ESTIMATING RELIABILITY OF SOFTWARE SYSTEMS, AN EVALUATION OF CURRENT METHODS

by

Scott Carey

A project report submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Systems Engineering

APPROVED:

Benjamin S. Blanchard, Chairman

Richard E. Nance

Konstantinos P. Triantis

December 1995

Blacksburg, Virginia

Key words: Reliability, Software, Model, Comparison, Estimation

c.2

ESTIMATING RELIABILITY OF SOFTWARE SYSTEMS, AN EVALUATION OF CURRENT METHODS

by

Scott Carey

Committee Chairman: Benjamin S. Blanchard
Systems Engineering

(ABSTRACT)

The purpose of this project is to evaluate several software reliability models and then determine which model is best suited for use on the Advanced Tomahawk Weapon Control System (ATWCS).

The models selected for evaluation are Musa's "Basic Execution Time Model", the model by Jelinski and Moranda, Musa and Okumoto's "Logarithmic Poisson Model", and Yamada's "S-Shaped Reliability Growth Model." Each model is evaluated against several comparison criteria. Each criterion is weighted in such a manner that indicates its significance to the target system and the system's environment. Data from the early phases of ATWCS system testing is utilized in the evaluation of each model.

The software system environment is held in the forefront throughout the entire project. The importance of the environment to the selection of the most accurate reliability model is crucial.

ACKNOWLEDGMENTS

I would first like to thank my wife, Marianne, for her faithful support throughout this endeavor. Even though we have lived apart for the last five months while I have been attending to the completion of this masters program, she has been by my side in spirit and forever in my heart.

Sincere thanks also go to my committee members Benjamin S. Blanchard (chairman), Richard E. Nance and Konstantinos P. Triantis for providing direction and encouragement in the development of this project report. A special note of appreciation goes to Loretta Tickle for her constant words of encouragement on topics both educational and personal.

Finally, I would like to thank the Naval Surface Warfare Center for allowing me the opportunity to pursue my educational goals. The Academic Fellowship Program is a tremendous benefit for the employees and the center.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Objectives

A significant amount of effort has been expended on reliability estimation methodologies for hardware. These systems have received the bulk of attention throughout history and the engineering principles surrounding them have been finely tuned and well exercised. The same is not true for software. Although the Systems Engineering approach to system development has been adapted to software development, the same level of tried and true engineering principles does not exist. Specifically, the methodologies available for software reliability estimation have yet to mature to the level of those existing for hardware.

New methodologies are appearing as the necessity to estimate the reliability of software becomes great. This necessity has steadily increased due to several factors such as: the high cost of software development and maintenance, the shear size and complexity of today's software, the increasing criticality of software, and extended system life cycles. With more methodologies available for use, a new problem emerges. Which method(s) are the most accurate given the environment for which the system is to be utilized?

Involvement in the Software Reliability Working Group for the Advanced Tomahawk Weapon Control System (ATWCS) highlighted the specific need that led to the topic selection. The reliability issues surrounding the hardware were few. However, the software created many issues concerning reliability. One issue is the realization that large amounts of operational data will not be available for analysis until approximately the

year 2000. As of August 1, 1995, the ATWCS is within approximately two months of beginning the first formal testing phase outside of the development house. The U.S. Navy has scheduled the initial ATWCS fleet deployment for late 1996 or early 1997. Due to scheduling issues surrounding ship availability, the system will not reside on a relatively large number of ships until approximately the year 2000. The data initially available is the result of various test phases such as: Design Agent Functional Quality Testing, Integration Agent Functional Quality Testing, Stress Testing, Endurance Testing, Performance Testing, and System Testing. As stated above, the necessity for accurately estimating the software reliability is great. The ATWCS is a mission critical system upon which lives may depend. It is a high cost, highly complex system with a planned extended life cycle. The need to have a high degree of confidence in the system is unquestioned. The high degree of confidence is in part a direct result of accurate software reliability estimation capabilities. Therefore, the focus of this project is on the software aspects of the ATWCS and the ultimate selection of a software reliability model that is best suited to the system.

1.2 The System Overview

The system from which data will be utilized for this study is the ATWCS. Therefore, it is important to have an ATWCS overview and to see how the software system relates to the definition of a system.

The ATWCS provides for the preparation and launch of the Tomahawk cruise missile. In this capacity, the system enables shipboard operators to perform all of the following:

- To generate and maintain a surface track data base in a given theater of operations,
- To coordinate strike activities performed both on own-ship and between own-ship and other ships in a battle group,
- To plan tomahawk engagements, and
- To initialize and launch tomahawk missiles.

The primary mission for the system is to prepare and launch any conventional Tomahawk Land Attack Missile variant. To do this, the system must support engagement planning, use a given mission or mission update (including control of the missile launcher, missile alignment, mission download, and launch functions). The secondary missions for the system are to support Battle Group Data Base Management, to support Strike Coordination, and to provide operator support. The system is responsible for providing a current tactical picture for engagement and mission planning to other ship platforms in the operating area. Operator training and reference capabilities such as on-line help, documentation, and training features are also part of the system.

Blanchard and Fabrycky (1990) define a system as "a set of interrelated components working together toward some common objective. " They also state that a system is composed of components, attributes, and relationships. Where components are parts of a system that perform input, processing and output; attributes are properties of the components; and relationships are the ties between the components and attributes.

The Advanced Tomahawk Weapon Control System is made up of hardware and software subsystems that when combined, make the whole system. The hardware subsystem, depicted in Figure 1, shows a high level diagram of the ATWCS Track Control Group (TCG) hardware architecture. There are six Tactical Advanced Computers (TACs), version 3, each containing a Hewlett Packard 755 processor. The primary interfaces between the processors and external systems are accomplished via the use of local area networks (LANs). The processors, or CPUs, labeled as COMMS-1, COMMS-2, CIC-1, and CIC-2 are connected to LANs that are classified for use at the Secret level while the CPUs labeled TACT-1 and TACT-2 are connected to LANs that are classified for use at the Top Secret level. Additionally, there are point-to-point interfaces both within the system and with external systems. The hardware architecture includes redundant processors to provide the system with the capability to meet its hardware reliability requirements (1600 hours as of Initial Operational Capability and 3200 hours after deployment). The software components that are associated with each CPU in Figure 1 simply show that each particular software component may execute on that CPU. In practice, the execution of software components is managed by the System Manager (SM) and is distributed across the available CPUs.

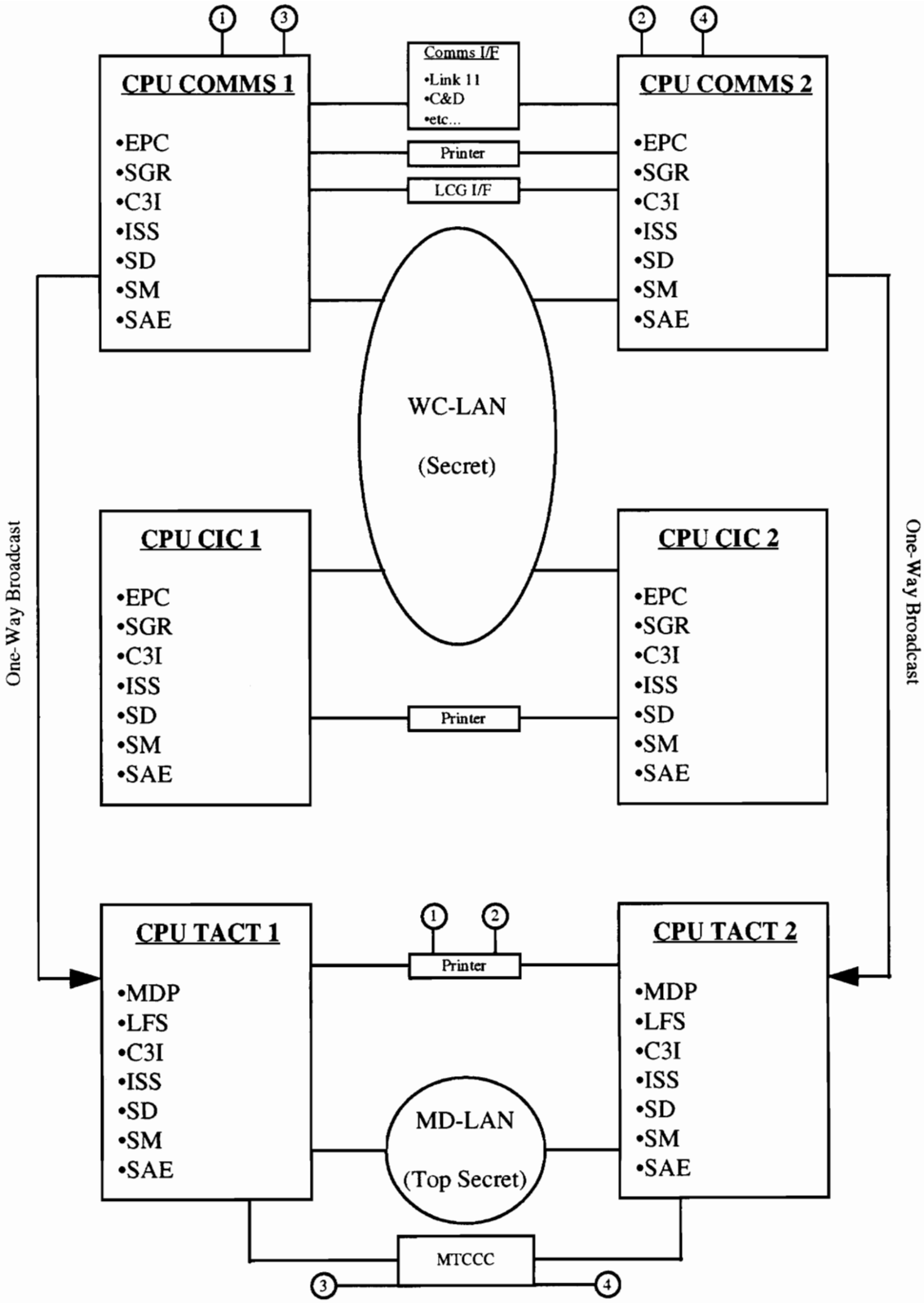


Figure 1. ATWCS Hardware Architecture

The software subsystem is broken into high level components (or subsystems) representing a Computer Software Configuration Item (CSCI). ATWCS has nine CSCIs. Each CSCI can be broken down further into modules (or components). The reliability value allocated to the entire software subsystem is 300 hours. That is, the combined reliability of the nine CSCIs must equal or surpass the required mean time between failure (MTBF) of 300 hours. It would be quite natural to break down the reliability requirement by developing a block diagram and assigning each of the CSCIs a component reliability value. However, in order to be consistent with the reality of the ATWCS, no such break down will be done.

Figure 2 shows a high level diagram of the ATWCS Track Control Group (TCG) software architecture. The figure depicts three layers of software: the Application layer, the Support layer, and the Base layer. The CSCIs in the Application layer are the primary system functions of Engagement Planning and Control (EPC), Scenario Generation and Reconstruction (SGR), Mission Data Processing (MDP), and Launch File Server (LFS). EPC and SGR reside on the Secret LAN while MDP and LFS are resident on the Top Secret LAN. Regardless of its resident LAN, each of the applications may use the support and base layers for needed services. The Support layer contains CSCIs that provide required services such as track management and communications which is encapsulated within the Command, Control, Communication and Intelligence (C3I) CSCI; the management of hardware and software resources, represented by the System Manager (SM) and the Startup and Diagnostics (SD) CSCIs; and help facilities which is provided by the Interactive Support System (ISS). All CSCIs in the support layer reside on both

LANs. The Base layer resides on both LANs and provides an interface to various Commercial Off-the-Shelf (COTS) products such as UNIX, X-Windows, Motif, the Informix database, as well a host of configuration data.

The layered software approach has several rules to guide the operation and communication between CSCIs. First, CSCIs in each layer provide services for use by CSCIs in the same or higher layers. Second, CSCIs in lower layers do not rely on any service from a CSCI in a higher layer. Third, the interactions among CSCIs are always initiated from the higher layers to lower layers or the same layer. Lastly, once communications are initiated between CSCIs, data may flow back and forth between layers. Figure 3 shows an alternate view of the layering of the ATWCS TCG CSCIs separated by LAN that more readily depicts the above guidelines.

As previously mentioned, each of the CSCIs are interconnected with other CSCIs. To show examples of the interconnection between software subsystem components (CSCIs), Figure 4 is an example of the interconnections between SM, a support layer CSCI, with the other CSCIs in the software subsystem and Figure 5 is an example of the interconnections between EPC, an application layer CSCI, with the other CSCIs in the software subsystem.

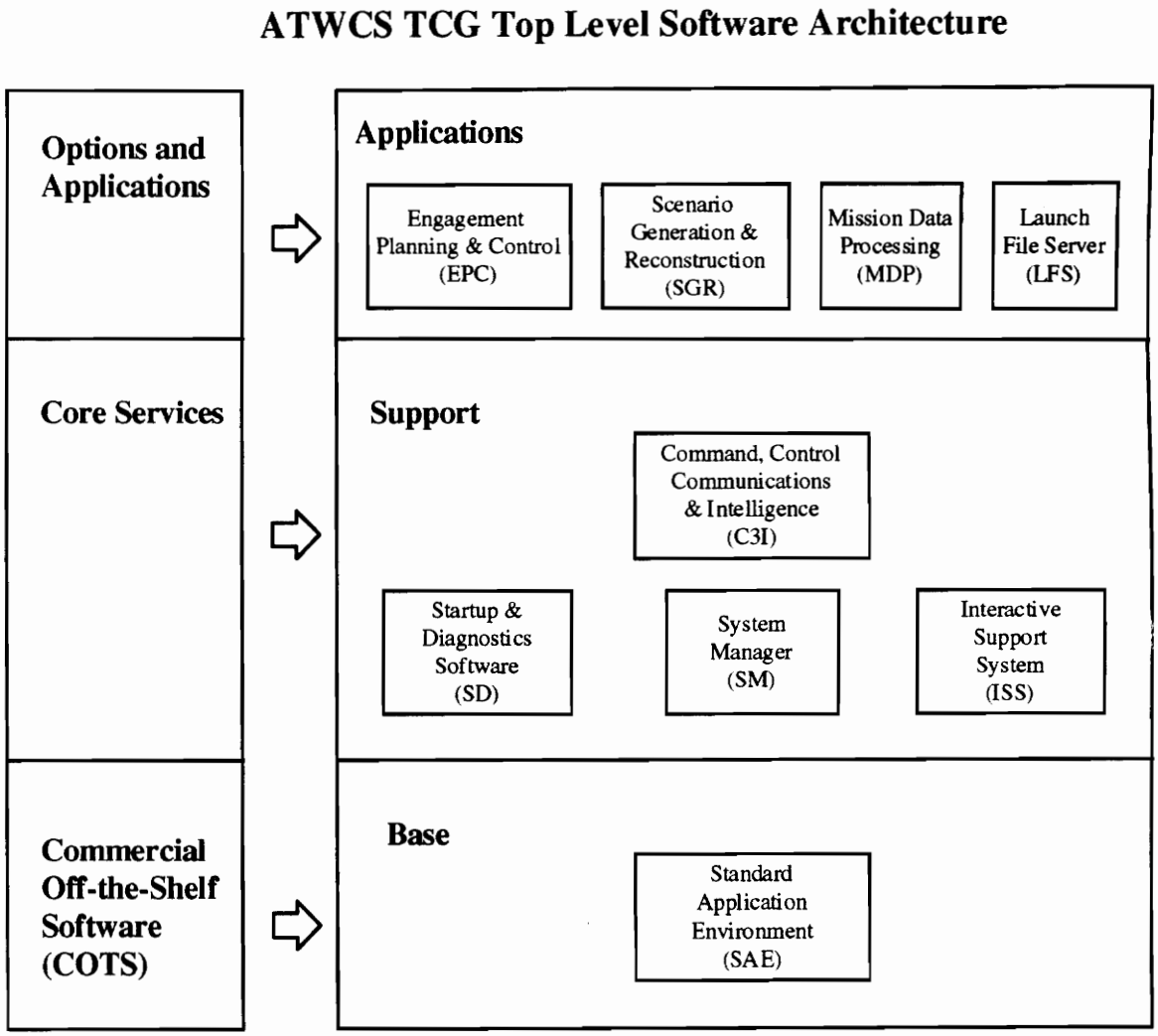
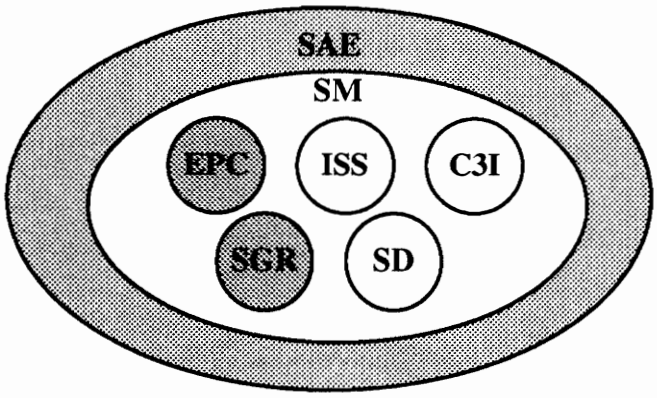
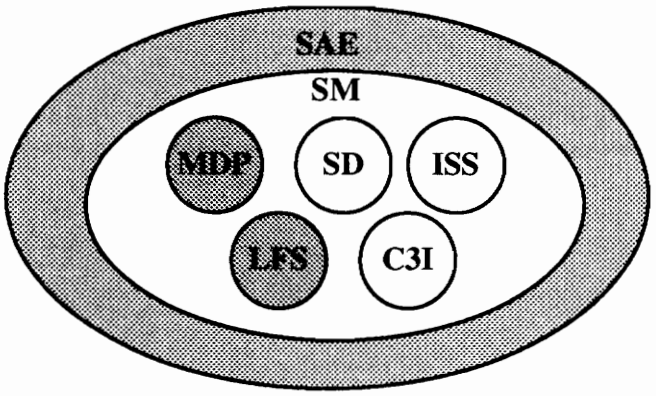


Figure 2. ATWCS Track Control Group (TCG) Software Architecture



Secret LAN (WC LAN)



Top Secret LAN (MD LAN)

Figure 3. ATWCS Track Control Group Layering of CSCIs, by LAN

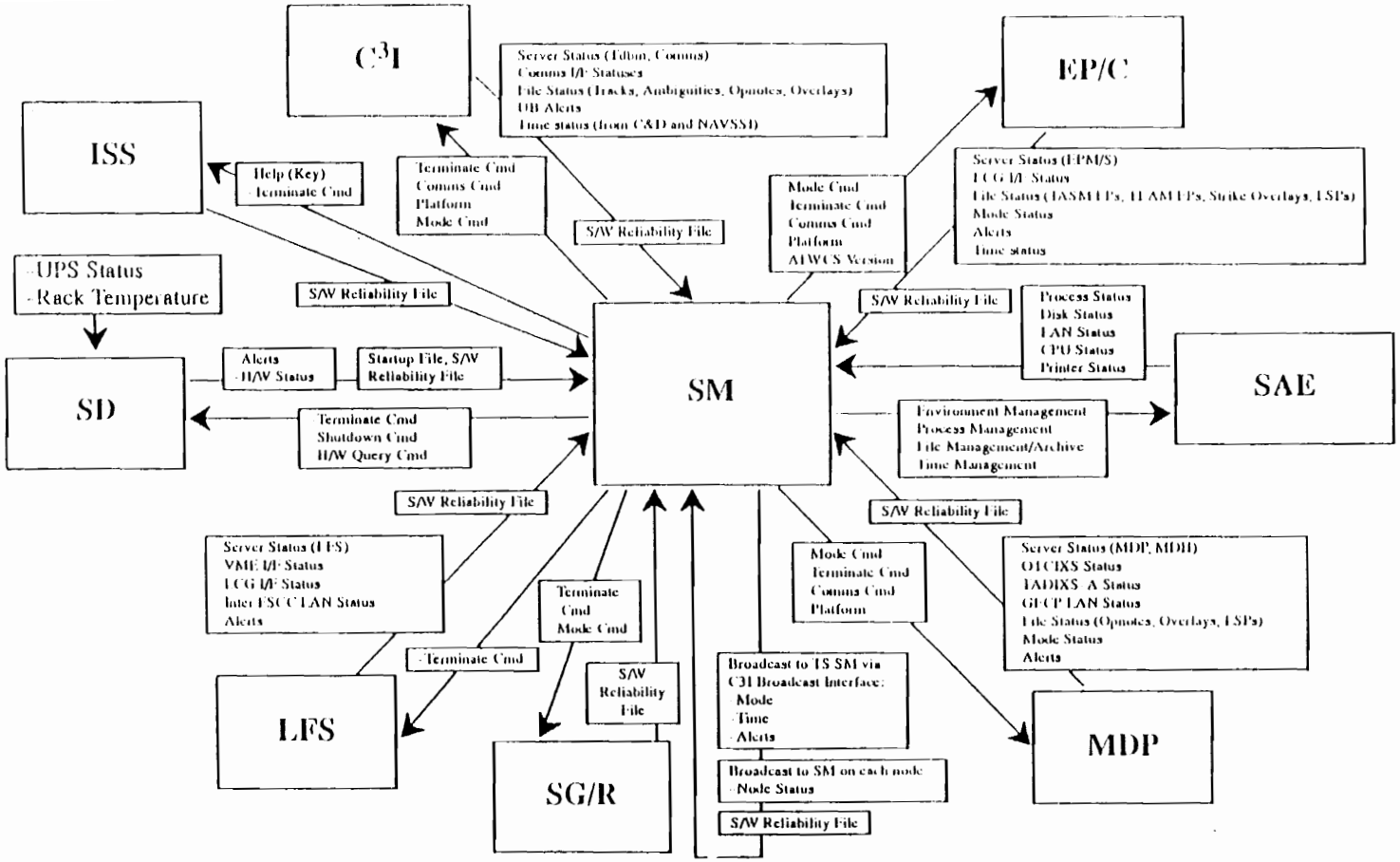


Figure 4. Software Interconnections Between the System Manager CSCI and All Other CSCIs

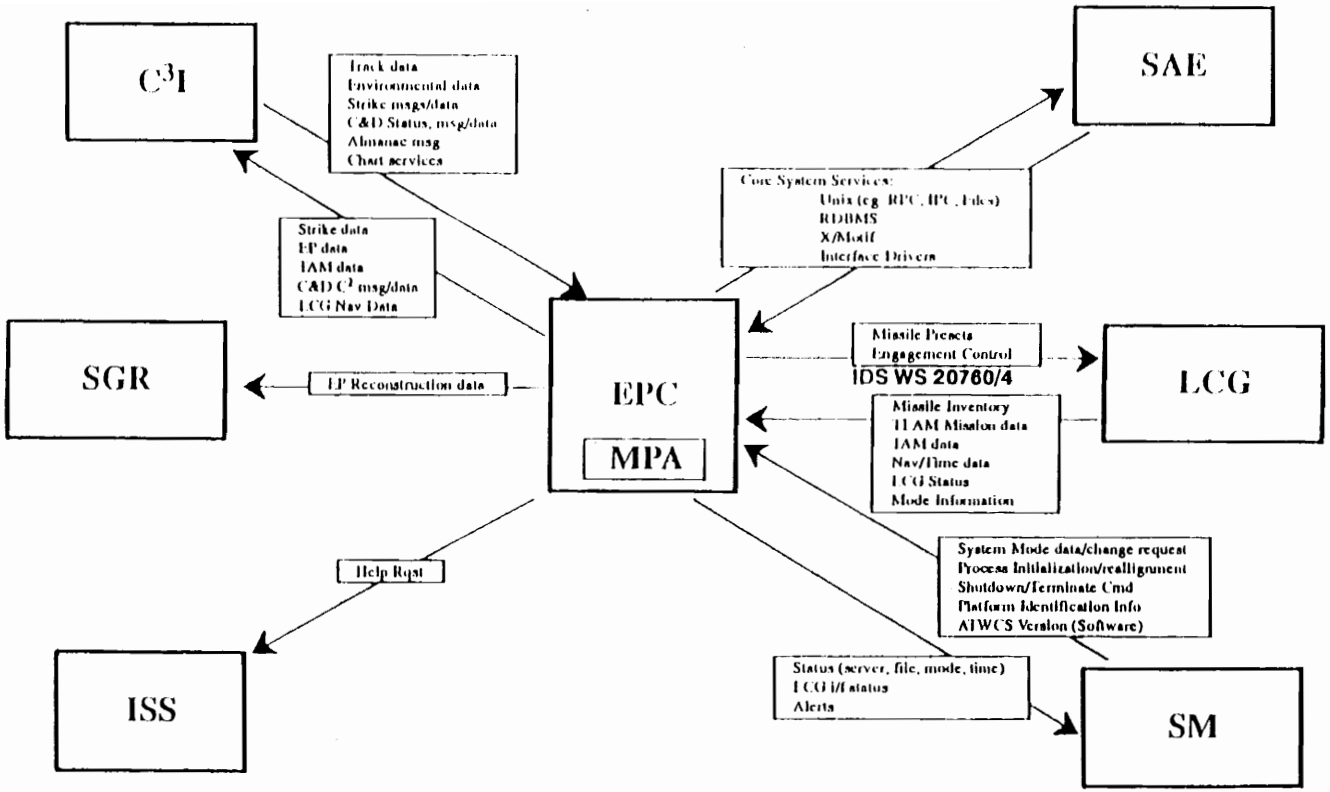


Figure 5. Software Interconnections Between the Engagement Planning and Control CSCI and All Other CSCIs

The previous figures pictorially indicate that each system (hardware and software) contains components, attributes, and relationships. The two systems, or subsystems if you prefer, interact with each other and within their own components to reach the desired objective. The most basic objective of the ATWCS is to successfully put a missile on a given target.

The software subsystem within the ATWCS has been developed under the Department of Defense (DOD) Standard 2167A. Figure 6, which is combination and slight modification of figures from Blanchard (1994), shows how the DOD standard 2167A ties into the system life-cycle process. The shaded boxes in the System Life-Cycle represent the system level activities that result in the corresponding software activities which are shown at the bottom of Figure 6.

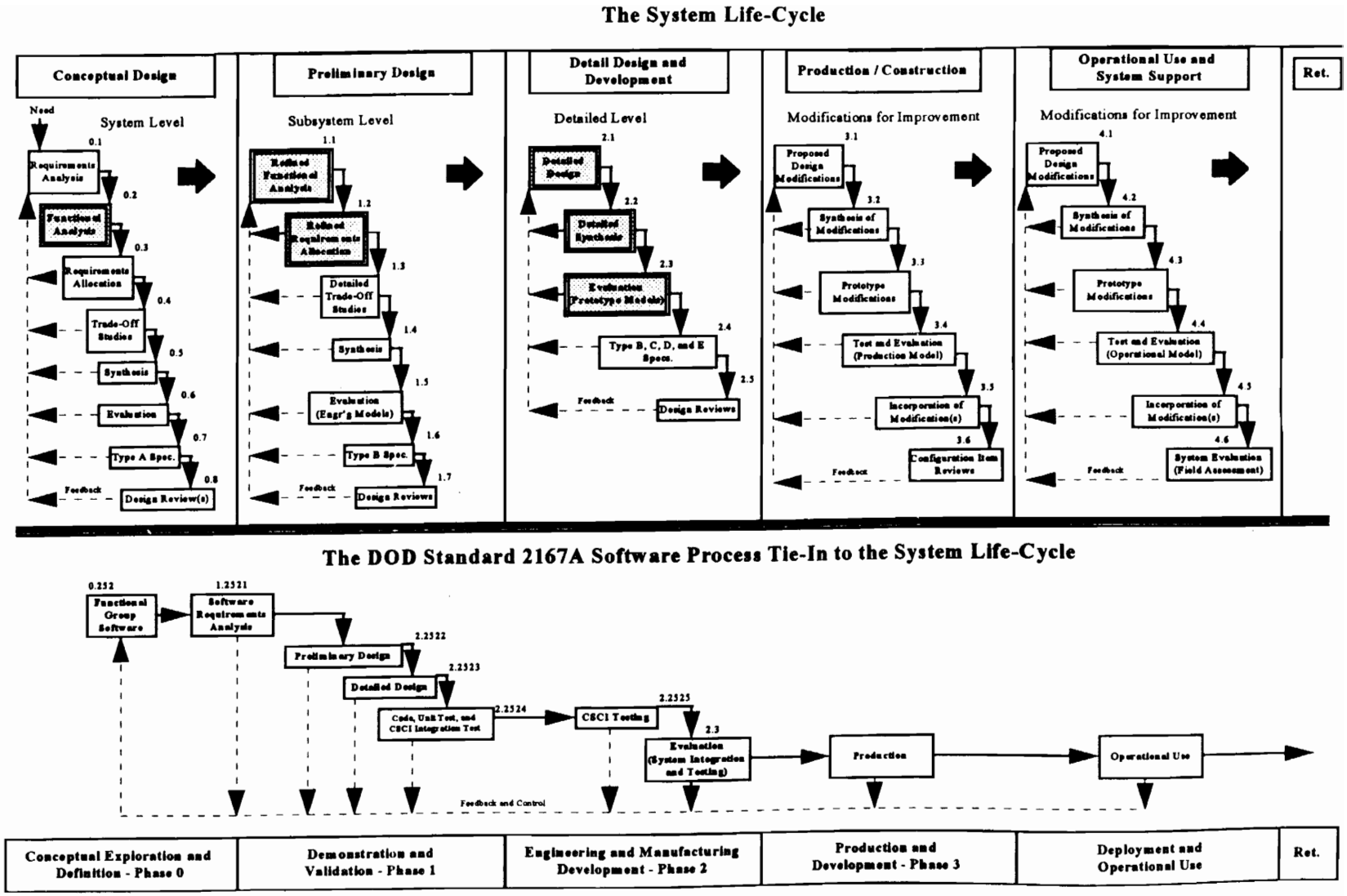


Figure 6. DOD Standard 2167A Tie-In to the System Life Cycle Process

1.3 The Environment

It is highly important to know the environment in which a system will be surrounded and this is no less true for the implementation of reliability estimation methods. Sage and Palmer (1990) indicate that two environments surround a during its lifetime. The first environment a system encounters is its "development environment" and the second environment a system encounters is its "operational environment."

The "development environment" is comprised of the chosen Development Model (Boehm's Waterfall Life-cycle, Yourdon's Structured Life-cycle, the Spiral model, etc.), design methods, management processes, tools, and standards. The list given is not all inclusive. The "operational environment" is the environment in which the system is designed to function. This environment is comprised of more physical attributes such as weather conditions and topography. While a system is in the "development environment" the "operational environment" highly influences the system development.

The ATWCS software system "development environment" is characterized as follows:

- Tailored use of the Department of Defense Standard 2167A
- Object Oriented Design
- Use of UNIX based software development tools
- Contracted and In-house development
- Utilization of Teams (e.g. Validation and Verification Teams, Integration Test Team, CSCI Development Teams, etc.)

and the ATWCS software system "operational environment" is reflected by these characteristics:

- Highly reliable
- Reconfigurable due to hardware unavailability
- Classified and Secure
- Operated by trained professionals
- Resident on U.S. Navy war ships

The environment that will more significantly impact the software system reliability is the "development environment." Design flaws, coding errors, or other development related error injection causes will all occur within this environment. The development environment is constantly revisited during the maintenance phase of the system life cycle. Any time the system is within the development environment, there is the possibility of introducing errors into the system. Failures caused by the "operational environment" will mainly come from the system being used in a manner not intended.

The ATWCS has numerous so called mission scenarios in which it may be utilized. When considering an overall operational profile for the system, the profile is wide ranging. From intensive use of C3I capabilities in combination with the multiple active engagement plans and crucial data from the MDP component in a combat situation to a very relaxed use of the tutorials provided by the ISS software component combined with a very light use of all other components. When taking into account the use of reliability models, this wide ranging operational profile is considered in conjunction with the system and its development and operational environments.

1.4 Relevant Definitions

Reliability in terms of software systems is defined in a multitude of ways. Sage and Palmer (1990) define reliability as the extent to which software can be expected to perform its intended function with required precision. Shere (1988) defines reliability as the probability that the software item does not fail for a specified period. The Reliability, Maintainability, and Supportability Guidebook (1992) defines reliability as the probability that software will not cause the failure of a system for a specified time under specified conditions. The latter two definitions are similar and both have the main theme of something called a failure occurring in a given time-frame. The first alludes to the software not encountering a failure.

This leads to the definition of a failure. The prevailing literature seems to enjoy the creation of unnecessary confusion surrounding terms fault and failure. Musa et. al. (1987) provide the definition of a failure as a departure of the external result of program operation from the defined requirements. A failure is a dynamic characteristic of software systems and cannot occur unless the software system is operating.

The classical definition of a fault is something that causes a failure, a "bug." This paper will not spend time portraying faults versus failures. The intention is that when discussing the reliability of software subsystems any reference to errors, bugs, faults, departures from requirements, or any other synonymic connotation will be in reference to the occurrence or possible occurrence of a "failure."

Prediction and estimation are terms that are used throughout the literature and are applied to the methods used to determine the specific parameters used by the models.

According to Aggarwal (1993), the definition of prediction when discussing model parameters is that the parameter values are determined by the properties of the software and of the development process. Estimation of model parameters refers to the parameter values being the result of a statistical procedure applied to failure data from the software under evaluation.

Additionally, both terms are used interchangeably by a host of authors when discussing the statistical determination of a reliability measure based upon current failure data. Troy and Moawad (1985) indicate that the prediction of software system reliability is a statement about the reliability based on the physical program characteristics and that estimation of software reliability is composed of relating failures to the probability of failures at a future time. For the purposes of this project, the term estimation is used to represent the statistical determination of a reliability measure, representing the probability of a failure at a future time, based upon current failure data. The utilized failure data is obtained from the software system under evaluation. Historical failure data from other similar software systems will not be used.

Reliability measures are generally defined in terms of time. Execution time and calendar time are the predominant basis for measuring time. Execution time for a software system is the amount of time that the processor(s) is actually executing instructions. Calendar time is the regular every day time. Quite often it is easier to collect failure data based on calendar time, but in practice, execution time has become the desired time base. The reason being is that calendar time often results in a significantly greater time value than that of the actual execution time. For example, if a computer is turned and the

software is idle, no instructions are being executed. This can result in 40 hours of calendar time for which the system is operational, but in reality, instructions were being executed in only a small portion of that time. Reliability models use varying definitions of time. This project will primarily concentrate on execution time, as it produces more accurate reliability measures.

Chapter 2

Criteria for Comparison and Evaluation of Software Reliability Models

2.1 The Comparison Criteria

In order to objectively compare software reliability models, comparison criteria need to be defined. Aggarwal (1993) suggests several comparison criteria which include predictive validity, capability, quality of assumptions, applicability, and simplicity. These criteria are general characteristics of what is considered a "good" reliability model. To obtain a more in-depth understanding of the criteria, the criteria are defined below. Furthermore, Troy and Mowad (1985) suggest two characteristics that describe the interfaces between the user, the model, and reality. A third characteristic is suggested which is a multi-dimensional definition of validity, see Figure 7. Each of these characteristics is described where appropriate in conjunction with Aggarwal's criteria.

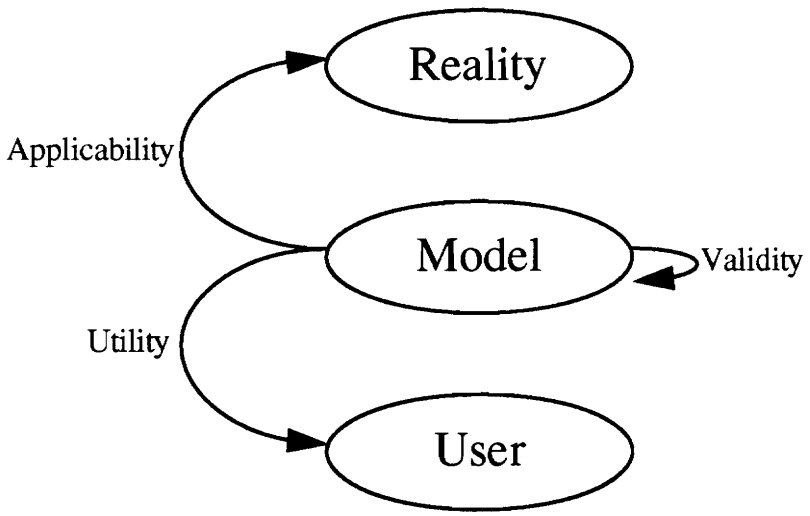


Figure 7. Characteristics of the Interfaces between Reality, the Model, and the User.

Predictive validity refers to the model's ability to accurately predict future failures from data representing present and past failure occurrences. The predictive validity ability of the model is highly important when modeling the reliability of a software system that is subject to a regular repair cycle. This can occur both during in-house testing and while the system is in operation. This criteria can be measured by plotting the relative error against the test time. As time passes, the relative error will approach zero. Negative points indicate a low estimate while positive points indicate a high estimate of the number of errors left. The closer to zero the points are, the more accurate the prediction and therefore a high degree of predictive validity.

Validity is also discussed at the operational, structural, and conceptual levels by Troy and Mowad (1985). At the operational level, predictive validity is characterized as above (with the addition of accuracy of the confidence interval and quality of fit of the model to reality) and it is paired with replicative validity (the aptitude of the model to reproduce previously measured system behaviors) to represent the more broad measure of Output Validity. Replicative validity can be measured by comparing the model's output against the behavior of the actual data. Input validity, which includes measures of data accuracy and impartiality, is the second component of the operational level. The structural level of validity is characterized by Parameter Estimator Validity (consisting of measures of existence, uniqueness, convergence of the algorithm, and accuracy) along with Mathematical Formulation Validity (consisting of the verification and correctness of the mathematical foundations of the model). Lastly, the conceptual level of validity relates to

Aggarwal's quality of assumptions but takes a step further by validating the hypothesis on which a given model is based.

There are numerous validity criteria, many of which deal with the mathematical and hypothetical aspects of reliability models (i.e. the Structural and Conceptual levels of validity). An assumption made in this paper is that those who have developed reliability models (and those who will in the future) have indeed validated their model(s) in respect to the mathematical and hypothetical domains. Therefore, only the operational levels of validity is utilized as comparison criteria.

Capability is the model's capability to accurately estimate various metrics required by those managing software development projects or operating software systems. There are three measures to consider for the degree of capability. First, and most importantly is the model's ability to estimate the system's present reliability, perhaps in terms of Mean Time to Failure (MTTF) or the Failure Intensity. Secondly, the model should be able to accurately estimate the date that a specified reliability (MTTF or Failure Intensity) is achieved. Lastly, the model should provide an accurate estimate of resource and cost information that is required to achieve the specified reliability. Troy and Moawad (1985) label this criteria as "utility" and indicate that utility applies to the relation between the model and the user of the model. The above metrics associated with capability all are intended to provide the user of the model a quantitative value in which to better manage the development and/or operation of software systems.

Quality of Assumptions is a fairly subjective measure that can greatly aid in the determination of whether a particular model applies to your software system. It may be

possible to test assumptions as long as there is data that is directly related to the assumption. More likely however, is an evaluation of the assumptions from the perspective of engineering experience and logical consistency. Additionally, each assumption should be evaluated on the level of clarity and explicitness in which they are stated.

Applicability, according to Aggarwal (1993), contains a broad range of model characteristics. Models should be evaluated on how well they function across differing software systems (size, structure, etc.) and differing development and operational environments. There are four special situations that a model should be capable of handling. The four situations are program evolution, classification of failure severity, the ability to handle incomplete failure data, and the ability to operate the same program on computers of different performance capabilities.

Troy and Moawad (1985) indicate that applicability applies to the relation between the model and the real system and addresses whether or not a given model should be approved for use for a given system operating in a given environment. The definition of applicability from Aggarwal addresses this by designating several of the parameters to be measured. Applicability is a key criteria in the evaluation of software reliability models. A software system's development and operational environments can greatly differ from one system to another. Reliability models that are selected for use must be able to reflect the realities of the target system(s).

This project will use a slightly modified definition for applicability. The statement that Aggarwal (1993) makes relating to evaluating a model on how well it functions across

differing software systems and differing development and operational environments will not be used. Although a model that has the ability to function well across software systems and environments is a nice feature, it does not necessarily make the model applicable to the target system/environment. Therefore, the measure of applicability will be based on a model's ability to function well when applied to the target system and environment.

Simplicity has three qualities. First and most importantly is the fact that it must be easy and inexpensive to collect data for the model. Secondly, the model should be based upon fairly simple concepts. That is, people without an extensive mathematical background should feel comfortable with the model and its assumptions. The last quality of simplicity is that the model must be easily implemented as a computer program and have a relatively fast execution time.

See Table 1 for a summary of the comparison criteria, a definition of each, and the measurement techniques that are applied. A significant reliance on expert opinion within the measurement techniques is worthy of note. Many of the criteria are subjective in nature and require expert opinion to achieve accurate results. Measurements are based upon data when possible.

Table 1. Comparison Criteria Summary

Criteria		Definition	Measurement Technique
Replicative Validity		The aptitude of the model to reproduce previously measured system behaviors.	• Comparison of the model's output and the behaviors of the actual data
Output Validity (Predictive Validity)	Accuracy	The ability of the model to produce strong or weak predictions.	• Model bias analysis.
	Quality of Fit	The fit between the model predictions and what is actually occurring.	• Goodness-of-Fit Test (Kolmogorov-Smirnov test, Chi Square)
Input Validity	Accuracy	The ability to correctly identify and measure what is desired.	• Expert opinion on data collection.
	Impartiality	The assurance that the data is recorded correctly	• Expert opinion on the condition of data collection.
Capability		The model's ability to estimate metrics required to manage.	• Present Reliability • Time-frame for Target • Resource and Cost
Quality of Assumptions		The quality of a model's assumptions.	• If data available, use to measure assumption. • Expert opinion based on logical consistency and engineering experience. • Expert opinion on clarity and explicitness.
Applicability		The model's ability to function well when applied to the target system and environment.	• Expert opinion on the match between the model, the system, and the environment factors.
Simplicity		Ease of data collection, model has simple concepts, and it is easily programmed.	• Expert opinion on the three aspects of simplicity and comparison of models.

Reliance on expert opinion is not necessarily bad, however, the availability of experts is required and may be a stumbling block. In an effort to reduce the reliance on expert opinion, efforts to lessen the necessity of expert opinion in the evaluation and increase the ability to have quantifiable measures to evaluate the criteria should be investigated.

2.2 Weighting the Comparison Criteria.

Each of the comparison criteria all are important characteristics of reliability models, however, some are more significant than others. Additionally, the environment in which the model will be used impacts the significance of the criteria. For example, a system that is to be utilized in a widely diverse set of environments would cause the applicability criteria to be heavily weighted. In a like manner, an organization with

employees having extensive statistical methods experience, clearly defined data collection processes, and access to highly skilled programmers, the simplicity criteria would be weighted very lightly. A weighting factor of 0.10 to 1.00 will be combined with each score to produce a weighted total score for each model. The weighting factor assigned to each criteria is an outcome of its importance to the target system, ATWCS.

The most important criteria is predictive validity. One of the most fundamental reasons for modeling software system reliability is to obtain a reasonable assurance that the system will perform reliably for a certain period of time. Two measures of predictive validity are used and each is assigned a weighting factor of 1.00.

Capability of a model is fairly consistent attribute amongst reliability models. An assumption concerning reliability models is that they will provide the measurements (metrics) you require. Most models are capable of producing the standard metrics and the standard reliability metrics are used by the ATWCS management. Therefore, capability is assigned a weighting factor of 0.50.

The quality of a model's assumptions and their match to a particular system greatly aid in the overall selection of a model. The assumptions should closely match the system environments. The ATWCS environments are somewhat less common and therefore a good fit is necessary. A weighting factor of 0.75 is assigned to the quality of assumptions criteria.

Applicability is a key criteria. If the model does not match the environment and the system, the likelihood of producing accurate reliability measures is greatly lessened. Due to this importance, applicability is assigned a weighting factor of 0.90.

The simplicity criteria is less important than probably any of the other criteria. If a model exists that is "perfect" with respect to all other criteria but is vastly complex, the tradeoff would be most likely be made. The aspect of ease of data collection is however, the most important of the three simplicity characteristics. If data collection is difficult, experience has shown and many authors agree that the model will not be used. In respect to ATWCS, the ease of the data collection process is important and therefore simplicity is assigned a weighting factor of 0.35.

The last criteria, replicative validity, is an important criteria. If a model cannot closely match past behaviors, the confidence that it can accurately estimate future behaviors is greatly lessened. The ATWCS community definitely values a high degree of confidence in its reliability measures. Therefore, replicative validity is assigned a weighting factor of 0.80.

Chapter 3

An Overview of Software Reliability Models

3.1 Categories of Software Reliability Models

In an attempt to organize the growing list of software reliability models into a manageable assembly, various categorization strategies have been developed. These strategies have identified similar characteristics among the models and have grouped those with like properties. Three separate categorization schemes are presented below.

One strategy is to categorize the models by their unit of error measurement. The Reliability, Maintainability, and Supportability Guidebook (1992) utilizes this method and defines three broad categories: Time to Failure, Failure Count, and Fault Seeding Models. The following descriptions are adapted from the above literature.

The Time to Failure Models anticipate that successive times between failures increase as errors are found and corrected. An assumption representative of these models is that corrections are made with certainty, that is the errors are fully corrected and no new errors were injected. Time is represented by either execution time (CPU time) or calendar time. The data input to the model must be evaluated prior to its use to ensure that it is consistent with the model assumptions. Examples of models in this category would be the Jelinski-Moranda model and the Schick-Wolverton model.

The Failure Count Models anticipate that the number of failures that occur in a successive time periods decreases as testing progresses. These models use the estimation of the total number of failures in the software to know when all of the failures have been

detected and therefore corrected. Once again, time is represented by either execution time (CPU time) or calendar time and the data input to the model must be evaluated prior to its use to ensure that it is consistent with the model assumptions. Examples of models in this category include the Goel-Okumoto model (non-homogeneous Poisson process), the Musa time execution model, and the Shooman exponential model.

The Fault Seeding Models operate on the basis of injecting a known number of errors into the software system. It is assumed that the system contains an unknown number of errors. The assumption is made that the probability of discovering an unknown error is the same as it for discovering a seeded error. Testing is completed when all seeded errors are discovered. An example of this model is the Mills hypergeometric model.

Musa et. al. (1987) use another method. They use two categories of models based on whether the model assumes Finite Failures or Infinite Failures. Within each of these categories, the models are further classified in terms of time domain, type, class (only for the finite failure category), and family (only for the infinite failure category). Time domain represents whether the model uses execution time or calendar time. Type refers to the distribution of the number of failures experienced, class is the functional form of the failure intensity in terms of time, and family is the functional form of failure intensity in terms of expected number of failures. To best describe these two categories, the following tables have been adapted from Musa et. al (1987):

Table 2. Finite Failures Categorization of Models

Class \ Type	Poisson	Binomial	Other types
Exponential	Musa (1975) Moranda (1975) Schneidewind (1975) Goel-Okumoto(1979)	Jelinski-Moranda (1972) Shooman (1972)	Goel-Okumoto (1978) Musa (1979) Keiller-Littlewood (1983)
Weibull		Schick-Wolverton (1973) Wagner (1973)	
C1		Schick-Wolverton (1978)	
Pareto		Littlewood (1981)	
Gamma	Yamada-Ohba-Osaki (1983)		

Table 3. Infinite Failures Categorization of Models

Family \ Type	T1	T2	T3	Poisson
Geometric	Moranda (1975)			Musa-Okumoto (1984)
Inverse Linear		Littlewood-Verrall (1973)		
Inverse Polynomial (2nd degree)			Littlewood-Verrall (1973)	
Power				Crow (1974)

Lastly, Shere (1988) uses the phases of the life-cycle to classify reliability models. Each model or model type is associated with the life-cycle phase(s) in which it is deemed most applicable. The following table is adapted from his text.

Table 4. Life-Cycle Classification of Reliability Models

Life Cycle Phase	**Reliability Model**
Development (debugging)	Error-Counting models Deterministic (Poisson models) Bayesian Markov Nonerror-counting models Deterministic Stochastic Bayesian
Validation phase	Nelson's model Shooman (path reliability) Input-domain-based model
Operational phase	Input-domain-based model Markov processes
Maintenance phase	Input-domain-based model
Correctness measures (test reliability)	Error seeding Phenomenological (Halstead) Statistical Input domain

3.2 Description of Several Reliability Models

This section will provide a basic description of several software reliability models. The models were selected for inclusion in this project to serve various purposes. First, a representative cross section of existing models is desired, i.e. models representing various categories as listed above. Secondly, each model has shown an ability to be accurate in its capability to estimate reliability for a given system and environment. With the exception of the Yamada model, each of the models has been utilized on a wide variety of systems. Lastly, the models are among the most popular models currently being utilized in the software industry. The selection of the four models is in no way intended to be a statement indicating that these are the absolute best models for use. A proper evaluation of all available software reliability models greatly exceeds the scope of this project. The models selected are Musa's Basic Execution Time Model, the Jelinski-Moranda Model, the Logarithmic Poisson Execution Time Model, and the S-Shaped Reliability Growth Model. A description of each follows.

Musa's Basic Execution Time Model

The equation used to represent the Basic Execution Time model with τ representing the execution time is: $\mu(\tau) = \upsilon_0 \left[1 - \exp\left(-\frac{\lambda_0}{\upsilon_0} \right) \right]$, where $\mu(\tau)$ is the total number of failures, υ_0 is the total number of failures that are expected to occur, and λ_0 is the initial failure density. The total number of failures expected to occur, $\mu(\tau)$ is

determined by $\upsilon_0 = \frac{\omega_0}{B}$, where ω_0 is the number of inherent faults and B is the fault reduction factor such that $\omega_0 = \omega_I \Delta I$, where ω_I is the inherent faults per line of computer program source code and ΔI is the total number of the lines of source code within the developed software system (or component). The initial failure intensity, λ_0 is determined by $\lambda_0 = f K \omega_0$, where $f = \frac{r}{I}$ is the linear execution frequency, such that r is the instruction execution rate (generally referred to in millions of instructions per second, or MIPS) and I is the total number of executable instructions. Additionally, $K = \frac{1}{Bf}\frac{\lambda_0}{\nu_0}$ is the fault exposure ratio where $\frac{\lambda_0}{\nu_0}$ is estimated.

An extremely interesting feature of the Basic Execution Time Model is the existence of the fault reduction factor, B. This component of the model allows for the all to true fact that software system fault removal is an imperfect process, just as the initial development process produces an imperfect product. Musa et. al. show the fault reduction factor is the ratio of the net number of faults corrected to the number of faults experienced. A clear example of this is if 100 failures were detected and 99 were fixed but then 4 new failures were created. The total number of faults corrected was 99, but the net number of faults corrected is 95. Therefore, the value for B is 0.95. Musa et. al. indicate that the initial value for B must be estimated based upon the following criteria, the descriptions of which have been paraphrased from his text:

- the degree to which it is recognized that given failures may be related to the same defect,
- the proportion of failures whose underlying defects can be found,
- the comprehensiveness of code inspections,
- the degree to which the relations between faults are understood, and
- the extent to which new faults are injected as a result of the correction process.

The capability to determine a value for B requires extensive knowledge and data of previous efforts by the development group and organization. Until that time occurs, studies have been done to determine a "project-independent" value for B. It is recommended that the latest literature be reviewed to obtain the latest data, however, figures supplied by Musa et. al. indicate that a value of 0.94 is representative of fairly large software systems while a value of 0.99 may be applicable to smaller software systems. The assumptions that underlie the model are as follows:

- The software is operated in a manner that is consistent with the expected usage upon operational deployment.
- All failures are equally likely to occur and are independent of each other.
- There is a finite number of faults within the software system.
- The debugging rate is proportional to the failure rate (debugging is imperfect).
- All software faults are observable (whether during execution or during analysis of data captured while the system is in operation).
- The execution times between failures are exponentially distributed.

The Jelinski-Moranda Model

The equation used to represent the Jelinski-Moranda model is:

$R_i(t) = \exp\left[-\phi(N_0 - i + 1)t\right]$, where $R_i(t)$ is the Reliability at any given time, N_0 is the estimated number of initial software faults, and ϕ represents the estimation of the constant of proportionality (the proportion of the number of faults remaining and the failure rate). Both N_0 and ϕ are estimated by utilizing the Maximum Likelihood method. For the theory surrounding this method, the text by Lloyd and Lipow (1977) is a good reference. The text presents the basic formula for determining maximum likelihood as $L = \theta^{-n} \exp\left[-\frac{1}{\theta}\sum_{i=1}^{n} t_i\right]$. Applying this method to the Jelinski-Moranda model yields the following: $N_0 = \sum_{i=1}^{N}(N_0 - i + 1) - 1$ and $\phi = N\left[\sum_{i=1}^{N}(N_0 - i + 1)t_i\right]^{-1}$.

The assumptions that underlie the model are as follows:

- The software is operated in a manner that is consistent with the expected usage upon operational deployment.
- All faults are equally likely to occur and are independent of each other.
- Debugging does not introduce additional faults and each fault is corrected at the time of occurrence.
- The failure rate is proportional to the current fault content of a software system and changes by a constant amount with the correction of each failure.

- The failure intensity in terms of expected number of failures is a binomial exponential distribution.

The Logarithmic Poisson Execution Time Model

Musa and Okumoto collaborated on this model and in 1984 presented it at the Seventh International Conference on Software Engineering. The model is considered to be a member of the Geometric family. The equation used to represent the Logarithmic Poisson Execution Time Model with τ representing the execution time is:

$\mu(\tau) = \frac{1}{\theta}\ln(\lambda_0\theta\tau + 1)$, where $\mu(\tau)$ is the total number of failures, θ is the failure intensity decay parameter (which must be greater than zero), and λ_0 is the initial failure density. The failures intensity decay parameter, θ, can be determined by plotting the failure intensity with respect to the faults remaining. The slope of the resultant curve is the value for θ. The initial failure density is an estimated value which is obtained via the maximum likelihood method in the same manner as stated for the Jelinski-Moranda Model.

The model's use of an exponentially decreasing failure intensity function indicates that a significant number of failures are expected to occur and be corrected early in the testing or operational time period due to the existence of a "frequent execution data set." That is, there is a highly utilized set of software components of which the vast majority of their faults are detected and corrected resulting in a significant decrease in the failure intensity. However, in direct relation to this is the existence of a "irregularly executed

data set" whose faults may remain undetected for extended periods which result in an increasingly gradual decrease in the failure intensity over time.

The assumptions that underlie the model are as follows:

- The software is operated in a manner that is consistent with the expected usage upon operational deployment.
- There is an infinite total number of failures, i.e., the software system will never be 100% fault free.
- The fault detection rate forms a geometric progression and is constant between fault occurrences.

The S-Shaped Reliability Growth Model

Yamada et. al. noticed that the actual reliability growth curve for several applications of the model developed by Goel and Okumoto (the Non-homogeneous Poisson Model, not described within this paper) was S-shaped around the estimated curve. They inferred that the s-shaped curve was attributed to a learning process through which the testing community needed to become familiar with a new test environment and/or a new test subject. As time passes, the testing community's testing skills improve.

Yamada et.al. modified the Non-homogeneous Poisson Model (NHPP) such that the NHPP's mean value function shows an S-shaped growth curve. The equation used to represent the S-Shaped Reliability Growth Model is:

$R(t|s) = \exp\left[-a\left\{(1+bs)e^{-bs} - (1+b(t+s))e^{-b(t+s)}\right\}\right]$ where $R(t|s)$ is the Reliability at any given time t, such that the last failure occurred at time s, a is the expected number

failures to be eventually detected, and b is the rate of failure detection per failure. The values for a and b can be determined by evaluating $y_n = a\left[1-(1+bt_n)e^{-bt_n}\right]$ where y_n is the number of failures that were detected up to time t.

The assumptions that underlie the model are as follows:

- The software is operated in a manner that is consistent with the expected usage upon operational deployment.
- The software system is subject to failures at random times caused by errors present in the system.
- All faults are equally likely to occur and are independent of each other.
- Each failure has the same impact upon the system as any other failure.
- The initial fault content of the software system is a random variable
- The time between failures depends on the time to failure.
- Faults are corrected immediately without injecting new faults.

Chapter 4

Evaluation of Software Reliability Models

4.1 Automated Tool Use

A tool being used to aide in the evaluation of the reliability models is a software package named SMERFS (Statistical Modeling and Estimation of Reliability Functions for Software). SMERFS (1993) was developed by Dr. William Farr of the Naval Surface Warfare Center (Dahlgren Division) and Oliver Smith of EG&G, Washington Analytical Services Center Inc. The software package has implemented each of the following models and provides a variety of statistical functions such as the Goodness-of-Fit, Model Bias, Model Trend, and Prequential Likelihood functions.

4.2 Data Collection Procedures

The data being used for this project has been obtained from the first three weeks of the System Test phase of the ATWCS development cycle. There is no significance associated with obtaining data for only the first three weeks of the System Test phase. The time frame surrounding the test phase corresponds directly with the development of this project and therefore, there is no additional data available at this time.

Some concern may arise due to the relatively small data set and whether or not there is enough data to determine if any one model is truly suitable for use on the target system. The burden of determining an adequate data set size is left to future research considerations. The unique timing of this project in relation to the actual ATWCS testing

phases provides a real world situation in regards to the size of the data set and to the difficulties faced by those assessing the software reliability.

Triantis (1995) stated, "Modeling in any situation presupposes the existence of an effective information system that provides the data required for the model." During testing, test logs were maintained by the test personnel. Start and stop times for each test were recorded along with Trouble Report numbers associated with any failures that were observed. The ATWCS community maintains a database of all Trouble Reports and their status's (valid, invalid, corrected, under investigation, etc.). Only valid Trouble Reports were considered for inclusion in the final data set. Additionally, each Trouble Report is assigned a severity level of either 1, 2, 3, 4, or 5. The seriousness of the severity levels is greatest at level 1 and is the lowest at level 5. A level 1 problem is generally representative of the software system being completely unusable, while level 5 is most often a trivial error.

The level of input validity associated with this data is considered fairly high. The data recorded was done in a consistent fashion. The ability to identify errors in the system is an unquestionable characteristic in the personnel performing the tests and the desired data associated with each failure is clearly defined.

4.3 Model Independent Comparison Criteria

Prior to a discussion concerning each model and its ability to adequately address the comparison criteria, the one criteria that is model independent will be discussed. Input Validity has little to do with any particular reliability model. The models do perform more

accurately when supplied with valid data, however, the ability to have valid data is not the responsibility of the model. Rather, it is the responsibility of the engineer or manager who is implementing the model. Without accurate and impartial data, the output of any reliability model can easily be considered suspect. The "garbage in garbage out" rule-of-thumb applies. Therefore, it may be appropriate to define an organizational (or project wide) process for the collection of reliability data. With such a process, the desired data is clearly specified as are the procedures to record the data.

Also noteworthy is the fact that each of the software reliability models is compared against the same data set. The data set used for the interval-time model is the same data, but simply converted into a different format.

Input validity is also important in the comparison of reliability models across similar projects. That is, projects that are developed in similar environments and operating in or targeted for similar environments. The comparison of models across projects will be less effective and may produce misleading results unless a measure of input validity is known.

4.4 Evaluation of Replicative Validity

The SMERFS tool was utilized to perform a replicative validity test on the data for each of the models. Each model used failures 1 to i/2 as the basis its calculations, where i is the total number of failures. Each model was then instructed to produce estimations for the next (i/2 + 1) to i failures. The results of the tests appear on Figure 8 and Figure 9.

The actual data is also plotted on the two graphs and is represented by the plus sign (+).

A short discussion on replicative validity appears for each of the models in section 4.5.

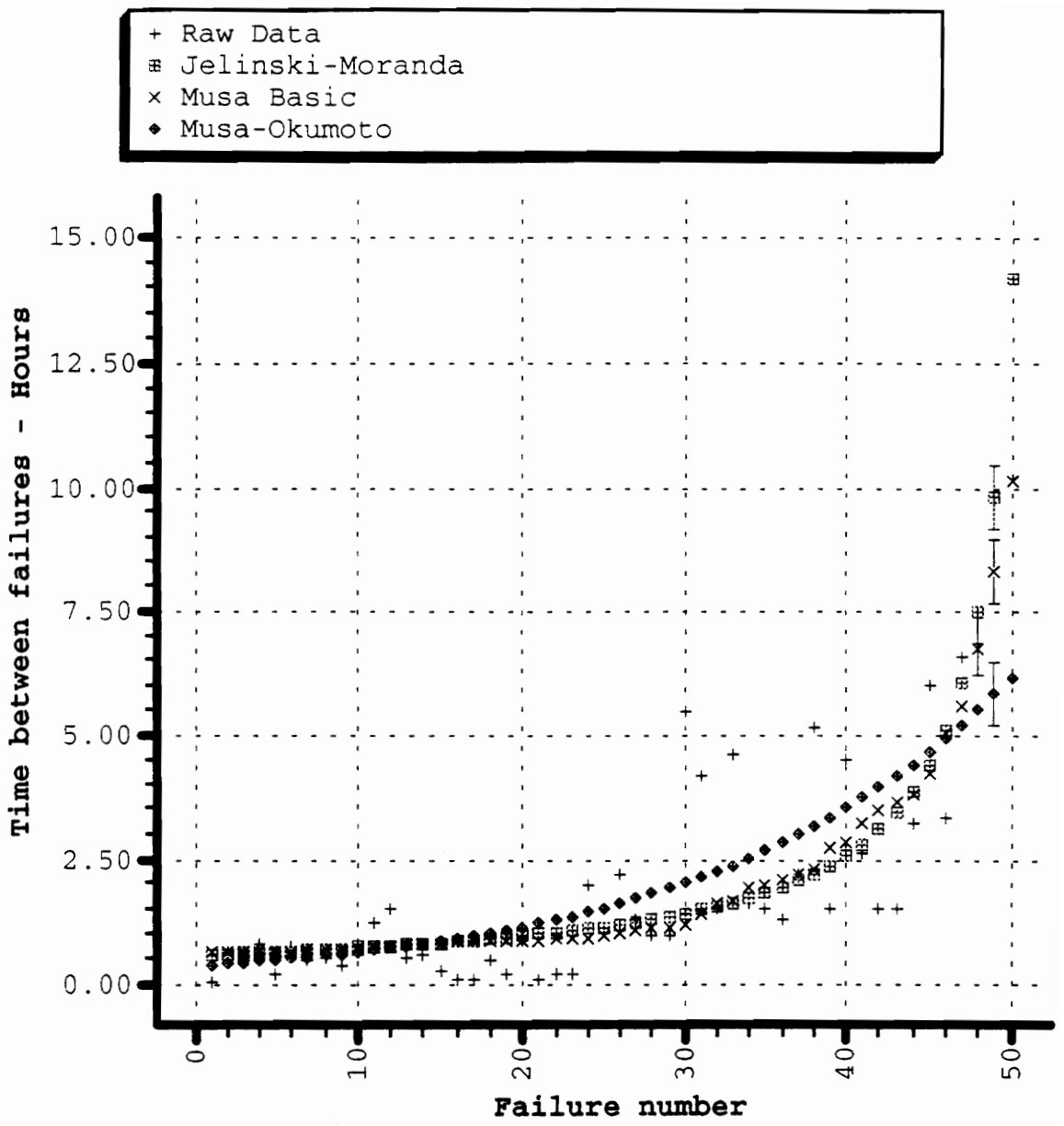


Figure 8. Time Between Failures: Execution Time Models

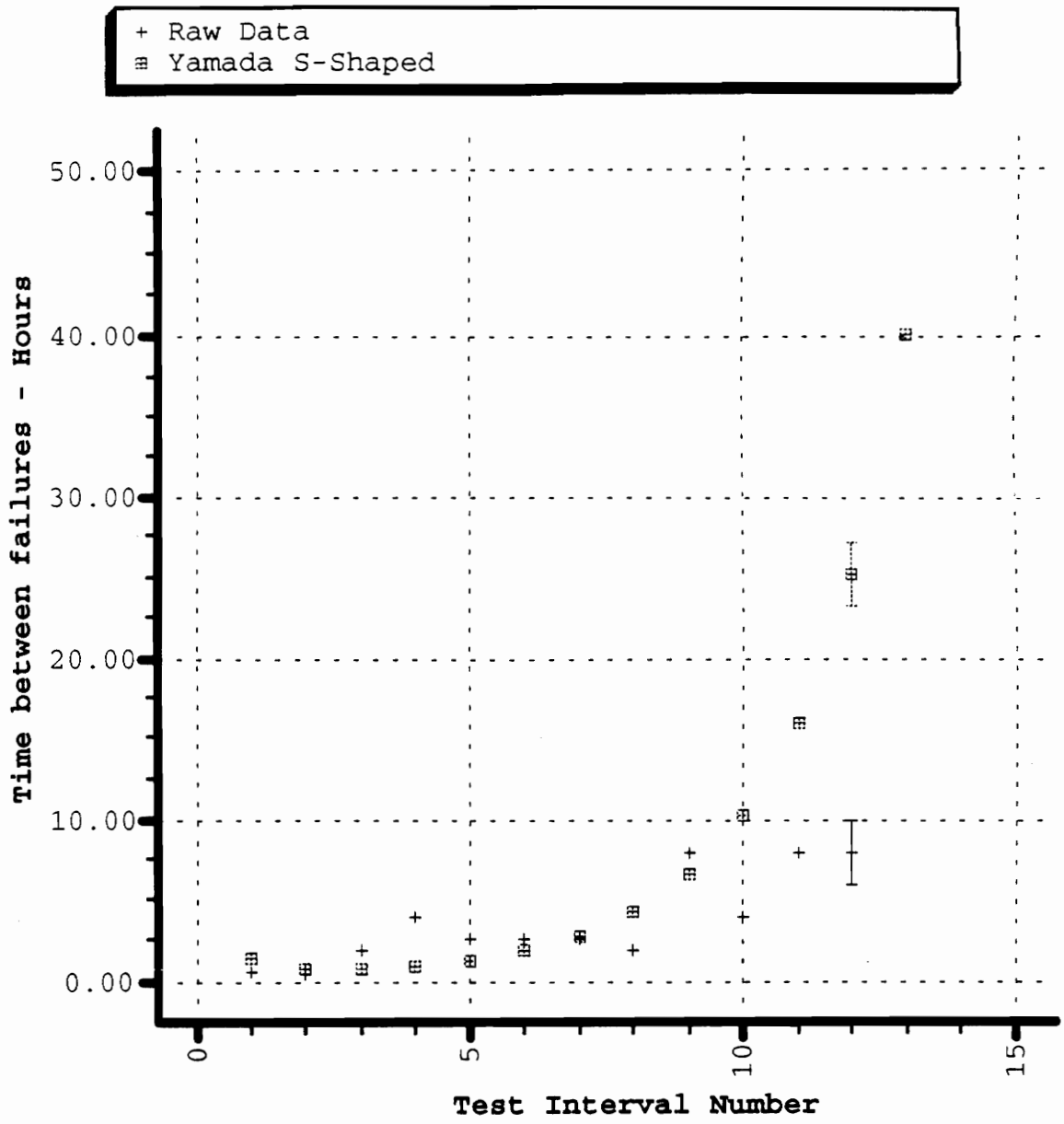


Figure 9. Time Between Failures: Yamada S-Shaped Reliability Growth Model

4.5 Evaluation of Models Using Model Dependent Comparison Criteria

The evaluation of the accuracy characteristic of predictive validity will be aided by the use of Figure 10 for each of the execution time models. Likewise, the evaluation of the quality of fit characteristic of predictive validity will be aided by the use of Figure 11 and Figure 12. Additionally, the S-Shaped Reliability Growth model will be evaluated using the Chi-square method for the predictive validity characteristic of quality of fit. The SMERFs software was used to produce the two figures and the various statistics associated with the predictive validity criteria for all models.

When evaluating the accuracy from Figure 10, one is looking for the best model in determining future values for time to next failure by using the past failures and the negative of the prequential likelihood function. The simplest way to interpret the accuracy values is to know that the larger the values produced, the better the model performs. More accurately, the larger numbers produce a stronger likelihood that future data points will come from the model's probability density function.

When evaluating the quality of fit from Figure 11, a determination needs to be made as to the degree the model is over estimating or under estimating based upon its position to the line $y = x$. If a model is uniformly distributed over and around the line, then its quality of fit characteristic will be scored high.

To further evaluate quality of fit and accuracy, Figure 12 is a graph of the model bias trend for each of the execution time models. Using the model bias trend analysis, the

fluctuations of estimations (either over or under estimating) is eliminated. Rather, the behavior of the model's output is evaluated over time. The closer the plot for each model is to the charted line, y = x, indicates a lesser degree of model bias. Additionally, a Kolmogorov-Smirnov (K-S) score is calculated for each model. Lower values for the K-S score indicate the model is more capable of adapting to behavior changes in the data. Therefore, the lower the K-S score, the more likely the model will produce more accurate estimations.

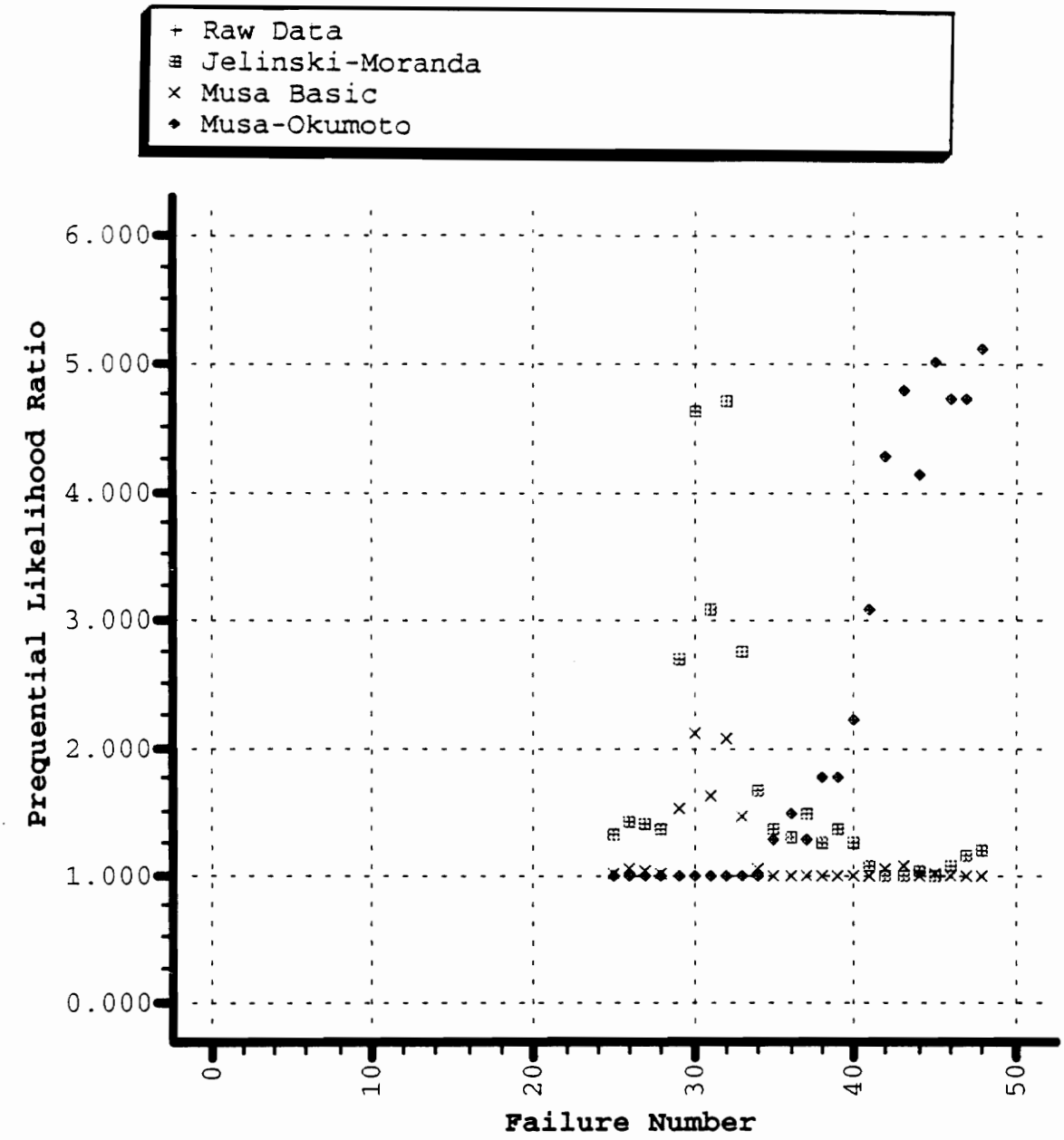


Figure 10. Relative Accuracy Graph

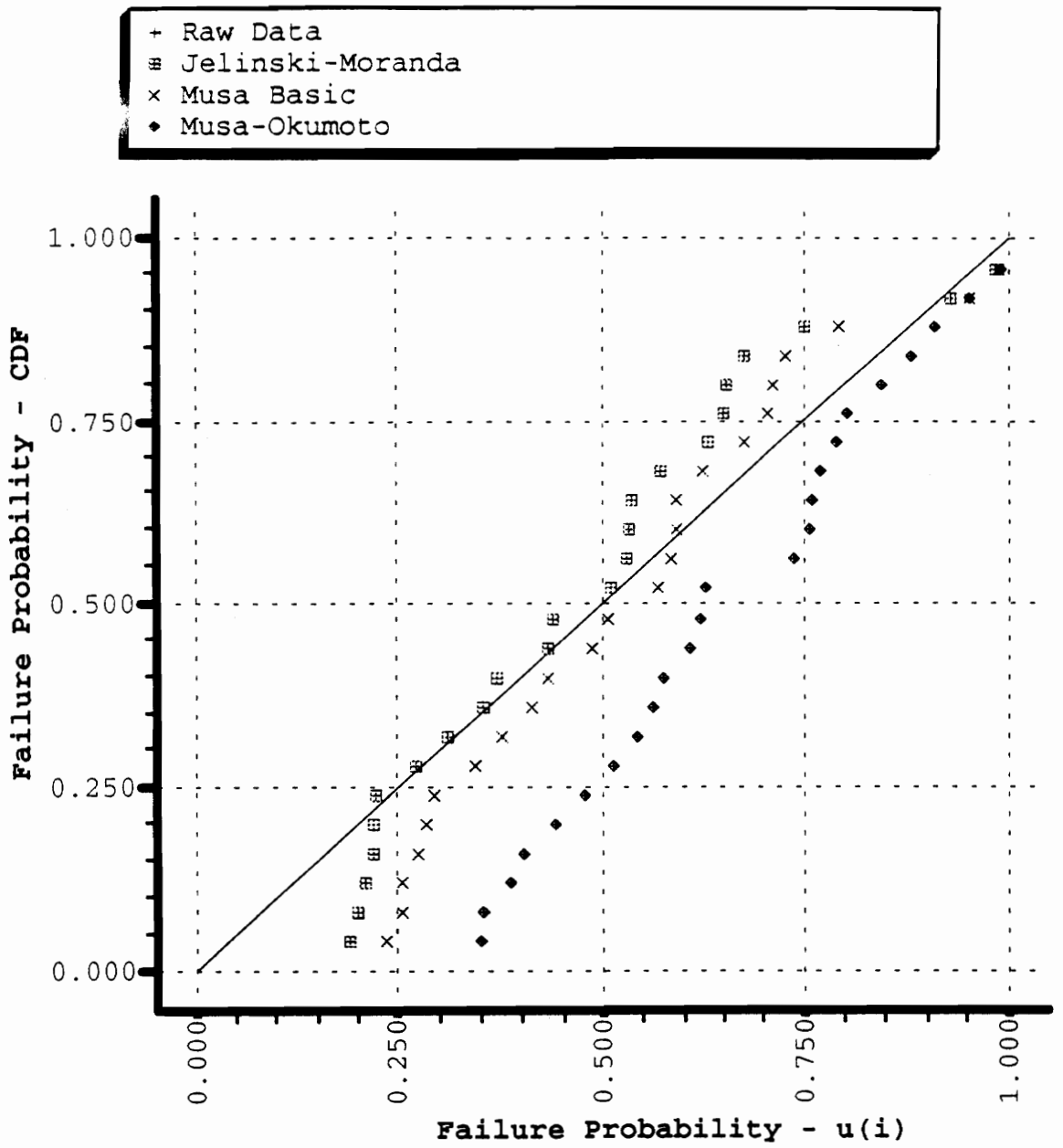


Figure 11. Quality of Fit Graph

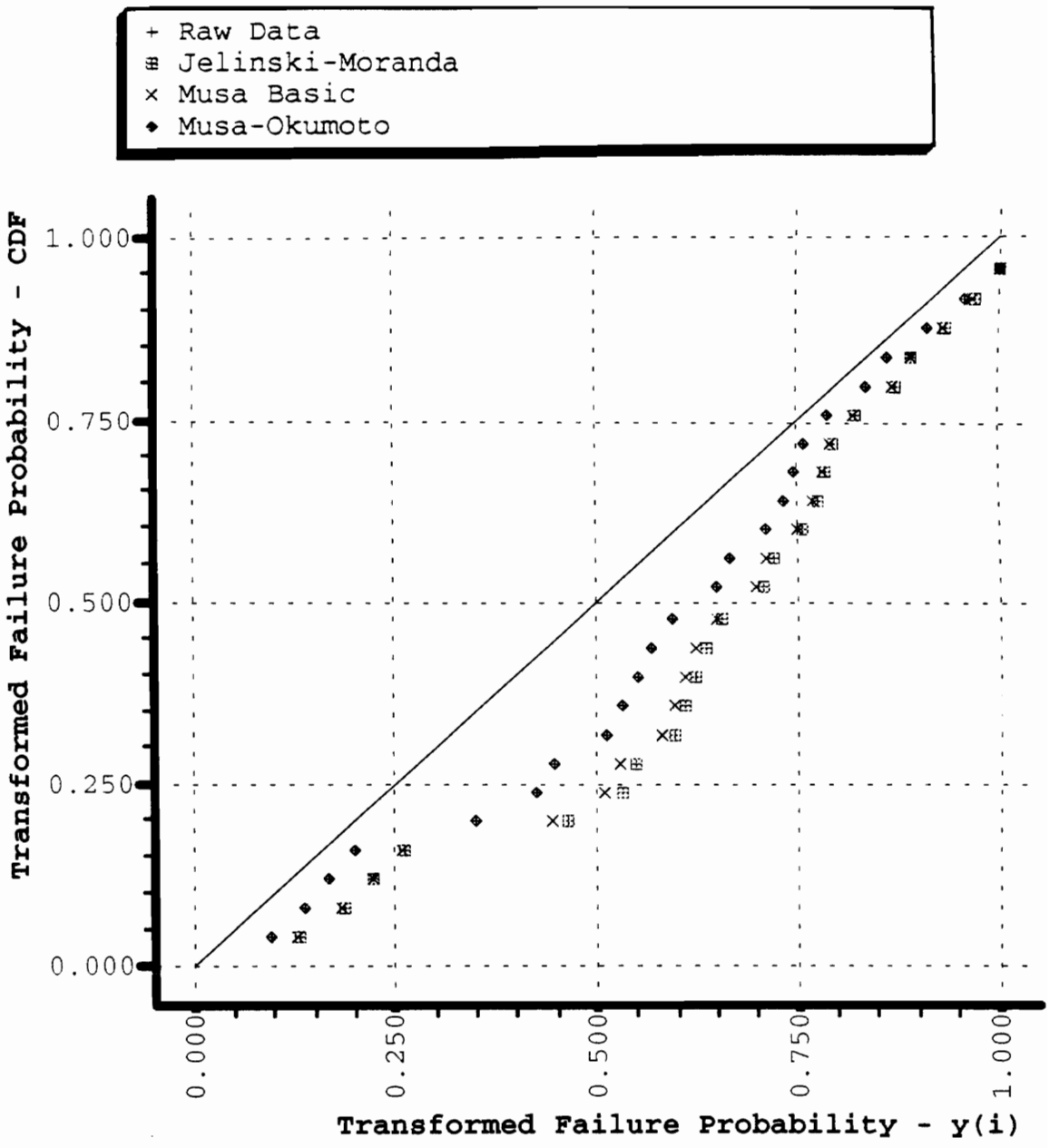


Figure 12. Model Bias Trend

Musa's Basic Execution Time Model

Replicative Validity

Figure 8 shows the curve for this model using the "x" marking. When tracing this curve, a very good estimation is made to reproduce known failure data over the vast majority of failures. The only notable exception occurs with the last three failures, such that the model produces a slightly high estimate.

Predictive Validity

Figure 10 shows that the model produces lower values than the others, its accuracy rating is a 1.000. This indicates that the model's future data points are less likely to be the result of the model's probability density function. Additionally, the K-S score for the model is 0.303. In comparison with the other two models, this model is relatively high when compared to the Logarithmic Poisson model and just slightly lower than the Jelinski-Moranda model. Therefore the accuracy characteristic of predictive validity for the model is not exceedingly high.

Figure 11 shows that the model performs quite well with respect to quality of fit. The model slightly under predicts for approximately half of the overall failure probability and then shifts to slightly over prediction, and then (upon close examination) back to slight under prediction. Figure 12 shows that the model performs closely with the Jelinski-Moranda model in terms of bias and has more bias than the Logarithmic Poisson model.

In each case, however, the model performs fairly well and therefore, the model has a relatively high quality of fit capability.

Capability

The Basic Execution Time model, as per the model's title, uses execution time to perform its modeling calculations. The model then uses calendar time components to convert the derived quantities into calendar time. This ability is of great use to those using the resultant data. The model readily determines the current software system reliability in terms of mean time to failure and failure intensity.

The model also provides for determining the amount of execution time required to reach a target failure intensity. This information can then be manually converted into the number of days (testing and debugging) required to reach the target.

Since the determination of resource and cost requirements are directly related to system characteristics (size, complexity, etc.) in the early stages of system development, the model must have parameters that can be related to these characteristics. Musa's model does have such a relation in several parameters.

Quality of Assumptions

Upon reviewing the model assumptions in section 3.2, there is a high level of quality in the assumptions. It is very difficult to insure a perfect debugging process, so the ability to model an imperfect debugging process is important and will better reflect reality. The assumption of an exponential distribution of failures also represents a more likely

view of reality. That is, a larger number of failures generally occur in the initial phases of testing. Once those failures are corrected, the rate of occurrence tends to follow a pattern of exponential decline. The only questionable assumption is the independent relation between failures. Experience has shown that failures within a software system can and will cause other failures to occur.

Applicability

The ability to model an imperfect debugging process, as mentioned above, is quite applicable to the ATWCS environment. Although perfection is strived for in initial development, debugging, and maintenance, the reality has proven to be contrary. Also, prior experience on the currently deployed Tomahawk Weapon Control System (TWCS) has shown that the size and complexity of the software system has a high likelihood of changing significantly with the addition of enhancements. The basic execution time model has the ability to reflect those changes. Additionally, the model is very capable in its ability to utilize a failure severity rating scheme in its calculations.

Simplicity

The basic execution time model is, overall, a fairly simple model with respect to concept, data collection, and programmability. It uses only two major parameters, total number of failures and failure intensity, each of which are appropriately defined. Some estimation is required to obtain the parameter values, such as the inherent faults per line of code and the fault reduction factor.

The Jelinski-Moranda Model

Replicative Validity

Figure 8 shows the curve for this model using the "box" marking. When tracing this curve, a very good estimation is made to reproduce known failure data over the vast majority of failures. It can be noted that this curve closely follows the Basic Execution model's curve. There seems to be an outlier on the last data point. It is extremely high according to the other models.

Predictive Validity

Figure 10 shows that the model produces higher values than that of the Basic Execution Time model, but slightly lower than the Logarithmic Poisson model. The accuracy rating for this model is 1.202. This indicates that the model's future data points are more likely to be the result of the model's probability density function. Additionally, the K-S score for this model is 0.324, the highest of the three execution time models. Due the having the highest degree of bias, the overall rating must decline. Therefore the accuracy characteristic of predictive validity for the model is scored at the moderate level.

Figure 11 shows that the model performs quite well with respect to quality of fit. The model mirrors the Basic Execution Time model in its behavior, that is, slightly better on the lower half of the failure probability but slightly worse on the middle quarter of the scale. This model continues to follow the Basic Execution Time model at the high end of the scale by crossing over the line to the under predicting side. Figure 12 shows that the

model performs closely with the Basic Execution Time model in terms of bias and has more overall bias than the other models. With a better than average overall performance, the model produces a moderately high score for quality of fit.

Capability

The Jelinski-Moranda model also uses execution time to perform its modeling calculations. The model has no problem calculating the current software system reliability in terms of mean time to failure and failure intensity. As with Musa's basic model, this model also provides for determining the amount of execution time required to reach a target failure intensity. Resource and cost information are more difficult to obtain through the use of this model, but certainly possible. The estimation of the initial number of faults and the constant of proportionality can be somewhat tedious during the early phases of system development.

Quality of Assumptions

Upon reviewing the model assumptions in section 3.2, the level of quality in the assumptions is moderate. The assumption that debugging is perfect and each fault is corrected at the time of occurrence is very questionable. Effective testing practices dictate that alternative test cases be executed when a given failure is severe enough to not allow testing to proceed. The reality is that the failure is reported and efforts are initiated towards its correction. The unknown factor is how long it will take to develop a solution to the problem. Additionally, it is generally not cost effective to produce another version

of the software containing the fix for just one failure. Work arounds are often developed when possible to allow testing to continue. Coupled with the assumption that the faults are completely independent, the overall quality of assumptions is reduced.

Applicability

Although widely used, the Jelinski-Moranda model is not a good match when considered for use in the ATWCS environment. The reliance on a perfect and immediate debugging process causes a significant departure in the model to reality interface. Both development and operational environments do not support this assumption. Of particular hindrance to this assumption is the operational environment. When failures are discovered in the fleet, minimally it may be a matter of weeks before the necessary corrective actions can be taken and a new version of software prepared. Then, a logistical problem of delivering the software to the user creates another lengthy delay. The model does however, adequately address the need to use a severity rating scheme for the input failure data.

Simplicity

The Jelinski-Moranda model is an uncomplicated model with respect to concept and data collection, but slightly more involved with respect to programmability. The methods used to estimate its parameters require functions such as the Maximum Likelihood and Least Squares to be implemented as well.

The Logarithmic Poisson Execution Time Model

Replicative Validity

Figure 8 shows the curve for this model using the "triangle" marking and denotes the model with the name of its creators, Musa-Okumoto. When tracing this curve, another good estimation is made to reproduce known failure data over the known failure data. The model seems to find the middle ground between the actual data points. Although the last few data points are on the lower side of reality, the difference is small. It would appear that the model takes a pessimistic view toward the end of its domain.

Predictive Validity

Figure 10 shows that the model produces consistently higher values than the others, its accuracy rating of 5.125 is significantly higher. This indicates that the model's future data points are highly likely to be the result of the model's probability density function. Additionally, the K-S score for the model is 0.222. In comparison with the other two models, this model scores the lowest and therefore the best in terms of the K-S rating. Therefore the accuracy characteristic of predictive validity for the model performs very well.

Figure 11 shows that the model has a tendency to consistently provide results that are under estimations. As time passes, it appears that the model performs much better and much closer to the desired behavior. In fact, in the last quarter of the failure probability, the model performs the best. Figure 12 supports the previous figure in terms of quality of

fit. In this case, however, it performs the best when bias is accounted for in the graph. Overall, the model has a moderately high quality of fit capability.

Capability

The Logarithmic Poisson Execution Time model, as per the model's title, uses execution time to perform its modeling calculations. In a similar fashion to Musa's Basic model, this model uses calendar time components to convert the derived quantities into calendar time. This ability is of great use to those using the resultant data. The model readily determines the current software system reliability in terms of mean time to failure and failure intensity.

The model also provides for determining the amount of execution time required to reach a target failure intensity. This information can then be manually converted into the number of days required to reach the target reliability.

Resource and cost requirement information is not readily producable for this model, although possible. The model is related to the basic execution model in such a way that the underlying parameters can be related to early phases of development.

Quality of Assumptions

Upon reviewing the model assumptions in section 3.2, there is a relatively high level of quality in the assumptions. Of most interest is the assumption that there are an infinite number of failures in a given system. On the outset, this may appear to be a ridiculous notion, however, upon closer examination, the assumption tends to gain

validity. Underlying the infinite failure assumption is the belief that the effectiveness of repair declines as time passes. This can occur for several reasons, failures corrected are only partially corrected, the level of experience of the persons performing the work has a tendency to decrease after the software system enters the system test phase, and the difficulty level associated with finding the failure cause drastically increases. From experience previously referred to on the currently deployed TWCS, the number of software trouble reports (STRs) has generally increased over time, regardless of the number corrected. Many of the factors listed above have contributed to this phenomena.

Applicability

The logarithmic poisson execution time model has some unique features that create a good match to the ATWCS. The model realizes the fact that the vast majority of early failures will be discovered in a set of highly utilized software components while faults inherent in the less utilized components are less likely to be discovered until a much later time. Therefore, the model is best utilized for systems which have highly non-uniform operational profiles. That is, a particular set of software components are executed more often that other components. The ATWCS definitely has such a situation. The engagement planning, track database management, mission data, and launch are utilized with significant frequency while other functions such as training, context sensitive help, on-line documentation, and copy to tape utilities are utilized with less frequency. The failure data associated with the ATWCS has a severity rating associated with it. This model is very capable in its ability to utilize a failure severity rating scheme in its

calculations. Additionally, the infinite number of failures assumption, as supported above, is likely to be applicable.

Simplicity

The logarithmic poisson execution time model is, overall, a fairly simple model with respect to concept and data collection while slightly more involved with respect to programmability. This model also uses only two major parameters, the failure decay parameter and the failure intensity. As with the Jelinski-Moranda model, estimation functions must be programmed for the estimation of the failure intensity.

The S-Shaped Reliability Growth Model

Replicative Validity

Figure 9 shows the curve for this model using the "box" marking. Each test interval represents an eight hour period. This interval reflects the actual testing environment of the ATWCS. When tracing this curve, the model produces significantly higher values for time between failure for the last three test intervals. The curve does however, fit fairly well up to and including the tenth interval.

Predictive Validity

The evaluation of predictive validity for the interval based models are slightly different. The Chi-square function will be used to evaluate the quality of fit attribute. The accuracy evaluation is equivalent to the execution time models, however, the SMERFs software is unable to graph them together.

As it turns out, the graph of the S-Shaped Reliability Growth model for Relative Accuracy (similar to Figure 10) is uneventful. All of the data points fall on the 1.0 level for the prequential likelihood ratio, creating a horizontal line. This behavior reflects a very low probability that future data points will be produced by the model's probability density function. This model is can be considered equivalent to the Basic Execution Time model for this characteristic.

The SMERFs software produced a value of 0.212 for the chi-square function when it is related to the input data. Using a table of percentage points for the chi-square function, a value of 0.412 was obtained. When the chi-square value is larger than the percentage point value, the quality of fit is considered not adequate. In this case, the chi-square value is approximately 45 percent lower than the percentage point value. Thus providing a fairly good indication of a high level of quality of fit.

Capability

The S-Shaped Reliability Growth model is the only model presented in this paper that uses interval based data for its reliability calculations. The interval data is however, in execution time rather than calendar time. The model produces the software system reliability measures in a variety of forms which include mean time to failure and failure intensity. Through manipulation of the model output, the amount of execution time required to reach a target failure intensity or when the i^{th} error will occur can be determined.

The S-Shaped Reliability Growth model is based on the Goel-Okumoto nonhomogeneous Poisson process model which is known to have the capability of relating model parameters to the characteristics of the software development process. The modification to the Goel-Okumoto model does not eliminate this relation and therefore the S-Shaped Reliability Growth model has the ability to provide resource and cost information.

Quality of Assumptions

Upon reviewing the model assumptions in section 3.2, there is a relatively moderate level of quality in the assumptions. The assumptions that failures are corrected immediately and perfectly and that failures are independent of each other have been thoroughly discussed for previous models. Each of the other model assumptions is appropriate and are not disputed.

Applicability

The S-Shaped Reliability Growth model's use of the mean-value function to model the learning process involved when testing within new environments provides for a natural match to the ATWCS. Since the ATWCS test environment and physical hardware are new to the testing groups, this model is quite applicable. To the extent of my knowledge, the S-shaped model is the only model that recognizes this issue. The model also incorporates the ability to use severity ratings that are associated with the failure data from the ATWCS. The only problematic issues concerning the applicability of this model are the two assumptions that are contrary to the realities of the ATWCS.

Simplicity

The data collection activities for the model require some slight modifications to provide the data in the proper form. No significant increase in complexity is added. Concerning the model concepts, the S-shaped model is relatively simple and likewise can be readily programmed.

Chapter 5

Identification of Desirable Software Reliability Models

5.1 Scoring Each Comparison Criteria.

A scoring system of 1.0 to 5.0 was used to score the models. The scale is defined as follows: 1.0 is deficient, 2.0 is somewhat deficient, 3.0 is moderate performance, 4.0 is good, and 5.0 is very good. The evaluation of each model that was performed in Chapter 4 was used in the determination of the scores. The Table 5 is a summary of the scores given to each model for each comparison criteria.

Table 5. Scoring Summary of Reliability Models

Comparison Criteria / Models	Replicative Validity	Accuracy (Predictive Validity)	Quality of Fit (Predictive Validity)	Capability	Quality of Assumptions	Applicability	Simplicity
Musa's Basic Execution Time Model	4.0	3.0	4.0	5.0	4.0	5.0	4.0
Jelinski-Moranda Model	3.0	3.0	4.0	4.0	3.0	2.0	4.0
Logarithmic Poisson Execution Time Model	4.0	5.0	4.0	5.0	4.0	5.0	4.0
S-Shaped Reliability Growth Model	3.0	3.0	4.0	5.0	3.0	3.0	4.0

5.2 Selection of the Most Desirable Model(s)

The weighting factors developed in Chapter 2 are used in the compilation of Table 6. Table 6 portrays the weighted scores for each of the comparison criteria. The total weighted sum is then given for each model.

Table 6. Weighted Scoring Summary of Reliability Models

Comparison Criteria / Models	Replicative Validity 0.80	Accuracy (Predictive Validity) 1.00	Quality of Fit (Predictive Validity) 1.00	Capability 0.50	Quality of Assumptions 0.75	Applicability 0.90	Simplicity 0.35	Total Weighted Sum
Musa's Basic Execution Time Model	3.20	3.00	4.00	2.50	3.00	4.50	1.40	21.60
Jelinski-Moranda Model	2.40	3.00	4.00	2.00	2.25	1.80	1.40	16.85
Logarithmic Poisson Execution Time Model	3.20	5.00	4.00	2.50	3.00	4.50	1.40	23.60
S-Shaped Reliability Growth Model	2.40	3.00	4.00	2.50	2.25	2.70	1.40	18.25

The Logarithmic Poisson Execution Time model has produced the highest score, indicating that it is most likely the best model of the four that were evaluated to be utilized in the reliability measurement of the target system (ATWCS). The term "most likely" is intentionally used due to the fact that 100 percent certainty in the assertion is improbable.

Musa's Basic Execution Time model obtained the second highest score, only two points lower than the Logarithmic Poisson Execution Time model. With such a relative closeness in rating, sincere consideration should be given to reevaluating the comparison criteria for these two models. However, the purpose of this project has been served. That is, the selection of the most appropriate reliability measurement method given information surrounding the target system and its environment.

Chapter 6

Recommendations for Future Research

This project is based upon a system developed under the DOD Standard 2167A Software Development Process. In late 1994, MIL-STD-498 was signed and is now in effect. An evaluation of software intensive systems that are developed under the latest standard would provide an interesting contrast to the system evaluated in this project.

Additionally, the reliability models and comparison criteria defined and used for this project should be applied in a similar fashion to an extensive assortment of software intensive systems. Doing so would provide invaluable data on the models and the criteria that is not available anywhere to date.

Research that should be considered specifically associated with the software reliability of the ATWCS is the development of a reliability block diagram of the nine software components. A fact to consider is the relative importance of each particular CSCI to the performance of the system as a whole. The reason that this has not been done for the ATWCS is mostly politics and partly because the initial systems engineering effort neglected this aspect. The contractors and government agency developing the software had very strong feelings concerning an effort to "back-fit" a reliability assignment to their particular software component.

As previously mentioned, many of the comparison criteria require an expert opinion in their evaluation. Efforts to lessen the necessity of expert opinion in the evaluation and increase the ability to have quantifiable measures to evaluate the criteria

should be investigated. A possible method that would address this issue is the development of checklists to support the evaluation of each comparison criteria. The checklists would significantly reduce possible scoring inconsistencies and evaluator bias. In conjunction with the checklists, efforts should be expended to refine the comparison criteria to a consistent level and to modify the weighting method such that the sum of the weighting factors equal one.

Additional study could also be applied to the appropriate size of the data set used to evaluate the software reliability models. When is a certain amount of data too little, when is there enough data, and is it possible to use too large of a data set?

Chapter 7

Concluding Remarks

This report addresses the problem of selecting the right software reliability model from the numerous models available and provides an evaluation of reliability models that are in use today with respect to a specific environment (ATWCS).

Comparison criteria to be applied to each model are developed, defined, and weighted in Chapter 2. Each criteria is intended to address a particular characteristic of reliability models. The evaluation of the comparison criteria against each reliability model is performed in Chapter 4. The evaluation is assisted by the use of the SMERFS software package.

Chapter 3 provides an overview of software reliability models. The predominant categories for models is discussed to provide a sense of organization and to touch upon the variety of models available, including references to models not covered by this project. Four reliability models are described to form a foundation for evaluating the models.

A method for determining the most applicable reliability model is presented in Chapter 5. The weighting of comparison criteria according to the system environment is a crucial task prior to the selection of the most desirable reliability model. The weighting of the criteria is discussed in Chapter 2. For each individual application of the process presented in this project, the weighting of the comparison criteria should have consensus among those in the organization in which the system is developed. Techniques such as the

Nominal Group Technique or the Delphi Technique are helpful methods to use in reaching a consensus. For a more in-depth analysis in the future, the use of these techniques is recommended.

A significant portion of literature that addresses the issue of software reliability and the use of reliability models takes a narrow view of the topic. This report takes a systems view of the use of reliability models and constantly strives to consider the system environment.

In several references that presented mathematical formulations of a variety of software reliability models, the author's concluding remarks concerning the use of the software reliability models directly supported the intent of this report. For example, Villemeur (1992) states "As it is difficult to choose the best model...the analyst must endeavor to find the most appropriate model for his data set."

This report has presented a method by which an analyst can find the most appropriate model. The process is long and tedious which causes a concern for the willingness of industry expend the resources required. Experience in the government sector has shown that reliability measurement is fully supported only when it is politically important. A sincere commitment must be made by government and industry in the area of software reliability. I believe that it is starting to happen, but slowly.

After reflecting on the project and the process by which it was developed, a few items would be done differently and or expanded in the future. First, the available literature on existing software reliability models and modeling is scarce. A more in-depth

search for applicable literature would be conducted and a possible relationship with the Reliability Analysis Center in Rome, New York would be pursued. There appears to be a wealth of reliability knowledge within the center that would be highly useful in the academic and work environments. However, information published by the center is costly for a student on a budget. Secondly, a wider range of models would be included in the evaluation regardless of their use in industry. Lastly, as a result of the evaluation, an extension of the project would be the development of a customized software reliability model specifically for the target system and then comparing it to the other models in the study.

APPENDIX A

Reliability data used for the project

Data used for all Execution Time models:

Failure Number	Hours Since Last Failure	Severity
1	.00660000E+01	1
2	.06800000E+01	2
3	.05000000E+01	4
4	.08300000E+01	4
5	.02500000E+01	4
6	.07500000E+01	1
7	.05000000E+01	4
8	.05500000E+01	4
9	.04100000E+01	4
10	.08300000E+01	4
11	.12500000E+01	1
12	.15000000E+01	4
13	.05500000E+01	4
14	.06160000E+01	4
15	.02900000E+01	4
16	.01160000E+01	1
17	.01160000E+01	4
18	.05000000E+01	4
19	.02500000E+01	4
20	.09200000E+01	4
21	.01160000E+01	4
22	.02500000E+01	4
23	.02500000E+01	2
24	.20000000E+01	4
25	.15000000E+01	4
26	.22000000E+01	4
27	.13300000E+01	4
28	.10000000E+01	4
29	.10000000E+01	1
30	.55000000E+01	4
31	.42200000E+01	4
32	.15000000E+01	4
33	.46600000E+01	1
34	.16600000E+01	4
35	.15000000E+01	4
36	.13300000E+01	4
37	.22200000E+01	4
38	.51670000E+01	4
39	.15000000E+01	4
40	.45000000E+01	4
41	.26700000E+01	4
42	.15000000E+01	4
43	.15000000E+01	4
44	.32500000E+01	4
45	.60000000E+01	4
46	.33300000E+01	4
47	.66000000E+01	2
48	.69100000E+01	2

Data used for Interval Data model:

Test Interval Number	Number of Failures	Hours in Interval	Severity
1	3.000000e+000	8.000000E+000	1
2	1.000000e+000	8.000000E+000	2
3	7.000000e+000	8.000000E+000	4
4	1.000000e+000	8.000000E+000	1
5	1.000000e+000	8.000000E+000	2
6	1.200000e+001	8.000000E+000	4
7	1.000000e+000	8.000000E+000	1
8	3.000000e+000	8.000000E+000	4
9	2.000000e+000	8.000000E+000	4
10	1.000000e+000	8.000000E+000	4
11	2.000000e+000	8.000000E+000	4
12	3.000000e+000	8.000000E+000	4
13	3.000000e+000	8.000000E+000	4
14	4.000000e+000	8.000000E+000	4
15	1.000000e+000	8.000000E+000	4
16	2.000000e+000	8.000000E+000	4
17	1.000000e+000	8.000000E+000	2

REFERENCES

Aggarwal, K.K. Reliability Engineering, Dordrecht, The Netherlands: Kluwer Academic Publishers. 1993

Blanchard, Benjamin S.; Verma, Dinesh. "Current Practices and Trends," First Annual Workshop on Engineering of Systems in the 21st Century: Facing the Challenge. Fredericksburg, Virginia. June 1994.

Blanchard, Benjamin S.; Wolter J. Fabrycky. Systems Engineering and Analysis, Second Edition. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1990.

Lloyd, David K.; Myron Lipow; Reliability: Management, Methods, and Mathematics, Second Edition. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1977.

Musa, John D; Anthony Iannino; Kazuhina Okumoto. Software Reliability: Measurement, Prediction, Application. New York: McGraw-Hill, 1987.

Musa, John D; Kazuhina Okumoto. "Application of Basic and Logarithmic Poisson Execution Time Models in Software Reliability Measurement," Software System Design Methods. NATO ASI Series, Volume 22. New York: Springer-Verlag, 1986

Sage, Andrew P., James D. Palmer; Software Systems Engineering, New York, New York. John Wiley & Sons, 1990.

Shere, Kenneth D. Software Engineering and Management, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1988.

SMERFS, a PC-based software package that automates several statistical software reliability models, Dr. W.H. Farr NSWC, Dahlgren, VA. September, 1993.

Triantis, Konstantinos P. "Course Notes: Management of Quality and Reliability," Virginia Polytechnic Institute and State University, Fall 1995.

Troy, Robert; Moawad, Ramadan; "Assessment of Software Reliability Models," IEEE Transactions on Software Engineering, Vol. SE-11, No. 9, September 1985.

Villemeur, Alain. Reliability, Availability, Maintainability and Safety Assessment, Chichester, England. John Wiley & Sons, 1992.

Yamada, S.; Ohba, M.; and Osaki, S., "S-Shaped Reliability Growth Modeling for Software Error Detection," IEEE Transactions on Reliability, Vol. R-32, No. 5, December 1983

Reliability, Maintainability, and Supportability Guidebook, Warrendale, PA: The Society of Automotive Engineers, Inc. 1992.