberg 894-4820

Sponsor technical contact

EUGENE FIORENTINO
DEPARTMENT OF THE AIR FORCE
ROME AIR DEVELOPMENT CENTER/RBET
GRIFFISS AFB, NY 13441-5700

Sponsor issuing office

GERARD J. BROWN/PKRM
(315)330-2308
ROME AIR DEVELOPMENT CENTER
DIRECTORATE OF CONTRACTING (PKRM)
GRIFFISS AFB, NY 13441-5700

Security class (U,C,S,TS) : U
Defense priority rating : DO-A7
Equipment title vests with: Sponsor
NONE PROPOSED OR ANTICIPATED.

ONR resident rep. is ACO (Y/N): Y
GOVT supplemental sheet
GIT

Administrative comments -
DELIVERY ORDER FULLY FUNDS TASK N-9-5514 (SOHAR INCORPORATED).

GEORGIA INSTITUTE OF TECHNOLOGY
OFFICE OF CONTRACT ADMINISTRATION

NOTICE OF PROJECT CLOSEOUT

Closeout Notice Date 03/30/90

Project No. E-21-T14 Center No. R6583-T14

Project Director JOY E B School/Lab EE

Sponsor AIR FORCE/GRIFFISS AFB, NY

Contract/Grant No. F30602-88-D-0025-0014 Contract Entity GTRC

Prime Contract No.

Title SYSTEM HARDWARE/SOFTWARE RELIABILITY AND ASSESSMENT HANDBOOK

Effective Completion Date 891015 (Performance) 891115 (Reports)

Closeout Actions Required:	Y/N	Date Submitted
Final Invoice or Copy of Final Invoice	Y	
Final Report of Inventions and/or Subcontracts	Y	
Government Property Inventory & Related Certificate	Y	
Classified Material Certificate	Y	
Release and Assignment	Y	
Other	N	

Comments

Subproject Under Main Project No.

Continues Project No.

Distribution Required:

Project Director	Y
Administrative Network Representative	Y
GTRI Accounting/Grants and Contracts	Y
Procurement/Supply Services	Y
Research Property Managment	Y
Research Security Services	Y
Reports Coordinator (OCA)	Y
GTRC	Y
Project File	Y
Other	N
	N

NOTE: Final Questionnaire sent to PDPI

CONTRACT FUNDS STATUS REPORT (DD FORM 1586)
CONTRACT NUMBER F30602-88-D-0025
QUARTER: MAY-JUN '88

	CURRENT QUARTER FUNDING		$0.00
	CURRENT QUARTER EXPENDITURES		$0.00
	CONTRACT CEILING		$4,200,000.00
	FUNDING TO DATE	-	$0.00
*	PENDING COMMITMENTS	-	$766,000.00
	AVAILABLE FUNDING		$3,434,000.00
	FUNDING TO DATE		$0.00
	YTD EXPENDITURES	-	$0.00
	OUTSTANDING EXPENDITURES		$0.00

*	C-8-2120	WESTINGHOUSE/BEAUDET	$56,000.00
	C-8-2129	RENSSELAER/DAS	$100,000.00
	E-8-7066	UNIV OF PENN/STEINBERG	$100,000.00
	E-8-7124	BOSTON COLLEGE/McFADDEN	$35,000.00
	E-8-7125	BRANDEIS UNIV/HENCHMAN	$23,000.00
	E-8-7126	PENN STATE/CASTLEMAN	$22,000.00
	A-8-1631	UNIV OF PENN/STEINBERG	$100,000.00
	B-8-3617	GA WASHINGTON UNIV/MELTZER	$100,000.00
	B-8-3618	GA WASHINGTON UNIV/BERKOVICH	$100,000.00
	C-8-2492	GA TECH/SMITH	$50,000.00
	A-8-1203	GA TECH/HUGHES	$80,000.00
		TOTAL PENDING	$766,000.00

CONTRACT FUNDS STATUS REPORT (DD FORM 1586)
CONTRACT NUMBER F30602-88-D-0025
QUARTER: JUL-SEPT '88

CURRENT QUARTER FUNDING			$698,034.00
DO #	0001	$56,000	
	0002	$95,141	
	0003	$78,854	
	0004	$230,000	
	0005	$45,561	
	0006	$25,000	
	0007	$20,000	
	0008	$98,374	
	0009	$29,403	
	0010	$19,701	
		$698,034	

CURRENT QUARTER EXPENDITURES			$0.00

CONTRACT CEILING			$4,200,000.00
FUNDING TO DATE		-	$698,034.00
* PENDING COMMITMENTS		-	$426,563.00
AVAILABLE FUNDING			$3,075,403.00

FUNDING TO DATE			$698,034.00
YTD EXPENDITURES		-	$0.00
OUTSTANDING EXPENDITURES			$698,034.00

* DO #	0001	INCREMENTAL FUNDING	$90,729.00
	0002	INCREMENTAL FUNDING	$66,680.00
	0003	INCREMENTAL FUNDING	$54,154.00
	0004	INCREMENTAL FUNDING	$20,000.00
	C-8-2400	STATE UNIV OF NY/FAM	$95,000.00
	C-8-2402	RENSSELAER/SAULNER	$100,000.00
TOTAL PENDING			$426,563.00

CONTRACT FUNDS STATUS REPORT (DD FORM 1586)
CONTRACT NUMBER F30602-88-D-0025
QUARTER: OCT-DEC '88

	CURRENT QUARTER FUNDING		$120,834.00
	DO # 0004	$66,680	
	0006	$54,154	
		$120,834	
	CURRENT QUARTER EXPENDITURES		$28,740.82
	CONTRACT CEILING		$4,200,000.00
	FUNDING TO DATE	-	$818,868.00
*	PENDING COMMITMENTS	-	$784,729.00
	AVAILABLE FUNDING		$2,596,403.00
	FUNDING TO DATE		$818,868.00
	YTD EXPENDITURES	-	$28,740.82
	OUTSTANDING EXPENDITURES		$790,127.18

*	DO # 0001	INCREMENTAL FUNDING	$90,729.00
	0007	INCREMENTAL FUNDING	$20,000.00
	C-8-2400	STATE UNIV OF NY/FAM	$95,000.00
	C-8-2402	RENSSELAER/SAULNER	$100,000.00
	B-9-3592	UNIV OF CA/DAVIS/LEVITT	$60,000.00
	N-9-5514	SOHAR INC./HECHT	$50,000.00
	C-9-2015	NCS/O'NEAL	$100,000.00
	A-9-1120	HITEC, INC./KAZAKOS	$75,000.00
	E-9-7057	UNIV OF TX/ARLINGTON/FUNG	$40,000.00
	E-9-7093	MONTANA STATE/JOHNSON	$34,000.00
	S-9-7552	ALFRED UNIV/SYNDER	$20,000.00
	C-9-2404	STANFORD UNIV/WIDROW	$100,000.00
		TOTAL PENDING	$784,729.00

CONTRACT FUNDS STATUS REPORT (DD FORM 1586)
CONTRACT NUMBER F30602-88-D-0025
QUARTER: JAN-MAR '89

CURRENT QUARTER FUNDING			$574,457.00
DO # 0001	$90,729		
0011	$75,000		
0012	$75,000		
0013	$59,989		
0014	$49,989		
0015	$70,000		
0016	$43,750		
0017	$30,000		
0018	$22,000		
0019	$38,000		
0020	$20,000		
	$574,457		
CURRENT QUARTER EXPENDITURES			$86,324.15
CONTRACT CEILING			$4,200,000.00
FUNDING TO DATE		-	$1,393,325.00
* PENDING COMMITMENTS		-	$594,651.00
AVAILABLE FUNDING			$2,212,024.00
FUNDING TO DATE			$1,393,325.00
YTD EXPENDITURES		-	$115,064.97
OUTSTANDING EXPENDITURES			$1,278,260.03

* DO # 0007	INCREMENTAL FUNDING		$20,000.00
0011	INCREMENTAL FUNDING		$19,568.00
0012	INCREMENTAL FUNDING		$24,700.00
0015	INCREMENTAL FUNDING		$29,783.00
0016	INCREMENTAL FUNDING		$31,250.00
0017	INCREMENTAL FUNDING		$10,000.00
0018	INCREMENTAL FUNDING		$12,000.00
0019	INCREMENTAL FUNDING		$12,000.00
C-8-2404	STANFORD UNIV/WIDROW		$100,000.00
N-9-5732	GRIFFIN		$25,000.00
A-9-1476	BOWDOIN COLLEGE/CHONACKY		$20,350.00
E-9-7110	UNIV OF LOWELL/SALES		$50,000.00
S-9-7559	UNIV OF MICHIGAN/ROBINSON		$20,000.00
B-9-3621	SRI/LUNT		$20,000.00
N-9-5308	KAMAN SCIENCES		$100,000.00
E-9-7119	DARTMOUTH COLLEGE/CRANE		$100,000.00
	TOTAL PENDING		$594,651.00

CONTRACT FUNDS STATUS REPORT (DD FORM 1586)
CONTRACT NUMBER F30602-88-D-0025
QUARTER: APR-JUN '89

CURRENT QUARTER FUNDING			$160,350.00
DO # 0021	$25,000		
0022	$45,000		
0023	$20,350		
0024	$50,000		
0025	$20,000		
	$160,350		
CURRENT QUARTER EXPENDITURES			$318,963.82
CONTRACT CEILING			$4,200,000.00
FUNDING TO DATE		-	$1,553,675.00
* PENDING COMMITMENTS		-	$718,994.00
AVAILABLE FUNDING			$1,927,331.00
FUNDING TO DATE			$1,553,675.00
YTD EXPENDITURES		-	$434,028.79
OUTSTANDING EXPENDITURES			$1,119,646.21

* DO # 0007	INCREMENTAL FUNDING		$20,000.00
0011	INCREMENTAL FUNDING		$19,568.00
0012	INCREMENTAL FUNDING		$24,700.00
0015	INCREMENTAL FUNDING		$29,783.00
0016	INCREMENTAL FUNDING		$31,250.00
0017	INCREMENTAL FUNDING		$10,000.00
0018	INCREMENTAL FUNDING		$12,000.00
0019	INCREMENTAL FUNDING		$12,000.00
0022	INCREMENTAL FUNDING		$54,693.00
B-9-3621	SRI/LUNT		$20,000.00
N-9-5308	KAMAN SCIENCES		$100,000.00
E-9-7119	DARTMOUTH COLLEGE/CRANE		$100,000.00
N-9-5740	CHRISTIANSON		$15,000.00
N-9-5317	UNIV OF CO/NORGARD		$50,000.00
S-9-7625	UNIV OF CA/DAVIS/KOWELL		$20,000.00
N-9-5314	KAMAN SCIENCES		$100,000.00
N-9-5315	KAMAN SCIENCES		$100,000.00
TOTAL PENDING			$718,994.00

CONTRACT FUNDS STATUS REPORT (DD FORM 1586)
CONTRACT NUMBER F30602-88-D-0025
QUARTER: JUL-SEP '89

CURRENT QUARTER FUNDING			$476,000.00
DO # 0017	$10,000		
0026	$15,000		
0027	$20,000		
0028	$50,000		
0029	$40,000		
0030	$30,000		
0031	$20,000		
0032	$66,000		
0033	$70,000		
0034	$85,000		
0035	$70,000		
	$476,000		
CURRENT QUARTER EXPENDITURES			$415,422.69
CONTRACT CEILING			$4,200,000.00
FUNDING TO DATE		-	$2,029,675.00
* PENDING COMMITMENTS		-	$253,994.00
AVAILABLE FUNDING			$1,916,331.00
FUNDING TO DATE			$2,029,675.00
YTD EXPENDITURES		-	$849,451.48
OUTSTANDING EXPENDITURES			$1,180,223.52

* DO # 0007	INCREMENTAL FUNDING		$20,000.00
0011	INCREMENTAL FUNDING		$19,568.00
0012	INCREMENTAL FUNDING		$24,700.00
0015	INCREMENTAL FUNDING		$29,783.00
0016	INCREMENTAL FUNDING		$31,250.00
0018	INCREMENTAL FUNDING		$12,000.00
0019	INCREMENTAL FUNDING		$12,000.00
0022	INCREMENTAL FUNDING		$54,693.00
N-0-5703	UNIV OF SOUTHERN FLA/WILSON		$50,000.00
	TOTAL PENDING		$253,994.00

T14

E-21-914

ROME AIR DEVELOPMENT CENTER
EXPERT SCIENCE AND ENGINEERING PROGRAM
CONTRACT NO. F30602-88-D-0025

R&D STATUS REPORT

PERIOD COVERED: June 1 - June 24, 1989

TASK NUMBER: N-9-5514

TITLE: System Hardware/Software Reliability and Assessment Handbook

PRINCIPAL INVESTIGATOR: Herbert Hecht, Ph. D.

INSTITUTION: SoHaR Incorporated

OTHER PARTICIPANTS: Jeffrey Miller, Sr. Systems Engineer

A. TECHNICAL PROGRESS ACHIEVED ON EFFORT:

The Handbook chapter on reliability modeling was generated. It describes a basic model for combined hardware/software systems that accounts for software utilization. An example applying the model to a simple hardware/software avionics system was included. In addition, a Notice was drafted that modifies MIL-STD-785B, Task 201 in accordance with the procedures developed in the Handbook.

B. TRAVEL: None

C. PRESENTATIONS AND PUBLICATIONS: None

D. LEVEL OF EFFORT IN LABOR-HOURS:

Participant	Current Effort	Cumulative
H. Hecht	36	116
Other Staff	93	185
Total	129	301

E-21-T 14

ROME AIR DEVELOPMENT CENTER
EXPERT SCIENCE AND ENGINEERING PROGRAM
CONTRACT NO. F30602-88-D-0025

R&D STATUS REPORT

PERIOD COVERED: June 25 - July 29, 1989

TASK NUMBER: N-9-5514

TITLE: System Hardware/Software Reliability and Assessment Handbook

PRINCIPAL INVESTIGATOR: Herbert Hecht, Ph. D.

INSTITUTION: SoHaR Incorporated

OTHER PARTICIPANTS: Jeffrey Miller, Sr. Systems Engineer

A. TECHNICAL PROGRESS ACHIEVED ON EFFORT:

A draft Handbook chapter on failure prediction was generated. In addition, a Notice was drafted that modifies MIL-STD-785B, Task 203 in accordance with the procedures developed in the Handbook.

B. TRAVEL: None

C. PRESENTATIONS AND PUBLICATIONS: None

D. LEVEL OF EFFORT IN LABOR-HOURS:

Participant	Current Effort	Cumulative
H. Hecht	64	180
Other Staff	160	345
Total	224	525

ROME AIR DEVELOPMENT CENTER
EXPERT SCIENCE AND ENGINEERING PROGRAM
CONTRACT NO. F30602-88-D-0025

R&D STATUS REPORT

PERIOD COVERED: July 30 - August 26, 1989

TASK NUMBER: N-9-5514

TITLE: System Hardware/Software Reliability and Assessment Handbook

PRINCIPAL INVESTIGATOR: Herbert Hecht, Ph. D.

INSTITUTION: SoHaR Incorporated

OTHER PARTICIPANTS: Jeffrey Miller, Sr. Systems Engineer

A. TECHNICAL PROGRESS ACHIEVED ON EFFORT:

A draft Handbook chapter on Reliability Development Growth Testing was generated. In addition, a Notice was drafted that modifies MIL-STD-785B, Task 302 in accordance with the procedures developed in the Handbook.

B. TRAVEL: Colorado Springs, Aug. 22-23. Meeting of AIAA SB09 Committee on Standards.

C. PRESENTATIONS AND PUBLICATIONS: Presentation of Handbook material to SB09 Committee's Software Reliability Working Group.

D. LEVEL OF EFFORT IN LABOR-HOURS:

Participant	Current Effort	Cumulative
H. Hecht	40	220
Other Staff	120	465
Total	160	685

ROME AIR DEVELOPMENT CENTER
EXPERT SCIENCE AND ENGINEERING PROGRAM
CONTRACT NO. F30602-88-D-0025

R&D STATUS REPORT

PERIOD COVERED: August 26 - September 30, 1989

TASK NUMBER: N-9-5514

TITLE: System Hardware/Software Reliability and Assessment Handbook

PRINCIPAL INVESTIGATOR: Herbert Hecht, Ph. D.

INSTITUTION: SoHaR Incorporated

OTHER PARTICIPANTS: Jeffrey Miller, Sr. Systems Engineer

A. TECHNICAL PROGRESS ACHIEVED ON EFFORT:

A draft copy of the Handbook was submitted to RADC.

B. TRAVEL: RADC, Sept. 27.

C. PRESENTATIONS AND PUBLICATIONS: Presentation of draft copy of the Handbook to E. Fiorentino.

D. LEVEL OF EFFORT IN LABOR-HOURS:

Participant	Current Effort	Cumulative
H. Hecht	8	228
Other Staff	3	468
Total	11	696

SoHaR
Incorporated

H. Hecht,
President

13 March 1990 L90-35

Georgia Institute of Technology
Office of Contract Administration
Atlanta GA 30332-0420

Attn. Laurie Rose

Subject: Subcontract E-21-T14-S1

Dear Ms. Rose:

In accordance with your phone request we are forwarding herewith a copy of the Final Report that had been submitted to Mr. Fiorentino at RADC.

Sincerely yours,

HH:ij

Encl. as noted

1040 South La Jolla Avenue • Los Angeles, California 90035 • Telephone (213) 935-7039
Other Locations: 8500 Wilshire Bl., #1027, Beverly Hills, CA 90211 • 5225 Pooks Kill Rd., #1513S, Bethesda, MD 20814

SYSTEM HARDWARE/SOFTWARE RELIABILITY HANDBOOK

FINAL TECHNICAL REPORT

prepared for

RADC/RBET
Contract No. F30602-88-D-0025
Task No. N-9-5514

under contract with

Georgia Institute of Technology
Subcontract No. E-21-T14-S1

by

Herbert Hecht, Jeffrey Miller
SoHaR Incorporated
Los Angeles, CA

September, 1989

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SYSTEM HARDWARE/SOFTWARE RELIABILITY HANDBOOK

1 SCOPE

1.1 Purpose. This handbook provides guidance for conducting a comprehensive and effective reliability program as defined in MIL-STD-785 on systems that contain both hardware and software. The guidance is limited to topics of significance to combined hardware/software systems as opposed to hardware-only systems.

1.2 Application. This handbook is intended as a guide for developing system reliability requirements and selecting reliability program tasks for DOD procurements specifying MIL-STD-785.

2 REFERENCED DOCUMENTS

3 TERMS, DEFINITIONS AND ACRONYMS

3.1 Terms. The terms used herein are defined in MIL-STD-721.

3.2 Definitions. Definitions applicable to this document are as follows (Sources for definitions are cited where applicable):

a. Error: The difference between an expected or specified result and the actual result.

b. Failure, Hardware: The termination of the ability of a [hardware] functional unit to perform its required function [ISO78].

c. Failure, Software: The termination of the ability of a [software] functional unit to perform its required function [ISO78].

d. Failure Intensity: The rate of change in the cumulative number of failure occurrences at a given point in time.

e. Failure Rate: The rate at which failures occur over a given time interval.

f. Fault: A defect in the software code that, when executed under certain conditions, causes a failure to occur.

g. Fault Content: The number of faults in the software code.

h. Fault Density: The average number of faults in a specified amount of code (*i.e. N* faults per *M* executable lines of code.

i. Mistake: An action by the software designer or developer that introduces a fault.

j. Software Reliability: The probability that software will not fail over a specified period of time under specified conditions.

3.3 Acronyms. Acronyms used in this document are defined as follows:

BITE - Built-In Test Equipment
FMEA - Failure Modes and Effects Analysis
FMECA - Failure Modes, Effects and Criticality Analysis
MTBCF - Mean Time Between Critical Failures
MTBM - Mean Time Between Maintenance
MTTF - Mean Time To Failure
QA - Quality Assurance

4 RELIABILITY REQUIREMENTS

4.1 General discussion. On development and production programs for systems and equipment, contractual, quantitative requirements are typically specified for both logistics (basic or inherent) and mission reliability. Logistics reliability parameters such as Mean-Time-Between-Maintenance (MTBM) account for all possible occurrences that require logistics support. Mission reliability parameters such as Mean-Time-Between-Critical-Failure (MTBCF) address a system's capability of performing a specific mission. However, where the system comprises both hardware and software, a statement of reliability requirements in terms of minimum MTBM, maximum failure rate, or minimum mission reliability is inadequate for the following reasons:

a. Many software faults cause minor deviations from normal system operation such as incorrect display formats. Specific definitions of one or more failed states are therefore necessary to formulate an effective reliability program.

b. Many software failures are of a temporary or intermittent nature (because software does not physically degrade). Separate requirements for permanent and transient failures may therefore have to be generated.

c. As faults are identified and removed, the reliability of software will tend to increase. Therefore the reliability objective has to be stated as a function of operational exposure and maintenance effort.

All of these factors are also present in hardware only systems but to a much lesser extent. The inclusion of software forces them to be addressed, and this will benefit the overall conduct of reliability programs. Specific factors to be addressed in the formulation of reliability requirements include the following:

Operating mode
Failure severity

Maintenance action
Fault duration
Fault tolerance provisions
Unrepaired faults
Reliability growth

4.2 Detailed discussion

4.2.1 Operating mode. A separate reliability requirement must be established for each system-level operating mode defined in the system specification. A mode is characterized by its requirement to perform a distinct set of functional operations associated with a specific hardware configuration, software path, or hardware/software workload. Examples of operating modes are full-up, reduced capability, and emergency (degraded modes) or ship, aircraft, and spacecraft communication (sectorized modes). Note that operating modes are distinct from missions in that the former reflect internal (system-derived) stresses whereas the latter reflect external stresses (*e.g.* environmental, or effects caused by other systems). A specific operating mode can occur in more than one mission, and a specific mission can involve several operating modes.

4.2.2 Failure Criteria. Most quantitative reliability parameters involve counting failure occurrences or their associated interarrival times. It thus becomes imperative to clearly define what events constitute a failure. This is particularly true for hardware/software systems since the types of potential failure events exceeds that for hardware only systems. Criteria for defining failures appear in the paragraphs that follow.

4.2.2.1 Failure severity. The severity criterion for failure must be clearly stated. Examples include (1) any deviation from the specification, (2) only significant deviations from the specification, or (3) only failures causing service interruptions of more than some threshold value. If necessary, several criteria with separate reliability requirements may be stated.

4.2.2.2 Maintenance action. The criterion for failure in terms of maintenance action is particularly significant where MTBM is an important figure of merit. The definition of maintenance and possibly separate definitions of operational, hardware and software maintenance need to be stated. Examples of this failure criterion include (1) any deviation from normal operating procedures, (2) only actions requiring program restart, or (3) only actions requiring system shut-down.

4.2.2.3 Fault duration. Failures should be defined in terms of the duration of the failure causing event (the fault), distinguishing between permanent, recurrent (intermittent), and singular (transient) faults. Note that the duration is not necessarily an indicator of severity. It is possible for a singular event to cause a very severe failure (a lightning strike) and for a permanent fault to cause only mild degradation (inability to shift to upper case letters). Separate reliability requirements should be stated for permanent and non-permanent faults; the need to distinguish between intermittent and transient failures depends on the application.

4.2.2.4 Fault tolerance provisions. The threshold for involvement of fault tolerance mechanisms in the definition of a failure must be defined. An example of a very strict

criterion is one that counts a fault masked by error corecting code. Less severe criteria do not count faults that are corrected by operation of a fault tolerance mechanism.

4.2.3 Unrepaired faults. The criteria for counting failures while they are being fixed must be specified. By doing so, the ability of software to operate without repair after a failure has occurred will be recognized in the system requirements. This is a drastic deviation from the hardware environment, and it brings about problems in

a. accounting for repetitive failures due to an already reported software fault that has not yet been fixed

b. introducing non-specified, degraded operating modes such as not permitting some transactions that are suspected to cause a failure in the presence of the existing software fault (*e.g.* not permitting station A to send a message to station B while an outgoing message from B has not yet been acknowledged).

4.2.4 Reliability growth. Reliability growth is usually more pronounced in software than in hardware. Therefore, requirements for reliability predictions and modelling need to be tied to a specific time or event in the development cycle. Where only an operational reliability has been defined but monitoring during development is desired, a growth relation has to be defined to permit meaningful assessment of the progress toward the operational reliability goal.

5 RELIABILITY MODELING

5.1 General discussion

Reliability models represent the contribution of individual components and operational modes to the reliability (or the failure probability) of a system. Good reliability models are a prerequisite for all subsequent steps of the reliability program. The key issue that arises from the inclusion of software is to define the exposure to failure such that hardware and software components can be represented in a series arrangement in a reliability block diagram. The following approach is described:

1. Hardware failures are a function of a stress-time product (or summation of stress-time products over different modes of operation).

2. Software failures are a function of code execution time.

3. For each mission operating mode, the failure probability is the sum of the hardware module failure probabilities computed on the basis of stress-time products and of the required software modules computed on the basis of their execution time.

The basic structure of this model is diagrammed in Figure 5-1. As depicted in the figure, mission-related data identify the configuration (both hardware and software), environmental stress factors, and utilization time (operating time for hardware, execution time for software) at the lowest indenture level of combined hardware/software items to be analyzed. The mission

data are combined with corresponding predicted or estimated hardware and software failure rates to obtain probabilities of hardware and software failure. The total item reliability is approximately given by one minus the sum of the hardware and software failure probabilities plus a term accounting for any hardware/software interaction[1]. In this manner, both hardware and software reliability are incorporated in the item reliability prediction. The model is individually applied to each level of degradation the system is expected to experience (*e.g.* by failure severity, fault duration, *etc.*).

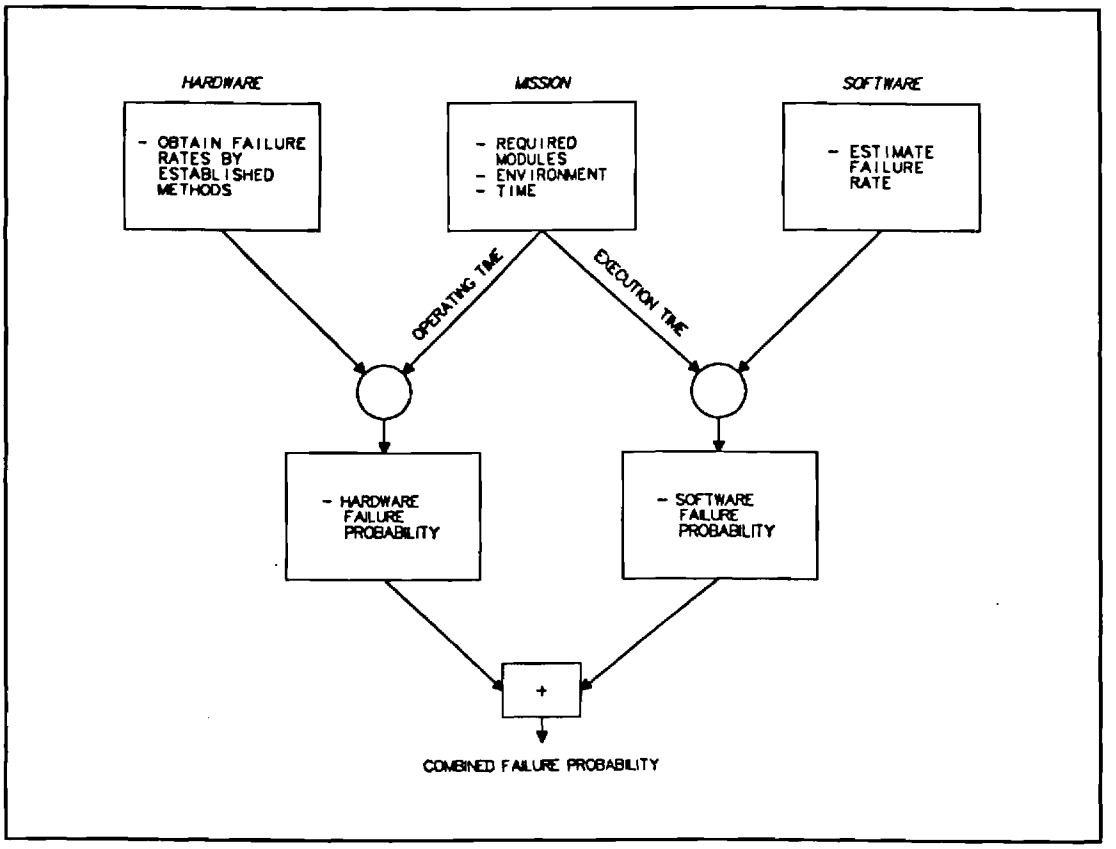

Figure 5-1. Basic Structure of the HW/SW Reliability Model

5.2 Development of the basic model

A reliability block diagram of the hardware portion of the system being modeled is first constructed in accordance with procedures specified in MIL-STD-756B. A block representing

---

[1] A more exact expression is 1 - $(1-F_H)(1-F_S)(1-F_I)$, where the terms involving F are the failure probabilities of hardware, software and hardware/software interactions, respectively.

software is then added in series with each hardware block involved in the operation or storage of the software. Where several hardware blocks are associated with the same software block, a single software block is drawn in series with the hardware blocks. Where several software blocks are associated with the same hardware block, a single hardware block is drawn in series with the software blocks.

The mathematical model associated with the combined hardware/software reliability diagram is constructed by following the same procedure specified in MIL-STD-785B; assuming constant failure rates, series blocks imply reliability product terms while parallel blocks imply failure probability product terms. However, the constant failure rate assumption for software must be carefully considered. During development, the assumption only applies over short time intervals since significant reliability growth typically occurs during this period. For operational systems, the constant failure rate assumption is usually acceptable for both hardware and software [HECH86].

The reliability term for each hardware block is evaluated from the exponential expression

$$R_H(t) = \exp[-\lambda_H t] \tag{5-1}$$

where R(t) is the probability of success, λ_H is the failure rate of the hardware, and t is the operating time interval. Where the failure probability 1-R(t) is less than 0.1, it can be approximated to within 0.5% by:

$$F_H(t) \approx \lambda_H t \tag{5-2}$$

The hardware failure rate is evaluated in accordance with the 200 series Tasks in MIL-STD-756B as discussed below in Section 7.

The reliability term for each software block is evaluated by combining the software failure rate λ_S and the software utilization time τ in the following manner. First, major mission phases and block operating modes are identified. For each mission phase $\emptyset_i$, the duration t_i of the phase and the fraction of time f_{ij} a given mode operates during the phase (*i.e.* the mode duty cycle) are estimated. The resulting data are organized as shown in Table 1. The effective operating time for the jth operational mode over a mission comprising N phases is calculated by the following expression:

$$T_j = \sum_{i=0}^{N} t_i * f_{ij} \tag{5-3}$$

The next step is to decompose the software block into major software components (modules) for which failure intensity predictions will be performed. The procedure for performing these predictions is described in Section 7. For the present discussion, assume the predicted failure rate for the kth component is given by λ_{Sk}. The fraction of time u_{jk} each module is utilized during each operational mode (*i.e.* the module duty cycle) is then estimated. The data are organized as shown in Table 2.

TABLE 1. EFFECTIVE MODE OPERATING TIME

Mission Phase		Op. Mode 1		• • • Op. Mode j		• • • Op. Mode M	
Phase	Time	% On	Time	% On	Time	% On	Time
$Ø_1$	t_1	$q_{11}*100$	t_1*q_{11}	$q_{1j}*100$	t_1*q_{1j}	$q_{1M}*100$	t_1*q_{1M}
⋮							
$Ø_i$	t_i	$q_{i1}*100$	t_i*q_{i1}	$q_{ij}*100$	t_i*q_{ij}	$q_{iM}*100$	t_i*q_{iM}
⋮							
$Ø_N$	t_N	$q_{N1}*100$	t_N*q_{N1}	$q_{Nj}*100$	t_N*q_{Nj}	$q_{NM}*100$	t_N*q_{NM}
	—		———		———		———
	Tot. Msn. Time		Eff. Op. Time τ_1		Eff. Op. Time τ_j		Eff. Op. Time τ_M

TABLE 2. SOFTWARE MODULE UTILIZATION

		S/W Module Duty Cycle					
	Eff. Op.	Module 1		• • • Module k		• • • Module S	
Op. Mode	Time	% On	Time	% On	Time	% On	Time
Mode 1	τ_1	$u_{11}*100$	τ_1*u_{11}	$u_{1k}*100$	τ_1*u_{1k}	$u_{1S}*100$	τ_1*u_{1S}
⋮							
Mode j	τ_j	$u_{j1}*100$	τ_j*u_{j1}	$u_{jk}*100$	τ_j*u_{jk}	$u_{jS}*100$	τ_j*u_{jS}
⋮							
Mode N		$u_{N1}*100$	τ_N*u_{N1}	$u_{Nk}*100$	τ_N*u_{Nk}	$u_{NS}*100$	τ_N*u_{NS}
	τ_N						
			———		———		———
			Util. Time 1		Util. Time k		Util. Time S

The effective software module utilization time is calculated by:

$$\tau_k = T_j * u_{jk} \tag{5-4}$$

Finally, the reliability of each software module is given by the series product expression:

$$R_S(t) = \prod_{\text{all } k} \exp[-\lambda_{Sk}\tau_k] \tag{5-5}$$

In most cases, the expression can be approximated as in equation (5-2) above, in which case the software failure probability is given by:

$$F_S(t) \approx \sum_{\text{all } k} \lambda_{Sk}\tau_k \tag{5-6}$$

The combined hardware/software success probability for a hardware/software block set appearing on the reliability diagram is given by the product of the individual reliability expressions in equations (1) and (5):

$$R_{SYSTEM}(T) = R_H(t) * R_S(t). \tag{5-7}$$

The corresponding reliablity block diagram is shown in part A of Figure 5-2. Alternatively, the total failure probability can be approximated by the sum of equations (2) and (6):

$$F_{SYSTEM}(T) = F_H(t) + F_S(t). \tag{5-8}$$

5.3 Hardware/software interactions

The above procedure assumes independent hardware and software failures when in fact interactions sometimes occur. The most common of these interactions can be described by two types which can be accounted for in the reliability model by the additional term $R_X(t)$ in the combined reliability expression:

$$R_{SYSTEM}(t) = R_H(t) * R_S(t) * R_X(t) \tag{5-9}$$

For cases where 1 - R(t) is small (less than 0.1), the combined failure probability can be approximated by:

$$F(t) = F_H(t) + F_S(t) + F_X(t) \tag{5-10}$$

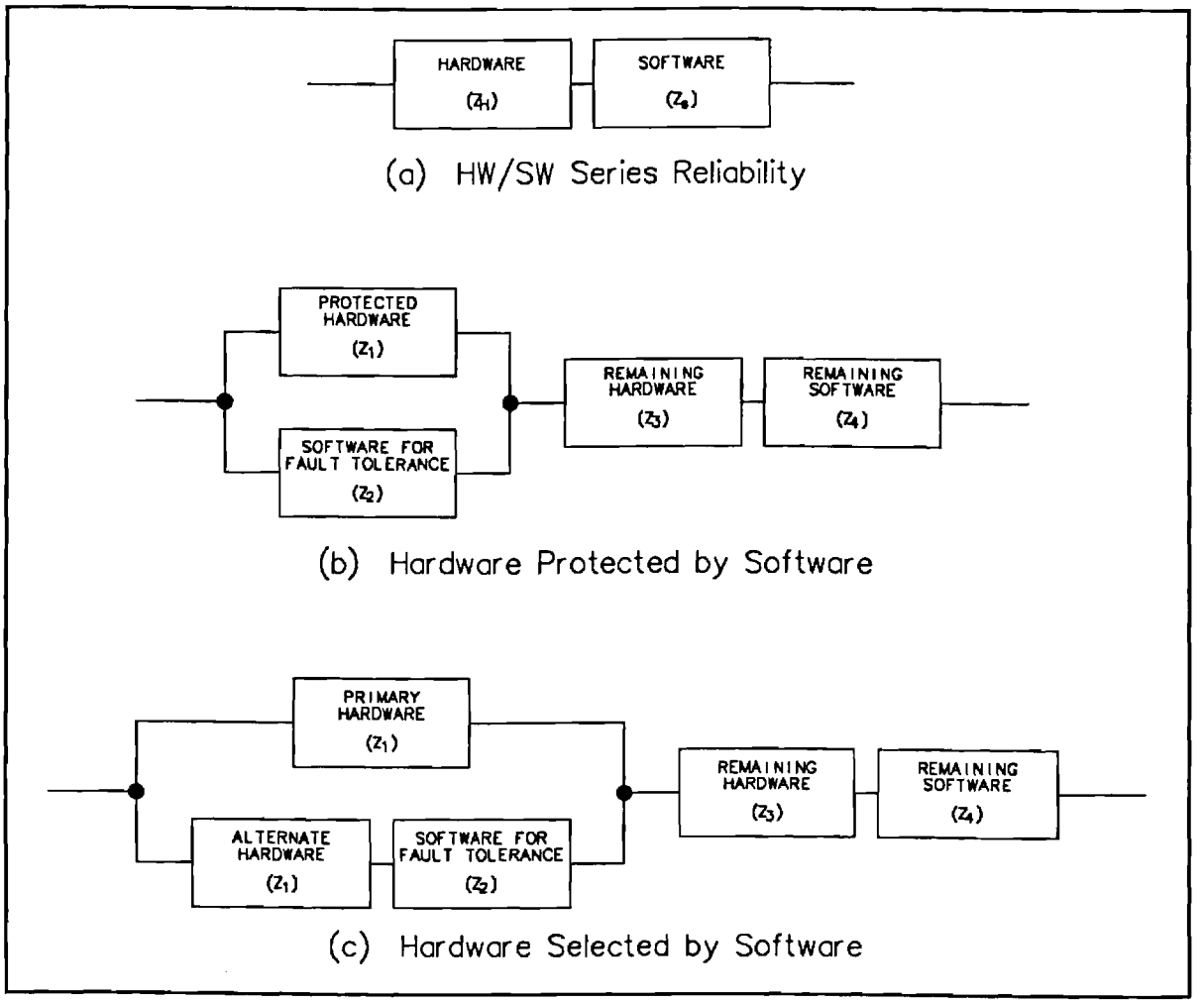

Figure 5-2. Reliability Block Diagrams

In the first type of interaction, software is used to circumvent a transient hardware failure as depicted by the reliability block diagram shown in part B of Figure 5-2. An example is computer memory error detection and correction (EDAC) implemented in software. By methods described in MIL-HDBK-756, the reliability expression corresponding to this case is given by:

$$R_{SYSTEM}(t) = [\exp(-z_1 t) + (1 - \exp(z_1 t))\exp(-z_2 t)] \times \exp(z_3 t) \times \exp(z_4 t) \quad (5\text{-}9)$$

where z_1 is the failure hazard of the protected hardware, z_2 is the hazard of the software implementing the fault tolerance, and z_3 and z_4 are the hazards of the remaining hardware and software, respectively. After expanding the expression and substituting $a = z_2/z_1$ and $k = z_1 t$, the bracketed term involving z_1 and z_2 becomes:

$$R_X(t) = \exp(-k) + \exp(-ak) - e\text{-}(1+a)k \quad (5\text{-}10)$$

In many applications, the software implementing the hardware fault tolerance provision is designed to have a much higher reliability than the protected, *i.e.* the term a is much smaller than one. In this case, it may be convenient to use the approximation

$$R_X(t) \approx \exp(-.5ak) \quad (5\text{-}11)$$

in regard to this type of interaction. The error is indicated by the curves shown in Figure 5-3. As can be seen from the figure, the approximation is quite adequate to support reliability prediction for feasibility studies for values of k less than 1. The approximation should not be used where the validity of extremely small failure probabilities are to be investigated.

The second type of interaction comprises software for reconfiguring hardware in the event of a hardware failure and is depicted in part C of Figure 5-2. An example is software that brings online a backup data storage device in the event the prinmary device fails. The reliability expression corresponding to this case is given by:

$$R_{SYSTEM} = [\exp(-z_1 t) + (1 - \exp(z_1 t))\exp(-z_1 t)\exp(-z_2 t)] \times \exp(z_3 t) \times \exp(z_4 t) \quad (5\text{-}12)$$

where z_1, z_2, z_3 and z_4 are as previously defined. After expanding the expression and substituting $a = z_2/z_1$ and $k = z_1 t$ as before, the bracketed term involving z_1 and z_2 becomes:

$$R_X(t) = \exp(-k) + \exp(-(a+1)k) - e\text{-}(2+a)k \quad (5\text{-}13)$$

Assuming as before that the parameter a is much smaller than one, equation (5-10) can be approximated by

$$R_X(t) \approx \exp(-.5k) \quad (5\text{-}14)$$

for this type of interaction. The error is indicated by the curves shown in Figure 5-4. As can be seen from the figure, the approximation is adequate to support reliability prediction for feasibility studies for values of k less than 1.

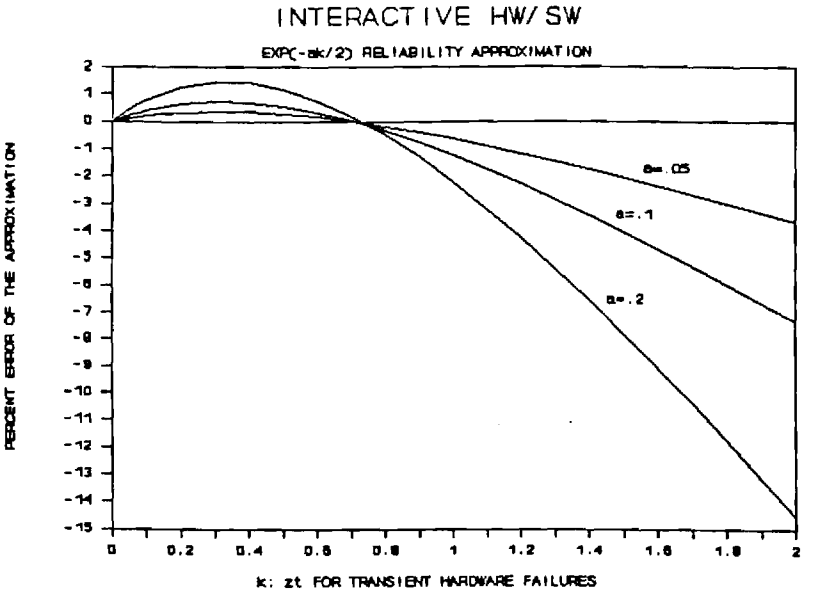

Figure 5-3. Error for Approximating Circumvention Interactions

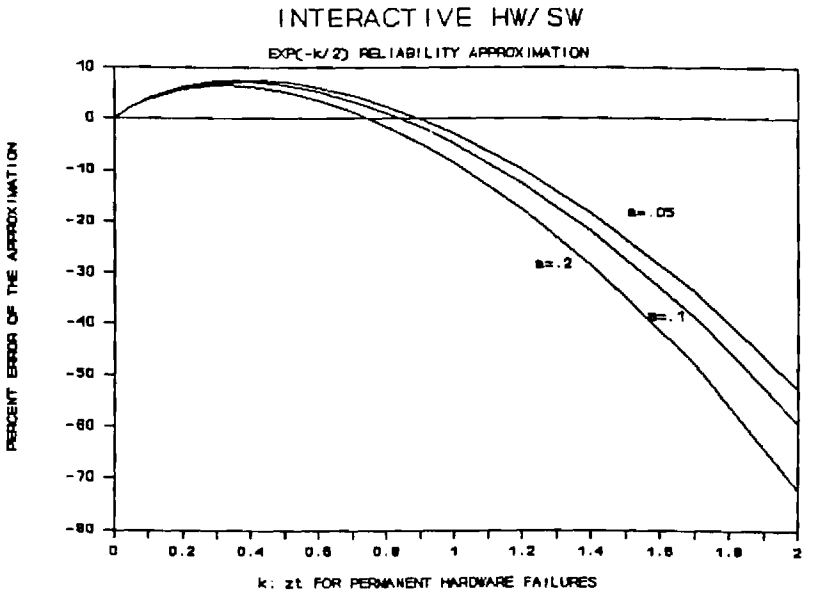

Figure 5-4. Error for Approximating Reconfiguration Interactions

5.4 Example 1

An example of applying modeling procedure to the offensive radar of a fighter aircraft is presented. As a first step, the radar system is decomposed into its hardware and software components and a reliability diagram constructed. The software is assumed to consist of an executive, self-test program (Test), scan, track, and calibrate (Cal) routines. The hardware is decomposed into a power supply (P.S), built-in test equipment (BITE), a pulse generator (Pls Gen), a scan module, and a track module. For this example, the components are assumed to be independent and represented as series elements as in the block diagram appearing in Figure 5-5.

Next, the effective mission time for each operational mode is computed, based on a typical mission profile. As shown in Table 3, the radar is assumed to have four major operational modes: idle, scan, track, and maintenance. The effective time for each mode (column) is computed as the summation of the product $t_i * f_{ij}$ for that column (j = 1 ... 4). Thus, for the idle mode the effective time is 0.1+0.1+0.1+0.1+0.1 = 0.5.

Then, software and hardware component utilization for each operating mode are computed as shown in Tables 4 and 5, respectively. For the hardware BITE module and for the software test and calibration modules the failure rate includes only those failures that can affect the mission. For the basic reliability model a higher failure rate that accounts for all failure modes will have to be used.

The effective hardware and software times are carried over from Table 3. The entries under the hardware module duty cycle are the expected loading or electrical duty cycle of each hardware module for the operating mode (row). Because several hardware modules are operational at the same time, the sum of the hardware duty cycles carries no particular significance. The entries under software module duty cycle represent the fraction of processing time for a given module in each operational mode. The sum of the software duty cycles is exactly unity in all modes because this system provides no parallel processing and the self-test program is scheduled to run whenever no other module is being processed. The utilization is computed as the sum in each column of the effective hardware or software time multiplied by module duty cycle. The failure rate is derived from sources discussed in Section 7.

The failure probability represents the product of utilization time and failure rate. This row is the key to design changes for reliability improvement because it clearly shows which modules affect mission failure probability the most. Although the track module has the highest failure rate among the hardware components, it is seen that there are three modules that contribute considerably more to the mission failure probability and should receive higher priority for reliability improvement (given that the cost per unit of failure reduction is the same in all modules). The mission probability is the sum of all the hardware and software component failure probabilities as given by equation (5-8), or 294.5 x 10^{-6}.

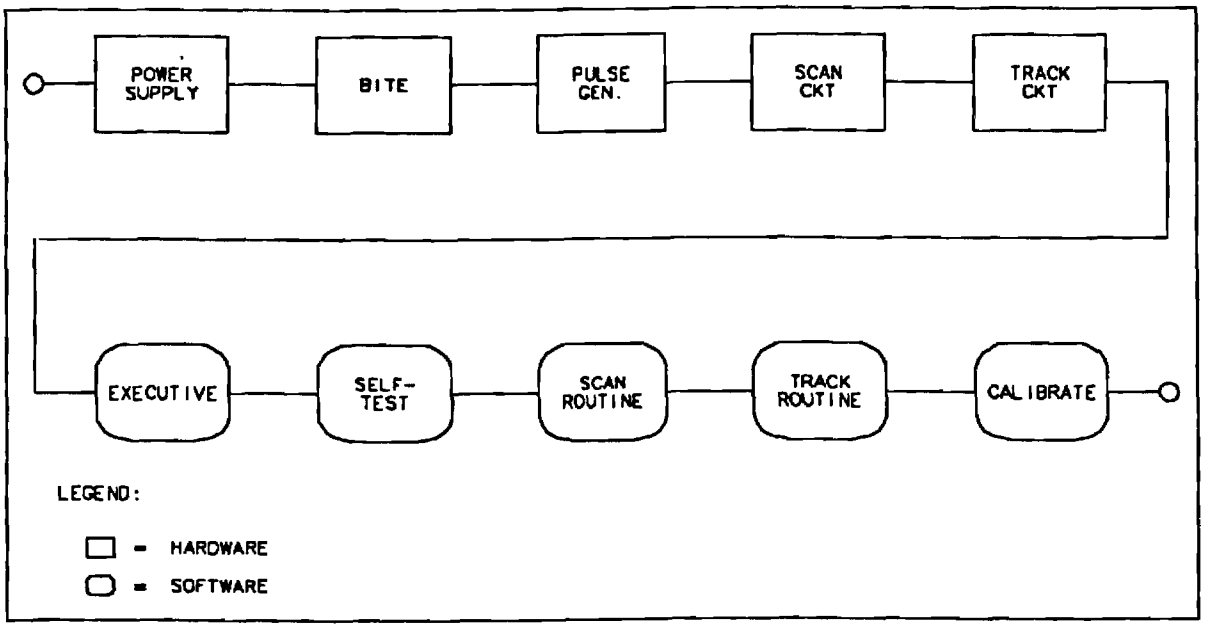

Figure 5-5. Example 1 - Series Reliability

TABLE 3. CALCULATION OF EFFECTIVE MODE DUTY C Y C L E TIME

Mission Phase	Time Hrs.	Mode Duty Cycle							
		Idle		Scan		Track		Maint.	
		Frac.	Hrs.	Frac.	Hrs.	Frac.	Hrs.	Frac.	Hrs.
	t_i	q_{i1}	t_i*q_{i1}	q_{i2}	t_i*q_{i2}	q_{i3}	t_i*q_{i3}	q_{i4}	t_i*q_{i4}
Start-up	0.1	1.0	0.1						
Taxi	0.1	1.0	0.1						
Climb	0.2	0.5	0.1	0.5	0.1				
Loiter	1.0			0.8	0.8	0.2	0.2		
Attack	0.3			0.33	0.1	0.67	0.2		
Return	0.2	0.5	0.1	0.5	0.1				
Land	0.1	1.0	0.1						
Shut-down	0.2							1.0	0.2
Effective Time, Hrs.			0.5		1.1		0.4		0.2

TABLE 4. SOFTWARE MODULE UTILIZATION

Oper. Mode	Eff. Time	S/W Module Duty Cycle									
		Exec		Test		Scan		Track		Calib.	
		Frac.	Time	Frac.	Time	Frac.	Time	Frac.	Time	Frac.	Time
Idle	0.5	0.05	0.025	0.75	0.375	0.05	0.025	0.05	0.025	0.1	0.05
Scan	1.1	0.05	0.055	0.05	0.055	0.9	0.99				
Track	0.4	0.05	0.02	0.05	0.02			0.9	0.36		
Maint.	0.2	0.2	0.04	0.4	0.08					0.4	0.08
Utilization, Hrs			0.1		0.5		1.0		0.4		0.1
Mod. Fail Rate, 10^{-6}/hr			50.0		10.0		100.0		100.0		10.0
Mod. Fail Prob, 10^{-6}			5.0		5.0		100.0		40.0		1.0

Software Failure Probability = 151.0 x 10^{-6}

TABLE 5. HARDWARE COMPONENT UTILIZATION

Oper. Mode	Eff. Time (hrs)	H/W Component Duty Cycle									
		P.S.		BITE		Pls Gen		Scan		Track	
		Frac.	Time	Frac.	Time	Frac.	Time	Frac.	Time	Frac.	Time
Idle	0.5	0.8	0.4	0.8	0.4	1.0	0.5	0.1	0.05	0.05	0.025
Scan	1.1	1.0	1.1	0.5	0.55	1.0	1.1	1.0	1.1	0.05	0.055
Track	0.4	1.0	0.4	0.5	0.2	1.0	0.4	0.1	0.04	1.0	0.4
Maint.	0.2	0.8	0.16	1.0	0.2	0.1	0.02	0.05	0.01	0.05	0.01
Total	2.2										
Utilization, hrs			2.06		1.35		2.02		1.2		0.49
Mod. Fail Rate, 10^{-6}/hr			15.0		11.0		25.0		25.0		35.0
Mod. Fail Prob, 10^{-6}			30.9		14.9		50.5		30.0		17.2

Hardware Failure Probability = 143.5 x 10^{-6}

5.5 Example 2

The preceeding example is extended to address hardware/software interactions. Consider the case where the radar's pulse generator includes a standby transmitter activated by a transmitter fault tolerance (Xmtr FT) software module, and where the tracker includes software fault tolerance (Tracker FT) protecting memory devices against single event upsets. The reliability diagram for this extended radar example is shown in Figure 5-6. Hazard rates for the interacting components and applicable utilization times and hazard rates carried over from Example 1 (Tables 4 and 5) for the remaining components are presented in Table 6. The utilization of the software implementing the hardware fault tolerance was assumed equal to the hardware utilization. Since the product of hazard and operating time (*i.e.* the parameter k) was less than 0.1, equations (5-10) and (5-13) were used rather than the approximations given by equations (5-11) and (5-14).

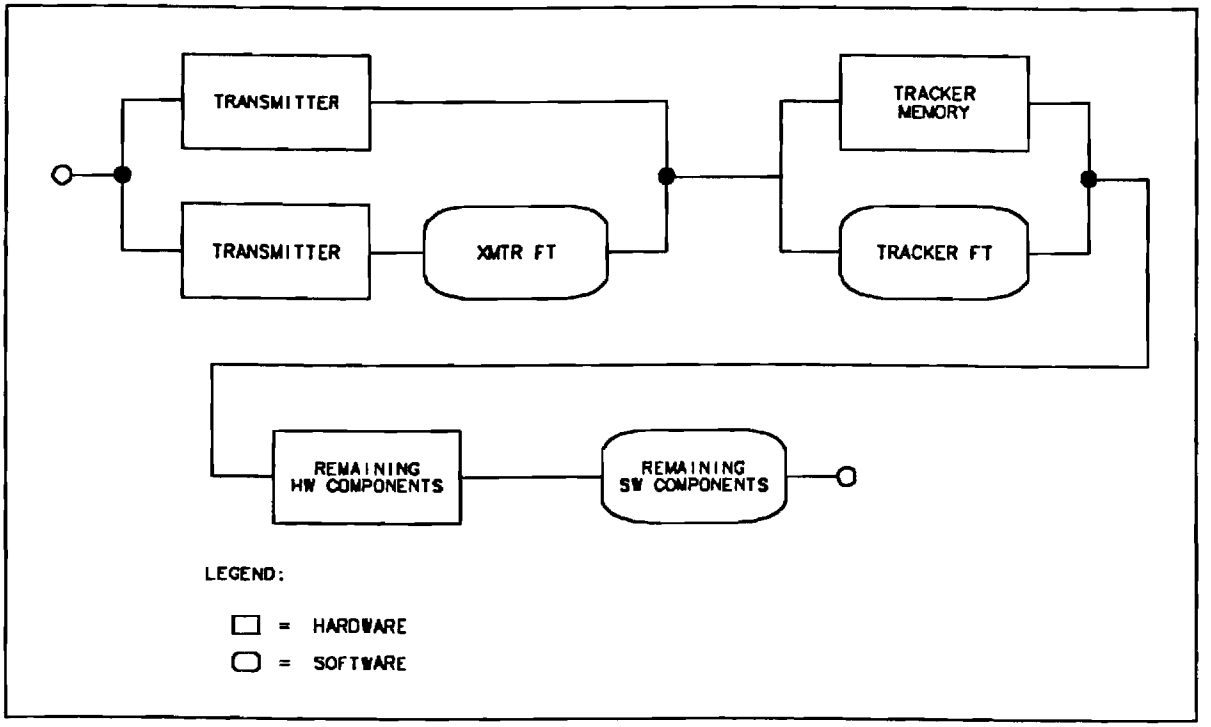

Figure 5-6. Example 2 - Interactive Components

TABLE 6. INTERACTIVE SYSTEM COMPONENTS

Component[1]	Hazard (10^{-6}/hr)	Utilization (hrs)	Fail. Prob. ($\times 10^{-6}$)	Ref. Eqn.
Pulse Gen:				
- Non-interact (H)	15.0	2.02	30.3	Eq. (5-2)
- Interactive:			0.0002	Eq. (5-13)
Xmtr #1 (H)	5.0	2.02		
Xmtr #2 (H)	5.0	2.02		
Xmtr FT (S)	0.5	2.02		
Tracker:				
- Permnt Fail (H)	25.0	0.49	12.3	Eq. (5-2)
- Interactive:			0.0002	Eq. (5-10)
Transnt Fail (H)	10.0	0.49		
EDAC (S)	1.0	0.49		
Remaining HW[3]			75.8	Eq. (5-2)
Remaining SW[4]			151.0	Eq. (5-6)
System Total			269.0	

Notes:

1. Includes hardware (H) and software (S) in interactive or non-interactive configurations.
2. Excludes redundant transmitters.
3. From Table 5.
4. From Table 4.

6 RELIABILITY ALLOCATION

6.1 General discussion

The objective of reliability allocation is to partition the system reliability requirements into reliability requirements for each of the major components and to subsequently partition down to lower levels of assembly or programs. The total reliability allocated at the lower levels must equal the reliability requirement established for the upper level. This is usually implemented in an approximate manner by requiring the sum of the failure probabilities to be no more than the maximum allowable failure probability for the upper level.

This failure probability allocation objective can be stated as

$$\sum_{i=1}^{n} f_i \leq F, \quad \text{for } f_i \geq 0 \text{ and } i = 1 \ldots n \tag{6-1}$$

where F represents the upper limit on system failure probability and f_i the allocated failure probability of the individual components. This inequality can be satisfied by an infinite number of allocations. The economically optimal allocation is arrived at when the cost of marginal failure rate reduction is the same for each component. In practice, the allocation is conducted by considering achievable goals, usually based on experience with similar hardware or software components in the past. The following areas are of particular concern when software is included in the allocation:

- Accounting for differences in the manner in which hardware and software reliability improvements affect cost
- Estimation of the software execution time for each module and mode of operation as identified in the model description
- Adequate consideration of the role of infrequently executed code (*e.g.* that used for system initialization) because many failures have been traced to these components
- Realistic representation of the failure rate of commercial software, particularly operating systems.

6.2 Detailed discussion

6.2.1 Cost considerations. Cost considerations are important in the hardware/software context because hardware reliability improvements usually involve significant recurring costs while software reliability improvements involve primarily non-recurring costs. Thus, for procurements involving large production quantities, it may be much more effective to require a lower failure rate for software whereas for systems that are procured only in small quantities it may be more effective to require a lower failure rate for hardware.

An example of how allocation costs depend upon procurement quantity is illustrated by the hardware/software system defined in Table 7. The overall failure probability of the system over a specified mission is approximately the sum of hardware and software failure

probabilities, *i.e.* 0.02 + 0.02 = 0.04. The table shows the cost of implementing a 25% reduction in the system failure probability by allocating a 50% improvement to either hardware (option 1) or software (option 2) for procurement quantities of 10 systems and 1000 systems.

Option 1 assumes the system allocation is achieved through hardware redundancy; redundant elements are to be provided for 50% of the hardware to achieve the requisite 50% reduction in hardware failure probability. Required expenditures for this option are a recurring cost of approximately 50% of the original recurring hardware cost and a relatively small non-recurring cost for development. For option 2, the 50% reliability improvement has been allocated entirely to the software and achieved through additional testing. The cost of implementing this option is approximately 100% of the original software development cost. (Note the non-linear relationship between software reliability improvement and added test time; longer test times are required for each improvement increment to uncover the fewer and harder to find bugs that remain.) The figures in the bottom row of the table indicate that for the low quantity procurement the hardware allocation costs significantly less than that of the software while for high quantity procurement the hardware allocation costs considerably more than that of the software.

TABLE 7. COST OF 0.01 RELIABILITY IMPROVEMENT

ITEM	OPTION 1 HARDWARE (LOW QTY)	OPTION 2 HARDWARE (HIGH QTY)	SOFTWARE
Procurement Qty	10	1000	-
Unit Recurring Cost	$0.5 M	$0.5 M	-
Total Recurring Cost	$5.0 M	$500 M	-
Non-Recurring Cost	$1.0 M	$1.0 M	$10.0 M
Unit Recurring Improvement	$0.25 M	$0.25 M	-
Total Recurring Improvement	$2.5 M	$250 M	-
Non-Recurring Improvement	$0.1 M	$0.1 M	$10.0 M
Total Cost of Improvement	$2.6 M	$250 M	$10.0 M

6.2.2 Software failure probability. Software failure probability is a function of fault content and execution time, and both must be considered when comparing software modules with those of earlier designs to generate an achievable reliability allocation goal. Fault content is affected by software attributes such as program structure, language, and executable lines of code. A common measure of fault content is fault density, *e.g.* errors per 1000 executable lines of code. However, modules may have an identical fault content but have widely differing failure probabilities due to the extent to which the code is utilized. It is for this reason that software utilization is included in the reliability model presented in Section 5.

The most desirable measure of software utilization is execution frequency, *i.e.* the number of times during a given interval that the computer central processing unit (CPU) executes the program. Comparisons involving execution time are meaningful only if applied to computers of the same type performance and word format. It is misleading to compare the failure rate of a module running on a 16-bit avionics computer operating at 2 MIPS with that of a 60-bit mainframe operating at 20 MIPS. Instead, the failure rates must be normalized (divided) by the product of the word length and instruction speed (*i.e.* 32 MBits/sec and 1200 MBits/sec for the computers described). Because the resulting execution frequency is not usually meaningful in an operational context, it should be used only for the purpose of global comparison.

For programs run on mainframes, execution time can usually be obtained from operating system reports. For cases where execution time cannot be obtained, the computer run time of the modules in question may be used if input/output activity is accounted for and if the modules run on the same computer type (*i.e.* similar performance and word format). Calendar time is only applicable for comparison of failure rates when the workload and execution time for the modules being compared are constant.

6.2.3 Fault consequences and severity

In addition to failure probability, failure consequences and severity must be considered when allocating reliability goals. Consequences of a software failure can range from improper display of a character or character set (*e.g.* loss of upper case text) to total system stoppage requiring manual intervention for restart (*i.e.* a system crash). Severity considers the effect on the overall mission including the possibility of human injury, cost impact and service interruption. The severity of software failures on mission performance can range from negligible to extremely severe, and modules that are likely to cause severe effects should be allocated a lower failure probability. Fault Tree Analysis or functional FMEA are useful tools for evaluating the consequences of software failures and their severity.

When system reliability requirements at the top level distinguish between severe and non-severe failures, the severity apportionment should be carried down to lower level allocations. Where no apportionment exists at higher levels of indenture, the lower level reliability allocation should account for criticality by methods such as applying only to severe failures or being reasonably weighted between severe and non-severe failures. For example, if a failure probability of 0.01 is to be allotted to a specific software component, the allocation could be 0.0001 for high severity failures and 0.0099 for non-severe failures.

6.2.4 Infrequently executed code

Software-based systems typically consist of two types of code; (1) that which is part of the normal operation and frequently accessed, and (2) that which is non-operational and consequently rarely used. Examples of the latter include

- exception handlers (communication channels busy, read errors on disk access, memory parity errors, etc.)
- system initialization
- hardware calibration routines
- special measurements
- computation of long term system statistics and report generation

Prior research has found that infrequently executed code has a higher fault density (faults per source line of code) and fails at a much higher rate (in some cases by two orders of magnitude) than the operational software [MCCA87]. This has been attributed to less thorough testing of rarely used code. If it is desireable to achieve a given incremental reliability improvement at minimum expenditure, the best strategy will be to conduct further testing of the rarely used code at least until its total fault density (counting faults found during test and operation) approaches that of the frequently accessed code.

6.2.5 Commercial software

Failures in commercial software can occur due to

- faults in the code
- problems at the computer interface (*e.g.* the software expects a memory structure or I/O port configuration that differs from the existing one)
- problems at the user or application interface (*e.g.* the software expects a data format or commands that differ from the existing ones).

Data provided by the vendor can only be expected to account for faults in the code. The other contributions must be estimated by the integrator. A good example of reliability problems in commercial software is found in [ADAM85].

7 RELIABILITY PREDICTION

7.1 General discussion

Reliability prediction for combined hardware/software systems requires formulation of reliability (or failure probability) expressions for software consistent with those that have been developed for hardware and defined in MIL-STD-756B. The standard recognizes two major categories of prediction; (1) by similarity, and (2) by elements.

The similarity methods for hardware are directed toward similar circuits or equipments. These methods carry over almost verbatim into software when "program" is substituted for "equipment" and "subroutine" for "circuit". Examples of subroutines covered by this substitution are search operations, sort operations, matrix operations, and Runge-Kutta numerical approximations. The similarity methods for both hardware and software must account for usage as well as type. For hardware, usage implies stress levels and environmental conditions. For software, significant usage parameters are data width (bits), instruction speed (*e.g.* MIPs), operating time (where reliability is to be predicted), and workload.

The element methods for hardware comprise Parts Count and Parts Stress failure rate predictions as defined in MIL-HDBK-217. Unlike the similarity category, the hardware element methods do not carry over directly to software. Software prediction by elements applies to fault content (or fault density when normalized by program length) and considers application area, development environment, test environment and maturity.

The significant factor regarding area of application is whether the program executes in real time or is a batch process. An additional factor is the program's primary function (*e.g.* control, message routing, signal processing, intelligence processing). Factors that may be considered under development environment include program size and structure (complexity), developer/user relationship[2], design and coding languages, automated development and coding tools, and programmer experience. Although the latter factor appears to have a significant impact on fault content, it is difficult to verify and enforce. Test factors include the fraction of project resources allocated for test, the test methodology, and the use of automated test tools. Maturity factors include the fraction of previously used code and the fraction of modified code. These considerations are summarized in Table 8.

There are three general categories of software reliability prediction models:

a. Fault seeding. These models are relevant only when the seeded faults are equivalent to the inherent faults. They are used for estimating the number of inherent faults by equating the uncovery of seeded faults to inherent faults [IEEE88]. This approach was discussed in the early 1970's but has not received much attention recently.

b. Input domain. These models are useful only to the extent that a convenient partition of the input domain can be generated. Software reliability is estimated based upon the number of successful runs compared to the total number of runs where the number of runs for a specific input domain is determined by the domain probability distribution. A generally applicable partitioning is normal input data and exceptional input data. There is little practical expertise with this approach.

c. Reliability growth. These models can be used for estimating either the average number of failures occurring over an interval of time or the average elapsed time between failures.

---

[2] Boehm's classification of organic, semi-detached and embedded as described in RADC-TR-87-171 has been found to be particularly effective in this regard.

TABLE 8. CONSIDERATIONS FOR SOFTWARE PREDICTION BY ELEMENTS

ATTRIBUTE	SIGNIFICANT FACTORS	METRIC RANGE*
Application area	Real time vs. batch process	Not Yet Evaluated
	Primary function	.0018 - .013
Development environment	Program structure:	
	Complexity	.8 - 1.5
	Modularity	.9 - 2
	Anomaly Management	.9 - 1.1
	Traceability	1 - 1.1
	Quality	1 - 1.1
	Standardization	.75 - 1.5
	Developer/user relationship	.76 - 1.3
	Design and coding languages	1 - 1.4
	Automated tools	Not Yet Evaluated
	Programmer experience	Not Yet Evaluated
Test environment	Test resources	Not Yet Evaluated
	Test methodology	Not Yet Evaluated
	Automated tools	Not Yet Evaluated
Maturity	Extent of re-used code	Not Yet Evaluated
	Extent of modified code	Not Yet Evaluated

* Based on [IITR87]

The third category is considered superior in capability and applicability [MUSA87] and will be discussed in greater length. These models generally apply to predicting reliability growth since they relate program usage or test time to the number of faults exposed during a test phase. Their hardware counterparts are models such as Duane and AMSAA that predict hardware reliability growth in terms of decreasing failure rate (increasing MTBF) [MH189]. No single model is applicable for all cases since software reliability growth is heavily dependent on management decisions and on the extent to which software faults are mutually dependent (*i.e.* fault B cannot be exposed until fault A has been detected and repaired).

The failure intensity for a software component *at a specific time* has the form[3]:

---

[3] See [MUSA87], pg. 121, where $U = fK$. The equivalence of the failure intensity expression derived in this report and in the referenced work implies that Musa's development of confidence factors is also applicable here (*ibid*, pp. 270-271).

$$\lambda = \omega \times U \tag{7-1}$$

where λ is the failure intensity, ω is the fault content, and U is the rate at which faults are being exposed (*i.e.* failure occurrences per fault per unit time). The latter term is a function of the stress placed on the software by the test cases, while ω is a function of the fault density and of the length of the code. Time dependency is introduced by assuming the failure rate is proportional to the fault correction rate:

$$\lambda(\tau) = \omega(\tau) \times U \tag{7-2}$$

Assuming perfect debugging (*i.e.* each fault exposed is properly corrected without introducing new failures or correcting additional faults), the rate of failure occurrences must equal the rate at which the fault content is being reduced:

$$\lambda(\tau) = \frac{d\omega(\tau)}{d\tau} \tag{7-3}$$

A time-dependent expression for the software failue rate is obtained from equations 7-2 and 7-3 and is given by:

$$\lambda(\tau) = \lambda_0 U e^{-U\tau} \tag{7-4}$$

In using these equations, the fault density (*i.e.* ω divided by program size) and/or the exposure rate U should be estimated on the basis of experience on similar software. When such experience is not available, the following default values may be used:

Fault density: .005/SLOC
Exposure rate: .005/CPU hr for a computer processing 10^6 source lines/sec

7.1.1 Quality Control

Although it may intuitively appear that quality assurance (QA) should have a significant effect on reducing software fault content, measurable evidence of this has been scanty. This should not be construed as detracting from QA; its benefit to software maintenance has been clearly documented. In addition, the team aspects of development programs subjected to total quality management (TQM) can lead to feedback of modes and causes of failures to developers and produce an overall improvement in product reliability and performance. However, the effect of software quality on fault content is yet to be quantitatively assessed.

The reliability of a hardware/software system is adversely affected by external stresses caused by changes in requirements or test methodologies and by management actions regarding workload and job composition. Statistical process control charts used on programs subjected to

TQM have been found to be useful for measuring variations in software stress and failure rate[4]. However, care must be exercised in interpreting these measurements. For example, a high failure rate during development and test may be due to (1) a high fault content or (2) a high rate of uncovering faults by effective testing. The difference between these two can only be judged after several months of observation.

7.2 Detailed discussion

The reliability prediction expressions defined in MIL-STD-756 must be selectively applied at different stages of the development cycle in order to satisfy the prediction needs stated in Table 9. The detailed discussion that follows regarding these needs concentrates on parts of the development cycle that do not usually permit reliability testing (either as part of development or operational testing), because it is during the early phases that a realistic and accurate reliability prediction is most needed.

TABLE 9. RELIABILITY PREDICTION NEEDS DURING DEVELOPMENT

Development Activity	Minimum Data Requirements	Objective
Feasibility Studies	Hypothetical system composition	Reliability goal
System Design	Actual system composition	Feasibility of goal
Prelim. Detailed Design	Parts count, code estimate	Compliance with goal
Detailed Design	Parts application, lines of code	Refined compliance

7.2.1 Feasibility studies

The purpose of reliability prediction during feasibility studies is to establish a realistic reliability goal. This goal must address the missions, mission phases, and operational environment identified in the Operational or Technical Needs document. If a reliability goal is stated in the document, the modeling task should demonstrate how this goal can be met. In the more likely case that no specific goal is identified, the reliability analyst may model a hypothetical system

[4] During test, the adoption of a new test methodology (*e.g.* changing from structural to functional testing) or shifting of responsibility to a new test team will frequently produce a sharp increase in failure rate.

and assess, together with other project participants, whether the predicted reliability will be satisfactory. As a minimum this assessment must consider:

- predicted permanent failure rates for each mission phase with identification of the assumed environment and parts quality level

- partitioning of these failures into hardware/software/other categories, and statement of assumptions regarding redundancy or protective mechanisms

- predicted reliability (with reliability growth) for one or more typical missions; where redundancy is employed the predicted basic reliability shall also be stated as an aid in maintenance planning.

Since no design is available during feasibility studies, a methodology based on parts count is useless. The Similar Item Method (Method 2001 in MIL-STD-756) is applicable when augmented with more definitive requirements such as the following:

- failure experience on the predecessor equipment to be identified by equipment maturity, assessment for deterministic failure modes, environmental stress level, operating mode, duty cycle, and hardware/software/other classification of failures

- detailed comparison between the proposed and predecessor equipment with regard to the above characteristics and complexity of hardware, software, and personnel components and interfaces

- description of the amount of re-use of existing hardware and software components for each functional component of the proposed system.

7.2.2 System design

Typically, Reliability Allocation (MIL-STD-785 Task 202) rather than reliability prediction is required during system design. The formal description of reliability allocation concentrates on the partitioning of reliability goals so that component reliability requirements are consistent with the system reliability goal. Although this is strictly speaking not a reliability prediction activity, it is generally realized that it is necessary to show the feasibility of the allocation by reconciling it with what can reasonably be expected from each component.

7.2.3 Preliminary design

The following presents a methodology for Preliminary Prediction (to support the allocation) based on the system definition and gross indicators of its composition that should be available at that stage: estimates of the number of ICs for hardware, number of modules for software, and number and type of interfaces the "other" category. The Active Element Group Method (MIL-STD-756, Method 2003) can be used as a starting point for the preliminary prediction. This must be augmented by consideration of the application factors listed above for the Feasibility Studies and by:

- considering not only numbers but also complexity of the ICs (or hybrids), software modules, and interfaces

- adjusting the prediction for stress levels and duty cycles for hardware and usage factors for software

- requiring allowance for a "design growth factor" to account for the traditional underestimate of the hardware and software components needed to meet the specification for the function.

During preliminary design hardware components are expected to be defined at least to the level where an approximate part count can be furnished, and for software components an estimate of the lines of code in each of the major execution areas (operating system, real-time application code, non-r/t application code, support programs, etc.) should be available. Both internal and external interfaces will also be sufficiently defined at this time to permit a reasonable assessment of their complexity. With this information it is possible to generate reliability prediction models which can establish whether the design complies with the reliability goals and their allocation.

Unless the system does not employ internal redundancy at least two models should be considered:

- the basic reliability model which is concerned with the prediction of all failures, regardless of their mission impact; this model supports maintenance and logistics assessments

- the mission reliability model which is concerned only with failures that impact the mission; this model supports mission planning and system effectiveness evaluation.

Additional models may be required for multi-purpose systems, *e.g.*, to model the effect of failures that affect some missions but not others, or to account for operation in degraded modes.

The basic methodology for reliability prediction at the preliminary design stage is based on the Parts Count Method (MIL-STD-756, Method 2004) modified to account for effective mission time for each operational mode based upon a typical mission profile. An example of this approach has been described in Section 5.3.

7.2.4 Detailed design

In detailed design every part is defined, and electrical and thermal stresses are determined. This permits the more accurate parts stress analysis method of reliability prediction to be applied (MIL-STD-756, Method 2005). At this time separate predictions can be generated for transient and permanent failures. Where effective means are provided to circumvent the effects of transient failures, all or a substantial fraction can be eliminated from the system reliability summation.

Detailed design is also an appropriate time to consider reliability growth predictions. The extent of reliability growth is dependent on the scope of test programs and the existence of a disciplined Failure Reporting, Analysis and Corrective Action System (FRACAS). Guidance for reliability growth testing and prediction is found in MIL-HDBK-781 and MIL-HDBK-189. In all other respects the reliability methodology is identical to that used in preliminary design. Computer programs and interfaces are in their final form at this stage of the development cycle and previous failure rate estimates should be updated with the current information.

8 FAILURE MODES AND EFFECTS ANALYSIS (FMEA)

8.1 General Discussion

FMEA is a part of Failure Modes, Effects and Criticality Analysis (FMECA) that is defined in Task 204 of MIL-STD-785 and for which further requirements are established in MIL-STD-1629. FMEA can aid in the reliability allocation process by

- identifying avoidable weaknesses that can be eliminated in circuit and system design
- focusing the need for fault tolerance on unavoidable reliability weaknesses
- identifying failure modes and their effects so that suitable self-test or external test provisions can be designed and the benefits of these accounted for in allocation.

There is nothing in the requirements or structure of an FMEA that prevents software from being considered as an element that can contribute to failure. Unfortunately the format and procedures for FMEA are not well adapted to the complex circuit and logic structures encountered in current hardware/software systems. The failure modes of even the simplest microprocessor can produce too many different effects for effective analysis. However, if types of events leading to serious top level failures can be identified by fault tree analysis, an FMEA that concentrates on potential causes of such events can be productive.

Fault tree analysis has been in use for many years in the hardware field and is a recognized technique in MIL-STD-785 as well as in MIL-STD-882 (System Safety Programs). The purpose of fault tree analysis is to identify low level failure modes that can, singly or in combination, produce significant failures at a high level. It is a top-down procedure which starts with a given failure condition at the system level and then identifies events at successively lower levels that can contribute to the top event.

An additional consideration for software FMEA is the assignment of severity factors to postulated failure modes as defined in MIL-STD-1629. More specific descriptors than those defined by the standard are required for the severity of software failures. As discussed under the heading Reliability Allocation (specifically, paragraph 6.2.3), the manifestation of a failure is only very vaguely dependent on the nature of the fault that caused it. A fault may be simple and its removal easily accomplished, yet its consequences in terms of system failure can be quite severe. On the other hand, a subtle, complex fault that is difficult to remove may only cause a minor system failure. Thus, it is imperative to properly classify failure (rather

than fault) severity in order to bring to bear appropriate resources for addressing the causes of failure.

9 RELIABILITY DEVELOPMENT/GROWTH TESTING (RDGT) PROGRAM

9.1 General Discussion

In general, the objective of RDGT is to employ a test, analyze and fix process to remove as many causes of systematic (design) faults as possible and to obtain a degree of assurance that the system will meet the stated reliability requirements. Test conditions involve actual, simulated or accelerated mission environments.

For hardware, a distinction is made between failures caused by defects in workmanship or parts and failures caused by design deficiencies. The former are removed by screening or attrition where their removal contributes to the reliability growth of the unit under test. Whenever these random failures are excessive, QA procedures will be tightened. The latter are removed during analyze and fix portions of the testing program; the removal involves design changes which benefit all subsequent copies of the item.

For software, reliability growth during development and test is taken for granted since software faults are inherently design faults and code changes in response to failures are intended to eleminate these faults.

9.2 Models

The most popular of the various hardware oriented mathematical models developed for analyzing reliability growth have been the Duane model [DUAN64] and the AMSAA model [CROW75]. These along with various others are described in MIL-HDBK-189.

Various mathematical relations have been proposed to model the effect of "debugging" on software reliability. Earlier studies indicate that even when considerable care was exercised to eliminate external factors, "good" fits (chi-square test at the 0.05 level) were achieved for only 28% of the tested programs and "no convergence" (meaning lack of consistent reliability growth for several sampling periods) was reported for 57% of the programs [ANGU83].

The best documented of the software reliability growth models are those developed by John Musa [MUSA87]. They assume that the failure rate is either proportional to, or a logarithmic function of, the remaining fault content, and that the detection of faults is a function of execution time (the are therefore referred to as *execution time* models). The Musa models are based on statistical estimations and provide confidence limits on the future failure rate based on past observations. Computer programs for data collection and analysis based on this model are also available.

Many factors can obscure the observation of reliability growth in software development, such as requirements changes, personnel turn-over, and differences in test procedures. These changes, if allowed to occur during software reliability growth testing without being properly accounted for,

will negate the test results. For example, the addition or modification of a system requirement can be expected to generate a negative perturbation in the growth curve. A change in usage (*e.g.* from two-dimensional input data to three dimensions) or in hardware (*e.g.* from a lower performance computer to a higher one) will produce a similar result. The benefits of introducing such changes must be weighed against the perturbations in growth testing that they cause. However, after the perturbation has stabilized, one would expect growth rate to continue as before, *i.e.* the slope of the growth curve before and after the interruption should be similar, and the projected growth curve can be corrected by an appropriately shift along the ordinate.

9.3 Test Strategies

RDGT for software is usually conducted separately from hardware because of differences in test strategies. Hardware is typically stressed by environmental exposure. During hardware tests, diagnostic software routines are used that are typically not representative of the mission software. The environmental conditions imposed upon the hardware during test is both unnecessary for software and presents an impediment to "debugging". Software is stressed by running under conditions of high workload and frequent exception handling, neither of which are necessarily appropriate for stressing hardware. Instrumentation for monitoring performance and uncovering failures are often quite different between hardware and software. Faults associated with the interface between hardware and software are detected during integration testing which may or may not be part of RDGT.

Integration of the software RDGT with other developmental testing enables acquisition of more test time and earlier application of the growth program within the development phase but requires more carefull analysis of the data and test conditions. As failures are exposed and solutions identified, several options are available. Fixes can be cut in immediately, their implementation can be delayed until the end of the test phase or later, or they can they can worked-around by changes in documentation or procedures. Each of these options will have a unique impact on growth rate, program cost and schedule. Immediate implementation can unpredictably affect growth from planned levels since new failure patterns can be triggered as a result of the fixes. However, this option maximizes the test time available for observing and correcting these failures. It should also be noted that cost and time constraints frequently dictate that testing continue in parallel with design of fixes for known failures.

Test strategies for software require the following additional considerations. An optimum test strategy is one that uncovers the greatest number of faults in the least amount of program execution time. One approach is to select test input states randomly with a probability distribution similar to that encountered during a field mission. Here we distinguish between *deterministic* or *random* selection of input states. The former implies the test input sequence is specified with certainty and can lead to inadvertent test biasing. The latter implies various alternative sequences are selected arbitrarily and can lead to unrealistic or non-optimum tests. By combining both selection processes (random selection within a predetermined distribution) we avoid their individual deficiencies. Other areas of consideration regarding test strategy include:

- non-repetition of selected inputs (selection without replacement vs. regression testing)

- grouping similar input and selecting one input per group (partition testing)

- applying input sequences that can trigger the occurence of certain fault patterns (mutation testing)

9.4 Data Collection Procedures

Operationally, data collection is required to objectively track reliability growth against a planned growth curve and to project future growth. A number of problem areas involving data collection for supporting reliability growth (particularly software related) have been recognized in the literature. (See for example [SOIS85], [ANGU83].) These problem areas include:

- Erroneously attributing software failures to hardware or vice versa.

- Improperly recording the correction of a software fault as a system "enhancement".

- Erroneously reporting a test procedure deficiency as a system failure.

- Improperly recording the time of the failure occurrence.

- Insufficient coverage of all mission modes rather than a few test scenarios.

- Insufficient range of test inputs.

- Unmotivated data collectors.

Simplification of data collection requirements, consistent error classification definitions, and automated techniques such as menu-formatted input requests aid in overcoming many of these problems.

9.5 Simultaneous Random Testing

Reliability growth measurement for systems having very high reliability requirements presents special problems. Providing sufficient test time for a single copy of the system is typically impractical. In this case, data can be collected from S multiple copies of the system running simultaneously for H hours. The failure data can then be evaluated as if a single copy running SH hours had been tested.

It is possible that more than one failure will be reported during the same reporting interval. Since reliability growth models do not accommodate concurrent failures, the distribution of the failure data must be randomized within each reporting interval and statistical calculations applied for estimating the expected number of failures.

10 RELIABILITY QUALIFICATION TEST (RQT) PROGRAM

10.1 General Discussion

When formualting an RQT plan, consideration should be given to testing hardware and software separately. In doing so, hardware testing would be guided by MIL-STD-781 and software by DOD-STD-2167 and -2168. Advantages for separating these tests include:

- Software testing can be conducted in a room temperature environment at much lower cost than over a simulated operational profile environment.

- For hardware test, it is frequently desirable to run diagnostic programs continuously or very frequently, and this conflicts with the requirements for software.

- While demonstration of the specified MTBF is an accepted technique for hardware, it is not necessarily so for software where demonstration of a minimum failure free period may be preferred.

- The allocation of test time can be tailored to the separate hardware and software reliability goals, and testing can be conducted at different points in the development cycle.

The major disadvantages of separating the test is that the combined system reliability will have to be generated analyticaly and that some hardware/software interactions may go unnoticed. The general methodology for combining hardware and software reliability described under Allocation in this handbook is suitable for resolving the first issue, and the second is addressed by other testing that is being conducted during development, notably system integration.

11 PRODUCTION RELIABILITY ACCEPTANCE TEST (PRAT) PROGRAM

The purpose of PRAT is to insure that the manufacturing process does not degrade the product reliability. PRAT is applicable only to hardware because there is no production variability associated with software.

REFERENCES

ADAM84 E. N. Adams, "Optimizing Preventive Service of Software Products", *IBM Journal of Research & Development*, pp 2-14, Jan. 1984.

ANGU83 J. E. Angus *et al.*, *Reliability Model Demonstration Study*, RADC TR-83-207, August 1983.

CROW75 L. H. Crow, "On Tracking Reliability Growth", *Proc. RAMS*, 1975, pp. 438-443.

DUAN64 J. T. Duane, "Learning Curve Approach to Reliability Monitoring", *IEEE Trans. Aerospace*, 1964, pp. 563-566.

HECH86 H. Hecht and M. Hecht, "Software Reliability in the System Context", *IEEE Trans. on Software Engineering*, January 1986.

IEEE88 IEEE Draft Standard, *A Standard Dictionary of Measures to Produce Reliable Software*, cat. no. P982.1/D3.0, May 1988.

ISO78 *International Standards Organization, Data Processing - Vocabulary, ISO 2382/XIV*, 1978.

MCCA87 J. McCall, System Reliability Prediction Study, RADC final report under contract no. F30602-85-C-0311, May 1987.

MH189 Military Handbook, *Reliability Growth Management*, MIL-HDBK-189, Feb. 1981.

MUSA87 J. D. Musa et al., *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill Book Co., 1987.

SOIS85 E. C. Soistman, Impact of Hardware/Software Faults on System Reliability, RADC final report, Martin-Marietta report no. OR 18,173, April 1985.