(1)

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio.

1

AFIT/GCE/ENG/91-09

A STANDARDIZED SOFTWARE RELIABILITY MEASUREMENT METHODOLOGY

THESIS

Joseph J. Stanko
Captain, USAF

AFIT/GCE/ENG/91-09

AFIT/GCE/ENG/91-09

A STANDARDIZED SOFTWARE RELIABILITY MEASUREMENT METHODOLOGY

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Computer Engineering

Joseph J. Stanko, B.S., M.S.

Captain, USAF

December, 1991

Preface

The purpose of this study was to determine if software reliability models can be applied to the Operational Test and Evaluation (OT&E) of a weapon system and, if this was the case, to implement a selected model.

An extensive review of current literature and research efforts was performed to identify the candidate models for evaluation and possible implementation. Models were evaluated based on predictive validity, capability, quality of assumptions, applicability to the finite-time environment, simplicity of design, diversity and applicability of output, and capability to use existing initial data. From these, the Musa Execution Time model and Musa-Okumoto Logarithmic Poisson Execution Time model were selected for implementation. The implementation was tested using data supplied by Headquarters Air Force Operational Test and Evaluation Center (HQ AFOTEC).

I would like to thank Capt Jim Cardow, my thesis advisor, for his guidance, suggestions, and especially the recurring question "What are you trying to do?" I would also like to thank Lt Col Lawlis and Maj Howatt for reviewing all the drafts and helping me to remember that there is a forrest and not just one tree. I also thank Dr Moore for reviewing my derivations and providing statistical insight.

A special thank you goes to my wife Lynn, and children Devon, Cheryl, and Cara, for their patience, understanding, and support throughout the past eighteen months.

Finally, I would like to give glory to the Lord Jesus Christ and thank Him for providing me an opportunity to learn and grow during the AFIT experience.

Joseph J. Stanko

Table of Contents

Page

List of Figures

List of Tables

AFIT/GCE/ENG/91-09

Abstract

Current Air Force practice is to perform Operational Test and Evaluation (OT&E) for each new weapon system. In support of this, Headquarters Air Force Operational Test and Evaluation Center (HQ AFOTEC) is responsible for measuring both suitability and effectiveness. While suitability is adequately measured, the current effort only addresses hardware effectiveness, or at best, system effectiveness. Since tools and metrics are in place for software suitability assessments related to OT&E (for example, software maintainability), there should be some effective way of measuring the operational effectiveness of software. Currently, HQ AFOTEC/LG5 has a data collection tool for collecting software failure data to analyze software maturity. This thesis proposes that the LG5 software maturity database could be used as the baseline for a software reliability metric that would map to the finite time OT&E environment.

This study does not predict software reliability, nor does it attempt to define what constitutes reliable software. Instead, this study evaluates software reliability measurement mapped to finite OT&E time frames (i.e.-failures per flight hour). This evaluation is conducted for several software reliability models, with two candidate models chosen based on the following criteria: predictive validity; capability; quality of assumptions; applicability to the finite-time environment; simplicity of design; diversity and applicability of output; and capability to use existing initial data.

Implementation of the candidate models was accomplished for an office computer environment to permit use by OT&E test teams at various locations. Testing was performed based on actual OT&E software maturity data.

A STANDARDIZED SOFTWARE RELIABILITY MEASUREMENT METHODOLOGY

I. Introduction

The overall reliability of new and modified weapon systems is of major importance to the United States Air Force (USAF), and is discussed in recent standards and documents that address system reliability and maintainability [54:355]. Indeed, many authors have addressed the need for software reliability evaluation, both in journals and in books (reference bibliography). While hardware reliability can be virtually guaranteed at delivery, the delivery of reliable software is not as predictable, and becomes the critical factor in determining *system* reliability [50:190]. This thesis explores the possibility of implementing software reliability measurement as part of the Operational Test and Evaluation (OT&E) of United States Air Force (USAF) weapon systems, with the goal of identifying one model and methodology that is appropriate for use in the Initial OT&E (IOT&E) phase. As Headquarters Air Force Operational Test and Evaluation Center (*HQ AFOTEC*) is responsible for conducting OT&E on USAF weapon systems, the results of this thesis, as well as a proposed implementation methodology, are then submitted to HQ AFOTEC for possible inclusion in their software evaluation efforts.

This chapter provides the background of software development and testing, and identifies the problem with software operational testing. The following sections will define hardware and software reliability, establish the scope of this thesis effort, identify applicable assumptions, and describe the research approach.

1.1 Background

The complexity of software in future systems will be at least an order of magnitude above that of current systems, which is even now too complex for one individual to grasp [13:3.5]. Software complexity is one of the factors affecting the overall software cost [30:12-6],[82:522]. Henry and Kafura state, "reducing cost and increasing quality are compatible goals which can be achieved when the complexity of the software structure is properly controlled" [37:510]. With respect to software cost, Myers suggests, "the high cost of software is largely due to reliability problems" [65:12]. Therefore, a software cost trend might be an indicator of the underlying complexity of the code and development effort, which could also be closely related to the software's reliability.

An increasing trend in software cost was first identified in a study by the Rand Corporation for the Air Force, and reported in [11] and [77] as a substantial increase in percentage of software cost accompanied by a corresponding decrease in percentage of hardware cost (see Figure 1.1) [11:1227],[77:11].

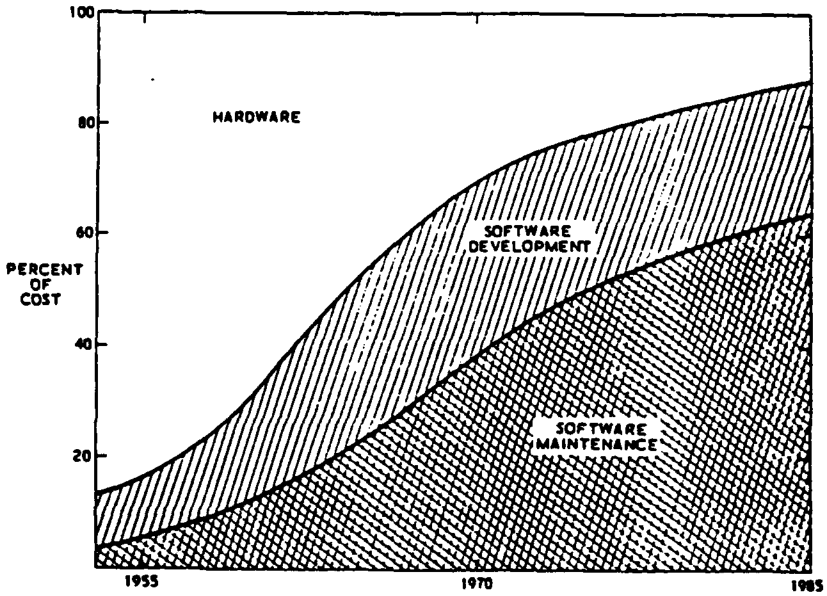

Figure 1.1. Hardware and Software Cost Trends (Reprinted with permission from IEEE)

1.1.1 Air Force Perspective. Increased software cost is inclusive of software development and support. The Air Force has doubled its spending on software development and support from $4 billion in 1985 to $8 billion in 1989 [72:71]. A recent study of 37 Air Force Mission Critical Computer Resource (MCCR) projects evaluated five application areas: avionics; communications; command, control, communication, and intelligence; electronic warfare; and radar systems [81:6]. The study stated the frequency and severity of change in software size contributes to cost overruns, and for three projects the actual amount of software developed for the Air Force exceeded the original estimate made at contract award by 100% [81:7].

Corresponding to increasing software costs, the size of weapon system software has increased dramatically, and will continue to increase. This increase was projected by Boehm in 1976 and reported in [77] (see Figure 1.2). Current estimates of the amount of software developed for DoD weapon systems have verified this trend (see Figure 1.3) [15:48].

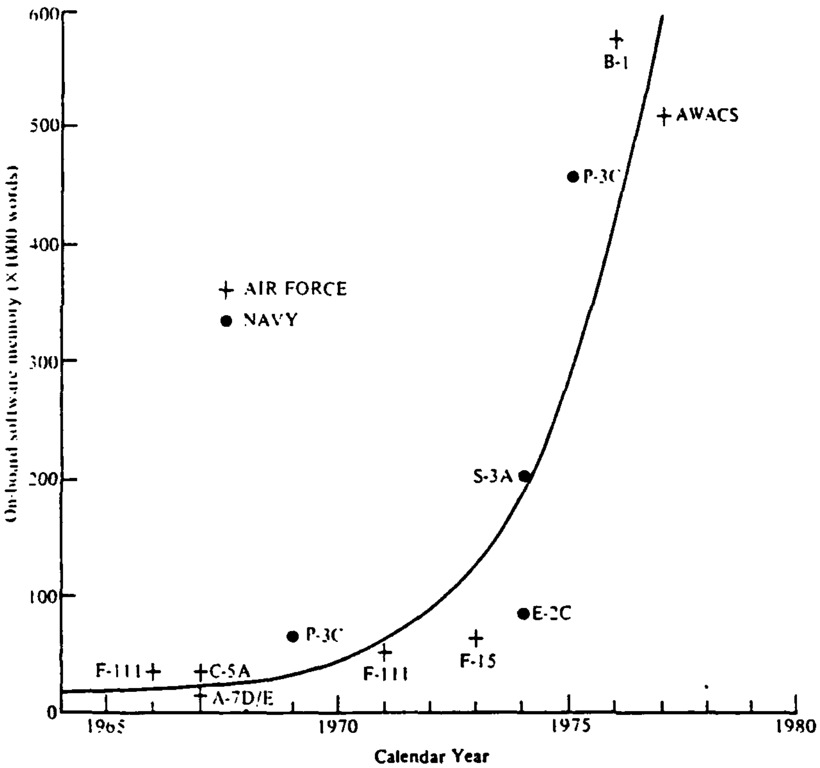

Figure 1.2. Projected Growth in Software Memory Requirements (Reprinted with permission from McGraw-Hill Book Company)

As software size increases, so will the task of software testing. Lieutenant Colonel Shumskas, of the Office of the Secretary of Defense, responsible for Air Force Test and Evaluation (T&E), suggested the following:

> It is possible to reduce acquisition costs, test in particular, and provide software intensive systems with increased reliability through the implementation of a proposed paradigm for a balanced T&E software approach utilizing a combination of statistical process control and test methodologies [78:1-1].

A disciplined test methodology could then help reduce, or at least stabilize, the cost of software.

With respect to software test and evaluation of a weapon system, current Air Force practice is to perform Operational Test and Evaluation (OT&E) under the direction of Headquarters Air

▲ Manned Systems

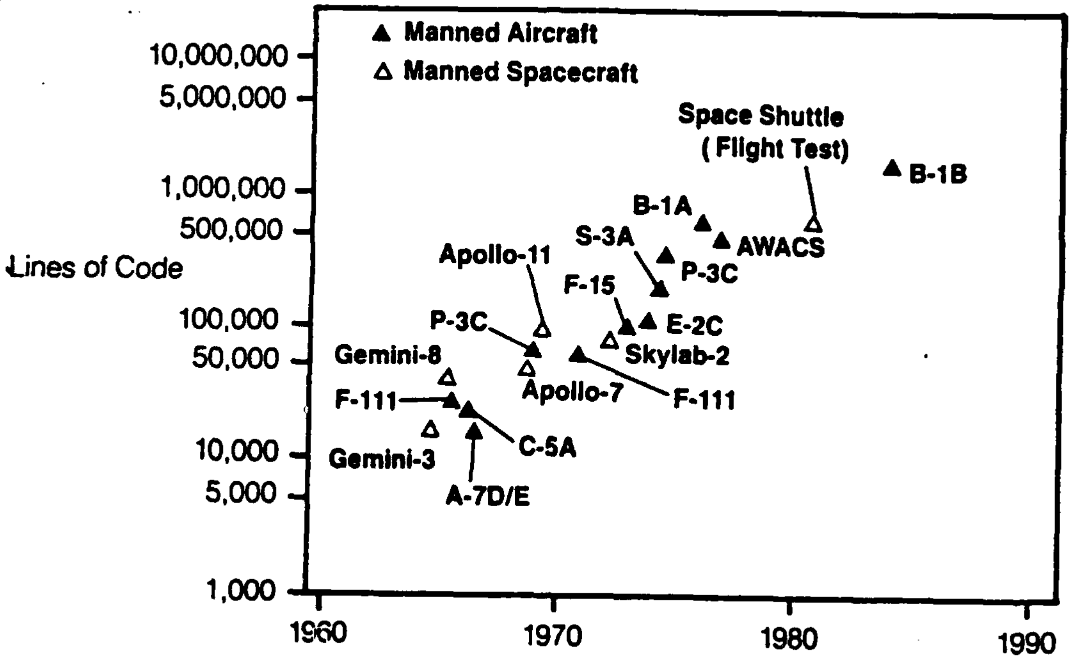

Figure 1.3. Measured Growth in Developed Software (Reprinted with permission from the AFA)

Force Operational Test and Evaluation Center (HQ AFOTEC) for each new weapon system fielded. This effort addresses system performance in an operational scenario (operational effectiveness) and system availability for operational use (operational suitability) [2:1]. HQ AFOTEC has tools in place for evaluating software operational suitability. Unfortunately, the current test and evaluation effort primarily addresses hardware OT&E or, at best, system OT&E, from an operational effectiveness standpoint.

1.1.2 Industry Perspective. Industry has also addressed the need for software reliability. In one of the first papers on this subject, Mr. Mulock of Lockheed Missiles & Space Company wrote:

> The Reliability Engineer should consider computer programming as another engineering discipline that is analyzable by the same techniques that he has used before ...the computer programmer is pushing the state of the art just as much as the transistor designer was in 1955 [59:497].

Many software reliability models were developed during the subsequent years, and recent efforts have defined the role of reliability engineering in the "typical software development team" [9:291]. Industry has applied several of the software reliability models to projects varying from remote terminal firmware (as discussed by Musa in [61]) to nuclear power plant software validation [70]. In contrast, there has been little use of software reliability measurement for military weapon systems [50].

As recent as the late 1980's software for major military command and control systems had proceeded past the software critical design reviews without any assessment of software reliability being performed [50:190]. In contrast, proposals for an integrated software reliability program were being presented as early as 1976, and more definitively in the context of reliability and maintainability in 1984 [9, 65]. The concept of software reliability has been in investigation for over 15 years, and several different organizations such as Hewlett-Packard Co., AT&T, and the Naval Surface Weapons Center have developed and used software reliability tools [29, 34, 61]. Goel states:

> Software reliability is a useful measure in planning and controlling resources during the development process so that high quality software can be developed [34:1412].

Therefore, the use of software reliability assessment could be one of the disciplined test methodologies needed to curb the rising cost of software.

1.2 Problem

While the tools and metrics are in place for software assessments of operational suitability (for example, AFOTECP 800-2 Vol 3, *Software Maintainability Evaluation Guide* [22]), there is no current way of measuring the operational effectiveness of software in order to perform adequate software OT&E. Reliability is considered a measure of operational suitability; however, software reliability is also one possible measure of the software's operational effectiveness, as software failures can reflect both the suitability question of "will it be available" as well as the effectiveness question of "does it work" [86:8-2],[2:8]. The purpose of this study is to determine if software reliability models can be applied specifically to the OT&E of Air Force weapon systems and, if this is the case, to propose a selected model implementation within a standardized methodology for use by HQ AFOTEC.

1.3 Definitions

Before defining software reliability, it is necessary to define both overall reliability and hardware reliability. ***Reliability*** is defined as

> *the duration or probability of failure-free performance under stated conditions* [20:8]

and also as

> *the ability of an item to perform a required function under stated conditions for a stated period of time* [5:29].

This definition is primarily based on the system's hardware attributes, which are discussed below in the definition of hardware reliability. A comparison to software reliability terms will then follow.

1.3.1 Hardware Reliability Terms. Essentially, ***hardware reliability*** is defined as [20]:

- Mean Time to Failure (MTTF). This is a basic measure of reliability for non-repairable items, and indicates the average amount of time until the failure of an item.
- Mean Time to Repair (MTTR). This is a basic measure of reliability that indicates the average amount of time necessary to repair an item once it has failed.
- Mean Time Between Failures (MTBF). This is a basic measure of reliability for repairable items and is defined as a combination of MTTF and MTTR [20:7]

$$MTBF = MTTF + MTTR \tag{1.1}$$

With hardware, these attributes can apply to different component, subsystem, and system levels, all of which can both relate to each other and have discrete values. For example, testing of components (such as integrated circuits (ICs)) can be performed to determine the MTTF, MTTR, and MTBF values for each IC. This value can then be included in the calculation of subassembly reliability, which can have cumulative failure rates expressed as a summation of the components

$$F_s = \sum_i f_i N_i \tag{1.2}$$

where F_s is the subassembly failure rate of a component i, given its failure rate f and a certain number of like components N [10:290]. This sort of analysis is also applicable to assemblies, subsystems, and finally the system as a whole.

1.3.2 Software Reliability Terms. Software does not permit the composition and decomposition analysis that is possible with hardware. Using a software instruction (i.e.—**x:=x+1**) as an analogy to the component, we find that an instruction functions perfectly 100% of the time without problems on its own; however, by combining instructions it is possible to develop subroutines that have failures (usually due to subroutine interactions) [10:291]. Therefore, software subroutine reliability can not necessarily be derived from the corresponding instruction reliabilities. Hardware assemblies are then created from subassemblies, just as software modules (discrete program units that are "a logically separable part of a program") are made up of subroutines (a routine, or "computer program segment that performs a specific task," that can be included in other routines) [5:24,30,34]. While hardware assembly reliability can be determined from subassemblies, a guarantee does not exist that software module reliability is based on the corresponding subroutine reliabilities. This correlation becomes even smaller as the software modules are linked together into subsystems, and finally systems. Thus, the standard terminology used for hardware reliability, including equations 1.1 and 1.2, is not applicable.

Instead, a different view of software must be taken. This is based on the ***design*** of the software, and not the physical implementation usually measured by hardware reliability [64:7]. Several studies have attempted to combine hardware and software theory into a system reliability perspective, citing software reliability models maturing to a point common with their hardware counterparts [31, 42]. By viewing software reliability as an integrated aspect of system reliability from a design viewpoint, it is possible to implement software reliability theory in a compatible way with hardware reliability theory [64:7]. Based on this, ***software reliability*** is defined as:

> *The probability that software will not cause the failure of a system for a specified time under specified conditions* [5:32],[40:14].
>
> *The probability that a software system will operate without a failure for a specified (mission) time* [19:9-1].
>
> *The probability of failure-free operation of a computer program for a specified time in a specified environment* [64:15].

A ***failure*** is defined (with respect to software) as:

An event in which a system or system component does not perform a required function within specified limits. A failure may be produced when a fault is encountered [5:19],[40:14]

where a *fault* is:

A manifestation of an error in software. A fault, if encountered may cause a failure. [5:19],[40:14].

Additionally a new concept, the *failure rate*, is defined as:

The ratio of the number of failures of a given category or severity to given period of time [5:19].

These definitions permit the use of reliability constructs similar to those used with hardware. One example of this is the ***failure intensity*** representation presented by Musa, et al. [64:11,528-529] which indicates the number of failures per unit time expressed by the function

$$\lambda(\tau) = \lambda_0 e^{-\frac{\lambda_0}{\nu_0}\tau} \tag{1.3}$$

where $\lambda(\tau)$ is the current *measured* software reliability based on time (τ), initial failure intensity (λ_0) and total estimated failures (ν_0). The value $\lambda(\tau)$ is the current failure intensity, and indicates the ratio of failures to operational time. A number such as this ratio could then be used to provide the operational assessment of computer software.

1.4 Scope

Currently, HQ AFOTEC/LG5 has a data collection tool for collecting software failure data to analyze and determine the software maturity. The data are identified by standard data item descriptions, and can be provided either as part of the initial contract or through a letter to the system program office (SPO) requesting the data to support software test (see Table 1.1) [23:4]. The existing software maturity databases are implemented on a format for office personal computers (PC's), and were enhanced to permit an operational effectiveness assessment of the software based on the candidate software reliability models. Compatibility with the maturity database and data collection tool required the software reliability model implementation to be in the same program development environment. This, in turn, allows for future software maturity databases to be used

Table 1.1. Software Maturity Data

Description	Variable Name	Format
Software Problem Number	PROB_NUM	Character 10
Software Configuration Item	CPCI	Character 10
Severity of Problem	SEV_CODE	Character 1
Date Problem Discovered	DATE	Date
Date Problem Fixed	DATE	Date
Description of Problem	TITLE	Character 42
Total Operating Time (minutes)	TOT_TIME	Character 10
Test Identification Number	TEST_ID	Character 10
Date Test Planned	TESTPLAN	Date
Date Test Completed	TESTCOMP	Date

as a baseline for software reliability assessment during the finite time OT&E period. The use of initial maturity data (collected prior to OT&E) to lay the foundation for software reliability assessment during OT&E is supported by Ferens, who states that software reliability models "are only useful after testing begins" [30:11-4].

A previous effort by Westgate [92] attempted to validate a *predictive* model of software reliability. While prediction has its necessary place in software evaluation, this study does not attempt to predict software reliability, nor attempt to identify what number correlates to reliable software. Instead, this study evaluates software reliability *measurement* models—models that indicate the current software reliability without making any determination of the overall quality of the software—mapped to finite OT&E time frames (i.e.—failures per flight hour). This study also attempts to determine the type of assessment such models could provide. The determination of "is the software reliable enough" and "how much more testing is needed" can then be decided by the decision makers based on a reporting of "where is the software right now."

1.5 Assumptions

This study presents no new models for evaluating software reliability. The existing models were assumed to be valid with respect to the entire life-cycle of a software development effort. The main focus is on the specific mapping to a finite OT&E time frame.

Several different categories of OT&E exist, and it is assumed that only the Initial OT&E (IOT&E) period will be used for time constraints [2]. As IOT&E supports procurement decisions, and Follow-on OT&E (FOT&E) begins *after* a weapon system enters production [2:1-2], the IOT&E timeframe is better suited to pre-production software assessment.

1.6 Approach

The first step was to conduct a literature review of the models available for software reliability evaluation. Identification and classification of these was performed based on their individual characteristics and focus. From these, models were selected for possible mapping into the IOT&E time frame based on the following criteria: predictive validity (of the model's parameters, *not* the reliability itself); capability; quality of assumptions; applicability to the finite-time environment; simplicity of design; diversity and applicability of output; and capability to use existing initial data [39],[64:384-387].

Implementation of two candidate models was attempted to further validate their usefulness for evaluating software operational effectiveness. The implementation was conducted in accordance with the software engineering discipline approach, and encompassed a relational database design to permit data persistence. The design environment for program development was the Clipper programming environment. The target system is any MS-DOS office computer environment to allow use by IOT&E test teams at various locations.

1.7 Thesis Organization

A review of available software reliability models is reported in Chapter 2. Chapter 3 describes the evaluation of the models and selection of the candidate model. Implementation of the candidate model is documented in Chapter 4, with the findings and results given in Chapter 5. Finally, conclusions and recommendations are presented in Chapter 6.

II. Literature Review

The software engineering discipline itself is concerned with "the systematic application of methods, tools and technical concepts to create complex, software-intensive systems that meet technical, economic and social objectives" [32:32]. One such technical concept is software quality, which is defined as "the totality of features and characteristics of a software product that bear on its ability to satisfy given needs" [5:32]. Indeed, software reliability has been identified as one of several software quality factors that affect the software life-cycle and its associated cost [30, 90, 91].

There are several suggested frameworks for identifying the software quality factors [52, 87, 91]. Tindell [87] investigated complexity for the maintenance of JOVIAL J73 software, and identified a software quality framework which included complexity and reliability (see Figure 2.1). Johnson also evaluated complexity and its metrics, in this case for use in the AFOTECP 800-2 Vol 3, *Software Maintainability Evaluation Guide* [44]. These works focused on the operational suitability assessments for software OT&E. In comparison, Westgate addressed the software quality of reliability in evaluating a software reliability model for software OT&E that uses calendar time as a basis [92]. To compliment these efforts, this thesis also explores software quality; however, the focus is on the operational effectiveness of the software based on reliability as derived from test, or execution, time. This chapter identifies endeavors in the literature to address software reliability.

2.1 Software Reliability Model Classifications

Many attempts have been made to define the concept of software reliability and determine some form of software reliability assessment model [10, 17, 28, 40, 43, 55, 60, 62, 65, 68, 69, 73, 74, 76, 86, 88, 96]. Such efforts have provided excellent insight into specific areas of software reliability evaluation. However, software reliability models currently available are not considered "universally appropriate" across all application domains and system usages, and Sommerville suggests that "it may be appropriate to use different reliability metrics for different parts of the system" [82:596].

Paralleling the efforts of model definition are consolidations of software reliability definitions and models into a single compendium or reference handbook [19, 27, 34, 41, 56, 57, 64, 80, 83, 85]. Such attempts take several software reliability models and group them by some classification, allowing the software engineer to select the appropriate method for a specific application. The following are major efforts to classify software reliability models.

2.1.1 IEEE Classification. The Institute of Electrical and Electronics Engineers (IEEE) classifies software reliability models in terms of product measures and process measures (see Table

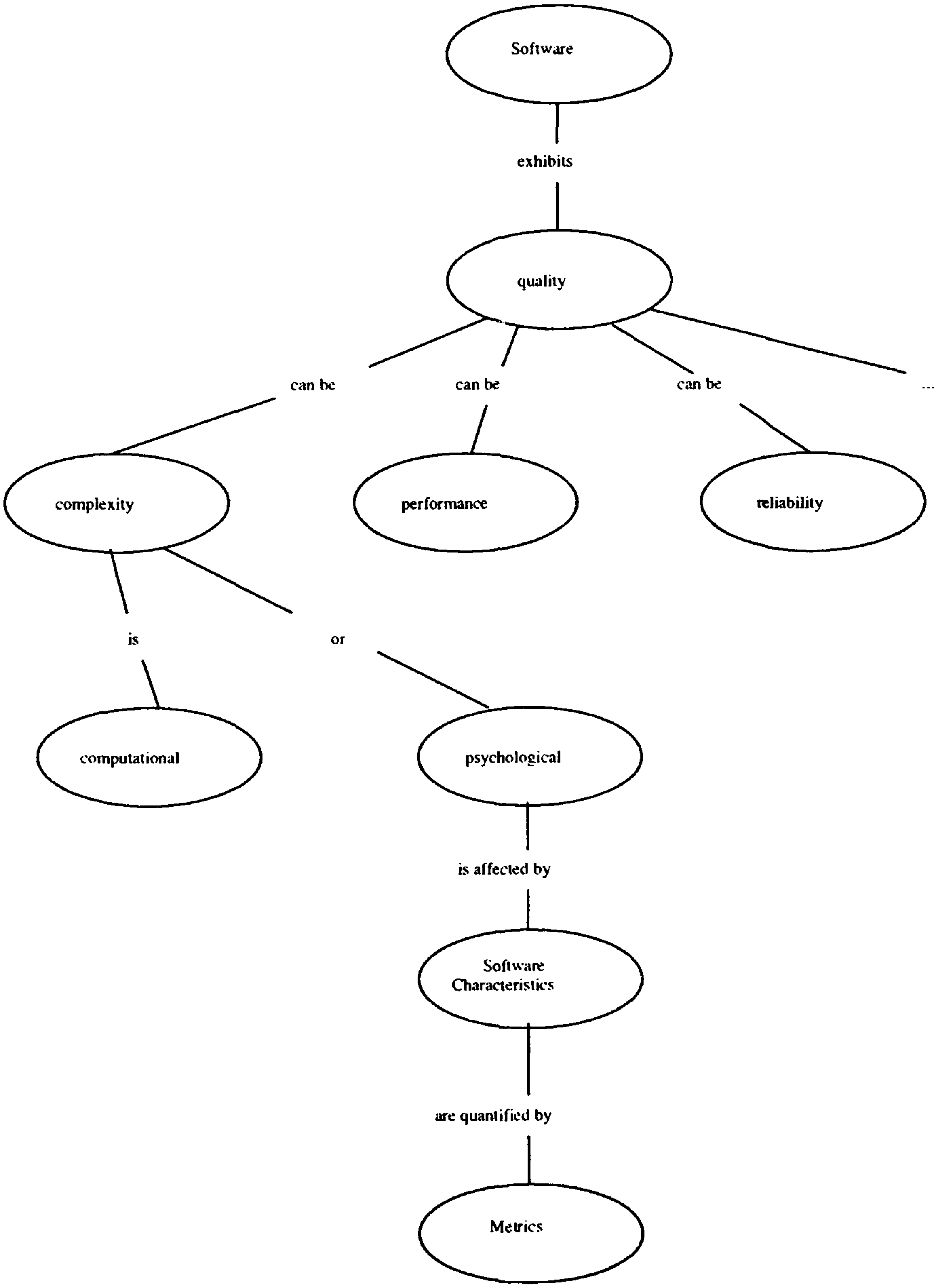

Figure 2.1. Software Quality Concept Map

2.1) [41:25-27]. Process measures provide input for the processes of both development and support management, and include: using management control measures for fault removal cost trends; coverage to ensure completeness of activities throughout the software life-cycle; and technical and cost evaluations for software delivery decisions [41:25]. Indicators, such as testing sufficiency, are similar to those in the Air Force Systems Command Pamphlet (AFSCP) 800-14, *Software Quality Indicators* [25]. A study by Lipow [53] identifies one approach that uses a form of residual fault count and error distribution measures [41].

Product measures, on the other hand, focus on the developed software objects and encompass many different metrics such as fault density, failure rate, and mean-time-to-failure [41:26-27]. These measures are applicable to both software reliability prediction and measurement models. With respect to software reliability *prediction* models, Wilson and Shen state:

> No growth model has demonstrated that it can be used with a high degree of confidence to predict operational reliability from data generated in the debugging phase in a general setting [93:5].

In contrast, the focus of OT&E is to field test and evaluate weapon systems to determine effectiveness and suitability [2]. AFR 800-18, *Air Force Reliability and Maintainability Policy*, requires implementing "reliability qualification and acceptance testing, ...[which] will be tailored for effectiveness and efficiency in terms of the management information they provide" [24:3]. Most software reliability models require data for calibration; however, it is not possible to directly measure software reliability during the design and coding stages where such calibration data does not exist [30, 63]. Therefore, specific product measures for OT&E assessment of software reliability should focus on *measurement* of: errors, faults and failures; mean-time-to-failure and failure rates; and remaining product faults [41:26].

2.1.2 NSWC Classification. The Naval Surface Weapons Center (NSWC) Technical Report (TR-82-171) classifies software reliability models into three categories: error seeding/tagging models; data domain approach models; and time domain approach models [27]. Error seeding/tagging models are "built on firm statistical ground" [65:336]. The original work by Mills (as described by Myers in [65]) developed a software reliability model requiring software engineering personnel to place, or "seed" errors intentionally in the computer software. The errors are seeded randomly, with the assumption that an equal probability exists of finding either seeded or original errors during testing. Since the number of seeded errors is known *a priori*, the ratio of the number of found seeded errors divided by total seeded errors would be equal to the ratio of the number of found original errors divided by total original errors [65].

Table 2.1. IEEE Software Reliability Model Classification

Product Measures	Process Measures
Errors, Faults, Failures	Management Control
Mean-Time-To-Failure, Failure Rate	Coverage
Reliability Growth and Projection	
Remaining Product Faults	Risk, Benefit, Cost Evaluation
Completeness and Consistency	
Complexity	

Data domain approach models are similar to error seeding/tagging models in that they estimate a program's current reliability from a ratio. In this case, the ratio is the number of successful test runs completed divided by total number of test runs attempted [27:3-1]. This ratio assumes that there is an equal probability of either failure or success for each test run [19:9-21]. The test inputs are chosen based on probability distributions estimated for operational use, and the success of a test run is defined with respect to these inputs [27:3-1].

Time domain approach models model the error generation process based on errors and time. The relationship between the two is based on either error occurrence times and the calculated times between error occurrences, or the number of errors that occur during a specified time period [27:4-1]. Several of these models are similar to hardware reliability models, and use major assumptions concerning the probability distribution of software failures [65:330].

Within the time domain approach models, there is a further distinction based on the specific mathematical method used. The NSWC report identifies three subcategories of time domain approach models: classical software models; Bayesian models; and Markov models [27]. Both the classical software models and Bayesian models treat software reliability as a function of continuous events. The classical software models use probabilities derived from software failure frequency analysis and software hazard (or failure) rates, the Bayesian models use a more subjective viewpoint in counting errors [38],[27:100-101].

In contrast to these, the Markov models view software reliability as a series of discrete events [27:116]. Markov models treat each software failure event as a separate occurrence, such that an event at time $t+1$ is not based on the reliability history previous to time t [38:545].

2.1.3 RADC Classification. Rome Air Development Center (RADC) drew upon an earlier work of Goel to identify four classes of software reliability models: fault seeding models; input domain models; times between failure models; and failure count models [56:3-35]. The categories of fault seeding models and input domain models are identical to the NSWC categories of error seeding/tagging models and data domain approach models, respectively [27, 56].

The times between failure models category is similar to the Bayesian subclass of the NSWC time domain approach models, while the failure count models are identical to the classical subclass of NSWC time domain approach models [27, 56]. Goel uses this same classification again in a later article addressing the assumptions, limitations, and applications of various software reliability models [34].

2.1.4 MIL-HDBK-338-1A Classification. MIL-HDBK-338-1A defines a higher level of abstraction, classifying software reliability models into two general categories: non-failure rate based models and failure rate based models [19].

2.1.4.1 Non-Failure Rate Based Models. The term non-failure rate implies that the software reliability model is independent of the software's failure rate [19, 65]. The two basic types of non-failure rate based models are combinatorial and input domain [19]. Combinatorial models derive their name from the mathematical formula of ratios of identified faults to expected faults [19]. The combinatorial models include both error seeding and binomial models [19].

While the input domain category is identical to the RADC input domain category and the error seeding combinatorial model category is identical to the RADC fault seeding category, there is no corresponding category for the binomial models [19, 56]. The binomial models use combinatoric mathematics to calculate reliability probability from the number of errors encountered, the number of attempted program test runs, and the probability of finding errors on any given program test run [19:9-21]. While such a method is appealing based on its simplicity, it is more a predictor than a measure of software reliability, and will not be considered further.

2.1.4.2 Failure Rate Based Models. In contrast, failure rate based models are concerned with the number of software failures and the frequency at which they are experienced during a period of time [65:330-331]. MIL-HDBK-338-1A identifies two categories of failure rate based models: classical, and Bayesian [19]. These categories map directly to the subclasses of classical and Bayesian of the NSWC time domain approach models, as well as the RADC categories of failure count models and times between failure models, respectively [19, 27, 56].

2.1.5 Musa and Okumoto Classification. In the book *Software Reliability: Measurement, Prediction, Application*, Musa et. al. give a different classification scheme first presented by Musa and Okumoto in 1983. Model classification is based on five attributes: time domain (calendar time or execution time); category (either a finite or infinite number of failures experienced in infinite time); the distribution type; class (only if the model is in the finite failure category); and family (only if the model is in the infinite failure category) [64:250-251]. The table from Musa et al. is shown in Figure 2.2.

Musa et al. discuss models with respect to both time domains; however, execution time better "characterizes the failure-inducing stress placed on software" [64:31]. Therefore, only the execution time based models will be discussed. Within the Musa and Okumoto classification, a model is first identified as either a finite or infinite failure model depending on whether the model assumes a finite or infinite number of failures will be reached at time $t = \infty$ [62:235]. Next, the failure quantity distribution for failure experienced at time t is identified [62:235],[64:250]. Three distributions have been identified for the finite failure category, while four have been identified for the infinite failure category [62:235],[64:250,251]. Against these, the failure intensity form is cross-referenced, using time as a basis for the class (finite failure category) and expected number of failures as a basis for the family (infinite failure category) [62:235],[64:250]. This type of analysis identifies the relationships between models within both of the times between failure and failure count categories [56, 64].

2.1.6 Overall Model Classification Schema. A comparison of the MIL-HDBK-338A, NSWC, and RADC software reliability model classifications is shown in Table 2.2. From this, and Figure 2.1, the concept map in Figure 2.3 was derived. This software reliability concept map reflects the overall classification as identified in the previous sections. The initial division of software reliability into process measures and product measures is based on the IEEE classification. While the process measures are very important to the management of the overall software life-cycle, the OT&E effort requires an approach that evaluates the software, and not the management process [2]. The additional level of abstraction defined by MIL-HDBK-338-1A (identifying failure and non-failure rate) would be placed between the IEEE product measures and the lower categories, and is omitted for clarity. Subsequent divisions of the product measurement into software reliability models are based on the categories derived from the NSWC, RADC, and Goel classifications, and are identified as the model categories of fault seeding, input domain, times between failures, and failure count. While the Musa et al. software reliability model classification differs from this more traditional hierarchy, it does prove useful for relating models to each other within appropriate classifications. This relationship is important for identifying initial models for evaluation.

TABLE 9.2
Software reliability model classification scheme

Finite failures category models

Class[2]	Type[1] Poisson	Binomial	Other types
Exponential	Musa (1975) Moranda (1975) Schneidewind (1975) Goel-Okumoto (1979b)	Jelinski-Moranda (1972) Shooman (1972)	Goel-Okumoto (1978) Musa (1979a) Keiller-Littlewood (1983)
Weibull		Schick-Wolverton (1973) Wagoner (1973)	
CI		Schick-Wolverton (1978)	
Pareto		Littlewood (1981)	
Gamma	Yamada-Ohba-Osaki (1983)		

Infinite failures category models

Family[3]	Type[1] T1	T2	T3	Poisson
Geometric	Moranda (1975)			Musa-Okumoto (1984b)
Inverse linear		Littlewood-Verrall (1973)		
Inverse polynomial (2nd degree)			Littlewood-Verrall (1973)	
Power				Crow (1974)

[1]Type: Distribution of number of failures experienced.
[2]Class: Functional form of failure intensity (in terms of time).
[3]Family: Functional form of failure intensity (in terms of expected number of failures).

Figure 2.2. Musa et al.'s Software Reliability Model Classification (Reprinted with permission from McGraw-Hill Book Company)

Table 2.2. Comparison of Software Reliability Model Classifications

RADC	NSWC	MIL-HDBK-338A
Fault Seeding	Error Seeding/Tagging	Non-Failure Rate Combinatorial Error Seeding
Input Domain	Data Domain Approach	Non-Failure Rate Input Domain
Times Between Failure	Time Domain Approach Bayesian Subclass	Failure Rate Based Bayesian
Failure Count	Time Domain Approach Classical Subclass	Failure Rate Based Classical

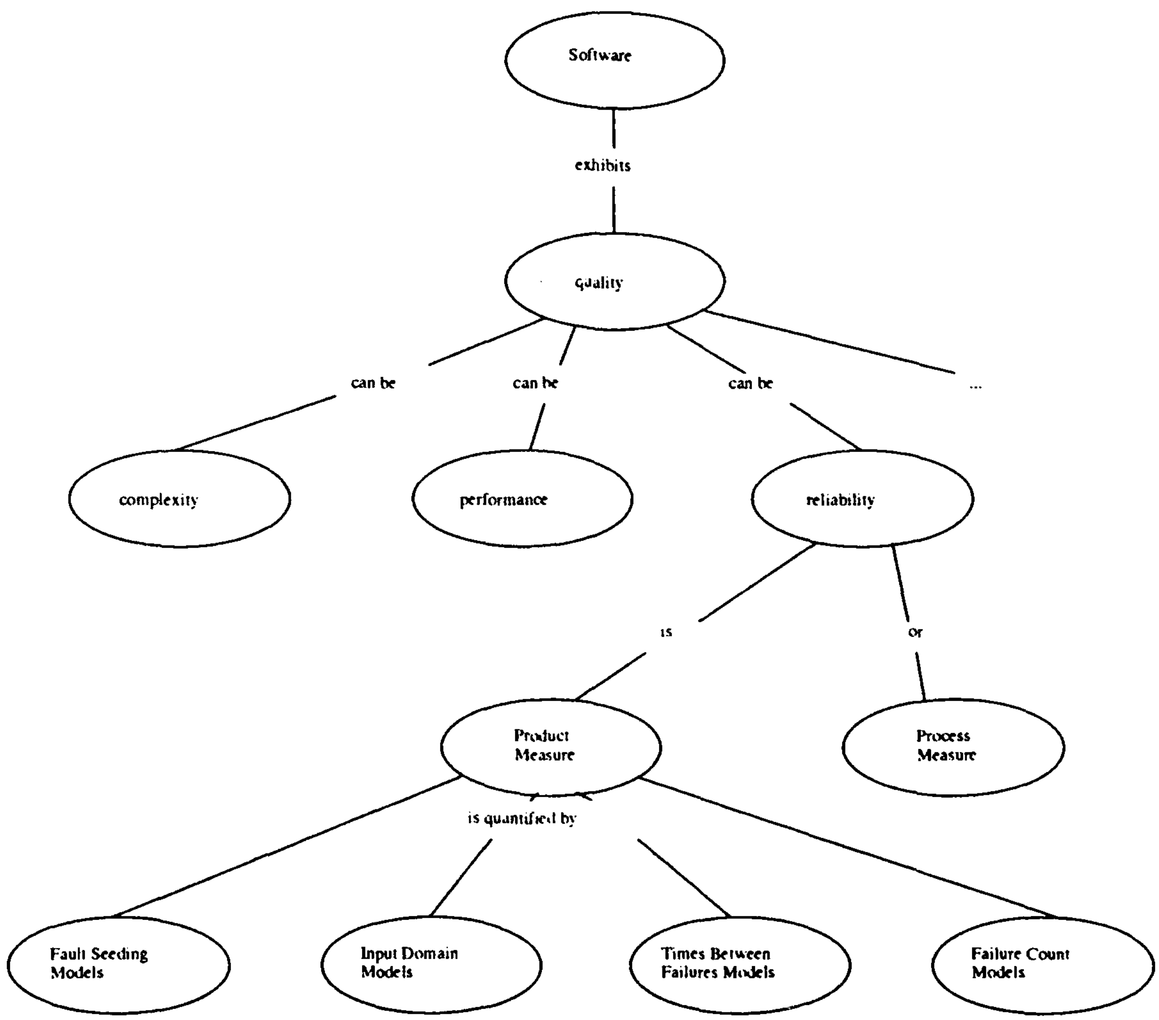

Figure 2.3. Software Reliability Concept Map

2.2 *Software Reliability Model Descriptions*

The following major models were identified for evaluation from the four model categories:

- Fault Seeding: Mill's Hypergeometric model [65]
- Input Domain: Ramamoorthy-Bastani model [69]
- Times Between Failures: Jelinski-Moranda model [43], Littlewood-Verrall model [55], Schick-Wolverton model [73],
- Failure Count: Goel-Okumoto Nonhomogeneous Poisson Process model [35], Musa Execution Time model [60], Musa-Okumoto Logarithmic Poisson Execution Time model [62], Shooman Exponential model [76], Yamada-Ohba-Osaki Power model [96]

Other software reliability models (especially times between failures and failure count models) are similar to these, being either more generalized or refined for specific applications [27, 34, 56, 64, 88]. The focus on evaluating older models is permissible, as there have been no significantly new software reliability models developed in the last eight years [79]. Each model's assumptions follow its description. Goel states the following concerning software reliability model assumptions

> . . . as a totality, they provide an insight into the kind of limitations imposed by them on the use of the software reliability models . . . The ultimate decision about the appropriateness of the underlying assumptions and the applicability of the models will have to be made by the user of a model [34:1417].

Therefore, the assumptions will be identified in this chapter, and their applicability will be assessed in the following chapter.

2.2.1 Fault Seeding Models. Goel identifies the two major assumptions necessary to use fault seeding models [34:1419]:

- Faults are seeded randomly throughout the program.
- Innate faults have the same probability as the seeded faults of being discovered during test.

A discussion with respect to the major model follows.

Mill's Hypergeometric Model.

Equation. The general equation for this model is given in Myers [65]

$$N = sn/v \tag{2.1}$$

where N is the maximum likelihood estimate of total number of innate errors, n is the number of detected innate errors, s is the total number of seeded errors, and v is the number of detected seeded errors [65:336-337]. The confidence calculation C is also given in Myers [65]

$$C = \begin{cases} 1 & \text{if } n > k \\ \frac{s}{s+k+1} & \text{if } n \leq k \end{cases}$$

where k is an upper bound assumption of the number of innate errors in the program [65:337].

Assumptions. Although the error detection probabilities are unknown, the Mill's model assumes both the innate and seeded errors have the same detection probability [65:337]. Random error seeding throughout the program is another important assumption; however, seeding errors that have the same probability of detection as innate errors is a major problem [6:12],[34:1419].

2.2.2 Input Domain Models. The major assumptions necessary for input domain models are summarized by Goel as [34:1419]:

- Testing performed is random.
- The distribution is known *a priori* of the input profile for test.
- Input domain equivalence classes can be determined.

A discussion with respect to the major model follows.

Ramamoorthy-Bastani Model.

Equation. The Ramamoorthy-Bastani model is defined as [69]

$$P\{E_i \mid n\} = e^{-\lambda V} \prod_{j=1}^{n-1} \left[\frac{2}{1 + e^{-\lambda x_j}} \right] \tag{2.2}$$

based on a program's continuous equivalence class specified by $E_i = [a, a+v]$, with n test cases each having successive distances x_j for $j = 1, \cdots, n-1$ [69:366]. Here, λ is the inverse of the mean length of intervals for E_i, and V is a determination of the number of errors [69:366]. The product λV is related to both the number N of elements in and degree D of an equivalence class [69]

$$\lambda V \approx \frac{D-1}{N}$$

Assumptions. The Ramamoorthy-Bastani model assumes the input can be divided into equivalence classes, and then requires an assumption of the equivalence class distribution; however, the determination of the equivalence classes is very costly [34:1419],[69:367]. It also allows the use of any test case selection strategy, and does not assume random sampling for test inputs [69:367].

2.2.3 Times Between Failures Models. Goel discusses several assumptions common to the times between failures models [34:1417-1419]:

- Faults are independent and have the same probability of exposure.
- Perfect debugging is done immediately after the occurrence of a fault.
- Successive times between failure occurrences are independent of each other.
- The software system failure rate decreases as testing proceeds.

A discussion with respect to the major models follows.

Jelinski-Moranda Model.

Equation. The Jelinski-Moranda model defines the probability of a time interval x_i between the $i-1$ and ith consecutive errors as [43]

$$P(x_i) = \phi[N-(i-1)]e^{-\phi[N-(i-1)]x_i} \tag{2.3}$$

where N is the initial error content and ϕ is a proportionality constant [43:473]. The hazard function $z(t_i)$ is defined by the software failure rate $\phi[N-(i-1)]$ [43:473]. Musa et al. takes this a step further, and derives the failure intensity function with respect to time ($\lambda(t)$) based on the constant hazard rate ϕ [64]

$$\lambda(t) = N\phi e^{-\phi t}$$

Assumptions. A major assumption of this and other times between failure models is based on perfect debugging, the act of fault correction without introducing new faults [34:1418]. Another assumption shared by models in this category is the independence of successive failure times from each other [34:1417]. The model also assumes the failure rate between errors is uniform [43:473]. This notion of a constant arrival rate for errors has been cited as a drawback [73:105]. Also, testing time periods which are of equal length are assumed to represent the same thoroughness of testing [43:477]. Musa et al. categorize the Jelinski-Moranda model as a finite failure exponential class model, which assumes that at infinite time the number of failures experienced is finite [64:278-280].

Littlewood-Verrall Model.

Equation. The equation for the Littlewood-Verrall model is [55]

$$g(l \mid i, \alpha) = \begin{cases} \frac{\psi(i)[\psi(i)l]^{\alpha-1} e^{-\psi(i)l}}{\Gamma(\alpha)} & l > 0, \psi > 0, \alpha > 0 \\ 0 & l \leq 0 \end{cases} \quad (2.4)$$

where the hazard rate l is expressed as the probability density function $g(l \mid i, \alpha)$, $\psi(i)$ is the growth function for the gamma distribution, and α is the shape parameter for the gamma distribution [55:110]. The probability density function for time of next failure t_i after repair of the previous failure given the failure rate λ is [55:110]

$$f(t \mid \lambda) = \begin{cases} \lambda e^{-\lambda t} & t \geq 0, \lambda > 0 \\ 0 & t < 0 \end{cases}$$

Musa et al. define the failure intensity function with respect to time ($\lambda(t)$) based on these probability density functions as [64]

$$\lambda(t) = \frac{1}{\sqrt{\beta_0^2 + 2\beta_1 t}}$$

with β_0 and β_1 being model parameters of the reliability growth function ψ [64:294-296].

Assumptions. A major assumption of this and other times between failure models is based on perfect debugging, where fault correction occurs before finding the next fault without introducing new faults [55:109]. The independence and randomness of successive failure times are other assumptions shared by models in this category [55:109]. As it takes a

Bayesian approach, this model assumes "subjective attitudes to the system under consideration, thus 'probability' means 'personal probability' or 'degree of belief' " [55:110]. Musa et al. classify the Littlewood-Verrall model as a member of both infinite failure inverse linear and inverse polynomial families, which assumes that at infinite time the number of failures experienced is infinite [64:293-296].

Schick-Wolverton Model.

Equation. The original Schick-Wolverton model (as described by Schick and Wolverton in [73]) is given by:

$$R(t_i) = e^{-\phi[N-(i-1)]\frac{t_i^2}{2}} \tag{2.5}$$

with the hazard rate $z(t_i) = \phi[N - (i - 1)]t_i$ [73:105,112]. This hazard rate is similar to that of equation 2.3. A modified version was subsequently proposed with a hazard rate of $z(t_i) = \phi[N - (i - 1)][-at_i^2 + bt_i + c]$ [73:112].

Assumptions. The error rate is not constant, and errors are corrected as soon as they are detected–"As errors occur, the routines are stopped, the error is identified, corrected, and the error modality is reduced" [73:111]. Musa et al. classify the Schick-Wolverton models as finite failure Weibull and modified Weibull class models, which assumes that at infinite time the number of failures experienced is finite [64:281-283]. Musa et al. state that for the modified model, "It does not appear to have practical applicability," and also that "it is more complex than the other models" with "no evidence of superior properties that would justify the complexity" [64:283].

2.2.4 Failure Count Models. In contrast to the times between failures models, the failure count model assumptions are based on test interval and not failure interval times [34:1418-1419]:

- The number of failures discovered during a test interval is independent of the number discovered during a different nonoverlapping test interval.
- Testing is similar and uniform throughout the different test intervals.
- Each test interval is independent of the others.
- The software system failure rate decreases as testing proceeds.

A discussion with respect to the major models follows.

Goel-Okumoto Nonhomogeneous Poisson Process Model.

Equation. The general equation for the Goel-Okumoto Nonhomogeneous Poisson Process (NHPP) model is [35]

$$P\{N(t) = y\} = \frac{(m(t))^y}{y!}e^{-m(t)} \quad y = 0, 1, 2, \ldots \tag{2.6}$$

with $m(t) = a(1 - e^{-bt})$ and $\lambda(t) \equiv m\prime(t) = abe^{-bt}$ where the cumulative number of failures at time t is denoted by $N(t)$, $m(t)$ represents the expected number of failures at time t, the failure rate is $\lambda(t)$, a is the eventual expected number of failures, and b is the fault detection rate per fault [34].

Assumptions. The number of failures is 0 at time $t = 0$, and the number of failures occurring during nonoverlapping time intervals are mutually exclusive [35:206]. Also, the number of remaining faults to be discovered is considered a variable of test and environmental factors instead of a fixed constant [34:1415]. This is considered a finite failure exponential class model [64].

Musa Execution Time Model.

Equation. Musa's Execution Time model has a hazard rate of [60:314]

$$z(\tau) = fKN_0 - fKn \tag{2.7}$$

where τ is the execution time, f is linear execution frequency (instruction execution rate per number of program instructions), K is the fault exposure ratio (as the machine state may vary, this accounts for the probability of a fault being exposed when the related instruction is being executed), N_0 is the number of inherent errors in the program, and n is the number of faults corrected during time τ [60]. This concept has also been applied to the determination of failures experienced (μ) for a given execution time (τ) [64:37]

$$\mu(\tau) = \nu_0 \left[1 - \exp\left(-\frac{\lambda_0}{\nu_0}\tau\right)\right] \tag{2.8}$$

as well as the measurement of current failure intensity (λ) based on either execution time (τ) as shown in Equation 1.3 [64:39]

$$\lambda(\tau) = \lambda_0 \exp\left(-\frac{\lambda_0}{\nu_0}\tau\right) \tag{2.9}$$

or actual failures experienced (μ) [64:33]

$$\lambda(\mu) = \lambda_0 \left(1 - \frac{\mu}{\nu_0}\right) \tag{2.10}$$

Here, ν_0 is the total expected number of failures, and λ_0 is the initial failure intensity (failures per unit time) [64:528-530].

Assumptions. The basic execution time model has been around for quite some time, and is actually considered a Poisson process model [34, 60, 64]. This model assumes that: program faults are independent; the "potential test space 'covers' its use space," not in a completeness sense but rather the test sets should be representative of operational program use; test inputs are randomly selected; all failures are observed; and discovered faults are corrected before continuing with testing or are not counted again if rediscovered [60:313]. This model is considered a finite failure exponential class model [64].

Musa-Okumoto Logarithmic Poisson Execution Time Model.

Equation. The Musa-Okumoto Logarithmic Poisson Execution Time model is expressed by [62:231]

$$\mu(\tau) = \frac{1}{\theta} \cdot \ln(\lambda_0 \theta \tau + 1) \tag{2.11}$$

Here, λ_0 is the initial failure intensity, and θ is the failure intensity decay parameter, identifying how fast the failure intensity is changing [62]. Again, μ is the number of failures expected for a given execution time τ [64:530-531]. As with the Musa Execution Time model, measurement of current failure intensity (λ) can be made from either execution time (τ) [64:39]

$$\lambda(\tau) = \frac{\lambda_0}{\lambda_0 \theta \tau + 1} \tag{2.12}$$

or actual failures experienced (μ) [64:34]

$$\lambda(\mu) = \lambda_0 \exp(-\theta \mu) \tag{2.13}$$

Assumptions. This model uses the same assumption as the Goel-Okumoto NHPP model in Equation 2.6 with respect to time $\tau = 0$; however, the Musa-Okumoto Logarithmic Poisson Execution Time model also assumes an exponentially decreasing failure intensity based on the number of failures experienced [62:230]. The model also uses τ to determine the function of the mean value of experienced failures with respect to time [62:231]. This is considered a geometric family model [64].

Shooman Exponential Model.

Equation. The Shooman Exponential model is given as [76]

$$\rho(\tau) = \frac{k_1 E_T}{I_T} e^{-k_1 \tau} \tag{2.14}$$

where $\rho(\tau)$ is the number of errors per total number of instructions detected per month, τ is the number of months after start of system test, k_1 is the proportionality constant, E_T is the total number of errors (a constant), and I_T is the number of program instructions [76].

Assumptions. The Shooman model uses the history of other similar software programs as a basis for determining the model constants [76:486]. This model assumes "the total number of errors in the program is fixed" and the number of errors remaining is the difference between total errors and errors encountered [76:487]. It also assumes "all detected errors are corrected errors," while also taking into account that "in any sizable program it is impossible to remove all errors" [76:488]. Another assumption is both the number of debugged errors and number of errors present should decrease as testing proceeds [76:492]. This, taken with the initial assumption that errors detected are proportional to the number present, results in an exponential error detection rate [76:492]. Musa et al. categorize the Shooman model as a finite failure exponential class model [64].

Yamada-Ohba-Osaki Power Model.

Equation. The Yamada-Ohba-Osaki Power model (also referred to as the S-Shaped model) is a NHPP model with the following mean value function for time t [96:476]

$$M(t) = a[1 - (1 + bt)e^{-bt}] \quad a, b > 0 \tag{2.15}$$

where a is the total number of errors and b is the error detection rate [96:475]. Additionally, the failure intensity is given by [96:476]

$$\lambda(t) = ab^2 t e^{-bt}$$
$$\lambda(0) = 0$$
$$\lambda(\infty) = 0$$

with the remaining expected number of errors determined by [96:476]

$$n(t) = a(1 + bt)e^{-bt}$$

Assumptions. This model assumes a steady-state for the error detection rate b [96:475]. Other assumptions include random occurrence of failures, the time to failure $(k-1)$ impacts the time to failure k from failure $(k-1)$, prompt correction of error(s) each time a failure occurs, and perfect debugging [96:475-476]. This model is considered a gamma class Poisson finite failure model [64].

2.3 Summary

This chapter started with the identification of software quality as a desirable result of software engineering. Software reliability was then described as one of several software quality factors that affects software life-cycle cost. Next, we proceeded to identify software reliability model classifications within the scope of software reliability measurement. As many papers on software reliability exist, it was necessary to define the overall framework for software reliability model evaluation before choosing specific models. We compared and contrasted different categories of software reliability models. The baseline framework was derived from a synthesis of categories, primarily following the RADC and Goel categories. Within each of the framework major categories, specific software reliability models were then identified for evaluation. The evaluation of these major models is described in Chapter 3.

III. Software Reliability Model Selection

This chapter identifies the selection of the candidate software reliability models. It begins with identification and discussion of the software reliability model selection criteria. The criteria are then applied to select candidate software reliability models for evaluation against software maturity data.

3.1 Model Selection Criteria and Discussion

The goal of this thesis is to identify one model and methodology that is appropriate for use in the IOT&E phase. Toward this end, the criteria defined in [39] and [64], as well as other implementation specific criteria defined in [16] will be used; however, an initial screening based on model requirements eliminates the two categories of fault seeding and input domain. Mill's Hypergeometric model requires fault seeding of intentional changes to the software. Such seeding is very difficult and could be disastrous for something complex like avionics flight software. As such intentional errors are not something to be introduced after the start of IOT&E, this model will not be considered. Similarly, the Ramamoorthy-Bastani model will not be considered. The IOT&E input domain for testing is based on operational usage, which is supported by the model's lack of random sampling assumption; however, the cost of determining equivalence classes for an integrated weapon system (such as a missile or aircraft) would be prohibitive. This leaves only the failure count and times-between-failure models. These models are discussed below with respect to the criteria of predictive validity, capability, quality of assumptions, applicability to the finite-time environment, simplicity of design, diversity and applicability of output, and capability to use existing data.

3.1.1 Predictive Validity. This criterion concerns the accuracy of a model's parameter estimation, and not the prediction of the reliability itself [64]. As such, *predictive validity* is

> *the capability of the model to predict future failure behavior during either the test or the operational phases from present and past failure behavior in the respective phase* [39].

With respect to a "weighted parameter estimation" of number of errors, both the Littlewood-Verrall model of the inverse polynomial family and the Musa-Okumoto Logarithmic Poisson Execution Time model were more accurate in the first 60% of testing than the Musa Execution Time model, the Yamada-Ohba-Osaki Power model, or the Crow model (described as a power family

Poisson model in [64]) [89:9]. After this initial phase, all of these models performed satisfactorily [89:9]. Of the models analyzed by Musa et al. in [64], the geometric and inverse polynomial families had the best initial predictive validity. This assessment was made against the different classes and families (the type, binomial or Poisson, made no difference), and was based on both maximum likelihood estimation (MLE) and least squares estimation (LSE) [64:390]. Musa et al. determined the Musa-Okumoto Logarithmic Poisson Execution Time model as being superior; however, the Musa Execution Time model becomes just as viable after the initial 60% of testing [64:398]. The applicability of an exponential class model is important, as software maturity data, which this thesis suggests could be the basis for parameter estimation, has historically been exponential [94].

In another study involving 16 data sets on various hardware platforms, Angus et al. found it difficult to estimate parameters for the Jelinski-Moranda and Schick-Wolverton models [4:195]. While the Jelinski-Moranda and Schick-Wolverton models are considered finite failure models, both geometric and inverse polynomial families are in the infinite failure category [64:251]. Thus, it appears that it is easier and more accurate to estimate parameters for models of the infinite failure category, as opposed to the finite failure category. For IOT&E, such parameter estimation could be based heavily on data previously collected prior to the start of IOT&E (either on the system undergoing test or from another similar system that has completed test). Initial parameters could then be predicted using a geometric or inverse polynomial model that is Poisson in type.

3.1.2 Capability. Another criterion, *capability*, is defined by Iannino et al. as

> *...the ability of the model to estimate with satisfactory accuracy quantities needed by software managers, engineers, and users in planning and managing software development projects or running operational software systems* [39].

Such accuracy of estimate could then be measured in the following quantities [64]:

- Present reliability, MTTF, and failure intensity.
- Expected date to reach specified reliability, MTTF, or failure intensity objective.
- Human and computer resource and cost requirements needed to reach the failure intensity objective.

This criterion is important for IOT&E, as the test director needs to know both the *current* quality of the software and what it will cost (in time and money) to reach an acceptable level of quality. Musa et al. conducted an evaluation and comparison of 18 major software reliability models [64]. Of the 18 models examined, those of the exponential class and geometric family appear to have the best capability to be used to make quality assessments of the software under test [64].

3.1.3 Quality of Assumptions. Iannino et al. recommend that assumptions should be tested; however, if this is not possible, the assumption's "plausibility" should be considered based on logical consistency and the user's software engineering experience [39]. For complex systems, it is difficult to test the validity of software reliability model assumptions. An example of this was the Hughes Joint Surveillance System (JSS) air defense system for North America, where it was not possible to confirm the validity or lack thereof of all software reliability model assumptions used in evaluating the software [3:268,270]. As IOT&E is performed on weapon systems of similar complexity to the JSS, there will be no attempt to prove or disprove all the assumptions for the models under consideration. Instead, a comparison of only the assumptions deemed necessary for IOT&E assessment will be performed against the models' assumptions. A model fails this comparison if ***only one*** major IOT&E assumption is not supported by the model's assumptions.

Both Musa et al. [64] and Goel [34] identify many critical assumptions that are necessary for model implementation. For application to the IOT&E environment, the major assumptions were derived from both HQ AFOTEC requirements and the author's experience in IOT&E of weapon system software and include [47]:

1. Operational testing is representative of the operational environment.
2. There is imperfect debugging for fault removal.
3. Errors might not be corrected after the test interval (i.e.–just after a test flight).
4. Execution time is used for the failure rates.

Assumption 1 allows both times between failures and failure count models to assess the software with respect to operational reliability [34:1418]. The assumption is based on the operational profiles used to assess the overall performance of system testing [2:1]. System testing is the usual level of test for a Test and Evaluation effort; however, there is usually insufficient test time to thoroughly test all the software due to the tremendous combinatorics that occur from integrating even the simplest subsystems together [48:110,114]. As a consequence, using operational profiles for testing differs in the degree of randomness (and thus thoroughness) that is possible with module or unit level testing. Since the test cases are then not likely to be independent, the test process will not follow a true random nature [34:1417]. This eliminates times between failure models, which assume times between failures occur independently [34:1417,1419]. In addition, this assumption makes an important contribution to determination of end of operational testing and start of operations. Since IOT&E testing is targeted for an operational environment, a final IOT&E value of a failure intensity would then be the constant failure intensity expected to occur throughout operations until the next major software release.

Assumption 2 also eliminates the Shooman model, and most, if not all, of the times between failures category models [27:4-7],[34:1418-1419]. Ohba and Chou have assessed the validity of the perfect debugging assumption found for the times-between-failures models, noting that "software reliability growth models sometimes give reasonable figures (fairly accurate estimations) in conditions where the perfect debugging assumption is not valid" [67:41]. They have also proposed modifications to the Jelinski-Moranda and Goel-Okumoto models to accommodate imperfect debugging; however, they cite that further study using actual project data is necessary to verify the modified models' applicability [67:45]. Goel and Okumoto have also proposed a modified model, the Goel-Okumoto Imperfect Debugging model, which is an extension of the Jelinski-Moranda model based on a Markov process [34:1414]; however, this model is eliminated from consideration by Assumptions 1 and 3. Ohba and Chou also note the necessity of verifying the impact of an imperfect debugging assumption on S-shaped software reliability models (such as the Yamada-Ohba-Osaki Power model discussed in Chapter 2) before concluding that the imperfect debugging assumption does not affect software reliability data analysis [67:46]. Until such proof exists, the Yamada-Ohba-Osaki Power model will still be counted under the perfect debugging assumption and thus excluded from further consideration [96:476]. In contrast, failure count models, such as the Musa Execution Time model, can incorporate imperfect debugging through a fault reduction factor of the ratio of net number of faults corrected per total number of faults corrected [64:120]. Musa et al. suggest such a ratio could be independent of specific project characteristics, and sufficient values have been determined to provide for boundary conditions and an average [64:120-121].

Assumption 3 further eliminates the times between failures models and the Yamada-Ohba-Osaki Power model, as these models require faults to be removed as soon as they are detected [34:1419], [96:476]. The last one, Assumption 4, is important, as the concept of IOT&E revolves around the time (flight, CPU, etc.) available for testing within given monetary constraints [48:114]. This assumption further eliminates fault seeding and input domain models (as neither define parameters in terms of time), and also restricts times between failures and failure count models to their execution instead of calendar time components.

3.1.4 Applicability to the Finite-Time Environment. Applicability addresses five general categories that the software reliability model should be able to deal with [39]:

- Phased integration of a program during test (result is that initial failure data is based on only a portion of the program).
- Design changes to the program.
- Failure severity classification using different categories.

- Ability to handle incomplete failure data or data with measurement uncertainties.
- Operation of the same program on computers of different performance.

Any model that meets these should then have the capability to be a single useful model, as well as something that will be applicable across different IOT&E efforts/systems. Musa et al. [64] identifies the characteristics of several models that allow for dealing with these categories. Of these models, those of both the exponential class and geometric family apply well, as initial parameters can be derived from data that exists prior to program testing (such as software size, machine execution rate, etc.) [64]. These parameters could then be further refined through data collected on any evaluation, such as software maturity, done prior to the start of IOT&E.

3.1.5 Simplicity of Design. Simplicity should be present in three areas [39]:

- It must be simple and inexpensive to collect the required data.
- The model itself should be simple in concept.
- The model must be implementable as both a useful management and engineering tool.

The Musa Execution Time model was found easy to use; however, would generally "underestimate the number of errors" [89:9]. In addition to this model, the Musa-Okumoto Logarithmic Poisson Execution Time model was also identified as one of the easiest to use models [64:398]. In contrast, the Goel-Okumoto NHPP and Jelinski-Moranda models were found to have such "numerical difficulties" that

> The issues concerning starting points for the iterative procedures, uniqueness of the parameter estimates, and even alternative estimation techniques must be studied and such problems solved before these models can be used by acquisition managers [3:273].

The Littlewood-Verrall model is very complex, very difficult to understand, and very difficult to apply on a computer [64:32]. Markov models, in general, were also found to have a "great deal of added complexity" with "much research still needed in this area" [27:4-116]. In contrast, the Poisson type models (of the exponential class) and the Musa-Okumoto Logarithmic Poisson Time model (of the geometric family) are the two simplest models to implement [34, 64, 89].

3.1.6 Diversity and Applicability of Output. The ability to express data and results in different formats is desirable considering the diversity of software systems that undergo IOT&E. Allowing the data to be presented in different formats will allow the software engineers/analysts to better convey the meaning of reliability measurements.

While all models possess the capability to provide meaningful data to the decision makers, the Poisson type and basic execution time models have the potential to encompass more than just the raw data. Of all the models evaluated, only the Musa Execution Time model and the Musa-Okumoto Logarithmic Poisson Time model have derived equations to compute current failure intensity as a function of either failures experienced or elapsed test time. No other models have straightforward equations to determine both the number of failures or amount of time that is expected to occur before reaching a desired failure intensity. Additionally, of the models evaluated, only these two had equations to relate system characteristics to the determination of initial parameters. Such equations allow for evaluation of a system where previous or similar failure data do not exist. These, and the other equations, also enable presentation of data ranging in form from engineering units vs. specific system parameters to overall trends of failures vs. system time.

3.1.7 Capability to Use Existing Initial Data. The criteria of simplicity of design addresses the ease and cost of collecting data for the reliability model. In contrast, the capability to use existing initial data evaluates a model's flexibility to be mapped to an existing database. HQ AFOTEC is developing a database of software failure data to analyze and determine the software maturity for different weapon systems. A software reliability model should then be able to use this initial data as a baseline for estimating parameters. Such estimation is important, and using initial data can reduce errors from the use of data from other "similar" systems.

Some Poisson process models use cumulative failures per test period [34]; however, the use of time *of* failure occurrence and not time *between* failure occurrence allows for modeling the failure occurrence as a random arrival event for those data points collected without time information. This process has been demonstrated in [64], and can be useful for using existing maturity data where failures per test time are the only available data.

3.2 Choice of a Reliability Model

Based on the criteria and discussions above, the following models can be dismissed as possible candidates for the following reasons:

- Mill's Hypergeometric Model. This and any other fault seeding models are not viable for IOT&E as the introduction of faults this late in software testing would adversely impact system delivery. Seeding such faults in a manner to be representative of the innate faults is very difficult, and is not practical for IOT&E of programs with extensive amounts of software. Also, the model does not support the use of execution times for failure rates.

- Ramamoorthy-Bastani Model. Input domain models are not workable due to the high cost of determining equivalence classes. Also, the model does not support the use of execution times for failure rates.
- Jelinski-Moranda Model. Parameter estimation was found to be difficult. The model does not support IOT&E assumptions of imperfect debugging for fault removal or errors not corrected immediately after a test interval. It is one of the more difficult models (numerically) to use.
- Littlewood-Verrall Model. The model does not support IOT&E assumptions of imperfect debugging for fault removal or errors not corrected immediately after a test interval. This model is also very complex and difficult to understand and apply on a computer.
- Schick-Wolverton Model. Parameter estimation was found to be difficult. The model does not support IOT&E assumptions of imperfect debugging for fault removal or errors not corrected immediately after a test interval.
- Goel-Okumoto NHPP Model. With respect to obtaining parameter estimates, it is one of the more difficult models to use. Also, this model does not support the IOT&E assumption of imperfect debugging.
- Shooman Exponential Model. This model does not support the IOT&E assumption of imperfect debugging. The model also relies on calendar time and not execution time.
- Yamada-Ohba-Osaki Power Model. Accuracy of parameter estimation not acceptable until approximately 60% into testing. The model does not support IOT&E assumptions of imperfect debugging for fault removal or errors not corrected immediately after a test interval.

Therefore, only two models from the failure count category were selected as candidate models for evaluation:

- Musa-Okumoto Logarithmic Poisson Execution Time Model. This model was found to have the best initial predictive validity for parameter estimation, as well as the best capability to be used to make software assessments. The model supports all IOT&E assumptions and easily accommodates diverse output. It can also use existing program data to determine initial model parameters.
- Musa Execution Time Model. This model was found to be one of the models having the best capability to be used to make software assessments. The model also supports all IOT&E

assumptions and easily accommodates diverse output. This model can also use existing program data to determine initial model parameters.

Although the Musa Execution Time model does not support adequate parameter estimation until 60% of testing is complete, this assessment is based on accumulated failure data and not existing program data. As the model is one of the simpler ones to implement, it is hoped that the simplicity and capability to use existing program statistics will enable closer parameter determination than is possible with using failure data alone. The Musa Execution Time model also contains the salient points from other models, such as the Goel-Okumoto NHPP model, Jelinski-Moranda model, and Shooman Exponential model [64:32]. One comparison even stated the Musa Execution Time model and Jelinski-Moranda model were equivalent, with the Musa Execution Time model considered to be "better developed" of the two [6:15]. Similarly, the Musa-Okumoto Logarithmic Poisson Execution Time model is considered a combination of the Musa Execution Time model's execution time characteristic and the "analytical ease" of the Goel-Okumoto NHPP model [58:83]. Other failure count models are similar to the candidate models, either being more generalized or more refined for specific applications [34, 64, 88, 96].

The final two selection criteria have additional impact on the implementation of these candidate models. Several tools exist which can assess software reliability with respect to different models [28, 83]; however, the thrust of these tools (and hence the model implementation) is to predict software reliability [83:1]. To fully examine the current assessment capability of the candidate models, a fresh implementation must be considered. This implementation is discussed in the next chapter.

3.3 Summary

This chapter took the models described in Chapter 2 and compared them against specific model selection criteria, with the goal of selecting one candidate model and methodology appropriate for use in the IOT&E phase of software test and evaluation. The model selection criteria were defined, and models were either vindicated or eliminated during the discussion of each criterion. The results were two, instead of a single one, software reliability models that should be appropriate for software IOT&E: the Musa Execution Time model; and the Musa-Okumoto Logarithmic Poisson Execution Time model. The implementation of these models is described in the next chapter.

IV. Software Reliability Model Implementation

This chapter contains the method and actual implementation of the candidate models identified in the previous chapter. Several software reliability implementation methodologies have been presented, including [34, 41, 64, 71]. The salient points of each have been extracted and are used as a basis for implementation of the candidate models:

- Plan a strategy [41:33-35].
- Determine software reliability goals [41:35].
- Assess existing data [34:1420].
- Select candidate model(s) [34:1420].
- Derive fitted model [34:1420],[71:50].
- Assess the model [34:1420],[71:50].
- Define and implement data collection procedures [41:35],[64:215-220].
- Assess the software reliability [34:1421],[41:36],[71:50].

A discussion of each follows.

4.1 Plan a Strategy

This step is defined as "initiate a planning process" [41:33], and will be performed at two levels. First, software reliability needs to be incorporated into the IOT&E test planning strategy. After that, the design and implementation of the candidate models will follow.

4.1.1 IOT&E Test Planning Strategy. With respect to the overall OT&E test planning strategy,

> Operational Test and Evaluation (OT&E) is conducted to estimate the system's operational effectiveness, operational suitability (including reliability, availability, maintainability, logistics supportability, and training requirements) and identify needed modification [21:3–4].

As the premise of this thesis is that software maturity data can be used as a basis for initial parameters of the software reliability measurement, the candidate models must be implemented, where possible, *after* software maturity data has been collected.

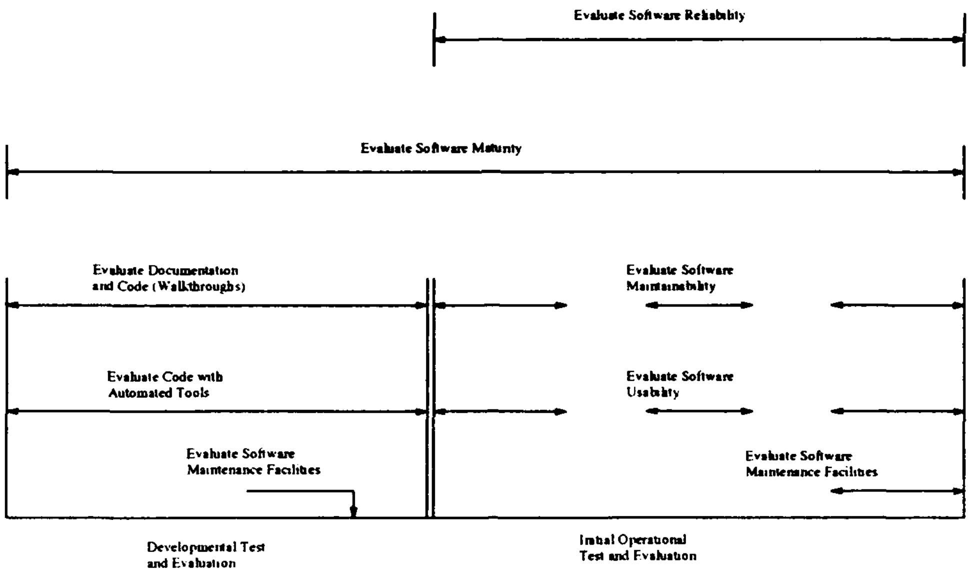

Figure 4.1. Software T&E with Software Reliability Assessment

Figure 4.1 indicates one possible method for integrating software reliability measurement into the IOT&E test effort. This figure identifies a possible relationship of software T&E during both the developmental T&E (DT&E) and OT&E phases. This method integrates software reliability evaluation with current HQ AFOTEC operational suitability assessments (software maintainability, usability, maturity, and support resources), and makes use of historical data for the same weapon system collected by software evaluation personnel prior to the start of IOT&E. Such a combined approach should provide a quantifiable way of assessing whether or not the soon-to-be operational system has "good code."

4.1.2 Program Design Strategy. The development plan for this software effort involved an analysis of the problem, specification of requirements, and development of a design based on the requirements. After this, code development and testing followed. While the waterfall model provides the structure for this type of effort, an iterative waterfall (or "waterfountain") approach was used to enable further refinement of the specifications prior to generation of data sets [84].

Structured analysis techniques were used for the initial analysis. The resultant data flow diagrams (DFDs) were used for an object-oriented design of the software. As part of the high-level design of the system, the possibility of using an abstract data type (ADT) to implement the software was considered. Program coding was done in the Clipper programming language, which is a dBASE compiler for any computer capable of running at least PC/MS-DOS version 2.0 [66:1-4]. The Clipper language was chosen for compatibility, as the current software maturity data base and supporting software were all previously developed using Clipper. Testing of the code was performed throughout the software life-cycle effort. Specific details on the analysis and design are discussed in Appendix C.

4.2 Determine Software Reliability Goals

The software reliability goals of this thesis are *not* to predict software reliability at any time in the future. Instead, the goal is to be able to define a current measure of the software such that a decision maker (the Test Director for IOT&E testing) may be able to assess how much longer it will take or how many more failures will be discovered to reach a failure intensity objective *of his/her choosing*. Typical values for operational reliability of critical software systems (such as air traffic control systems, nuclear power plants, and space systems) have ranged from 10^{-7} failures per CPU hour to 10^{-9} failures per CPU hour [64:93]. Another suggested value is a reliability of 0.999999 for a mission duration of 5 hours [71:50]. Therefore, the suggested reliability goals will be 0.999999, 0.9999, 0.99, 0.95, 0.90, 0.85, and 0.80, all of which are within the range [0,1].

In order to determine which of these is the optimum reliability goal, there are two concepts that must be considered: failure intensity at the end of IOT&E is the same as that for beginning of the software's operational life; and given an unchanging failure intensity during operations, different reliability values for operational periods can be used to assess the software reliability at end of IOT&E. While this might seem like a back-door method, it does have some merit given that engineers can not determine (with any degree of accuracy) the future reliability of software in major weapon systems. Thus, the decision maker should be able to pick a desired operational reliability (with respect to failure intensity), with the engineer then assessing the cost to reach that goal. This follows the concept that an acceptable range of reliability values should be established, given the user's requirements and needs [41:35].

In specifying the user's requirements, we will start with the basic reliability function, $R(t)$, which is given by [38:524]

$$R(t) = 1 - F(t) = \int_t^{\infty} f(x)\,dx$$

where t is the time of reliability assessment, $F(t)$ is the cumulative distribution function for failures, and $f(x)$ is the probability density function for failures [38:54,56,524]. Assuming only random failures are used (this gives an exponential time to the failure density), the reliability function is described in terms of a Poisson distribution with a mean occurrence rate λ by [38:524,526]

$$R(t) = e^{-\lambda t}$$

Musa et al. applies this to software reliability, resulting in a similar reliability function $R(\tau)$ given by [64:50]

$$R(\tau) = e^{-\lambda\tau} \tag{4.1}$$

The major assumption for this is a constant failure intensity λ for the execution time period τ [64:50]. However, this works to the advantage of the decision maker. Taking the natural logarithm of Equation 4.1 gives

$$\ln(R(\tau)) = -\lambda\tau \tag{4.2}$$

Equation 4.2 can then be used for decision support alternatives. For example, assuming a weapon system is projected to operate for (an average) of 500 hours per each calendar year, the Test Director would pick the reliability goal and the required failure intensity from the range of values derived for various λ values specified above (see Table 4.1). The reliability would be defined by the Test Director as a success criterion, and the implemented model should be able to support analysis based on current operational assessment as well as a potentially changing success criterion. In this case, the additional test time needed to reach the desired failure intensity (determined from the reliability defined by the Test Director) would then be calculated.

Table 4.1. Range of Software Reliability Goals for $\tau = 500$ Hours

λ	$R(500)$
2.00×10^{-9} Failures/Hr	0.999999
2.00×10^{-7} Failures/Hr	0.9999
2.01×10^{-5} Failures/Hr	0.99
1.03×10^{-4} Failures/Hr	0.95
2.11×10^{-4} Failures/Hr	0.90
3.25×10^{-4} Failures/Hr	0.85
4.46×10^{-4} Failures/Hr	0.80

This is supported by the candidate models, as predicted and measured quantities (number of failures remaining and mean time to fail, respectively) at the start of operations "are constant and equal to those at the end of the last test phase (unless errors are corrected, in which case the operational phase should be considered as a 'test' phase or phase of reliability growth)" [60:313]. Thus, the desired final reliability value (λ_F) is determined from Equation 4.2, and the present failure intensity during IOT&E testing (λ_P) is determined from either Equations 2.9 and 2.10 or Equations 2.12 and 2.13. The amount of additional test time ($\Delta\tau$) necessary to reach the desired software reliability level is then determined by the Musa Execution Time model from [64:45]

$$\Delta\tau = \frac{\nu_0}{\lambda_0} \ln \frac{\lambda_P}{\lambda_F} \tag{4.3}$$

and by the Musa-Okumoto Logarithmic Poisson Execution Time model from [64:45]

$$\Delta\tau = \frac{1}{\theta}\left(\frac{1}{\lambda_F} - \frac{1}{\lambda_P}\right) \tag{4.4}$$

Therefore, if the test time needed to reach a desired failure intensity objective was deemed to be too much by the Test Director, he/she would then have to choose a lower reliability goal. obtain additional test time, or alter some other aspect of the software development process to compensate. Thus, the actual software reliability goals will be determined by the decision maker. and are subject to change based on the availability of test resources (primarily time). This means the implementation must support some form of decision support scenario.

4.3 Assess Existing Data.

Shaw noted

> The problem in applying software metrics is to find appropriate measures and make sense out of the data, not simply to obtain the data [75:257].

The goal of software reliability assessment is to make the data useful, thus something must be determined from the data, even if that means discovering that nothing can be determined from the data. From the HQ AFOTEC software maturity data, 17 initial data sets were available that included aircraft, communications, missile, radar, and space systems. For these data sets, the number and type of record fields varied; however, there was a common set of fields across all 17 data sets. These fields are identified in Table 4.2.

None of the data in the 17 different data base files contained information about test durations or specific descriptions of the system under test (for example, number of source lines of code or processor execution rate). Such additional information was necessary to run the models; however, due to the very recent incorporation of software maturity assessment in the IOT&E planning strategy, this initial data was "fragmented and incomplete" [45]. Therefore, a data assessment strategy was devised where candidate data sets were chosen based on the availability of *any* test

Table 4.2. Common Software Maturity Data Fields

Field Name	Description
Date	Date of Problem
CPCI	Software Configuration Item
Sev_Code	Severity of Problem
Date_Fix	Date Problem Fixed
Title	Description of Problem
Prob_Num	Software Problem Number

duration data. This limited the data sets to three types of weapon systems: aircraft (denoted by A), space systems (denoted by S), and weapon system trainers (denoted by W). These data sets were then plotted with failure count indicated as a function of execution time [34:1420]. An assessment was then made as to the applicability of the candidate models based on the initial curve of the data. The results of this, as well as the application of the models to the data, are discussed in the next chapter.

4.4 Selection of Candidate Models.

Assumptions for each model, evaluation of each model with respect to specific acceptance criteria, and selection of candidate models were discussed in the previous chapter.

4.5 Derive the Fitted Model

This procedure involves both estimating the parameters for the model, and then substituting these parameters into the model to fit the model for the data [34:1420]. An additional version of each fitted model was derived for those models that had prior DT&E test data. A discussion of initial parameter estimates appears in the first section, followed by a discussion of the derived parameter estimates.

4.5.1 Model Parameter Estimation. Musa et al. define equations for failure intensity and mean value functions for both the Execution Time model and Logarithmic Poisson Execution Time model (see Table 4.3) [64:307]. From these, the parameters $\beta_0 = \nu_0$ (the total failures at time $t = \infty$ for the Execution Time model) and $\beta_0^{-1} = \theta$ (the failure intensity decay parameter for the Logarithmic Poisson Execution Time model) need to be determined [64:351]. Parameter β_0, as well as other estimated values, are a function of β_1 which is defined as $\beta_1 = \lambda_0/\nu_0$ for the Execution Time model, and $\beta_1 = \lambda_0\theta$ for the Logarithmic Poisson Execution Time model [64:351,529]. The

Table 4.3. Specific Model λ and μ Functions

Model	$\mu(t;\beta)$	$\lambda(t;\beta)$
Execution Time	$\beta_0[1-e^{-\beta_1 t}]$	$\beta_0\beta_1 e^{-\beta_1 t}$
Logarithmic Poisson Execution Time	$\beta_0 \ln(1+\beta_1 t)$	$\frac{\beta_0\beta_1}{1+\beta_1 t}$

parameter β_1 itself is estimated for the Execution Time model by [64:325]

$$\frac{m_e}{\beta_1} - \frac{m_e t_e}{e^{\beta_1 t_e} - 1} - \sum_{i=1}^{m_e} t_i = 0 \tag{4.5}$$

and is estimated for the Logarithmic Poisson Execution Time model by [64:326]

$$\frac{1}{\hat{\beta}_1}\sum_{i=1}^{m_e}\frac{1}{1+\hat{\beta}_1 t_i} - \frac{m_e t_e}{(1+\hat{\beta}_1 t_e)\ln(1+\hat{\beta}_1 t_e)} = 0 \tag{4.6}$$

4.5.1.1 Newton-Raphson Method. One way of estimating parameters is with the Newton-Raphson method, which has the general form [14:48]

$$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})} \quad n \geq 1$$

This is calculated based on a simple algorithm, such as the one presented by Burden and Faires [14:49]:

To find a solution to $f(x) = 0$ given an initial approximation p_0:

INPUT initial approximation p_0; tolerance TOL; maximum number of iteration N_0.

OUTPUT approximate solution p or message of failure.

Step 1. Set $i = 1$

Step 2. While $i \leq N_0$ do Steps 3-6.

Step 3. Set $p = p_0 - f(p_0)/f'(p_0)$. (*Compute* p_i.)

Step 4. If $| p - p_0 | < TOL$ then OUTPUT p; STOP.

Step 5. Set $i = i + 1$.

Step 6. Set $p_0 = p$.

Step 7. OUTPUT ('Method failed after N_0 iterations, N_0 =',N_0; STOP.

Angus et al. note a problem with the Newton-Raphson method, and state

> In the actual use of the Newton-Raphson method, convergence of the estimators to finite values could not always be obtained. The major problem seemed to be in finding successful starting points for the parameter estimates as inputs to the program. In general, no real guidelines were found [4:194].

As the maximum likelihood estimation of parameters for both models is based on the single parameter β_1, this requires only one initial starting point necessary for the Newton-Raphson method [64:526]. Musa et al. suggest an initial estimate for β_1 to be t_e^{-1}, the inverse of the total testing time, and state that "this value almost always results in the initial convergence of the Newton-Raphson procedure" [64:527]. Therefore, t_e^{-1} will be used as the initial estimate for the parameter β_1.

Applying the Burden and Faires algorithm to equations 4.5 and 4.6 requires the first derivative of each. Taking the first derivative of Equation 4.5, we get

$$f'(\hat{\beta}_1) = (m_e)\left(-\frac{1}{\hat{\beta}_1^2}\right) - (m_e t_e)\left[\frac{-t_e e^{\hat{\beta}_1 t_e}}{(e^{\hat{\beta}_1 t_e} - 1)^2}\right] \tag{4.7}$$

and taking the first derivative of Equation 4.6 gives

$$f'(\hat{\beta}_1) = \left[\left(\frac{1}{\hat{\beta}_1}\right)\left(\sum_{i=1}^{m_e}\frac{-t_i}{(1+\hat{\beta}_1 t_i)^2}\right) + \left(\sum_{i=1}^{m_e}\frac{1}{1+\hat{\beta}_1 t_i}\right)\left(-\frac{1}{\hat{\beta}_1{}^2}\right)\right] - \frac{\left[-(m_e t_e^2)(1+\ln(1+\hat{\beta}_1 t_e))\right]}{[(1+\hat{\beta}_1 t_e)\ln(1+\hat{\beta}_1 t_e)]^2} \tag{4.8}$$

4.5.1.2 Additional Initial Parameter Estimation. Equation 4.7 then is used to calculate an estimated $\nu_0 = \beta_0$ for the Execution Time model [64:325]

$$\hat{\beta}_0 = \frac{m_e}{1 - e^{-\hat{\beta}_1 t_e}} \tag{4.9}$$

Recalling that $\beta_1 = \lambda_0/\nu_0$ for the Execution Time model, the estimated initial failure intensity value λ_0 is then calculated as [64:351,529]

$$\lambda_0 = \beta_0\beta_1 \tag{4.10}$$

Similarly, Equation 4.8 is used to calculate an estimated $\theta^{-1} = \hat{\beta}_0$ for the Logarithmic Poisson Execution Time model [64:326]

$$\hat{\beta}_0 = \frac{m_e}{\ln(1 + \hat{\beta}_1 t_e)} \tag{4.11}$$

Recalling that $\beta_1 = \lambda_0 \theta$ for the Logarithmic Poisson Execution Time model, the estimated λ_0 value can also be calculated from Equation 4.10 [64:351].

4.5.1.3 Confidence Intervals. Confidence intervals for the estimated parameters were developed based on the assumptions of a normal distribution, zero mean, unit variance, and a desired confidence interval of 95 percent [64:316]. Such an approach allows a $100(1 - \alpha)$ percent confidence interval to be calculated for the unknown mean μ from the sampling distribution of $\bar{X}$ the sample mean [38:242]. The general form of the equation for this two-sided confidence interval is [38:242]

$$\bar{X} - \frac{Z_{\alpha/2}\sigma}{\sqrt{n}} \leq \mu \leq \bar{X} + \frac{Z_{\alpha/2}\sigma}{\sqrt{n}} \tag{4.12}$$

For a 95 percent confidence interval $\alpha = .05$ with $\alpha/2 = .025$. From a cumulative standard normal distribution table, the test statistic $Z_{.025} = 1.96$ [38:243,593]. Musa et. al. apply this, as well as the unit variance assumption of $\sigma = 1$, to *Equation 4.12 and derive* the following version of the two-sided confidence interval for the estimated parameter $\hat{\beta}_k$ [64:316]

$$\hat{\beta}_k \pm \frac{\kappa_{1-\alpha/2}}{\sqrt{I(\hat{\beta}_k)}} \tag{4.13}$$

with $\kappa_{1-\alpha/2}$ being "the appropriate normal deviate" and $I(\hat{\beta}_k)$ being the "expected, or Fisher, information" [64:315-316]. The appropriate normal deviate equates to the test statistic

$$\kappa_{1-\alpha/2} = Z_{.025} = 1.96$$

and the expected information for $I(\beta_1)$ can be determined for the Execution Time model from [64:351]

$$I(\beta_1) = m_e \left\{ \frac{1}{\beta_1^2} - \frac{t_e^2 e^{\beta_1 t_e}}{[e^{\beta_1 t_e} - 1]^2} \right\} \tag{4.14}$$

with the value for the Logarithmic Poisson Execution Time model determined from [64:334]

$$I(\beta_1) = m_e\left\{\frac{2t_e}{\beta_1(1+\beta_1 t_e)\ln(1+\beta_1 t_e)} - \frac{1}{2\beta_1^2\ln(1+\beta_1 t_e)}\left[1-\frac{1}{(1+\beta_1 t_e)^2}\right] - \frac{t_e^2[\ln(1+\beta_1 t_e)+1]}{[(1+\beta_1 t_e)\ln(1+\beta_1 t_e)]^2}\right\} \tag{4.15}$$

Equations 4.14 and 4.15 are then substituted into Equation 4.13 to determine the upper and lower 95 percent confidence parameters of $\hat{\beta}_1$. All three values (β_1, $\hat{\beta}_{1_{low}}$, and $\beta_{1_{high}}$) are used in Equation 4.9 to determine ν_0 and its confidence boundary, and also in Equation 4.11 to determine θ and its confidence boundary. The results of these are then used in Equation 4.10 to determine λ_0 and its 95 percent boundary. The different values of λ_0 and ν_0 are used in Equations 2.8, 2.9, and 2.10 to evaluate the applicability of the Execution Time model, while the different values of λ_0 and θ are used in Equations 2.11, 2.12, and 2.13 to evaluate the applicability of the Logarithmic Poisson Execution Time model.

4.5.2 Model Parameter Derivation. Applying the techniques and equations identified in the previous section to strictly DT&E data results in a final failure intensity that can be based either on time of last failure ($\lambda(\tau)$) or on the number of failures experienced at that time ($\lambda(\mu)$) [64]. As these values are at the end of DT&E, they also represent the failure intensity values at the start of the next phase of testing, IOT&E. Therefore, the value of λ_0 is known at the start of IOT&E. Assuming additional data are not available (either with respect to failures or system characteristics), calculation of the initial parameter β_1 was based on the equations used to derive λ_0.

The equation for λ_0 for the Execution Time model is based on Equation 4.10, and in its expanded form is [64:351]

$$\lambda_0 = \frac{m_e\beta_1}{1-e^{-\beta_1 t_e}} \tag{4.16}$$

with the expanded form of the Logarithmic Poisson Execution Time model also based on Equation 4.10 and given by [64:351]

$$\lambda_0 = \frac{m_e\beta_1}{\ln(1+\beta_1 t_e)} \tag{4.17}$$

Subtracting λ_0 from both sides and setting these equations equal to 0 allowed the Newton-Raphson method to be used to determine the value of β_1.

4.5.2.1 Newton-Raphson Method. Again applying the Burden and Faires algorithm, the first derivative of Equation 4.16 is

$$f'(\hat{\beta}_1) = \frac{(1 - e^{-\hat{\beta}_1 t_e})(m_e) - (m_e\hat{\beta}_1)(t_e e^{-\hat{\beta}_1 t_e})}{(1 - e^{-\hat{\beta}_1 t_e})^2} \tag{4.18}$$

and taking the first derivative of Equation 4.17 gives

$$f'(\hat{\beta}_1) = \frac{(\ln(1 + \hat{\beta}_1 t_e))(m_e) - (m_e\hat{\beta}_1)(\frac{1}{1+\hat{\beta}_1 t_e})(t_e)}{(\ln(1 + \hat{\beta}_1 t_e))^2} \tag{4.19}$$

Therefore, the initial derivation of $\hat{\beta}_1$ was determined after λ_0. While, these equations use typical end-of-test variables, such as t_e and m_e, these variables are cumulative and can reflect even the early stages of testing. For the purpose of this study, only final IOT&E data was used after initial parameter derivation from DT&E data, as this was believed to provide a better description of the mapped models.

4.5.2.2 Additional Initial Parameter Derivation. Once $\hat{\beta}_1$ was derived, other initial values were then derived. For the Execution Time model, $\nu_0 = \hat{\beta}_0$ was derived from Equation 4.9, while Equation 4.11 was used to derive $\theta^{-1} = \hat{\beta}_0$.

4.5.2.3 Confidence Intervals. Confidence intervals for the derived parameters were developed based on the assumptions and equations presented in the previous section on model parameter estimation. Once boundary values were derived, those values along with λ_0 and ν_0 were used in Equations 2.8, 2.9, and 2.10 to evaluate the applicability of the Execution Time model, and the different values of λ_0 and θ were used in Equations 2.11, 2.12, and 2.13 to evaluate the applicability of the Logarithmic Poisson Execution Time model.

4.6 Assess the Models

Implementation and code development was conducted in accordance with the software development lifecycle, and documented as such. A modular approach was used with the code to facilitate changes during the experimental process. This proved useful, as an additional module was added during the models' evaluation. The exact implementation details of the analysis code are included in Appendix D. An assessment of the models and their performance follows in the next chapter.

4.7 *Define and Implement Data Collection Procedures.*

As failure and date data had already been collected, the only additional effort was to locate the test duration and time information needed for the models. The results of this are given in the following chapter. Future efforts to collect software reliability data must include such test duration and test time as important information. This also will be discussed in the following chapters.

4.8 *Assess the Software Reliability.*

This is the next logical step, and involves actual implementation of the candidate models on a real project with actual data. Such an assessment of the software would be based on the models' results. As the goal of this thesis is to evaluate the software reliability models and not the reliability of the test data software, comments concerning the reliability of the test data software is limited to discussion of the models' applicability and not the software systems' reliability. From this, a proposed IOT&E software reliability methodology will be discussed in the following chapters.

4.9 *Summary*

This chapter identified the implementation strategy for assessing the candidate software reliability models. As the integration of software reliability is new to operational test and evaluation of weapon systems, this chapter also identified the place a software reliability model implementation strategy would have in the IOT&E environment. Results and discussion of the candidate software reliability models' implementation follow in the next chapter.

V. Findings

This chapter presents the initial data analysis findings, the findings of the fitted models with respect to the actual failure data, a comparison of the failure intensity values for each data set, and an evaluation of each model fitted for IOT&E failure data from historical DT&E failure data.

5.1 Initial Data Analysis

The basic data fields listed in Table 4.3 were not sufficient for use with a software reliability measurement model, as they were lacking some sort of failure time indication. Additional information on timing and system characteristics was identified [45, 47]; however, of the initial 17 data sets available, only five had sufficient supplemental information to make the maturity data meaningful in a software reliability sense. Therefore, the initial data analysis was conducted using only these five data sets. Line charts were plotted for each using cumulative total failures for the y-axis and execution test time for the x-axis to visually determine the trends of each curve. The results are shown in Figures 5.1 through 5.6 and discussed below.

5.1.1 Data Set A1. The test durations used for this data set varied from 30 to 738 minutes. Although the IOT&E dates were from July 1984 to June 1989 and test durations and dates were available for the entire IOT&E period, the available data from the HQ AFOTEC software maturity database (named SYSTERR) covered only the dates of 27 August 1987 to 30 April 1988, inclusive, with one lone data point on 1 October 1986 (see Appendix B) [45]. This totaled six months of data, with a total of 1465 failures indicated. The lone data point was excluded from initial and subsequent analysis as the test duration time span between this and the next data point was too great for the point to be meaningful. This assumption was based on the author's personal experience from performing IOT&E on this weapon system. Also, if the trend is statistically sound, the absence of one data point on either end should not affect the overall integrity of the data.

Initial analysis of the cumulative failure data reveals an exponential-like trend with respect to execution test time (see Figure 5.1). This is encouraging, as the software maturity data is based on calendar time (independent of execution time or test duration), and itself has an exponential tendency [94]. Although the exact time of each failure occurrence was not known, times were assumed to follow a uniform distribution, and were assigned randomly to each failure event within the total test duration for that calendar day [64:158].

There was some difficulty mapping dates of failures to the dates of actual test durations. In some cases, dates listed for failures did not have a date of test duration, and conversely some

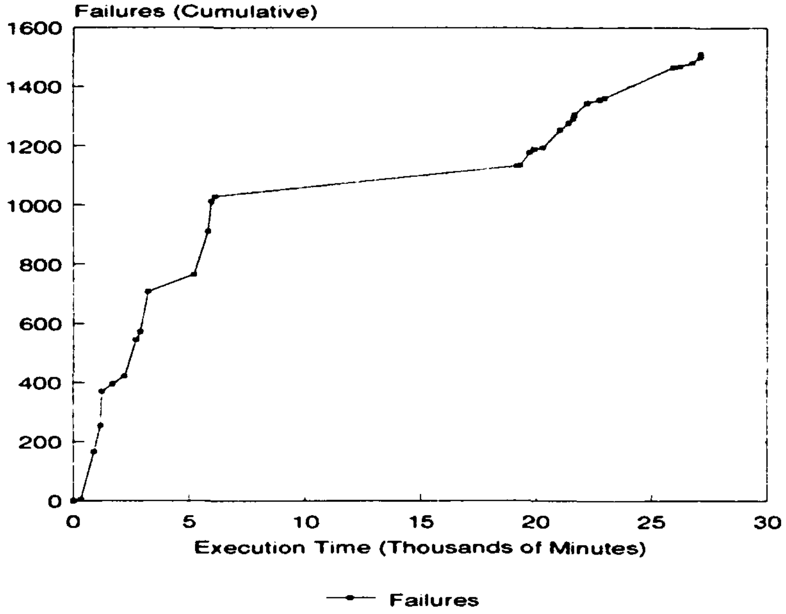

Figure 5.1. Cumulative Failures vs Execution Time for Data Set A1

test durations did not have associated dates of failures. The software listed in Appendix D was modified to include a special module that would compensate for this discrepancy as follows. If test durations did not have associated failures (or dates had multiple test durations), the test times were added to the total test duration as an offset. Failures that had no associated test durations were then added together until an existing test duration date was reached, and all were applied against that date. Admittedly this approach seems unfair in that failures listed between test durations should be applied against the previous test duration (as that is likely to be where the failures were found); however, given the seeming randomness in association between date of failure and date of test duration the method used should not unduly skew the data. The only visible instance of this smoothing is the rather flat slope directly in the middle of the curve. Again, as the overall curve tended towards exponential, this smoothing should not have any affect on the data or subsequent calculations.

Thus, it appears initially that both the Execution Time Model and the Logarithmic Execution Time Model should fit this data distribution; however, as HQ AFOTEC is involved with several different types of weapon systems, additional data sets must be analyzed for model applicability.

5.1.2 Data Set A2. The IOT&E for this system was from December 1988 to September 1989, during which there were 512.4 hours of testing with 304 total testing periods [45]. The failure data available ranged from 24 February 1987 through 25 July 1989. During the IOT&E timeframe, there were eight months of testing and a total of 47 recorded failures. An initial assumption was made that each test duration was 1.686 hours long (512.4/304=1.686); however, there were only 37 failure dates listed from the SYSTERR database for the IOT&E period which would leave 267 test durations unaccounted (see Appendix B).

Since the number of failure dates did not correspond in any way to the number of test periods, another way to determine the failure to test duration relationship was needed. Available information for average test durations of similar weapon systems was used as a starting point to determine an approximate relationship. The average number of test flights per aircraft per month for a fighter type aircraft is 10 flights/aircraft/month, with the average number for a larger type aircraft (such as a bomber) being 5 flights/aircraft/month [1:3],[48:144]. A similar test program used four total aircraft for testing [45]. Therefore, an assumption was made that four aircraft were used with each having 10 test flights per month. This gave an approximate total of

(*4 aircraft*)(*10 flights/month*)(10 months)=400 sorties (or test durations)

that would have occurred from December 1988 to September 1989. As the actual number of test durations was less than the estimated number, and assuming a standard normal distribution, either the assumed number of test aircraft should be reduced giving

(3 aircraft)(10 flights/month)(10 months)=300 sorties

or the number of flights per month should be reduced giving

(4 aircraft)(8 flights/month)(10 months)=320 sorties

Varying the number of test aircraft yields the closer approximation, with the additional time from the last four sorties easily applied to the last month of testing (which is acceptable, as there is no failure data for any month past July 1989). Therefore, 30 test durations of 1.686 hours each (50.58 total test hours) were assumed to occur each month, with 34 test durations of 1.686 hours each (57.32 total test hours) assumed to occur in the last month of testing.

Cumulative total number of failures were determined for these durations based on the following. Assuming a normal distribution for the dates of test, each month was treated as a total test duration of 50.58 hours (57.32 for the final month). The number of failures per month were then added, and assigned randomly within that test duration. The results are shown in Figure 5.2. By inspection, the data appears to follow some form of exponential curve. While the trend is more S-shaped, there appears to be enough of an exponential shape to proceed with the candidate models on this data set as well.

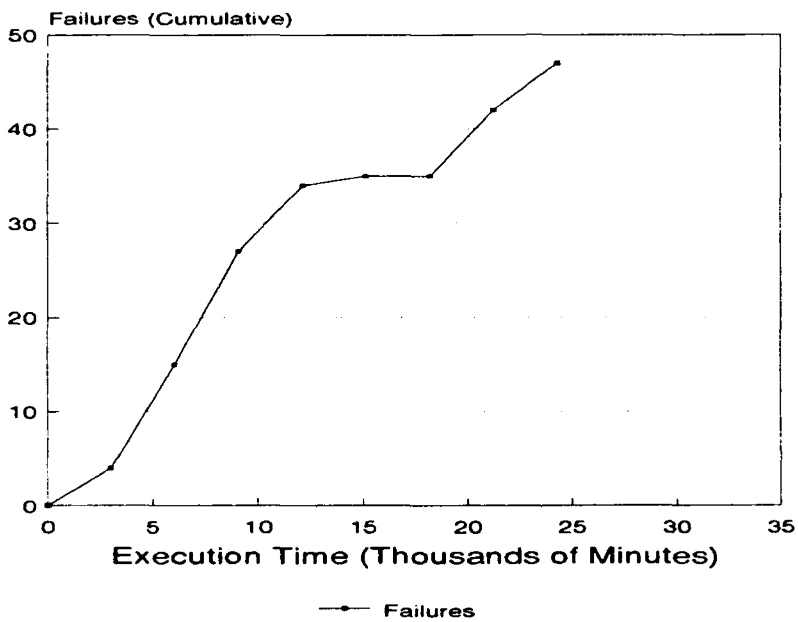

Figure 5.2. Cumulative Failures vs Execution Time for Data Set A2

5.1.3 Data Set A3. There were 219 test periods, four test aircraft, and an average test duration of 1.5 hours for IOT&E of this system which lasted from 23 May 1989 to 1 November 1989 [45]. This gave 5.25 months of IOT&E and 50 recorded failures. Using the relationship defined above, that gives

(4 aircraft)(10 flights/month)(5.25 months)=210 sorties

which is extremely close to the 219 actual test flights. Varying the number of flights per month (which is itself an average) to 11 gives

(4 aircraft)(11 flights/month)(5.25 months)=231 sorties

A closer approximation was obtained by taking the 219 sorties and dividing back by the number of months (5.25), which yields 41.7 test flights per month. At 1.5 hours each (on the average) the total test time per month is then

(41.7 test flights)(1.5 hours/test flight)=62.55 hours

with the first month of testing having only 15.6 test hours due to only 8 days of testing occurring in the first month.

Cumulative total number of failures were determined for these test durations based on the same assumptions that were used with the A2 data set. A normal distribution was assumed for the dates of test, with each month treated as having a total test duration of 62.55 hours (15.6 for the first month). The number of failures per month were then added, and assigned randomly within that test duration. The results are shown in Figure 5.3. This curve exhibits more dramatic changes in the cumulative failures than the previous data sets. Even so, the general trend should permit the use of the candidate models.

5.1.4 Data Set S1. IOT&E for this system lasted from 3 February 1988 until 6 July 1989 [45]. A total of five two-week test periods occurred at three different test sites (two two-week periods of testing at one site, two two-week periods of testing at another site, and one two-week period of testing at the third site), with an average of 20 hours per day of testing for 14 straight days [47].

The use of three different test sites normally requires adjusting the test durations and times of failure occurrences. Musa et al. provides an excellent description of how to interleave test time and failure occurrence for multiple test installations [64:162-165]. Normally, one would think to use independent failure intervals for each program, as with the hardware for a system; however, due to the logical nature of software a failure and test time interleaving is more appropriate [64:162-165].

For this application, the exact time of each failure occurrence is not known. Therefore, interleaving is not applicable, and it will be sufficient to take the total test duration of

(20 hours testing per system per day)(3 systems)=60 hours testing per day

and divide that by the number of failures occurring on that day. Since the five two-week test periods were well within the start and stop dates for IOT&E, and there were failure data for other

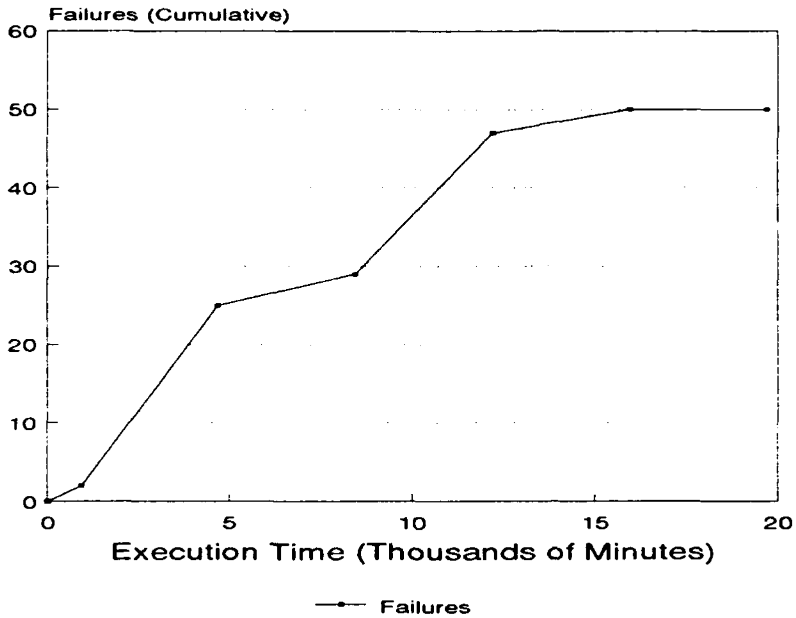

Figure 5.3. Cumulative Failures vs Execution Time for Data Set A3

dates inside the IOT&E timeframe, the total IOT&E time was considered to be $\sum$ (60 hours per day)(number of days in the month) for a total of 16 months of failure data (see Appendix B). The exact test time, therefore, varied with the number of days in the month and totaled 27780 hours (an average of 1736.25 test hours per month). There were 413 recorded failures. The results of this are shown in Figure 5.4. This data set has, by far, exhibited the closest approximation to an exponential curve. Therefore, the candidate models should work very well with this data set.

5.1.5 Data Set W1. There were no IOT&E dates nor test durations given for this system [45]. The total SYSTERR database was used, resulting in an assumed 7 months of IOT&E with 450 recorded failures. Therefore, based on the author's limited involvement with a similar system and the frequency of failure dates, an initial assumption was made that all tests dates were valid data points, test durations only occurred on the dates of failure identification (as determined from the SYSTERR database), and that each test duration was six hours long. This resulted in the data increasing in a linear fashion (see Figure 5.5).

Subsequent research indicated that actual test durations were 16 hours each, and another assumption was made that testing was conducted for five working days each week [46]. At an

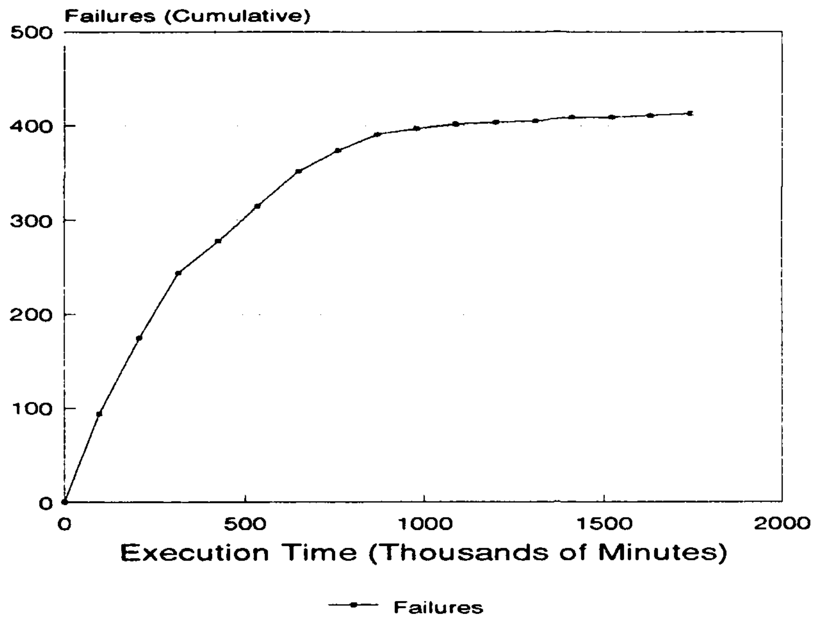

Figure 5.4. Cumulative Failures vs Execution Time for Data Set S1

average of 22 working days per month (or 4.4 weeks per month), and still using all failure dates, that results in

(22 working days per month)(16 hours testing per day)=352 hours testing per month

or 80 hours of testing per week. The results of this new calculation are shown in Figure 5.6, and the data distribution is much more exponential than under the previous assumptions. This is not an instance where the assumptions were changed to provide data that fit the models; instead, the initial assumptions were modified as additional data became available. Based on the additional data, the candidate models should also be applicable to this data set.

5.2 *Calculated Values for Current Number of Failures Compared to Actual Number of Failures*

After an initial model feasibility assessment of the data indicated the candidate models were feasible for the data sets based on the data sets' apparent exponential distributions, parameter estimates were obtained, fitted models were derived, and goodness-of-fit tests performed for each

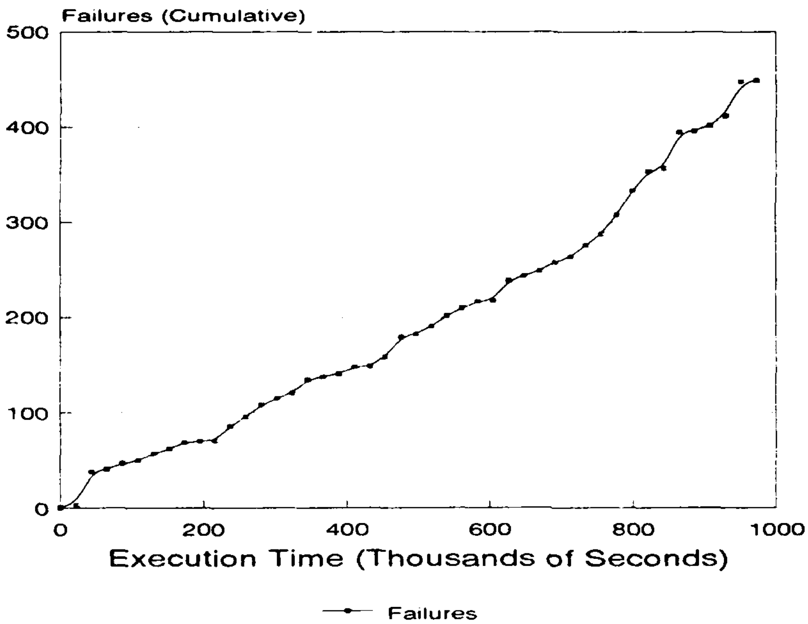

Figure 5.5. Cumulative Failures vs Execution Time for Data Set W1, Initial

model/data set combination. It is helpful at this point to redefine the goal of this thesis in terms of a null hypothesis such that [38:280]

$$H_0 : \theta = \theta_0$$
$$H_1 : \theta \neq \theta_0$$

where θ_0 is a parameter being assessed against an $[L, U]$ interval with $100(1-\alpha)$ percent confidence [38:280]. The test then leads to rejection of the null hypothesis H_0 if the parameter θ_0 is outside the 95 percent confidence interval [38:280].

The second assessment is concerned with the calculated values for "current" number of failures compared to the actual number of failures for any given time during the entire IOT&E test period. Therefore, θ is the actual number of failures experienced, and the parameter θ_0 is expected number of failures at time τ, or $\mu(\tau)$, derived from Equations 2.8 and 2.11. The $[L, U]$ boundaries are calculated for the initial parameter β_1 based on Equation 4.13 and either Equation 4.14 for the Execution Time model or Equation 4.15 for the Logarithmic Poisson Execution Time model. The parameter and its boundaries are then used in Equation 2.8 for the Execution Time model and

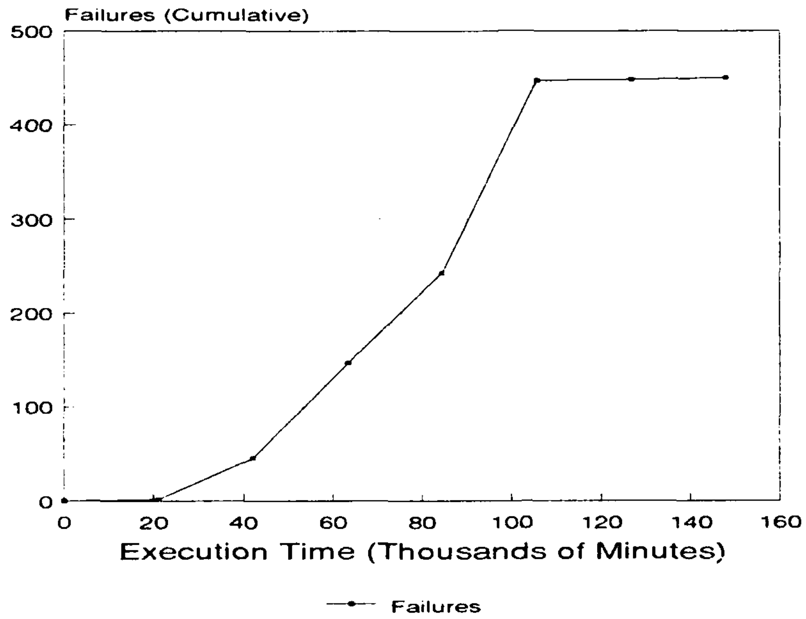

Figure 5.6. Cumulative Failures vs Execution Time for Data Set W1

Equation 2.11 for the Logarithmic Poisson Execution Time model. The results of this are shown in Figures 5.7 through 5.16 and discussed below.

5.2.1 Data Set A1. The results of the fitted Execution Time model application to the data are shown in Figure 5.7. Equation 2.8 was fitted with values of the initial parameters λ_0 = 0.162871611 and ν_0 = 1628.74 to get the following equation for failures expected at time τ

$$\mu(\tau) = 1628.74\left[1 - \exp\left(-\frac{0.162871611}{1628.74}\tau\right)\right] \tag{5.1}$$

The actual data tends outside the projected 95% confidence intervals; however, this represents only a small part of this weapons system's entire IOT&E effort. The tendency outside the confidence intervals could be due to the small snapshot of data used (6 months of recorded maturity data compared to almost 5 years of IOT&E), or to the failure time assignment process. This process prohibits identifying exact failure times (there was no initial correlation between maturity failures and dates of testing), and results in reporting the failures as "lump sums" at varying time intervals based on a calendar date relationship. Therefore, while we apparently reject the null hypothesis, additional test data on either end of the curve for a substantial amount of time would provide a more accurate assessment of the Execution Time model.

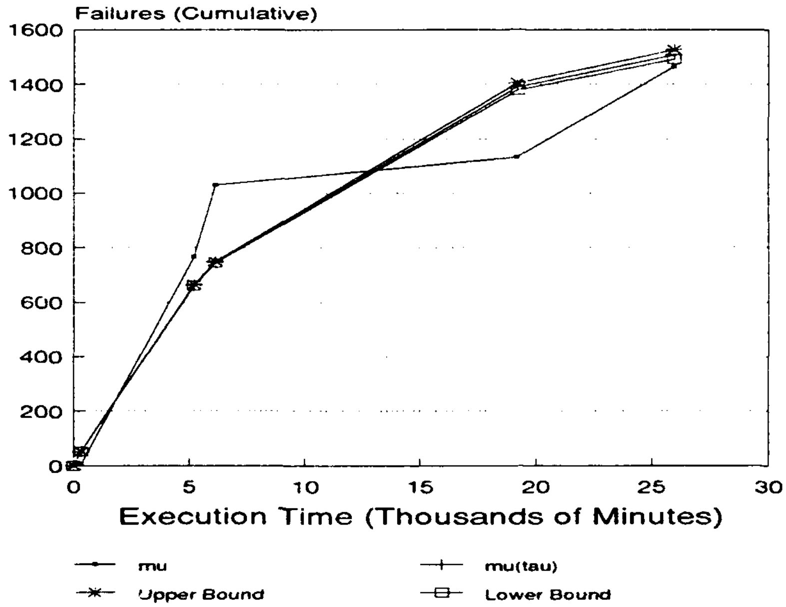

Figure 5.7. Expected Failures Using Execution Time Model for Data Set A1

This same observation holds for the Logarithmic Poisson Time model, whose results are shown in Figure 5.8. Equation 2.11 was fitted with $\lambda_0 = 0.322609809$ and $\theta = 0.001883754$ as initial parameters to get the following equation for failures expected at time τ

$$\mu(\tau) = \frac{1}{0.001883754} \cdot \ln((0.322609809)(0.001883754)\tau + 1) \tag{5.2}$$

The Logarithmic Poisson Time model did provide a closer fit to the data; however, three of the five data "lump sums" were significantly outside the confidence intervals, and we therefore reject the null hypothesis. In this case, a more accurate determination of failure times could provide a better representation of failure times, and possibly an even closer fit of this model.

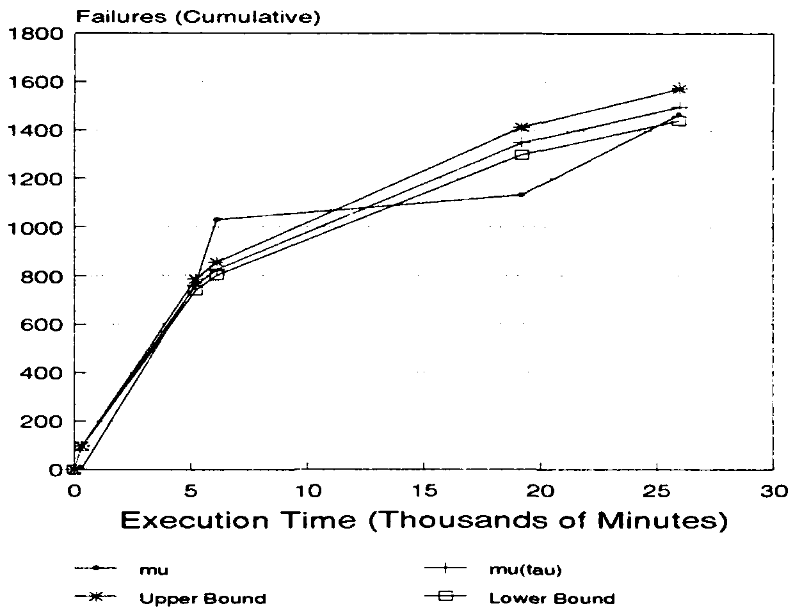

Figure 5.8. Expected Failures Using Logarithmic Poisson Execution Time Model for Data Set A1

5.2.2 Data Set A2. The fitted form of the Execution Time model had initial parameters $\lambda_0 = 0.003096631$ and $\nu_0 = 73.24$ to get the following equation for failures expected at time τ

$$\mu(\tau) = 73.24\left[1 - \exp\left(-\frac{0.003096631}{73.24}\tau\right)\right] \tag{5.3}$$

There was a substantially better fit of the Execution Time model to the A2 data than the A1 data, as can be seen in Figure 5.9. All but the two initial data points were within the 95% confidence intervals. This trend is not uncommon for the Execution Time model, which tends to perform more satisfactorily after the first 60% of the test time period [64, 89]. Overall, there was a good fit of the model to the data, and we fail to reject the null hypothesis.

Similarly, the Logarithmic Poisson Execution Time model performed better for this data set than for the previous data set (see Figure 5.10). The model was fitted with $\lambda_0 = 0.003267847$ and $\theta = 0.020626162$ as initial parameters to get the following equation for failures expected at time τ

$$\mu(\tau) = \frac{1}{0.020626162} \cdot \ln((0.003267847)(0.020626162)\tau + 1) \tag{5.4}$$

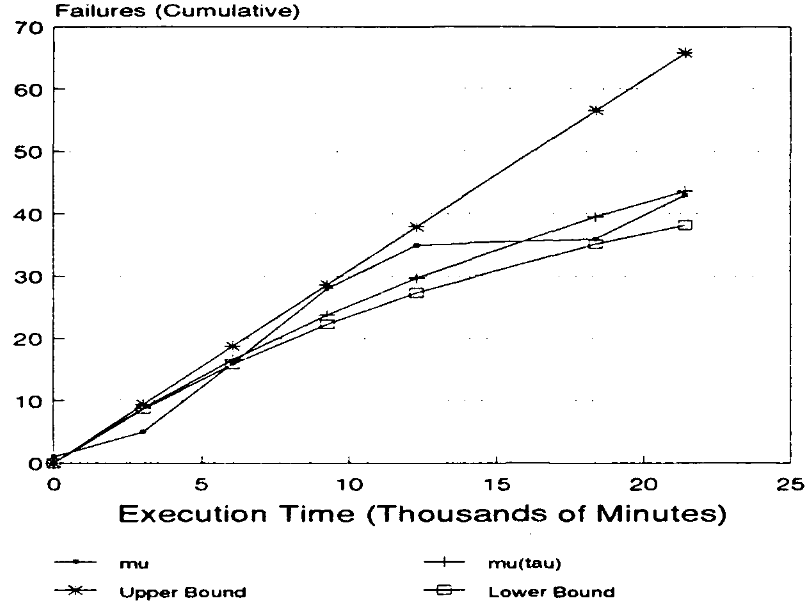

Figure 5.9. Expected Failures Using Execution Time Model for Data Set A2

One possible reason for the better fit could be the data set being complete with respect to the amount of IOT&E test time and number of failures recorded, while the previous A1 data set contained only a portion of the overall operational testing effort. It is interesting to note the Logarithmic Poisson Execution Time model does not fit as well to the data as does the Execution Time model. This could be due to failures not having specific occurrence times–the combination of using average test durations per month and assigning normally distributed random times as failure occurrence times could produce clustering of data. While these clustered points do provide adequate trend analysis, a more accurate representation of the failure time data could indicate a much closer model fit. As it stands, we must reject the null hypothesis for this candidate model with the data set.

5.2.3 Data Set A3. The results for data set A3 are similar to those of data set A2, and are shown in Figure 5.11. There appears to be a closer model fit for A3 than either of the previous two data sets using the Execution Time model, leading us to fail to reject the null hypothesis. Again, this could be due to this data set being complete with respect to the data that was available, even though the test times per month were derived from an average. The fitted form of the Execution

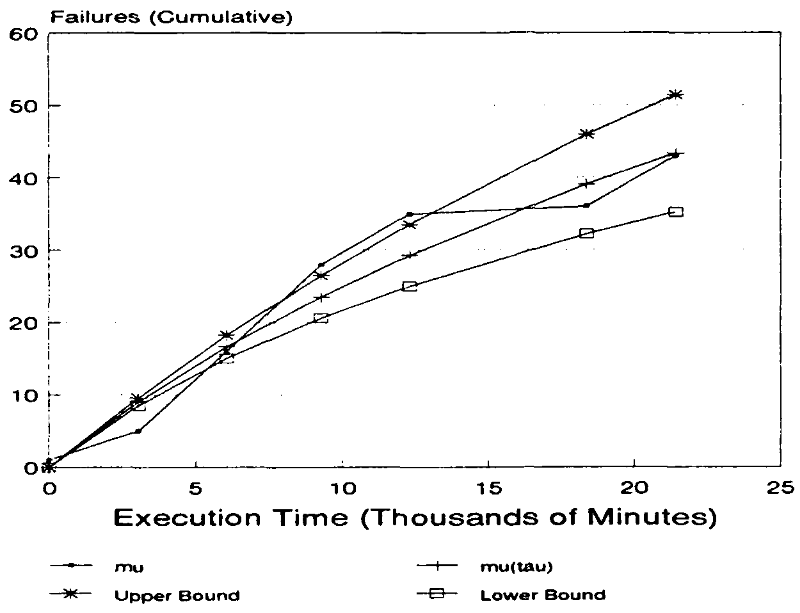

Figure 5.10. Expected Failures Using Logarithmic Poisson Execution Time Model for Data Set A2

Time model had initial parameters $\lambda_0 = 0.005358151$ and $\nu_0 = 82.74$ to get the following equation for failures expected at time τ

$$\mu(\tau) = 82.74 \left[1 - \exp\left(-\frac{0.005358151}{82.74}\tau\right)\right] \tag{5.5}$$

The actual number of failures (μ) at any given time (τ) appears to exhibit an s-shaped tendency. This is also true for the previous data sets A2 and A1. While this might lead to the conclusion that a model such as the Yamada-Ohba-Osaki Power model could be feasible, there is another possible interpretation. The shift in the curve could be due to additional software releases during the IOT&E time frame. Musa et al. present a method of adjusting failure times for evolving programs [64:440-448]; however, the limited scope of IOT&E should not require such adjusting, especially when the data are located within the confidence intervals.

The Logarithmic Poisson Execution Time model also fit well to the actual data on failures experienced (see Figure 5.12). The model was fitted with $\lambda_0 = 0.005505088$ and $\theta = 0.017004749$

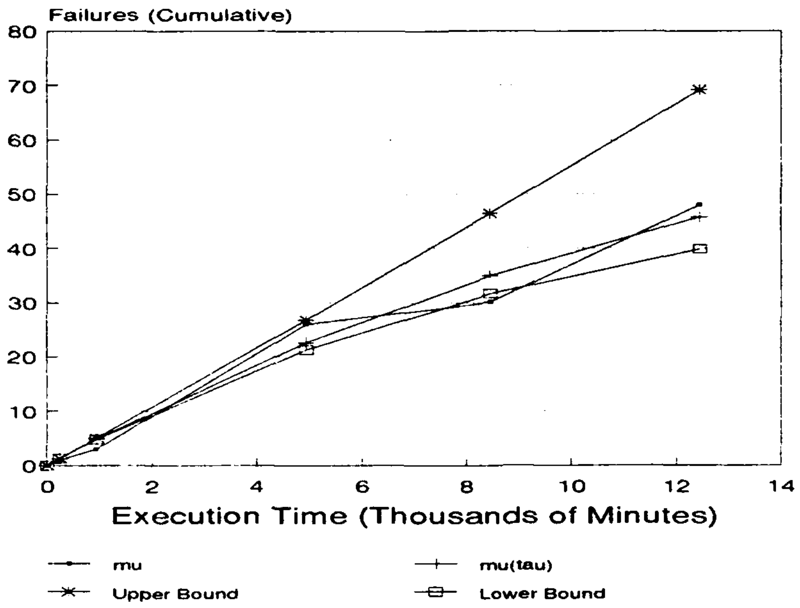

Figure 5.11. Expected Failures Using Execution Time Model for Data Set A3

as initial parameters to get the following equation for failures expected at time τ

$$\mu(\tau) = \frac{1}{0.017004749} \cdot \ln((0.005505088)(0.017004749)\tau + 1) \tag{5.6}$$

Any potential reasons for the minor deviations have been previously discussed for the data sets A1 and A2. Overall, the model appears to have a very good fit. Thus, we fail to reject the null hypothesis.

5.2.4 Data Set S1. The Execution Time model had a fitted form with initial parameters $\lambda_0 = 0.001173446$ and $\nu_0 = 417.03$ which gave the following form of the equation

$$\mu(\tau) = 417.03 \left[1 - \exp\left(-\frac{0.001173446}{417.03}\tau\right)\right] \tag{5.7}$$

Figure 5.13 shows the closeness of the curve to the actual data, and while the S1 data curve appears to be steeper than the estimated curve, the fit is still very close. One possible reason for the steepness of the curve and tightness of the 95 percent confidence intervals could be the assumption

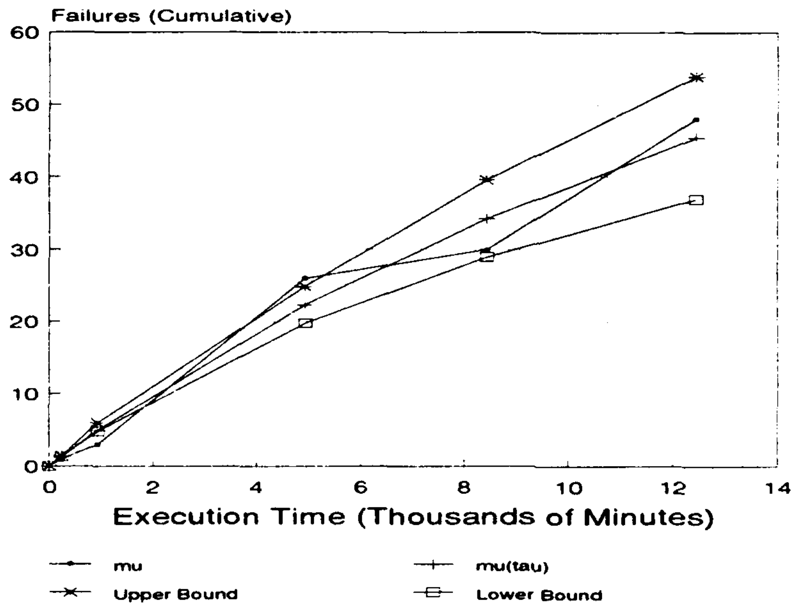

Figure 5.12. Expected Failures Using Logarithmic Poisson Execution Time Model for Data Set A3

of uniform test time (60 hours per day) throughout the entire month. It is possible that actual test times could flatten out the curve, resulting in a closer fit of the model. Even so, we fail to reject the null hypothesis due to the closeness of the data and actual curve.

With the exception of the steepness of the actual curve, the Logarithmic Poisson Execution Time model also fit well to the actual data (see Figure 5.14); however, there were enough data points outside the confidence intervals that we reject the null hypothesis. The Logarithmic Poisson Execution Time model was fitted with the initial parameters $\lambda_0 = 0.001770339$ and $\theta = 0.007616991$ to get the following form of the equation

$$\mu(\tau) = \frac{1}{0.007616991} \cdot \ln((0.001770339)(0.007616991)\tau + 1) \tag{5.8}$$

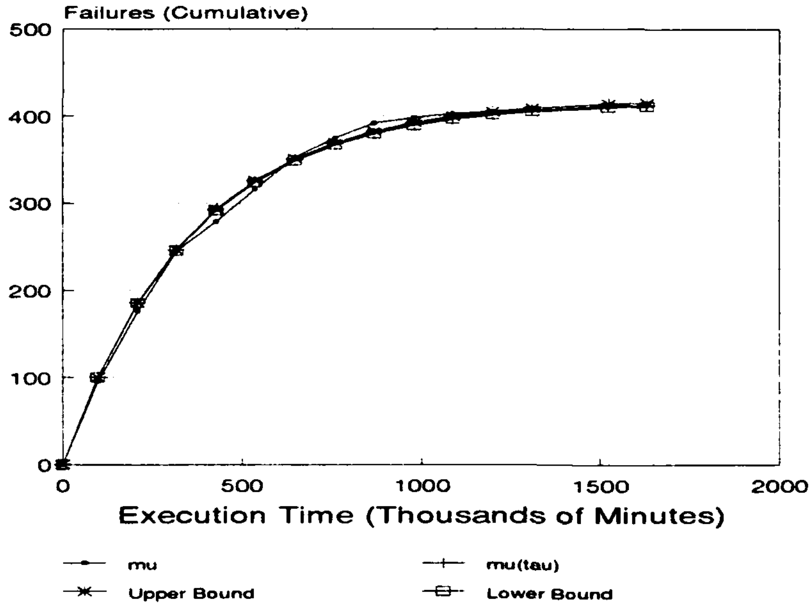

Figure 5.13. Expected Failures Using Execution Time Model for Data Set S1

5.2.5 Data Set W1. The fitted form of the Execution Time model had initial parameters $\lambda_0 = 0.001927052$ and $\nu_0 = -221.00$ which gave the following form of the equation

$$\mu(\tau) = -221.00 \left[1 - \exp\left(-\frac{0.001927052}{-221.00}\tau\right)\right] \tag{5.9}$$

This was by far the most interesting of the data sets to analyze. Figure 5.15 reveals an *increasing* failure rate. Musa et al. note that if both the initial parameters β_0 and β_1 are less than 0, the model will exhibit an increasing failure intensity [64:310]. Such an indication does not invalidate the model's application, since this model is of the exponential Poisson group which "can accommodate increasing and decreasing failure intensities," making sure that $\mu(t)$ and $\lambda(t)$ are both nonnegative [64:310].

The reason for this increasing failure intensity could be the operational tests were designed to exercise the easier parts of the system first, and then the more critical ones later. The rapid flattening towards the end of testing would then be indicative of a regression test where only one or two new failures are identified. Still, the Execution Time model does provide a fairly accurate mapping to the actual failure data for the last half of the test time. This concurs with other

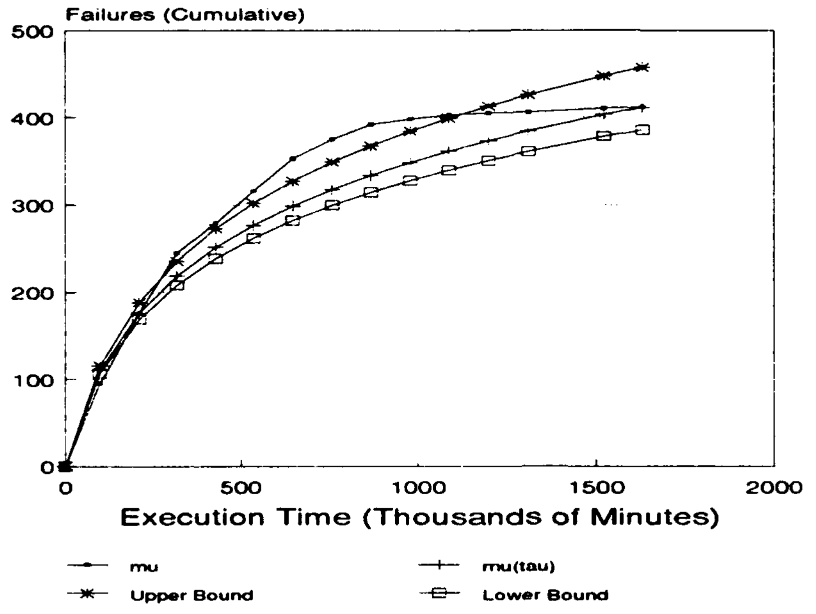

Figure 5.14. Expected Failures Using Logarithmic Poisson Execution Time Model for Data Set S1

observations of applications of this model [64, 89]. As the test for the null hypothesis is regardless of an increasing or decreasing failure intensity, we fail to reject the null hypothesis.

The results of the Logarithmic Poisson Execution Time model are a little more dramatic. As shown in Figure 5.16, the curve has a very steep incline and then a drastic flattening. This could be based on the fact that this data set has an increasing failure intensity, and although geometric Poisson group models can "accommodate decreasing and a certain type of increasing failure intensities," the initial model parameter β_1 still diverges [64:312]. Indeed, The Logarithmic Poisson Execution Time model was fitted with the initial parameters $\lambda_0 = 73.957034969$ (which indicates divergence in the Newton-Raphson estimation method) and $\theta = 0.046388798$, resulting in the following form of the equation

$$\mu(\tau) = \frac{1}{0.046388798} \cdot \ln((73.957034969)(0.046388798)\tau + 1) \qquad (5.10)$$

The level of initial parameter divergence appears to affect the slope of the curve in a proportional way. One possible way to reduce the steep slope is to test the more failure-likely areas

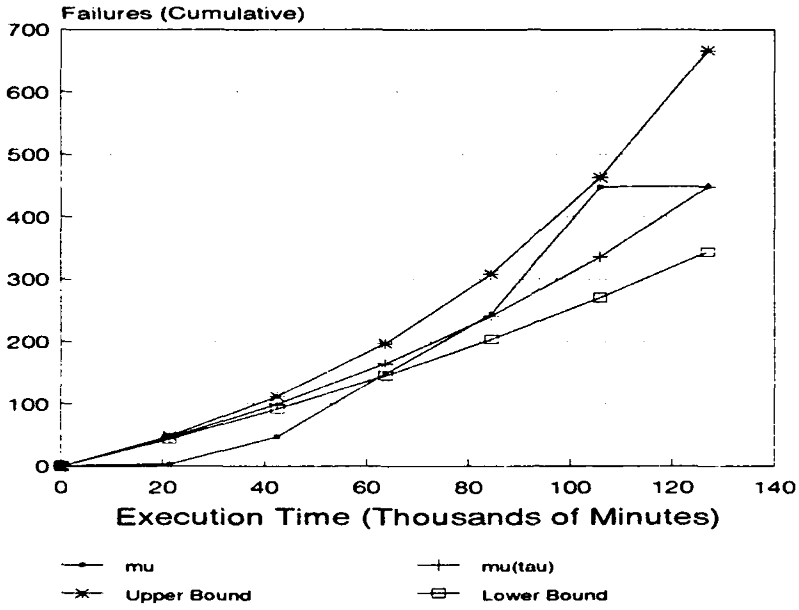

Figure 5.15. Expected Failures Using Execution Time Model for Data Set W1

first, before checking the least-likely failure areas of the software. With the calculated data clearly differing from the actual data, we reject the null hypothesis.

5.3 Assessment of Failure Intensity Values

The previous two assessments established the models' feasibility with respect to the initial data, as well as the "fit" of the model based on parameter derivation. This section addresses the failure intensity calculations of both models.

The initial failure intensity (λ_0) and final failure intensities for each data set are shown in Table 5-1. The final failure intensity values are listed for both time ($\lambda(\tau)_f$ from Equations 2.9 and 2.12) and failures experienced ($\lambda(\mu)_f$ from Equations 2.10 and 2.13). The values for data set W1 are very much skewed based on the increasing failure intensity characteristic of the data, and provide no insight into any relationship between the failure intensities. Data set A1 does not cover its final IOT&E testing time. Therefore, the final failure intensities can not be used to determine any operational reliability; however, it is interesting to note the closeness of values between the two different models' final failure intensity calculations. While there is considerable disagreement

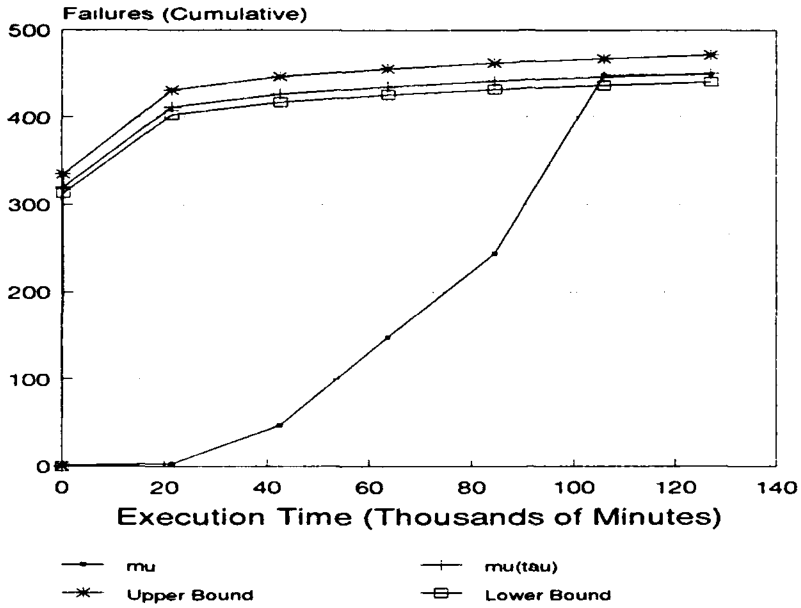

Figure 5.16. Expected Failures Using Logarithmic Poisson Execution Time Model for Data Set W1

between the Execution Time model and the Logarithmic Poisson Execution Time model concerning the final failure intensities for the test period, the basis of calculation (time vs. experienced failures) does not seem to impact the specific calculations for each model.

Similarly, data sets A2 and A3 have final failure intensity values for each model that are relatively close to each other regardless of calculation basis (i.e.–$\lambda(\tau)_f \approx \lambda(\mu)_f$). There is a substantial difference between the two candidate models for each data set. Within each data set the models yield close results regardless of the input parameter (time or failures).

Data set S1 seems to exhibit a more ideal failure intensity trend. The values for each model are almost identical regardless of the input parameter (time or failures) and appear to decrease to a more favorable level. Taking one of the final failure intensities, such as $\lambda(\tau)_f = 0.000011327$ for the Execution Time model, the operating assumption can be extended to

(60 hours per day)(365 days per year) = 21900 hours of operations per year

Table 5.1. Comparison of Software Reliability Failure Intensities

Data Set and Model	**Initial Failure Intensity λ_0**	**Final Failure Intensity $\lambda(\tau)_f$**	**Final Failure Intensity $\lambda(\mu)_f$**
A1 (Exec)	0.162871611	0.010574044	0.010573845
A1 (Log)	0.322609809	0.018310707	0.018310712
A2 (Exec)	0.003096631	0.001138657	0.001109443
A2 (Log)	0.003267847	0.001259321	0.001239492
A3 (Exec)	0.005358151	0.002120178	0.002120206
A3 (Log)	0.005505088	0.002352398	0.002352398
S1 (Exec)	0.001173446	0.000011327	0.000011340
S1 (Log)	0.001770339	0.000076181	0.000076181
W1 (Exec)	0.001927052	0.005850893	0.005850914
W1 (Log)	73.957034969	0.000169250	0.000000064

which can then be applied to Equation 4.1 to give a reliability assessment of

$$\begin{aligned} R(21900) &= e^{-(0.000011327)(21900)} \\ &= 0.780312109 \end{aligned}$$

5.4 *Calculated Values for Current Number of Failures (Based on DT&E Data) Compared to Actual Number of Failures*

A fourth model feasibility assessment was made of the candidate models based on the available DT&E data. Parameter estimates were obtained and fitted models were derived for DT&E, from which the final failure intensity values were determined. These values then served as initial inputs to the models, and another evaluation similar to the second assessment was conducted. The same null hypothesis criteria and goals apply, only the data set has been expanded to provide more realistic values of the initial parameters. Only data sets A2, A3 and S1 had identifiable DT&E failures as well as some measure of test durations for the DT&E timeframe. The results are given below, and shown in Figures 5.17 through 5.22.

5.4.1 Data Set A2. In order to determine the final DT&E failure intensities, the assumption was made that DT&E had the same test times per month as IOT&E (50.58 hrs). The final DT&E failure intensities for both $\lambda(\tau)$ and $\lambda(\mu)$ of the Execution Time model were identical, providing the IOT&E initial parameter $\lambda_0 = 0.001687372$. From this, the fitted model was derived as

$$\mu(\tau) = -125.65\left[1 - \exp\left(-\frac{0.001687372}{-125.65}\tau\right)\right] \tag{5.11}$$

The data were, for the most part, within the 95 percent confidence intervals (see Figure 5.17). The interesting shape of this curve could be due to the initial λ_0 value derived from the DT&E data. The resulting negative value for μ_0 is an indication of an increasing failure intensity. Since the first two assessments demonstrated data set A2 as having a decreasing failure intensity, the only conclusion is the curve is affected by the initial λ_0 parameter. This, in turn, could be a function of the assumptions used to determine the test times for the DT&E assessment. Thus, while there was a good fit of the model to the data, the shape of the curve makes the initial parameters suspect; however, we still fail to reject the null hypothesis based on the coverage the model provided.

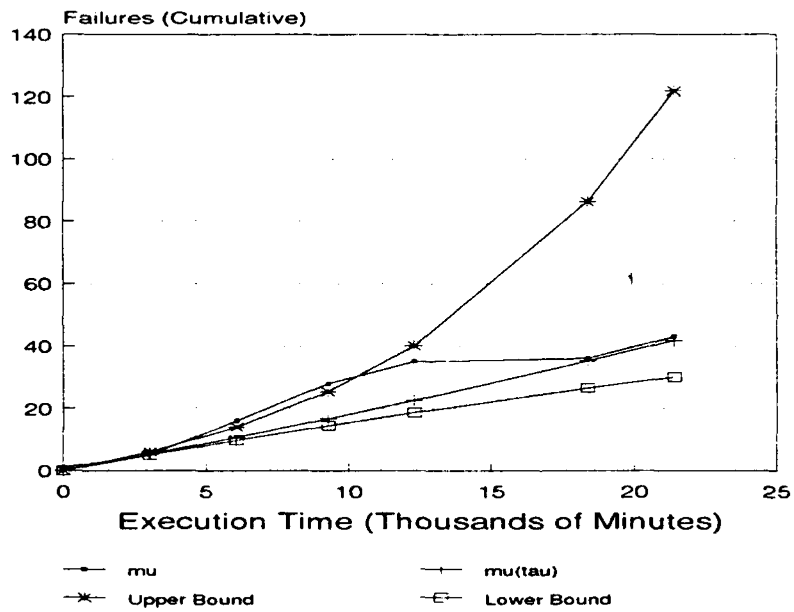

Figure 5.17. Expected Failures Using Execution Time Model for Data Set A2

The Logarithmic Poisson Execution Time model also exhibited an increasing failure intensity trend (see Figure 5.18). The initial DT&E failure intensity estimate was $\lambda_0 = 15.504426266$, indicating divergence. Therefore, the model was not able to calculate a final value of either $\lambda(\tau)$ or $\lambda(\mu)$ for DT&E. Instead, the IOT&E model was fitted with same initial failure intensity as the Execution Time model: $\lambda_0 = 0.001687372$. The corresponding $\theta = -0.007139135$ was derived, and the equation for failures expected at time τ was

$$\mu(\tau) = \frac{1}{-0.007139135} \cdot \ln((0.001687372)(-0.007139135)\tau + 1) \tag{5.12}$$

Again, the same factors that affected the Execution Time model could also have affected the Logarithmic Poisson Execution Time model, especially since both models used the same initial λ_0 parameter; however, in this case the model does not fit the data, and we reject the null hypothesis. Accurate values for test times and failure times of occurrence could indicate a much closer fit of model and data.

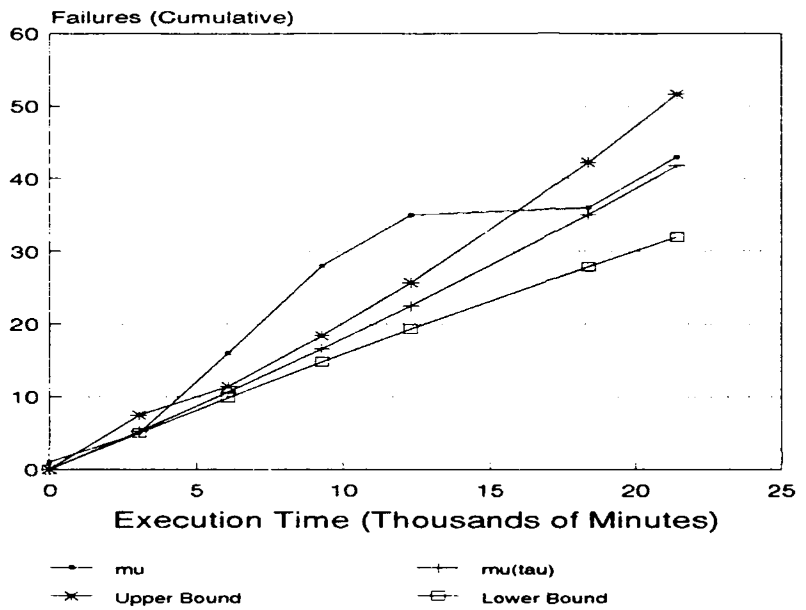

Figure 5.18. Expected Failures Using Logarithmic Poisson Execution Time Model for Data Set A2

5.4.2 Data Set A3. The DT&E version of the fitted model had $\lambda_0 = 0.002136669$, and the resultant fitted form of the Execution Time model for IOT&E data had initial parameters $\lambda_0 = 0.005729161$ and $\nu_0 = 75.42$. This gave the following equation for failures expected at time τ

$$\mu(\tau) = 75.42 \left[1 - \exp\left(-\frac{0.005729161}{75.42}\tau\right)\right] \tag{5.13}$$

Data set A3 had a closer fit of the Execution Time model to data than did data set A2 (see Figure 5.19). The results were very similar to those shown in Figure 5.11. This closeness could be due to a closer approximation of DT&E final failure intensity values based on a better test time approximation (even though the time used was an average). Therefore, the model maps well to the failure data, and we fail to reject the null hypothesis.

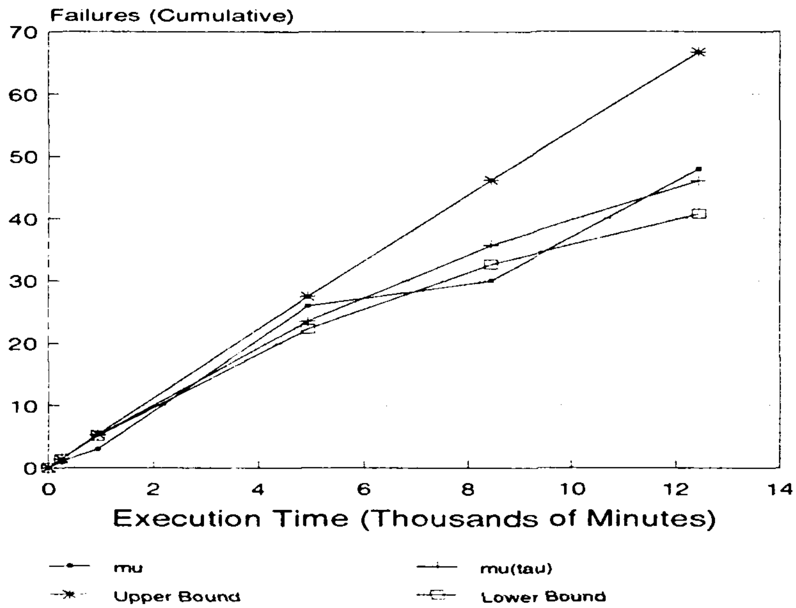

Figure 5.19. Expected Failures Using Execution Time Model for Data Set A3

The Logarithmic Poisson Execution Time model was able to calculate a final DT&E λ value for both time and failures experienced. Both numbers were identical, with a value of 0.000173741; however, when this number was used as the initial parameter estimate for the IOT&E data, the software encountered a math overflow due to the ratio of the small initial value compared to the IOT&E data set. Thus, the IOT&E initial failure intensity parameter was taken from DT&E final failure intensity calculations for the Execution Time model. The initial parameter was then $\lambda_0 = 0.005729161$, from which $\theta = 0.018397759$ was calculated. This gave the following equation for failures expected at time τ

$$\mu(\tau) = \frac{1}{0.018397759} \cdot \ln((0.005729161)(0.018397759)\tau + 1) \qquad (5.14)$$

The results are shown in Figure 5.20, and appear to be identical to the second assessment (see Figure 5.12). The model fit is sufficiently close that we fail to reject the null hypothesis. The closeness of both the Execution Time model and the Logarithmic Poisson Execution Time model indicate that using DT&E data to derive the initial parameters could be a feasible method.

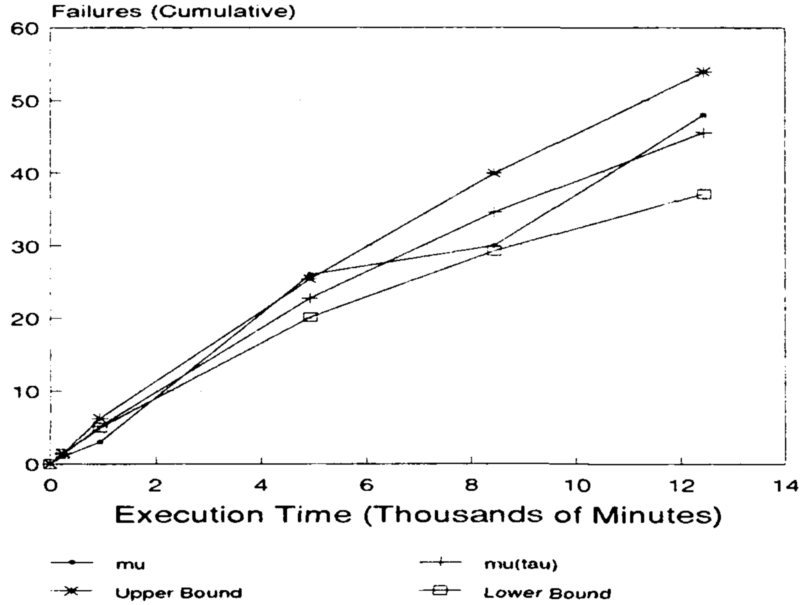

Figure 5.20. Expected Failures Using Logarithmic Poisson Execution Time Model for Data Set A3

5.4.3 Data Set S1. The Execution Time model used the DT&E final failure intensity value $\lambda_0 = 0.001850621$ as the initial parameter to calculate $\nu_0 = 413.26$ and give the following form of the equation

$$\mu(\tau) = 413.26 \left[1 - \exp\left(-\frac{0.001850621}{413.26} \tau \right) \right] \tag{5.15}$$

In contrast to Figure 5.13, Figure 5.21 shows the data lagging behind the model throughout the entire IOT&E period. The uniformity of the models curve and closeness of the estimated and 95 percent confidence values could be due to the assumption of uniform test time during each month of testing. As with the second assessment, it is possible that actual test times could flatten out the curve, resulting in a closer fit of the model; however, as it stands now there is no fit between the model and the data, and we reject the null hypothesis.

As the initial failure intensity calculation for the DT&E version of the Logarithmic Poisson Execution Time model diverged, the IOT&E version was fitted with the initial parameter $\lambda_0 =$

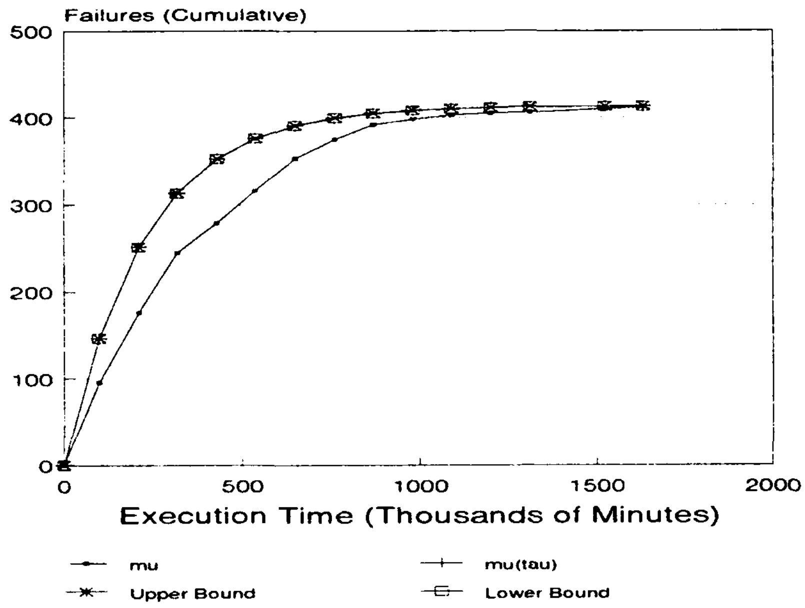

Figure 5.21. Expected Failures Using Execution Time Model for Data Set S1

0.001850621, from which $\theta = 0.007764353$ was derived to get the following form of the equation

$$\mu(\tau) = \frac{1}{0.007764353} \cdot \ln((0.001850621)(0.007764353)\tau + 1) \quad (5.16)$$

The Logarithmic Poisson Execution Time model had a closer fit to the data then did the Execution Time model (see Figure 5.22). This trend is very similar to the one found during the second assessment (see Figure 5.14). While the model does somewhat approximate the actual data, there is a sufficient number of data points outside the 95 percent confidence interval to reject the null hypothesis.

5.5 *Summary*

This chapter presented an initial assessment of the applicability of each candidate model to the available data sets. Next, the results of the fitted models were discussed, with a comparison made between actual and estimated failures. The failure intensity values [illegible] ves were assessed for any sort of trend data. Finally, for those sets with sufficient failure and time data, the models

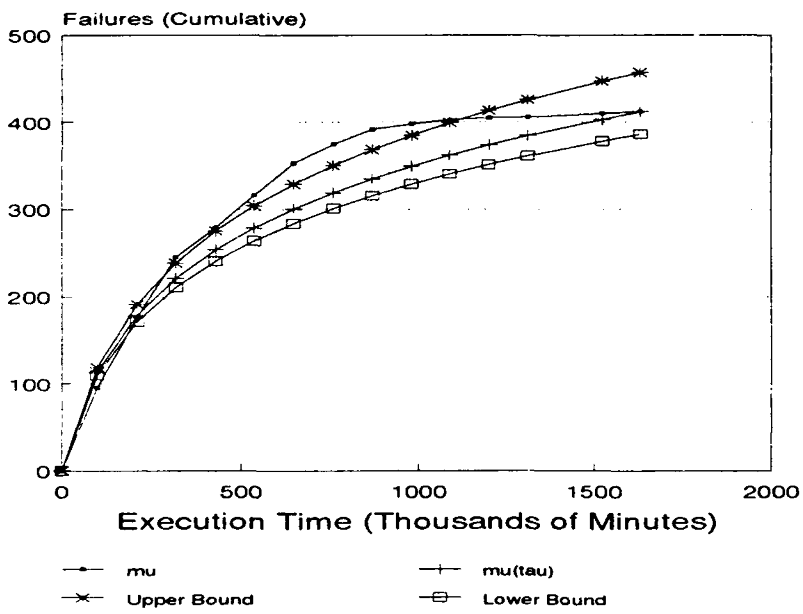

Figure 5.22. Expected Failures Using Logarithmic Poisson Execution Time Model for Data Set S1

were run on DT&E data to determine the initial parameters for IOT&E, and a fourth assessment was performed to see if the models were affected by previously existing failure data. The next chapter contains conclusions and recommendations concerning these evaluations.

VI. Conclusions and Recommendations

Current operational test and evaluation of weapon system software by HQ AFOTEC primarily emphasizes the operational suitability of the software. There is no current measure of the operational effectiveness of the software. In order to provide some assessment of a weapons system's software, this thesis proposed that a software reliability model could provide the needed level of operational effectiveness assessment.

Only existing software reliability models were considered–no new models were proposed. A hierarchy of software reliability models was defined, with emphasis on product vs. process models. Within this overall grouping, four categories of software reliability models were identified:

- Fault seeding
- Input domain
- Times-between-failures
- Failure count

Software reliability model evaluation criteria were established that included:

- Predictive validity
- Capability
- Quality of Assumptions
- Applicability to the Finite-Time Environment
- Diversity and Applicability of Output
- Capability to Use Existing Data

Potential software reliability models from the four categories were evaluated against these criteria. A final selection was made of two candidate models: the Musa Execution Time model, and the Musa-Okumoto Logarithmic Poisson Execution Time model.

Implementation of the candidate models was performed, and five test data sets were run to assess the models' fit and applicability. Analysis was conducted both on the initial test data sets and calculated values for number of failures and confidence intervals. An analysis was also performed on the calculated failure intensity values. Finally, three test data sets were run on historical DT&E data to determine initial parameter estimates, which were then used for OT&E assessment of the models' fit and applicability.

6.1 Conclusions

The summary results of the null hypothesis test for each candidate model are shown in Table 6.1. There is a generally good mapping of the Execution Time model to the actual failure data, while the Logarithmic Poisson Execution Time model did not map as well. The deviations outside the 95 percent confidence intervals could be attributed to the manner in which unknown time parameters were estimated for the failure data. While the data (failure and times) were not exactly accurate and complete on all accounts, this variation did give a chance to evaluate both candidate models' robustness with respect to missing or incomplete data. With both models, there was sufficient parameter estimation available to compensate for the lack of exact failure and time data; however, the lack of data appears to have a significant impact on the Logarithmic Poisson Execution Time model.

Table 6.1. Summary Analysis of H_0 Test

Data Set	Musa Execution Time Model	Musa-Okumoto Log Model
A1	Reject	Reject
A2	Fail to Reject	Reject
A3	Fail to Reject	Fail to Reject
S1	Fail to Reject	Reject
W1	Fail to Reject	Reject

There is nothing definitive that can be concluded from the comparison of failure intensity values. Possibly, after gathering enough information from different weapon systems, it might be possible to identify a trend in reduction of the failure intensities from start of IOT&E to end of IOT&E, or it might be possible to identify target values for final failure intensity based solely on the category and type of weapon system (e.g.-fighter aircraft could have the same operational profile, and, therefore, fly roughly the same number of hours per sortie or per year). Another potential application is in determining release time for the software; however, that requires prediction of the software's reliability, and is left to future research for validation.

The two previous analyses were preliminary, and led to the final assessment of using DT&E data as the basis for parameter estimation, which was then used with the models on IOT&E data. The results of this assessment are shown in Table 6.2. Again, the Execution Time model appears to perform better than the Logarithmic Poisson Execution Time model; however, on data set A3 where the execution time data was more accurate, both models performed well. This could be due to the use of the Execution Time model DT&E final failure intensity value $\lambda(\tau)_f$ as the

Logarithmic Poisson Execution Time model IOT&E initial failure intensity parameter λ_0. Another possible explanation is the execution time data available being more complete than time data for the other data sets. A combination of the two is also possible. The closeness of the fit does indicate the merit of using DT&E maturity data as the basis for parameter estimation of the models for IOT&E reliability measurement; however, additional analysis with complete test data is necessary to state this conclusively.

Table 6.2. Summary Analysis of H_0 Test for Data Sets With DT&E Based Initial Parameters

Data Set	Musa Execution Time Model	Musa-Okumoto Log Model
A2	Fail to Reject	Reject
A3	Fail to Reject	Fail to Reject
S1	Reject	Reject

An extra evaluation criterion discussed by Mr Siefert in the recent *American Society for Quality Control 1st International Conference on Software Quality* was a more subjective assessment of a software reliability model, namely "is it good" [79]. The candidate models presented in this thesis exhibit a definite "goodness" about them, which stems from their straightforward implementation as well as capability to use existing initial failure intensity data or derive this information from system characteristics. These capabilities were not found in any of the other models.

6.2 *Recommendations*

There are three other aspects of IOT&E software reliability models that should be investigated: data needed for software reliability evaluation; additional analysis of the candidate models; and applicability of software reliability. These are described in the following sections.

6.2.1 Data Needed for Software Reliability Evaluation. The most important aspect of software reliability models that appeared throughout the literature was that of collecting enough accurate and complete data. Unfortunately, the data sets used for this study were not that accurate nor complete. The AFOTECP 800-2 Vol 6, *Software Maturity Evaluation Guide*, does include a field for total operating time in minutes, which is the time of failure from the very beginning of IOT&E [23, 46]. While such a measure is good to have (time of failure is needed), multiple testing that can occur with weapon systems such as aircraft require a simpler approach to collecting test and failure times. One way to simplify this is to require tracking test duration (or test start and stop times), as well as *local* time of failure (failure time with respect to that test, e.g.-failure 1 occurs

Table 6.3. Proposed Software Maturity Data

Description	Variable Name	Format
Software Problem Number	PROB_NUM	Character 10
Software Configuration Item	CPCI	Character 10
Severity of Problem	SEV_CODE	Character 1
Date Problem Discovered	DATE	Date
Date Problem Fixed	DATE	Date
Description of Problem	TITLE	Character 42
Test Identification Number	TEST_ID	Character 10
Date Test Planned	TESTPLAN	Date
Date Test Completed	TESTCOMP	Date
Start Time (minutes)	START_TIME	Character 10
Finish (End) Time (minutes)	END_TIME	Character 10
Time of Failure Occurrence	TIME_OCCUR	Character 10

at +2.00 minutes). The software model implementation can then calculate cumulative test times, cumulative failure times, and any other needed statistics.

In general, the data necessary to applying software reliability models to IOT&E would include the current software maturity fields, with the exception of replacing the one field for total operating time with the specific time fields described above (see Table 6.3).

In support of data persistence, an object-oriented database (OODB) should be implemented; however, due to the newness and complexity of OODBs, a transitional approach is acceptable where the database is described by an object-based semantic data model (SDM) and then transformed into an entity-relationship diagram (ERD) for implementation at the physical level. This implementation can then be carried out with an existing relational database model, such as the one used by Clipper, with virtually no loss to the data meaning or relationships.

The complete description of these models and their interrelationships is given in Appendix E, along with the SDM description for aircraft reliability data. This description can easily be expanded into a superclass that would include aircraft, radar, missiles, and any other categories of weapon systems. The SDM description includes not only the failure data needed for the weapon system, but also the data that will be calculated by the software reliability models. From this, the entity-relationship (E-R) diagram shown in Figure 6.1 was derived. This diagram would then be the basis for implementing the relational model to track software reliability.

6.2.2 Additional Analysis of the Candidate Models. Additional analysis of the candidate models is needed in the following areas: additional different weapon systems; use of system char-

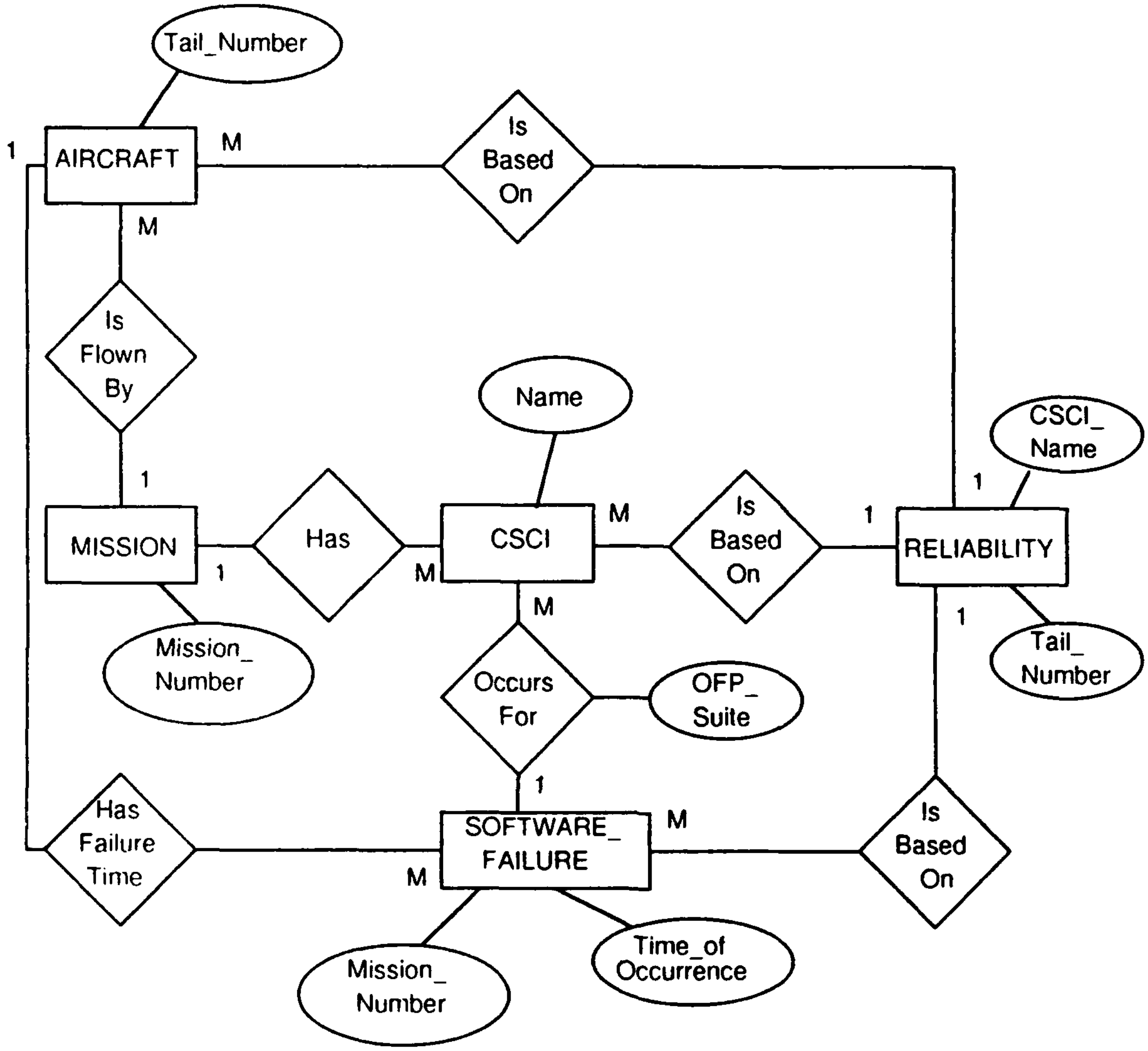

Figure 6.1. E-R Diagram for Software Reliability Database

acteristics to determine initial parameters; evaluation of model adequacy based on goodness-of-fit tests; impact of failure classification and weighting; sources of additional test time.

Additional Different Weapon Systems. First, as sufficient data is accumulated on different weapon systems, the same tests performed in this thesis should be applied to see if there is agreement on the results.

Use of System Characteristics to Determine Initial Parameters. Next, the capability to use system characteristics instead of failure data to determine initial parameter values should also be done and compared to the results of the other tests. If there is a high correlation between the three model implementations (using parameters determined from actual IOT&E failure data, parameters determined from previous DT&E failure data, and parameters derived from system characteristics), then the models should be implemented for all IOT&E test teams. The viability of the Musa-Okumoto Logarithmic Poisson Execution Time model has already been established by an American Institute of Astronautics and Aeronautics (AIAA) independent study. The study was conducted by a special "Blue Ribbon Panel" consisting of such software reliability professionals as Dr Farr, Dr Hecht, Mr Musa, Dr Shooman, Mr Siefert, and others. The AIAA panel identified the Musa-Okumoto Logarithmic Poisson Execution Time model as the best software reliability model in the time domain category, non-exponential class [80:186].

Evaluation of Model Adequacy Based on Goodness-of-Fit Tests. Finally, as data on failure counts per time interval becomes more thorough, it will be possible to group the failure data by number of sample observations. Trends could emerge that would provide an indication of the expected number of observations for time intervals throughout IOT&E. This would then allow χ^2 and other goodness-of-fit tests to be used to test the candidate models' adequacy [95].

Impact of Failure Classification and Weighting. This study did not progress to the point of analyzing the individual categories of software failures. Research should continue in that direction to see if there is some relationship between the severity of the failure and the cumulative test time. Also, potential acceptability thresholds could be established that allow some categories of failures while requiring others to be corrected prior to the end of IOT&E.

Sources of Additional Test Time. Should the Test Director begin to run out of test time before reaching his/her desired failure intensity, alternative methods of testing might increase the test time. For example, Adolph and Montgomery identified the Integration Facility for Avionics System Test (IFAST), which was essentially a hot-mock-up of some of the aircraft undergoing test and evaluation at Edwards AFB, CA [1]. The use of the IFAST facility provided additional test time, without requiring additional sorties from the aircraft and crew. An installation such as this would be included as a multiple installation, and the additional test time could help reduce the failure intensity without creating additional operational test costs. Methods of including such additional test time and data should be considered for integration into the model databases.

6.2.3 Applicability of Software Reliability. The candidate software reliability models can potentially be used together with system capability assessments, combined with hardware reliability models, applied to theoretical hardware designs, integrated with other software reliability models, or applied to cost estimation.

Software System Effectiveness. Software reliability provides one way of measuring the operational effectiveness of the weapon system software; however, a measure of the impact of software reliability on the total weapon system effectiveness could be determined as follows. The ratio of software up-time to total software "mission" or test time would be determined, and this value would be the Software System Effectiveness (SSE) [8]. This number would then be multiplied against the desired Mission Capable (MC) rate of the overall weapon system, giving the Total Weapon System Effectiveness (TWSE) [8]. This result is actually an adjusted MC rate that takes into account the current effectiveness of the software. In support of this, software failure data that indicates mean-time-to-recover software (MTTRS) should also be collected, and could be included as an additional field of either UP_TIME (time the software was available during the test) or DOWN_TIME (time the software was not available during the test).

Combined Hardware and Software Reliability Models. The concept of SSE was somewhat suggested in [42] as part of a combined hardware/software reliability model. A combined model must consider such "random phenomena" as the "software 'repair' process" where the system is restored "to an operational state without correcting the software fault" [42:1-1]. Therefore, even if a combined model is not available in the near future, MTTRS and SSE data should be collected and calculated now to provide both an initial assessment of mission capability and also provide a historical database for a future integrated hardware/software reliability model.

Applicability to Hardware Design Reliability. With the growing use of hardware modeling techniques such as VHDL (Very High-Speed Integrated Circuit (VHSIC) Hardware Description Language), the possibility exists that software reliability measurement (with its focus on the ***design*** as opposed to the ***physical*** aspect) might one day be necessarily applied to hardware designs that exist only in the memory of a computer. Toward this end, software reliability models for IOT&E could provide the foundation for determining the IOT&E *logical* design reliability of hardware from components to systems.

Integrated Software Reliability Tools. One of the current trends in software reliability is to have many different reliability models integrated into one tool. Many different tools are being identified to perform software reliability prediction, measurement, and analysis, and it is possible that not all software reliability models are applicable to all phases of the software life-cycle. Indeed, it may be possible or even desirable to implement a different software reliability model during each phase of the software life-cycle [69]. This would require standardization of data to be used between models. By having many different models in one tool, the software evaluator in the field can become overburdened with understanding the intricacies of each model and when they apply, as well as possibly collecting data that could vary from one model to the next. An example of this is the SMERFS tool, which has two different sets of models selectable from the main menu, and requires different types of data for each set [29]. Clearly, having one standardized model (or at least set) with one basic database will make software reliability evaluation easier for the software evaluator in the field, as well as making the data collection job easier for the data point of contact at HQ AFOTEC.

Cost Estimation. This thesis proposes using the candidate models to determine the time needed to reach a desired failure intensity objective given a current failure intensity value. A recent paper ties this to actual testing cost [90]. The paper demonstrated that, due to the dependency of testing costs on software failure behavior, a quantitative cost model can be incorporated with the Logarithmic Poisson Execution Time model to determine marginal costs [90:423-424]. Additional research into the area of combining software cost models and software reliability models could then provide a more useful tool to both engineer and manager.

6.3 Summary

This evaluation reached important conclusions about the application of software reliability to IOT&E of weapon systems. It is clear that candidate models exist which can work with some degree of certainty in evaluating the software reliability, and hence, the operational effectiveness of weapon system software. The applicability of these models extends far beyond the IOT&E of software, and as the software evaluation process matures a better understanding and assessment of both software and the overall weapon system will be gained. To what ever extent software reliability is pursued, the fact that it is being considered is just one step closer to obtaining "good code" for the user.

Appendix A. *Software Definitions*

The following definitions were taken from multiple sources, and are included here as additional information.

Error. Human action that results in software containing a fault. Examples include: omission or misinterpretation of user requirements in a software specification, and incorrect translation or omission of a requirement in the design specification.

Fault. A manifestation of an error in software. A fault, if encountered, may cause a failure. Synonym - Bug.

Failure. The inability of a system or system component to perform a required function within specified limits. A failure may be produced when a fault is encountered.

Failure Intensity. The number of failures per unit time. Failure intensity can be identified for average number of software failures per flight hour (SF/FH) and average number of software failures per mission (SF/M).

Maintainability. The ease with which software can be maintained. The extent to which a component facilitates updating to satisfy new requirements or to correct deficiencies.

Maturity. The extent to which a component has been used in the development of deliverable software by typical users and to which feedback from that use has been reflected in modifications to the component.

Mean Time to Recover Software. The amount of time required to recover from a software failure and restore operational capability of the software. This could be the time necessary to "reboot" the system, or the amount of time spent by an operator clearing an error display and selecting an alternate menu option.

Model. A representation of a real world process, device, or concept.

Requirement. A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formally imposed document.

Software Maintenance. Modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a changed environment.

Software Maturity. An assessment of the software based on the number of *faults* in a computer program. This includes known and undiscovered (latent) faults. Latent faults might not be discovered until several years after full scale production, if at all. Emphasis here is on *development activities.* A measure of the software's progress in its evolution toward satisfaction of all documented user requirements.

Software Reliability. The probability of *failure*-free operation of a computer program for a specified period of time. The emphasis here is on *operational activities.* If the software fails, then there could be faults that must be corrected; however, not all faults result in failures. Software Reliability Evaluation can be divided into three distinct parts:

- **Measurement.** Software reliability measurement determines the present failure intensity, additional failures that would be experienced before reaching an identified failure intensity objective, and additional execution time necessary to reach an identified failure intensity objective.
- **Prediction.** Software reliability prediction attempts to determine what the reliability of software will be at some time t from a present software reliability measurement. •
- **Threshold.** The level of software reliability identified or desired by the decision maker. This can be expressed as a reliability number (which can be translated with respect to execution time) or a failure intensity threshold or objective.

Software System Effectiveness. A measure of the percentage of time the software system operates correctly (no failures) versus the total attempted operational time. The SSE can be multiplied by the Mission Capable (MC) rate to give the effect of software on Total Weapon System Effectiveness (TWSE).

Total Weapon System Effectiveness. The Mission Capable (MC) rate for a weapon system adjusted to account for the effectiveness of the software. The Software System Effectiveness (SSE) can be multiplied by the MC rate to determine the TWSE.

Appendix B. *Software Maturity Data*

This appendix contains the reduced data set from the initial software maturity data provided by HQ AFOTEC/LG5.

B.1 Data Set A1

Database for AIRCRFT1

Date	# 1	# 2	# 3	# 4	# 5	NSC	Total	Cum Total
10/01/86	0	0	0	0	6	0	6	6
08/27/87	1	1	0	0	2	0	4	10
08/28/87	0	2	0	0	29	0	31	41
08/29/87	0	1	0	0	37	0	38	79
08/31/87	1	0	0	0	16	0	17	96
09/02/87	0	2	0	0	25	0	27	123
09/03/87	0	0	0	0	49	0	49	172
09/04/87	0	0	0	0	58	0	58	230
09/05/87	0	2	0	1	38	0	41	271
09/08/87	0	7	0	0	69	0	76	347
09/09/87	0	0	0	0	28	0	28	375
09/10/87	0	1	0	0	24	0	25	400
09/11/87	0	0	0	0	27	0	27	427
09/12/87	0	0	0	0	32	0	32	459
09/14/87	0	10	0	0	42	0	52	511
09/15/87	0	0	0	0	16	0	16	527
09/16/87	0	2	0	0	20	0	22	549
09/17/87	0	0	0	0	27	0	27	576
09/18/87	0	0	0	0	10	0	10	586
09/19/87	0	1	0	0	50	0	51	637
09/21/87	0	2	0	0	37	0	39	676
09/22/87	0	1	0	0	35	0	36	712
09/23/87	0	0	0	0	27	0	27	739
09/24/87	0	3	0	0	29	0	32	771
09/25/87	0	1	0	0	58	0	59	830
09/26/87	0	0	0	0	41	0	41	871
09/28/87	0	1	0	0	30	0	31	902
10/19/87	0	0	0	0	14	0	14	916
10/20/87	0	0	0	0	71	0	71	987
10/21/87	0	0	0	0	30	0	30	1017
10/22/87	0	0	0	0	18	0	18	1035
10/23/87	0	0	0	0	81	0	81	1116
10/24/87	0	0	0	0	9	0	9	1125
11/09/87	0	0	0	0	13	0	13	1138
03/05/88	0	0	1	0	0	0	1	1139
03/08/88	0	0	1	0	21	0	22	1161
03/09/88	0	0	1	0	21	0	22	1183
03/10/88	0	0	3	0	8	0	11	1194
03/11/88	0	0	1	0	5	0	6	1200
03/12/88	0	0	2	0	31	0	33	1233
03/14/88	0	0	4	0	17	0	21	1254
03/15/88	0	0	0	0	5	0	5	1259
03/16/88	0	0	1	0	22	0	23	1282
03/17/88	0	0	2	0	13	0	15	1297
03/18/88	0	0	0	0	13	0	13	1310
03/19/88	0	0	0	0	9	0	9	1319
03/21/88	0	0	1	0	14	0	15	1334
03/22/88	0	0	2	0	13	0	15	1349
03/23/88	0	0	0	0	5	0	5	1354
03/24/88	0	0	0	0	6	0	6	1360
03/25/88	0	0	0	0	6	0	6	1366
03/28/88	0	1	1	0	36	0	38	1404
03/29/88	0	0	2	0	28	0	30	1434
03/30/88	0	0	2	0	34	0	36	1470

04/26/88	0	0	0	0	2	0	2	1472
04/27/88	0	0	1	0	15	0	16	1488
04/28/88	0	0	0	0	18	0	18	1506
04/29/88	0	0	1	0	11	0	12	1518
04/30/88	0	0	0	0	11	0	11	1529

B.2 Data Set A2

Database for AIRCRFT2

Date	# 1	# 2	# 3	# 4	# 5	NSC	Total	Cum Total
02/24/87	0	0	1	0	0	0	1	1
03/04/87	0	1	0	0	0	0	1	2
05/20/87	0	0	0	1	0	0	1	3
06/10/87	0	0	3	0	0	0	3	6
06/25/87	0	0	1	0	0	0	1	7
07/01/87	0	0	1	0	0	0	1	8
08/03/87	0	0	3	0	0	0	3	11
09/01/87	0	0	1	0	0	0	1	12
09/03/87	0	0	0	1	0	0	1	13
09/11/87	0	1	1	0	0	0	2	15
09/17/87	0	1	0	0	0	0	1	16
09/21/87	0	0	3	0	0	0	3	19
11/04/87	0	0	2	0	0	0	2	21
11/05/87	0	0	0	1	0	0	1	22
11/12/87	0	0	2	0	0	0	2	24
11/19/87	0	0	1	0	0	0	1	25
11/20/87	0	1	2	0	0	0	3	28
12/08/87	0	0	0	1	0	0	1	29
12/09/87	0	0	2	0	0	0	2	31
12/14/87	0	0	1	1	0	0	2	33
01/04/88	0	0	2	0	0	0	2	35
01/05/88	0	0	2	0	0	0	2	37
01/14/88	0	1	0	0	0	0	1	38
02/01/88	0	0	1	0	0	0	1	39
02/23/88	0	0	2	0	0	0	2	41
03/04/88	0	0	1	0	0	0	1	42
03/10/88	0	0	1	0	0	0	1	43
03/16/88	0	0	0	1	0	0	1	44
03/17/88	0	0	1	0	0	0	1	45
03/29/88	0	0	1	0	0	0	1	46
04/12/88	0	0	1	0	0	0	1	47
04/28/88	0	0	1	0	0	0	1	48
05/03/88	0	0	1	0	0	0	1	49
05/04/88	0	0	2	1	0	0	3	52
05/06/88	0	0	3	0	0	0	3	55
05/10/88	0	0	1	0	0	0	1	56
05/12/88	0	0	1	0	0	0	1	57
05/13/88	0	0	1	0	0	0	1	58
05/16/88	0	1	0	0	0	0	1	59
05/20/88	0	0	1	0	0	0	1	60
05/23/88	0	0	1	0	0	0	1	61
05/30/88	0	0	0	1	0	0	1	62
06/03/88	0	0	1	0	0	0	1	63
06/10/88	0	0	2	0	0	0	2	65
06/13/88	0	0	1	0	0	0	1	66
06/30/88	0	0	1	1	0	0	2	68
07/01/88	0	2	0	0	0	0	2	70
07/05/88	0	0	0	1	0	0	1	71
07/11/88	0	0	1	0	0	0	1	72
07/20/88	0	0	1	0	0	0	1	73
09/16/88	0	0	1	0	0	0	1	74
09/23/88	0	0	0	1	0	0	1	75
09/27/88	0	0	1	0	0	0	1	76
10/31/88	0	0	1	0	0	0	1	77
11/14/88	0	1	0	0	0	0	1	78
11/15/88	0	0	1	0	0	0	1	79
12/01/88	0	1	1	0	0	0	2	81
12/12/88	0	1	0	0	0	0	1	82
12/23/88	0	1	0	0	0	0	1	83
01/03/89	0	0	2	0	0	0	2	85
01/05/89	0	1	0	0	0	0	1	86
01/13/89	0	0	0	1	0	0	1	87
01/18/89	0	2	0	0	0	0	2	89
01/19/89	0	1	0	0	0	0	1	90

01/20/89	0	0	1	0	0	0	1	91
01/24/89	0	1	1	0	0	0	2	93
01/26/89	0	0	1	0	0	0	1	94
02/01/89	0	2	0	0	0	0	2	96
02/02/89	0	0	1	0	0	0	1	97
02/06/89	0	1	1	0	0	0	2	99
02/15/89	0	0	1	0	0	0	1	100
02/16/89	0	0	1	0	0	0	1	101
02/18/89	0	1	0	0	0	0	1	102
02/22/89	0	0	1	0	0	0	1	103
02/24/89	0	0	1	0	0	0	1	104
02/27/89	0	0	2	0	0	0	2	106
03/04/89	0	0	1	0	0	0	1	107
03/06/89	0	0	1	0	0	0	1	108
03/13/89	0	2	0	0	0	0	2	110
03/20/89	0	0	1	0	0	0	1	111
03/27/89	0	0	1	0	0	0	1	112
03/29/89	0	1	0	0	0	0	1	113
04/19/89	0	0	1	0	0	0	1	114
06/08/89	0	1	0	0	0	0	1	115
06/13/89	0	0	1	0	0	0	1	116
06/16/89	0	1	1	0	0	0	2	118
06/21/89	0	0	0	1	0	0	1	119
06/23/89	0	0	1	1	0	0	2	121
07/07/89	0	1	0	0	0	0	1	122
07/17/89	0	0	1	0	0	0	1	123
07/21/89	0	0	1	0	0	0	1	124
07/23/89	0	0	1	0	0	0	1	125
07/25/89	0	0	1	0	0	0	1	126

B.3 Data Set A3

Database for AIRCRFT3

Date	# 1	# 2	# 3	# 4	# 5	NSC	Total	Cum Total
12/29/87	0	1	0	0	0	0	1	1
01/12/88	0	1	1	1	0	0	3	4
02/25/88	3	2	1	0	0	0	6	10
03/17/88	4	3	1	0	0	0	8	18
03/22/88	0	1	0	0	0	0	1	19
04/06/88	0	0	0	1	0	0	1	20
04/11/88	1	0	0	0	0	0	1	21
04/20/88	2	1	0	0	0	0	3	24
04/21/88	2	0	0	0	0	0	2	26
04/26/88	0	0	1	0	0	0	1	27
04/28/88	1	2	0	0	0	0	3	30
05/25/88	0	1	0	0	0	0	1	31
06/01/88	0	3	0	0	0	0	3	34
06/02/88	0	1	0	0	0	0	1	35
06/13/88	1	1	0	0	0	0	2	37
07/01/88	1	3	0	0	0	0	4	41
07/14/88	1	0	0	0	0	0	1	42
07/20/88	1	0	0	0	0	0	1	43
07/21/88	0	3	0	0	0	0	3	46
07/28/88	1	2	0	0	0	0	3	49
08/04/88	0	1	0	0	0	0	1	50
08/05/88	2	5	1	0	0	0	8	58
08/10/88	3	3	0	0	0	0	6	64
08/12/88	0	1	0	0	0	0	1	65
08/15/88	1	1	1	0	0	0	3	68
08/17/88	2	2	0	0	0	0	4	72
08/18/88	0	0	1	0	0	0	1	73
08/24/88	1	1	0	0	0	0	2	75
08/25/88	0	2	0	0	0	0	2	77
08/26/88	2	2	0	0	0	0	4	81
08/30/88	2	7	0	0	0	0	9	90
08/31/88	1	1	0	0	0	0	2	92
09/01/88	4	0	0	0	0	0	4	96
09/02/88	2	3	0	0	0	0	5	101
09/06/88	0	2	0	0	0	0	2	103
09/09/88	1	3	0	0	0	0	4	107
09/12/88	0	1	0	0	0	0	1	108
09/13/88	4	0	0	0	0	0	4	112
09/14/88	0	3	0	0	0	0	3	115
09/15/88	1	4	2	0	0	0	7	122
09/16/88	2	2	0	0	0	0	4	126
09/19/88	1	0	0	0	0	0	1	127
09/20/88	0	1	0	0	0	0	1	128
09/21/88	0	2	0	0	0	0	2	130
09/27/88	2	0	0	0	0	0	2	132
09/29/88	0	1	1	0	0	0	2	134
10/04/88	0	2	0	0	0	0	2	136
10/07/88	2	1	0	0	0	0	3	139
11/02/88	0	2	0	0	0	0	2	141
11/03/88	1	1	0	0	0	0	2	143
11/07/88	0	0	1	0	0	0	1	144
11/15/88	1	0	0	0	0	0	1	145
11/21/88	0	1	0	0	0	0	1	146
11/22/88	1	1	0	0	0	0	2	148
11/28/88	2	0	0	0	0	0	2	150
11/29/88	1	3	0	0	0	0	4	154
12/02/88	1	1	0	0	0	0	2	156
12/05/88	0	2	1	0	0	0	3	159
12/06/88	0	2	0	0	0	0	2	161
12/12/88	4	0	0	0	0	0	4	165
12/20/88	1	3	0	0	0	0	4	169
12/22/88	0	1	1	0	0	0	2	171
12/27/88	1	0	0	0	0	0	1	172
12/28/88	6	10	1	0	0	0	17	189

01/03/89	0	0	1	0	0	0	1	190
01/04/89	0	1	0	0	0	0	1	191
01/09/89	1	0	0	0	0	0	1	192
01/13/89	1	0	0	0	0	0	1	193
01/23/89	0	1	0	0	0	0	1	194
01/30/89	2	0	0	0	0	0	2	196
02/14/89	0	1	0	0	0	0	1	197
02/17/89	3	0	0	0	0	0	3	200
02/27/89	0	1	0	0	0	0	1	201
02/28/89	0	1	0	0	0	0	1	202
03/01/89	2	0	0	0	0	0	2	204
03/09/89	0	1	0	0	0	0	1	205
03/10/89	0	2	0	0	0	0	2	207
03/14/89	0	2	1	0	0	0	3	210
03/15/89	0	2	0	0	0	0	2	212
03/16/89	3	0	0	0	0	0	3	215
03/22/89	6	4	0	0	0	0	10	225
03/27/89	0	1	0	0	0	0	1	226
04/20/89	0	1	0	0	0	0	1	227
04/26/89	0	0	1	0	0	0	1	228
04/27/89	0	1	0	0	0	0	1	229
05/04/89	0	1	0	0	0	0	1	230
05/09/89	2	2	0	0	0	0	4	234
05/10/89	1	0	0	0	0	0	1	235
05/15/89	0	1	0	0	0	0	1	236
05/16/89	1	0	0	0	0	0	1	237
05/18/89	0	1	0	0	0	0	1	238
05/23/89	0	1	0	0	0	0	1	239
05/25/89	0	1	0	0	0	0	1	240
06/01/89	0	1	1	0	0	0	2	242
06/05/89	0	2	0	0	0	0	2	244
06/06/89	0	1	0	0	0	0	1	245
06/07/89	2	1	0	0	0	0	3	248
06/15/89	1	1	0	0	0	0	2	250
06/16/89	0	1	0	0	0	0	1	251
06/21/89	1	0	0	0	0	0	1	252
06/22/89	1	1	0	0	0	0	2	254
06/23/89	0	3	1	0	0	0	4	258
06/26/89	1	0	2	0	0	0	3	261
06/27/89	0	1	0	0	0	0	1	262
06/29/89	0	1	0	0	0	0	1	263
07/05/89	1	1	0	0	0	0	2	265
07/10/89	0	1	0	0	0	0	1	266
07/13/89	0	1	0	0	0	0	1	267
08/01/89	3	1	3	0	0	0	7	274
08/03/89	0	1	0	0	0	0	1	275
08/10/89	1	1	0	0	0	0	2	277
08/15/89	1	0	0	0	0	0	1	278
08/22/89	0	3	0	0	0	0	3	281
08/23/89	0	0	1	0	0	0	1	282
08/29/89	1	1	0	0	0	0	2	284
08/31/89	1	0	0	0	0	0	1	285
09/11/89	0	1	0	0	0	0	1	286
09/15/89	0	1	0	0	0	0	1	287
09/22/89	0	1	0	0	0	0	1	288

B.4 Data Set S1

Database for SPACE1

Date	# 1	# 2	# 3	# 4	# 5	NSC	Total	Cum Total
01/10/86	0	1	0	0	0	0	1	1
01/15/86	0	1	0	0	0	0	1	2
01/30/86	0	1	0	0	0	0	1	3
02/10/86	0	1	0	0	0	0	1	4
02/20/86	0	1	0	0	0	0	1	5
03/03/86	0	1	0	0	0	0	1	6
03/05/86	0	4	0	0	0	0	4	10
03/11/86	0	1	0	0	0	0	1	11
03/24/86	0	1	0	0	0	0	1	12
03/26/86	0	1	0	0	0	0	1	13
03/28/86	0	1	0	0	0	0	1	14
03/31/86	0	1	0	0	0	0	1	15
04/02/86	0	1	0	0	0	0	1	16
04/07/86	0	1	0	0	0	0	1	17
04/08/86	0	2	0	0	0	0	2	19
04/09/86	0	1	0	0	0	0	1	20
04/10/86	0	1	0	0	0	0	1	21
04/11/86	0	1	0	0	0	0	1	22
04/12/86	0	2	0	0	0	0	2	24
04/14/86	0	1	0	0	0	0	1	25
04/17/86	0	1	0	0	0	0	1	26
04/22/86	0	1	0	0	0	0	1	27
04/28/86	0	2	0	0	0	0	2	29
04/30/86	0	1	0	0	0	0	1	30
05/06/86	0	1	0	0	0	0	1	31
05/07/86	0	1	0	0	0	0	1	32
05/08/86	0	1	0	0	0	0	1	33
05/12/86	0	3	0	0	0	0	3	36
05/13/86	0	2	0	0	0	0	2	38
05/18/86	0	4	0	0	0	0	4	42
05/19/86	0	1	0	0	0	0	1	43
05/20/86	0	2	0	0	0	0	2	45
05/21/86	0	1	0	0	0	0	1	46
05/28/86	0	1	0	0	0	0	1	47
05/29/86	0	1	0	0	0	0	1	48
05/30/86	0	1	0	0	0	0	1	49
06/02/86	0	3	0	0	0	0	3	52
06/04/86	0	1	0	0	0	0	1	53
06/05/86	0	1	0	0	0	0	1	54
06/06/86	0	1	0	0	0	0	1	55
06/11/86	0	2	0	0	0	0	2	57
06/13/86	0	1	0	0	0	0	1	58
06/14/86	0	9	0	0	0	0	9	67
06/18/86	0	1	0	0	0	0	1	68
06/24/86	0	5	0	0	0	0	5	73
06/25/86	0	2	0	0	0	0	2	75
06/29/86	0	2	0	0	0	0	2	77
07/03/86	0	1	0	0	0	0	1	78
07/07/86	0	4	0	0	0	0	4	82
07/08/86	0	1	0	0	0	0	1	83
07/09/86	0	1	0	0	0	0	1	84
07/10/86	0	1	0	0	0	0	1	85
07/11/86	0	5	0	0	0	0	5	90
07/14/86	0	1	0	0	0	0	1	91
07/15/86	0	2	0	0	0	0	2	93
07/16/86	0	7	0	0	0	0	7	100
07/18/86	0	1	0	0	0	0	1	101
07/22/86	0	2	0	0	0	0	2	103
07/23/86	0	1	0	0	0	0	1	104
07/29/86	0	3	0	0	0	0	3	107
08/01/86	0	2	0	0	0	0	2	109
08/02/86	0	1	0	0	0	0	1	110
08/04/86	0	11	0	0	0	0	11	121
08/05/86	0	3	0	0	0	0	3	124

08/06/86	0	2	0	0	0	0	2	126
08/07/86	0	2	0	0	0	0	2	128
08/09/86	0	1	0	0	0	0	1	129
08/12/86	0	1	0	0	0	0	1	130
08/13/86	0	2	0	0	0	0	2	132
08/14/86	0	1	0	0	0	0	1	133
08/15/86	0	2	0	0	0	0	2	135
08/19/86	0	3	0	0	0	0	3	138
08/20/86	0	2	0	0	0	0	2	140
08/21/86	0	8	0	0	0	0	8	148
08/22/86	0	6	0	0	0	0	6	154
08/25/86	0	1	0	0	0	0	1	155
08/26/86	0	1	0	0	0	0	1	156
09/02/86	0	2	0	0	0	0	2	158
09/03/86	0	3	0	0	0	0	3	161
09/04/86	0	3	0	0	0	0	3	164
09/05/86	0	1	0	0	0	0	1	165
09/08/86	0	2	0	0	0	0	2	167
09/09/86	0	15	0	0	0	0	15	182
09/10/86	0	1	0	0	0	0	1	183
09/11/86	0	1	0	0	0	0	1	184
09/12/86	0	1	0	0	0	0	1	185
09/15/86	0	3	0	0	0	0	3	188
09/16/86	0	1	0	0	0	0	1	189
09/17/86	0	6	0	0	0	0	6	195
09/18/86	0	2	0	0	0	0	2	197
09/19/86	0	4	0	0	0	0	4	201
09/22/86	0	6	0	0	0	0	6	207
09/23/86	0	1	0	0	0	0	1	208
09/25/86	0	1	0	0	0	0	1	209
09/26/86	0	2	0	0	0	0	2	211
09/29/86	0	1	0	0	0	0	1	212
09/30/86	0	5	0	0	0	0	5	217
10/01/86	0	1	0	0	0	0	1	218
10/02/86	0	2	0	0	0	0	2	220
10/03/86	0	2	0	0	0	0	2	222
10/05/86	0	1	0	0	0	0	1	223
10/06/86	0	4	0	0	0	0	4	227
10/07/86	0	2	0	0	0	0	2	229
10/09/86	0	1	0	0	0	0	1	230
10/13/86	0	5	0	0	0	0	5	235
10/14/86	0	6	0	0	0	0	6	241
10/15/86	0	9	0	0	0	0	9	250
10/16/86	0	1	0	0	0	0	1	251
10/20/86	0	1	0	0	0	0	1	252
10/21/86	0	3	0	0	0	0	3	255
10/22/86	0	1	0	0	0	0	1	256
10/23/86	0	6	0	0	0	0	6	262
10/24/86	0	1	0	0	0	0	1	263
10/27/86	0	1	0	0	0	0	1	264
10/28/86	0	6	0	0	0	0	6	270
10/29/86	0	2	0	0	0	0	2	272
10/30/86	0	1	0	0	0	0	1	273
11/03/86	0	10	0	0	0	0	10	283
11/04/86	0	9	0	0	0	0	9	292
11/05/86	0	3	0	0	0	0	3	295
11/06/86	0	2	0	0	0	0	2	297
11/07/86	0	3	0	0	0	0	3	300
11/08/86	0	1	0	0	0	0	1	301
11/10/86	0	22	0	0	0	0	22	323
11/11/86	0	3	0	0	0	0	3	326
11/12/86	0	3	0	0	0	0	3	329
11/13/86	0	3	0	0	0	0	3	332
11/14/86	0	3	0	0	0	0	3	335
11/17/86	0	1	0	0	0	0	1	336
11/18/86	0	2	0	0	0	0	2	338
11/19/86	0	5	0	0	0	0	5	343
11/20/86	0	2	0	0	0	0	2	345
11/21/86	0	3	0	0	0	0	3	348
11/24/86	0	21	0	0	0	0	21	369

11/25/86	0	6	0	0	0	0	6	375
11/26/86	0	1	0	0	0	0	1	376
12/02/86	0	3	0	0	0	0	3	379
12/03/86	0	4	0	0	0	0	4	383
12/04/86	0	3	0	0	0	0	3	386
12/05/86	0	1	0	0	0	0	1	387
12/08/86	0	4	0	0	0	0	4	391
12/09/86	0	6	0	0	0	0	6	397
12/10/86	0	2	0	0	0	0	2	399
12/11/86	0	5	0	0	0	0	5	404
12/12/86	0	2	0	0	0	0	2	406
12/15/86	0	8	0	0	0	0	8	414
12/16/86	0	3	0	0	0	0	3	417
12/17/86	0	4	0	0	0	0	4	421
12/19/86	0	3	0	0	0	0	3	424
12/22/86	0	6	0	0	0	0	6	430
12/23/86	0	8	0	0	0	0	8	438
01/05/87	0	4	0	0	0	0	4	442
01/07/87	0	4	0	0	0	0	4	446
01/09/87	0	7	0	0	0	0	7	453
01/12/87	0	2	0	0	0	0	2	455
01/13/87	0	2	0	0	0	0	2	457
01/16/87	0	2	0	0	0	0	2	459
01/19/87	0	4	0	0	0	0	4	463
01/20/87	0	1	0	0	0	0	1	464
01/21/87	0	4	0	0	0	0	4	468
01/22/87	0	5	0	0	0	0	5	473
01/23/87	0	5	0	0	0	0	5	478
01/25/87	0	1	0	C	0	0	1	479
01/26/87	0	12	0	0	0	0	12	491
01/27/87	0	2	0	0	0	0	2	493
01/28/87	0	5	0	0	0	0	5	498
01/29/87	0	6	0	0	0	0	6	504
01/30/87	0	5	0	0	0	0	5	509
02/01/87	0	1	0	0	0	0	1	510
02/02/87	0	6	0	0	0	0	6	516
02/03/87	0	12	0	0	0	0	12	528
02/04/87	0	5	0	0	0	0	5	533
02/05/87	0	17	0	0	0	0	17	550
02/06/87	0	9	0	0	0	0	9	559
02/10/87	0	3	0	0	0	0	3	562
02/11/87	0	5	0	0	0	0	5	567
02/12/87	0	11	0	0	0	0	11	578
02/13/87	0	4	0	0	0	0	4	582
02/15/87	0	1	0	0	0	0	1	583
02/16/87	0	1	0	0	0	0	1	584
02/17/87	0	10	0	0	0	0	10	594
02/18/87	0	3	0	0	0	0	3	597
02/19/87	0	4	0	0	0	0	4	601
02/20/87	0	6	0	0	0	0	6	607
02/22/87	0	2	0	0	0	0	2	609
02/23/87	0	7	0	0	0	0	7	616
02/24/87	0	6	0	0	0	0	6	622
02/25/87	0	5	0	0	0	0	5	627
02/26/87	0	2	0	0	0	0	2	629
02/27/87	0	5	0	0	0	0	5	634
03/01/87	0	4	0	0	0	0	4	638
03/02/87	0	6	0	0	0	0	6	644
03/03/87	0	8	0	0	0	0	8	652
03/04/87	0	11	0	0	0	0	11	663
03/05/87	0	4	0	0	0	0	4	667
03/06/87	0	5	0	0	0	0	5	672
03/09/87	0	11	0	0	0	0	11	683
03/10/87	0	8	0	0	0	0	8	691
03/11/87	0	8	0	0	0	0	8	699
03/12/87	0	2	0	0	0	0	2	701
03/13/87	0	8	0	0	0	0	8	709
03/16/87	0	5	0	0	0	0	5	714
03/17/87	0	6	0	0	0	0	6	720
03/18/87	0	2	0	0	0	0	2	722

03/19/87	0	4	0	0	0	0	4	726
03/20/87	0	8	0	0	0	0	8	734
03/23/87	0	1	0	0	0	0	1	735
03/24/87	0	9	0	0	0	0	9	744
03/25/87	0	4	0	0	0	0	4	748
03/26/87	0	1	0	0	0	0	1	749
03/27/87	0	3	0	0	0	0	3	752
03/30/87	0	11	0	0	0	0	11	763
03/31/87	0	15	0	0	0	0	15	778
04/01/87	0	4	0	0	0	0	4	782
04/02/87	0	6	0	0	0	0	6	788
04/03/87	0	5	0	0	0	0	5	793
04/05/87	0	1	0	0	0	0	1	794
04/06/87	0	3	0	0	0	0	3	797
04/07/87	0	7	0	0	0	0	7	804
04/08/87	0	1	0	0	0	0	1	805
04/09/87	0	6	0	0	0	0	6	811
04/10/87	0	7	0	0	0	0	7	818
04/13/87	0	4	0	0	0	0	4	822
04/14/87	0	4	0	0	0	0	4	826
04/15/87	0	3	0	0	0	0	3	829
04/16/87	0	2	0	0	0	0	2	831
04/17/87	0	3	0	0	0	0	3	834
04/20/87	0	8	0	0	0	0	8	842
04/21/87	0	3	0	0	0	0	3	845
04/22/87	0	1	0	0	0	0	1	846
04/24/87	0	8	0	0	0	0	8	854
04/27/87	0	1	0	0	0	0	1	855
04/28/87	0	1	0	0	0	0	1	856
04/30/87	0	9	0	0	0	0	9	865
05/01/87	0	3	0	0	0	0	3	868
05/02/87	0	1	0	0	0	0	1	869
05/04/87	0	13	0	0	0	0	13	882
05/05/87	0	17	0	0	0	0	17	899
05/06/87	0	2	0	0	0	0	2	901
05/07/87	0	2	0	0	0	0	2	903
05/08/87	0	5	0	0	0	0	5	908
05/09/87	0	1	0	0	0	0	1	909
05/11/87	0	15	0	0	0	0	15	924
05/12/87	0	1	0	0	0	0	1	925
05/13/87	0	2	0	0	0	0	2	927
05/14/87	0	5	0	0	0	0	5	932
05/15/87	0	3	0	0	0	0	3	935
05/18/87	0	2	0	0	0	0	2	937
05/19/87	0	2	0	0	0	0	2	939
05/20/87	0	5	0	0	0	0	5	944
05/21/87	0	11	0	0	0	0	11	955
05/22/87	0	1	0	0	0	0	1	956
05/24/87	0	1	0	0	0	0	1	957
05/26/87	0	8	0	0	0	0	8	965
05/27/87	0	7	0	0	0	0	7	972
05/28/87	0	1	0	0	0	0	1	973
05/29/87	0	2	0	0	0	0	2	975
06/01/87	0	3	0	0	0	0	3	978
06/02/87	0	5	0	0	0	0	5	983
06/04/87	0	9	0	0	0	0	9	992
06/05/87	0	14	0	0	0	0	14	1006
06/08/87	0	3	0	0	0	0	3	1009
06/09/87	0	8	0	0	0	0	8	1017
06/10/87	0	8	0	0	0	0	8	1025
06/11/87	0	7	0	0	0	0	7	1032
06/12/87	0	10	0	0	0	0	10	1042
06/13/87	0	1	0	0	0	0	1	1043
06/15/87	0	3	0	0	0	0	3	1046
06/16/87	0	8	0	0	0	0	8	1054
06/17/87	0	8	0	0	0	0	8	1062
06/18/87	0	16	0	0	0	0	16	1078
06/19/87	0	7	0	0	0	0	7	1085
06/21/87	0	2	0	0	0	0	2	1087
06/22/87	0	12	0	0	0	0	12	1099

06/23/87	0	14	0	0	0	0	14	1113
06/24/87	0	6	0	0	0	0	6	1119
06/25/87	0	9	0	0	0	0	9	1128
06/26/87	0	12	0	0	0	0	12	1140
06/27/87	0	4	0	0	0	0	4	1144
06/28/87	0	3	0	0	0	0	3	1147
06/29/87	0	17	0	0	0	0	17	1164
06/30/87	0	7	0	0	0	0	7	1171
07/01/87	0	11	0	0	0	0	11	1182
07/02/87	0	6	0	0	0	0	6	1188
07/06/87	0	4	0	0	0	0	4	1192
07/07/87	0	17	0	0	0	0	17	1209
07/08/87	0	5	0	0	0	0	5	1214
07/09/87	0	8	0	0	0	0	8	1222
07/10/87	0	13	0	0	0	0	13	1235
07/13/87	0	14	0	0	0	0	14	1249
07/14/87	0	13	0	0	0	0	13	1262
07/15/87	0	9	0	0	0	0	9	1271
07/16/87	0	2	0	0	0	0	2	1273
07/17/87	0	5	0	0	0	0	5	1278
07/20/87	0	7	0	0	0	0	7	1285
07/21/87	0	4	0	0	0	0	4	1289
07/22/87	0	1	0	0	0	0	1	1290
07/23/87	0	3	0	0	0	0	3	1293
07/24/87	0	1	0	0	0	0	1	1294
07/27/87	0	8	0	0	0	0	8	1302
07/28/87	0	5	0	0	0	0	5	1307
07/29/87	0	12	0	0	0	0	12	1319
07/30/87	0	10	0	0	0	0	10	1329
07/31/87	0	2	0	0	0	0	2	1331
08/03/87	0	17	0	0	0	0	17	1348
08/04/87	0	32	0	0	0	0	32	1380
08/05/87	0	11	0	0	0	0	11	1391
08/06/87	0	11	0	0	0	0	11	1402
08/07/87	0	4	0	0	0	0	4	1406
08/08/87	0	3	0	0	0	0	3	1409
08/10/87	0	3	0	0	0	0	3	1412
08/11/87	0	13	0	0	0	0	13	1425
08/12/87	0	6	0	0	0	0	6	1431
08/13/87	0	11	0	0	0	0	11	1442
08/14/87	0	8	0	0	0	0	8	1450
08/17/87	0	12	0	0	0	0	12	1462
08/18/87	0	18	0	0	0	0	18	1480
08/19/87	0	4	0	0	0	0	4	1484
08/20/87	0	19	0	0	0	0	19	1503
08/21/87	0	7	0	0	0	0	7	1510
08/22/87	0	9	0	0	0	0	9	1519
08/24/87	0	9	0	0	0	0	9	1528
08/25/87	0	11	0	0	0	0	11	1539
08/26/87	0	7	0	0	0	0	7	1546
08/27/87	0	14	0	0	0	0	14	1560
08/28/87	0	10	0	0	0	0	10	1570
08/29/87	0	3	0	0	0	0	3	1573
08/30/87	0	1	0	0	0	0	1	1574
08/31/87	0	3	0	0	0	0	3	1577
09/01/87	0	2	0	0	0	0	2	1579
09/02/87	0	4	0	0	0	0	4	1583
09/03/87	0	13	0	0	0	0	13	1596
09/04/87	0	8	0	0	0	0	8	1604
09/07/87	0	2	0	0	0	0	2	1606
09/08/87	0	17	0	0	0	0	17	1623
09/09/87	0	13	0	0	0	0	13	1636
09/10/87	0	19	0	0	0	0	19	1655
09/11/87	0	1	0	0	0	0	1	1656
09/13/87	0	2	0	0	0	0	2	1658
09/14/87	0	5	0	0	0	0	5	1663
09/15/87	0	27	0	0	0	0	27	1690
09/16/87	0	10	0	0	0	0	10	1700
09/17/87	0	4	0	0	0	0	4	1704
09/18/87	0	12	0	0	0	0	12	1716

09/20/87	0	13	0	0	0	0	13	1729
09/21/87	0	7	0	0	0	0	7	1736
09/22/87	0	7	0	0	0	0	7	1743
09/23/87	0	4	0	0	0	0	4	1747
09/24/87	0	2	0	0	0	0	2	1749
09/25/87	0	2	0	0	0	0	2	1751
09/28/87	0	2	0	0	0	0	2	1753
09/29/87	0	8	0	0	0	0	8	1761
09/30/87	0	2	0	0	0	0	2	1763
10/01/87	0	8	0	0	0	0	8	1771
10/02/87	0	2	0	0	0	0	2	1773
10/03/87	0	1	0	0	0	0	1	1774
10/04/87	0	2	0	0	0	0	2	1776
10/05/87	0	11	0	0	0	0	11	1787
10/06/87	0	8	0	0	0	0	8	1795
10/07/87	0	5	0	0	0	0	5	1800
10/08/87	0	6	0	0	0	0	6	1806
10/09/87	0	3	0	0	0	0	3	1809
10/12/87	0	6	0	0	0	0	6	1815
10/13/87	0	5	0	0	0	0	5	1820
10/14/87	0	3	0	0	0	0	3	1823
10/15/87	0	12	0	0	0	0	12	1835
10/16/87	0	1	0	0	0	0	1	1836
10/19/87	0	3	0	0	0	0	3	1839
10/20/87	0	6	0	0	0	0	6	1845
10/21/87	0	5	0	0	0	0	5	1850
10/22/87	0	5	0	0	0	0	5	1855
10/23/87	0	6	0	0	0	0	6	1861
10/26/87	0	14	0	0	0	0	14	1875
10/27/87	0	2	0	0	0	0	2	1877
10/28/87	0	3	0	0	0	0	3	1880
10/29/87	0	2	0	0	0	0	2	1882
10/30/87	0	3	0	0	0	0	3	1885
11/02/87	0	2	0	0	0	0	2	1887
11/03/87	0	7	0	0	0	0	7	1894
11/04/87	0	5	0	0	0	0	5	1899
11/05/87	0	3	0	0	0	0	3	1902
11/06/87	0	6	0	0	0	0	6	1908
11/09/87	0	6	0	0	0	0	6	1914
11/10/87	0	5	0	0	0	0	5	1919
11/11/87	0	6	0	0	0	0	6	1925
11/12/87	0	2	0	0	0	0	2	1927
11/13/87	0	1	0	0	0	0	1	1928
11/16/87	0	6	0	0	0	0	6	1934
11/17/87	0	6	0	0	0	0	6	1940
11/18/87	0	1	0	0	0	0	1	1941
11/19/87	0	14	0	0	0	0	14	1955
11/20/87	0	4	0	0	0	0	4	1959
11/23/87	0	7	0	0	0	0	7	1966
11/24/87	0	1	0	0	0	0	1	1967
11/25/87	0	4	0	0	0	0	4	1971
11/30/87	0	7	0	0	0	0	7	1978
12/01/87	0	7	0	0	0	0	7	1985
12/02/87	0	1	0	0	0	0	1	1986
12/03/87	0	5	0	0	0	0	5	1991
12/04/87	0	1	0	0	0	0	1	1992
12/07/87	0	6	0	0	0	0	6	1998
12/08/87	0	1	0	0	0	0	1	1999
12/09/87	0	3	0	0	0	0	3	2002
12/10/87	0	6	0	0	0	0	6	2008
12/11/87	0	3	0	0	0	0	3	2011
12/12/87	0	2	0	0	0	0	2	2013
12/14/87	0	5	0	0	0	0	5	2018
12/16/87	0	2	0	0	0	0	2	2020
12/17/87	0	1	0	0	0	0	1	2021
12/18/87	0	1	0	0	0	0	1	2022
12/19/87	0	1	0	0	0	0	1	2023
12/21/87	0	1	0	0	0	0	1	2024
12/22/87	0	5	0	0	0	0	5	2029
12/23/87	0	2	0	0	0	0	2	2031

01/01/88	0	2	0	0	0	0	2	2033
01/04/88	0	2	0	0	0	0	2	2035
01/05/88	0	1	0	0	0	0	1	2036
01/06/88	0	7	0	0	0	0	7	2043
01/07/88	0	4	0	0	0	0	4	2047
01/08/88	0	1	0	0	0	0	1	2048
01/09/88	0	4	0	0	0	0	4	2052
01/11/88	0	2	0	0	0	0	2	2054
01/12/88	0	13	0	0	0	0	13	2067
01/13/88	0	6	0	0	0	0	6	2073
01/14/88	0	1	0	0	0	0	1	2074
01/15/88	0	2	0	0	0	0	2	2076
01/18/88	0	1	0	0	0	0	1	2077
01/19/88	0	4	0	0	0	0	4	2081
01/20/88	0	2	0	0	0	0	2	2083
01/22/88	0	3	0	0	0	0	3	2086
01/23/88	0	2	0	0	0	0	2	2088
01/25/88	0	2	0	0	0	0	2	2090
01/26/88	0	7	0	0	0	0	7	2097
01/27/88	0	2	0	0	0	0	2	2099
01/28/88	0	3	0	0	0	0	3	2102
01/29/88	0	9	0	0	0	0	9	2111
01/31/88	0	1	0	0	0	0	1	2112
02/03/88	0	5	0	0	0	0	5	2117
02/05/88	0	10	0	0	0	0	10	2127
02/09/88	0	15	0	0	0	0	15	2142
02/10/88	0	1	0	0	0	0	1	2143
02/11/88	0	10	0	0	0	0	10	2153
02/12/88	0	20	0	0	0	0	20	2173
02/15/88	0	2	0	0	0	0	2	2175
02/16/88	0	2	0	0	0	0	2	2177
02/17/88	0	5	0	0	0	0	5	2182
02/18/88	0	3	0	0	0	0	3	2185
02/19/88	0	3	0	0	0	0	3	2188
02/21/88	0	1	0	0	0	0	1	2189
02/22/88	0	2	0	0	0	0	2	2191
02/23/88	0	2	0	0	0	0	2	2193
02/24/88	0	4	0	0	0	0	4	2197
02/25/88	0	2	0	0	0	0	2	2199
02/26/88	0	3	0	0	0	0	3	2202
02/29/88	0	4	0	0	0	0	4	2206
03/01/88	0	2	0	0	0	0	2	2208
03/02/88	0	3	0	0	0	0	3	2211
03/03/88	0	8	0	0	0	0	8	2219
03/04/88	0	1	0	0	0	0	1	2220
03/05/88	0	1	0	0	0	0	1	2221
03/07/88	0	1	0	0	0	0	1	2222
03/08/88	0	2	0	0	0	0	2	2224
03/09/88	0	5	0	0	0	0	5	2229
03/10/88	0	3	0	0	0	0	3	2232
03/11/88	0	3	0	0	0	0	3	2235
03/12/88	0	2	0	0	0	0	2	2237
03/15/88	0	3	0	0	0	0	3	2240
03/16/88	0	9	0	0	0	0	9	2249
03/17/88	0	13	0	0	0	0	13	2262
03/18/88	0	3	0	0	0	0	3	2265
03/19/88	0	1	0	0	0	0	1	2266
03/20/88	0	3	0	0	0	0	3	2269
03/21/88	0	2	0	0	0	0	2	2271
03/22/88	0	1	0	0	0	0	1	2272
03/23/88	0	3	0	0	0	0	3	2275
03/24/88	0	3	0	0	0	0	3	2278
03/25/88	0	2	0	0	0	0	2	2280
03/28/88	0	1	0	0	0	0	1	2281
03/30/88	0	4	0	0	0	0	4	2285
03/31/88	0	2	0	0	0	0	2	2287
04/01/88	0	7	0	0	0	0	7	2294
04/05/88	0	6	0	0	0	0	6	2300
04/06/88	0	4	0	0	0	0	4	2304
04/07/88	0	7	0	0	0	0	7	2311

04/08/88	0	9	0	0	0	0	9	2320
04/11/88	0	2	0	0	0	0	2	2322
04/12/88	0	4	0	0	0	0	4	2326
04/13/88	0	1	0	0	0	0	1	2327
04/14/88	0	2	0	0	0	0	2	2329
04/15/88	0	1	0	0	0	0	1	2330
04/16/88	0	1	0	0	0	0	1	2331
04/17/88	0	1	0	0	0	0	1	2332
04/19/88	0	4	0	0	0	0	4	2336
04/21/88	0	3	0	0	0	0	3	2339
04/22/88	0	2	0	0	0	0	2	2341
04/25/88	0	6	0	0	0	0	6	2347
04/26/88	0	2	0	0	0	0	2	2349
04/27/88	0	3	0	0	0	0	3	2352
04/28/88	0	2	0	0	0	0	2	2354
04/29/88	0	2	0	0	0	0	2	2356
05/01/88	0	5	0	0	0	0	5	2361
05/02/88	0	2	0	0	0	0	2	2363
05/03/88	0	2	0	0	0	0	2	2365
05/05/88	0	3	0	0	0	0	3	2368
05/06/88	0	1	0	0	0	0	1	2369
05/07/88	0	2	0	0	0	0	2	2371
05/09/88	0	3	0	0	0	0	3	2374
05/11/88	0	3	0	0	0	0	3	2377
05/13/88	0	1	0	0	0	0	1	2378
05/15/88	0	1	0	0	0	0	1	2379
05/19/88	0	4	0	0	0	0	4	2383
05/20/88	0	1	0	0	0	0	1	2384
05/25/88	0	2	0	0	0	0	2	2386
05/26/88	0	2	0	0	0	0	2	2388
05/27/88	0	2	0	0	0	0	2	2390
06/01/88	0	3	0	0	0	0	3	2393
06/02/88	0	2	0	0	0	0	2	2395
06/06/88	0	4	0	0	0	0	4	2399
06/09/88	0	1	0	0	0	0	1	2400
06/10/88	0	1	0	0	0	0	1	2401
06/14/88	0	2	0	0	0	0	2	2403
06/15/88	0	1	0	0	0	0	1	2404
06/16/88	0	2	0	0	0	0	2	2406
06/17/88	0	1	0	0	0	0	1	2407
06/20/88	0	2	0	0	0	0	2	2409
06/21/88	0	1	0	0	0	0	1	2410
06/22/88	0	2	0	0	0	0	2	2412
06/24/88	0	1	0	0	0	0	1	2413
06/27/88	0	4	0	0	0	0	4	2417
06/28/88	0	7	0	0	0	0	7	2424
06/29/88	0	3	0	0	0	0	3	2427
07/01/88	0	1	0	0	0	0	1	2428
07/04/88	0	2	0	0	0	0	2	2430
07/05/88	0	1	0	0	0	0	1	2431
07/06/88	0	1	0	0	0	0	1	2432
07/07/88	0	2	0	0	0	0	2	2434
07/08/88	0	1	0	0	0	0	1	2435
07/11/88	0	3	0	0	0	0	3	2438
07/12/88	0	3	0	0	0	0	3	2441
07/14/88	0	4	0	0	0	0	4	2445
07/15/88	0	4	0	0	0	0	4	2449
07/18/88	0	2	0	0	0	0	2	2451
07/20/88	0	2	0	0	0	0	2	2453
07/21/88	0	2	0	0	0	0	2	2455
07/22/88	0	1	0	0	0	0	1	2456
07/25/88	0	3	0	0	0	0	3	2459
07/26/88	0	1	0	0	0	0	1	2460
07/27/88	0	2	0	0	0	0	2	2462
07/29/88	0	2	0	0	0	0	2	2464
08/03/88	0	3	0	0	0	0	3	2467
08/05/88	0	1	0	0	0	0	1	2468
08/09/88	0	1	0	0	0	0	1	2469
08/12/88	0	1	0	0	0	0	1	2470
08/15/88	0	1	0	0	0	0	1	2471

08/17/88	0	1	0	0	0	0	1	2472
08/18/88	0	2	0	0	0	0	2	2474
08/19/88	0	2	0	0	0	0	2	2476
08/24/88	0	2	0	0	0	0	2	2478
08/25/88	0	2	0	0	0	0	2	2480
08/26/88	0	1	0	0	0	0	1	2481
08/29/88	0	1	0	0	0	0	1	2482
08/30/88	0	4	0	0	0	0	4	2486
09/01/88	0	1	0	0	0	0	1	2487
09/02/88	0	2	0	0	0	0	2	2489
09/08/88	0	2	0	0	0	0	2	2491
09/11/88	0	1	0	0	0	0	1	2492
09/12/88	0	1	0	0	0	0	1	2493
09/13/88	0	1	0	0	0	0	1	2494
09/14/88	0	1	0	0	0	0	1	2495
09/15/88	0	1	0	0	0	0	1	2496
09/16/88	0	1	0	0	0	0	1	2497
09/19/88	0	1	0	0	0	0	1	2498
09/20/88	0	1	0	0	0	0	1	2499
09/21/88	0	1	0	0	0	0	1	2500
09/23/88	0	2	0	0	0	0	2	2502
09/28/88	0	1	0	0	0	0	1	2503
10/05/88	0	3	0	0	0	0	3	2506
10/11/88	0	1	0	0	0	0	1	2507
10/14/88	0	1	0	0	0	0	1	2508
10/18/88	0	1	0	0	0	0	1	2509
11/08/88	0	1	0	0	0	0	1	2510
11/16/88	0	2	0	0	0	0	2	2512
11/17/88	0	1	0	0	0	0	1	2513
11/22/88	0	1	0	0	0	0	1	2514
12/12/88	0	1	0	0	0	0	1	2515
12/13/88	0	1	0	0	0	0	1	2516
01/30/89	0	1	0	0	0	0	1	2517
02/06/89	0	1	0	0	0	0	1	2518
02/13/89	0	3	0	0	0	0	3	2521
04/01/89	0	1	0	0	0	0	1	2522
04/18/89	0	1	0	0	0	0	1	2523
05/10/89	0	2	0	0	0	0	2	2525

B.5 Data Set W1

Database for WST1

Date	# 1	# 2	# 3	# 4	# 5	NSC	Total	Cum Total
02/12/90	0	0	0	2	0	0	2	2
03/26/90	0	0	35	0	0	0	35	37
03/29/90	0	0	0	1	2	0	3	40
03/30/90	0	0	0	2	4	0	6	46
04/02/90	0	0	0	3	0	0	3	49
04/03/90	0	0	0	7	0	0	7	56
04/05/90	0	0	0	5	0	0	5	61
04/10/90	0	0	0	7	0	0	7	68
04/11/90	0	0	0	1	0	0	1	69
04/12/90	0	0	0	1	0	0	1	70
04/18/90	0	0	0	15	0	0	15	85
04/19/90	0	0	0	10	0	0	10	95
04/20/90	0	0	2	10	0	0	12	107
04/23/90	0	0	3	3	1	0	7	114
04/24/90	0	0	0	6	0	0	6	120
04/25/90	0	0	0	14	0	0	14	134
04/26/90	0	0	0	3	0	0	3	137
04/27/90	0	0	0	3	0	0	3	140
04/30/90	0	0	0	7	0	0	7	147
05/01/90	0	0	0	1	0	0	1	148
05/02/90	0	0	0	9	1	0	10	158
05/03/90	0	0	1	20	0	0	21	179
05/04/90	0	0	0	3	0	0	3	182
05/08/90	0	0	0	8	0	0	8	190
05/09/90	0	0	2	9	0	0	11	201
05/22/90	0	0	1	7	0	0	8	209
05/23/90	0	0	0	7	0	0	7	216
05/29/90	0	0	1	0	0	0	1	217
05/30/90	0	0	0	21	0	0	21	238
05/31/90	0	0	0	5	0	0	5	243
06/01/90	0	0	0	5	0	0	5	248
06/04/90	0	0	0	9	0	0	9	257
06/05/90	0	0	0	6	0	0	6	263
06/06/90	0	0	0	12	0	0	12	275
06/07/90	0	0	0	12	0	0	12	287
06/14/90	0	1	3	15	1	0	20	307
06/15/90	0	0	0	22	3	0	25	332
06/18/90	0	0	0	19	2	0	21	353
06/19/90	0	0	1	2	0	0	3	356
06/21/90	0	0	3	31	4	0	38	394
06/22/90	0	0	1	0	0	0	1	395
06/27/90	0	0	0	6	0	0	6	401
06/28/90	0	0	3	7	0	0	10	411
06/29/90	0	0	2	28	6	0	36	447
07/31/90	0	0	1	0	0	0	1	448
08/01/90	0	0	2	0	0	0	2	450

Appendix C. *Detailed Analysis and Design*

This appendix contains the detailed analysis and design of software to implement the candidate software reliability models.

C.1 Background

Five categories of software failures exist, ranging from critical to noncritical, each one described in terms of mission success [86:8-2]. These categories have been applied to IOT&E, with the following software failure severity levels applied [23:14]:

- **System Abort**. Severity Level 1. Software or firmware problem that results in a system abort.
- **System Degraded No Workaround**. Severity Level 2. Software or firmware problem that degrades the system and no alternative workaround exists (program restarts not acceptable).
- **System Degraded Workaround**. Severity Level 3. Software or firmware problem that degrades the system and there exists an alternative workaround (e.g., system rerouting through operator switchology; program restart not acceptable).
- **System Not Degraded**. Severity Level 4. An indicated software or firmware problem that does not degrade the system or any essential system function.
- **Minor Fault**. Severity Level 5. All other minor nonfunctional software deficiencies.

Currently, most software reliability models assume either all errors have the same weight (or severity level) or the weighting is based on observations with respect to time, e.g.–the most current observations will have more weight than older ones [34, 64, 89]. This thesis effort focuses on the use of constant weighting for all software failures; however, the implementation design must be such that a weighting scheme based on severity levels can be implemented in the future.

C.2 Requirements Analysis

Structured analysis techniques were used to determine requirements. The initial requirements definition was then expressed as a requirements specification through the use of a context diagram

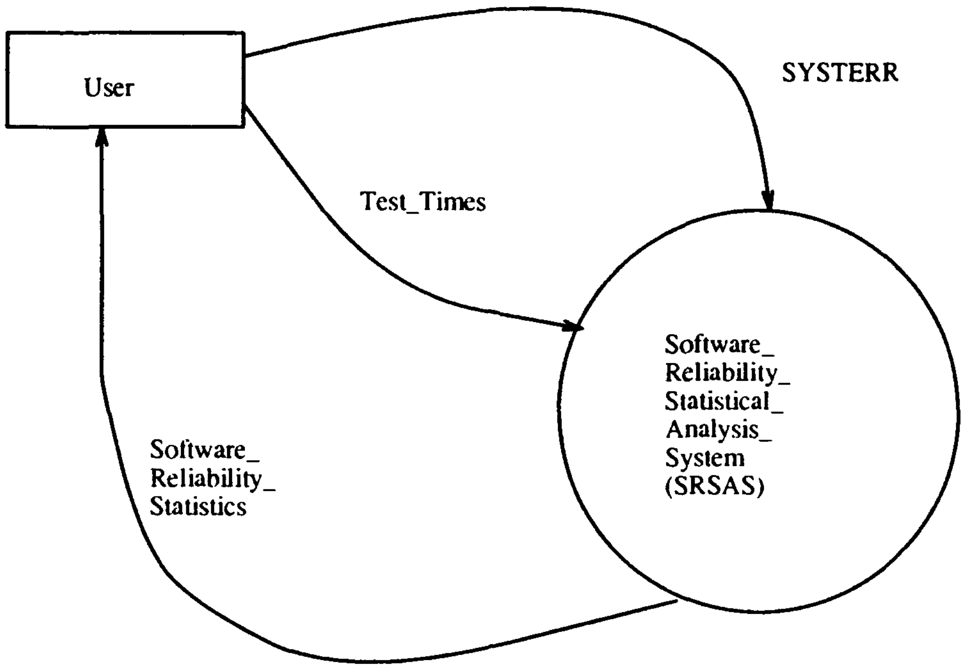

Figure C.1. Level 0 Context Diagram for SRSAS

and data flow diagrams (DFDs) [82]. Although a classical requirements analysis approach was used, the resulting specification can be applied to either structured design (with functional decomposition) or Object Oriented Design (OOD) [13]. The initial context diagram for the Software Reliability Statistical Analysis System (SRSAS) is shown in Figure C.1. The HQ AFOTEC software maturity data base, SYSTERR, provides the initial input into the SRSAS, with additional test times and durations input as necessary. The output is the Software_Reliability_Statistics, in terms of failures, failure intensities, and confidence intervals, that are necessary to assess the candidate models. The SRSAS is further refined in the breakout of lower level DFDs.

C.2.1 Level 1 DFD. The Level 1 DFD is shown in Figure C.2. The SRSAS was decomposed into the four functions Reduce_Data, Assign_Times, Determine_Execution_Time_Data, and Determine_Logarithmic_Time_Data. Reduce_Data uses the incoming SYSTERR data base to generate a reduced set of failure count data. Assign_Times uses this data output (as well as any additional time duration data from the user) to assign execution times to failures and calcu-

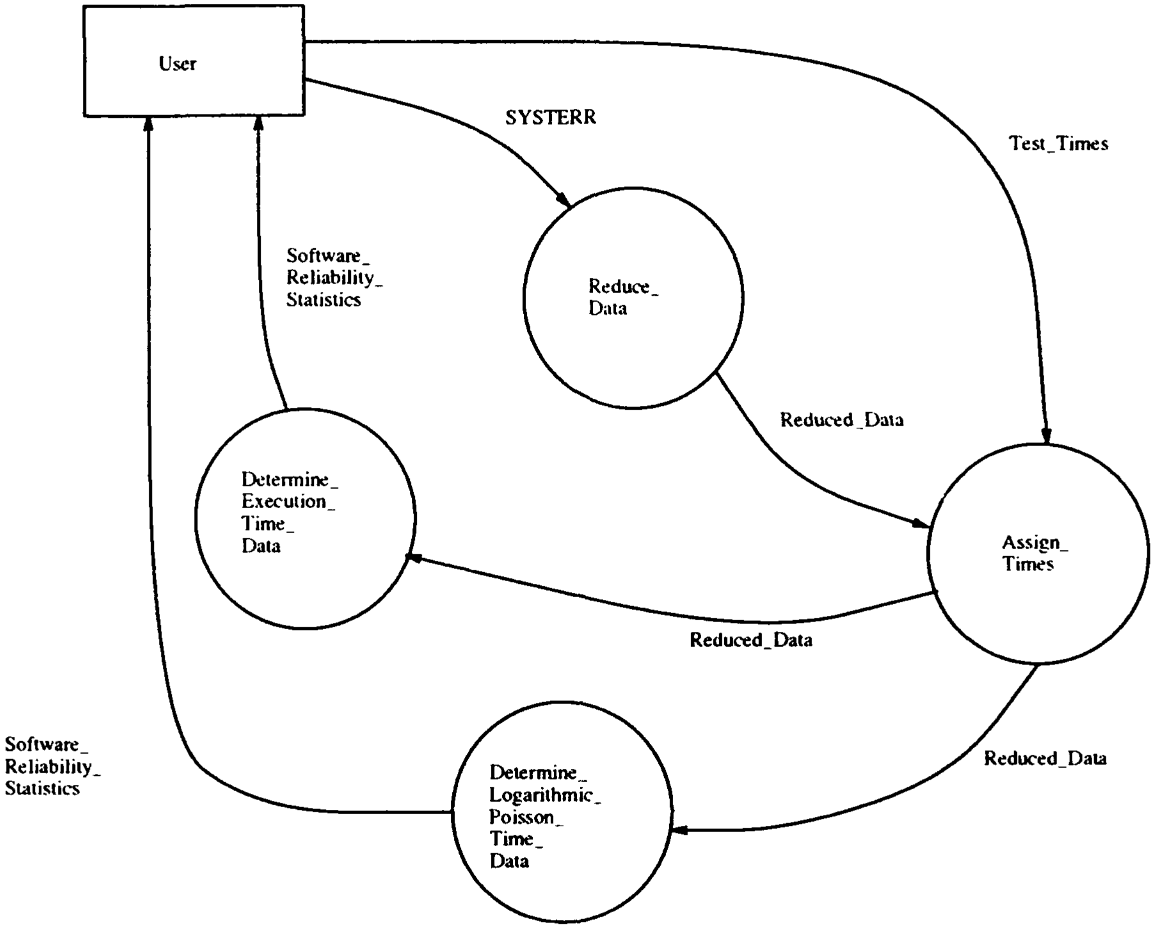

Figure C.2. Level 1 DFD for SRSAS

late time statistics, such as total test time. This information is used by the other two functions Determine_Execution_Time_Data and Determine_Logarithmic_Time_Data. The functions Determine_Execution_Time_Data and Determine_Logarithmic_Time_Data are based on examples and equations in Musa et al.; however, no further decomposition of the functions Reduce_Data and Assign_Times is possible without making design decisions.

C.2.1.1 Level 2 DFD: Determine_Execution_Time_Data. The Level 2 DFD for Determine_Execution_Time_Data is shown in Figure C.3. The time and failure information is taken and applied against the program structure identified in Musa et al. for a tabular software re-

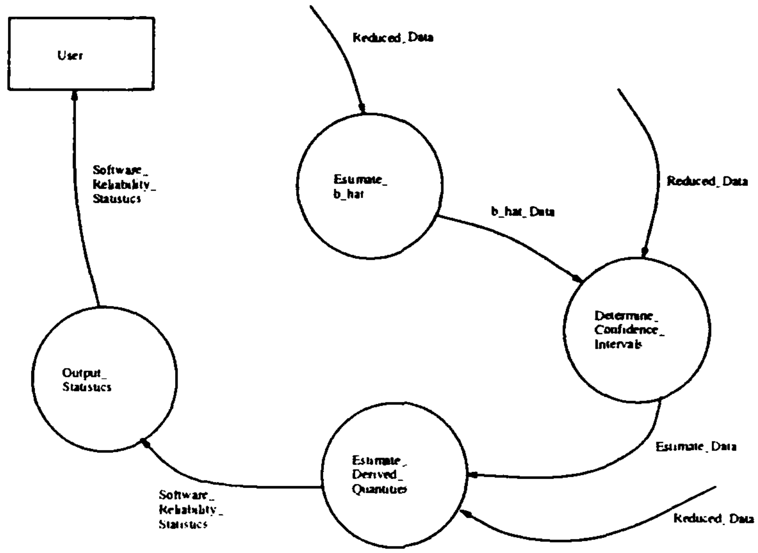

Figure C.3. Level 2 DFD for Determine_Execution_Time_Data

liability program [64:588-589]. The output of this then goes to the user in the form of Software_Reliability_Statistics.

C.2.1.2 Level 2 DFD: Determine_Logarithmic_Time_Data. The Level 2 DFD for Determine_Logarithmic_Time_Data is shown in Figure C.4. As with the Determine_Execution_Time_Data module, time and failure information is taken and applied against the program structure identified in Musa et al. for a tabular software reliability program [64:588-589]. The output of this also goes to the user in the form of Software_Reliability_Statistics.

C.3 Requirements Specification

Specification of the initial requirements was performed based on the Level 0 Context Diagram, and the lower level DFDs, which defined the objects and functions of the system. These specifications formed the baseline for the software design.

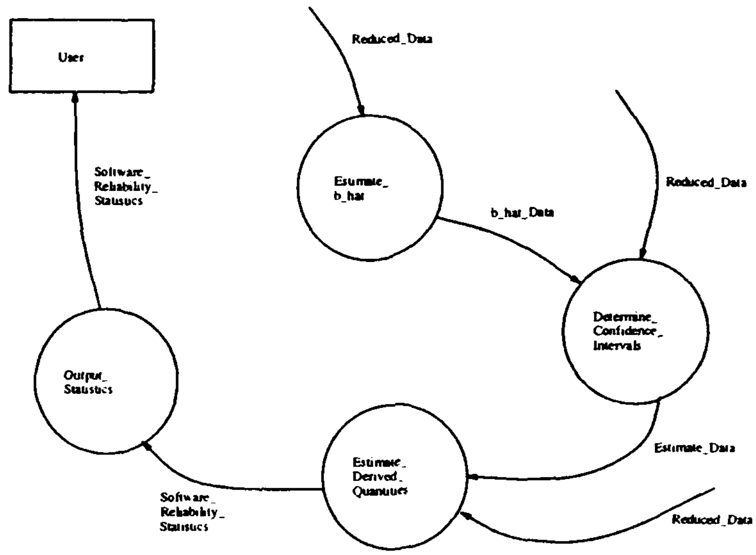

Figure C.4. Level 2 DFD for Determine_Logarithmic_Time_Data

C.4 Software Design

The software design effort was at two levels: high-level design effort (which included transition from structured analysis to OOD), and abstract data type (ADT) selection and low-level design effort. The waterfountain approach allowed a chance to revisit the different design levels, as well as the initial analysis, throughout the design process [84]. The discussion in the following paragraphs reflects that iterative nature, and will present the design effort.

The front end of the analysis was based on structured analysis techniques; however, there is a need to establish historical data from previous software reliability analysis to enable future validation efforts of other software reliability models [26:200]. Toward this end, the concept of data persistence will be encorporated into the design effort, specifically in the development of data stores for the different functions to use. In order to optimize the design and implementation of the data base and the accompanying software, a selection of the most appropriate data model must be made. The model itself is simply a collection of conceptual tools that can be used to describe the actual data, data semantics, data relationships, and existing consistency constraints between data [49:6].

While there have been many data models proposed and implemented for databases, they fall into four basic categories: physical data models; record-based logical models; object-based logical models; and object-oriented models [49:6],[97:7]. Of the first three, object-based logical models are the best suited for the logical and external schemas of describing data at both the conceptual and view levels [49:6]. The physical design of the data base would then be done in a relational model for the internal schema. Object-oriented models include the aspects of object-based models (object identity and type hierarchy), as well as data abstraction and user defined operations [97:92]. This makes the object-oriented models better suited for schema description at all three levels (internal, logical, and external); however, the Clipper programming language supports a more relational implementation of the data base at the internal schema level [66:3-7]. This requires at least an object-based model, if not an object-oriented model. In support of this, a transformation to the method of OOD was done using the following steps [12:17]:

- Identify objects and their attributes from all sources and destinations of data as well as data stores.
- Identify all operations suffered by and required of each object.
- Establish visibility among objects.
- Establish interfaces of objects.

C.4.1 Identification of Objects and Their Attributes. The initial Level 1 DFD was revised, taking into account the design decision to incorporate data persistence (Figure C.5). From this final DFD, the following objects and attributes were identified:

- SYSTERR Data, with attributes Date, Severity Level, Software Problem Report Number, and Description.
- Failure_Count, with attributes of Date and Number of Failures for each Severity Level.
- Test_Time, with attributes of Test_Duration, Time_Grouping,

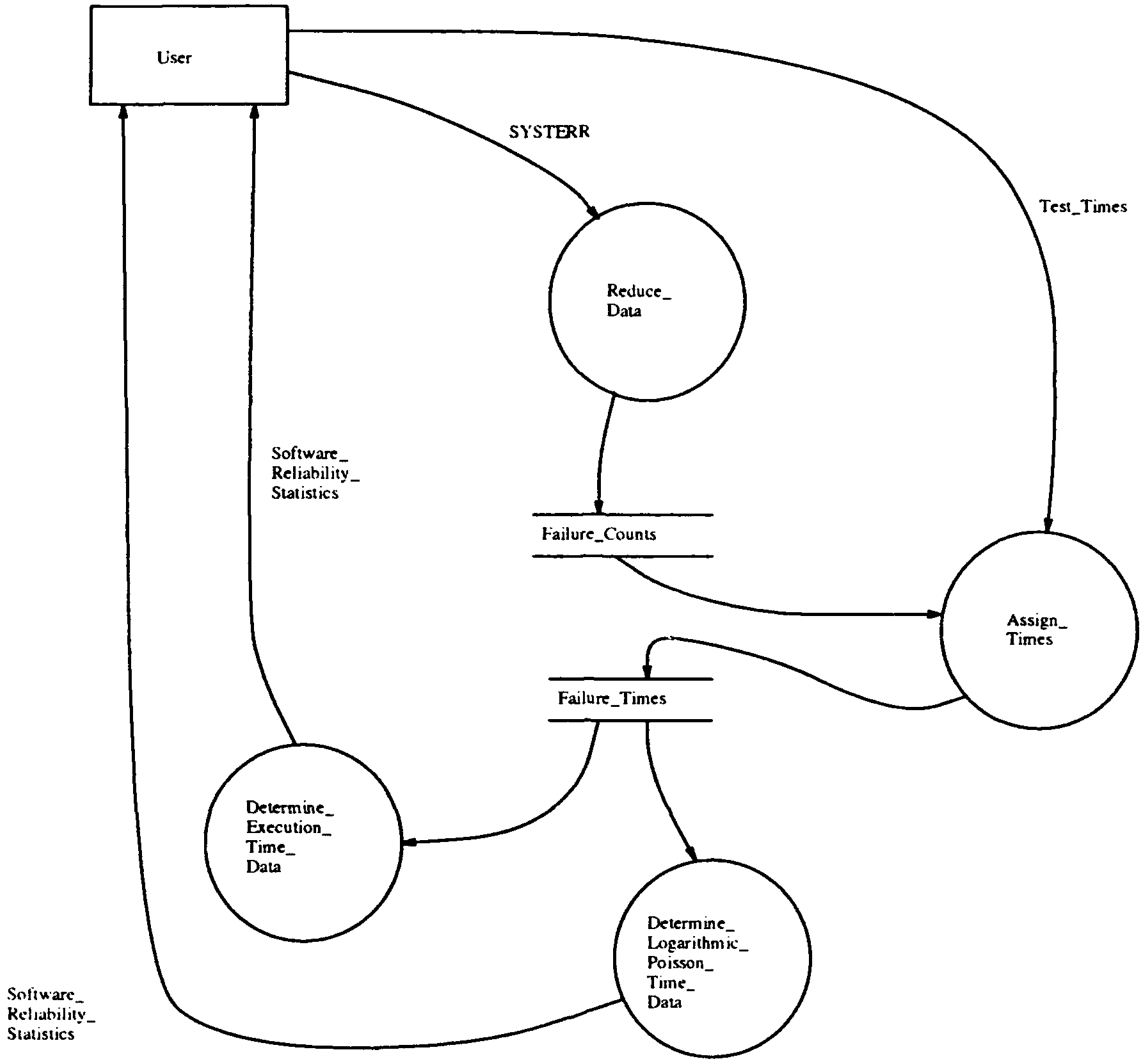

Figure C.5. Revised Level 1 DFD for SRSAS

- Failure_Time, with attributes of Date, Local_Time_Of_Occurrence, Test_Duration, Total_Time, and Total_Time_Of_Occurrence.
- Software_Reliability_Statistics, with attributes of Time, Number_Of_Failures_Experienced, Number_Of_Failures_Expected, and Failure_Intensity.

C.4.2 Operations Suffered by and Required of Each Object. Operations that identify the behavior of each object were identified as follows:

- SYSTERR Data: none.

- Failure_Count: add up data common to the same Date, sort the data in chronological order.

- Test_Time: none.

- Failure_Time: determine Local_Time_Of_Occurrence, Test_Duration, Total_Time, and Total_Time_Of_Occurrence.

- Software_Reliability_Statistics: determine Number_Of_Failures_Experienced, Failure_Intensity, and Number_Of_Failures_Expected.

Operations that are required of each object were identified as follows:

- SYSTERR Data: provide Date and Severity Level.

- Failure_Count: provide Date and Total_Failures_to_Date.

- Test_Time: provide Test_Duration and Time_Grouping.

- Failure_Time: provide Local_Time_Of_Occurrence, Test_Duration, Total_Time_Of_Occurrence, and Total_Time.

- Software_Reliability_Statistics: provide Number_Of_Failures_Experienced, Failure_Intensity, and Number_Of_Failures_Expected.

C.4.3 Establish Visibility Among Objects. The visibility among objects is based on the relationships between the databases, and is shown in Figure C.6. The module diagram is the basis for transformation of the design information into an implementation (in this case, Clipper code). The objects come directly from those identified above. As naming conventions for an MS-DOS environment are limited to eight characters, with Clipper supplying the `.PRG` extension, and Clipper is more functional than object-oriented, the objects were broken out into program modules and databases with a basic naming convention (see Table C.1).

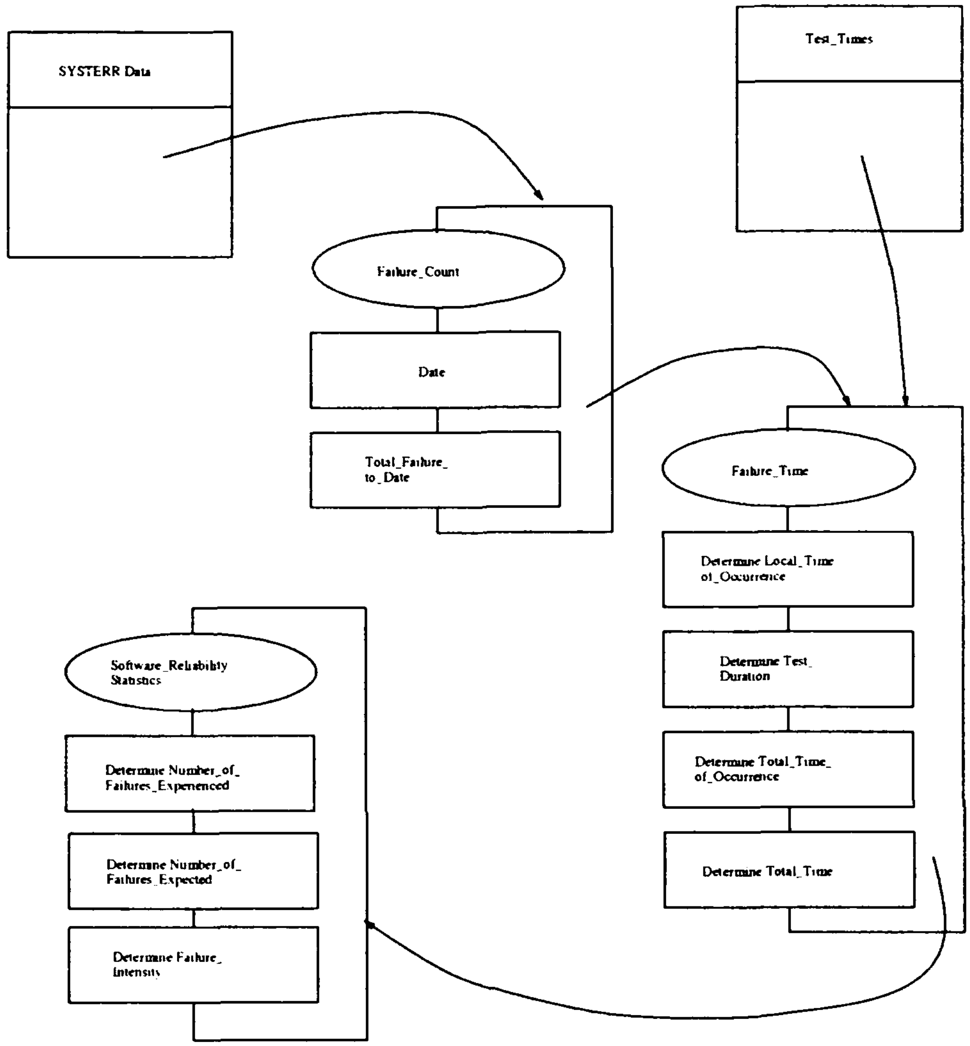

Figure C.6. Visibility Among SRSAS Objects

C.4.4 Establish Interfaces of Objects. The interfaces of the objects are normally written as part of the code, such as the package specification in Ada. This also was performed for Clipper, which is a more functional than object-oriented programming language (see Appendix D).

C.5 Determine Need for Abstract Data Type

Based on the requirements provided and the availability of the Clipper programming environment, a data base implementation was chosen over the development and implementation of a specialized ADT. This further simplified the low-level design process, as the software was required

Table C.1. Listing of Objects and Implementation Name

Object	Program Module	Database
FAILURE_COUNT	COUNT.PRG	COUNT.DBF
FAILURE_TIME	SRTIME.PRG	
	SRTBUILD.PRG	TIME.DBF
	SRTDATE.PRG	TIME.DBF
	SRTB1.PRG	B1DATA.DBF
		COUNT.DBF
		TIME.DBF
	SRTEST.PRG	COUNT.DBF
		TIME.DBF
		TIMEDTE.DBF
SOFTWARE_RELIABILITY_STATISTICS	SREXEC.PRG	TIME.DBF
	SRLOG.PRG	TIME.DBF

only to manipulate the data in the database, and not perform the database implementation itself. Thus, no specific ADT was necessary or would provide additional capabilities for software development.

Appendix D. *Candidate Software Reliability Model Implementation Code*

This appendix contains the code that was developed in order to perform evaluation of the candidate software reliability models.

D.1 *Software Reliability Statistical Analysis Software (SRSAS)*

```
*****************************************************************************
*                                                                           *
*  Title    : Software Reliability Statistical Analysis System (SRSAS)      *
*  Version  : 3.3                                                           *
*  Date     : 15 Oct 91                                                     *
*  Author   : Capt Joseph J. Stanko                                         *
*  Security : Unclassified                                                  *
*  Purpose  : This program does three things:                               *
*             1.) Calculate initial statistics from existing SYSTERR        *
*                 software maturity database for use in software reliability *
*                 model evaluation                                          *
*             2.) Generate a database of failure times based on average test *
*                 times, actual test time times, or estimated test times.   *
*             3.) Perform calculations on data to determined goodness-of-fit *
*                 for each candidate software reliability model.            *
*                                                                           *
*  Theory   : 1.) The program checks for the existence of a summary database *
*                 and if one does not exist, one is constructed solely from *
*                 the SYSTERR fields of the software maturity database      *
*             2.) Next, the program prompts for data not in the SYSTERR     *
*                 database (such as total test time or test durations) and  *
*                 generates a database of failure times.                    *
*             3.) Finally, the program takes the failure time information   *
*                 and calculates estimates of model parameters and their    *
*                 confidence intervals. Outputs are given (in tabular form) *
*                 of actual and estimated data.                             *
*                                                                           *
*             NOTE:                                                         *
*             This is the initial transition from existing SYSTERR databases *
*             to the software reliability database for the Reliability     *
*             Analysis System (RAS). Future SYSTERR database configuration *
*             based on the 1 Oct 1990 AFOTECP 800-2 Vol 6 will have        *
*             fields that will be handled by RAS itself as an integrated   *
*             package.                                                      *
*                                                                           *
*  Database : This program uses three databases:                            *
*                                                                           *
*             SYSTERR.DBF  -  There were several different "versions" of this *
*                             database done by each test team. The following *
*                             are the fields found common to each that might *
*                             be useful for software reliability analysis:  *
*                                                                           *
*               Name     Type  Length  Decimal          Description         *
*             --------   ----  ------  -------   -------------------------- *
*             DATE       Date                    Date of occurrence of failure *
*             CPCI        C      10              CPCI associated with failure *
*             SEV_CODE    C       1              Severity Code (1-5) of failure *
*             DATE_FIX   Date                    Date failure fixed         *
*             TITLE       C      42              Description of failure     *
*             PROB_NUM    C      10              Software Problem Number (SPR) *
*                                                                           *
*             While this data is available from the existing SYSTERR database *
*             it does not include the time values necessary for reliability *
*             evaluation. This data must be prompted for from the user.     *
*                                                                           *
*             COUNT.DBF    -  This is an intermediate summary database of   *
*                             dates and number of failures:                 *
*                                                                           *
```

```
*               Name     Type  Length  Decimal          Description            *
*             --------   ----  ------  -------   ----------------------------- *
*             CAL_DATE   Date                    Date of occurrence of failure *
*             SEV_CODE_1  N      4               # of Severity Code 1 failures *
*             SEV_CODE_2  N      4               # of Severity Code 2 failures *
*             SEV_CODE_3  N      4               # of Severity Code 3 failures *
*             SEV_CODE_4  N      4               # of Severity Code 4 failures *
*             SEV_CODE_5  N      4               # of Severity Code 5 failures *
*             NO_SEV_CODE N      4               # of failures not coded       *
*             TOT_NUM     N      4               Total Number for this date    *
*             TOTAL       N      4               Overal total of failures      *
*                                                                              *
*             TIME.DBF    -  This is a final database of dates and estimated   *
*                            failure times and test durations:                 *
*                                                                              *
*               Name     Type  Length  Decimal          Description            *
*             --------   ----  ------  -------   ----------------------------- *
*             CAL_DATE   Date                    Date of occurrence of failure *
*             L_TIME_OCC  N     10        2      "Local" time of failure occur *
*                                                (wrt to start of that test)   *
*             TEST_DUR    N     10        2      Duration of test for that day *
*             T_TIME_OCC  N     10        2      "Total" time of failure occur *
*                                                (wrt to all total test time)  *
*             TOTAL       N      4               Total failures to that point  *
*                                                                              *
*  Modules  : Calls the following modules for operation:                       *
*             SRCOUNT.PRG    - Initializes the database COUNT.DBF, reduces     *
*                              the SYSTERR.DBF entries into a count summary    *
*                              form, and puts in ascending chronological order.*
*             SRPRINT.PRG    - Prints the COUNT.DBF.                           *
*             SRTIME.PRG     - Initializes and generates the database TIME.DBF.*
*             SREXEC.PRG     - Perform calculations on the TIME.DBF data with  *
*                              Musa Execution Time Model.                      *
*             SRLOG.PRG      - Perform calculations on the TIME.DBF data with  *
*                              Musa-Okumoto Logarithmic Exection Time Model.   *
*                                                                              *
********************************************************************************
clear screen
*-------------------------------------------------------------------------------
*    Variable Section:
option = "C"                    && memvar for main menu
*-------------------------------------------------------------------------------
*    Set Section:
set decimal to 9                && set decimal length beyond default (2)
*-------------------------------------------------------------------------------
*    Main Loop:
do while upper(option) <> "X"
  set color to w+/b,g/n
  @  0,0  clear
  @  3,12 say "Software Reliability Statistical Analysis System (SRSAS)"
  @  4,12 say "                 Version 3.3, Oct 1991"
  @  6,20 say "C - Create Count Data Base"
  @  8,20 say "P - Print Count Data Base"
  @ 10,20 say "T - Create Time Data Base"
  @ 12,20 say "E - Execution Time Model"
  @ 14,20 say "L - Logarithmic Poisson Execution Time Model"
  @ 16,20 say "X - Exit"
  @ 20,20 say "Please Enter Option:";
          get option picture "@K !" valid(option$"CPTELX")
  read
  do case                       &&  Call sr programs based on menu input:
    case upper(option)="C"
      do srcount
    case upper(option)="P"
      do srprint
    case upper(option)="T"
      do srtime
    case upper(option)="E"
```

```
      do srexec
    case upper(option)="L"
      do srlog
    case upper(option)="X"
      @ 24,20 say "Exiting This Program ..."
    otherwise  && standard exception handling
      @ 23,20 say "Invalid Entry - Please Use C,P,T,E,L, or X"
  endcase
enddo
*-------------------------------------------------------------------------------
clear all
clear screen
quit
*******************************************************************************
```

D.2 Software Reliability COUNT.DBF Module

```
******************************************************************************
*                                                                            *
*  Title    : Software Reliability COUNT.DBF Module (SRCOUNT.PRG)            *
*  Version  : 3.3                                                            *
*  Date     : 2 Oct 91                                                       *
*  Author   : Capt Joseph J. Stanko                                          *
*  Security : Unclassified                                                   *
*  Purpose  : This program:                                                  *
*             1.) Calculates initial summary statistics from the SYSTERR     *
*                 software maturity databases for use in software reliability *
*                 model evaluation.                                          *
*                                                                            *
*  Theory   : One pass module. The program checks for the existence of a    *
*             summary database and if one does not exist, one is constructed *
*             solely from the SYSTERR fields of the software maturity        *
*             database.                                                      *
*                                                                            *
*  Database : This program uses two databases:                               *
*                                                                            *
*             SYSTERR.DBF  -  There were several different "versions" of this *
*                             database done by each test team.  The following *
*                             are the fields found common to each that might  *
*                             be useful for software reliability analysis:    *
*                                                                            *
*               Name     Type  Length  Decimal           Description          *
*             --------   ----  ------  -------   ---------------------------- *
*             DATE       Date                    Date of occurrence of failure *
*             CPCI        C      10              CPCI associated with failure  *
*             SEV_CODE    C       1              Severity Code (1-5) of failure *
*             DATE_FIX   Date                    Date failure fixed            *
*             TITLE       C      42              Description of failure        *
*             PROB_NUM    C      10              Software Problem Number (SPR) *
*                                                                            *
*             While this data is available from the existing SYSTERR database *
*             it does not include the time values necessary for reliability  *
*             evaluation.  This data must be prompted for from the user.     *
*                                                                            *
*             COUNT.DBF    -  This is an intermediate summary database of    *
*                             dates and number of failures:                  *
*                                                                            *
*               Name       Type  Length  Decimal         Description          *
*             --------     ----  ------  -------  --------------------------- *
*             CAL_DATE     Date                   Date of occurrence of failure *
*             SEV_CODE_1    N      4              # of Severity Code 1 failures *
*             SEV_CODE_2    N      4              # of Severity Code 2 failures *
*             SEV_CODE_3    N      4              # of Severity Code 3 failures *
*             SEV_CODE_4    N      4              # of Severity Code 4 failures *
*             SEV_CODE_5    N      4              # of Severity Code 5 failures *
*             NO_SEV_CODE   N      4              # of failures not coded      *
*             TOT_NUM       N      4              Total Number for this date   *
*             TOTAL         N      4              Overal total of failures     *
*                                                                            *
*  Modules  : None.                                                          *
*                                                                            *
******************************************************************************

*----------------------------------------------------------------------------
*     Check for COUNT.DBF or create if it does not exist:

if .not. file("COUNT.DBF")
  @ 23,20 say "Building COUNT.DBF Database ..."
  create template
  use template
  append blank
  replace field_name with "Cal_Date",     field_type with "DATE"
  append blank
  replace field_name with "Sev_Code_1",   field_type with "N",    ;
          field_len  with 4
  append blank
  replace field_name with "Sev_Code_2",   field_type with "N",    ;
          field_len  with 4
  append blank
  replace field_name with "Sev_Code_3",   field_type with "N",    ;
```

```
        field_len  with 4
append blank
replace field_name with "Sev_Code_4",   field_type with "N",    ;
        field_len  with 4
append blank
replace field_name with "Sev_Code_5",   field_type with "N",    ;
        field_len  with 4
append blank
replace field_name with "No_Sev_Code",  field_type with "N",    ;
        field_len  with 4
append blank
replace field_name with "Tot_Num",      field_type with "N",    ;
        field_len  with 4
append blank
replace field_name with "Total",        field_type with "N",    ;
        field_len  with 4
go top
close all
file = "COUNT.DBF"
create &file. from template
erase template.dbf
*-------------------------------------------------------------------------------
*     Now reduce the SYSTERR.DBF data into the COUNT.DBF database:
use COUNT alias COUNT          &&  Aliases sure do help disambiguate vars:
select 2
use SYSTERR alias MATURITY
store 0 to mtot                &&  Initialize counts for total and all
store 0 to msc1                &&    severity codes (sc's 1-5)
store 0 to msc2
store 0 to msc3
store 0 to msc4
store 0 to msc5
store 0 to mnsc                &&  Just in case some are "no severity code"
store DATE to mdate
@  0,0  clear
@  5,20 say "Tabulating Count Data ..."
do while .not. eof()           &&  Since each entry in the SYSTERR database
  mtot = mtot + 1              &&   is a single and separate failure, all
  store SEV_CODE to msevcode   &&   must be added up by date with summary
  do case                      &&   information on all severity codes
    case msevcode = "1"
      msc1 = msc1 + 1
    case msevcode = "2"
      msc2 = msc2 + 1
    case msevcode = "3"
      msc3 = msc3 + 1
    case msevcode = "4"
      msc4 = msc4 + 1
    case msevcode = "5"
      msc5 = msc5 + 1
    otherwise
      mnsc = mnsc + 1
  endcase
  skip
  if DATE <> mdate             &&  Check to see if we've moved to another date
    select COUNT               &&  If we have, save off the summary data
    append blank
    replace CAL_DATE with mdate
    replace SEV_CODE_1 with msc1
    replace SEV_CODE_2 with msc2
    replace SEV_CODE_3 with msc3
    replace SEV_CODE_4 with msc4
    replace SEV_CODE_5 with msc5
    replace NO_SEV_CODE with mnsc
    replace TOT_NUM with mtot
    replace TOTAL with 0
    store 0 to msc1            &&  reinitialize the summary variables
    store 0 to msc2
    store 0 to msc3
    store 0 to msc4
    store 0 to msc5
```

```
    store 0 to mnsc
    store 0 to mtot
    select MATURITY          &&  and do it again for the new date
    store DATE to mdate
  endif
enddo
*------------------------------
*     Since many of the entries in the SYSTERR database were not
*     in straight chronological order, the data needs to be sorted
*     and then compressed so that only one entry exists for any given date:
@ 7,20 say "Sorting the Tabulated Data ..."
select COUNT
sort on CAL_DATE to temp1
select 3
use temp1
@ 9,20 say "Compressing Tabulated Data ..."
store 1 to rec_num
store CAL_DATE to mdate
store SEV_CODE_1 to msc1
store SEV_CODE_2 to msc2
store SEV_CODE_3 to msc3
store SEV_CODE_4 to msc4
store SEV_CODE_5 to msc5
store NO_SEV_CODE to mnsc
store TOT_NUM to mtot
skip
do while .not. eof()
  rec_num = rec_num + 1
  if CAL_DATE = mdate
    msc1 = SEV_CODE_1 + msc1
    msc2 = SEV_CODE_2 + msc2
    msc3 = SEV_CODE_3 + msc3
    msc4 = SEV_CODE_4 + msc4
    msc5 = SEV_CODE_5 + msc5
    mnsc = NO_SEV_CODE + mnsc
    mtot = TOT_NUM + mtot
    replace SEV_CODE_1 with msc1
    replace SEV_CODE_2 with msc2
    replace SEV_CODE_3 with msc3
    replace SEV_CODE_4 with msc4
    replace SEV_CODE_5 with msc5
    replace NO_SEV_CODE with mnsc
    replace TOT_NUM with mtot
    goto rec_num - 1
    delete
    goto rec_num
  endif
  store CAL_DATE to mdate
  store SEV_CODE_1 to msc1
  store SEV_CODE_2 to msc2
  store SEV_CODE_3 to msc3
  store SEV_CODE_4 to msc4
  store SEV_CODE_5 to msc5
  store NO_SEV_CODE to mnsc
  store TOT_NUM to mtot
  skip
enddo
*------------------------------
*     Now sum the totals and include in the COUNT.DBF:
go top
mtot = 0
do while .not. eof()
  store TOT_NUM + mtot to mtot
  replace TOTAL with mtot
  skip
enddo
*------------------------------
*     And close up shop:
pack
```

```
  close all
  erase COUNT.DBF
  rename temp1.dbf to COUNT.DBF

else
  @ 23,20 say "COUNT.DBF Database Already Exists ..."
  wait "Hit any key to continue ..."
endif
*---------------------------------------------------------------------------------

return                          &&  to SRSAS main menu
**********************************************************************************
```

```
*****************************************************************************
*                                                                           *
*  Title     : Software Reliability Print Module (SRPRINT.PRG)              *
*  Version   : 3.3                                                          *
*  Date      : 2 Oct 91                                                     *
*  Author    : Capt Joseph J. Stanko                                        *
*  Security  : Unclassified                                                 *
*  Purpose   : This program:                                                *
*              Prints the contents of the COUNT.DBF to either a screen,     *
*              printer, or data file.                                       *
*              Currently only the COUNT.DBF is useful for output--the TIME.DBF *
*              has one entry for each failure recorded, and that would use  *
*              a lot of paper to print.  However, it would be simple to modify *
*              this program to print the TIME.DBF information to a file if  *
*              needed.                                                      *
*                                                                           *
*  Theory    : User is given option of where to print the COUNT.DBF database. *
*              It's a one pass, with default values initialized for screen  *
*              output (saves on paper!).                                    *
*                                                                           *
*  Database : This program uses one database:                               *
*                                                                           *
*              COUNT.DBF    -  This is an intermediate summary database of  *
*                              dates and number of failures:                *
*                                                                           *
*                Name      Type  Length  Decimal          Description       *
*              --------    ----  ------  -------  ------------------------- *
*              CAL_DATE    Date                   Date of occurrence of failure *
*              SEV_CODE_1   N      4              # of Severity Code 1 failures *
*              SEV_CODE_2   N      4              # of Severity Code 2 failures *
*              SEV_CODE_3   N      4              # of Severity Code 3 failures *
*              SEV_CODE_4   N      4              # of Severity Code 4 failures *
*              SEV_CODE_5   N      4              # of Severity Code 5 failures *
*              NO_SEV_CODE  N      4              # of failures not coded   *
*              TOT_NUM      N      4              Total Number for this date *
*              TOTAL        N      4              Overall total of failures *
*                                                                           *
*  Modules   : N/A                                                          *
*                                                                           *
*****************************************************************************

*----------------------------------------------------------------------------
*     Variable Section:

prvar   = "S"                     && Variable for print option
store   1        to LOC           && Line of Code--used for printing information
store "        " to dbname        && Name of database for output header
*----------------------------------------------------------------------------
*     First, see if COUNT.DBF exists:

if file("COUNT.DBF")

*----------------------------------------------------------------------------
*     If it does then do print, etc.

@ 23,10 say "Print Data to (S)creen, (P)rinter, (F)ile, or (R)eturn:";
        get prvar picture "@K !" valid(prvar$"SPFR")
read
@ 23,10 clear

if upper(prvar) <> "R"          &&  Make sure we don't want to return to SRSAS

  @  0,0  clear
  use COUNT
  replace TOTAL with TOT_NUM
  @  5,20 say "Data Base Name for Header:"  get dbname picture "!!!!!!!!"
  read

  if upper(prvar) = "F"         &&  Specific parameters for file output
    @ 7,20 say "Sending Data to File SRCOUNT.PRN ..."
    set printer to SRCOUNT.PRN
    set device to print
    pagelength = 4000           &&    Pagelength large so header info used once
  elseif upper(prvar) = "P"     &&  Specific parameters for printer output
```

```
    @ 7,20 say "Printing Results ..."
    set device to print
    pagelength = 56
  else                          && Specific parameters for screen output
    clear
    pagelength = 20
  endif

  do while .not. eof()
    if LOC = 1                  && Output header information
      @ LOC,20 say "Database for "
      @ LOC,33 say dbname
      store LOC+2 to LOC
      @ LOC,1  say "Date"
      @ LOC,10 say "# 1"
      @ LOC,15 say "# 2"
      @ LOC,20 say "# 3"
      @ LOC,25 say "# 4"
      @ LOC,30 say "# 5"
      @ LOC,35 say "NSC"
      @ LOC,40 say "Total"
      @ LOC,50 say "Cum Total"
      @ LOC+1,1 say "----------------------------------------------------------"
      store LOC + 2 to LOC
    endif

    @ LOC,1  say CAL_DATE         && Output summary database information
    @ LOC,10 say SEV_CODE_1
    @ LOC,15 say SEV_CODE_2
    @ LOC,20 say SEV_CODE_3
    @ LOC,25 say SEV_CODE_4
    @ LOC,30 say SEV_CODE_5
    @ LOC,35 say NO_SEV_CODE
    @ LOC,40 say TOT_NUM
    @ LOC,50 say TOTAL
    store LOC + 1 to LOC

    if LOC = pagelength         && Reset for beginning of new page
      store 1 to LOC
      if upper(prvar) = "S"
        wait "Hit any key to continue ..."
        clear
      endif
    endif

    skip
  enddo

  if upper(prvar) = "P"         && Reset all parameters for printer and file
    eject
    set device to screen
  elseif upper(prvar) = "F"
    set device to screen
    set printer to              && reset if used for file output
  else                          && If screen output, pause for one last look
    wait "Hit any key to continue ..."
  endif

  close all
endif
*-------------------------------------------------------------------------------
*     If COUNT.DBF did not exist:

else
  @ 23,10 say "COUNT.DBF Does Not Exist."
  wait "Hit any key to continue ..."
endif
*-------------------------------------------------------------------------------
return                          && to SRSAS.PRG main menu
********************************************************************************
```

```
*****************************************************************************
*                                                                           *
*  Title    : Software Reliability TIME.DBF Module (SRTIME.PRG)             *
*  Version  : 3.3                                                           *
*  Date     : 2 Oct 91                                                      *
*  Author   : Capt Joseph J. Stanko                                         *
*  Security : Unclassified                                                  *
*  Purpose  : This program:                                                 *
*             Creates a TIME.DBF data base if needed.                       *
*             Determines the initial time statistics from the summary       *
*             COUNT.DBF and either average test durations, actual test      *
*             durations, or estimated test durations.                       *
*                                                                           *
*  Theory   : This module allows the user to create the time database, if it *
*             does not exist, from both COUNT.DBF and other user/file input *
*             information.                                                  *
*                                                                           *
*  Database : This module uses two databases (see note below):              *
*                                                                           *
*             COUNT.DBF    -  This is an intermediate summary database of   *
*                             dates and number of failures:                 *
*                                                                           *
*               Name     Type  Length  Decimal         Description          *
*             ---------  ----  ------  -------  --------------------------- *
*             CAL_DATE   Date                   Date of occurrence of failure *
*             SEV_CODE_1  N      4              # of Severity Code 1 failures *
*             SEV_CODE_2  N      4              # of Severity Code 2 failures *
*             SEV_CODE_3  N      4              # of Severity Code 3 failures *
*             SEV_CODE_4  N      4              # of Severity Code 4 failures *
*             SEV_CODE_5  N      4              # of Severity Code 5 failures *
*             NO_SEV_CODE N      4              # of failures not coded     *
*             TOT_NUM     N      4              Total Number for this date  *
*             TOTAL       N      4              Overal total of failures    *
*                                                                           *
*             TIME.DBF     -  This is a final database of dates and estimated *
*                             failure times and test durations:             *
*                                                                           *
*               Name     Type  Length  Decimal         Description          *
*             --------   ----  ------  -------  --------------------------- *
*             CAL_DATE   Date                   Date of occurrence of failure *
*             L_TIME_OCC  N      10       2     "Local" time of failure occur *
*                                               (wrt to start of that test) *
*             TEST_DUR    N      10       2     Duration of test for that day *
*             T_TIME_OCC  N      10       2     "Total" time of failure occur *
*                                               (wrt to all total test time) *
*             TOTAL       N      4              Total failures to that point *
*                                                                           *
*             Note: An additional database is used to input the B-1B flight *
*             test data:                                                    *
*                                                                           *
*             B1DATA.DBF   -  This is a summary database of B-1B flight test *
*                             hours and dates:                              *
*                                                                           *
*               Name     Type  Length  Decimal         Description          *
*             --------   ----  ------  -------  --------------------------- *
*             DATE       Date                   Date of mission flown       *
*             FLT_HRS     N      7        2     Mission duration in hours   *
*             FLT         C      6              Mission identifier          *
*                                                                           *
*  Modules:   This program calls the following modules:                     *
*             SRTBUILD.PRG  - Creates the structure for the TIME.DBF if one *
*                             does not already exist.                       *
*             SRTDATE.PRG   - Generates the TIME.DBF based on the assumption *
*                             that failure dates from COUNT.DBF are the only *
*                             test dates, and uses an average test duration *
*                             input from the user.                          *
*             SRTB1.PRG     - Generates the TIME.DBF from specific B-1B     *
*                             flight test data and the COUNT.DBF.           *
*             SRTEST.PRG    - Generates the TIME.DBF from estimates of total *
*                             test time per month and the COUNT.DBF.        *
*                                                                           *
*****************************************************************************

clear screen
```

```
*------------------------------------------------------------------------------
*    Variable Section:
timeoption = "C"              && memvar for main menu
*------------------------------------------------------------------------------
*    Main Loop:
do while upper(timeoption) <> "R"
  @  0,0  clear
  @  3,12 say "Software Reliability Statistical Analysis System (SRSAS)"
  @  4,12 say "              Generate TIME.DBF Module"
  @  6,20 say "C - Create Time Data Base Structure"
  @  8,20 say "D - Use Failure Dates & Average Test Duration for Data"
  @ 10,20 say "B - Use B-1B Flight Test Data for Data"
  @ 12,20 say "E - Use Estimated Test Time per Month for Data"
  @ 14,20 say "R - Return"
  @ 20,20 say "Please Enter Option:";
          get timeoption picture "@K !" valid(timeoption$"CDBER")
  read
  do case                     &&  Call srt programs based on menu input:
    case upper(timeoption)="C"
      do srtbuild
    case upper(timeoption)="D"
      do srtdate
    case upper(timeoption)="B"
      do srtb1
    case upper(timeoption)="E"
      do srtest
    case upper(timeoption)="R"
      note : returning to main program ...
    otherwise  && standard exception handling
      @ 23,20 say "Invalid Entry - Please Use C,D,B,E, or R"
  endcase
enddo
*------------------------------------------------------------------------------
return
*******************************************************************************
```

```
*****************************************************************************
*                                                                           *
*  Title    : Software Reliability TIME.DBF Build Module (SRTBUILD.PRG)     *
*  Version  : 3.3                                                           *
*  Date     : 25 Oct 91                                                     *
*  Author   : Capt Joseph J. Stanko                                         *
*  Security : Unclassified                                                  *
*  Purpose  : This program:                                                 *
*             Creates the structure for both TIME.DBF and TIMEDTE.DBF.      *
*                                                                           *
*  Theory   : The program creates the necessary database structure if it does *
*             not already exist.                                            *
*                                                                           *
*  Database : This program creates the following database structure:        *
*                                                                           *
*             TIMEDTE.DBF  -  This is the DTE version of TIME.DBF to find the *
*                             initial failure intensity for OT&E calculation. *
*                             It has the identical structure to TIME.DBF.   *
*                                                                           *
*             TIME.DBF     -  This is a final database of dates and estimated *
*                             failure times and test durations:             *
*                                                                           *
*               Name      Type  Length  Decimal        Description          *
*                                                                           *
*             --------    ----  ------  -------  -----------------------------  *
*             CAL_DATE    Date                   Date of occurrence of failure *
*             L_TIME_OCC   N      10       2     "Local" time of failure occur *
*                                                (wrt to start of that test)  *
*             TEST_DUR     N      10       2     Duration of test for that day *
*             T_TIME_OCC   N      10       2     "Total" time of failure occur *
*                                                (wrt to all total test time)  *
*             TOTAL        N       4             Total failures to that point  *
*                                                                           *
*  Modules  : Calls procedure DBBUILD (see below).                          *
*                                                                           *
*****************************************************************************
store "O" to dbvar

@ 23,20 say "(D)T&E or (O)T&E Database?" ;
        get dbvar picture "@K !" valid(dbvar$"DO")
read

if upper(dbvar) = "D"
  if .not. file("TIMEDTE.DBF")
    @ 23,20 say "Building DT&E TIME Database ...          "
    do dbbuild
    file = "TIMEDTE.DBF"
    create &file. from template
    delete file template.dbf
  endif
else
  if .not. file("TIME.DBF")
    @ 23,20 say "Building TIME Database ...          "
    do dbbuild
    file = "TIME.DBF"
    create &file. from template
    delete file template.dbf
  endif
endif
*----------------------------------------------------------------------------

return                         && to srtime.prg module
*============================================================================
*     Procedure Section:
*****************************************************************************
*                                                                           *
*  Procedure: Database Build Procedure (DBBUILD)                            *
*  Version  : 3.3                                                           *
*  Date     : 23 Oct 91                                                     *
*  Author   : Capt Joseph J. Stanko                                         *
*  Security : Unclassified                                                  *
*  Purpose  : This module has the implementation code for creating          *
*             the structure for either TIMEDTE.DBF or TIME.DBF              *
*                                                                           *
*  Database : This program creates the following database structure:        *
```

```
*                                                                                *
*            TIME.DBF     -  This is a final database of dates and estimated    *
*                             failure times and test durations:                 *
*                                                                                *
*              Name      Type  Length  Decimal         Description              *
*            --------    ----  ------  -------  ---------------------------- *
*            CAL_DATE    Date                    Date of occurrence of failure *
*            L_TIME_OCC   N      10       2     "Local" time of failure occur *
*                                               (wrt to start of that test)   *
*            TEST_DUR     N      10       2     Duration of test for that day *
*            T_TIME_OCC   N      10       2     "Total" time of failure occur *
*                                               (wrt to all total test time)  *
*            TOTAL        N       4             Total failures to that point  *
*                                                                                *
*  Modules  : N/A                                                                *
*                                                                                *
**********************************************************************************
procedure dbbuild

create template
use template
append blank
replace field_name with "Cal_Date",       field_type with "DATE"
append blank
replace field_name with "L_Time_Occur",   field_type with "N",     ;
        field_len  with 10,                field_dec  with 2
append blank
replace field_name with "Test_Dur",       field_type with "N",     ;
        field_len  with 10,                field_dec  with 2
append blank
replace field_name with "T_Time_Occur",   field_type with "N",     ;
        field_len  with 10,                field_dec  with 2
append blank
replace field_name with "Total",          field_type with "N",     ;
        field_len  with 4
go top
close all

return   && to procedure SRTBUILD
**********************************************************************************
```

```
******************************************************************************
*                                                                            *
*  Title    : Software Reliability TIME.DBF Date Module (SRTDATE.PRG)        *
*  Version  : 3.3                                                            *
*  Date     : 2 Oct 91                                                       *
*  Author   : Capt Joseph J. Stanko                                          *
*  Security : Unclassified                                                   *
*  Purpose  : This program:                                                  *
*             Generates the data for the TIME.DBF database from average test *
*             times.                                                         *
*                                                                            *
*  Theory   : The program generates TIME.DBF data from the use of average test*
*             times assumed to occur ONLY ON THE DATES OF FAILURES as found  *
*             in the COUNT.DBF and SYSTERR.DBF databases.  This assumption is *
*             valid if testing occurred only on the dates that failures were *
*             identified; however, as failures are often "boarded" by a panel *
*             and recognized at dates that could be different than actual    *
*             test dates, another option should be used.                     *
*                                                                            *
*             This module was used for initial analysis of data until more   *
*             definitive test times and durations were available.            *
*                                                                            *
*  Database : This program uses the following database:                      *
*                                                                            *
*             TIME.DBF     -  This is a final database of dates and estimated *
*                             failure times and test durations:              *
*                                                                            *
*               Name      Type  Length  Decimal          Description         *
*             --------    ----  ------  -------   -------------------------- *
*             CAL_DATE    Date                    Date of occurrence of failure *
*             L_TIME_OCC   N      10       2      "Local" time of failure occur *
*                                                 (wrt to start of that test)  *
*             TEST_DUR     N      10       2      Duration of test for that day *
*             T_TIME_OCC   N      10       2      "Total" time of failure occur *
*                                                 (wrt to all total test time)  *
*             TOTAL        N       4              Total failures to that point *
*                                                                            *
*  Modules  : N/A                                                            *
*                                                                            *
******************************************************************************

*----------------------------------------------------------------------------
*     First, check to see if the TIME.DBF exists:

if file("TIME.DBF")

  *---------------------------------------------------------------------------
  * Variable Section:

  use COUNT alias COUNT         &&  Database of failure COUNT data
  select 2
  use TIME alias TIME           &&  Database of failure TIME data
  select COUNT

  store    0 to d_offset        &&  Day offset to determine total test time
  store 3600 to day_val         &&  Day value for test duration (minutes)
  store    0 to m_e             &&  Total number of failures
  store    0 to max_dur         &&  Max partition for assigning failure times
  store    0 to mtestdur        &&  Local value for test duration
  store    0 to my_tot          &&  Local total number of failures
  store    0 to num_sec         &&  Number of seconds from system clock
  store    0 to p_offset        &&  Partition offset for local failure times
  store CAL_DATE to mdate       &&  Local date for failure occurrence
  store CAL_DATE to strtdate    &&  Starting date for data analysis
  go bottom
  store CAL_DATE to enddate     &&  Ending date for data analysis
  go top
  *---------------------------------------------------------------------------
  * Data Entry Section:

  set confirm on
  @ 0,0  clear
  @ 3,10 say "Enter Starting Date for Data   :" get strtdate picture "99/99/99"
  @ 5,10 say "Enter Ending Date for Data     :" get enddate picture "99/99/99"
  @ 7,10 say "Enter Daily Test Duration (min):" get day_val picture "999999"
```

```
  read
  set confirm off
  *------------------------------------------------------------------------
  *  Data Calculation Section:

  locate for CAL_DATE = strtdate
  do while (.not. eof()) .and. (CAL_DATE <= enddate)
    @ 9,10 say "Generating data ..."
    store CAL_DATE to mdate
    store TOT_NUM to my_tot
    store 0 to p_offset
    store day_val to mtestdur
    max_dur = (mtestdur) / my_tot
    select TIME
    for loop_var = 1 to my_tot
      @ 15,10 say "Data Point # "
      @ 15,24 say loop_var
      append blank
      replace CAL_DATE with mdate
      replace TEST_DUR with mtestdur
      *------------------------------
      *  My version of a random number generator.
      *  Takes the system time and finds a value for
      *  the local offset of failure occurrence within
      *  a time "window" by using sqrt() and modulo:

      num_sec = seconds()
      do while num_sec > max_dur
        num_sec = num_sec % sqrt(num_sec)     && % is the modulus operator
      enddo
      *------------------------------
      *  Estimate time of failure from number of partitions, duration
      *  of partitions, and time offset:

      failtime = (p_offset*max_dur) + (num_sec)
      replace L_TIME_OCCUR with failtime            && Local Time of Occurrence

      replace T_TIME_OCCUR with failtime + d_offset && Total Time of Occurrence

      m_e = m_e + 1
      replace TOTAL with m_e                        && Total Failures

      p_offset = p_offset +1                        && Move to next partition
    next
    d_offset = d_offset + day_val                   && Move to next test period
    select COUNT
    skip
  enddo
  close all
*--------------------------------------------------------------------------
*  Else, database does not exist:

else
  @ 23,10 say "TIME.DBF Does Not Exist."
  wait "Hit any key to continue ..."
endif
*--------------------------------------------------------------------------

return                         && to srtime.prg module.
***************************************************************************
```

```
******************************************************************************
*                                                                            *
*  Title    : Software Reliability TIME.DBF B-1B Module (SRTB1.PRG)          *
*  Version  : 3.3                                                            *
*  Date     : 2 Oct 91                                                       *
*  Author   : Capt Joseph J. Stanko                                          *
*  Security : Unclassified                                                   *
*  Purpose  : This program:                                                  *
*             Generates TIME.DBF data from the summary COUNT.DBF database    *
*             and the specific B-1B flight test database B1DATA.DBF.         *
*             B1DATA.DBF.                                                    *
*             As this uses a specific database, it also allows the user to   *
*             print the B1SDATA.DBF (sorted version of B1DATA.DBF).          *
*                                                                            *
*  Theory   : This is a one pass program that generates failure times from   *
*             actual test durations.  It was a little involved, as there were *
*             instances of COUNT.DBF dates that had no corresponding flight  *
*             test times, as well as B1DATA.DBF dates that had no            *
*             corresponding failure occurrences.  This required a "running   *
*             summation" of either failures or test times until one caught up *
*             with the other.  Once dates for both failures and times were   *
*             the same, that was considered the date of failure (for this    *
*             database only) with the time of failure assigned within the    *
*             total test time for that date.  While this might not take into *
*             account the possibility that failures were discovered on       *
*             previous flights, it does preserve the relationship of test    *
*             durations and intervals to occurrence of failures (in a        *
*             relative manner).                                              *
*                                                                            *
*  Database : This program uses three databases:                             *
*                                                                            *
*             B1DATA.DBF    -  This is a summary database of B-1B flight test *
*                              hours and dates:                              *
*                                                                            *
*               Name     Type  Length  Decimal           Description         *
*             --------   ----  ------  -------  ---------------------------- *
*             DATE       Date                   Date of mission flown        *
*             FLT_HRS     N       7        2    Duration of mission (in hours)*
*             FLT         C       6             Mission identifier           *
*                                                                            *
*             COUNT.DBF     -  This is an intermediate summary database of   *
*                              dates and number of failures:                 *
*                                                                            *
*               Name     Type  Length  Decimal           Description         *
*             --------   ----  ------  -------  ---------------------------- *
*             CAL_DATE   Date                   Date of occurrence of failure *
*             SEV_CODE_1  N       4             # of Severity Code 1 failures *
*             SEV_CODE_2  N       4             # of Severity Code 2 failures *
*             SEV_CODE_3  N       4             # of Severity Code 3 failures *
*             SEV_CODE_4  N       4             # of Severity Code 4 failures *
*             SEV_CODE_5  N       4             # of Severity Code 5 failures *
*             NO_SEV_CODE N       4             # of failures not coded      *
*             TOT_NUM     N       4             Total Number for this date   *
*             TOTAL       N       4             Overal total of failures     *
*                                                                            *
*             TIME.DBF      -  This is a final database of dates and estimated *
*                              failure times and test durations:             *
*                                                                            *
*               Name     Type  Length  Decimal           Description         *
*             --------   ----  ------  -------  ---------------------------- *
*             CAL_DATE   Date                   Date of occurrence of failure *
*             L_TIME_OCC  N      10        2    "Local" time of failure occur *
*                                               (wrt to start of that test)  *
*             TEST_DUR    N      10        2    Duration of test for that day *
*             T_TIME_OCC  N      10        2    "Total" time of failure occur *
*                                               (wrt to all total test time) *
*             TOTAL       N       4             Total failures to that point *
*                                                                            *
*  Modules  : N/A                                                            *
*                                                                            *
******************************************************************************

*-----------------------------------------------------------------------------
*     First, make sure the data is sorted:
```

```
clear screen

if .not. file("B1SDATA.DBF")
  @ 3,10 say "Sorting the B1 Data ..."
  use B1DATA
  sort on DATE to temp1
  close all
  rename temp1.dbf to B1SDATA.dbf
endif
*-----------------------------------------------------------------------------
*     Then, check to see if user wants the data printed:
store "N" to prvar
@ 5,10 say "Would you like to print sorted B-1B data (Y/N)?" ;
       get prvar picture "@K !" valid(prvar$"YN")
read
if upper(prvar) = "Y"
  use B1SDATA
  store 1 to LOC
  store "          " to dbname
  store "s" to printvar
  @ 9,10 say "What is DB name?"  get dbname picture "!!!!!!!!"
  read
  @ 11,10 say "Send to (S)creen, (P)rinter, or (F)ile?" ;
          get printvar picture "@K !" valid(printvar$"SPF")
  read
  if upper(printvar) = "P"
    set device to print
    pagelength = 56
  elseif upper(printvar) = "F"
    set printer to SRB1DATA.PRN
    set device to print
    pagelength = 4000
  else
    clear
    pagelength = 20
  endif
  @ 13,10 say "Printing results ..."
  do while .not. eof()
    if LOC = 1
      @ LOC,20 say "Database for "
      @ LOC,33 say dbname
      store LOC+2 to LOC
      @ LOC,10 say "Date"
      @ LOC,21 say "Flt Hrs"
      @ LOC,30 say "Flt Num"
      @ LOC+1,10 say "---------------------------"
      store LOC + 2 to LOC
    endif
    @ LOC,10 say DATE
    @ LOC,21 say FLT_HRS
    @ LOC,30 say FLIGHT
    store LOC + 1 to LOC
    if LOC = pagelength
      store 1 to LOC
      if upper(printvar) = "S"
        wait "Hit any key to continue ..."
        clear
      endif
    endif
    skip
  enddo
  if upper(printvar) = "P"
    set device to screen
  elseif upper(printvar) = "F"
    set device to screen
    set printer to
```

```
  else
    wait "Hit any key to continue ..."
  endif
  close all
endif
*------------------------------------------------------------------------
*     Now generate TIME.DBF data:
clear screen
*------------------------------------------------------------------------
*     First, check to see if the TIME.DBF exists:
if file("TIME.DBF")

  *----------------------------------------------------------------------
  *  Variable Section:
  use COUNT alias COUNT        &&  Database of failure COUNT data
  select 2
  use TIME alias TIME          &&  Database of failure TIME data
  select 3
  use B1SDATA alias B1         &&  Database of test time and duration data
  select COUNT
  store    0 to d_offset       &&  Day offset to determine total test time
  store    0 to m_e            &&  Total number of failures
  store    0 to max_fail       &&  Test var for failures with no test times
  store    0 to mtestdur       &&  Local value for test duration
  store    0 to my_tot         &&  Local total number of failures
  store    0 to num_sec        &&  System time (sec) for random failure times
  store    0 to p_offset       &&  Partition offset for failure times
  store CAL_DATE to mbldate    &&  Local date for flt test occurrence
  store CAL_DATE to mdate      &&  Local date for failure occurrence
  store CAL_DATE to strtdate   &&  Starting date for data analysis
  go bottom
  store CAL_DATE to enddate    &&  Ending date for data analysis
  go top
  *----------------------------------------------------------------------
  *  Data Entry Section:
  set confirm on
  @ 3,10 say "Enter Starting Date for Data    :" get strtdate picture "99/99/99"
  @ 4,10 say "Enter Ending Date for Data      :" get enddate picture "99/99/99"
  read
  set confirm off
  *----------------------------------------------------------------------
  *  Data Calculation Section:
  locate for CAL_DATE >= strtdate
  select B1
  locate for DATE >= strtdate
  select COUNT
  do while (.not. eof()) .and. (CAL_DATE <= enddate)
    @ 7,10 say "Generating data ..."
    store CAL_DATE to mdate
    store TOT_NUM to my_tot
    select B1
      store 0 to mtestdur
      store 0 to max_fail
      *-----------------------------
      *  The order of these next conditionals acts like a filter to
      *  synch the test dates and failure dates.
      *-----------------------------
      *  First, check to see if there is a flt record for corresponding
      *  failure date.  If not, then add up total failures until we
      *  get to or pass the next flt record:
      if (DATE > mdate)
        store DATE to mbldate
        select COUNT
        do while (CAL_DATE < mbldate) .and. (.not. eof())
          skip
          store (TOT_NUM + my_tot) to my_tot
```

```
  enddo
  store CAL_DATE to mdate
  select B1
endif
*------------------------------
*  Then, check to see if the failure date is past the flt record.
*  If so, add up flight times for interval offset value until
*  we get to or pass the next failure date record:

do while (DATE < mdate) .and. (.not. eof())
  store (FLT_HRS*60)+d_offset  to d_offset
  skip
enddo
*------------------------------
*  If we pass the failure date again, use the previous flt record
*  test time for test duration (must decrement the day offset
*  by the test duration so it's not used twice):

if (DATE > mdate)
  skip -1
  store (d_offset - (FLT_HRS*60)) to d_offset
  store (FLT_HRS*60)+mtestdur to mtestdur
  skip
endif
*------------------------------
*  If we did not pass the failure date again, than the dates
*  must be equal.
*  Add up multiple test durations for the same day to make sure
*  the entire same day test duration is used for failure times:

do while (DATE = mdate) .and. (.not. eof())
  store (FLT_HRS*60)+mtestdur to mtestdur
  skip
enddo
*------------------------------
*  This is an error check to make sure there are test times for
*  the failures:

max_fail = (mtestdur) / my_tot
if max_fail = 0
  @ 20,10 say "******* TEST DURATION = 0 *******"
endif
*------------------------------
*  Now that we've made it this far, assign the failure times randomly
*  (assuming a normal distribution for ease of calculation) within
*  the test duration:

select TIME
store 0 to p_offset
for loop_var = 1 to my_tot
  @ 15,10 say "Making Entry "
  @ 15,24 say loop_var picture "99"
  @ 15,27 say "of "
  @ 15,31 say my_tot
  append blank
  if mdate ="  /  /  "
    replace CAL_DATE with enddate        && We exceeded the end of file
  else
    replace CAL_DATE with mdate
  endif
  replace TEST_DUR with mtestdur
  *--------------------------
  *  My random number generator:

  num_sec = seconds()
  do while (num_sec/60) > max_fail
    num_sec = num_sec % sqrt(num_sec)     && % is the modulus operator
  enddo

  failtime = (p_offset*max_fail) + (num_sec/60)
  replace L_TIME_OCCUR with failtime           && Local Occurrence Time
```

```
        replace T_TIME_OCCUR with failtime + d_offset && Total Occurrence Time
        m_e = m_e + 1
        replace TOTAL with m_e                       && Total Failures
        p_offset = p_offset +1
      next
      d_offset = d_offset + mtestdur
      select COUNT
      skip
    enddo
*-------------------------------------------------------------------------------
*     If TIME.DBF does not exist:
else
  @ 23,10 say "TIME.DBF Does Not Exist."
  wait "Hit any key to continue ..."
endif
*-------------------------------------------------------------------------------
return                         && to srtime.prg module.
********************************************************************************
```

```
*****************************************************************************
*                                                                           *
*  Title    : Software Reliability TIME.DBF Estimate Module (SRTEST.PRG)    *
*  Version  : 3.3                                                           *
*  Date     : 25 Oct 91                                                     *
*  Author   : Capt Joseph J. Stanko                                         *
*  Security : Unclassified                                                  *
*  Purpose  : This program:                                                 *
*             Generates TIME.DBF data from the summary COUNT.DBF database   *
*             and estimated test times for monthly periods.  Generates      *
*             TIMEDTE.DBF data the same way (used to determine the failure  *
*             intensity at end of DT&E/start of OT&E).                      *
*                                                                           *
*  Theory   : This is a one pass program that generates failure times from  *
*             estimated test durations.  The user is prompted for number of *
*             months, then the program iterates for each month asking the   *
*             user for the estimated test time for that month.  The program *
*             then accesses the COUNT.DBF database, and locates the records *
*             for failures occurring during that month.  The estimated test *
*             time is divided by the number of days in the month, and the   *
*             times are summed up to each date of failure data for local    *
*             values of test duration.  For example, if there were 30 hrs   *
*             estimated for September, that would be (assuming the standard *
*             normal distribution again) an average of 1 hr a day testing.  *
*             While this is probably not that accurate, taking the COUNT.DBF *
*             data of failures and summing up to the failure dates (in this *
*             case, they could be 09/11/89, 09/15/89, and 09/22/89) that    *
*             would give us 3 test durations of 11 hours, 4 hours, and      *
*             7 hours, with the additional 8 hours rounded into the offset  *
*             for the following month's first test duration.                *
*             This is the best I can do as I am working with summary data. *
*             Praise the Lord Jesus Christ!                                 *
*                                                                           *
*  Database : This program uses two databases:                              *
*                                                                           *
*             COUNT.DBF     -  This is an intermediate summary database of  *
*                              dates and number of failures:                *
*                                                                           *
*               Name      Type  Length  Decimal          Description        *
*             --------    ----  ------  -------  -------------------------- *
*             CAL_DATE    Date                   Date of occurrence of failure *
*             SEV_CODE_1   N      4              # of Severity Code 1 failures *
*             SEV_CODE_2   N      4              # of Severity Code 2 failures *
*             SEV_CODE_3   N      4              # of Severity Code 3 failures *
*             SEV_CODE_4   N      4              # of Severity Code 4 failures *
*             SEV_CODE_5   N      4              # of Severity Code 5 failures *
*             NO_SEV_CODE  N      4              # of failures not coded     *
*             TOT_NUM      N      4              Total Number for this date  *
*             TOTAL        N      4              Overal total of failures    *
*                                                                           *
*             TIMEDTE.DBF   -  This is DT&E database of failure time.  Used *
*                              as basis for OT&E failure intensity.  Same   *
*                              structure as TIME.DBF.                       *
*                                                                           *
*             TIME.DBF      -  This is a final database of dates and estimated *
*                              failure times and test durations:            *
*                                                                           *
*               Name      Type  Length  Decimal          Description        *
*             --------    ----  ------  -------  -------------------------- *
*             CAL_DATE    Date                   Date of occurrence of failure *
*             L_TIME_OCC   N     10        2     "Local" time of failure occur *
*                                                (wrt to start of that test) *
*             TEST_DUR     N     10        2     Duration of test for that day *
*             T_TIME_OCC   N     10        2     "Total" time of failure occur *
*                                                (wrt to all total test time) *
*             TOTAL        N      4              Total failures to that point *
*                                                                           *
*  Modules  : N/A                                                           *
*                                                                           *
*****************************************************************************
*---------------------------------------------------------------------------
```

```
*     Determine the TIME.DBF needed:
store "O" to dbvar
@ 23,20 say "(D)T&E or (O)T&E Database?" ;
        get dbvar picture "@K !" valid(dbvar$"DO")
read
@ 23,20 say "                                       "
*-------------------------------------------------------------------------------
*     First, check to see if either TIME.DBF or TIMEDTE.DBF exists:
if (file("TIME.DBF")    .and. upper(dbvar)="O") .or. ;
   (file("TIMEDTE.DBF") .and. upper(dbvar)="D")
  *-----------------------------------------------------------------------------
  *  Variable Section:
  use COUNT alias COUNT         &&  Database of failure COUNT data
  select 2
  if upper(dbvar)="O"
    use TIME alias TIME         &&  Database of OT&E failure TIME data
  else                          &&  upper(dbvar)="D"
    use TIMEDTE alias TIME      &&  Database of DT&E failure TIME data
  endif
  select COUNT
  store    0 to avg_time        &&  Average test time per month for OT&E (hrs)
  store    0 to d_offset        &&  Day offset to determine total test time
  store    0 to day_val         &&  Day value for test duration (minutes)
  store   60 to hour            &&  Number of minutes in an hour
  store    0 to last_month      &&  Carry over time from previous test month
  store    0 to m_e             &&  Total number of failures
  store    0 to m_offset        &&  Month offset to determine total test time
  store    0 to max_dur         &&  Max partition for assigning failure times
  store  "A" to mode_var        &&  Mode for diagnostic write output
  store    0 to month_end       &&  Last day of month (varies from 28 to 31)
  store    0 to mtestdur        &&  Local value for test duration (min)
  store    0 to my_tot          &&  Local total number of failures
  store    0 to num_days        &&  Number of days used to calculate mtestdur
  store    0 to num_month       &&  Total number of months for OT&E test
  store    0 to num_sec         &&  Number of seconds from system clock
  store    0 to p_offset        &&  Partition offset for local failure times
  store CAL_DATE to mdate       &&  Local date for failure occurrence
  store CAL_DATE to strtdate    &&  Starting date for data analysis
  go bottom
  store CAL_DATE to enddate     &&  Ending date for data analysis
  go top
  *-----------------------------------------------------------------------------
  *  Data Entry Section:
  clear screen
  set confirm on
  @ 3,10 say "Enter Number of Months for Test:" get num_month picture "99"
  @ 4,10 say "Enter Starting Date for Data   :" get strtdate picture "99/99/99"
  @ 5,10 say "Enter Ending Date for Data     :" get enddate picture "99/99/99"
  read
  @ 6,10 say "(A)uto or (S)ingle Step Mode?" get mode_var picture "!"
  read
  set confirm off
  *-----------------------------------------------------------------------------
  *  Data Calculation Section:
  locate for CAL_DATE = strtdate     && Go to the first applicable record
  store CAL_DATE to mdate            && Update mdate to match strtdate
  for loop_var = 1 to num_month      && I assume the first month is in COUNT.DBF
    @ 10,10 say "                          "
    @ 7,10 say "Working on Test Month #"
    @ 7,33 say loop_var
    if loop_var < num_month          && Get appropriate input
      set confirm on
      @ 8,10 say "Enter Avg Test Time/Month (hrs):";
             get avg_time  picture "9999.99"
      read
```

```
    set confirm off              && Put this inside both or get 2 in fields
    @ 8,8 clear to 8,78
  else
*   clear gets                   && Remove get from above
    set confirm on
    @ 8,10 say "Enter Final Month's Test Time (hrs):";
           get avg_time picture "9999.99"
    read
    set confirm off
  endif
  @ 10,10 say "Generating data ..."
  *------------------------------
  *    Check to see if the next month in COUNT.DBF is a consecutive
  *    month of testing, including wrap-around (0 is reset condition):

  m_offset = month(CAL_DATE)-month(mdate)
  if (m_offset=0) .or. (m_offset=1) .or. ;
     ((month(CAL_DATE)=1) .and. (month(mdate)=12))
    *------------------------------
    *    Determine the average daily test time and perform all
    *    calculations for the TIME.DBF database:

    do case                        && Assign daily average test time
      case (month(CAL_DATE)=4) .or. (month(CAL_DATE)=6) ;
      .or. (month(CAL_DATE)=9) .or. (month(CAL_DATE)=11)
        month_end = 30             && April, June, September, and November
      case (month(CAL_DATE)=2)
        if (year(CAL_DATE)%4)=0    && Check for Leap Year
          month_end = 29           &&   February has 29 days
        else
          month_end = 28           &&   February has 28 days
        endif
      otherwise
        month_end = 31             && All the others have 31 days
    endcase
    day_val = (avg_time*hour)/month_end

    @ 16,10 say "Daily Test Time = "    &&  Echo the information
    @ 16,28 say day_val
    *------------------------------
    *    For each entry in COUNT.DBF within the same month:

    store day(CAL_DATE) to num_days  &&  Number days used for mtestdur
    store CAL_DATE to mdate          &&  Reset date for failure occurrence
    store 0 to prev_days             &&  Initialize each month

    do while (month(CAL_DATE) = month(mdate));
       .and. (.not. eof())

      if ((day(CAL_DATE)-num_days)<>0)  && Determine # of days of test
        num_days = day(CAL_DATE)-prev_days
      endif
      store day(CAL_DATE) to prev_days

      mtestdur = (day_val * num_days) &&+ last_month
      store CAL_DATE to mdate       &&  Update for change in day
      store TOT_NUM to my_tot
      max_dur  = mtestdur / my_tot  &&  Set maximum partition time duration

      @ 17,10 say "Num Days = "    &&  Output the calculation data
      @ 17,21 say num_days         &&    to verify that it works
      @ 18,10 say "Prev Days= "
      @ 18,21 say prev_days
      @ 19,10 say "MTESTDUR = "
      @ 19,21 say mtestdur
      @ 20,10 say "Max Dur  = "
      @ 20,21 say max_dur
      @ 21,10 say " "
      if mode_var = "S"
        wait "Hit any key to continue ..."
      endif
      *------------------------------
```

```
      *  Now that we've got the test duration and number of failures,
      *  assign times (assuming normal distribution for ease of calculation)
      *  within the test duration:
      select TIME
      store 0 to p_offset
      for loop2_var = 1 to my_tot
        @ 15,10 say "Making Entry "
        @ 15,24 say loop2_var picture "99"
        @ 15,27 say "of "
        @ 15,31 say my_tot
        append blank
        replace CAL_DATE with mdate
        replace TEST_DUR with mtestdur
        *-------------------------
        *  My random number generator:
        num_sec = seconds()
        do while (num_sec/60) > max_dur
          num_sec = num_sec % sqrt(num_sec)     && % is the modulus operator
        enddo
        if num_sec = 0                          && Check for 0 time interval
          num_sec = 60                          &&  and set to a min value
        endif
        failtime = (p_offset*max_dur) + (num_sec/60)
        replace L_TIME_OCCUR with failtime     && Local Occurrence Time
        replace T_TIME_OCCUR with failtime+d_offset  && Total Occurrence Time
        m_e = m_e + 1
        replace TOTAL with m_e                  && Total Failures
        p_offset = p_offset +1
      next
      d_offset = d_offset + mtestdur
      select COUNT
      skip
    enddo
    *-------------------------------
    *     Check for time at end of month after last COUNT entry:
    if .not. eof()
      skip -1
      if (day(CAL_DATE) < month_end)
        num_days = month_end - day(CAL_DATE)
        d_offset = (num_days*day_val) + d_offset
      endif
      skip
    endif
  *------------------------------
  *     If the months are not consecutive, then include the "between-test"
  *     time as part of the offset:
  else
    if (month(CAL_DATE)>month(mdate))         && ie, 11 > 9
      m_offset=month(CAL_DATE)-month(mdate)-1
    else                                      && ie, 2 /> 12
      m_offset=(12-month(mdate))+(month(CAL_DATE)-1)
    endif
    d_offset = (m_offset*avg_time*hour) + d_offset
    loop_var = loop_var+m_offset-1
    store CAL_DATE to mdate                   &&  Reset for new month's data
  endif
  @ 21,10 say "D_Offset = "    &&  Echo the information
  @ 21,21 say d_offset
  if mode_var = "S"
    wait "Hit any key to continue ..."
  endif
next
```

```
  close all                     && Saves off the database data
*-----------------------------------------------------------------------------
*     Else, neither TIME.DBF nor TIMEDTE.DBF does not exist:
else
  @ 23,20 say "TIME Database Does Not Exist.                 "
  wait "Hit any key to continue ..."
endif
*-----------------------------------------------------------------------------
return                         && to srtime.prg module.
******************************************************************************
```

```
*****************************************************************************
*                                                                           *
*  Title    : Software Reliability Execution Time Module (SREXEC.PRG)       *
*  Version  : 3.3                                                           *
*  Date     : 25 Oct 91                                                     *
*  Author   : Capt Joseph J. Stanko                                         *
*  Security : Unclassified                                                  *
*  Purpose  : This program:                                                 *
*             1.) Performs calculations on data to determine initial        *
*                 parameters for the fitted model.                          *
*             2.) Performs calculations on data to determine goodness-of-fit *
*                 for the Execution Time model.                             *
*                                                                           *
*  Theory   : In order to apply the Execution Time model to the failure data, *
*             initial parameter estimation must be accomplished from the    *
*             overall data.  Once the parameters are calculated, they are   *
*             then used in the model to calculate estimated values (such as *
*             current number of failures and failure intensity) along with the*
*             95% percent confidence intervals.  These estimations are then *
*             compared against the actual data to determine the             *
*             goodness-of-fit.                                              *
*                                                                           *
*  Database : This program uses one database:                               *
*                                                                           *
*             TIME.DBF      -  This is a final database of dates and estimated *
*                              failure times and test durations:            *
*                                                                           *
*               Name      Type  Length  Decimal          Description        *
*             --------    ----  ------  -------   ------------------------- *
*             CAL_DATE    Date                    Date of occurrence of failure *
*             L_TIME_OCC   N      10       2      "Local" time of failure occur *
*                                                 (wrt to start of that test) *
*             TEST_DUR     N      10       2      Duration of test for that day *
*             T_TIME_OCC   N      10       2      "Total" time of failure occur *
*                                                 (wrt to all total test time) *
*             TOTAL        N       4              Total failures to that point *
*                                                                           *
*  Modules  : Calls internal procedures BHATOTE and BHATDTE.                *
*                                                                           *
*****************************************************************************
store "O" to dbvar
@ 23,20 say "(D)T&E or (O)T&E Database?" ;
        get dbvar picture "@K !" valid(dbvar$"DO")
read
@ 23,20 clear

if upper(dbvar) = "O"
  use TIME alias TIME
else
  use TIMEDTE alias TIME
endif
*----------------------------------------------------------------------------
*    Variable Section (Note: mu's and lambda's are defined later on):

go bottom
store "          "    to dbname     && Name of database for output headers
store 0.0000001       to delta      && Accuracy difference for parameters
store "M"             to intervar   && Data output intervals (monthly or daily)
store 0.000           to lambda_0   && Initial failure intensity value
store TOTAL           to m_e        && Total number of failures
store 30              to max_iter   && Max iterations to perform Newton-Raphson
store 1               to num_iter   && Current iteration number for Newton-Raphson
store "S"             to printvar   && Default print option (screen)
store T_TIME_OCCUR to t_e           && Total test time
*----------------------------------------------------------------------------
*    Data Entry Section:

@ 0,0  clear
set confirm on
@ 3,10 say "Enter Total Test Time:" get t_e      picture "9999999.99"
@ 4,10 say "Enter Max # Iteration:" get max_iter picture "999"
```

```
@ 5,10 say "Enter MLE Delta      :" get delta    picture "9.99999999"
if upper(dbvar)="O"
  @ 6,10 say "Enter Initial Failure "
  @ 7,10 say "Intensity (0 for none):" get lambda_0 picture "9.999999999"
endif
read
@ 9,10 say "What is DB Name for Output?"  get dbname picture "!!!!!!!!"
read
set confirm off
@ 11,10 say "Data Output Interval: (M)onthly or (D)aily:" ;
        get intervar picture "@K !" valid(intervar$"MD")
read
@ 13,10 say "Send Data to (S)creen, (P)rinter, or (F)ile:" ;
        get printvar picture "@K !" valid(printvar$"SPF")
read
*-------------------------------------------------------------------------------
*     Data Direction Section:

if upper(printvar) = "F"
  if upper(dbvar) = "O"
    @ 15,10 say "Sending Data to File SREXEC.PRN ..."
    set printer to SREXEC.PRN
  else   && upper(dbvar) = "D"
    @ 15,10 say "Sending Data to File SREXECD.PRN ..."
    set printer to SREXECD.PRN
  endif
  set device to print
  pagelength = 4000
elseif upper(printvar) = "P"
  @ 15,10 say "Printing Data ..."
  set device to print
  pagelength = 55
else
  @ 0,0 clear
  pagelength = 20
endif
*-------------------------------------------------------------------------------
*     Data Calculation Section: Maximum Likelihood Estimation

@ 3,7  say "MLE Calculations for"
@ 3,31 say dbname
@ 3,40 say "using Musa Execution Time Model:"
*-----------------------------
*     Make initial model parameter estimation, and sum failure occurance
*     times to make calculations easier:

b_hat = 1/(t_e)
go top
@ 5,10 say "Total Failures:          m_e   = "
@ 5,43 say m_e picture "99999.99"
@ 6,10 say "Failure Data End Time:   t_e   = "
@ 6,43 say t_e picture "99999999.99"
@ 7,10 say "Initial Model Param Est: b_hat = "
@ 7,43 say b_hat picture "99.999999999"
*-----------------------------
*     Determine a better estimation for b_hat by making f_stat as close
*     as possible to 0 (uses Newton-Raphson method):

if upper(printvar) = "P"
  set device to screen
  @ 17,10 say "Refining b_hat Estimate, Please Wait ..."
  set device to print
elseif upper(printvar) = "S"
  @ 8,10 say "Refining b_hat Estimate, Please Wait ..."
endif
*-----------------------------
*     Iterate while out of tolerance or within allotted looping time:

num_iter = 1
```

```
not_in_tol = .T.
do while (not_in_tol) .and. (num_iter <= max_iter)
  *---------------
  *     Determine the f(b_hat) and f'(b_hat) for Newton-Raphson method
  *     based on known initial value of failure intensity (lambda_0):
  if upper(dbvar) = "O" .and. ; &&  Different equation set for OT&E using
     lambda_0 <> 0.0            &&    previous DT&E failure intensity #
    f_stat = ((m_e*b_hat)/(1-exp(-b_hat*t_e))) - lambda_0
    fp_stat = (((1-exp(-b_hat*t_e))*m_e)-((m_e*b_hat)*(t_e*exp(-b_hat*t_e))));
              / ((1-exp(-b_hat*t_e))^2)
  *---------------
  *     Determine the f(b_hat) and f'(b_hat) for Newton-Raphson method
  *     with no clues at all:
  else                          &&  We're either looking at DT&E data
                                &&    or OT&E data without a priori info
    go top
    t_i = 0                     &&  Summation of failure occur times
    do while .not. eof()
      t_i = t_i + T_TIME_OCCUR
      skip
    enddo
    f_stat = (m_e/b_hat) - ((m_e*t_e)/(exp(b_hat*t_e)-1)) - t_i
    fp_stat = (m_e * (-1/b_hat^2)) -      ;
              (m_e*t_e)*((-1*t_e*exp(b_hat*t_e))/(exp(b_hat*t_e)-1)^2)
  endif
  *---------------
  *     The rest is the same for both cases from above:
  bp_hat = b_hat - (f_stat/fp_stat)  &&  Burden and Faires Step #3
  if abs(bp_hat-b_hat) < delta       &&  Check for within tolerance delta
    not_in_tol = .F.
  endif
  if upper(printvar) = "P"           &&  Output the data as it is calculated
    set device to screen             &&    to verify convergence
    @ 19,0  clear
    @ 19,10 say "b_hat   = "
    @ 19,20 say b_hat
    @ 20,10 say "bp_hat  = "
    @ 20,20 say bp_hat
    @ 21,10 say "F_Stat  = "
    @ 21,20 say f_stat
    @ 22,10 say "Fp_Stat = "
    @ 22,20 say fp_stat
    @ 17,10 say "Refining b_hat Estimate, Please Wait ..."
    set device to print
  elseif upper(printvar) = "S"       &&  Same thing here, but must use
    @  9,0  clear                    &&    different screen output positions.
    @  9,10 say "b_hat   = "
    @  9,20 say b_hat
    @ 10,10 say "bp_hat  = "
    @ 10,20 say bp_hat
    @ 11,10 say "F_Stat  = "
    @ 11,20 say f_stat
    @ 12,10 say "Fp_Stat = "
    @ 12,20 say fp_stat
    @ 8,10 say "Refining b_hat Estimate, Please Wait ..."
  endif
  b_hat = bp_hat
  num_iter = num_iter + 1
enddo
```

```
*--------------------------------------------------------------------------
*     Output additional data on refined values:

if upper(printvar) = "P"
  set device to screen
  @ 19,0  clear
  @ 19,10 say "Printing Data ..."
  set device to print
elseif upper(printvar) = "S"
  @ 9,0  clear
endif

if num_iter > max_iter
  @ 9,10 say "Method Failed After "
  @ 9,30 say max_iter picture "999"
  @ 9,34 say "Iterations."
endif

@ 10,10 say "Final Model Param Est:   b_hat = "
@ 10,43 say b_hat picture "99.999999999"
@ 11,10 say " "
*--------------------------------------------------------------------------
*     Data Calculation Section: Parameters and Confidence Intervals
*     Determine the Expected (Fisher) Information, and then
*     Calculate 95% Confidence Intervals:

@ 12,10 say "Parameter Calculations: nu_0, lambda_0, and 95th Percentile:"

fisher = m_e * ( (1/b_hat^2) - ( (t_e^2*exp(b_hat*t_e))    ;
           / (exp(b_hat*t_e)-1)^2 ))

                                              &&  Initial model parameter
b_hat_low = b_hat - (1.96/sqrt(fisher))       &&  From Z statistic
b_hat_hi  = b_hat + (1.96/sqrt(fisher))

                                              &&  Derived model parameter
b_0     = (m_e) / (1-exp(-(b_hat*t_e)))       &&  Calculated from parameter
b_0_low = (m_e) / (1-exp(-(b_hat_low*t_e)))   &&    b_hat.
b_0_hi  = (m_e) / (1-exp(-(b_hat_hi*t_e)))

                                              &&  Total Failures at t=infinity
nu_0     = b_0                                &&  By Definition
nu_0_low = b_0_low
nu_0_hi  = b_0_hi

                                              &&  Initial Failure Intensity
if lambda_0 = 0                               &&  If we don't have a user
  lambda_0     = b_0 * b_hat                  &&    input, calculate it
endif
lambda_0_low = b_0_low * b_hat_low            &&  Varying b_hat effects both
lambda_0_hi  = b_0_hi * b_hat_hi              &&    b_hat and b_0, etc.

@ 14,10 say "Expected (or Fisher) Value          = "
@ 14,48 say fisher picture "9999999999.999999999"
@ 16,10 say "95% Boundary:               nu_0_low = "
@ 16,48 say nu_0_low picture "99999999.99"
@ 17,10 say "Total Estimated Failures:  nu_0     = "
@ 17,48 say nu_0 picture "99999999.99"
@ 18,10 say "95% Boundary:               nu_0_hi  = "
@ 18,48 say nu_0_hi picture "99999999.99"

@ 20,10 say "95% Boundary:           lambda_0_low = "
@ 20,48 say lambda_0_low picture "99.999999999"
@ 21,10 say "Initial Failure Intensity: lambda_0 = "
@ 21,48 say lambda_0 picture "99.999999999"
@ 22,10 say "95% Boundary:            lambda_0_hi = "
@ 22,48 say lambda_0_hi picture "99.999999999"
@ 23,10 say " "
*--------------------------------------------------------------------------
*     Output of Model Results:

if upper(printvar) = "P"
```

```
  set print on
  eject
  set print off
elseif upper(printvar) = "S"
  wait "Hit any key to continue ..."
  clear
endif
go top
LOC = 1
do while .not. eof()
  if LOC = 1                                    &&  Header information:
    @ LOC,10 say "Generating Plot Data for"
    @ LOC,35 say dbname
    @ LOC+1,5  say "----------------------------------------------"
    @ LOC+1,51 say "------------------------"
    LOC = LOC + 3
  endif
  store CAL_DATE to mdate
  store T_TIME_OCCUR to tau
  store TOTAL to mu
  *-----------------------------
  *     Calculate this info for each pass in the loop:
  *     Failures Experienced at time t=tau
  mu_tau     = nu_0 * (1 - exp(-(lambda_0/nu_0)*tau))
  mu_tau_low = nu_0_low * (1 - exp(-(lambda_0/nu_0_low)*tau))
  mu_tau_hi  = nu_0_hi * (1 - exp(-(lambda_0/nu_0_hi)*tau))
  *     Failure Intensity at time t=tau
  lambda_tau   = lambda_0 * exp(-(lambda_0/nu_0)*tau)
  lambda_t_low = lambda_0_low * exp(-(lambda_0_low/nu_0_low)*tau)
  lambda_t_hi  = lambda_0_hi * exp(-(lambda_0_hi/nu_0_hi)*tau)
  *     Failure Intensity at mu failures experienced
  lambda_mu    = lambda_0 * (1-(mu/nu_0))
  lambda_m_low = lambda_0_low * (1-(mu/nu_0_low))
  lambda_m_hi  = lambda_0_hi * (1-(mu/nu_0_hi))
  *-----------------------------
  *     Now output the info for each pass in the loop:
  @ LOC,10 say "Day : "
  @ LOC,16 say mdate
  @ LOC,30 say "mu(tau) low = "
  @ LOC,44 say mu_tau_low picture "9999.99"
  LOC=LOC+1
  @ LOC,10 say "mu  = "
  @ LOC,16 say mu picture "9999.99"
  @ LOC,30 say "mu(tau)     = "
  @ LOC,44 say mu_tau picture "9999.99"
  LOC=LOC+1
  @ LOC,10 say "tau = "
  @ LOC,16 say tau picture "9999999.99"
  @ LOC,30 say "mu(tau) hi  = "
  @ LOC,44 say mu_tau_hi picture "9999.99"
  LOC=LOC+1
  @ LOC,10 say "lambda(tau) low = "
  @ LOC,28 say lambda_t_low picture "99.999999999"
  @ LOC,43 say "lambda(mu) low = "
  @ LOC,60 say lambda_m_low picture "99.999999999"
  LOC=LOC+1
  @ LOC,10 say "lambda(tau)     = "
  @ LOC,28 say lambda_tau picture "99.999999999"
  @ LOC,43 say "lambda(mu)      = "
  @ LOC,60 say lambda_mu picture "99.999999999"
  LOC=LOC+1
  @ LOC,10 say "lambda(tau) hi  = "
  @ LOC,28 say lambda_t_hi picture "99.999999999"
  @ LOC,43 say "lambda(mu) hi   = "
```

```
  @ LOC,60 say lambda_m_hi picture "99.999999999"
  LOC=LOC+2
  if LOC > pagelength
    LOC = 1
    if upper(printvar) = "S"
      wait "Hit any key to continue ..."
      clear
    endif
  endif
  *-------------------------------
  *     Skip for either every entry or for first entry of each month:
  skip
  if upper(intervar)="M"
    do while (month(CAL_DATE) = month(mdate)) .and. (.not. eof())
      skip
    enddo
  endif
enddo
*------------------------------
if upper(printvar) = "F"
  set device to screen
  set printer to
  ? chr(7)                         &&  Wake me up when done!
elseif upper(printvar) = "F"
  set device to screen
  set print on
  eject
  set print off
else
  wait "Hit any key to continue ..."
endif
*-------------------------------------------------------------------------------
close all
return        && to SRSAS
********************************************************************************
```

```
*****************************************************************************
*                                                                           *
*  Title    : Software Reliability Logarithmic Poisson Module (SRLOG.PRG)   *
*  Version  : 3.3                                                           *
*  Date     : 15 Oct 91                                                     *
*  Author   : Capt Joseph J. Stanko                                         *
*  Security : Unclassified                                                  *
*  Purpose  : This program:                                                 *
*             1.) Performs calculations on data to determine initial        *
*                 parameters for the fitted model.                          *
*             2.) Performs calculations on data to determine goodness-of-fit *
*                 for the Logarithmic Execution Time model.                 *
*                                                                           *
*  Theory   : In order to apply the Logarithmic Execution Time model to the *
*             failure data, initial parameter estimation must be            *
*             accomplished from the overall data.  Once the parameters      *
*             are calculated, they are then used in the model to calculate  *
*             estimated values (such as current number of failures and      *
*             failure intensity) along with the 95% confidence intervals.   *
*             These estimations are then compared against the actual data to *
*             determine the goodness-of-fit.                                *
*                                                                           *
*  Database : This program uses one database:                               *
*                                                                           *
*             TIME.DBF     -  This is a final database of dates and estimated *
*                             failure times and test durations:             *
*                                                                           *
*               Name      Type  Length  Decimal        Description          *
*             --------    ----  ------  -------  -------------------------- *
*             CAL_DATE    Date                   Date of occurrence of failure *
*             L_TIME_OCC   N      10       2     "Local" time of failure occur *
*                                                (wrt to start of that test)  *
*             TEST_DUR     N      10       2     Duration of test for that day *
*             T_TIME_OCC   N      10       2     "Total" time of failure occur *
*                                                (wrt to all total test time) *
*             TOTAL        N       4             Total failures to that point *
*                                                                           *
*  Modules  : N/A                                                           *
*                                                                           *
*****************************************************************************
store "O" to dbvar
@ 23,20 say "(D)T&E or (O)T&E Database?" ;
        get dbvar picture "@K !" valid(dbvar$"DO")
read
@ 23,20 clear

if upper(dbvar) = "O"          && Use OT&E version of time database
  use TIME alias TIME
else
  use TIMEDTE alias TIME       && Use DT&E version of time database
endif

*----------------------------------------------------------------------------
*     Variable Section (Note: mu's and lambda's are defined later on):

go bottom
store "         "   to dbname   && Name of database for output header
store 0.0000001     to delta    && Accuracy difference of parameters
store "M"           to intervar && Data output intervals (monthly or daily)
store 0.000         to lambda_0 && Initial failure intensity value
store TOTAL         to m_e      && Total number of failures
store 30            to max_iter && Max iterations to perform Newton-Raphson
store "S"           to printvar && Default print option (screen)
store T_TIME_OCCUR to t_e       && Total test time

*----------------------------------------------------------------------------
*     Data Entry Section:

@ 0,0  clear
set confirm on
@ 3,10 say "Enter Total Test Time:" get t_e      picture "9999999.99"
@ 4,10 say "Enter Max # Iteration:" get max_iter picture "999"
@ 5,10 say "Enter MLE Delta      :" get delta    picture "9.99999999"
```

```
if upper(dbvar) = "O"
  @ 6,10 say "Enter Initial Failure "
  @ 7,10 say "Intensity (0 for none):" get lambda_0 picture "9.999999999"
endif
read
@ 9,10 say "What is DB Name for Output?"  get dbname picture "!!!!!!!!"
read
set confirm off
@ 11,10 say "Data Output Interval: (M)onthly or (D)aily:" ;
        get intervar picture "@K !" valid(intervar$"MD")
read
@ 13,10 say "Send Data to (S)creen, (P)rinter, or (F)ile:" ;
        get printvar picture "@K !" valid(printvar$"SPF")
read
*-------------------------------------------------------------------------------
*     Data Direction Section:
if upper(printvar) = "F"
  if upper(dbvar) = "O"
    @ 15,10 say "Sending Data to File SRLOG.PRN ..."
    set printer to SRLOG.PRN
  else   && upper(dbvar) = "D"
    @ 15,10 say "Sending Data to File SRLOGD.PRN ..."
    set printer to SRLOGD.PRN
  endif
  set device to print
  pagelength = 4000
elseif upper(printvar) = "P"
  @ 15,10 say "Printing Data ..."
  set device to print
  pagelength = 55
else
  @ 0,0 clear
  pagelength = 20
endif

*-------------------------------------------------------------------------------
*     Data Calculation Section: Maximum Likelihood Estimation
@ 3,7  say "MLE Calculations for"
@ 3,31 say dbname
@ 3,40 say "using Logarithmic Poisson Model:"
*------------------------------
*     Make initial model parameter estimation, and sum failure occurrance
*     times to make calcuations easier:
b_hat = 1/(t_e)                      &&  Musa's recommended guess
go top
@ 5,10 say "Total Failures:         m_e    = "
@ 5,43 say m_e picture "99999.99"
@ 6,10 say "Failure Data End Time:  t_e    = "
@ 6,43 say t_e picture "99999999.99"
@ 7,10 say "Initial Model Param Est: b_hat = "
@ 7,43 say b_hat picture "99.999999999"
*------------------------------
*     Determine a better estimation for b_hat by making f_stat as close
*     as possible to 0 (uses Newton-Raphson method):
num_iter = 1
not_in_tol = .T.
if upper(printvar) = "P"
  set device to screen
  @ 17,10 say "Refining b_hat Estimate, Please Wait ..."
  set device to print
elseif upper(printvar) = "S"
  @ 8,10 say "Refining b_hat Estimate, Please Wait ..."
endif
*------------------------------
```

```
*     Iterate while out of tolerance or within alloted looping time:
do while (not_in_tol) .and. (num_iter <= max_iter)
  *---------------
  *     Determine the f(b_hat) and f'(b_hat) for Newton-Raphson Method
  *     based on a known initial value of failure intensity (lambda_0):
  if upper(dbvar)="O" .and. ;        && Different equation set for OT&E using
     lambda_0 <> 0.0                 &&    previous DT&E failure intensity #
    b_one = (1 + (b_hat*t_e))        && Shortens equation notation
    f_stat = ((m_e*b_hat) / log(b_one)) - lambda_0
    fp_stat = (((log(b_one))*m_e)-((m_e*b_hat)*(t_e/b_one))) / ((log(b_one))^2)
  *---------------
  *     Determine the f(b_hat) and f'(b_hat) for Newton-Raphson Method
  *     with no initial clues at all:
  else                               && We're either looking at DT&E data
                                     &&    or OT&E data without a priori info
    go top                           && Requires looping thru database again
    t_i_sum  = 0                     && Summation of failure occur times (ti)
    t_i2_sum = 0                     && Sum of square of fail occur times (ti)
    do while .not. eof()
      t_i_sum  = t_i_sum + (1/ (1+ (b_hat*T_TIME_OCCUR)))
      t_i2_sum = t_i2_sum + (-T_TIME_OCCUR / (1 + (b_hat*T_TIME_OCCUR))^2)
      skip
    enddo
    b_one = (1 + (b_hat*t_e))        && Shortens equation notation
    f_stat = (1/b_hat)*(t_i_sum) - ((m_e*t_e)/(b_one * log(b_one)))
    fp_stat = ((1/b_hat) * t_i2_sum) + ((t_i_sum) * (-1/b_hat^2))     ;
              - ( (-(m_e*t_e^2) * (1+ log(b_one)))                    ;
                / (b_one * log(b_one))^2 )
  endif
  *---------------
  *     The rest of this is the same for either case from above:
  bp_hat = b_hat - (f_stat/fp_stat)  && Burden & Faires Step #3
  if abs(bp_hat-b_hat)<delta         && Check for within tolerance delta
    not_in_tol = .F.
  endif
  if upper(printvar) = "P"           && Output the data as it is calculated
    set device to screen             &&    to verify convergence.
    @ 19,0 clear
    @ 19,10 say "b_hat   ="
    @ 19,20 say b_hat
    @ 20,10 say "bp_hat  ="
    @ 20,20 say bp_hat
    @ 21,10 say "F_Stat  = "
    @ 21,20 say f_stat
    @ 22,10 say "Fp_Stat = "
    @ 22,20 say fp_stat
    @ 17,10 say "Refining b_hat Estimate, Please Wait ..."
    set device to print
  elseif upper(printvar) = "S"       && Same thing here, but must use
    @  9,0 clear                     && different screen position for output.
    @  9,10 say "b_hat   ="
    @  9,20 say b_hat
    @ 10,10 say "bp_hat  ="
    @ 10,20 say bp_hat
    @ 11,10 say "F_Stat  = "
    @ 11,20 say f_stat
    @ 12,10 say "Fp_Stat = "
    @ 12,20 say fp_stat
    @ 8,10 say "Refining b_hat Estimate, Please Wait ..."
```

```
  endif
  b_hat = abs(bp_hat)
  num_iter = num_iter + 1
enddo
*------------------------------
*     Output additional data on refined values:
if upper(printvar) = "P"
  set device to screen
  @ 19,0  clear
  @ 19,10 say "Printing Data ..."
  set device to print
elseif upper(printvar) = "S"
  @ 9,0 clear
endif
if num_iter > max_iter
  @ 9,10 say "Method Ended After "
  @ 9,29 say max_iter picture "999"
  @ 9,33 say "Iterations."
endif
@ 10,10 say "Final Model Param Est:   b_hat = "
@ 10,43 say b_hat picture "99.999999999"
@ 11,10 say " "
*-------------------------------------------------------------------------------
*     Data Calculation Section: Parameters and Confidence Intervals
*     Determine the Expected (Fisher) information, and then
*     Calculate 95% Confidence Intervals:
@ 12,10 say "Parameter Calculations: theta, lambda_0, and 95th Percentile:"
b_one = (1 + (b_hat*t_e))
fisher = m_e * ( ((2*t_e)/(b_hat*b_one*log(b_one)))          ;
                - ( (1/(2*b_hat^2*log(b_one)))               ;
                   * (1 - (1/b_one^2)))                      ;
                - ( (t_e^2*(log(b_one)+1))                   ;
                   / ((b_one*log(b_one))^2))                 ;
               )
b_hat_low = b_hat - (1.96/sqrt(fisher))       &&  From Z statistic.
b_hat_hi  = b_hat + (1.96/sqrt(fisher))
b_0     = (m_e) / log(1+(b_hat*t_e))          &&  Calculated from parameter
b_0_low = (m_e) / log(1+(b_hat_low*t_e))      &&    b_hat.
b_0_hi  = (m_e) / log(1+(b_hat_hi*t_e))
theta     = 1/b_0                             &&  By definition.
theta_low = 1/b_0_low
theta_hi  = 1/b_0_hi
lambda_0     = b_0 * b_hat                    &&  Varying b_hat affects both
lambda_0_low = b_0_low * b_hat_low            &&    b_hat and b_0, etc.
lambda_0_hi  = b_0_hi * b_hat_hi
@ 14,10 say "Expected (or Fisher) Value          = "
@ 14,48 say fisher picture "9999999999.999999999"
@ 16,10 say "95% Boundary:              theta_low = "
@ 16,48 say theta_low picture "99.999999999"
@ 17,10 say "Failure Intensity Decay:       theta = "
@ 17,48 say theta picture "99.999999999"
@ 18,10 say "95% Boundary:               theta_hi = "
@ 18,48 say theta_hi picture "99.999999999"
@ 20,10 say "95% Boundary:           lambda_0_low = "
@ 20,48 say lambda_0_low picture "99.999999999"
@ 21,10 say "Initial Failure Intensity: lambda_0 = "
@ 21,48 say lambda_0 picture "99.999999999"
@ 22,10 say "95% Boundary:            lambda_0_hi = "
@ 22,48 say lambda_0_hi picture "99.999999999"
```

```
@ 23,10 say " "
*------------------------------------------------------------------------------
*     Output of Model Results:
if upper(printvar) = "P"
  set print on
  eject
  set print off
elseif upper(printvar) = "S"
  wait "Hit any key to continue ..."
  clear
endif
go top
LOC = 1
do while .not. eof()
  if LOC = 1                                        && Header information:
    @ LOC,10 say "Generating Plot Data for"
    @ LOC,35 say dbname
    @ LOC+1,5  say "---------------------------------------------"
    @ LOC+1,51 say "-----------------------"
    LOC = LOC + 3
  endif
  store CAL_DATE to mdate
  store T_TIME_OCCUR to tau
  store TOTAL to mu
  *----------------------------
  *     Calculate this info for each pass in the loop:
  mu_tau     = (1/theta) * log((lambda_0*theta*tau)+1)
  mu_tau_hi  = (1/theta_hi) * log((lambda_0*theta_hi*tau)+1)
  if ((lambda_0*theta_low*tau)+1) > 0
    mu_tau_low = (1/theta_low) * log((lambda_0*theta_low*tau)+1)
    brackets = .F.
  else
    mu_tau_low = abs(mu_tau_hi - mu_tau) + mu_tau
    brackets = .T.
  endif
  lambda_tau   = lambda_0 / ((lambda_0*theta*tau)+1)
  lambda_t_low = lambda_0_low / ((lambda_0_low*theta_low*tau)+1)
  lambda_t_hi  = lambda_0_hi / ((lambda_0_hi*theta_hi*tau)+1)
  lambda_mu    = lambda_0 * exp(-theta*mu)
  lambda_m_low = lambda_0_low * exp(-theta_low*mu)
  lambda_m_hi  = lambda_0_hi * exp(-theta_hi*mu)
  *----------------------------
  *     Now output the info for each pass in the loop:
  @ LOC,10 say "Day : "
  @ LOC,16 say mdate
  @ LOC,30 say "mu(tau) low = "
  @ LOC,44 say mu_tau_low picture "9999.99"
  LOC=LOC+1
  @ LOC,10 say "mu  = "
  @ LOC,16 say mu picture "9999.99"
  @ LOC,30 say "mu(tau)     = "
  @ LOC,44 say mu_tau picture "9999.99"
  LOC=LOC+1
  @ LOC,10 say "tau = "
  @ LOC,16 say tau picture "9999999.99"
  @ LOC,30 say "mu(tau) hi  = "
  @ LOC,44 say mu_tau_hi picture "9999.99"
  LOC=LOC+1
  @ LOC,10 say "lambda(tau) low = "
  @ LOC,28 say lambda_t_low picture "99.999999999"
  @ LOC,43 say "lambda(mu) low = "
  if brackets
    @ LOC,60 say "("
    @ LOC,61 say lambda_m_low picture "99.999999999"
    @ LOC,73 say ")"
```

```
  else
    @ LOC,60 say lambda_m_low picture "99.999999999"
  endif
  LOC=LOC+1
  @ LOC,10 say "lambda(tau)     = "
  @ LOC,28 say lambda_tau picture "99.999999999"
  @ LOC,43 say "lambda(mu)     = "
  @ LOC,60 say lambda_mu picture "99.999999999"
  LOC=LOC+1
  @ LOC,10 say "lambda(tau) hi  = "
  @ LOC,28 say lambda_t_hi picture "99.999999999"
  @ LOC,43 say "lambda(mu) hi  = "
  @ LOC,60 say lambda_m_hi picture "99.999999999"
  LOC=LOC+2
  if LOC > pagelength
    LOC = 1
    if upper(printvar) = "S"
      wait "Hit any key to continue ..."
      clear
    endif
  endif
  *-------------------------------
  *     Skip for either every entry or for first entry of month:
  skip
  if upper(intervar)="M"
    do while (month(CAL_DATE) = month(mdate)) .and. (.not. eof())
      skip
    enddo
  endif
enddo
*------------------------------------------
if upper(printvar) = "F"
  set device to screen
  set printer to
  ? chr(7)                               &&  Wake me up when done!
elseif upper(printvar) = "P"
  set device to screen
  set print on
  eject
  set print off
else
  wait "Hit any key to continue ..."
endif
*-------------------------------------------------------------------------------
close all
return
*******************************************************************************
```

Appendix E. *Proposed Software Reliability Database*

This appendix contains a description of the semantic data model, and the representation of a proposed software reliability database using this model.

E.1 Semantic Data Model

Korth identifies the entity-relationship (E-R) data model and the semantic data model (SDM) as two of the more widely known object-based models [49:6]. The E-R data model is very appropriate as the "front end" logical design that would then be implemented by a relational database model for the physical design; however, the E-R model requires users to explicitly define relationships between entities, even if the relationship itself has no data [51:228]. Additionally, the abstract concept of aggregation must be used by E-R models to express relationships among relationships, being handled by the concept of a virtual relation or "view" mechanism [49:40], [18:231].

SDM does not have these limitations, as it permits the meaning of the database to be specified in a more natural way [36:124]. This moves one step away from the physical implementation description toward the real-world description of the data. A recent study evaluated usability of the semantic models (in this case, the extended entity relationship model) versus that for a relational model by non-expert database users [7:126,137]. The results indicated that the users performed a conceptual representation task better with the semantic data model than the relational model. Applying a "bridge" approach, SDM could then be used as the initial logical design to capture as much of the meaning and representation of the real-world as possible. The database designer would then use E-R diagrams as an intermediate step to put the data in a format for the physical design [51:228] Finally, the E-R diagram would then be used for the internal schema, which would most likely be an implementation of the relational model.

The following sections provide SDM descriptions of proposed software maturity database objects, Musa Execution Time model objects, software system effectiveness objects, and logical schemas for the different classes.

E.2 Objects Identified for the Proposed Software Maturity Database

Mission Number	- Flight Number
Date of Mission	- Calendar Date
Tail Number	- Aircraft Tail Identification Number
OFP Suite Number	- Unique number for the suite of Operational Flight Programs on this specific mission
CSCI	- Computer Software Configuration Item
Start Time	- Aircraft Start Time (military time, no seconds)
Shut Down Time	- Aircraft Shut Down Time (military time, no seconds)
Time of Occurrence	- Military time when failure occurred. Note: if time is not available, it can be estimated using random arrival event calculation.
Severity	- Mission Impact due to failure: 1 System Abort. Software or firmware failure that results in a system abort. 2 System Degraded, No Workaround. Software or Firmware failure that severely degrades the system and no alternative workaround exists. Note: Program restarts are not an acceptable workaround. 3 System Degraded, With Workaround. Software or firmware failure that severely degrades the system and there exists a workaround (ie.--system rerouting through operator switchology). Note: Program restarts are not an acceptable workaround. 4 Software Failure, System Not Degraded. Software or firmware failure that does not severely degrade the system or any essential system function. 5 Minor Failure. All other minor or non-functional failures.
SPR Number	- This number will be different for each separate problem/failure, but will be the same for each occurrence of the same problem/failure.
Problem Description	- Brief description of the software problem.

E.3 Objects Identified from Musa Execution Time Software Reliability Model

Reliability	- Probability of failure free operation.
Failure Intensity	- Failures per unit time; the derivative with respect to time of the mean value function of failures.
Execution Time	- Processor time spent executing the program.
Initial Failure Intensity	- Initial value for failure intensity at start of operational assessment.
Present Failure Intensity	- Current value of failure intensity during operational assessment.
Failure Intensity Objective	- Desired value for failure intensity at end of operational assessment.
Expected Failures	- Expected number of failures experienced by time t.
Expected Total Failures	- Expected number of failures experienced in infinite time.
Additional Failures to Failure Intensity Objective	- Increment of expected failures associated with reaching failure intensity objective.
Additional Execution Time to Failure Intensity Objective	- Increment of execution time associated with reaching failure intensity objective.
Inherent Faults	- A fault associated with the original software product at completion of coding.
Fault Reduction Factor	- Net reduction in faults per failure experienced.
Fault Exposure Ratio	- Fraction of time the program passage results in failure.
Inherent Faults per Developed Source Instruction	- Ratio: inherent faults / source instructions.
Developed Executable Source Instruction	- The amount of developed code measured in executable source instructions.
Executable Object Instructions	- Amount of code measured in object instructions.
Instruction Execution Rate	- Speed at which instructions are executed
Linear Execution Frequency	- The number of times the program wold be executed per unit time if it had no branches or loops.

E.4 Objects Identified for Software System Effectiveness

Total Weapon System Effectiveness	- Total measure of the weapon system's effectiveness (includes software system effectiveness).
Software System Effectiveness	- Ratio of the non-failure operational time o a software system to its total operational time.
Mission Capable Rate	- Percentage of time the weapon system
Total Operational Test Time	- Total time spent in operational test of a weapon system.
Total Failure Duration Time	- Total time duration of the software system failure.
Mean Time to Restore Software	- Average time between occurrence f a software failure and when the software system has been returned to an operational state.

E.5 Logical Schema for the AIRCRAFT Class

```
AIRCRAFT
 description: all aircraft that participate in the flight test of
              a particular weapon system.
 member attributes:
    Tail_Number
         value class: ID_NUMBER
         may not be null
         not changeable
    Mission
         value class: MISSION_NUMBER
         may not be null
    Number_Of_ACUs
         value class: NUMBER_OF_COMPUTERS
         multivalued with size between 1 and 4
         may not be null
    Flight_Test_Time
         value class: TIME_AMOUNT
         may not be null
    Failure_Time
         value class: TIME_DURATION
         match: Failure_Duration of SOFTWARE_FAILURE on
                MISSION_NUMBER
         may not be null
 class attributes:
    Total_Flight_Test_Time
         description: Total of all flight test hours from all
                      missions for all aircraft.
         value class: TIME_AMOUNT
         derivation: Sum of Flight_Test_Time over members of this
                     class.
    Total_Failure_Time
         description: Total of all failure time from all missions
                      for all aircraft.
         value class: TIME_DURATION
         derivation: Sum of Failure_Time over members of this
                     class.
    Mission_Capable_Rate
         description: The percentage of time an aircraft is
                      capable of performing its mission; this
                      value is user input and not derived.
         value class: PERFORMANCE_RATE
    Software_System_Effectiveness
         description: Percentage of time the software system
                      operates correctly vs. the total attempted
                      operational time.
         value class: PERFORMANCE_RATE
         derivation: (Total_Flight_Test_Time - Total_Failure_Time)
                      / Total_Flight_Test_Time
    Total_Weapon_System_Effectiveness
         description: The effect of software performance on
                      mission accomplishment.
         value class: PERFORMANCE_RATE
         derivation: (Software_System_Effectiveness) * (Mission_Capable_Rate)
 identifiers:
    Tail_Number
```

E.6 Logical Schema for the MISSION Class

```
MISSION
 description: A single flight test activity of at least one
              aircraft for a specified length of time.
 member attributes:
    Number
         value class: MISSION_NUMBER
         may not be null
    Date
         value class: DATES
         may not be null
    Flown_By
         value class: AIRCRAFT
         match: Tail_Number of AIRCRAFT on Mission_Number
         may not be null
    OFP_Suite
         value class: CSCI_NAME
         match: Name of CSCI on Mission_Number
         may not be null
    Start_Time
         value class: TIME_AMOUNT
         may not be null
    Stop_Time
         value class: TIME_AMOUNT
         may not be null
    Duration
         value class: TIME_AMOUNT
         derivation: (Stop_Time) - (Start_Time)
 identifiers:
    Number
```

E.7 Logical Schema for the SOFTWARE_FAILURE Class

```
SOFTWARE_FAILURE
 description: The inability of a software system or system component
              to perform a required function within specified limits.
 member attributes:
    Mission_Number
         value class: MISSION_NUMBER
         may not be null
    Time_of_Occurrence
         value class: TIME_AMOUNT
         may not be null
    OFP_Suite
         value class: CSCI_NAME
         match: Name of CSCI on Mission_Number
         may not be null
    Failure_Duration
         value class: TIME_DURATION
         may not be null
    Severity_Level
         value class: SEVERITY_NUMBER
         multivalued with size between 1 and 5
         may not be null
    SPR_Number
         value class: SPR_NUMBERS
    Problem_Description
         value class: TEXT
 class attributes:
    Total_Failures
         description: The total number of software failures.
         value class: FAILURE_NUMBERS
         derivation: Number of members in this class.
    Total_Failure_Time
         description: The total time of all software failures.
         value class: TIME_AMOUNT
         derivation: Sum of all software Failure_Durations from
                     all missions for all aircraft.
 identifiers:
    Mission_Number + Time_Of_Occurrence + OFP_Suite
```

E.8 Logical Schema for the CSCI Class

```
CSCI
 description: Computer software configuration item--a specific
              software program.
 member attributes:
    Name
         value class: CSCI_NAME
         may not be null
    Mission_Number
         value class: MISSION_NUMBER
         may not be null
    Source_Lines_Of_Code
         value class: KSLOC
         may not be null
    Fault_Density
         description: OMEGA_I
         value class: DECIMAL_NUMBER
         may not be null
    Fault_Reduction_Factor
         description: B
         value class: DECIMAL_NUMBER
         may not be null
    Total_Failures_Expected
         description: NU_0
         value class: DECIMAL_NUMBER
         derivation: (Source_Lines_Of_Code) * (Fault_Density) /
                     (Fault_Reduction_Factor)
    Instruction_Expansion_Ratio
         description: Qx
         value class: DECIMAL_NUMBER
         may not be null
    Number_of_Object_Instructions
         description: I
         value class: DECIMAL_NUMBER
         derivation: (Source_Lines_of_Code) * (Instruction_
                      Expansion_Ratio)
    Instruction_Execution_Rate
         description: r
         value class: DECIMAL_NUMBER
         may not be null
    Linear_Execution_Frequency
         description: f
         value class: DECIMAL_NUMBER
         derivation (Instruction_Execution_Rate) / (Number_of_
                     Object_Instructions)
    Fault_Exposure_Ratio
         description: K
         value class: DECIMAL_NUMBER
         may not be null
    Initial_Failure_Intensity
         description: LAMBDA_0
         value class: DECIMAL_NUMBER
         derivation: (Linear_Execution_Frequency) * (Fault_
                      Exposure_Ratio) * (Fault_Density) *
                      (Source_Lines_Of_Code)
 identifiers:
    Name
```

E.9 Logical Schema for the RELIABILITY Class

```
RELIABILITY
 description: Values for measured reliability of software.
 member attributes:
    CSCI_Name
         value class: CSCI_NAMES
         may not be null
    Tail_Number
         value class: ID_NUMBER
         may not be null
    Initial_Failure_Intensity
         value class: DECIMAL_NUMBER
         match: Initial_Failure_Intensity of CSCI on Name
         may not be null
    Total_Failures_Expected
         value class: DECIMAL_NUMBER
         match: Total_Failures_Expected of CSCI on Name
         may not be null
    Failures_Experienced
         value class: FAILURE_NUMBERS
         match: Total_Failures of SOFTWARE_FAILURE on OFP_Suite
         may not be null
    Flight_Test_Time
         value class: TIME_AMOUNT
         match: Total_Flight_Test_Time of AIRCRAFT on Tail_Number
    Failure_Intensity_Objective
         description: A user-defined value; future version could
                      derive this from an operational profile
                      of the failure intensity.
         value class: DECIMAL_NUMBER
    Expected_Failures_Experienced
         description: MU_TAU
         value class: DECIMAL_NUMBER
         derivation: (Total_Failures_Expected) * (1 -
                      exp(-((Initial_Failure_Intensity) /
                      (Total_Failures_Expected)) * (Flight_Test_Time)))
    Present_Failure_Intensity_from_Time
         description: LAMBDA_TAU
         value class: DECIMAL_NUMBER
         derivation: (Initial_Failure_Intensity) * exp(-((Initial_
                      Failure_Intensity) / (Total_Failures_Expected))
                      * (Flight_Test_Time))
    Present_Failure_Intensity_from_Failures
         description: LAMBDA_MU
         value class: DECIMAL_NUMBER
         derivation: (Initial_Failure_Intensity) * (1 -
                      ((Failures_Experienced) / (Total_Failures_Expected)))
    Additional_Failures_to_Failure_Intensity_Objective
         description: DELTA_MU
         value class: DECIMAL_NUMBER
         derivation: ((Total_Failures_Expected) / (Initial_Failure_Intensity))
                      * (Present_Failure_Intensity -
                      Failure_Intensity_Objective)
    Additional_Execution_Time_to_Failure_Intensity_Objective
         description: DELTA_TAU
         value class: DECIMAL_NUMBER
         derivation: ((Total_Failures_Expected) / (Initial_Failure_Intensity))
                      * ln((Present_Failure_Intensity) /
                      (Failure_Intensity_Objective))
    Current_Reliability
         description: R_TAU
         value class: DECIMAL_NUMBER
         derivation: exp(-(Present_Failure_Intensity) * (Flight_Test_Time)
 class attributes:
    Total_Expected_Failures_Experienced
         description: MU_TAU_Tot
```

```
      value class: DECIMAL_NUMBER
      derivation: Sum of all Expected_Failures_Experienced.
   Total_Present_Failure_Intensity_from_Time
      description: LAMBDA_TAU_Tot
      value class: DECIMAL_NUMBER
      derivation: Average of Present_Failure_Intensities_from_Time.
   Total_Present_Failure_Intensity_from_Failures
      description: LAMBDA_MU_Tot
      value class: DECIMAL_NUMBER
      derivation: Average of Present_Failure_Intensities_from_Failures.
   Total_Additional_Failures_to_Failure_Intensity_Objective
      description: DELTA_MU_Tot
      value class: DECIMAL_NUMBER
      derivation: Sum of all Additional_Failures_to_Failure_
                  Intensity_Objective.
   Total_Additional_Execution_Time_to_Failure_Intensity_Objective
      description: DELTA_TAU_Tot
      value class: DECIMAL_NUMBER
      derivation: Sum of all Additional_Execution_Time_to_
                  Failure_Intensity_Objectives.
   Total_Current_Reliability
      description: R_TAU_Tot
      value class: DECIMAL_NUMBER
      derivation: Multiplicative specification of Current_Reliability
                  that includes all members.
identifiers:
   CSCI_Name + Tail_Number
```

E.10 Logical Schema for Other Classes

```
CSCI_NAME
 description: Computer System Configuration Item name.
              interclass connection: subclass of STRINGS where length is
              less than or equal to 8 alphanumeric characters.

DATES
 description: Day of the year.
 interclass connection: subclass of STRINGS where value is DDMMMYY;
              DD is a positive integer between 1 and 31 inclusive;
              MMM has value 'JAN','FEB','MAR','APR','MAY','JUN',
                            'JUL','AUG','SEP','OCT','NOV','DEC';
              YY is a positive integer with format 99.

DECIMAL_NUMBER
 description: A decimal number for arithmetic computations.
 interclass connection: subclass of STRINGS where format is
              number 99.99999999.

FAILURE_NUMBERS
 description: The number of software failures.
 interclass connection: subclass of STRINGS where format is
              number 9999.

ID_NUMBER
 description: Aircraft tail identification number.
 interclass connection: subclass of STRINGS where length is less than
              or equal to 8 alphanumeric characters.

KSLOC
 description: The number of source lines of code in thousands.
 interclass connection: subclass of STRINGS where format is
              number 9999.

MISSION_NUMBER
 description: Specific flight/mission number identifier.
 interclass connection: subclass of STRINGS where length is less than
              or equal to 5 alphanumeric characters.

NUMBER_OF_COMPUTERS
 description: Integer number of computers on-board the aircraft.
 interclass connection: subclass of STRINGS where positive integer
              is between 1 and 4 inclusive.

PERFORMANCE_RATE
 description: A ratio of time values: non-failure operational time
              per total operational time.
 interclass connection: subclass of STRINGS where format is
              number 9.99.

SEVERITY_NUMBER
 description: Integer number of severity of the software failure.
 interclass connection: subclass of STRINGS where positive integer
              is between 1 and 5 inclusive.

SPR_NUMBER
 description: Software Problem Report Number.
 interclass connection: subclass of STRINGS where length is less than
              or equal to 8 alphanumeric characters.

TEXT
 description: Narrative text describing the software failure.
 interclass connection: subclass of STRINGS where length is less than
              or equal to 80 alphanumeric characters.

TIME_AMOUNT
 description: An amount of time expressed in hours and minutes.
 interclass connection: subclass of STRINGS where value is HH:MM;
              HH is a positive integer between 0 and 23 inclusive;
              MM is a positive integer between 0 and 59 inclusive.

TIME_DURATION
 description: An amount of time expressed in minutes.
 interclass connection: subclass of STRINGS where format is
              number 99.9.
```

Bibliography

1. Adolph, Charles E. and Phillip Montgomery. "Cost-Effective Testing of Software-Intensive Systems," Paper presented at *Society of Flight Test Engineers Sixteenth Annual Symposium*. Seattle, Washington, 29 July–2 August 1985.

2. Air Force Operational Test and Evaluation Center. *An Introduction to AFOTEC: Mission, History, and Programs*. Albuquerque, New Mexico: HQ AFOTEC, 1990.

3. Angus, John E. "The Application of Software Reliability Models to a Major C3I System," *Proceedings of the 1984 Annual Reliability and Maintainability Symposium*. San Francisco, California, January 24-26, 1984: IEEE, 1984.

4. Angus, John E. et. al. "Software Reliability Model Evaluation," *Proceedings of the 1980 Annual Reliability and Maintainability Symposium*. San Francisco, California, January 22-24, 1980: IEEE, 1980.

5. American National Standards Institute and Institute of Electrical and Electronics Engineers. "ANSI/IEEE Standard 729-1983, Glossary of Software Engineering Terminology," *Software Engineering Standards*. Third Printing. New York: The Institute of Electrical and Electronics Engineers, Inc. 1987.

6. Bastani, F. B. and C. V. Ramamoorthy. "Software Reliability," *Handbook of Statistics*, Vol. 7, edited by P. R. Krishnaiah and C. R. Rao. New York: Elsevier Science Publishers, 1988.

7. Batra, Dinesh et al. "Comparing Representations with Relational and EER Models," *Communications of the ACM, 33(2)*: 126-139 (February 1990).

8. Baumann, Maj James. *Proposal for New Operational Software Evaluation Method*. Unpublished paper, 1990.

9. Behun, David J. "Software Reliability–Let's Start Doing It," *Proceedings of the 1984 Annual Reliability and Maintainability Symposium*. San Francisco, California, January 24-26, 1984: IEEE, 1984.

10. Beizer, Boris. *Software System Testing and Quality Assurance*. New York: Van Nostrand Reinhold Company, 1984.

11. Boehm, Barry W. "Software Engineering," *IEEE Transactions on Computers, C-25(12)*: 1226-1241 (December 1976).

12. Booch, Grady. *Software Components with Ada*. Menlo Park, California: Benjamin/Cummings, 1987.

13. Booch, Grady. *Object Oriented Design with Applications*. Menlo Park, California: Benjamin/Cummings, 1991.

14. Burden, Richard L. and J. Douglas Faires. *Numerical Analysis* (Fourth Edition). Boston: PWS-KENT Publishing Company, 1989.

15. Canan, James W. "The Software Crisis," *Air Force Magazine*: 46-53 (May 1986).

16. Cardow, James E. and Joseph J. Stanko. "A Standardized Software Reliability Measurement Methodology," *Proceedings of the 1991 International Software Quality Conference*. Dayton, Ohio, October 7-9, 1991: American Society for Quality Control, 1991.

17. Collas, Gerard. "Prediction for System Reliability and Availability," *Proceedings of the 1989 Annual Reliability and Maintainability Symposium*. Atlanta, Georgia, January 24-26, 1989: IEEE, 1989.

18. Date, C. J. *An Introduction to Database Systems, Volume II.* United States: Addison-Wesley, 1983.

19. Department of Defense. *Electronic Reliability Design Handbook.* MIL-HNDBK-338-1A. Department of Defense, 1988.

20. Department of Defense. *Definitions of Terms for Reliability and Maintainability.* MIL-STD-721C. Department of Defense, 1981.

21. Department of the Air Force. *Software Operational Test and Evaluation Guidelines, Management of Software Operational Test and Evaluation.* AFOTEC Pamphlet 800-2, Vol. 1. Albuquerque: HQ AFOTEC, 1 August 1986 (AD-A178234).

22. Department of the Air Force. *Software Maintainability Evaluation Guide.* AFOTEC Pamphlet 800-2, Vol. 3. Albuquerque: HQ AFOTEC, 31 October 1989 (AD-A218934).

23. Department of the Air Force. *Software Maturity Evaluation Guide.* AFOTEC Pamphlet 800-2, Vol. 6. Albuquerque: HQ AFOTEC, 1 October 1990 (AD-A228445).

24. Department of the Air Force. *Air Force Reliability and Maintainability Policy.* AFR 800-18, 20 January 1987.

25. Department of the Air Force. *Software Quality Indicators.* AFSC Pamphlet 800-14, Andrews Air Force Base, Maryland: AFSC, 20 January 1987.

26. Duvall, Lorraine et al. "Data Needs for Software Reliability Modelling," *Proceedings of the 1980 Annual Reliability and Maintainability Symposium.* San Francisco, California, January 22-24, 1980: IEEE, 1980.

27. Farr, William H. *A Survey of Software Reliability Modeling and Estimation.* Naval Surface Weapons Center Technical Report NSWC TR 82-171. Dahlgren, Virginia: Naval Surface Weapons Center, September 1983 (AD-A154874).

28. ---. "A PC Tool for Software Reliability Measurement," *National Conference Proceedings on Software Reliability and Testing.* Presented by The National Institute for Software and Productivity, Washington, D. C. November 16-17, 1988.

29. Farr, William H. and Oliver D. Smith. *Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS) User's Guide.* Naval Surface Weapons Center Technical Report NSWC TR 84-373 (Revision 1). Dahlgren, Virginia: Naval Surface Weapons Center, December 1988 (AD-B101292).

30. Ferens, Daniel V. *Mission Critical Computer Software Support Management.* Wright-Patterson Air Force Base, Ohio: Air Force Institute of Technology, 1987.

31. Ferrara, K. C. et al. "Software Reliability from a System Perspective," *Proceedings of the 1989 Annual Reliability and Maintainability Symposium.* Atlanta, Georgia, January 24-26, 1989: IEEE, 1989.

32. Freeman, Peter A. and Marie-Claude Gaudel. "Building a Foundation for the Future of Software Engineering," *Communications of the ACM, 34(5)*: 30-33 (May 1991).

33. Furtado, Antonio L. and Erich J. Neuhold. *Formal Techniques for Data Base Design.* Germany: Springer-Verlag, 1986.

34. Goel, Amrit L. "Software Reliability Models: Assumptions, Limitations, and Applicability," *IEEE Transactions on Software Engineering, SE-11(12)*: 1411-1423 (December 1985).

35. Goel, Amrit L. and Kazu Okumoto. "Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures," *IEEE Transactions on Reliability, R28(3)*: 206-211 (August 1979).

36. Hammer, Michael and Dennis McLeod. "Database Description with SDM: A Semantic Database Model," reprinted in *Readings in Object-Oriented Database Systems*. United States: Morgan Kaufmann, 1990.

37. Henry, Sallie and Dennis Kafura. "Software Structure Metrics Based on Information Flow," *IEEE Transactions on Software Engineering, SE-7(5)*: 510-518 (September 1981).

38. Hines, William W. and Douglas C. Montgomery. *Probability and Statistics in Engineering and Management Science* (Second Edition). New York: John Wiley & Sons, 1980.

39. Iannino, Anthony et al. "Criteria for Software Reliability Model Comparisons," *IEEE Transactions on Software Engineering, SE-10(6)*: 687-691 (November 1984).

40. Institute of Electrical and Electronics Engineers. *IEEE Standard 982.1-1988, IEEE Standard Dictionary of Measures to Produce Reliable Software*. New York: The Institute of Electrical and Electronics Engineers, Inc. 1989.

41. Institute of Electrical and Electronics Engineers. *IEEE Standard 982.2-1988, IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software* (Corrected Edition). New York: The Institute of Electrical and Electronics Engineers, Inc. 1989.

42. James, L. E. et al. *Combined HW/SW Reliability Models*. Rome Air Development Center Technical Report RADC-TR-82-68. New York: Rome Air Development Center, April 1982 (AD-A116566).

43. Jelinski, Z. and P. Moranda. "Software Reliability Research," *Statistical Computer Performance Evaluation*, edited by Walter Freiberger. New York: Academic Press, 1972.

44. Johnson, Capt Stephen K. *Modifying AFOTEC's Software Maintainability Evaluation Guidelines*. MS Thesis AFIT/GCS/ENG/88D-10. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson Air Force Base OH, December 1988 (AD-A203381).

45. Johnson, Capt Stephen K. Personal Correspondence. HQ AFOTEC/LG5. Kirtland Air Force Base, New Mexico. 14 August 1991.

46. Johnson. Capt Stephen K. Telephone interview. HQ AFOTEC/LG5, Kirtland Air Force Base. New Mexico, 16 August 1991.

47. Johnson. Capt Stephen K. Personal Correspondence. HQ AFOTEC/LG5. Kirtland Air Force Base, New Mexico, 13 September 1991.

48. Kilp, Stephen G. *Management of Avionics Software During Flight Test*. MS Thesis. College of Engineering, West Coast University, Los Angeles. California, July 1987.

49. Korth. Henry F. and Abraham Silberschatz. *Database System Concepts*. United States: McGraw-Hill. 1986.

50. Koss, W. Edwards. "Software Reliability Metrics for Military Systems," *Proceedings of the 1988 Annual Reliability and Maintainability Symposium*. Los Angeles, California, January 26-28, 1988: IEEE. 1988.

51. Kroenke, David. *Database Processing: Fundamentals, Modeling, Applications* (Second Edition). United States: Science Research Associates, 1983.

52. Lawlis. Maj Patricia K. *Supporting Selection Decisions Based on the Technical Evaluation of Ada Environments and Their Components*. PhD Dissertation. Arizona State University, Tempe AZ, 1989.

53. Lipow, M. "Quantitative Demonstration and Cost Considerations of a Software Fault Removal Methodology," *Quality and Reliability Engineering International, 1(1)*: 27-35 (1985).

54. Lipow, Myron and Erwin Book. "Implications of R&M 2000 on Software," *IEEE Transactions on Reliability, R36(3)*: 355-361 (August 1987).

55. Littlewood, Bev and John L. Verrall. "A Bayesian Reliability Model with a Stochastically Monotone Failure Rate," *IEEE Transactions on Reliability, R23(2)*: 108-114 (June 1974).

56. McCall, J. et al. *Methodology for Software Reliability Prediction.* Rome Air Development Center Technical Report RADC-TR-87-171, Vol 1. New York: Rome Air Development Center, November 1987 (AD-A190018).

57. McCall, J. et al. *Methodology for Software Reliability Prediction.* Rome Air Development Center Technical Report RADC-TR-87-171, Vol 2. New York: Rome Air Development Center, November 1987 (AD-A190019).

58. Mazzuchi, Thomas A. and Nozer D. Singpurwalla. "Software Reliability Models," *Handbook of Statistics*, Vol. 7, edited by P. R. Krishnaiah and C. R. Rao. New York: Elsevier Science Publishers, 1988.

59. Mulock, Richard B. "Software Reliability," *Proceedings of the 1969 Annual Reliability Symposium.* Chicago, Illinois, January 21-23, 1969: IEEE, 1969.

60. Musa, John D. "A Theory of Software Reliability and its Application," *IEEE Transactions on Software Engineering, SE-1(3)*: 312-327 (September 1975).

61. ——. "Tools for Measuring Software Reliability," *IEEE Spectrum, 26(2)*: 39-42 (February 1989).

62. Musa, John D. and K. Okumoto. "A Logarithmic Poisson Execution Time Model for Software Reliability Measurement," *Proceedings of the 7th International Conference on Software Engineering.* Orlando, Florida, March 26-29, 1984: IEEE, 1984.

63. Musa, John D. and William W. Everett. "Software-Reliability Engineering: Technology for the 1990's," *IEEE Software*: 36-43 (November 1990).

64. Musa, John D. et al. *Software Reliability: Measurement, Prediction, Application.* New York: McGraw-Hill Book Company, 1987.

65. Myers, Glenford J. *Software Reliability.* New York: John Wiley & Sons, 1976.

66. Nantucket Corporation. *Clipper User Manual.* United States: Nantucket Corporation, 1988.

67. Ohba, Mitsuru and Xiao-Mei Chou. "Does Imperfect Debugging Affect Software Reliability Growth?" reprinted in *Software Reliability Models: Theoretical Developments, Evaluation & Application*, edited by Yashwant K. Malaiya and Pradip K. Srimani. Los Alamitos, California: IEEE Computer Society Press, 1991.

68. Parnas, David L. et al. "Evaluation of Safety-Critical Software," *Communications of the ACM, 33(6)*: 636-648 (June 1990).

69. Ramamoorthy, C. V. and Farokh B. Bastani. "Software Reliability–Status and Perspectives," *IEEE Transactions on Software Engineering, SE-8(4)*: 354-371 (July 1982).

70. Ramamoorthy, C. V. et al. "A Systematic Approach to the Development and Validation of Critical Software for Nuclear Power Plants," *Proceedings of the 4th International Conference on Software Engineering.* Munich, Germany, September 17-19, 1979: IEEE, 1979.

71. Reibman, Andrew L. and Malathi Veeraraghavan. "Reliability Modeling: An Overview for System Designers," *Computer, 24(4)*: 49-57 (April 1991).

72. Rhea, John. "The Next Generation of Avionics," *Air Force Magazine*: 68-72 (January 1990).

73. Schick, George J. and Ray W. Wolverton. "An Analysis of Competing Software Reliability Models," *IEEE Transactions on Software Engineering, SE-4(2)*: 104-120 (March 1978).

74. Selby, Richard W. "Empirically Based Analysis of Failures in Software Systems," *IEEE Transactions on Reliability, R39(4)*: 444-454 (October 1990).

75. Shaw, Mary. "When is 'Good' Enough? Evaluating and Selecting Software Metrics," *Software Metrics: An Analysis and Evaluation*, edited by Alan Perlis, et al. Cambridge, Massachusetts: MIT Press, 1981.

76. Shooman, Martin L. "Probabilistic Models for Software Reliability Prediction," *Statistical Computer Performance Evaluation*, edited by Walter Freiberger. New York: Academic Press, 1972.

77. ---. *Software Engineering: Design, Reliability, and Management.* New York: McGraw-Hill Book Company, 1983.

78. Shumskas, Anthony F. "Why Higher Reliability Software Should Result from Reduced Test and Increased Evaluation," *National Conference Proceedings on Software Reliability and Testing*. Arlington, Virginia, November 16-17, 1988: The National Institute for Software and Productivity, 1988.

79. Siefert, David M. NCR Corporation. "Achieving Reliable Software: The Software Reliability Handbook." Presentation to the American Society for Quality Control, 1st International Conference on Software Quality, Dayton Convention Center, Dayton, Ohio, 6-9 October 1991.

80. ---. "Software Reliability Handbook: Achieving Reliable Software," *Proceedings of the 1991 International Software Quality Conference*. Dayton, Ohio, October 7-9, 1991: American Society for Quality Control, 1991.

81. Siegel, Jane A. L. et al. *National Software Capacity: Near-Term Study*. Software Engineering Institute Technical Report CMU/SEI-90-TR-12, ESD-TR-90-13. Carnegie Mellon University: Software Engineering Institute, 1990.

82. Sommerville, Ian. *Software Engineering* (Third Edition). Wokingham, England: Addison-Wesley Publishing Company, 1989.

83. Stark, George E. "A Survey of Software Reliability Measurement Tools," Paper presented at the *Symposium on Software Reliability Engineering*, Austin, Texas, 1991. Paper received from author.

84. Stytz, Maj Martin. Class lecture for CSCE 594, Software Design. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson Air Force Base OH, November 1990.

85. Sukert, Alan and Amrit L. Goel. "A Guidebook for Software Reliability Assessment," *Proceedings of the 1980 Annual Reliability and Maintainability Symposium*. San Francisco, California, January 22-24, 1980: IEEE, 1980.

86. Thayer, T. A. et al. *Software Reliability Study*. Rome Air Development Center Technical Report RADC-TR-76-238. New York: Rome Air Development Center, August 1976 (AD-A030798).

87. Tindell, Capt Douglas R. *Maintenance Metrics for JOVIAL (J73) Software*. MS Thesis AFIT/GE/ENG/88D-57. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson Air Force Base OH, December 1988 (AD-A202713).

88. Trachtenberg, Martin. "A General Theory of Software-Reliability Modeling," *IEEE Transactions on Reliability, R39(1)*: 92-96 (April 1990).

89. Verma, Pradeep and Yashwant K. Malaiya. "In Search of the Best Software Reliability Model," *7th Annual Software Reliability Symposium, IEEE Denver Section*. Colorado Springs: IEEE Reliability Society Denver Chapter, 1989.

90. Vienneau, Robert L. "The Cost of Testing Software," *Proceedings of the 1991 Annual Reliability and Maintainability Symposium*. Orlando, Florida, January 29-31, 1991: IEEE, 1991.

91. Walters, Gene F. and James A. McCall. "Software Quality Metrics for Life-Cycle Cost-Reduction," *IEEE Transactions on Reliability, R28(3)*: 212-219 (August 1979).

92. Westgate, Capt Charles J. III. *Validation of an Exponentially Decreasing Failure Rate Software Reliability Model.* MS Thesis AFIT/GLM/LSY/89S-71. School of Systems and Logistics, Air Force Institute of Technology (AU), Wright-Patterson Air Force Base OH, September 1989 (AD-A215546).

93. Wilson, Larry and Wenhui Shen. *Software Reliability Perspectives, Final Report.* For the Period Ending March 31, 1988. NASA-CR-181523. Research Grant NAG1-750. Norfolk Virginia: Department of Computer Science, Old Dominion University, December 1987.

94. Wiltse, J. D. and M. McPherson. *A Software Reliability Prediction Method Using Software Maturity Data.* Unpublished paper. HQ AFOTEC, Kirtland AFB, New Mexico.

95. Woodruff, B. W. and A. H. Moore. "Application of Goodness-of-Fit Tests in Reliability," *Handbook of Statistics*, Vol. 7, edited by P. R. Krishnaiah and C. R. Rao. New York: Elsevier Science Publishers, 1988.

96. Yamada, Shigeru et al. "S-Shaped Reliability Growth Modeling for Software Error Detection," *IEEE Transactions on Reliability, R32(5)*: 475-478 (December 1983).

97. Zdonik, Stanley B. and David Maier editors. *Readings in Object-Oriented Database Systms.* United States: Morgan Kaufmann, 1990.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
December 1991

3. REPORT TYPE AND DATES COVERED
Master's Thesis

4. TITLE AND SUBTITLE
A Standardized Software Reliability Measurement Methodology

5. FUNDING NUMBERS

6. AUTHOR(S)
Joseph J. Stanko, Captain, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCE/ENG/91D-09

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
HQ AFOTEC/LG5, Kirtland AFB NM 87117-7001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Current Air Force practice is to perform Operational Test and Evaluation (OT&E) for each new weapon system. In support of this, Headquarters Air Force Operational Test and Evaluation Center (HQ AFOTEC) is responsible for measuring both suitability and effectiveness. While suitability is adequately measured, the current effort only addresses hardware effectiveness, or at best, system effectiveness. Since tools and metrics are in place for software suitability assessments related to OT&E (for example, software maintainability), there should be some effective way of measuring the operational effectiveness of software. Currently, HQ AFOTEC/LG5 has a data collection tool for collecting software failure data to analyze software maturity. This thesis proposes that the LG5 software maturity database could be used as the baseline for a software reliability metric that would map to the finite time OT&E environment. This study does not predict software reliability, nor does it attempt to define what constitutes reliable software. Instead, this study evaluates software reliability measurement mapped to finite OT&E time frames (i.e.-failures per flight hour). This evaluation is conducted for several software reliability models, with two candidate models chosen based on selection criteria. Implementation of the candidate models was accomplished for an office computer environment to permit use by OT&E test teams at various locations. Testing was performed based on actual OT&E software maturity data.

14. SUBJECT TERMS
Software Reliability, Software Testing, Reliability, Software, Software Engineering

15. NUMBER OF PAGES
177

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL