NPS-SM-97-002

NAVAL POSTGRADUATE SCHOOL
Monterey, California

MCTSSA Software Reliability Handbook

Volume I

Software Reliability Engineering Process and Modeling for a Single Process

by

Norman F. Schneidewind
Julie Heineman

10 January 1996

19970918 138

Approved for public release; distribution is unlimited.

DTIC QUALITY INSPECTED 3

Prepared for: U.S. Marine Corps
Tactical Systems Support Activity
Camp Pendleton, CA 92244-5171

NAVAL POSTGRADUATE SCHOOL
Monterey, California

RADM M.J. Evans
Superintendent

Richard Elster
Provost

This report was prepared for and funded by the U.S. Marine Corps, Systems Support Activity, Camp Pendleton, CA 92255-5171.

Reproduction of all or part of this report is authorized.

This report was prepared by:

Norman F. Schneidewind

Norman F. Schneidewind
Department of Systems Management

Reviewed by:

Reuben T Harris

Reuben T. Harris, Chairman
Systems Management Department

Released by:

[illegible]

D.W.Netzer, Associate Provost and
Dean of Research

REPORT DOCUMENTATION PAGE	Form Approved OMB No 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE 10 January 1996	3. REPORT TYPE AND DATES COVERED Technical Report

4. TITLE AND SUBTITLE MCTSSA Software Reliability Handbook Volume I Software Reliability Engineering Process and Modeling for a Single Process	5. FUNDING RLACH
6. AUTHOR(S) Dr. Norman F. Schneidewind and LCDR Julie Heineman	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Department of Systems Management Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER NPS-SM-97-002
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Marine Corps Tactical Systems Support Activity Box 555171 Building 31345 Camp Pendleton, CA 92255-5171	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this report are those of the authors and do not reflect the official policy or position of the Department of Defense or the United States Government.

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

13. ABSTRACT (Maximum 200 words.)

The purpose of this handbook is threefold. Specifically, it:
o Serves as a reference guide for implementing standard software reliability practices at Marine Corps Tactical Systems Support Activity and aids in applying the software reliability model
o Serves as a tool for managing the software reliability program
o Serves as a training aid

This handbook consists of four volumes. The content of each of the volumes is as follows:

Volume I:	Software Reliability Engineering Process and Modeling for a Single Function System
Volume II:	Data Collection Demonstration and Software Reliability Modeling for a Multi-Function Distributed System
Volume III:	Integration of Software Metrics with Quality and Reliability
Volume IV:	Schneidewind Software Reliability and Metrics Models Tool List

14. SUBJECT TERMS	15. NUMBER OF PAGES 47
	16. PRICE CODE

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

MCTSSA SOFTWARE RELIABILITY HANDBOOK

VOLUME I

SOFTWARE RELIABILITY ENGINEERING PROCESS and MODELING FOR a SINGLE FUNCTION SYSTEM

10 January 1996

Revised: 15 July 1997

Dr. Norman F. Schneidewind
LCDR Judie A. Heineman

Naval Postgraduate School
Code SM/Ss
Monterey, California 93943

Voice: 408-656-2719
Fax: 408-656-3407

Email: schneidewind@nps.navy.mil

DTIC QUALITY INSPECTED 3

PREFACE

This handbook consists of four volumes the first of which is contained in this document. The content of each of the volumes is as follows:

VOLUME I:	SOFTWARE RELIABILITY ENGINEERING PROCESS and MODELING FOR a SINGLE FUNCTION SYSTEM
VOLUME II:	DATA COLLECTION DEMONSTRATION and SOFTWARE RELIABILITY MODELING FOR a MULTI-FUNCTION DISTRIBUTED SYSTEM
VOLUME III:	INTEGRATION OF SOFTWARE METRICS WITH QUALITY AND RELIABILITY
VOLUME IV:	SCHNEIDEWIND SOFTWARE RELIABILITY AND METRICS MODELS TOOL LIST

Demonstrations of the use of software reliability and metrics tools are included in the handbook. The software reliability tool *Statistical Modeling and Estimation of Reliability Functions for Software*, SMERFS, is a public domain tool available for the cost of reproduction from the Naval Surface Warfare Center, Dahlgren, Virginia. Although a DOS program, it can run under any Windows operating system. *Statgraphics* (version 5.2 for DOS, which can run under Windows and Windows for Workgroups) is one of the few statistical programs which combines statistical procedures with an equation editor and capability of executing user created equations. The latter capability was needed because not all of the reliability equations and none of the metrics equations are available in SMERFS. The author has written a large number of equations in version 5.2 of *Statgraphics* over a period of several years. However, version 5.2 is no longer available for sale and the Windows version has not retained the equation editing and execution capability. Therefore the author is in the process of converting the non-SMERFS equations to a commercially available Windows-based package for future use with this handbook. In addition, complete definitions, descriptions, and examples of all equations are provided in the handbook so that users who may wish to implement the non-SMERFS equations in the package of their choice, have the documentation to do so.

Software failure and metrics data were not available from MCTSSA for use in this handbook. In lieu of this data -- for illustrative purposes -- failure and metrics data from the *Space Shuttle* -- are used. However, defect data from *LOGAIS*, a Marine Corps multi-function distributed system, was available for use in VOLUME II.

Although not essential, it would be helpful for the user of this handbook to have completed a first course in probability theory and statistics or have equivalent experience.

TABLE OF CONTENTS

SECTION 1: IMPLEMENTING AN SRE PROGRAM

A. PURPOSE

The purpose of this handbook is threefold. Specifically, it:

Serves as a reference guide for implementing standard software reliability practices at Marine Corps Tactical Systems Support Activity and aids in applying the software reliability model

Serves as a tool for managing the software reliability program

Serves as a training aid

B. INTRODUCTION

Representing the "intellectual effort" of its authors, software includes not only the source code, but the supporting documentation and test results. With this in mind, software is a complex concept to evaluate and measure. Trying to predict its reliability is just as challenging.

Reliability is seen as the ability of a system to perform as expected under specific conditions for a specified period of time. This also includes the "**probability** that the software will not cause the failure of a system for a specified time under specified conditions." (AIA93) This concept must be matched with appropriate measurement techniques that provide a mechanism to evaluate the software's ability to perform.

Software Reliability Engineering (SRE) is a new discipline that is maturing as more organizations see the need to develop standard reliability practices. The American Institute of Aeronautics and Astronautics (AIAA) defines SRE as "the application of statistical techniques to data collected during system development and operation to specify, predict, estimate, and assess the reliability of software-based systems." (AIA93)

C. DEFINITION OF FAULT MEASUREMENT

As with any intellectual product, errors in design may occur. An error can be defined as "a discrepancy between a computed, observed or measured value or condition and the true, specified or theoretically correct value or condition." (AIA93) In software, these errors may appear while completing requirements formulation or, as is often the case, during design, coding, and testing the product. The software development process should include measures to discover and correct faults resulting from these errors. [In this context, faults are defined as "defects in the code that can be the cause of one or more failures." (AIA93)

These measures can address reviews, audits, screening by language-dependent tools, and several layers of testing. One way to reduce the number and criticality of errors is by modeling the effects of the remaining faults in the delivered product. This can be achieved through a dedicated measurement process by which each defect or fault is noted and formally recorded for inclusion in the reliability model. (AIA93) As a point of clarification, a fault is technically different from a failure. A failure can be defined as "the inability of a system or system component to perform a required function within specified limits" or the "departure of program operation from program requirements." (AIA93) In simpler terms, a fault usually leads to a failure.

D. MANAGERIAL IMPACT OF FAULT MEASUREMENT

Handling, identifying and correcting faults is a significant concern for the manager because the entire software reliability process is expensive. "It also impacts development schedules and system performance (through increased use of computer resources such as memory, CPU time and peripherals requirements)." (AIA93) This addresses the key issue regarding SRE -- *it provides the manager with information about which he can make informed decisions*. There will always be a tradeoff between reliability, frequently referred to as the failure rate, and cost. (Cost is directly related to testing time). The manager will need to decide on a certain level of reliability for the product, resulting in a set cost. Thus, higher reliability will result in a higher cost. The converse is also true.

In general, the failure rate of a software system is seen as a curve with a decreasing slope which results from the identification and removal of errors as time passes. It is the primary purpose of reliability modeling to define the shape of this resulting curve using statistical methodologies. The model used in these reliability assessments can provide prediction information regarding the software execution time needed to discover a specified number of faults, or predict the time period when the next fault will occur. Figure 1 provides a sample software reliability curve that can be generated by using a software reliability model. (AIA93)

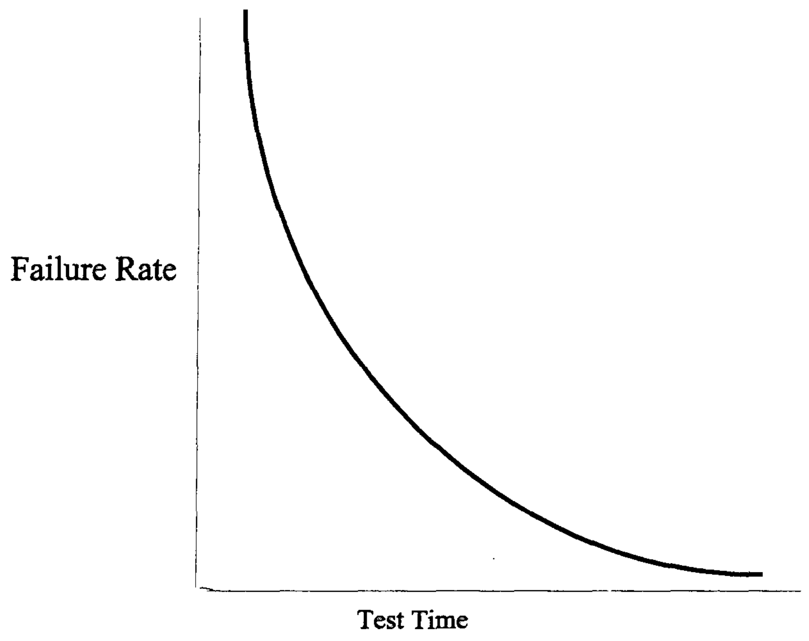


Figure 1: Software Reliability Tradeoff Curve

E. COMPONENTS OF AN SRE PROGRAM

A successful software reliability program does not consist of just a model. It also consists of the support structure: reliability requirements; reliability measurements to meet those requirements; data collection procedures to obtain the necessary data; definition of severity levels of failures; applications of reliability predictions; interpretation of model predictions; and user feedback for model improvements. Although the conceptualization of the model does not occur in a sequence of steps as mentioned above, its *implementation* does. The practitioner can best understand this process from a description of the chronology of implementing and applying the model. Therefore, this approach will be used in explaining the process. To illustrate the process, many equations, figures, and tables will be used. Many real-world examples from the *Space Shuttle* will be used, because the process can be illustrated with real data and real predictions. However, it should not be concluded that the examples are not applicable to MCTSSA; they are. The approach is generic and its feasibility can be tested against MCTSSA systems. The *Shuttle* is a safety critical system where human life and expensive equipment are at risk. This is also the case with MCTSSA systems.

Failure data is preferred to defect data for both empirical reliability assessment and reliability prediction, using a model, because the former is a "departure of program operation from program requirements" observed while the program is *executing*, and includes chronologically ordered *test start time* or *operation start time* and *failure occurrence time,* whereas defect data do not contain this time record. Defect data are used more for administrative control to ensure that defects have been resolved than as data for reliability assessment and prediction. However in some systems , such as the Marine Corps' LOGAIS, only defect data are available. In this case the "reliability predictions" will not be as accurate as when failure data are available, but useful predictions can be made nevertheless. Examples of such predictions for LOGAIS are shown in Volume II of the Handbook.

The existing methodology is based on the *Schneidewind Software Reliability Model* (SCH97, SCH93, SCH75), one of the four models recommended in the *ANSI/AIAA Recommended Practice for Software Reliability.* (AIA93) The validation is based on the fact the model is used to *assist* in assessing the reliability of the *Shuttle* flight software. According to Ted Keller, Manager, Project Coordination, Onboard Shuttle Software Systems, Lockheed-Martin Space Mission Systems & Services: "The *Shuttle* software project is experimenting with a promising algorithm which involves the use of the *Schneidewind Software Reliability Model* to compute a parameter: *fraction of remaining failures* as a function of the archived failure history during testing and operation" (KEL95) Obviously remaining failures, fraction of remaining failures and time to next failure would. not be used to the exclusion of other approaches in making reliability assessments. These metrics would be combined with process procedures such as inspections, defect prevention, project control boards, process assessment, and fault tracking, to provide a quantitative basis for achieving reliability objectives. (BIL94)

The standard practices described under *Implementing a Software Reliability Program* are essentially those recommended in the *ANSI/AIAA Recommended Practice for Software Reliability* (AIA93) and the *ANSI/IEEE Standard for a Software Quality Metrics Methodology.* (IEE93)

F. IMPLEMENTING A SOFTWARE RELIABILITY PROGRAM

Implementing a software reliability program is a two phased process. It consists of (1) identifying the reliability goals and (2) testing the software to see how it conforms to the stated objectives. The reliability goals can be ideal or conceptual, e.g., zero defects, but should have some basis in reality. The testing phase is the most complex since it involves the actual collection of raw defect data and using it with the selected model.

With these phases being the stated objective, the following steps should be considered by the organization as it begins to develop a software reliability program. These steps provide a "cookbook" approach to the SRE process and are ordinarily followed sequentially. Each step will be discussed briefly to provide a general understanding of the purpose of each phase. Stages that require numerical calculations and application of specific model parameters will be noted. Discussion of those parameters will be deferred until Section 2.

The SRE steps are:

- o State the Reliability Requirement
- o Establish a Measurement Framework
- o Collect the Data
- o Establish Problem Severity Levels
- o Estimate Model Parameters
- o Select the Optimal Set of Failure Data
- o Identify the Operational Profile
- o Make Reliability Predictions
- o Validate the Model
- o Make Reliability Decisions
- o Use Software Reliability Tools

Step 1: State the Reliability Requirement

In this step, the software manager should describe the condition that must be fulfilled for the software to be considered satisfactory (reliable). This is a managerial decision. An example of such a requirement may be the following statement: "The product will have no software failure that would result in loss of life, loss of mission, or cancellation of mission."

Step 2: Establish a Measurement Framework

One approach the organization could employ would be to take the software from the developer at delivery and run it on its own systems and see how well, or poorly, it performed. However, if the manager adopted this approach and waited until the software was delivered to him and then began testing, many months could possibly be wasted if the software is deemed unreliable. In the ideal world, he would have some indications of the system's reliability before it was delivered to him. Although this is not an ideal world, the manager does have at his disposal some techniques he can use to get a "feel" for how the software will perform once it is delivered. He would do this by establishing a measurement framework or plan using the fault data collected by the developer during the product's design phase.

The organization should consider a comprehensive measurement plan that would include *indirect* measures of quality like problem report counts, size and complexity metrics. Figure 2 captures this idea. In this diagram, *Level 1* shows the most direct measurement (e.g., a *time to next failure*). These are the metrics that can be captured directly by the use of a wall clock and the continuous running of the software. *Level 2* shows an indirect measurement (e.g., *discrepancy report count*) one level removed from the direct measurement. At this level a report is written whenever a discrepancy is observed between the required operation and the actual operation of the software. Most of these reports are derived from static analysis (i.e., inspections), although these reports could record the fact that a failure has occurred; however there would be no data about when tests and operations started and when failures occurred. Hence, it would not be possible to directly predict the time to next failure. Finally, *Level 3* shows an indirect measurement two levels removed from the direct measurement (e.g., *size* and *complexity*). These are the basic attributes of the software itself. How many lines of code were developed? How complicated are the routines in the program? Traditionally, the more complicated the coding, the more likely faults will appear.

The advantage of *Level 1* measurements is that they are the most accurate representations of reliability; their disadvantage is that they cannot be collected until the software is tested. Conversely, the indirect measurements are less accurate as representations of reliability, but they can be collected earlier in the development process. This permits an early indication of the reliability of the software.

In addition to collecting failure data, other metrics can be collected during the software design phase to provide the evaluator with an early indication of software quality. However, the applicability of these metrics will need to be determined through various metric evaluation techniques. This evaluation will indicate whether a relationship exists between the metric and the quality of the software under evaluation. Examples of these metrics include the number of executable statements, comments (non-executable code), paths, cycles, and total lines of code (total non-commented lines of code). A complete discussion of metric evaluation can be found in Volume III of the Handbook and in (SCH92a, WAR94).

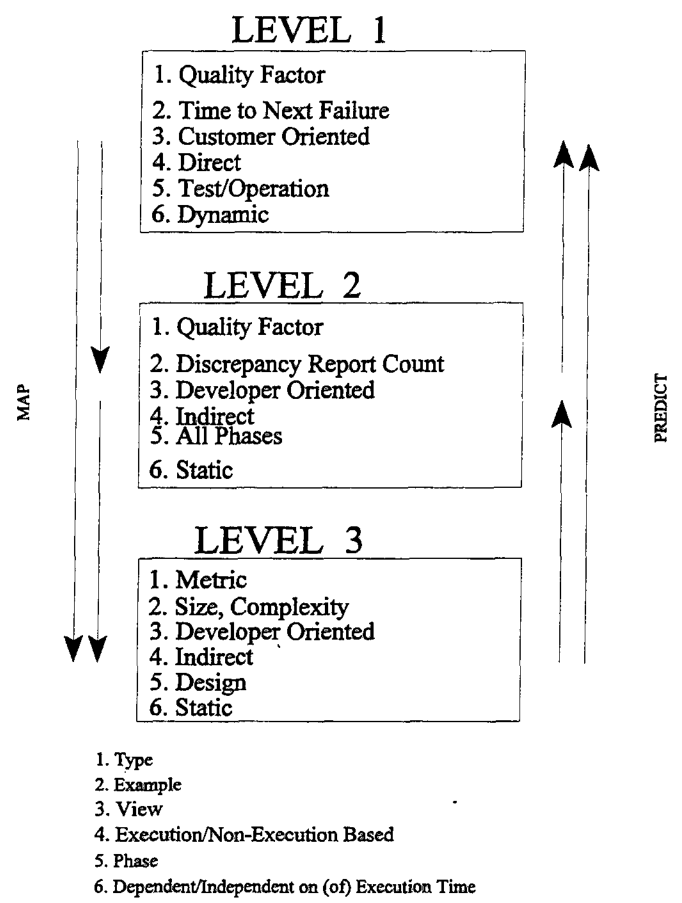


Figure 2: Levels of Measurement

This figure also shows, on the right side, that we want to predict the quality of later phases, using metrics that are available in the early phases. In addition, this figure shows on the left side that we want to map from failures observed in later phases to the metrics of early phases in order to identify the cause of the failures.

Step 3: Collect the Data

Without data, reliability predictions cannot be made. For this data collection, a Data Base Management System (DBMS) would be helpful. For computational purposes, the file management system of certain software reliability tools (e.g. SMERFS and Statgraphics, which are discussed later in the handbook) are usually adequate. However, to manipulate large amounts of failure and metrics data, a specially designed DBMS may be beneficial. This DBMS would allow for data sorting for various analyses and reporting purposes. This is easily accomplished by identifying the key fields of the data (date, time of failure, type of failure, degree of failure) and relating those fields with others. By using the DBMS's query capability, various statistics and reports can be produced by the touch of a few keys. This data can then be properly formatted to be input into the model and further evaluated for trends.

The elements of the database are shown in Table 1.

System ID	Days # (since start of test)	Problem Report ID	Problem Severity	Failure Date	Module with Fault	Description of Problem

Table 1 Failure Data Collection Format

For each system, there should be a brief description of its purpose and functions. The *Days #* field could be noted in *hours* or *minutes*, as appropriate. It is recommended that the *Problem Report ID* field be coded to indicate Software (S) failure, Hardware (H) failure, or People (P) failure.

A more detailed discrepancy report is found in Appendix A. This detailed report could be implemented by the organization as it becomes more familiar with the Software Reliability Process.

Step 4: Establish Problem Severity Levels

The organization will need to establish some consistency in describing the faults it discovers. This will allow better analysis and classification of failures in the analysis and reliability predictions. Some recommended severity level descriptions are as follows:

Level 1. Loss of life, loss of mission, abort mission

Level 2. Degradation in performance

Level 3. Operator annoyance

Level 4. System ok, but documentation in error

Level 5. Error in classifying a problem (i.e., no problem existed in the first place)

Note: Not all problems result in failures.

These levels should be recorded as part of Table 1.

Step 5: Estimate Model Parameters

Once a model has been chosen to be applicable to a particular system, the necessary model parameters must be estimated, using SMERFS. For the purposes of this project, the Schneidewind Software Reliability Model is being used. Three parameters are used in this model and will be used for MCTSSA: α , which is the failure rate at the beginning of the testing interval "s", β , which is the failure rate per failure, and "s," the first interval used in parameter estimation. These parameters are discussed further later in the handbook.

Step 6: Select the Optimal Set of Failure Data

This stage selects the subset of failure data, starting with the beginning interval, "s" through "t," the last observed interval, that will give the best parameter estimates and the most accurate predictions. It relies on the observation that both the software process and product change over time. Therefore old data may no longer be representative of the current and future state of the process and product and , therefore, not as applicable for reliability prediction as the more recent data. This step is discussed in detail later in the handbook.

Step 7: Identify the Operational Profile

The operational profile describes the system's environment. It is usually discussed in terms of modes (single node or multi node operation), frequency of use of a particular station with each station performing a different function (e.g. Workstation 1 performing database functions, Workstation 2 performing word processing functions), and the frequency of function execution (the amount of time the application has been running). It includes the input variables (e.g. a listing of available equipment or a ship's destination), the functional environment of the program (i.e. a specific function the system is to perform such as sorting the available equipment by minor property number), and the output variable (e.g. a printout of the ship's destinations for the next two months). In this framework, a failure can be seen as a departure of the output variable from what it is expected to be. (Musa, 1987). In the *Shuttle* example, it is appropriate to use a single software system (i.e., single node). The applicability of the Schneidewind Software Reliability Model to Marine Corps multi-node systems is discussed in Volume II of the Handbook. A description of the attributes of this environment can be found there.

As part of the operational profile, the organization would be using the obtained failure data and estimating the various parameter inputs to be used in the reliability model.

Step 8: Make Reliability Predictions

This step is the key to predicting the reliability of the software under evaluation. Each of the listed predictions and the applicability to a managerial decision is described in detail in Section 2 of the handbook. The possible predictions resulting from the model application are:

1. Time to Next Failure
2. Cumulative Failures for a Specified Time
3. Remaining Failures and Fraction of Remaining Failures
4. Number of Failures Remaining in One More Test Period
5. Test Time to Achieve Desired Reliability Level
6. Mean Square Error (MSE)
7. Test Time to Achieve Specified Remaining Failures
8. Test Time Needed to Obtain "Fault Free" Software
9. Operational Quality

Step 9: Validate the Model

This step evaluates the model to determine if it actually measures what the model is designed to measure. The predicted values are compared to the actual values to make a determination of the model's validity. As an example, if the model predicts the time to next failure will be two periods, this predicted time would be compared to the actual time. Validation is achieved after certain numbers and types of predictions have been made with a specified accuracy (e.g., average relative error of $\leq$ 20%).

If, however, the values do not compare favorably, the data used in the model should be carefully examined to identify if anything unusual can be found. If the data appears valid, and the model prediction does not match reality, different models would need to be investigated. For the purposes of this handbook, the Schneidewind Reliability Model will be used exclusively.

Step 10: Make Reliability Decisions

The purpose of implementing a reliability program is to provide the manager with additional information through which he can make informed decisions. Reliability decisions such as "Is the software safe enough to not cause loss of life or mission?" can be made as a result of the model's predictions. This particular decision can be applied to the *Shuttle* software. In this handbook, we make the simplifying assumption -- for illustrative purposes -- that the reliability predictions are the only criterion for launching the *Shuttle*, realizing that there are other important considerations, such as the results of inspections, fault tracking, and tests. Here the manager must decide whether to launch the *Shuttle* based on the software reliability predictions . For this example, the predicted remaining failures must be less than a specified critical value and the predicted time to next failure must be at least as long as the mission duration plus some safety margin. This application will be addressed later in the handbook using numerical examples.

For any organization, the predicted software reliability can be key to the managerial decision to accept final delivery of the product. If the software is *predicted* to perform within specifications, the software can be accepted by the organization as fulfilling the contractual obligations. If it is predicted to fall short of the desired goals, further discussion may be needed in addition to further testing and evaluation.

Step 11: Use Software Reliability Tools

There are software reliability tools available to make the model predictions easier to achieve. The *Statistical Modeling and Estimation of Reliability Functions for Software*, SMERFS, is a software package available for this purpose. (Farr, 1993) A sample SMERFS session is outlined in Section 3.D of the Handbook.

G. SUMMARY OF SOFTWARE RELIABILITY IMPLEMENTATION PLAN

In summary, the first phase in the software reliability engineering (SRE) process is to state the organization's reliability goals. These goals can be ideal or conceptual but must have some basis in reality. A goal of "0%" defects might be the ideal objective, but it would not occur in the real world. Imagining for the moment that it could happen, it would cost an extraordinarily large sum of money to obtain. (Recall Figure 1, the Software Reliability Tradeoff Curve).

The second phase of the SRE involves testing. It is here that the failure data is collected and formatted for inclusion in the model of choice. The test plan used must be consistent with the goals established. If a goal is to have a maximum number of remaining failures set at less than one, then the test plan must be able to predict the remaining number of failures in the software. The tests provide insight into the future -- what may occur as a result of using this software. This insight is used to either forge ahead with actual implementation of the software or return to the drawing board and reassess the system. It will provide an indication as to whether or not additional testing is needed because the results to date may be inconclusive or show an undesirable trend. The test results also allow the manager to prioritize his assets. It can help him to decide where he should assign his resources. Is Module C predicted to be more reliable than Module B? If this is true, he may decide to allocate the majority of his resources to Module B to improve its reliability.

Achieving software reliability goals is an iterative process. The organization must continually update its expectations about its software and software reliability. It should not stop with one trial run of the model; it must continue to collect data over long periods of time for each of the systems in use. In light of this, the organization must be constantly looking ahead. As more data is collected over longer periods of testing and operation, this larger data set can be used in a reliability model to make more accurate predictions for longer times into the future. It is an integral part of the SRE process to have the data stored and available in a data repository.

These steps provide the reader with the general overview of the methodology that should be carried out as part of the software reliability engineering (SRE) process. The next section provides amplifying information regarding the data that must be collected, how it is analyzed by the model, and how the results of the model can be interpreted.

SECTION 2: BASIC CONCEPTS USED IN THE SCHNEIDEWIND MODEL

In the previous section, this handbook presented an overview of the SRE process by briefly introducing its key components. This section will further discuss software reliability predictions the Schneidewind Model produces as a result of the data collected by the organization. Applications of the usefulness of these predictions are briefly described. Specifically, this section gives the manager additional information on the mathematical foundations of software reliability engineering. The mechanisms MCTSSA can employ to calculate these predictions can be found in Section 3 .

A. INTRODUCTION

Data collection must be started at the design and developmental phases of the process including any failure data obtained from the developer-run tests. Data obtained from these early stages can then be used during the independent verification and validation phases to predict the software's reliability. However, this data collection would not stop at the development phase; data should be collected throughout field operations. Data obtained at this stage can be used for future software design projects and could lend itself to further model validation.

As discussed in the earlier sections of this handbook, a model is only able to make **predictions** regarding the reliability of the software. These predictions can be used as a management aid for resource allocation and identifying the need for additional testing. Tests evaluate how reliable the software is. They measure how well the software performs compared to the desired performance levels stated by management in the design specifications.

Modeling allows the manager to get a "feel" for how well the software will perform based on actual data. This permits him to "look into the future" and predict how well the software will perform a week from now, a month from now, a year from now. . . The Schneidewind Software Reliability Model addresses the optimal selection of actual test data to be used in making software reliability predictions. The following sections describe the basic concepts used in this model and their implications for management. Numerous examples from the space *Shuttle* will be used because of the abundance of available test data.

Although an abstract discussion of the model may help some individuals understand its applicability, the following scenario is presented to give the practitioner an understanding of the model application and the uses for the application results. Keep this scenario in mind as each of the model components and predictions is discussed.

B. SCENARIO

A manager must decide whether or not to launch the space *Shuttle* for a mission expected to last ten days. He has collected failure data on the software to be used in the launch and has input the data into the model. Based on his confidence in the model, and the predictions made by the model, he will make his decision to launch or not.

C. PREDICTIONS

The following predictions can be made by the Schneidewind Software Reliability Model:

1. Time to Next Failure
2. Cumulative Failures for a Specified Time
3. Remaining Failures and Fraction of Remaining Failures
4. Number of Failures Remaining in One More Test Period
5. Test Time to Achieve Desired Reliability Level
6. Mean Square Error (MSE)
7. Test Time to Achieve Specified Remaining Failures
8. Test Time Needed to Obtain "Fault Free" Software
9. Operational Quality

Each prediction and its managerial applications are discussed in the following sections.

1. TIME TO NEXT FAILURE

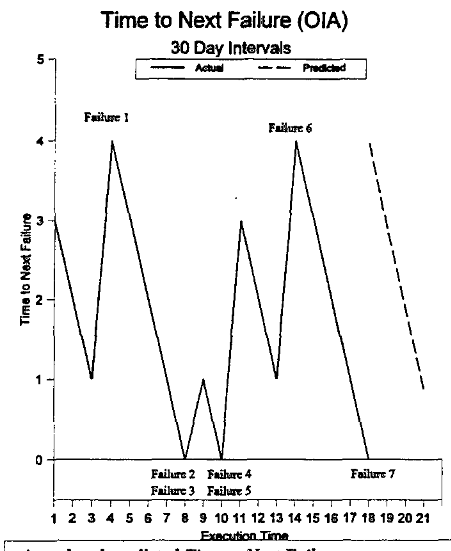
(a) RATIONALE

The following section discusses the significance of time to next failure predictions as it relates to software reliability predictions. This information is important for the manager in that it permits him to make an informed, educated decision on the reliability of the software. As a simplistic example, if the predicted time to next failure is three days, but the software is scheduled to be run for ten days, the manager can anticipate that a failure will occur before the mission is complete. He must then decide whether or not he wants to take that risk.

(b) DEFINITION AND PREDICTION OF TIME TO NEXT FAILURE

The time to next failure can be described as the amount of time that will elapse from the present time, t, until the next recorded failure occurs. In other words, it is the predicted amount of time it will take for the next failure to occur. Execution time is measured from the beginning of a test. This execution time is recorded in convenient intervals of time. As an example, a convenient interval of time for the *Shuttle* program is 30 days. This will be seen on the graphs displaying predictions of time to next failure. However, an organization can set its own interval. In some MCTSSA examples, an appropriate interval would be one week (five workdays).

Figure 3 is a tool that can be used as a management aid. It shows the predicted and actual times to next failure for current execution times. The graph can be read in the following way. If we take a given failure, Failure 1, for example, it occurs at t = 4 (read from the x-axis); therefore, at t = 1, the time to next failure will be equal to 3 (read from the y-axis), (4 - 1 = 3). At t = 2, the time to next failure will be equal to 2, (4 - 2 = 2). At t = 4, Failure 1 occurs, so the time to next failure is 4, (8 - 4 = 4). In this figure, we **predict** the time to next failure to be 4 (at t=18) for Operational Increment A (OIA) on the **dashed curve**, where an Operational Increment is the software system that flies in the *Shuttle*. This curve is derived from additional information and testing (using the Schneidewind Model). Table 2 shows the failure data that was used to construct the **actual** part of Figure 3.



Actual and predicted *Time to Next Failure* are obtained for given *Execution Time*. Failures were observed in intervals 1-18 and predicted in 18-21.

Figure 3: Time to Next Failure

Time Interval	**Failure Identification Number**	**Time to *Next* Failure**
1	--	3
2	--	2
3	--	1
4	1	4
5	--	3
6	--	2
7	--	1
8	2, 3	0
9	--	1
10	4, 5	0
11	--	3
12	--	2
13	--	1
14	6	4
15	--	3
16	--	2
17	--	1
18	7	--

Table 2. Data Used to Construct Time to Next Failure Graph

(c) SCENARIO REVISITED

With the *Shuttle* mission scheduled to last ten days, the ideal situation regarding time to next failure would be to have the next predicted failure occur at a period of time greater than the mission length. In this situation, the next failure should be predicted to occur after the *Shuttle* has safely returned home, i.e., the time to next failure should be greater than ten days. Although this is a simplistic approach, and does not include other factors, it can give the manager some quick information about the reliability of his software. Other predictions should be included in the decision process. These predictions are discussed in the following sections.

2. CUMULATIVE FAILURES FOR A SPECIFIED TIME

(a) DEFINITION AND APPLICATION

Cumulative failures are the total failures predicted to occur at a specific point of time in the future. The benefit of this prediction is that it can be used to anticipate the total failures, for a given execution time, and help the manager prepare to deal with them. Also, if the predicted number of failures is considered unacceptable, the software and its processes can be investigated to see where the problems lie.

3. REMAINING FAILURES, (R), AND FRACTION OF REMAINING FAILURES, (p)

(a) RATIONALE

The *number of remaining failures* provides the manager with valuable information about the reliability of his software. Specifically, it gives him an indication of the software's reliability by predicting the remaining failures (undiscovered failures) that still exist in the software. With this information, he can make an informed decision as to whether the software meets his requirements. If the number of remaining failures is high, the software will typically not satisfy the reliability requirements.

The *fraction of remaining failures* can be used as both a program quality goal in predicting test time requirements and, conversely, as an indicator of program quality as a function of test time expended.

(b) DEFINITION AND PREDICTION OF NUMBER OF REMAINING FAILURES, (R)

The *number of remaining failures* is measured from a given interval and identifies the predicted count of failures remaining in the software. If one predicts the total number of failures that will occur in the software, the remaining failures can be predicted though simple subtraction: total number of failures over the life of the software minus the number of failures found to date. The *fraction of remaining failures, p,* is calculated by taking the number of remaining failures and dividing that number by the total failures predicted for the software.

(c) APPLICATIONS

Management will set guidelines on the desired value for R. Normally, R is set to be less than one. This means that the *expected* number of remaining failures that will occur from the present time to the end of the software execution cycle (also known as run time or "mission time") should be less than one. If the predicted value for R is greater than one, this indicates that the software *could* contain remaining faults and failures that are unacceptable. If the system is mission critical or has the potential to cause harm to human life, the prediction of $R > 1$ should tell the manager that there would be serious risk if he uses the software as it is currently designed.

(d) SCENARIO REVISITED

With the *Shuttle* mission scheduled to last ten days, and with a prediction of time to next failure of four 30 day intervals, coupled with a prediction of **R**<1, the manager would have confidence that the software would operate reliably during the mission. If on the other hand, one or both of these predictions do not meet the thresholds, the manager should seriously consider postponing the launch.

4. NUMBER OF FAILURES REMAINING IN ONE MORE TEST PERIOD

(a) DISCUSSION AND APPLICATION

The number of failures remaining in one more test period gives the manager information about the reliability of the software during that particular time interval. This information can prompt the manager to continue testing or to deploy the software, provided that the time to next failure and predicted number of remaining failures are acceptable. A test period of thirty days of **execution time**, as is done in the *Shuttle* software can be used; or it can be a **calendar time** of one work-week (5 days), as is done in LOGAIS (see Volume II of the Handbook), or any other convenient measure of time.

If the manager must make a decision whether to deploy the software and discontinue testing, he will look for an acceptable value for the predicted number of failures remaining in one more test period. Normally, this number should be significantly less than one. The ideal figure for this prediction would be close to zero, e. g. .0001. If the value is close enough to zero for the manager, he may decide to take the risk, discontinue testing, and deploy the software.

5. TEST TIME TO ACHIEVE DESIRED RELIABILITY LEVEL

(a) DISCUSSION

This information provides the manager with a prediction of the amount of time needed for software testing to achieve a given level of reliability, similar to time needed to obtain "fault free" software. This prediction is based on two key calculations: the fraction of remaining failures, "p," and the predicted total number of failures over the life of the software.

(b) PREDICTIONS

(1) Total Failures

The predicted total number of failures over the life of the software ($t=\infty$) is defined as: $F(\infty)=\alpha/\beta+X_{s-1,}$ where X_{s-1} is the failure count in the range 1,s-1 (i.e., including the first failure count interval and up to and including the interval prior to interval "s").

The benefit of this prediction is that it provides an indication of the total failures and faults that will occur over the life of the software. Thus the software manager can be alerted during test that there could be problems with the software during operation. Also, total failures are used in the prediction of remaining failures.

(2) Remaining Failures and Fraction of Remaining Failures

The predicted number of remaining failures is: $R(t)=(\alpha/\beta)-X_{s,t}=F(\infty)-X_t$, where $X_{s,t}$ is the observed failure count in the range s,t and X_t is the observed failure count in the range 1,t, where "t" is the last observed failure count interval. As already mentioned, the benefit of this prediction is that it may indicate residual or remaining problems with the software. Furthermore, *fraction of remaining failures,* ($p=R(t)/F(\infty)$), can be used as both a *program quality* goal in predicting *test time requirements* and, conversely, as an indicator of *program quality* as a function of *test time* expended.

(c) APPLICATION

Figure 4 provides an example of the *Shuttle* software entity designated *OIA* and illustrates how *p* might behave as increased *test time* is applied (represented by "test intervals"). From this type of information a program manager can determine whether more testing is warranted, or whether the software is sufficiently tested to allow its release or unrestricted use. Note that required test time rises very rapidly at small values of **p** and **R(t)**. ***Note***: You should read the test time from the left axis as a function of **p**, and read the remaining failures from the right axis, as a function of **p**. Do not combine a value from the test time axis with a value from the remaining failures axis.

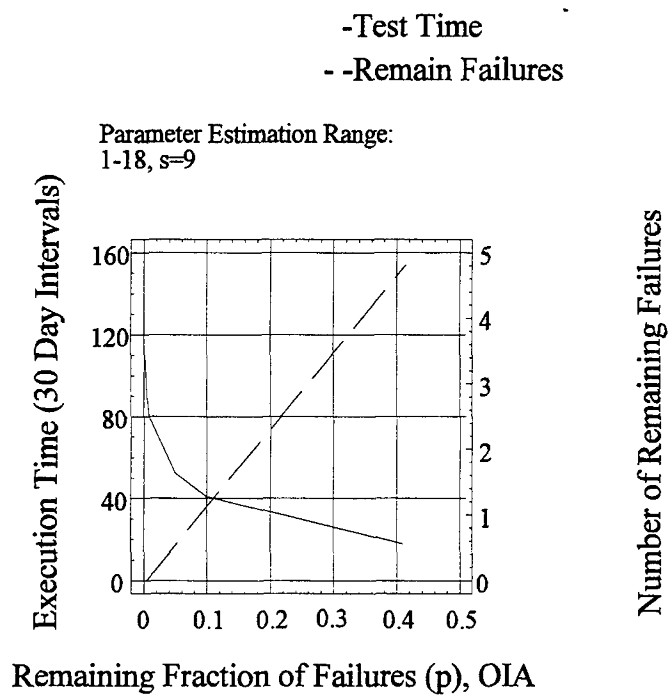


Figure 4: Test Time for Given Remaining Failures

6. MEAN SQUARE ERROR (MSE)

(a) APPLICATION

This section is included here for continuity purposes in discussing the components of the Schneidewind model. Although MSE is not a "prediction" as are the other numerical calculations previously discussed, its determination is key to the success of the model. It is an important statistical value that must be calculated to determine the correct numerical inputs for the model.

Data used in the model is collected from the beginning of the project cycle. However, the software and process used in the software development can change over time. Old data may not have the same relevance as it had when it was "new." For this reason, one may want to ignore "old" data in favor of "new" or more recent data. It may be possible to obtain more accurate predictions of future failures by excluding or giving lower weight to the earlier failure counts. The MSE identifies the time interval where this distinction should be made. There are three types of predictions where MSE can be applied: failure count, time to next failure, and remaining failures.

(b) DEFINITION

The MSE minimizes the sum of the variance and the square of the bias of predicted failures (or time to next failure). It is a statistic that computes the sum of the squared differences between model predictions and actual cumulative failure counts in the range of s, t. This value is used to select the optimum value of the interval where measurements will begin. The following sections describe the computations needed for calculation of MSE. They should be read by the interested reader who desires a mathematical understanding of the calculation process. Other readers may proceed to Section 2.C.7.

(1) **Mean Square Error Criterion for Failure Count**

The Mean Square Error (MSE_F) criterion for failure count is used to select the optimal value of s (i.e., the value of s that results in the minimum value of MSE_F). The result is an optimal triple (β, α, s). The MSE_F computes the mean of the squared differences between model predictions and actual cumulative failure counts $X_{s,i}$ in the range $s \le I \le t$, where $X_{s,i}=X_i-X_{s-1}$.

$$MSE_F=\frac{\sum_{i=s}^{t}[\alpha/\beta(1-\exp(-\beta(i-s+1)))-X_{s,i}]^2}{t-s+1}$$

Figure 5 shows an example of MSE_F in both the parameter estimation range 1,20 (MSE_F computed prior to prediction) and the prediction range 21,30 (MSE_F computed after prediction). Because the latter MSE_F is a minimum at s=11 -- the same as the former -- it confirms that s=11 would have been the best interval to start using the failure data.

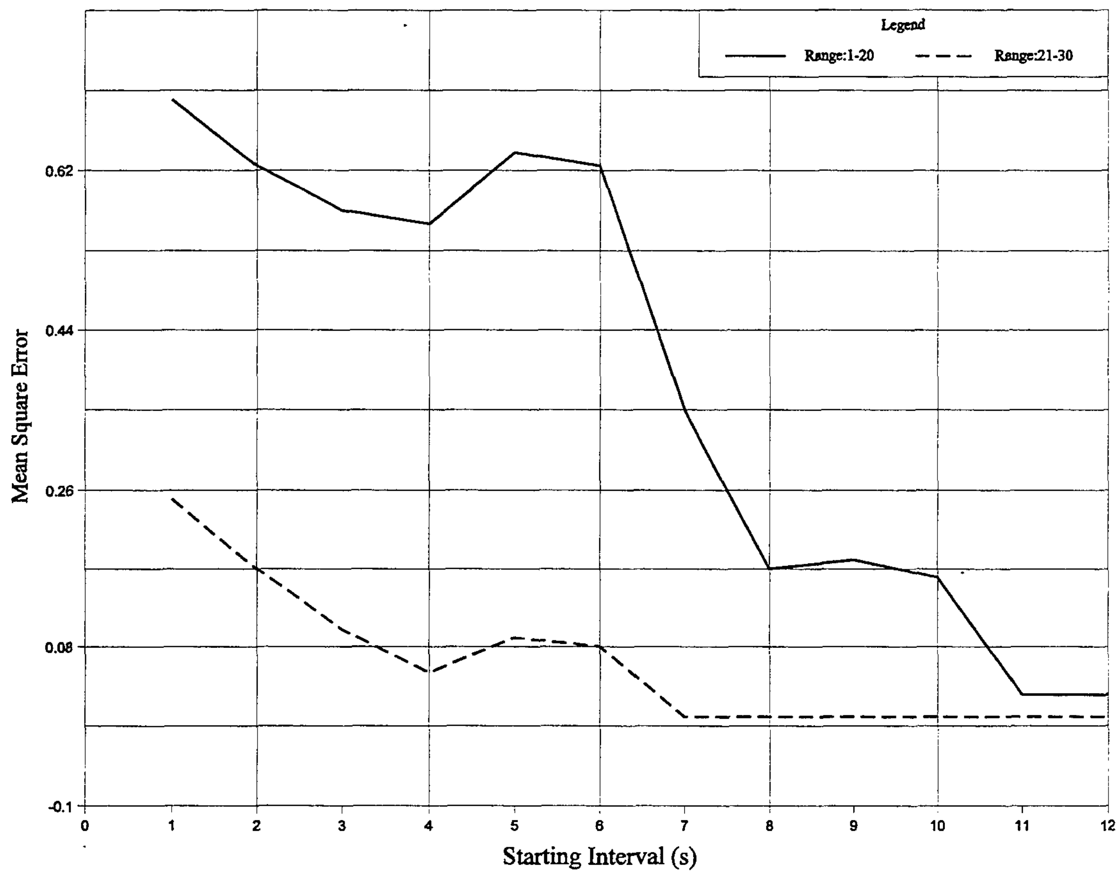


Figure 5: Prediction 21-30. Parameter Estimation 1-20

(2) **Mean Square Error Criterion for Time to Next Failure(s)**

The Mean Square Error (MSE_T) criterion for time to next failure(s) is defined similarly and is given by:

$$MSE_T = \frac{\sum_{i=s}^{J-1} [[\log[\alpha/(\alpha-\beta(X_{s,i}+F_{ij}))]/\beta-(i-s+1)]-T_{ij}]^2}{(J-s)}$$

for $(\alpha/\beta)>(X_{s,i}+F_{ij})$

The terms in MSE_T have the following definitions:

i:	Current interval;
j:	Next interval j>i where $F_{ij}>0$;

$X_{s,i}$:	Cumulative number of failures observed in the range s,i;
F_{ij}:	Number of failures observed during j since i;
T_{ij}:	Time since i to observe number of failures F_{ij} during j (i.e., T_{ij}=j-i)
t:	Upper limit on parameter estimation range; and
J:	Maximum $j \leq t$ where $F_{ij}>0$.

Figure 6 shows both MSE_T and Mean Relative Error (MRE=Σ_i ($|X_i-F_i|/X_i$)/N for N intervals) versus *s* for the post-prediction range. The same MSE_T result was obtained for the observed range. In this case, s=5 was identified as best prior to prediction but s=6 turned out to be best after prediction.

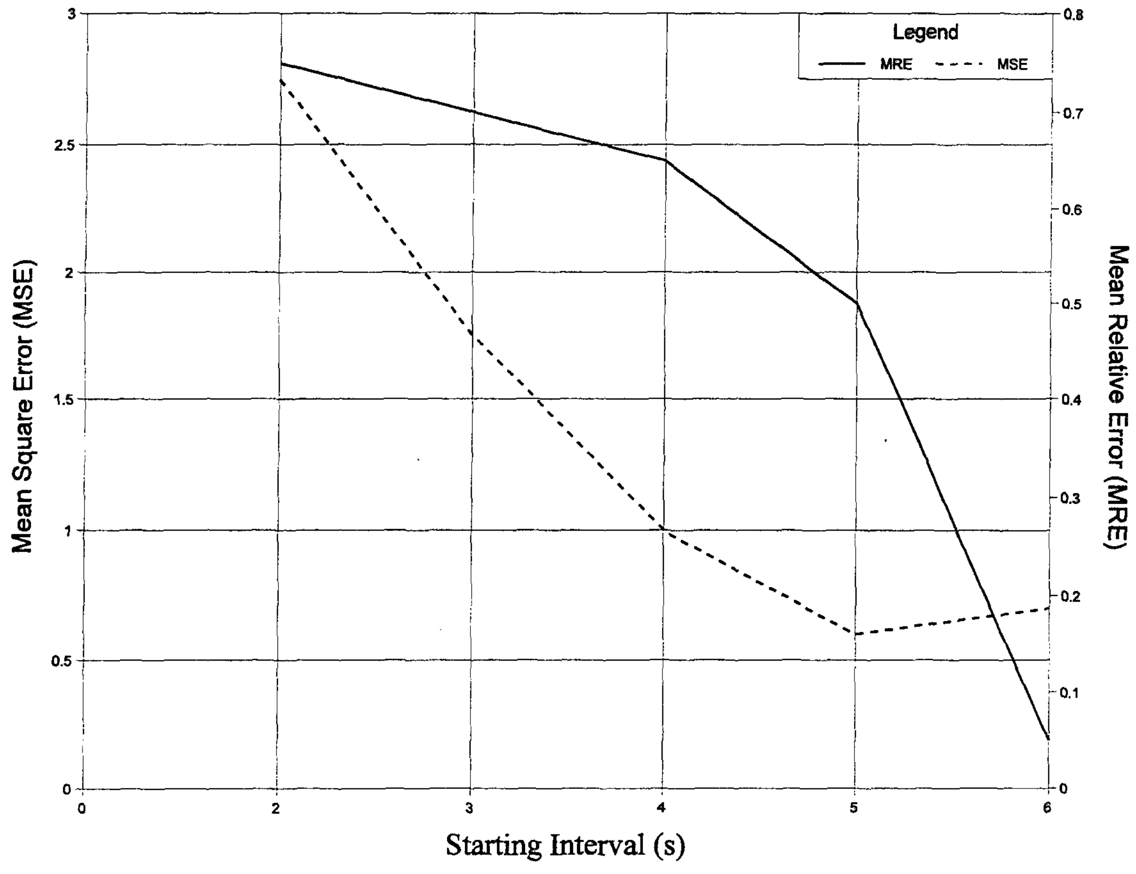


Figure 6: MSE and MRE: Time to Failure

(3) **Mean Square Error Criterion for Remaining Failures**

The Mean Square Error (MSE_R) criterion for number of remaining failures is given by:

$$MSE_R = \frac{\sum_{i=s}^{t} [F(i)-X_i]^2}{t-s+1}$$

where F(I) is the predicted cumulative failures at time I and X_i is the cumulative observed failures at time I.

It should be noted that parameter estimates and MSE evaluations are model setup operations -- not predictions of the future. Rather, during setup, the model is tuned to obtain the best estimates of the parameters by making the best fit of the model to the observed failure data (MSE). Once this has been accomplished, the model is ready to be used for future predictions.

7. TEST TIME TO ACHIEVE SPECIFIED REMAINING FAILURES

(a) DEFINITION

The predicted test time required to achieve a specified number of remaining failures, where $R(t_t)$ is the specified number of remaining failures at t_t, is:

$$t_t=[\log[\alpha/(\beta[R(t_t)])]]/\beta+(s-1)$$

(b) APPLICATION

This concept is shown in Figure 7 for *OIA*, where *remaining failures*=.6 at $\mathbf{t_t}$=52 is marked. This value of $\mathbf{t_t}$ also is in the region of the graph where further increases in $\mathbf{t_t}$ would not result in a significant increase in reliability. The value of this prediction is that software managers can: 1) plan for the amount of test time necessary to achieve a specified reliability goal and 2) determine whether the reliability goal will be achieved with a given amount of test time.

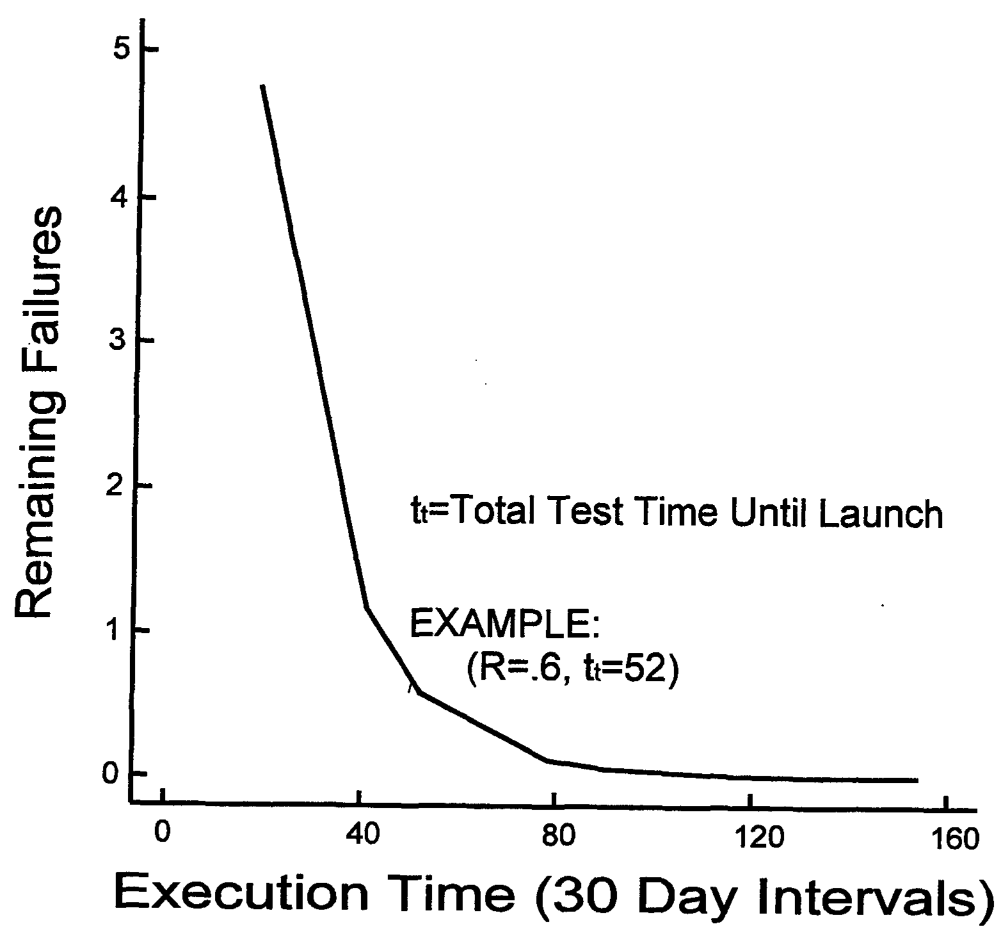


Figure 7: Remaining Failures vs. Test Time

Another type of analysis that can be made with *test time* is shown in Figure 8 where $\mathbf{t_t}$ is plotted as a function of **p** for three modules. The benefit of this prediction is that the software manager can predict how much test time should be allocated to each module to achieve a given level of reliability,

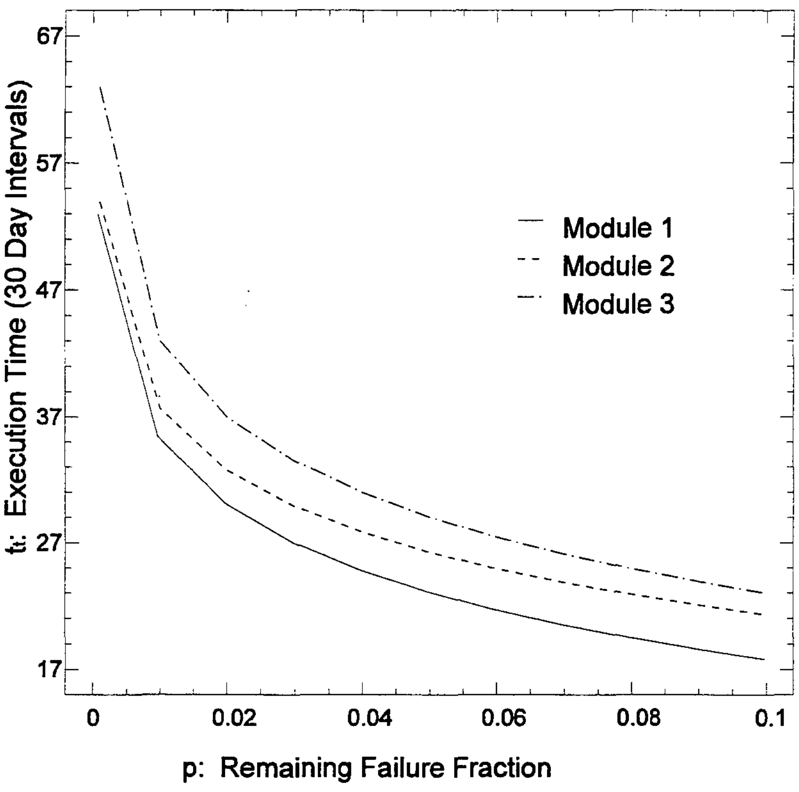


Figure 8: Execution Time to Reach Fraction of Remaining Failures

as specified by **p**. For example, in Figure 8, for a given **p,** *Module 3* will require the most test time. Conversely, for a given $\mathbf{t_t}$ this module will have the worst predicted reliability (SCH92).

These figures can be used as management decision tools. The graphical representations of test time predictions provide the manager with valuable information. He can use this information to allocate his resources to include additional test time and personnel. These decisions will be based on his priorities and the predicted software reliability.

8. TEST TIME NEEDED TO OBTAIN "FAULT FREE" SOFTWARE

(a) DISCUSSION

"Fault Free" software can be described as software where the remaining number of failures over the life of the software is, for practical purposes, "zero," (e. g. .0001). There would be no failures remaining in the software. The predicted test time required to achieve a specified number of remaining failures is calculated using the Schneidewind model.

(b) APPLICATIONS

This value can provide management with an approximate time value, and hence, dollars, it would take to test the software until there are "zero" failures remaining. He may decide to allocate all his resources to testing this particular piece of software, or he may decide to stop testing and send the software back to the developers for repairs and modifications.

9. OPERATIONAL QUALITY

(a) DEFINITION

The operational quality of software is defined as: **Q=1-p** (Where "p" was defined as the fraction of remaining failures).

This equation is a useful measure of the *operational quality* of software because it measures the degree to which faults have been removed from the software, relative to predicted *total failures*. Operational Quality is plotted against Execution Time in Figure 9. We again observe the asymptotic nature of the reliability-testing relationship in the great amount of testing required to achieve high levels of quality.

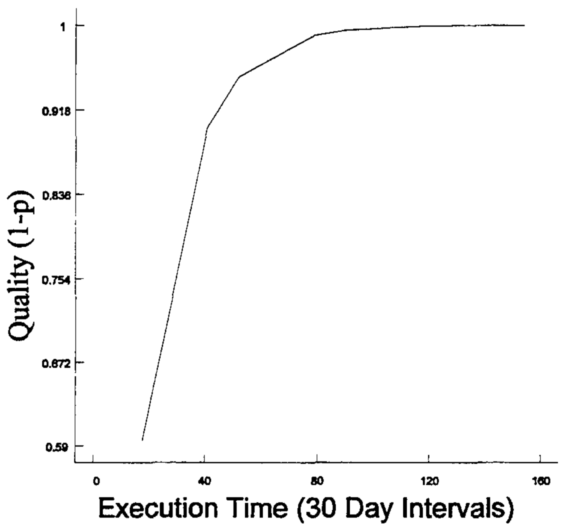


Figure 9: Quality versus Test Time

(b) APPLICATION

When management is provided with this information, it can make trade-off decisions regarding quality and cost (inspection time). Higher quality will require more inspection time. The converse is also true. The manager can inspect the trade-off curve and decide where he receives the best gains for his investment. The curve will eventually show decreasing marginal gains.

D. SUMMARY

This section provided some background information on the types of predictions available by employing the Schneidewind Software Reliability Model. It also gave managerial applications for use of the predictions. An important prerequisite using a reliability model is good failure data. For without data, no predictions would be possible. It cannot be emphasized enough how important it is to collect data as early in the development process as possible.

The next section will discuss how an organization can make the predictions discussed in Section 2 by using certain software packages. Additionally, application of these predictions to the *Shuttle* program will be discussed.

SECTION 3: TESTING METHODOLOGIES EMPLOYED

The following section discusses the three key components to making software reliability predictions. These components include the two software packages that make the necessary predictions easier to compute (SMERFS and Statgraphics) and the reliability model itself (Schneidewind Software Reliability Model).

A. SMERFS

Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS) is a software reliability modeling tool that can be used to gain insight into the reliability of the software being tested. SMERFS is a tool that implements the models developed by Schneidewind and a number of other software reliability researchers. Using the Schneidewind Model component of SMERFS, two types of predictions can be made: for a given number of time intervals, how many failures will occur? secondly, for a given number of failures, how many time intervals will be required for the failures to occur? After inputing the software failure count data, usually from an input failure data file, the first step is to determine the optimal starting value for "s" as determined by the table of MSE values; usually the "s" with the minimum MSE will be selected.

B. THE SCHNEIDEWIND SOFTWARE RELIABILITY MODEL

As stated above, SMERFS is a statistical software tool that can perform various calculations on an input failure data file to predict both the number of failures and the time to next failure. However, before these predictions can be made with the Schneidewind Software Reliability Model, SMERFS must calculate the Mean Square Error, as previously discussed to determine the optimal starting interval, "s," which corresponds to the minimum MSE between predicted and actual values of failure counts or time to failure.

C. STATGRAPHICS

Statgraphics is a software tool designed to aid in calculations of mathematical formulas and provides statistical analysis and graphing capabilities. This tool is used to predict the required test time to achieve a desired reliability level, using the following formula:

$$t_t = [\log[\alpha/(\beta\,[R(t_t)])]]/\beta + (s-1)$$

The values for alpha, beta, and "s" are retrieved from the data collected using SMERFS. Volume IV of the Handbook contains a complete list of additional equations that are implemented in Statgraphics.

D. TESTING PROCEDURES

Using SMERFS, one can address the following two objectives: (1) How a reliability model can be used to predict execution time to next failure, and (2) How a reliability model can be used to predict how long the software should be tested in order for it to be "fault free." The following instructions for SMERFS will achieve these objectives.

1. USING SMERFS

Although most of the instructions for SMERFS show up on the computer screen and are self-explanatory, the following amplifying instructions will assist the first-time user in successfully completing his session. See Appendix B to follow along with the SMERFS printout. User inputs are highlighted (in bold print) for ease of use. ***Note***: Calculation results should be rounded to no more than one or two decimal places, because reliability cannot be predicted with greater precision. However, to be consistent with the SMERFS printouts in Appendix B, the results shown in this section will be left as calculated.

a. Once SMERFS is accessed, the first input required from the user will be the name of the file where he would like the SMERFS output (results) stored. As an example, **a:\smerfs1** would store the resulting SMERFS ASCII file on the computer's A-drive if a disk is inserted. This will make data retrieval easier once the session is complete. The user can then access his "output" file via a word processing program, format the data as he wishes, and print the results.

b. The user will then be asked if he would like to store a plot file for later retrieval. The recommended answer for this question is **0**, (zero), meaning "No".

c. SMERFS will next require the failure data type the user will be working with. At this point the user will enter **4**, for the interval failure counts and testing lengths.

d. Now he will be asked to **enter a 1** for the standard SMERFS file input. This should be followed by the name of the file where his sample data is stored, for example, a file name of **oid18.in**. [This sample file contains the number of failures recorded against an operational increment (OI) of the *Shuttle*. This OI consists of a build of various modules in the *Shuttle* software library. There are 18 count intervals in **oid18.in**. Each interval is 30 days of continuous execution time.]

e. This step will ask the user how he would like the input displayed. The recommended response is to **enter a 3**. This entry will show a table of all the data input through the oid18.in file. However, the user may enter a 0 to display a list of his options at this point.

f. Following the display of data, the same question will reappear regarding the input display. This time the user is recommended to **enter 4** to take him to the SMERFS main menu. He will then be asked if he would like to make some new data files. He should **enter a 0** to void the data restore option.

g. He should then **enter 0** to display the listings available at the main menu. This will present him with nine choices. He should select **option 8 (Executions of the models).**

h. Upon this selection, the user will then **enter a 0** to display the available count model options. He should select **option 4 (The Schneidewind Model)**.

i. The next displays will permit the user to see descriptions of the model or the treatment type. For these options, a **0** should be entered for each option unless he desires the descriptions.

j. The next step will be to investigate the "optimum s" from the various count intervals input into the program. A **1** should be entered here. He will then be asked to enter the range over which "s" should be tested. The user should enter the range of the input failure data: **enter 1,18**. This entry will display the table of s, beta, alpha, WLS, MSE_F and MSE_T. The last two terms are the mean square error, as a function of "s", for number of failures and time to failure predictions, respectively (ignore the "WLS" column).

The user should note the table results and select those values for "s" which give him the **smallest** MSE_F and MSE_T.

NUMBER OF FAILURES PREDICTIONS

k. This step moves the user into predicting the number of failures that will occur in one more test period. He will be prompted to enter the model treatment number. He should **enter a 2.**

l. He will then be prompted to enter the associated value of "s" he would like to investigate. He should enter the "s" value corresponding to the minimum value for MSE_F he recorded earlier. For this example, the **value of 6** should be entered. The key values obtained are the *total number of failures*, the number corresponding to *plus those skipped*, and the *number of failures remaining.* If the value for *plus those skipped* is not equal to zero, this value must be added to the total number of failures and the number of failures remaining. **The user should record these values**. The example values correspond to

Total number of failures:	14.363
Plus those skipped:	3
# of failures remaining:	4.3626

m. The program will present the user with two options for data evaluation. He should **choose option 1** for the number of failures expected in the next testing period. He will be prompted to enter the number of periods to examine. He should **enter a 1.** This will display the number of failures expected. For this example, it will be .36888. This implies that the number of remaining failures occurring in the next execution cycle (30 days) will be .37.

TIME TO FAILURE PREDICTIONS

n. Enter **0** to conclude the ***NUMBER OF FAILURES PREDICTIONS***. He will then be asked to enter the desired model treatment number. He should enter **2.** For the number of associated values of "s" he should enter the corresponding "s" value that gives the smallest MSE_T, for time to failure prediction. In this example, the minimum value for MSE_T is seen for "s" equal to 5. A **5** should be entered. This entry will result in a display of model estimates.

o. The user will then be prompted to select from two options regarding future predictions. For the sample run, he should select 2 for the prediction of the number of periods needed to discover the next "M" failures. This will allow him to determine the value of "M". He should **enter a 1.** The result will predict the number of *additional* test periods required to discover one more failure. A result of 6.3443 periods results (190.32 days). This implies that the time to next failure, from the present time, will be 190 days.

p. When asked to enter a value of M, the user should **enter 0**. The user will be prompted again to **enter a 0** to end the current predictions.

q. The user can exit the program by entering the following values in sequence: 0 to end period to examine, 0 to end predictions, followed by a 4 to terminate the model execution, 0 to conclude analysis of model fit, 0 for count model options, 6 to return to the main menu, 0 for a list of main module options, and finally, 9 to stop execution of SMERFS.

(1). INTERPRETING SMERFS RESULTS

Using the sample file and the SMERFS software, the following results were achieved:

Number of Failures Data ("s" = 6):

Number of remaining failures (from present time): 7.36

Total number of failures: 17.36

Calculated *fraction of remaining failures*: .42

Number of failures that will occur in one more period: .37

Time to Failure Data ("s" = 5):

Time to next failure (from present time): 6.34 periods (190 days)

Note: Because in this example s=6 is optimal for *number of failures* predictions and s=5 is optimal for *time to failure* predictions, different results are obtained for *number of remaining failures* and *total number of failures*. Because MSE_F applies to failure count quantities like these, the values obtained for s=6 should be used in this example (i.e., *number of remaining failures*=7.36 and *total number of failures*=17.36).

These results provide the manager with useful information regarding the reliability of his software, provided he looks at all the data as complementary information. He should not make a decision based on only one piece of the above information, rather, he needs to look at the data in its entirety.

(2). SCENARIO REVISITED

A manager must decide whether or not to launch the space *Shuttle* for a mission to last ten days. He has collected failure data on the software to be used in the launch and has input the data into the model as described in the above sections.

Looking at the data in its entirety, he should **not launch** the *Shuttle*. Even though the time to next failure is predicted at 190 days and only .37 failures are predicted for the next interval (30 days), the predicted number of remaining failures is 7.36. This is a significantly high number. (As discussed previously, the manager desires this number to be less than one.) The decision must be based on the available model evidence, his confidence in the model, his risk aversion, and any other factors at his disposal. Using only the data from this analysis, the overriding factor of 7.36 possibly life-threatening remaining failures, the manager should not launch the *Shuttle*.

2. USING STATGRAPHICS

Statgraphics is used to augment the reliability predictions obtained from SMERFS. Equations, like the one for t_t below, can be created using the Statgraphics equation editor feature. Of particular interest in this phase of the predictions is the formula for predicting the test time required to achieve a given reliability level, as measured by the number of remaining failures $R(t_t)$ and fraction remaining failures, p. As discussed in earlier sections of this handbook, this amount of test time is defined by the following equation:

$$t_t = [\log[\alpha/(\beta[R(t_t)])]]/\beta + (s-1)$$

For this example, given values of $R(t_t)$ will be one, two, three, and four.

a. Once Statgraphics has been accessed, the user will be presented with a menu showing various options for calculations and presentations. He will **depress the F8** function key which will cause a new screen to be superimposed on the menu. Here, he will type "**exec**" for the execution screen to appear.

b. Once the blank screen appears, he should **type tt** at the colon prompt if he wants to see the equation before he uses it in a calculation. Otherwise, he can skip this step. This will display the above t_t equation which has already been preloaded for the user. For Statgraphics to calculate the numerical value for this equation, the user must input the values for alpha, beta, and s and evaluate (EVAL) it. The alpha, beta, and s values correspond to the values obtained from the SMERFS session for the smallest MSE_F value. The values of p to be calculated correspond to the desired number of remaining failures of one, two, three, and four.

c. The user will now enter the above mentioned values in the following format:

```
alpha GETS .738
beta GETS .051
```

```
Rtt GETS 1 2 3 4
s GETS 6
EVAL tt 56.84 43.35 35.47 29.87
p GETS Rtt/17.36
p .0576 .115 .173 .230
```

These commands will display the value for the test time required to achieve a given reliability level. For this input, the predicted test time required to achieve the reliability level of having one remaining failure is 56.84 thirty day intervals. This will correspond to a fraction of remaining failures equal to .0576. The other three values of t_t and p have the same interpretation.

The above results could be plotted to compare the effect that changing the remaining failures has on the amount of test time needed to achieve that end. An asymptotic relationship is seen between t_t and the fraction of remaining failures, p. Figure 10 is a sample graph that could be obtained.

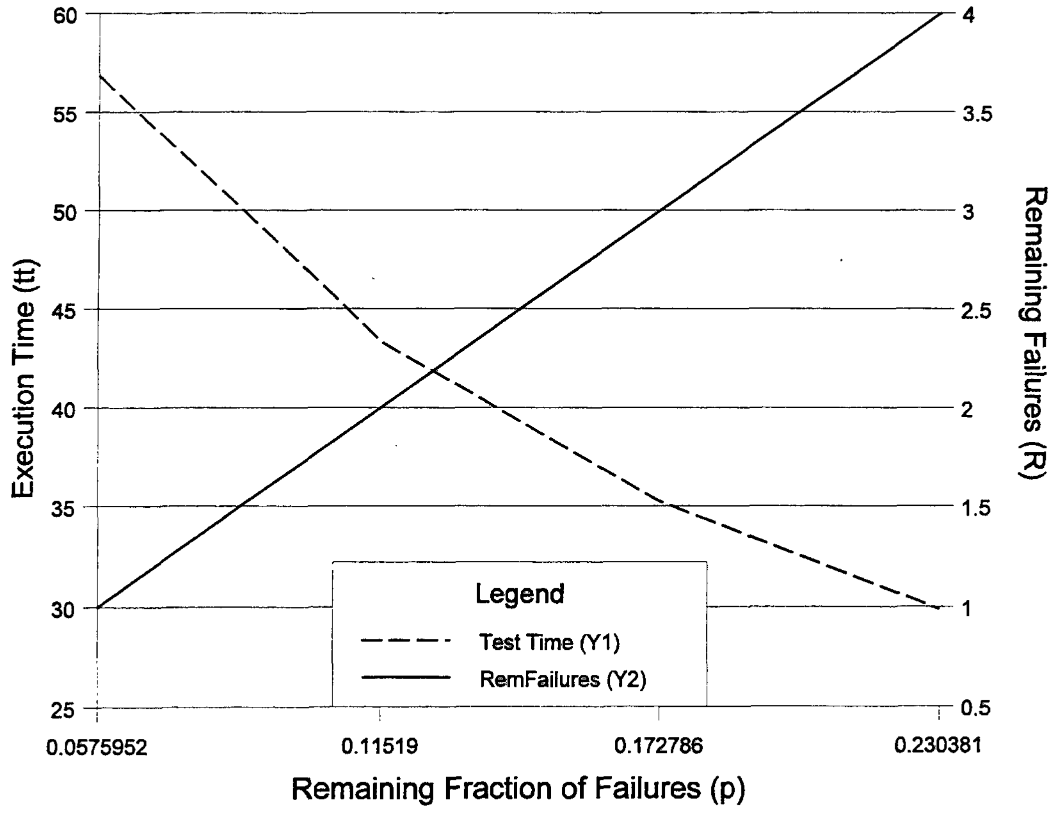


Figure 10: t_t and R versus Remaining Fraction of Failures (p)

(1). APPLICATION

With this information, the manager could gain insight into the predicted amount of time it would take to achieve given reliability levels. Using the scenario mentioned previously, as an example, one could see that it is predicted to take almost 57 periods (totaling 4.7 years) from t=0 to reduce the fraction of remaining failures to .058. The test time curve indicates that there will be a point where there are only marginal returns achieved by additional testing.

Looking at the shape of the curves on Figure 10, the software manager must understand that as predicted reliability increases (the number of predicted failures decreases) there will be a significant increase in the amount of testing time needed to achieve those results. There will come a point were the additional cost of testing will result in only minimal gains in reduced software failures. The manager must make the decision whether to stop testing and deploy the software, based on available funding for testing and the desired reliability levels.

E. CONCLUSION

Management must use all resources available to it to come to a sound, information-supported decision. The model is only a tool to help make this decision. The predictions provided by the Schneidewind Software Reliability Model can give management additional information on the predicted reliability of its software. This can be accomplished by both the developer and customer using the software reliability engineering process that has been described in this handbook. Using appropriate failure data, the predictions can be used to help make an informed reliability decision. However, the final decision must be made by the manager based on *all* the information he has available to him.

Acknowledgments

We acknowledge the support provided for this project by Colonel David Chadwick, former Commanding Officer and Captain Kenneth Warburton, U.S. Marine Corps Tactical Systems Support Activity; Dr. William Farr, Naval Surface Warfare Center; and Mr. Ted Keller and Ms. Patti Thornton of Lockheed-Martin Space Mission Systems & Services, and Prof. John Munson, University of Idaho for providing data.

REFERENCES

[AIA93] Recommended Practice for Software Reliability, R-013-1992, American National Standards Institute/American Institute of Aeronautics and Astronautics, 370 L'Enfant Promenade, SW, Washington, DC 20024, 1993.

[BIL94] C. Billings, et al, "Journey to a Mature Software Process", IBM Systems Journal Vol. 33, No. 1, 1994, pp. 46-61.

[FAR93] William H. Farr and Oliver D. Smith, Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS) Users Guide, NAVSWC TR-84-373, Revision 3, Naval Surface Weapons Center, Revised September 1993.

[IEE93] ANSI/IEEE Standard for a Software Quality Metrics Methodology, IEEE 1061 June, 1993.

[KEL95] Ted Keller, Norman F. Schneidewind, and Patti A. Thornton "Predictions for Increasing Confidence in the Reliability of the Space Shuttle Flight Software", Proceedings of the AIAA Computing in Aerospace 10, San Antonio, TX, March 28, 1995, pp. 1-8.

[MUS87] John Musa, et al, Software Reliability: Measurement, Prediction, Application, McGraw-Hill, New York, 1987.

[SCH97] Norman F. Schneidewind, "Reliability Modeling for Safety Critical Software", IEEE Transactions on Reliability, Vol. 46, No.1, March 1997, pp.88-98.

[SCH93] Norman F. Schneidewind, "Software Reliability Model with Optimal Selection of Failure Data", IEEE Transactions on Software Engineering, Vol. 19, No. 11, November 1993, pp. 1095-1104.

[SCH92a] Norman F. Schneidewind, "Methodology for Validating Software Metrics", IEEE Transactions on Software Engineering, Vol. 18, No. 5, May 1992, pp. 410-422.

[SCH92] Norman F. Schneidewind and T.W. Keller, "Application of Reliability Models to the Space Shuttle", IEEE Software, Vol. 9, No. 4, July 1992 pp. 28-33.

[SCH75] Norman F. Schneidewind, "Analysis of Error Processes in Computer Software", Proceedings of the International Conference on Reliable Software, IEEE Computer Society, 21-23 April 1975, pp. 337-346.

[WAR94] Kenneth M. Warburton, "Towards Better Quality and Reliability in the Software Reuse Library Environment," Master's Thesis, Naval Postgraduate School, March 1994.

APPENDIX A. SOFTWARE DISCREPANCY REPORT

Date of Failure: Report Originator: Office Code: Project/System Name:	**Discrepancy Report Number**: Telephone Number: Organization: Program Designation: Version:
Category: S (software) H (hardware) P (people)	**Priority/Severity**: 1 (Loss of Life, Mission Aborted) 2 (Degradation in performance) 3 (Operator Annoyance) 4 (Documentation Error) 5 (Error Classification Problem)
Test Procedure: Simulation Used: Linking with: Configuration/Transients in Memory:	
Failure Data: (check one) CPU Time since Last Failure: Clock Time since Last Failure: Manhours expended since Last Failure: **Problem Duplicated**: Yes or No During Run:	**Project Phase**: (check one) Software Requirements Detailed Design Software Integration & Testing Operations / Maintenance Preliminary Design Code / Unit Testing Systems Integration Test
Symptom Classification: **Operating System Crash**: **Program Hang up**: Input Problem: Correct input not accepted Description incorrect or missing Parameters incomplete or missing Output Problem: Wrong format Incorrect result Incomplete or missing output Failed Required Performance: Perceived Total Product Failure: System Error Message: Other (Explain):	

SOFTWARE DISCREPANCY REPORT (CONTINUED)

Dump Date:
Documents Affected:
Responsible Modules:
Reference Document:
Function Affected:

Project Activity:
Analysis
Inspection
Review
Compile
Audit
Test
Operation
Validation/Qual Test

Actual Cause of Problem
- Product Software/ Database
- Product Hardware
- Test Software
- Documentation
- Interface
- Operator Error
- Enhancement (Perceived inadequacies)

Testing to Verify Fix:
Source of Problem:
Time Required for Analysis:

Disposition:
- Closed:
  - Corrective Action Taken
  - Non-Software Problem
  - Duplicate Problem in STR #
  - Fix not justified
- Open:
  - Deferred to a Later Release
  - Other:
- Merged with another Problem

QA Sign-Off: **Date**:

APPENDIX B. EDITED SAMPLE SMERFS SESSION

```
*ASTERISKS INDICATE COMMENTS TO DISTINGUISH THEM FROM SMERFS OUTPUT*

*READ IN DATA THAT WAS PREVIOUSLY GENERATED BY SMERFS FROM ASCII FILE
INPUT*

*TESTING INTERVAL WILL ALWAYS BE "1" IN SMERFS. IN THE APPLICATION IT WILL
BE THE ACTUAL LENGTH OF EACH INTERVAL (E.G., 1 HOUR, 1 DAY, 30 DAYS)*

 ENTER DESIRED DATA TYPE, OR ZERO FOR A LIST.
      4  *REFERS TO FAILURE COUNT DATA*

 ENTER ONE FOR A STANDARD SMERFS FILE INPUT; ELSE ZERO.
      1

 ENTER INPUT FILE NAME FOR INTERVAL DATA.
oid18.in

 THE INPUT OF  18 INTERVAL ELEMENTS WAS PERFORMED.

 ENTER INPUT OPTION, OR ZERO FOR A LIST.
      0
 THE AVAILABLE INPUT OPTIONS ARE:
  1 ASCII FILE INPUT
  2 KEYBOARD INPUT
  3 LIST THE CURRENT DATA
  4 RETURN TO THE MAIN PROGRAM
 ENTER INPUT OPTION.
      3

INTERVAL  NO. OF FAULTS    TESTING LENGTH
 ======   =============    =============

    1     .00000000E+00    .10000000E+01
    2     .00000000E+00    .10000000E+01
    3     .00000000E+00    .10000000E+01
    4     .00000000E+00    .10000000E+01
    5     .30000000E+01    .10000000E+01
    6     .10000000E+01    .10000000E+01
    7     .00000000E+00    .10000000E+01
    8     .10000000E+01    .10000000E+01
    9     .00000000E+00    .10000000E+01
   10     .10000000E+01    .10000000E+01
   11     .10000000E+01    .10000000E+01
```

```
12     .00000000E+00    .10000000E+01
13     .20000000E+01    .10000000E+01
14     .00000000E+00    .10000000E+01
15     .00000000E+00    .10000000E+01
16     .00000000E+00    .10000000E+01
17     .00000000E+00    .10000000E+01
18     .10000000E+01    .10000000E+01
```

FIND THE BEST STARTING INTERVAL FOR USING THE FAILURE DATA. SINCE THERE IS A TOTAL OF 18 INTERVALS OF DATA, USE THE RANGE 1,18. SMERFS WILL ONLY PRODUCE A RESULT FOR "S" WHERE IT CAN OBTAIN CONVERGENCE. FOR FAILURE COUNT PREDICTIONS, USE THE MINIMUM MSE-F "S"; FOR TIME TO FAILURE PREDICTIONS, USE THE MINIMUM MSE-T "S".

```
ENTER ONE TO INVESTIGATE FOR THE OPTIMUM S (USING TREATMENT TYPE
 NUMBER 2); ELSE ZERO TO CONTINUE WITH THE MODEL EXECUTION.
    1
 ENTER RANGE OVER WHICH S  SHOULD BE TESTED.  NOTE, AN EXECUTION
 ON A GIVEN S WHICH FAILED THE CONVERGENCE CRITERIA  WILL NOT BE
 INCLUDED IN THE FOLLOWING RESULTS TABLE.  THE OPTIMUM S FOR EI-
 THER MSE-F OR MSE-T IS THE ONE  RESULTING IN THE SMALLEST VALUE
 FOR YOUR CHOSEN CRITERIA.
    1     18
 S   BETA        ALPHA       WLS         MSE-F       MSE-T
 === =========== =========== =========== =========== ===========

 1 .37154E-02  .57434E+00  .71189E+00  .89573E+00  .15098E+01
 2 .25076E-01  .72250E+00  .84899E+00  .68418E+00  .12947E+01
 3 .52370E-01  .92300E+00  .10130E+01  .47735E+00  .10803E+01
 4 .88195E-01  .12021E+01  .12214E+01  .34612E+00  .86076E+00
 5 .13700E+00  .16059E+01  .15409E+01  .47758E+00  .60788E+00
 6 .51401E-01  .73825E+00  .58125E+00  .24450E+00  .11042E+01
 7 .28025E-01  .58878E+00  .50090E+00  .30476E+00  .13863E+01
```

9 .60985E-01 .66786E+00 .61535E+00 .28068E+00 .13683E+01

ENTER DESIRED MODEL TREATMENT NUMBER, OR FOUR TO TERMINATE MODEL EXECUTION.

2 *METHOD WHEREBY INTERVALS 1,..., S-1 ARE DISCARDED*

ENTER ASSOCIATED VALUE OF S (LESS THAN THE NUMBER OF PERIODS).

6 *CORRESPONDS TO MINIMUM MSE-F ABOVE BECAUSE WE WILL BE MAKING A FAILURE COUNT PREDICTION.*

```
TREATMENT 2 MODEL ESTIMATES ARE:
BETA                 .51401E-01
ALPHA                .73825E+00
TOTAL NUMBER OF FAULTS   .14363E+02
 PLUS THOSE SKIPPED      .30000E+01 IN PERIODS 1 THROUGH   5 *(INTERVALS
 1,...,S-1)*
# OF FAULTS REMAINING   .43626E+01
WEIGHTED SUMS-OF-SQUARES
 BETWEEN PREDICTED AND
 OBSERVED FAULTS       .58125E+00
MEAN SQUARE ERROR FOR
 CUMULATIVE FAULTS      .24450E+00
MEAN SQUARE ERROR FOR
 TIME TO NEXT FAILURE   .11042E+01
```

* CORRECT PREDICTED TOTAL NUMBER OF FAILURES = 14.36+3.0 (NUMBER SKIPPED)=17.36*

* ACTUAL TOTAL NUMBER OF FAILURES=14 (FAILURES OBSERVED AFTER 65.03 INTERVALS)*

* CORRECT PREDICTED NUMBER OF REMAINING FAILURES=4.26+3.0 (NUMBER SKIPPED)=7.36*

```
*ACTUAL NUMBER OF REMAINING FAILURES=4 (FAILURES OBSERVED BETWEEN 18
AND 65.03 INTERVALS)*

 THE AVAILABLE FUTURE PREDICTIONS ARE:
  1) THE NUMBER OF FAULTS EXPECTED IN THE NEXT TESTING PERIOD
  2) THE NUMBER OF PERIODS NEEDED TO DISCOVER THE NEXT M FAULTS
 ENTER PREDICTION OPTION, OR ZERO TO END PREDICTIONS.
       1
  ENTER NUMBER OF PERIODS TO EXAMINE, OR ZERO TO END.
     1.000000000000000      *WANT PREDICTION FOR INTERVAL T=19*
 # OF FAULTS EXPECTED      .36888E+00
 *ACTUAL NUMBER OF FAILURES IN NEXT INTERVAL (T=19)=0*
 *MODEL IS ENTERED AGAIN SO THAT THE BEST VALUE OF "S" FOR TIME TO
FAILURE PREDICTION CAN  BE USED*

 ENTER DESIRED MODEL TREATMENT NUMBER, OR FOUR TO TERMINATE MODEL
EXECUTION.
       2
 ENTER ASSOCIATED VALUE OF S (LESS THAN THE NUMBER OF PERIODS).
       5        *CORRESPONDS TO MINIMUM MSE-T ABOVE BECAUSE WE WILL BE
MAKING A TIME TO FAILURE  PREDICTION.*
*THE USUAL LISTING IS NOT SHOWN BECAUSE THE TOTAL FAILURES AND
REMAINING FAILURES WERE OBTAINED AS PART OF THE FAILURE COUNT
PREDICTION*

 THE AVAILABLE FUTURE PREDICTIONS ARE:

  1) THE NUMBER OF FAULTS EXPECTED IN THE NEXT TESTING PERIOD
  2) THE NUMBER OF PERIODS NEEDED TO DISCOVER THE NEXT M FAULTS
```

```
ENTER PREDICTION OPTION, OR ZERO TO END PREDICTIONS.
      2

  ENTER VALUE OF M (BETWEEN ONE AND  .17221E+01), OR ZERO TO END. *(THIS IS
THE RANGE OF   PREDICTED REMAINING FAILURES)*
     1.000000000000000      *WANT PREDICTION FOR ONE MORE FAILURE*

  # OF PERIODS EXPECTED      .63443E+01 *(I.E., T=18+6.34=24.34)*

*ACTUAL TIME TO NEXT FAILURE=6.2 (I.E., T=18+6.2=24.2)*
```

DISTRIBUTION LIST

Agency	No. of Copies
Defense Technical Information Center 8725 John J. King Rd., STE 0944 Ft. Belvoir, VA 22314	2
Dudley Knox Library, Code 013 Naval Postgraduate School Monterey, CA 93943	2
Office of Research Administration, Code 91 Naval Postgraduate School Monterey, CA 93943	1
Department of Systems Management Library, Code SM/Eb Naval Postgraduate School 555 Dyer Rd Rm 239 Bldg. 330 Monterey, CA 93943	1
Commanding Officer Marine Corps Tactical Systems Support Activity Box 555171 Camp Pendleton. CA 92055-5171	1
Capt. Kenneth Warburton Marine Corps Tactical Systems Support Activity Box 555171 Bldg. 31345 Camp Pendleton, CA 92055-5171	5
Dr. Norman F. Schneidewind Naval Postgraduate School Code SM/Ss Monterey, CA 93943	10