NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

SOFTWARE EVALUATION FOR DEVELOPING SOFTWARE RELIABILITY ENGINEERING AND METRICS MODELS

by

Dennis J. Brophy
and
James D. O'Leary

March 1999

Thesis Advisor: Norman F. Schneidewind
Associate Advisor: Douglas Brinkley

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1999	Master's Thesis

4. TITLE AND SUBTITLE :
Software Evaluation for Developing Software Reliability Engineering and Metrics Models

5. FUNDING NUMBERS

6. AUTHOR(S)
Brophy, Dennis J. and O'Leary, James D.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

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

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

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

Today's software is extremely complex, often constituting millions of lines of instructions. Programs are expected to operate smoothly on a wide variety of platforms. There are continuous attempts to try to assess what the reliability of a software package is and to predict what the reliability of software under development will be. The quantitative aspects of these assessments deal with evaluating, characterizing and predicting how well software will operate. Experience has shown that it is extremely difficult to make something as large and complex as modern software and predict with any accuracy how it is going to behave in the field. This thesis proposes to create an integrated system to predict software reliability for mission critical systems. This will be accomplished by developing a flexible DBMS to track failures and to integrate the DBMS with statistical analysis programs and software reliability prediction tools that are used to make calculations and display trend analysis. It further proposes a software metrics model for fault prediction by determining and manipulating metrics extracted from the code.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Software Reliability, Software Metrics, Excel, Database	74
	16. PRICE CODE

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

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Approved for public release; distribution is unlimited

SOFTWARE EVALUATION FOR DEVELOPING SOFTWARE RELIABILITY ENGINEERING AND METRICS MODELS

Dennis J. Brophy
Commander, United States Naval Reserve
B.S., State University of New York, 1981

James D. O'Leary
Lieutenant Commander, United States Navy
B.A., Assumption College, 1987

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
March 1999

Authors: _____
Dennis J. Brophy

_____
James D. O'Leary

Approved by: _____
Norman F. Schneidewind, Thesis Advisor

_____
Douglas Brinkley, Associate Advisor

_____ for
Reuben T. Harris, Chairman
Department of Systems Management

_____
Dan Boger, Dean
Division of Computer and Information Sciences and Operations

ABSTRACT

Today's software is extremely complex, often constituting millions of lines of instructions. Programs are expected to operate smoothly on a wide variety of platforms. There are continuous attempts to try to assess what the reliability of a software package is and to predict what the reliability of software under development will be. The quantitative aspects of these assessments deal with evaluating, characterizing and predicting how well software will operate. Experience has shown that it is extremely difficult to make something as large and complex as modern software and predict with any accuracy how it is going to behave in the field. This thesis proposes to create an integrated system to predict software reliability for mission critical systems. This will be accomplished by developing a flexible DBMS to track failures and to integrate the DBMS with statistical analysis programs and software reliability prediction tools that are used to make calculations and display trend analysis. It further proposes a software metrics model for fault prediction by determining and manipulating metrics extracted from the code.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. INTRODUCTION

A. BACKGROUND

The computer revolution of the last four decades has transformed the world. We have been witnesses to the most rapid period of technological advancement in history. Today, our society is totally dependent on computer hardware and software. Together they permeate every aspect of our lives. Computers are imbedded in wristwatches, telephones, home appliances, buildings, and aircraft. We can look at virtually any industry - automotive, avionics, oil, telecommunications, banking, semiconductor, pharmaceuticals - they all are highly, if not totally reliant on computers for their functioning [LYU97].

When considering software, people commonly think of the programs that run their PCs, track their bank accounts, or control their microwaves, but the technology goes much deeper than that. The infrastructure on which our society depends is monitored and operated entirely with software. The power grid that supplies us with electricity, our water supply, and food distribution, are all completely reliant on software to function.

The size and complexity of computer intensive systems has grown dramatically during the past decade. The very fact that these systems have grown so large and complicated has produced the need for other systems to help us determine their true capabilities and reliability. Decision Support Systems, Expert Systems, and Intelligent Systems are being used to assist in judging the effectiveness of these powerful tools. Examples of highly complex systems can be found in projects undertaken by NASA, the Department of Defense, the Federal Aviation Administration, the telecommunications industry, and a variety of other private industries. For instance, the Space Shuttle flies

with approximately 400,000 lines of software code on board and approximately 3.5 million lines of code in ground control and processing. After being scaled down significantly from its original plan, the International Space Station is still projected to have millions of lines of software to operate many hardware pieces for its navigation, communication, and experimentation. In the telecommunications industry, operations for phone carriers are supported by hundreds of software systems, with hundreds of millions of lines of source code. In the avionics industry, almost all new payload instruments contain their own microprocessor system with extensive embedded software. A massive amount of hardware and complicated software also exists in the Federal Aviation Administration's Advanced Automation System, the new generation air traffic control system [LYU97].

The demand for complex systems has increased more rapidly than the ability to design, implement, test, and maintain them. When the requirements for and dependencies on computers increase, the possibility of crises from computer failures also increases. The impact of these failures ranges from inconvenience (malfunctions of home appliances), economic damage (interruptions of banking systems), to loss of life (failures of flight systems or medical equipment). Needless to say, the reliability of computer systems has become a major concern for our society [LYU97].

One of the most important areas of research in the software-engineering field is that of Software Reliability Engineering (SRE). The purpose of SRE is to improve the reliability of fielded systems and to treat today's Multi-Function Distributed Systems (MFDS) in a realistic manner for the purpose of software reliability modeling and prediction [SCH97]. Representing the "intellectual effort" of its authors, software

includes not only the source code, but also 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 [SCH96].

Reliability is seen as the ability to perform as expected under specific conditions for a specified period of time. Software reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment [AIA93]. Software reliability is generally accepted as the key factor in software quality since it quantifies software failures - the most unwanted event that makes software useless or even harmful to the whole system, and malfunctioning software may kill people. As a result, it is regarded as the most important factor contributing to customer satisfaction [LYU97].

There are continuous attempts to try to assess what the reliability of a software package is and to predict what the reliability of software under development will be. The quantitative aspects of these assessments deal with evaluating, characterizing and predicting how well software will operate. Experience has shown that it is extremely difficult to take something as large and complex as modern software and predict with any accuracy how it is going to behave in the field.

Reliability engineering techniques are practiced daily in many engineering disciplines. Civil engineers use it to build bridges and computer hardware engineers use it to design chips and computers. Using a similar concept, we view SRE as the quantitative study of the operational behavior of software based systems with respect to user requirements concerning reliability.

SRE includes:

1. Software reliability measurement, which includes estimation and prediction, with the help of software reliability models.
2. The attributes and metrics of product design, development process, system architecture, software operational environment, and their implications on reliability.
3. The application of this knowledge in specifying and guiding system software architecture, development, testing, acquisition, use, and maintenance [LYU97].

Many software development errors are not detected until the testing phase of the development cycle. Once the software actually goes into execution, there will be valid data on failures, which permits an accurate assessment of what's happening in the software. This was the basis for the development of SRE. In this field, models are developed based on failure data from executing software. The goal is to predict what the software reliability will be if the software is placed in service. Current methods to perform these evaluations must use a variety of software packages to collect, analyze and use the failure data. A Database Management System (DBMS) or Spreadsheet is used for collection. Then the data is loaded into a software reliability tool that estimates the model parameters and uses the model equations to make predictions. Unfortunately, systems currently in use were developed years ago and it is very difficult to get these individual software packages to communicate with one another. The data is often printed from one package and manually loaded into another. This practice is very time consuming and

expensive. An ideal system would perform all of the desired collection, analysis and modeling functions internally, but such a system doesn't exist and would be very costly to develop. An alternative might be to use commonly available commercial software packages that utilize current data transfer methods. The software can then be integrated so that the data can be passed from one application to another to perform the desired functions.

The goal of this thesis is to create an improved method to predict software reliability. This will be accomplished by developing a flexible DBMS to track failures and to integrate the DBMS with a statistical analysis program and a software reliability prediction tool that are used to perform reliability calculations. The data should move from one part of this analysis system to another. Data collected in the database should be able to be retrieved and manipulated by the statistical analysis package, which in turn sends the results to the reliability tool to make predictions. This would eliminate having to take data from one tool and manually input it into another.

B. SCOPE OF THE THESIS

This thesis will cover issues associated with software reliability as they apply to:

1. Software engineering in a distributed environment.
2. Database development.
3. Statistical analysis.
4. Mathematical modeling.
5. Software reliability prediction.

The thesis will conclude with a recommendation for utilizing commercially available software for creating an integrated software reliability prediction system in a distributed environment.

C. METHODOLOGY

The Software Metrics Laboratory in Ingersoll Hall will be used to establish a Software Reliability Prediction System in a distributed environment. Initially, a dedicated database will be created to track software development failures. Students and faculty who are developing programs in the Software Design and Software Engineering courses will track failures on the database.

Software reliability prediction tools and statistical analysis programs will be installed and tested. Modeling equations that are currently used in legacy software will be manually loaded into the modern software, as it is not possible to import them directly from the older programs. The feasibility of using modern software packages for mathematical modeling will be tested.

D. BENEFITS OF THE STUDY

The results of this study will provide software engineers with a cost effective and efficient tool for predicting software reliability based on current, commonly available commercial software. It will serve as an example for other organizations seeking to improve the efficiency and timeliness of their reliability prediction models.

E. SOFTWARE METRICS LAB

The Software Metrics Lab (SML) is located in Ingersoll Hall Room 158, on the Naval Postgraduate School campus. It contains a 13 node Windows NT 4.0 based Local Area Network configured using star topology. The Pentium Pro server is connected to

the workstations and a Hewlett Packard printer through an Addtron Ethernet 10 Base-T hub. The workstations consist of a mix of 486, Pentium and Pentium Pro PCs. RG-45, Category 5 cable provides connectivity between the Server, hub printer and workstations.

II. SOFTWARE RELIABILITY ENGINEERING

A. OVERVIEW

The word *reliability* is applied to describe the level of confidence one has in the ability of a system to perform a specific function or duty. If the system is said to be reliable, the implication is that it will not fail to carry out the assigned function or duty. The field of Reliability Engineering concerns itself with defining reliability quantitatively, developing models to predict the reliability attribute, confirming models through data collection and evaluation, and inferring design and development rules that can improve product reliability. The importance of reliability to system and software design has many facets. Reliability in information systems is increasingly being identified by businesses as a significant factor in the generation of revenue, credibility and customer goodwill.

The ability to predict future behavior of a system accurately requires a model. A model is a representation of reality through a mathematical expression involving one or more measurable variables and parameters. The model is usually configured iteratively as a result of observations followed by analysis of the observation data and synthesis of a model. The model is then used to predict future behavior. The model is either confirmed or modified to better match reality as a result of comparison between actual and predicted outcomes.

Reliability models are structured based on observations and subsequently on controlled tests that provide confirmation of model efficacy. The value of the model, if the model is an accurate replication of the real world, is that we can control real-world performance by modulating or controlling certain parameters identified in the model. Hardware reliability models have led to improved design methods, better testing, better

products, and ultimately to better models. The intention is for software reliability as a discipline to catch up with hardware reliability. While hardware models have been predicated on an observed constant failure rate, software failure rates are anything but constant. Software products are released when failure rates reach some established threshold. Shortly after release there is usually an increase in failure rates due to instability in the maintenance process: personnel learn how to maintain the new software, more users (more execution time), different environments, and different test cases [SCH98]. This scenario is followed by a declining failure rate as faults are removed and as personnel become more skilled in maintaining the software. Each subsequent release follows essentially the same failure rate pattern. This pattern has been observed frequently in software development and in operational use.

Several reliability models have been developed for software. For reasons already discussed, the problem of validating a software reliability model is difficult. Nevertheless, several of the available models, if validated with data from the intended application, can produce useful results. The most useful applications are in testing the software product. Reliability models can be effective in predicting the time to failure of a software product; in estimating the number of faults remaining in a given software product, in estimating the number of failures that will have to occur and faults located and repaired in order to reach a given time to failure; and in estimating the number of hours of CPU execution time required to achieve a given number of remaining failures. Estimates of these software-related reliability attributes are valuable in making decisions regarding test duration, test resources, risk, and product release.

There is an intuitive connection between cost and the requirement for a software product with high reliability. The increased cost comes from the application of formal quality assurance, additional reviews, more formal documentation, and extended formal testing. It will require extensive data gathering and thoughtful analysis for a development team to establish an accurate, quantifiable link between reliability and specific quality assurance measures.

B. THE SOFTWARE RELIABILITY ENGINEERING PROGRAM

The implementation of a complete SRE program is beyond the scope of this thesis. Knowledge and understanding of what this implementation entails, however, is essential to the proper design and development of the system. A successful software reliability program consists of several key elements. These include: definitions of severity levels of failures, data collection procedures to obtain the necessary data, reliability requirements, reliability measurements to meet those requirements, applications of reliability predictions, interpretation of model predictions, and user feedback for model improvements. Failure data is preferred to defect data for reliability assessment and reliability prediction because the former is 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 these time records. Defect data are used more for administrative control to ensure that defects have been resolved than as data for reliability assessment and prediction [SCH96].

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 execution of the software, identification of software failures, and conversion of raw failure data into reliability predictions through the use of the selected model. With these phases being the stated objective, the following steps, which were taken from *MCTSSA Software Reliability Handbook, Volume I, Software Reliability Process and Modeling for a Single Process*, by Dr. Norman Schneidewind, 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.

Step 1: State the Reliability Requirement

In this step, management decides on the conditions that must be fulfilled for the software to be considered reliable. An example may be "The product will have no software failures that would result in loss of life, loss of mission, or cancellation of mission."

Step 2: Establish a Measurement Framework

The organization should develop a measurement plan to determine the likely reliability of the software. This plan should include *direct* and *indirect* measures of quality. *Level 1* will be the direct measurements. These are the metrics that can be captured directly by the use of a wall clock and the continuous running of the software, such as time to next failure, number of failures likely to occur within a given time frame. *Level 2* is an indirect measurement, one level removed from the direct measurement. At this level a report is written every time a discrepancy is observed between requirements

and implementation. Most of these reports are derived from static analysis, therefore it would not be possible to directly predict the time to next failure. Finally, *Level 3* is an indirect measurement two levels removed from the direct measurement. These are the basic attributes (or metrics) of the software itself. How many lines of code does it have? How complicated are the routines in the program? Traditionally, the more complicated the coding, the more likely faults will appear.

Level 1 measurements are the most accurate representations of reliability, however they cannot be collected until the software is tested. 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.

Step 3: Data Collection

Without data, reliability predictions cannot be made. For this data collection, a DBMS is essential to manipulate large amounts of failure and metrics data. The DBMS allows data sorting and query functions necessary for thorough analyses and reporting. The minimum data required for the analysis should include *date, time of failure, type of failure, and severity of failure*. Once collected, the data can be interpreted statistically and used in a reliability prediction model. For each software system there should be a brief description of its purpose and functions.

Step 4: Establish Problem Severity Levels.

A consistent way of describing faults and failures will need to be established. This will result in better analysis and classification of failures. Some recommended severity levels include:

Level 1. Loss of life, loss of mission.

Level 2. Degradation of performance.

Level 3. Operator annoyance.

Level 4. System ok, but documentation in error.

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

Step 5: Estimate Model Parameters

Once a model has been chosen, model parameters must be estimated. An example is that of *the Marine Corps Tactical Systems Support Activity* (MCTSSA) Volume I through IV, which was developed at the Naval Postgraduate School using the *Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS)* modeling tool [FAR93]. Three parameters are used in the model: α, represents the failure rate at the beginning of the testing interval "s"; β, is the failure rate constant (i.e., a measure of how fast the failure rate decays); and s is the starting interval for using observed failure data in parameter estimation.

Step 6: Select the Optimal set of Failure Data.

Using the parameter "s", determined using a Mean Square Error criterion as part of *SMERFS*, this stage selects the subset of failure data that will give the best parameter estimates and the most accurate predictions. A subset is used because both the software process and product change over time. Old data may no longer represent the current state of the product and may not be applicable for reliability prediction as more recent data.

Step 7: Identify the Operational Profile.

The operational profile describes the system's environment. It is usually discussed in terms of number of nodes, frequency of use of a particular node, and the frequency of

function execution. It includes the input variables, the functional environment of the program, and the output variable. A failure would be a departure of the output variable from what it is expected to be [MUS87].

Step 8: Make Reliability Predictions.

This step is the key to predicting the reliability of the software under evaluation. The following is a list of predictions that could result from testing and analysis:

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 Specified Remaining Failures.
6. Test Time to Achieve Specified Remaining Failures.
7. Test Time Needed to Obtain "Fault Free" Software.
8. 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 $\leq 20\%$).

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 augmented with the model's predictions. 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 prediction provides additional confidence that the software can be accepted as fulfilling contractual obligations. If the software is predicted to fall short of the desired goals, further discussion may be needed in addition to further testing and evaluation before the software can be accepted.

C. THE CURRENT RELIABILITY PREDICTION PROCESS

This thesis is primarily concerned with steps 3, 5, 6 and 8 in section B above. In these steps, the failure data is collected, the model parameters α (the failure rate at the beginning of the testing interval "s"), β (the failure rate per failure , and "s" (the first interval used in parameter estimation) are derived, the optimal set of failure data (optimal "s") is determined, and reliability predictions are made.

Data is collected as the software is tested prior to release, or as failures occur subsequent to release. The software failures are annotated in a spreadsheet or database and, after a period of time, sent to an analyst who processes the raw data and makes a reliability analysis. The process of analysis can be performed in a number of ways using a number of different models. We will describe the process as it exists using SMERFS

and the Schneidewind model. The Schneidewind model itself will be described in detail in the next section.

Failures are sorted according to the length of time the software has been operating since the test began or since it was released. A standard time period (interval) is determined, such as 30 days, and then each failure is categorized according to the number of intervals that the software was in operation prior to failure. The number of failures per interval is determined, and this information is then inputted into SMERFS for analysis using the Schneidewind model.

Processing the interval data in SMERFS using the Schneidewind model, will produce the optimal set of failure data: optimal "s", α, and β. These are the parameters required to make reliability predictions.

An example of this process would be a software program used for an industrial process. Suppose this software is put into operation on 1 January 1997, and the system administrator records all software failures in a spreadsheet. A small section of which is shown in Figure 2.1.

System Introduced	Reference Number	Days from Build	Severity	Failure Date	Build	Module in Error #1	Module in Error #2	Module in Error #3
RELEASE 1.0	102	339		12/06/97	4.01	GSR		
	743	257		09/15/97	4.01	CDK		
Released	159	289		10/17/97	4.04	SBI		
01/01/97	158	255		09/13/97	4.02	PPF		
	151	244		09/02/97	4.04	PPK		
	600	250		09/08/97	4.04	PPN	PPK	
	674	201		07/21/97	4.04	CGZ	GKL	GPN
	269	353		12/20/97	4.05	PPN		
	676	345		12/12/97	4.04	GJA		
	645	374	3	01/10/98	4.07	PPN		

Figure 2.1: Sample Failure Log

On 30 June 1998, the administrator sends the software manufacturer 18 months worth of data showing all failures. An analyst at the manufacturers' headquarters takes the spreadsheet and determines the number of failures to occur per interval by dividing the value in the "Days from Build" column by 30 (for a 30 day interval). Now he has information on the number of failures per interval over the course of 18 intervals. This is the data used in SMERFS to determine the optimal "s", and its' associated α and β. These parameters are then used to make a reliability prediction.

The inherent complexity of the process described above is magnified by the system currently employed to achieve the goal. Software packages, such as SMERFs Version 5 are used to perform mathematical modeling and statistical analysis. While this product is easy to install and run in networks under Windows 95 and Windows NT, it was written for DOS and thus lacks a GUI and a copy and paste feature for easy manipulation of data between programs.

D. THE SCHNEIDEWIND MODEL

Dr. Norman Schneidewind of the Naval Postgraduate School developed the Schneidewind model. It was first described in *Analysis of Error Processes in Computer Software* (IEEE Computer Society *Proceedings of the International Conference on Reliable Software,* 1975) . Dr. Schneidewind observed that failure rates tend to change over time, therefore using data on all failures since the product first began testing would tend to lead to false predictions of its present reliability. The reason for this is that the old data may not be as representative of the current and future failure process as recent data (i.e., the reliability of the software may change over time). More specifically, changes in reliability trends could be caused by dependency of faults (i.e., some faults mask others)

and variation in the time between failure occurrence and fault correction [SCH93]. This concept is only used when there has been a significant change in the failure rate over time. If the rate has been steady over a long period, most or all of the data should be used. The difficulty in using this technique is in selecting the proper length of time over which to evaluate the software failure rate. The Schneidewind Model provides a built-in method for choosing the optimal subset of failure data. Actually, three separate models were developed to reflect the importance of the data as functions of time. The data used are the number of failures per unit of time where all of the time periods are of the same length. Suppose there are n units of time, all of some fixed length; then the three forms of the model are:

Method 1 Use all of the failure counts from interval 1 through t. This method is used if it is assumed that all of the historical failure counts from 1 through t are representative of the future failure process.

Method 2 Use failure counts only in the intervals s through t. This method is used if it is assumed that only the historical failure counts from s through t are representative of the future failure process.

Method 3 Use the cumulative failure count in the interval 1 through s-1 and individual failure counts in the intervals s through t. This method is used if it is assumed that the historical cumulative failure count from 1 through s-1 and the individual failure counts from s through t are representative of the future failure process. This method is intermediate to Method 1, which uses all the data, and Method 2, which discards "old" data.

The basic assumptions of the Schneidewind Model are as follows:

1. The cumulative number of failures by time t, M(t), follows a Poisson process with mean value function $\mu(t)$. The mean value function is such that the expected number of fault occurrences for any time period is proportional to the expected number of undetected faults at that time. It is also assumed to be a bounded, nondecreasing function of time with $\lim_{t \to \infty} \mu(t) = a/ß < 4$; for some constants a, ß > 0 (i.e., it is a finite failure model).
2. The failure intensity function is assumed to be an exponentially decreasing function of time. The failure intensity function $\lambda(t)$ is taken to be of the form $\lambda(t) = \alpha \exp(-\beta t)$. Therefore, large β implies a small failure rate, small β implies a large one. Moreover, we see α is the initial failure rate at time t = s.
3. The number of faults (f_i) detected in each of the respective intervals is independent.
4. The fault correction rate is proportional to the number of faults to be corrected.
5. The intervals over which the software is observed are all taken to be of the same length, that is, $t_i = il$, for $i = 1, \ldots, n$ and l being some positive constant.

Because the Schneidewind model is only one of the four models, and the only failure count model, recommended in the Recommended Practice for Software Reliability, R-013-1992, *American National Standards Institute/American Institute of Aeronautics and Astronautics*, and has been used for Shuttle and DoD reliability applications, it will be used as the example model in this research.

II. SOFTWARE METRICS

A. OVERVIEW

Reliability models are not the only methods used for estimating software reliability. Software metrics can be used long before the product-testing phase to estimate the reliability of the software. Metrics are characteristics of the software related to its size, documentation, and structure. Examples of these metrics are: the number of lines of code, the number of comments, and the number of different paths of execution [LEW92]. Metrics are *static* predictors, in that, the fault is discovered and documented by observing these characteristics rather than observing how the software performs in a real-time. The metrics are collected from design documentation, code listings and metric code analyzers which analyze the source code [MAYR90].

Software quality management techniques need to be implemented from the outset of the software development process to ensure that fewer faults are introduced into the software. These practices are assumed to produce software with fewer faults, reduced development time, and a better product for the end-user. Metrics are used to quantify characteristics such as complexity, testability and portability. Metrics assist the software development manager to assess whether project goals are being achieved, long before the testing phase begins.

Metrics need to be validated with respect to the specific application. Using validated metrics, the decision-maker has increased confidence that the metrics chosen measure the right quality factor [IEE93]. Metrics target many sectors of a software project. A *process metric* is used to quantify attributes of the development process and environment

[MAYR90]. Process metrics measure resources, the effects of interim changes applied to a project, and determine compliance with planned cost and time estimates.

A *product metric* is used to measure some aspect of product quality, such as the quality of a deliverable. In a well-reasoned software development plan, the two metrics are difficult to differentiate because each metric supports the other, to produce quality software [MAYR90].

Metrics can be used as management information tools during software development. These tools act as predictors of future size, cost and complexity of the design. Metrics need to be used throughout the development life cycle of the product for estimation and evaluation of resources that will be required in future phases. They can be used to monitor stability of the product through maintenance and enhancement [SCH98]. To be useful, metrics need to be reported in a timely fashion and must be user-friendly to the software developer.

As measures of software quality, metrics should be linked to support the functions of quality assessment, control and prediction [SCH92]. As a means of quality assessment, metrics can act as thresholds and can be used as either acceptance standards by the software acquisition manager or as an indicator of potential problems in the code during software validation and verification, prior to operational implementation. The metrics used should be those which are associated with the quality requirements of the software project. Any organization can apply our approach because the methods are not domain specific. Different organizations may obtain different numerical results and trends than the ones we obtained for the Shuttle because many metrics are meaningful only in the context of a specific software system.

B. PRESENT SOFTWARE METRICS SYSTEM

This study is based on a software metrics system designed by Dr. Norman Schneidewind at the Naval Postgraduate School. The present system is divided into two sections. The first of these includes formatting the data in preparation for analysis . The second involves various statistical analyses that support software quality control, assessment, and prediction. Although the formatting of data is always required, many other manual manipulations are required during the current process. Many of these would not be necessary with the use of contemporary tools. Failure data is used in the *Statistical Modeling and Estimation of Reliability Functions for Software* (SMERFS), as discussed in the reliability section, to provide the researcher with the required parameters of alpha (α), beta (β) and a starting time interval (s). Once these parameters are obtained, Statgraphics, a commercial, off the shelf statistical analysis package, is used to make various reliability predictions. Metrics data is also used in Statgraphics in a "four-step" process designed to optimize the number of metrics used in the quality calculations [SCN97]. Optimization reduces the number of metrics needed to make predictions concerning the software. Many researchers are of the mindset that "more is better" and try to include many metrics in calculations but any metrics overlap in measuring quality. Using more than necessary is costly.

The four-step process is outlined below [SCN97].

Step 1: The first step in the process is to use the Krushal-Wall (K-W) [LAR86] test to place the metrics recorded against modules in an ordinal series. This test essentially ranks the metrics in order of their ability to predict software quality. This is a screening test to separate the better predictors from the less promising ones. In addition, the K-W

test involves a lot less work that Step 2, as performed in Statgraphics. Therefore, rather than perform Step 2 on all metrics, K-W is performed on all metrics and only the most promising ones are promoted to Step 2.

Step 2: The next step is to perform the Inverse Kolomogorov-Smirnov test [SPLUS97] on the top three or four metrics, as ranked by Step 1 [SCN97]. This process inputs the cumulative distribution function (CDF) of two sets of data in vector form (e.g. statement and comment counts) and plots them against each other (Figure 3.1).

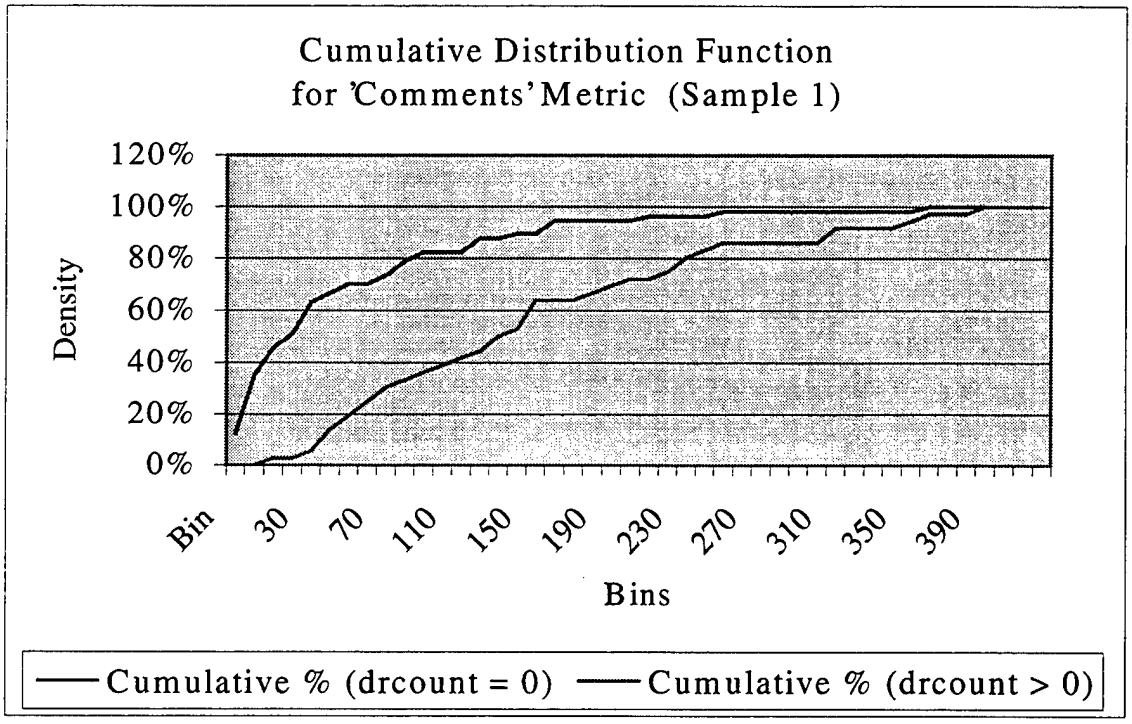

Figure 3.1: Cumulative Distribution Function for 'Comments' Metric

The modules are examined for discrepancy reports in order to separate them into two sets. A module that has no discrepancies is placed in set A. A module with at least one discrepancy report is placed into set B. A discrepancy report is a report of a discrepancy between requirements and implementation. . In this evaluation, discrepancy reports are labeled, "drcount" [SCH94].

Once the two sets have been determined, the CDFs are computed , one for drcount = 0 and the other for drcount > 0. The analyst chooses the appropriate value for the discrepancy report count, depending on the priority concern. In this case, zero was chosen. Zero discrepancies indicate this is a safety critical system, Level 1 severity [SCHN97].

From these computations, the analyst can find the value of the metric that corresponds to the point where the distance between the CDFs is maximum. This value provides the critical value for a particular metric. In general, several metrics are used in the form of Boolean discriminant functions in the analysis. If the value of one or more metrics on a module is greater than its critical value, this indicates the module needs to be investigated for possible quality problems.

Step 3: Once the critical value of each metric has been determined, a contingency table is constructed to categorize the quality of each module [LAR86]. A contingency table separates the data into specific categories. The rows are distinguished by the drcounts (drcounts = 0 vs. drcount > 0) and the columns are distinguished by using the Boolean discriminant functions. This is depicted in Figure 3.2.

	(Comments <= 38) AND (Stmts. <= 26)	(Comments > 38) OR (Stmts. > 26) (Type II errors)
Drcount = 0	**I**	**II**
Drcount > 0	(Type I errors) **III**	**IV**

Figure 3.2: Contingency Table

Particular attention is paid to Quadrant III from Figure 3.2. This quadrant depicts the modules that meet the criteria for acceptance yet still contain discrepancies. Since the metrics' values are less than or equal to the critical values, the module would not be identified as problematic. However, problems have occurred, since there are discrepancy reports logged against the module. Reduction of these "Type I errors" is the focus of the process " [SCN97].

Step 4: The fourth step in the process allows the researcher to decide on the number of metrics that provides optimal predictive capability when the cost of inspection is considered, . If too many metrics are used, the process could be costly. Collection of unnecessary data increases the processing and inspection time, which in turn costs money. Therefore, by knowing when the number of metrics in the process is optimized, the researcher can cease adding metrics and concentrate on analyzing the data.

It can be seen from the description above that the current process is slow and labor intensive. There are a number of interfaces between different types of software and between the user and the process. Data must be manipulated and transferred from one

application to the next in order to derive the necessary information. Revising the system with contemporary tools will reduce the many transitions and simplify the process.

C. FUTURE SOFTWARE METRICS SYSTEM

The proposed revision for the system is an update in tools only. The logic behind the process is sound. There are two goals associated with the system revision. The goals are:

1. To reduce or eliminate manual manipulations or transitions between applications.
2. To establish this metrics/reliability model on the distributed system in the Software Metrics Laboratory in the Ingersoll Building, room 158.

The revision will allow a user with minimal background in the process, to extract useful, timely information concerning a development project. The four-step method described above will remain the basis for the calculations. The exception to the revision is that the team will not upgrade the SMERFS software. SMERFS is a self-contained application written for DOS. It would require extensive modification that is beyond the scope of this work. There is a Windows based version of SMERFS that has been developed and its integration into this process may be the topic for further research by future students.

The software package chosen for the revision of the software metrics and reliability system is the *Microsoft Office 97* suite. The use of a single suite of software, which has compatible components like *Excel* and *Access*, will allow easier integration of the data and analysis.

The raw data is received from the Shuttle flight software developer in an Excel spreadsheet. This can be input to the database by formatting and importing the complete

file. The relational database has the advantage of storing data in a form readily recognizable to the user. The data is stored in tables and relationships among rows of the tables are visible in the data. The data, as well as the relationships between the data, is stored in the DBMS [IDG97].

Once the data is in the DBMS, it can be easily ported to an imbedded Excel worksheet. The worksheet will contain the necessary equations and macros to perform as many of the "four-step" calculations from *Statgraphics* as possible. However, a completely self-contained automated system is not attainable at this time, because Excel does not include all the necessary statistical tools required for detailed statistical calculations. Therefore, SMERFS and a Windows compatible statistical package will still be required for the system to operate properly.

Even with these limitations, the ability to access data and reports, make queries, and reduce the number of transitions between applications, using Excel, will significantly decrease the time required to extract usable information from the system.

IV. CREATING A NEW SYSTEM

A. INTRODUCTION

As stated in Chapter I, the goal of this thesis was to create an improved method of using tools to predict software reliability. This was to be accomplished by developing a flexible Database Management System to track failures and to integrate the Database Management System with a statistical analysis program and a software reliability prediction tool that are used to predict reliability. The data should move from one part of this analysis system to another, data collected in the database should be able to be retrieved and manipulated by the statistical analysis package which in turn sends the results to the reliability prediction software to provide prediction capability. This would avoid having to take data from one tool and manually input it into another.

In order to achieve this goal, several commercial statistical analysis and mathematical software packages were considered. The criteria used for the evaluation were:

1. **Cost**. Includes initial purchase price and the cost of maintaining upgrades.
2. **Suitability**. Does the software do the job required?
3. **Availability**. Is the software widely used and available to most users?
4. **Acceptability**. Will users be willing to use the software if it is available?
5. **Ease of Use**. Time spent in learning and using the software.

The software packages that were considered for use were **S-Plus**, a statistical analysis package; **MATLAB** and **Mathematica**, mathematical tools; and **Microsoft Excel**, a common spreadsheet program that has numerous statistical analysis and mathematical functions. *S-Plus*, *MATLAB*, and *Mathematica* are all very capable tools in

general, but are not capable of making all of the mathematical and statistical calculations that are required for metrics and reliability applications. All of the programs have current, windows-based, interfaces that support Cut & Paste and Import/Export functions. Unfortunately, these programs also failed to meet the remaining criteria. Each of these programs is costly to purchase and requires a great deal of training and practice before most users become proficient. They are not readily available at most sites, meaning most of the target users would have to purchase them and learn to use them. Therefore, the target users will not readily accept them.

The software package that met most of the criteria was *Microsoft Excel*. *Excel* can be pre-installed on desktop computers. It was uncertain, however, whether *Excel* was suitable to the task. The main research question became "Can *Excel* handle these complex equations and routines?" Surprisingly, the answer was largely "yes". *Excel* proved that it was able to perform many of the software reliability estimation and software metrics calculations. A summary of the conversion to *Excel* follows in subsequent sections of this chapter.

B. SOFTWARE RELIABILITY

The current state of the software reliability prediction process was outlined in Chapter II. The prime software used in these calculations is SMERFS 5.0, a DOS based application which can operate on a network under Windows; however, it does not have cut and paste and import/export capability. However, a new version SMERFS is now available in Beta form which is Windows-based and contains both hardware and software reliability models. A statistical program, Statgraphics that has an equation editor has also been heavily used. Although both SMERFS and Statgraphics are useful, they have the

disadvantage of not being used by a large number of people who may be interested in doing reliability and metrics analysis. Therefore it is difficult for a researcher to transfer technology to practitioners with these tools. *Excel* supports all of the needed functions except reliability model parameter estimation. Table 4.1 demonstrates the ability of *Excel* to calculate Time to Next Failure and Remaining Failures estimations.

OI	s_f	s_t	alpha	beta	t	$X_{s,t}$	F_t	$T_F(t)$	r(t)	
F		5	1.3593	0.18285	13	6	1	6.54	1.43	
G		1	0.46009	0.01026	14	6	1	2.54	38.84	
			$T_F(t)$ =	((LN(alpha/(alpha-(beta*(Xst+Ft))))))/beta)-(t-st+1)						
			r(t) =	((alpha/beta)-Xst)						

Table 4.1: Time to Next failure and Remaining Failures Estimations

In Table 4.1, s_t, α and β are values estimated from SMERFS that are used to calculate Time to Next Failure, $T_F(t) = [(\log[\alpha/(\alpha-\beta(X_{s,t}+F_t))])/\beta])-(t-s+1)$, and Remaining Failures, $r(t) = (\alpha/\beta) - X_{s,t}$.

Where: $X_{s,t}$ = observed failure count in the range [s,t].

F_t = given number of failures to occur after interval t.

t = last interval of observed failure data.

s = starting interval for using observed failure data.

As is demonstrated in the table 4.1, when s, α and β are obtained from SMERFS, these equations were readily implemented in *Excel*. The limit in the use of *Excel* is reached, however, when the attempt was made to estimate s, α and β. The algorithms and calculations required to estimate these parameters are too complex to be performed in a spreadsheet; they must be obtained from SMERFS. The parameter estimation

calculations from the Schneidewind Model that are implemented in SMERFS are listed below.

$$\text{Log } L = X_t[\log X_t - 1 - \log(1 - \exp(-\beta t))] + X_{s-1}[\log(1 - \exp(-\beta(s-1)))]$$

$$+ X_{s,t}[\log(1 - \exp(-\beta))] - \beta \sum_{k=0}^{t-s} (s + k - 1)\, x_{s+k}.$$

Where: L = Likelihood Function

X_t = observed failure count in the range [1,t]

X_{s+k} = number of observed failures in interval s+k

k = a given interval

This function is used to derive the equations for estimating α and β for each of the three methods in the Schneidewind model. In the equations that follow, α and β are *estimates* of the population parameters.

Method 1 uses all of the failure counts from interval 1 through t (s = 1). The following two equations are used to estimate β and α, respectively.

$$\frac{1}{\exp(\beta) - 1} - \frac{t}{\exp(\beta t) - 1} = \sum_{k=0}^{t-1} k \frac{x_{k+1}}{X_t}$$

$$\alpha = \frac{\beta X_t}{1 - \exp(-\beta t)}$$

Method 2 uses failure counts only in the intervals s through t ($1 \leq s \leq t$). The following two equations are used to estimate β and α, respectively.

$$\frac{1}{\exp(\beta) - 1} - \frac{t - s + 1}{\exp(\beta(t - s + 1)) - 1} = \sum_{k=0}^{t-s} k \frac{x_{s+k}}{X_{s,t}}$$

$$\alpha = \frac{\beta X_{s,t}}{1 - \exp(-\beta(t - s + 1))}$$

Method 3 uses the cumulative failure count in the interval 1 through s-1 and individual failure counts in the interval s through t ($2 \leq s \leq t$). The following two equations are used to estimate β and α, respectively.

$$\frac{(s-1)X_{s-1}}{\exp(\beta(s-1))-1} - \frac{X_{s,t}}{\exp(\beta)-1} - \frac{tX_t}{\exp(\beta t)-1} = \sum_{k=0}^{t-s}(s+k-1)x_{s+k}$$

$$\alpha = \frac{\beta X_t}{1-\exp(-\beta t)}$$

Examination of these equations shows that they require a number of iterative calculations and summing through several ranges of data to find the optimal set of failure data, and the estimates for α and β for that set. *Excel* is unable to perform these calculations because it was not designed for them.

Assuming that it takes a relatively long period of time for failure rates to change significantly, it is possible to use a particular set of values for s, α, and β repeatedly and still get reliable estimates with subsequent failure data. For example, suppose that the software developers who had tested OI G for one interval (30 days) continued testing the software for an additional 3 months ($\cong$ 3 intervals). It is reasonable to assume that the failure rate will not have changed significantly during that time and they could still use the same values for s, α and β to get accurate reliability estimates without having to re-estimate the parameters using SMERFS. Therefore, the decision was made that, until a better alternative is available, SMERFS will continue to be used for making these estimates and *Excel* will be used to make the Time to Next Failure, Remaining Failures, and other reliability estimates.

C. METRICS

One problem that was essential to overcome in designing the new system was the conversion of an extensive amount of legacy programming source code into a form that is usable in the new system. Most of this code was written in an older DOS-based program, Statgraphics. This source code is readable, but has proven to be non-exportable to the newer, Windows-based versions of the software, because that version did not retain the equation editor.

The legacy systems contain very capable mathematical and statistical algorithms. Many mathematical equations required by the researcher can be constructed and applied using the specific built-in functions [SCH94]. The applications were designed for advanced users with mathematical, statistical, and programming experience. This type of design is not desirable since it limits the type of user who would regularly employ such an application to those with advanced knowledge. This eliminates the low-level user who needs to process data but does not have the requisite skills to operate complex programs.

Even simple tasks, such as saving screens and printing screens to a file can be an intensive endeavor. *Statgraphics* for example, will not allow the user to scroll through more than one screen of data at a time. The user must perform several keystrokes in order to get the next screen of data [CON6-98]. Each page must be specifically recalled and displayed. In order to print screen data, each page has to be saved individually to a file. This requires five keystrokes per page of data saved. To save all 1397 modules of Shuttle data from *Statgraphics* to a file required individual screens, containing approximately 16 lines of data, to be recalled. This screen of data was then saved to a file and the process was repeated for the next 16 lines. This required 87 screens of saved data to be saved,

436 individual key stokes and 30 minutes of time. The same task in *Excel* required the "cut" and "paste" functions, five keystrokes, one worksheet of data, and about 15 seconds of time. Performing even the simplest of tasks can significantly increase the workload for the user of legacy systems. Another benefit of using *Excel* is that there is no need to transfer the data from one application to another. Since the data is already in an *Excel* spreadsheet when the analyst receives it, this data can be formatted and sorted in the received spreadsheet. In this way, the sample data for the actual calculations is available directly from the inputted raw data. This reduces the number of possible errors that may be introduced by moving, sorting or porting the data from one application to another.

1. The Updated Process

The updated process using *Excel* can be subdivided into three areas:

a) Random selection of sample data.

b) Extraction of the metric values for each sample.

c) Applying the metrics equations to the randomly selected data.

2. Random Selection of Data

The proper extraction of a random sample from the population is essential to produce data that is unbiased. The *Excel* function RANDOMBETWEEN() was used to generate a sample and minimize bias. This function generates any number of samples (N), from the population using a given starting and ending value. In this study, N = 100 samples were taken from the total population of 1397 software modules.

This function requires the user to input a range of values from which the requested number of samples is taken. In this case one hundred numbers were chosen between 1 and 1397 inclusive.

3. Attribute Value Extraction

Once a sample set has been chosen, the values for each particular metric that the analyst is testing must be extracted from the data.

Using the metric "statements" as an example, the data is filtered such that only rows that contain values in the 'statements' column which are greater than zero are displayed. This eliminates the possibility of attempting to divide by zero when performing certain metrics calculations.

After filtering, it is important to uniquely link a specific module with its metric's value from the given range (1 to 1397). The data extraction process is then able to link the specific row (software module) to the metric data contained in that row. The use of the RANDOMBETWEEN() function simply selects one hundred random numbers from a set of values [LIE97]. It does not extract the data contained in that particular row of the table. This extraction is accomplished using the VLOOKUP() function.

VLOOKUP() is an *Excel* function that extracts a value in a particular cell from a table [LIE97]. It uses three arguments to ascertain this cell value. The first argument is the random number vector generated using the RANDOMBETWEEN() function. The second is the table of interest from which the information is to be extracted. The last is the particular column in the table of interest from which the data is to be removed. The random number vector acts as the row index for the function. This value points to a specific row and the column argument points to a specific column number in the table.

Excel then returns the value of the cell at the intersection of the row and column. Using the VLOOKUP() function, the researcher can find a particular metric value and link the value directly to a software module. An example of the extracted table data is shown in Table 4.2. Table 4.3 shows the data as it is displayed on the worksheet before extraction. The reader should note the correspondence of "Index #" 20 and 22 and the values extracted for the random sample set.

RANDOM INDEX # FOR SAMPLE 1	COMMENT SET SAMPLE 1	DRCOUNT SET SAMPLE 1
20	58	3
22	19	0
35	13	0
64	0	0
66	0	0
68	0	0
73	114	3
83	0	0
89	0	0
90	0	0

Table 4.2: Random Index Vectors with Corresponding Metrics Values

INDEX NUMBER	MODULE NAME	NUMBER OF COMMENTS	DRCOUNT	NUMBER OF STMTS
20	ASCT	58	3	68
21	ASGC	46	4	65
22	ASHR	19	0	64
23	ASIS	110	1	20
24	ASLT	226	4	317

Table 4.3: Sample of Raw Metric Data before Extraction

The random number vector, generated using the RANDOMBETWEEN() function is shown in the column labeled "Random Index # for Sample 1". Columns labeled "Comment Set Sample 1" and "drcount Set Sample 1" are examples of the values extracted using the VLOOKUP() function.

The first step in the calculation phase is to build the Inverse Kolmogorov-Smirnov function. It should be noted that the Kruskal-Wallis (K-W) test (Step 1 from Chapter III) is not used in these calculations. The K-W test was used to determine an approximate initial rank for the metrics because the previous method of calculating the Inverse Kolmogorov-Smirnov (IKS) was time consuming. The K-W test was used first to identify a few metrics with the highest approximate ranks with respect to discriminating between high and low quality software. Then the IKS procedure was applied to these metrics to obtain the exact ranking. The Inverse Kolmogorov-Smirnov in Excel negates the need for this test because it provides an exact ranking.

The reader will recall that the Inverse Kolmogorov-Smirnov function provides the metric critical values [SCH92] from which a contingency table can be created. The metric critical values are used as threshold values for the columns of the contingency table. The critical value is determined by finding the metric value that corresponds to the maximum vertical separation between the two cumulative distribution functions (CDF) from specific sets of data. The problem of creating the two distribution functions proved to be relatively simple because *Excel* has a built-in histogram function [LIE97]. The next and more significant problem was the selection of "bin" size.

The bins can be set in one of two ways. Bins can "hold" a range of values or a single value. The values in this case are the actual metric counts taken from each module. An example taken from module eight (AIDD) shows a metric count for "Comments" that equals 53. This indicates there are 53 lines of comment in that module.

The bin size for this tool was set to "hold" a single value. The histogram function was used and displays the cumulative percentage for single bins. A question arises as to

the required bin size due to the large variation of individual counts of the metrics used. In Sample 1, the largest value of comments was 2140 and the largest value of statements was 367. This may lead one to believe that different size bins are required to contain such wide-ranging values. This is not the case however, since an essential step in the process is to subtract the cumulative distribution functions in order to determine the maximum value. A simple solution is to use a bin size for each sample (e.g. Sample 1, Sample 2 etc.) equal to one. Using a larger bin size created erroneous conclusions when compared to previously verified results. In Sample 1 for Comments, bin 38 was identified as the location of the largest difference in CDFs. A histogram was constructed using a bin size equal to one.

The sample data for each metric was divided into two sets. The sets, as discussed in Chapter III, are for 'drcount equal to zero' and for 'drcount greater than zero'. Each set is individually fed into the histogram function. The cumulative probability is returned for each data set. The histogram is displayed in tabular form and can also be displayed graphically. A dialogue box allows the user to make selections to receive specific results. In this case "Cumulative Percentage" and "Chart Output" were selected from the dialogue box. The chart was helpful for visualization but not for calculations and was only used for initial trials. The tabular form made the calculations straightforward. An example is shown below for Sample 1, Comments, in Table 4.4. In this example the bin size = 1. The reader should notice that the differences in CDFs for bin 38, 39 and 40 are the same value. In this study an assumption was made that in cases where equal values occur, the metric value of the first bin in the series will be used as the critical value.

BIN	FREQUENCY	CUMULATIVE % FOR SAMPLE 1 DRCOUNT = 0	BIN	CUMULATIVE % FOR SAMPLE 1 DRCOUNT > 0	ABSOLUTE DIFFERENCE OF CUMULATIVE %'S
30	0	50.88%	30	2.33%	48.55%
31	1	52.63%	31	2.33%	50.31%
32	0	52.63%	32	2.33%	50.31%
33	1	54.39%	33	2.33%	52.06%
34	1	56.14%	34	2.33%	53.81%
35	1	57.89%	35	2.33%	55.57%
36	0	57.89%	36	2.33%	55.57%
37	1	59.65%	37	2.33%	57.32%
38	2	63.16%	38	4.65%	58.51%
39	0	63.16%	39	4.65%	58.51%
40	0	63.16%	40	4.65%	58.51%

Table 4.4: Absolute Difference Between Sample CDFs

By using the tabular form, the columns labeled the "Cumulative % for Sample 1" from each set (drcount = 0 and drcount > 0) are subtracted, using the 'absolute difference' function in *Excel*. This provides a single column labeled, "Absolute Difference of Cumulative %'s" in Table 4.4, from which the max difference can easily be extracted. This difference is ascertained by highlighting both the column that contains the absolute difference of the CDFs and the associated bin column. Then using the auto-sort function both columns are sorted using the leftmost column as a reference [LIE97]. If the "bin" column is "pasted" to the right of the absolute difference column, the columns are linked and sorted by CDF, not bin size. It is easier to sort in descending order since this method brings the largest value to the top of the column. The user then can easily read the bin value associated with the largest difference in CDFs, the value needed for the metrics calculations that are to follow. In Sample 1, comments, the bin with the largest difference in CDFs occurred in bin 38. In Sample 1, statements, the bin with the largest difference in CDFs was bin 26.

4. Applying Metrics Equations

The metrics equations discussed in this section will use the critical value (i.e. bin), determined from the process above. Using this critical value, each metric value can be evaluated and placed in a particular quadrant of a contingency table. The values from each quadrant of this table are used in the metrics equations [SCH97]. Project managers can use these equations to determine the level of quality in a particular software module. The equations are listed in Table 4.5 below.

EQUATION	TITLE	COMMENTS
N	Sample size	In this case 100
$n2 = C_{21} + C_{22}$	Modules containing an error	Drcount > 0
$P1(\%) = C_{21}/ N$	Proportion Type I	Proportion of Type I misclassifications
$P2(\%) = C_{12}/ N$	Proportion Type II	Proportion of Type II misclassifications
$P12 = (C_{21} + C_{12})/ N$	Combined proportion	Proportion of Type I and Type II misclassifications
$LQC(\%) = C_{22}/n2$	Low Quality Software	Proportion of low quality software correctly classified
$RF = \Sigma F_i\ (F_i > F_c) \wedge$ $(M_{i1} \leq M_{c1}) \ldots \wedge$ $(M_{ij} \leq M_{cj}) \ldots \wedge$ $(M_{im} \leq M_{cm})$	Remaining Factor Sum of F_i not caught by inspection	Sum i from 1 to N for all factors
RFD =	Remaining Factor Density	Density of the Remaining Factors
$TF = \Sigma Fi$	Total Factors	TF is the total F_i prior to inspection summed from I = 1 to N
$RFP(\%) = RF/TF$	Remaining Factor Proportion	Proportion of Remaining Factors
RFM = COUNT FOR ($(Fi > 0) \wedge (M_{i1} \leq M_{c1}) \ldots \wedge$ $(M_{ij} \leq M_{cj}) \ldots \wedge$ $(M_{im} \leq M_{cm})$)	Remaining Factor Modules	Count of module that have a Remaining Factor in them
$RMP(\%) = RMF/ N$	Proportion Remaining Modules	Proportion of module remaining where $F_i > F_c$
$I(\%) = (C_{12}+C_{22})/ N$	Inspection	Proportion of modules that must be inspected
$RI = C_{22}/C_{12}$	Wasted inspection	Modules that were incorrectly identified as problematic

Table 4.5: Metrics Equations

The initial set-up of the contingency table (Table 4.7) is similar to that of the set-up for the Inverse Kolmogorov-Smirnov function. The random sample that was generated, using the RANDOMBETWEEN() function, is used and the VLOOKUP() function

extracts the values from the data table. The values are then placed in a separate table and the contingency table is constructed.

The contingency table is constructed using a series of "if-then" statements to act on the columns of the new table containing the statement, comment, and the drcount values. The "if-then" statements are constructed so that each module, selected using the random function, will fall into a particular quadrant of the contingency table as discussed in Chapter III. If the combination of "statement", "comment" and "drcount" meet the conditions of the "if-then" function, the statement is true. If the conditions are not satisfied then the statement is false. Four such columns are constructed, one for each quadrant. The conditional statements [SCH97] for each quadrant using the bins derived above are:

1. C11 - drcount equal to zero AND Comments less than or equal to 38 AND statements less than or equal to 26.
2. C12 - drcount equal to zero AND (Comments greater than 38 OR statements greater than 26).
3. C21 - drcount greater than zero AND Comments less than or equal to 38 AND Statements less than or equal to 26.
4. C22 - drcount greater than zero AND (Comments greater than 38 OR statements greater than 26).
5. The number of "true" values in each column is then summed using the "AutoSum" function and the result is placed into a cell with a similar design as a contingency table for better user visualization. Examples taken from the

"Calculations" worksheet of the "Inverse K-S" workbook and are shown in Tables 4.6 and 4.7.

INDEX #	MODULE NAME	RANDOM INDEX # FOR SAMPLE 1	COMMENTS C11 FOR SAMPLE 1	COMMENTS C12 FOR SAMPLE 1	COMMENTS C21 FOR SAMPLE 1	COMMENTS C22 FOR SAMPLE 1
1	A01X	20	FALSE	FALSE	FALSE	TRUE
2	A01Y	22	FALSE	TRUE	FALSE	FALSE
3	A02X	35	TRUE	FALSE	FALSE	FALSE
4	A03X	64	TRUE	FALSE	FALSE	FALSE
5	A04X	66	TRUE	FALSE	FALSE	FALSE
6	A05X	68	TRUE	FALSE	FALSE	FALSE
7	AIBG	73	FALSE	FALSE	FALSE	TRUE
8	AIDD	83	TRUE	FALSE	FALSE	FALSE
9	AIES	89	TRUE	FALSE	FALSE	FALSE
10	AIGD	90	TRUE	FALSE	FALSE	FALSE

Table 4.6: Conditional Statement Results for Randomly Selected Modules

	COMMENTS≤ 38 AND S ≤26	COMMENTS > 38 OR S >26	TOTAL
Drcount=0	30	27	57
Drcount>0	1	42	43
Total	31	69	100

Table 4.7: Contingency Table Results from Excel Worksheet Calculation

Once the values for the contingency table are known, the remaining calculations [SCH97] are easily solved using *Excel*. The equation results are listed above in Table 4.8. The calculations have been validated against the *Statgraphics* results for one set of data.

N =	100
n2 =	43
P1(%) =	1.0
P2(%) =	27.0
P12 =	0.28
LQC(%) =	97.7
RF =	1
RFD =	0.01
TF =	192
RFP(%) =	0.52
RFM =	1
RMP(%) =	1.00
I(%) =	69
RI =	1.56

Table 4.8: Metric Equation Results

D. A DATABASE MANAGEMENT SYSTEM

Software reliability engineering requires the collection of large amounts of data. During the testing phase of the software development cycle, failures will occur and discrepancy reports will be generated to document these failures. To be useful, the discrepancy reports must contain as much information as possible regarding the nature of the failure. In the case of the Space Shuttle software, failure data are currently provided as a *Microsoft Excel* spreadsheet by the flight software contractor, as shown in Table 4.9. Analysts use the data to make software reliability predictions. Storing the accumulated failure data in the form of a spreadsheet would seem to be a perfectly suitable solution to the problem of maintaining an accurate record for the analyst. As has been previously noted, spreadsheets have many attractive characteristics that make them a desirable solution for this purpose. Spreadsheets have an additional advantage in this project because our goal is to use them for the final data analysis. There are, however, several major disadvantages in the use of a spreadsheet that can have detrimental effects on the reliability prediction process. The major disadvantage is that spreadsheets don't allow for the definition of all the desired relationships between data fields.

When conducting an analysis of software failures, it is often necessary to search the accumulated data in order to identify associations between fields or combinations of fields. In Table 4.9, for instance, an analyst may want to know how the number of *Days From Release* relate to the *Module in Error*, or how often does a particular module fail.

Release	Days from Build	Reference Number	Severity	Failure Date	Build Date	Build	Module in Error
F'	385	199	1	10/31/91	10/11/90	1	GEBA
	19	75	3	05/21/91	05/02/91	3	FCMC
							FCMS
G	189	399	3	11/07/91	04/29/91	5	CGEI
	249	306	3	10/22/91	02/13/91	4	FIOM
	72	324	2	11/06/91	09/04/91	7	DMPM
	162	372	1	12/20/91	06/07/91	6	GGEE
	175	377	1	01/02/92	07/07/91	6	CGCZD
OIG'	346	1	3	01/27/92	02/15/91	8	FCMM

Table 4.9: Software Failure Log

These examples are rather obvious associations; other connections that exist in the data may not always be so intuitive. It is likely that there are associations between the software failures and the metrics of the software. This is where the use of a spreadsheet for maintenance of the data is deficient. While it is possible to sort a spreadsheet of discrepancy reports to make the obvious associations, correlating metrics to failures requires a much more powerful tool. For this reason, it is advantageous to store the data in a database and transfer the data to the spreadsheet as necessary.

Using a database permits the development of a system that will hold all of the data generated in a tabular format very similar to the spreadsheet, yet provides flexibility in the manipulation of the data. After thorough testing and collection of the failure data, the analyst can then use queries to examine complex relationships. Data mining techniques can be employed to examine historical data. A database allows flexibility in the analysis of large amounts of data.

A Database Management System is simply software that manages data. A Relational Database Management System is a Database Management System that is based on the relational model. The relational model relies on the mathematics of set theory, providing

a solid theoretical base for the management of data. Relational databases are typically easier to use and maintain than non-relational databases. The relational model provides for improved data access, data integrity, and data security.

Every build of the software under development will have it's own unique set of characteristics or metrics. When the metrics are defined in a table in the same database as the failure table, associations between the failures and the metrics can be established. This will allow the analyst a much more detailed and comprehensive view of what is actually occurring within the software as it is executing.

Since Microsoft Office is mandated for use in U. S. government PCs, the RDBMS selected for use in this project was Microsoft Access. Microsoft Access is a relational database management system that can be used for simple data storage, or to develop stand-alone applications. It contains full support for Structured Query Language (SQL), which is used for defining the structure and processing of a relational database.

The first step in designing the Software Reliability Engineering Database was to decide which tables to include and the relationships that exist between them. Tables in the relational model are used to represent things in the real world. Each table should represent only one type of thing. These "things" can be real world objects, events, or logical concepts [KRO97]. In this project the "things" are the logical concepts of software releases, the builds within those releases, the failures that occur in the builds, and the physical characteristics (metrics) of the software. In the relational model, a table must have the following properties [KRO97]:

1. The entries must be single valued; neither repeating groups nor arrays are allowed.

2. All of the entries in any column are of the same kind. For example, if one column contains Failure Reference Numbers, it must have a Failure Reference Number for every row of the table. The columns are called attributes. Each attribute has a domain, which is a physical and logical description of allowed values.

3. No two rows can be identical, and the order of the rows is not important.

Examination of Table 4.9 shows that it violates the second requirement, since every row does not have a failure reference number. In order to meet the requirements for being relational, this table must be reconfigured. By making new columns for each additional module in error, a new table is produced that meets the criteria for being relational.

In order to ensure that the database operates as smoothly as possible, with a minimum number of data anomalies*, it is essential that we construct tables in such a way that each represents a single concept within our model. As stated previously, examination of Table 4.10, shows that three logical concepts exist within it. The first is the concept of **Release**; each release has a name, identifying OI (Operational Increment) number and release date. The second concept is that of **Build**; each build has a build number and date. The third concept is that of **Failure**; each failure has a reference number, date, severity, and set of modules that failed. Therefore, Table 4.10 will have to be divided into three separate tables; Release, Build and Failure.

---

* A *data anomaly* is an error or inconsistency in the database. There are three types of anomalies: insert, delete, and update. These anomalies occur during the insertion, deletion, and updating of rows. For example, an insert anomaly will occur if the insertion of a new row caused a calculated total field in another table to report the wrong total. [LIT96]

Release	Reference Number	Days from Build	Severity	Failure Date	Build	Module in Error #1	Module in Error #2
F'	199	385	1	10/31/91	1	GEBA	
	75	19	3	05/21/91	3	FCMC	FCMS
	150	81	3	07/22/91	3	CGMC	CGMM
	301	238	1	10/11/91	2	GEHR	GN1D
G	399	189	3	11/07/91	5	CGEI	
	306	249	3	10/22/91	4	FIOM	
	308	172	3	10/21/91	5	GCJA	
	324	72	2	11/06/91	7	DMPM	
	372	162	1	12/20/91	6	GGEE	
	377	175	1	01/02/92	6	GCZD	
G'	1	346	3	01/27/92	8	FCMM	FCMSA

Table 4.10: Updated Software Failure Log

One final relation will be added to the database to make it fit with the project; we will add a Metrics relation. Going back to the Entity-Relationship model, we see that Release, Build, Failure, and Metrics are Entities with the following relationships:

- Release has a one-to-many relationship with Build (for each Release, there may be more than one build.
- Build has a one-to-many relationship with Failure and a many-to-one relationship with Metrics (each build can have many failures and several builds can have the same set of metrics values).

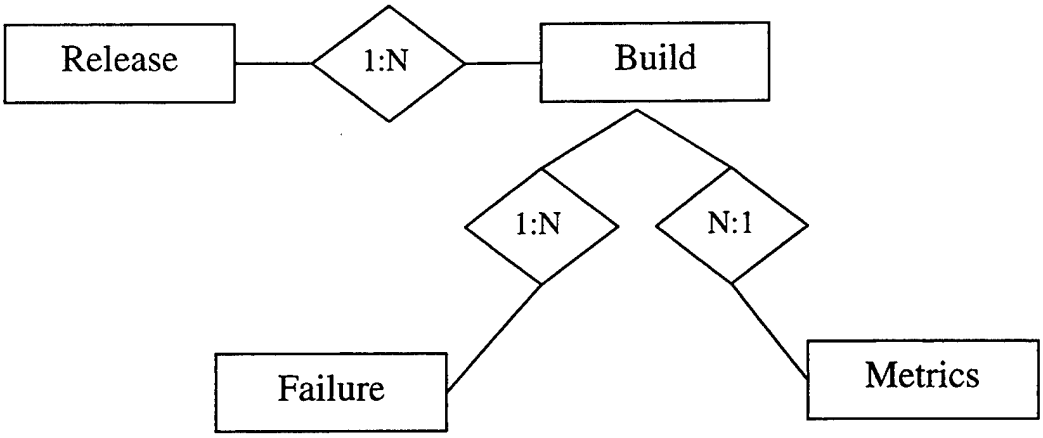

Figure 4.1: Entity-Relationship Model for Software Failure Database

The next step is to define the attributes or fields for each relation. Since there is already a great amount of data in Excel spreadsheets, we will want to define the fields in a way that allows us to easily transfer the data from the spreadsheet to the database. For the failure data, this will be most easily accomplished if the field definitions in the database closely match the column definitions in the spreadsheet provided by the developers. The Release relation will have the attributes Release -- to coincide with the

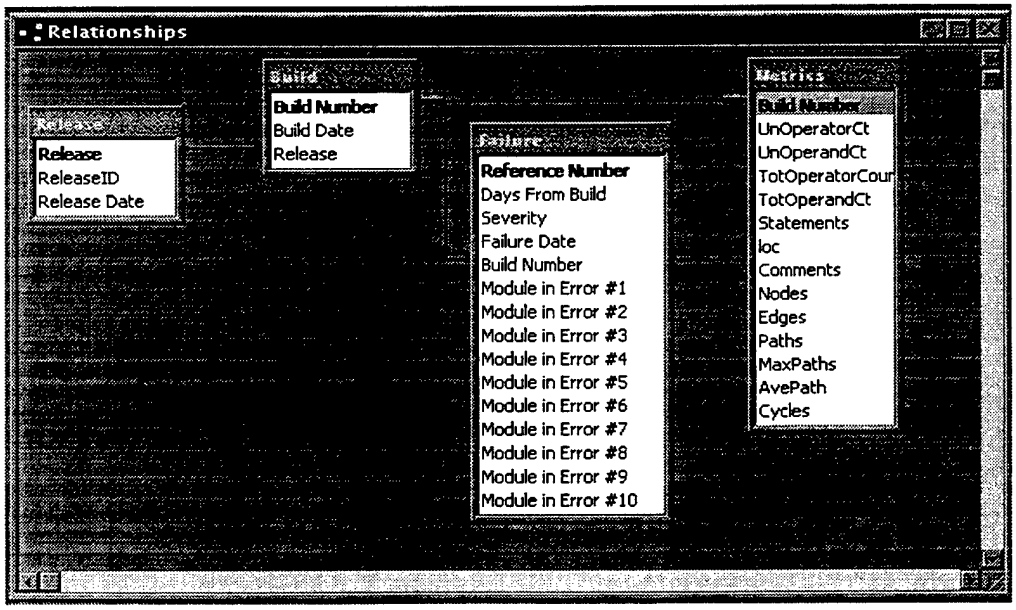

Figure 4.2: Software Failure Database Relations as Defined by Microsoft Access

Release Name --, Release ID for the number identifier, and Release Date. The Build relation will have the attributes Build Number, Build Date and Release, for the unique release that a particular build belongs to. The Failure relation will have the attributes Reference Number, Days From Build, Severity, Failure date, Build Number, and place holders for up to ten Modules in Error. The Metrics relation will have 13 attributes to correspond with the characteristics of the software that we wish to measure. Figure 4.2 shows the relationships after the tables have been constructed.

The Software Reliability Engineering database is now ready for data input. The historical data can be copied and pasted from the existing spreadsheet into the appropriate tables of the database, or brought in using the *Connect* property in Microsoft Access, which allows tables to be linked to Excel spreadsheets for automated data entry. The result is seen in Figure 4.3, a view from a portion of the Release relation.

Release	ReleaseID	Release Date
A	OIA	9/1/83
B	OIB	9/1/83
C	OIC	12/12/83
C'	OIC'	4/17/84
C"	OIC"	8/2/84
D	OID	6/8/84
D'	OID'	10/26/84
E	OIE	10/5/84
E'	OIE'	8/27/85
F	OIF	4/10/85
F'	OIF'	9/25/85
G	OIG	12/17/85
G'	OIG'	4/3/86

Figure 4.3: Example View from Release Relation

Ideally the database would be used by the developers as a replacement for the spreadsheet that has been used until now for logging software failures. Figure 4.4 shows an input form that has been created to make the use of the database for this purpose a

more attractive alternative. When software failures are entered into the database using this form, the data will automatically be entered into the proper table, obviating the need for the analyst to reformat the spreadsheet to comply with the database design.

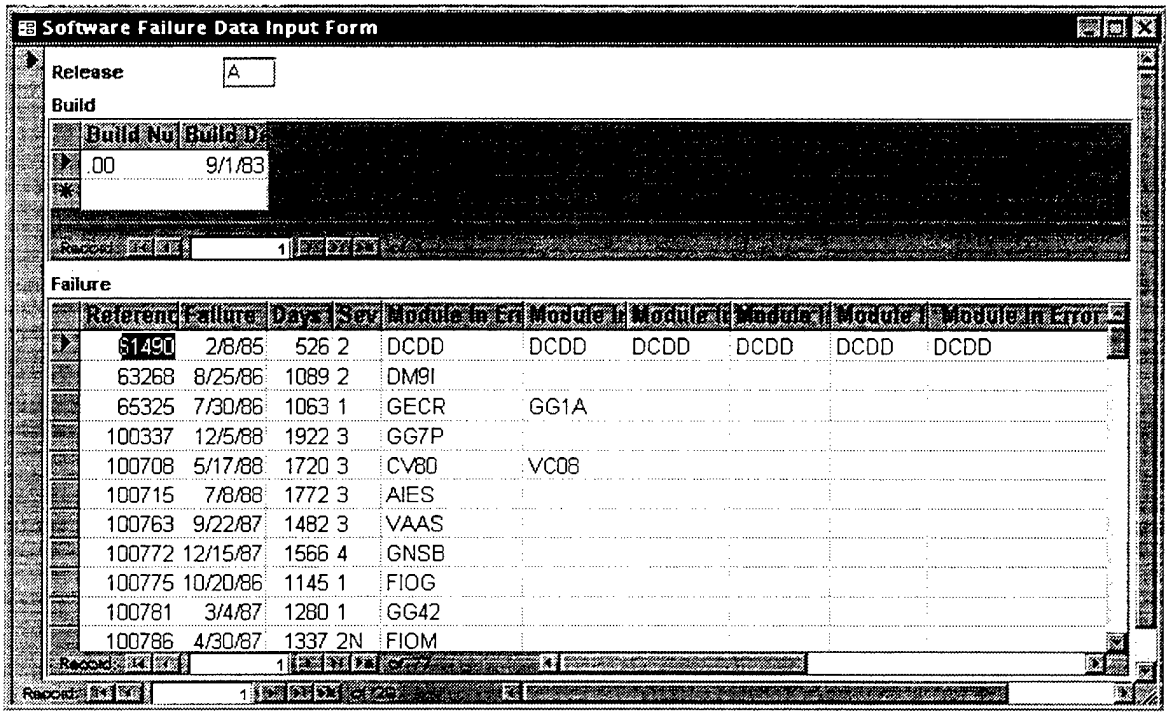

Figure 4.4: Software Failure Database Input Form

The user can use SQL to create a query to capture the data in any from he chooses. The following query will generate a report that looks like a stylized version of the original spreadsheet, as shown in Figure 4.5 below.

```
SELECT DISTINCTROW Release. Release, Build.[Build Number], Build.[Build Date],
Failure.[Reference Number], Failure.[Failure Date], Failure.[Days From Build],
Failure.Severity, Failure.[Module in Error #1], Failure.[Module in Error #2],
Failure.[Module in Error #3], Failure.[Module in Error #4], Failure.[Module in Error #5],
Failure.[Module in Error #6]
FROM Failure INNER JOIN (Release INNER JOIN Build ON Release.Release =
Build.Release) ON Failure.[Build Number] = Build.[Build Number]
ORDER BY Release.Release, Build.[Build Number];
```

This report can be used by those who are more comfortable with the layout of the original spreadsheet. Additional queries can be developed by the analyst, making it possible to view the data in any way he desires.

Failure Data

Release	Build No.	Build Date	Failure Ref Number	Failure Date	Days From Build	Severity	Modules in Error	
F	1	9/25/85	199	10/31/91	385	10	GEBA	
	2		301	10/11/91	238	1	GEHR	GN1D
	3		75	5/21/91	19	3	FCMC	FCMS
			150	7/22/91	81	3	CGMC	CGMM
G	4	6/7/85	306	10/22/91	249	3	FIOM	
	5	2/27/85	308	10/21/91	172	3	GCJA	
			399	11/7/91	189	3	CGEI	
	6	4/29/85	372	12/20/91	162	1	CGEE	
			377	1/2/92	175	1	GCZD	
	7		324	11/6/91	72	2	DMPM	
G'	8	4/3/86	1	1/27/92	346	3	FCMM	FCMS
	9		15	2/17/92	291	2	CGMG	GIXG
			22	2/24/92	298	3	GIXG	
H	10	7/7/86	998	7/11/93	867	2	FCMC	FIOM
	11	12/6/86	736	11/18/94	1296	1	GCJA	
	12	1/30/87	21	2/19/92	223	3	GCJA	
	13		85	2/12/96	1631	3	FIOC	

Figure 4.5: Example Failure Report from Software Failure Database

The Software Reliability Prediction Database is designed to be an important tool used in the updated process. It provides the means with which the software failure data and the software metrics data can be joined to form a complete prediction system. If accepted, this system could provide a means for a better understanding of how software characteristics affect software reliability and thus lower development time and cost.

V. CONCLUSIONS AND RECOMMENDATIONS

This chapter presents conclusions and recommendations drawn from the analysis of software tools that could be used for making software reliability predictions. The chapter concludes with recommendations for action and further research.

A. CONCLUSIONS

The primary motivation for performing this analysis was to provide software quality and reliability prediction tools that would be useable for software developers as well as analysts. In order to achieve the desired level of usefulness, software must be used for the tools that are widely available. We have found that it is possible to perform most of the analysis using such software. *Microsoft Office*, currently the U. S. Government standard productivity application, provided the desired level of usefulness. The only problem encountered was that the estimation of reliability model parameters used in most of the reliability calculations, s, alpha, and beta, must be calculated by SMERFS. It has been demonstrated however, that if s, alpha and beta are provided by SMERFS, the reliability calculations can be performed using *Excel.*

This analysis has shown that:

- Developers can use this system to perform reliability predictions throughout a project's lifecycle.
- It is possible, using only *Microsoft Excel*, to use metrics as indicators of software reliability.

Throughout this study, emphasis has been placed on the effective use of the data collected during a software development project. For any such project to be successful, it is imperative that data collection begin at the earliest stage and continue throughout the

entire development cycle. As the software is developed, the pertinent metrics need to be collected. The data collected can then be transferred directly into the metrics table of the software reliability database. From the database, queries can be developed to extract the desired information, which can be used to perform statistical analysis on the metrics themselves or, as demonstrated in chapter four, can be used as input to specific *Excel* equations to make quality predictions. These calculations can be used to judge reliability long before software testing begins, thus providing the developers with an early warning of potential problems in the software.

B. RECOMMENDATIONS

The following recommendations are based on the results of this study. These recommendations reflect the authors' desire to provide a standardized means to make software quality and reliability predictions.

1. Adapt the software reliability prediction database as one of the tools for entering and storing software failure and metrics data.

In the ideal situation, once testing has begun, software failures would be logged into the reliability tables of the software reliability database. The database provides a common repository for all information pertaining to the project. Although all of the data may not be used in the analyst's current calculations, they should be maintained for possible future reference.

The database has the additional benefit of providing an easily searchable storage facility. Queries can be used to capture desired data such as the number of failures and when they occurred. This information can be used as input for SMERFS in order to

estimate model parameters. These parameters are then used in equations in a spreadsheet to make reliability predictions.

2. Update the metrics analysis tools to *Excel* in order to automate metrics calculations more fully.

Using these tools to analyze metrics data in the design phase could eliminate significant faults early in the project and save both time and money during later phases.

3. Build another, more comprehensive system.

Since the authors are not computer scientists, this study was never intended to produce a new software reliability prediction system. The study could, however, be the foundation for a follow on thesis to produce one. We recommend that a future student, with programming knowledge, use the database design and equations developed here and add to them a Visual Basic program that uses the SMERFS algorithms to derive the parameters, s, alpha and beta. Then, the entire prediction system could exist in one tool.

LIST OF REFERENCES

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

[BEH96] Behforooz, Ali and Hudson, Frederick J. *Software Engineering Fundamentals*, Oxford University Press, 1996.

[CON6-98] Conversation between James O'Leary and Professor Norman F. Schneidewind, Naval Postgraduate School, August 6 1998.

[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.

[IDG97] Ruth Maran, *Teach Yourself Access97 Visually*, IDG Books Worldwide Inc. Foster City Ca., 1997, pp. 35-52.

[IEE93] "Standard for a Software Quality Metrics Methodology", IEEE Std 1061-1992, March 12 1993.

[KRO97] Kronke, David M., *Database Processing : Fundamentals, Design, and Implementation,* Prentice-Hall, September 1997.

[LAR86] Richard Larsen and Morris Marx, *An Introduction to Mathematical Statistics and its Applications*, 2nd ed., Prentice-Hall, Englewood Cliffs, NJ, 1986.

[LIE97] Bernard V. Liengme, *A Guide to Microsoft Excel for Scientists and Engineers,* John Wiley & Sons, July 1997.

[LEW92] Robert O. Lewis, *Independent Verification and Validation: A Life Cycle Engineering Process for Quality Software*, John Wiley & Sons, NY, 1992, pp. 253-262.

[LIT96] Litvin, Paul, et al, *Microsoft Access 95 Developer's Handbook, Sybex, Inc., 1996,* pp. 59.

[LYU96] Lyu, Michael R., et al, *Handbook of Software Reliability Engineering,* IEEE Computer Society Press, November 1996.

[MAYR90] Anneliese Von Mayrhauser, *Software Engineering Methods and Management*, Academic Press Inc., 1990, pp.553-562.

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

[SCH92] Schneidewind, Norman, , "Methodology for Validating Software Metrics", IEEE Transactions on Software Engineering, Vol. 18, No. 5, May 1992.

[SCH94] Schneidewind, Norman, "Validating Metrics for Controlling and Predicting the Quality of Space Shuttle Flight Software", IEEE Computer, Vol. 27, No. 8, August 1994.

[SCH96] Schneidewind, Norman, *MCTSSA Software Reliability Handbook, Volume I, Software Reliability Process and Modeling for a Single Process*, Naval Postgraduate School, January, 1996.

[SCH97] Schneidewind, Norman, *MCTSSA Software Reliability Handbook, Volume II, Data Collection Demonstration and Software Reliability Modeling for a Multi-Functional Distributed System*, Naval Postgraduate School, August, 1997.

[SCHN97] Norman F. Schneidewind, "Reliability Modeling for Safety Critical Software", IEEE Transactions on Reliability, Vol. 46, No. 1, March 1997.

[SCH98] Norman F. Schneidewind, "How to Evaluate Legacy System Maintenance", IEEE Software, Vol. 15, No. 4, July/August 1998, pp. 34-42. Also translated into Japanese and reprinted in: Nikkei Computer Books, Nikkei Business Publications, Inc., 2-1-1 Hirakawacho, Chiyoda-Ku, Tokyo 102 Japan, 1998, pp. 232-240.

[SCN97] Norman F. Schneidewind, *A Software Metrics Model for Integrating Quality Control and Prediction*, Proceedings of the International Symposium on Software Reliability Engineering, Albuquerque, New Mexico, November 4, 1997, pp. 402-415.

[SPLUS97] *S-Plus 4: Guide to Statistics*, Mathsoft Inc. Seattle Wash., July 1997, pp.75-81.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Road, Ste 0944
   Fort Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Road
   Monterey, California 93943-5101

3. Norman F. Schneidewind..... 1
   Professor, Information Systems Management, Code IS/Ss
   Naval Postgraduate School
   Monterey, CA 93940

4. Douglas E. Brinkley 1
   Lecturer, Information Systems Management, Code SM/Bi
   Naval Postgraduate School
   Monterey, CA 93940

5. CDR Dennis Brophy 2
   657 Persimmon Rd.
   Walnut Creek, CA 94598

6. LCDR James O'Leary. 2
   211 14th St
   Pacific Grove, CA 93950