

To the Graduate Council:

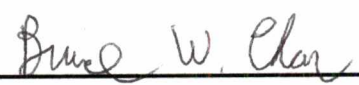
I am submitting herewith a thesis written by Clifford Kimbrough Cobb entitled "Selecting a Software Reliability Model Based Upon Failure Data Characteristics." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.



Dr. J.H. Poore, Major Professor

We have read this thesis
and recommend its acceptance:





Accepted for the Council:



Vice Provost
and Dean of The Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature Clifford K. Cobb

Date April 12, 1988

**SELECTING A SOFTWARE RELIABILITY
MODEL BASED UPON FAILURE
DATA CHARACTERISTICS**

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Clifford Kimbrough Cobb

June 1988

For Craig

*May he have inherited from me the love for learning
that I inherited from my father.*

ACKNOWLEDGMENTS

I wish to acknowledge the many contributions of Dr. Jesse Poore to this thesis project. Throughout the course of this research, his insightful ideas, comments, and suggestions have invariably facilitated solution of the problem at hand. I am fortunate to have studied under his influence, and I hope that he feels that his faith in me has been justified.

I wish to thank Dr. Ronald Leinius and Dr. Bruce Char for their service on the thesis committee and for their helpful suggestions.

I wish to thank my management at Martin Marietta Energy Systems, Inc., especially Bob Easterday and Robin Textor, for their encouragement and support.

I also wish to thank Dr. Bill McClain, whose unselfishness in teaching courses at night over the past five years made the pursuit of a graduate degree much easier.

Finally, I wish to thank my wife Kathy, whose support and patience over the past five years have been appreciated more than she could ever know.

ABSTRACT

Classical reliability theory is fundamentally applicable and useful in the software environment, notwithstanding the observation that software does not deteriorate with age. Several models have been proposed that attempt to better characterize the failure behavior of software. Each software reliability model has advantages and disadvantages in terms of analytical complexity, predictive validity, required data, ease of understanding, and ease of use.

Nine software reliability models are presented. The assumptions underlying each of the models are presented and analyzed to determine their accuracy in modeling the behavior of software. The mathematical expressions that are used to compute reliability measures for each of the models are derived from the assumptions that form the basis of the respective models. These mathematical expressions contain parameters that are required for the computation of the various reliability measures. The methods used to estimate such parameters in the implementations of each of the models are given.

The models presented are compared for reported failure data sets chosen to be representative of failure data distributions for software systems of high, medium, and low quality. Based upon these comparisons, specific models are recommended for predicting reliability measures for software systems of high, medium, and low quality, respectively.

TABLE OF CONTENTS

CHAPTER	PAGE
1 INTRODUCTION	1
1.1 Reliability Concepts	3
1.2 Types of Software Reliability Models	6
1.3 Overview of Subsequent Chapters	8
2 INTRODUCTORY RELIABILITY THEORY	11
2.1 Motivation for the Hazard Function	11
2.2 The Exponential Distribution	15
3 TIMES-BETWEEN-FAILURES MODELS	17
3.1 The Jelinski-Moranda De-Eutrophication Model	18
3.1.1 Model Assumptions	18
3.1.2 Model Development	21
3.1.3 Parameter Estimation	22
3.2 The Geometric De-Eutrophication Model	25
3.2.1 Model Assumptions	25
3.2.2 Model Development	27
3.2.3 Parameter Estimation	28

CHAPTER	PAGE
3 (Continued)	
3.3 The Schick-Wolverton Model	30
3.3.1 Model Assumptions	31
3.3.2 Model Development	32
3.3.3 Parameter Estimation	35
3.4 The Goel-Okumoto Imperfect Debugging Model	37
3.4.1 Model Assumptions	37
3.4.2 Model Development	38
3.4.3 Parameter Estimation	43
3.5 The Littlewood Bayesian Differential Debugging Model	44
3.5.1 Model Assumptions	45
3.5.2 Model Development	46
3.5.3 Parameter Estimation	52
3.6 The Certification Model	54
3.6.1 Model Assumptions	55
3.6.2 Model Development	56
3.6.3 Parameter Estimation	58
3.7 Example Reliability Measure Calculations	59

CHAPTER	PAGE
4 FAILURE-COUNT MODELS	70
4.1 The Goel-Okumoto Nonhomogeneous Poisson Process Model . .	71
4.1.1 Model Assumptions	71
4.1.2 Model Development	73
4.1.3 Parameter Estimation	76
4.2 The Musa-Okumoto Logarithmic NHPP Execution Time Model	78
4.2.1 Model Assumptions	79
4.2.2 Model Development	80
4.2.3 Parameter Estimation	83
4.3 The Musa Execution Time Model	85
4.3.1 Model Assumptions	86
4.3.2 Model Development	88
4.3.3 Parameter Estimation	92
4.4 Example Reliability Measure Calculations	95
5 MODEL COMPARISONS AND RECOMMENDATIONS . .	101
5.1 Model Comparison Criteria	101
5.2 General Recommendations	107

CHAPTER	PAGE
5 (Continued)	
5.3 Model Comparisons	109
5.3.1 High Quality Software Failure Data	110
5.3.2 Medium Quality Software Failure Data	111
5.3.3 Low Quality Software Failure Data	116
5.4 Specific Recommendations	117
5.4.1 High Quality Software	117
5.4.2 Medium Quality Software	121
5.4.3 Low Quality Software	122
5.4.4 Summary of Recommendations	123
6 SUMMARY AND CONCLUSIONS	125
LIST OF REFERENCES	127
APPENDIX	132
Data Values Plotted, Figure 5-1 Through Figure 5-6	133
VITA	142

LIST OF TABLES

TABLE	PAGE
3-1 Musa failure data times (CPU seconds).	60
3-2 Illustrated reliability measures, Figures 3-9 through 3-19.	62
4-1 Illustrated reliability measures, Figures 4-4 through 4-8.	96
5-1 Predicted reliability measures.	104
5-2 Model attributes.	106
5-3 Mills failure data times (CPU seconds).	110
5-4 Abdel-Ghaly failure data times (CPU seconds).	116
5-5 Model recommendations.	124
A-1 Data values plotted, Figure 5-1 (Mills failure data).	134
A-2 Data values plotted, Figure 5-2 (Mills failure data).	135
A-3 Data values plotted, Figure 5-3 (Musa failure data).	136
A-4 Data values plotted, Figure 5-4 (Musa failure data).	137
A-5 Data values plotted, Figure 5-5 (Abdel-Ghaly failure data).	139
A-6 Data values plotted, Figure 5-6 (Abdel-Ghaly failure data).	140

LIST OF FIGURES

FIGURE	PAGE
1-1 Reliability as a function of time.	4
1-2 Illustration of MTTF.	5
2-1 The exponential distribution.	16
3-1 Jelinski-Moranda model failure rate process.	22
3-2 Geometric de-eutrophication model failure rate process.	28
3-3 Schick-Wolverton model failure rate process.	34
3-4 Illustration of the imperfect debugging process.	39
3-5 Imperfect debugging model failure rate process.	42
3-6 Littlewood model failure rate process.	51
3-7 Certification model failure rate process.	57
3-8 Musa failure data distribution.	61
3-9 Failure rates, Jelinski-Moranda model.	63
3-10 Percentages of faults detected, Jelinski-Moranda model.	63
3-11 Failure rates, geometric de-eutrophication model.	64
3-12 Percentages of faults detected, geometric de-eutrophication model.	64
3-13 Failure rates, Schick-Wolverton model.	65
3-14 Percentages of faults detected, Schick-Wolverton model.	65
3-15 Failure rates, imperfect debugging model.	66
3-16 Percentages of faults detected, imperfect debugging model.	66

FIGURE	PAGE
3-17 Failure rates, Littlewood model.	67
3-18 Percentages of faults detected, Littlewood model.	67
3-19 Failure rates, certification model.	68
4-1 Goel-Okumoto NHPP model failure intensity process.	75
4-2 Musa-Okumoto model failure intensity process.	81
4-3 Musa model failure intensity process.	91
4-4 Failure rates, Goel-Okumoto NHPP model.	97
4-5 Percentages of faults detected, Goel-Okumoto NHPP model. . .	97
4-6 Failure rates, Musa model.	98
4-7 Percentages of faults detected, Musa model.	98
4-8 Failure rates, Musa-Okumoto model.	99
5-1 Current MTTF predictions, Mills failure data.	112
5-2 Future MTTF predictions, Mills failure data.	113
5-3 Current MTTF predictions, Musa failure data.	114
5-4 Future MTTF predictions, Musa failure data.	115
5-5 Current MTTF predictions, Abdel-Ghaly failure data.	118
5-6 Future MTTF predictions, Abdel-Ghaly failure data.	119

CHAPTER 1

INTRODUCTION

The most important attribute of a computer software system is the degree to which it may be relied upon to perform its intended function. Discrepancies often exist between the intended function of the software and the actual function performed by the software. These discrepancies, or failures, are caused by the existence of errors, or faults, in the software.

Two methods for indicating the existence of software faults are proving the correctness of software and software testing. Proving the correctness of software is a formal and mathematical process. In order to prove a software system is correct, a sequence of logically related statements is constructed, ending with the statement to be proved. This final statement is usually the output specification statement [1].

Methods for proving the correctness of software include inductive assertion, symbolic execution, and more recently, the method of functional verification developed by Mills [2]. Functional verification is the most promising of these methods because as the complexity of a software system increases, the effort required to prove its correctness using functional verification techniques increases linearly. However, functional verification techniques may be applied only to

software systems that are expressed using a combination of the seven structures described by Mills.

As the complexity of a software system increases, the effort required to prove its correctness using the inductive assertion method or the symbolic execution method increases geometrically. Therefore, it is generally not feasible to prove the correctness of software systems using such methods, even with the aid of automated tools [3].

Software testing is a heuristic process. If all possible sets of input data to a software system are known, then the software may be executed for each set of input data and a statement may be made concerning the correctness of the software based upon those results. However, it is very unlikely that all possible sets of input data to a software system are known. Therefore, the software must be tested on a small fraction of all possible inputs to determine how it will function in its intended computing environment. Since a software system cannot be tested for correctness for all possible sets of input data, not all of the possible faults can be exposed during testing.

Since neither software testing nor proving the correctness of software can guarantee complete confidence in the correctness of a software system, a measure is needed that reflects the degree of software system correctness. One such quantifiable measure [1] is software reliability.

1.1 Reliability Concepts

The ability to measure the reliability of software is important for the following reasons [4,5,6]:

- software reliability measures may be used to predict future failures of software;
- software reliability measures may be used to determine confidence limits of the future performance of software;
- software reliability measures may be used to monitor the progress of software testing or the operational performance of software using continually updated estimates of current reliability.

The measures used most often are reliability, the hazard function, and the mean time to failure (MTTF).

The reliability of a unit of age t is defined as the probability that no failure occurs in the time interval $[0, t]$. For software, a failure is defined as any departure of program output from requirements as the program is executed [6,7]. A failure is caused by an error, or fault, in the software. Since reliability is a probabilistic measure, it ranges between 0 and 1, inclusive. Reliability at time $t = 0$ is defined to be 1. Generally, the reliability of a unit decreases as the time interval increases. This process is depicted in Figure 1-1.

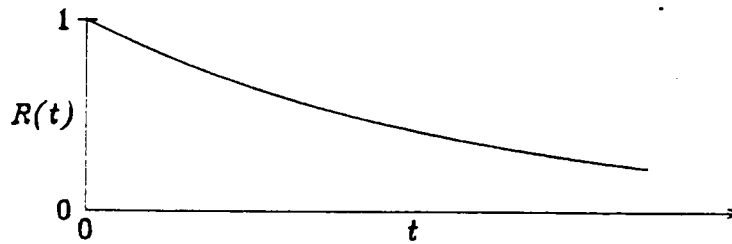


Figure 1-1: Reliability as a function of time.

The hazard function, which is also referred to as the failure intensity function in the literature, is defined as the conditional probability that a failure occurs in the time interval $(t, t + \Delta t]$ given that no failure occurred in the time interval $[0, t]$ [8]. As Δt approaches zero, the hazard function represents an instantaneous failure rate. The terms failure rate and instantaneous failure rate are used interchangeably here.

MTTF is defined as the expected length of the time interval from the current failure to the next failure [9]. MTTF is generally determined by having some knowledge of the probability distribution of failure occurrences and using observed failure data to estimate parameters of the probability distribution. The resulting model is then used to predict future failures [8]. The concept of MTTF is illustrated in Figure 1-2.

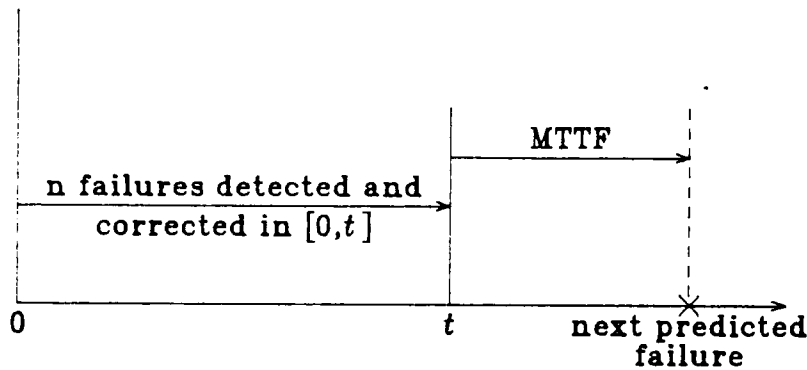


Figure 1-2: Illustration of MTTF.

In addition to the definitions of the reliability measures given above, the following precise definition of software reliability is credited to Moranda by Goel [1]:

Let F be a class of faults, defined arbitrarily, and T be a measure of relevant time, the units of which are dictated by the application at hand. Then the reliability of the software with respect to the class of faults F and with respect to the metric T is the probability that no fault of the class occurs during the execution of the program for a pre-specified period of relevant time.

A software reliability model should be able to predict future failure behavior of software from observed failure data. This thesis presents an analysis and comparison of various time-dependent software reliability models. Assumptions underlying the models are analyzed to determine their accuracy in modeling the behavior of software. Similarities and differences among the models are discussed. Reliability measures predicted by the various models using published

software failure data are presented and compared. Reliability measures predicted by the various models using failure data distributions chosen to be typical of failure data distributions likely to be observed for software systems developed in various environments are also presented and compared. Guidelines and recommendations for selecting a software reliability model based upon the failure data characteristics of the software being modeled are presented.

1.2 Types of Software Reliability Models

Beginning in the early 1970s, the literature contains a number of analytical models for measuring software reliability. A software reliability model is a representation of a random process through which software reliability, or a directly related quantity such as the hazard function or MTTF, is characterized as a function of the software [6]. Software reliability models may be divided into the following four basic classes:

1. times-between-failures models;
2. failure-count models;
3. fault seeding models;
4. input domain based models.

Times-between-failures models and failure-count models are time-dependent. Fault seeding models and input domain based models are time-independent.

A brief description of each type of model is now given. However, since fault seeding models and input domain based models are time-independent, they do not have a hazard function or MTTF associated with them and will not be considered further.

In times-between-failures models, the approach is generally to assume that the time between failures $(i - 1)$ and i follows a distribution whose parameters depend upon the number of faults in the software at that time interval. Estimates of the parameters are determined from the previously observed times between failures. Software reliability and related quantities are then determined from the models. It is expected that times between failures increase as faults are removed from the software.

Failure-count models estimate the number of failures during specified time intervals. The failure counts are assumed to have a time-dependent hazard function. Parameters of the hazard function are estimated from the previously observed failure counts or times between failures. Software reliability and related quantities are then determined from the models. It is expected that failure counts decrease as faults are removed from the software.

In fault seeding models, the approach is to seed a known number of faults into software that is assumed to have an unknown number of indigenous faults. The software is then tested and the number of observed seeded and indigenous faults are counted. An estimate of the fault content of the software prior to seeding

is determined. The most popular of the fault seeding models was developed by Mills [10].

In input domain based models, the approach is to execute the software for a sample of input data sets that is assumed to be representative of the input distribution. It is desirable to divide the input domain into equivalence classes based upon path coverage of the software. An estimate of the software reliability is determined from the failures observed during execution of the test cases sampled from the input distribution. Input domain based models have been developed by Nelson [11] and Ramamoorthy and Bastani [12].

1.3 Overview of Subsequent Chapters

Chapter 2 presents a discussion of reliability theory applicable to software reliability models. The motivation for the hazard function is developed, and expressions for reliability and MTTF in terms of the hazard function are given.

Chapter 3 presents an analysis of each of the following times-between-failures software reliability models:

1. the Jelinski-Moranda de-eutrophication model [4];
2. the geometric de-eutrophication model [13];
3. the Schick-Wolverton model [14];
4. the Goel-Okumoto imperfect debugging model [15,16];

5. the Littlewood Bayesian differential debugging model [17,18];
6. the certification model [19,20].

Assumptions underlying each of the models are analyzed to determine their accuracy in modeling the behavior of software. Similarities and differences among the models are discussed. Reliability measures predicted by these models using published software failure data are presented.

Chapter 4 presents an analysis of each of the following failure-count software reliability models:

1. the Goel-Okumoto nonhomogeneous Poisson process model [21];
2. the Musa-Okumoto logarithmic Poisson execution time model [22,23];
3. the Musa execution time model [5,7].

Again, assumptions underlying each of the models are analyzed to determine their accuracy in modeling the behavior of software, and similarities and differences among the models are discussed. Reliability measures predicted by these models using published software failure data are presented.

To date, no single software reliability model has been shown to perform well in all contexts. Therefore, no definitive model can be recommended to potential users [24]. An appropriate model should be chosen based upon an understanding of the failure data characteristics of the software being modeled and an understanding of the assumptions underlying the models [1].

In order to determine the predictive validity of software reliability models and to compare software reliability models, reliability measures predicted by the models must be studied for a more extensive collection of software failure data sets than is available in the literature. Therefore, each of the models listed above has been implemented as a computer program. Each model being studied requires that a distribution of observed failures be known in order to make predictions for various reliability measures. If potential users do not collect failure data properly, then clearly none of these models may be used to make software reliability predictions. Chapter 5 presents a discussion of guidelines and recommendations for selecting a software reliability model based upon the failure data characteristics of the software being modeled. Outputs from the computer programs for a variety of failure data sets form the basis for the recommendations and guidelines given.

Chapter 6 summarizes results and conclusions.

CHAPTER 2

INTRODUCTORY RELIABILITY THEORY

Approaches for measuring software reliability parallel those used for measuring hardware reliability with appropriate modifications to account for the differences in the behavior of software and hardware [1]. For hardware reliability, there is a well established mathematical theory. Several mathematical models have been developed and shown to be applicable [25].

This chapter presents a discussion of reliability theory applicable to software reliability models. The motivation for the hazard function is developed, and expressions for reliability and MTTF in terms of the hazard function are derived. It will be shown that the exponential distribution plays an important role in reliability theory. Therefore, a brief discussion of the exponential distribution is also presented. Much of the material presented in this chapter is a summary of material presented by Myers [8], Apostol [26], Barlow and Proschan [27], and Martz and Waller [28].

2.1 Motivation for the Hazard Function

Recall from Chapter 1 that the reliability of a unit of age t , denoted by $R(t)$, is defined as the probability that no failure occurs in the time interval $[0, t]$.

Therefore, the probability that a failure does occur in the time interval $[0, t]$ is given by

$$F(t) = 1 - R(t). \quad (2.1)$$

It follows that

$$R(t) = 1 - F(t). \quad (2.2)$$

The function F is a distribution function of the random variable X , where the event X is a failure in this case. The value of F at any time is the probability that a failure has occurred at or before time t . Using distribution function notation,

$$F(t) = P(X \leq t). \quad (2.3)$$

Recall also from Chapter 1 that the hazard function, denoted by $z(t)$, is defined as the conditional probability that a failure occurs in the time interval $(t, t + \Delta t]$ given that no failure occurred in the time interval $[0, t]$. According to a distribution function theorem proved by Apostol (p. 515),

$$P(a < X \leq b) = F(b) - F(a) \quad \text{if } a < b. \quad (2.4)$$

Therefore, the probability that a failure occurs in the time interval $(t, t + \Delta t]$ is given by

$$F(t + \Delta t) - F(t). \quad (2.5)$$

The probability that no failure occurred in the time interval $[0, t]$ is simply $R(t)$. Using the definition of conditional probability, the hazard function over a time interval of duration Δt may be written as

$$z(t)\Delta t = \frac{F(t + \Delta t) - F(t)}{R(t)}. \quad (2.6)$$

Dividing both sides of (2.6) by Δt results in the equation

$$z(t) = \frac{1}{\Delta t} \frac{F(t + \Delta t) - F(t)}{R(t)}. \quad (2.7)$$

Taking the limit as Δt approaches zero of both sides of (2.7) allows the hazard function to represent an instantaneous failure rate and results in the equation

$$\lim_{\Delta t \rightarrow 0} z(t) = \lim_{\Delta t \rightarrow 0} \frac{1}{\Delta t} \frac{F(t + \Delta t) - F(t)}{R(t)}. \quad (2.8)$$

Since $z(t)$ does not depend upon Δt , (2.8) may be rewritten as

$$z(t) = \lim_{\Delta t \rightarrow 0} \frac{1}{\Delta t} \frac{F(t + \Delta t) - F(t)}{R(t)}. \quad (2.9)$$

A probability density function, denoted by $f(t)$, is defined as the derivative of a distribution function. Therefore, since

$$f(t) = F'(t), \quad (2.10)$$

the hazard function is given by

$$z(t) = \frac{f(t)}{R(t)}. \quad (2.11)$$

From (2.1), it follows that

$$F'(t) = -R'(t), \quad (2.12)$$

and then (2.11) may be rewritten as

$$z(t) = \frac{-R'(t)}{R(t)}. \quad (2.13)$$

Reliability as a function of the hazard function at an arbitrary time x may be derived by the following steps. First, integrating both sides of (2.13) results in the equation

$$\int_0^x z(t) dt = \int_0^x \frac{-R'(t)}{R(t)} dt. \quad (2.14)$$

Then,

$$\int_0^x z(t) dt = -\ln R(x) - [-\ln R(0)]. \quad (2.15)$$

Since $R(0)$ is defined to be 1, (2.15) may be rewritten as

$$\int_0^x z(t) dt = -\ln R(x). \quad (2.16)$$

Negating and exponentiating both sides of (2.16) gives the equation

$$R(x) = \exp \left[- \int_0^x z(t) dt \right]. \quad (2.17)$$

Notice that the formulation of the hazard function guarantees that $R(x)$ has an exponential distribution.

Mean time to failure is defined by expectation as

$$E(t) = \int_0^\infty t f(t) dt, \quad (2.18)$$

in which case, if $\lim_{t \rightarrow \infty} tR(t) = 0$, then a more convenient formulation is possible, namely

$$\text{MTTF} = \int_0^\infty R(x) dx. \quad (2.19)$$

2.2 The Exponential Distribution

A property of the exponential distribution which makes it especially important in reliability theory is that the remaining life of a used unit is independent of its age. Therefore, the hazard function $z(t)$ is independent of t for the exponential distribution; it is a positive constant, denoted by λ . A graphical

representation of the exponential distribution is shown in Figure 2-1. For the exponential distribution, from (2.17), reliability is given by

$$R(x) = e^{-\lambda x} \quad \text{for all } \lambda > 0, x \geq 0. \quad (2.20)$$

For the exponential distribution, from (2.19) and (2.20), MTTF is given by

$$\text{MTTF} = \frac{1}{\lambda}. \quad (2.21)$$

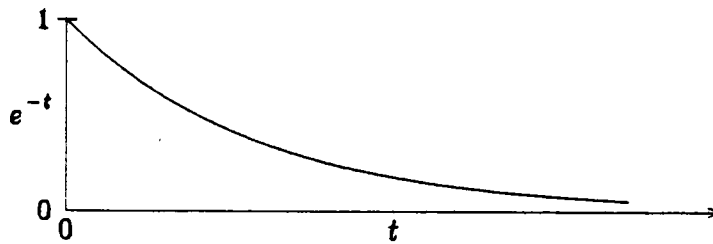


Figure 2-1: The exponential distribution.

CHAPTER 3

TIMES-BETWEEN-FAILURES MODELS

This chapter presents an analysis of the six times-between-failures software reliability models being studied. They are:

1. the Jelinski-Moranda de-eutrophication model;
2. the geometric de-eutrophication model;
3. the Schick-Wolverton model;
4. the Goel-Okumoto imperfect debugging model;
5. the Littlewood Bayesian differential debugging model;
6. the certification model.

Assumptions underlying each of the models presented in this chapter are analyzed to determine their accuracy in modeling the behavior of software. The mathematical expressions that are used to compute reliability measures for each of the models are derived from the assumptions that form the basis of the respective models. These mathematical expressions contain parameters (such as the total number of faults in a software system) that are required for the computation of the various reliability measures. The methods used to estimate such

parameters in the implementations of the models are given. For illustration purposes, reliability measures are presented for each of the models using a software failure data set that is widely used in the literature for analysis of software reliability models.

3.1 The Jelinski-Moranda De-Eutrophication Model

The Jelinski-Moranda model is credited with being the first reliability growth model specifically created for software [24]. It is the least complex of the models being studied. However, many of the assumptions upon which the Jelinski-Moranda model is based have been shown to be inaccurate in terms of modeling the behavior of software. The Jelinski-Moranda model therefore has limited applicability, but it may be a useful tool for obtaining general ideas about the reliability of software as long as its limitations are recognized [29].

3.1.1 Model Assumptions

The following assumptions form the basis of the Jelinski-Moranda model:

1. each of the N faults in a software system will cause a failure after a time which is distributed exponentially;
2. software faults are independent of each other and are equally likely to cause a failure;

3. the failure rate at any time is proportional to the current fault content of a software system;
4. a detected fault is corrected with certainty in a negligible amount of time and no new faults are introduced.

It has been shown in Chapter 2 that an exponential distribution of failures is a reasonable assumption.

The assumption that software faults, and therefore times between failures, are independent would be true if test cases are chosen randomly [1]. The degree to which this assumption is satisfied is dependent upon the degree to which the testing process is statistical. The time to the next failure may depend upon the nature of a previously detected fault, especially in the case of functional testing. Testing the reliability of a specified function performed by a software system is generally not a random process with respect to the entire software system. Therefore, strict adherence to this assumption is unlikely, but ensuring some degree of independence of faults is recommended [1]. It should be noted that all times-between-failures models make this assumption.

The assumptions that faults are equally likely to cause a failure and that the failure rate at any time is proportional to the current fault content of a software system are closely related; both are probably unrealistic. Littlewood [30] points out that a fault in a frequently executed portion of a software system will cause more failures than a fault in a rarely executed portion of a software system. Correcting a fault in a frequently executed portion of a software system will

therefore result in a greater reduction in the failure rate than correcting a fault in a rarely executed portion of a software system. In fact, recent studies indicate that the contributions of different faults to the overall failure rate of a software system may differ by orders of magnitude [24]. These two assumptions are the primary limitations of the Jelinski-Moranda model because the failure rate estimate has a direct bearing on the MTTF and reliability estimates.

The assumption that a detected fault is corrected in a negligible amount of time is clearly inaccurate. However, if the assumption is made that a detected fault is corrected with certainty, then the time required to correct the fault has no bearing on the resulting reliability growth. The assumption of negligible or instantaneous repair time is common in software reliability modeling.

The two assumptions that a detected fault is corrected with certainty and that no new faults are introduced during the correction process together ensure that the failure rate decreases monotonically. These appear to be inaccurate assumptions. However, recent work by Mills [31] suggests that many software faults are syntactic in nature as opposed to being semantic in nature. Since syntactic faults are theoretically easier to correct than semantic faults, these assumptions may be less inaccurate than has previously been thought. These two assumptions are common in software reliability modeling.

3.1.2 Model Development

The key Jelinski-Moranda model assumption is that the failure rate is proportional to the current fault content of a software system. The initial fault content of a software system is denoted by N and the proportionality constant is denoted by ϕ . Therefore, the failure rate is originally ϕN and remains constant at ϕN until the first fault is detected since an exponential distribution of faults is assumed. The failure rate decreases to $\phi(N - 1)$ after the first fault is detected and decreases by an amount ϕ after each successive fault is detected.

In general, the failure rate during t_i , the time interval between failures $(i - 1)$ and i , is given by

$$\lambda(t_i) = \phi(N - i + 1). \quad (3.1)$$

This process, referred to as “de-eutrophication” by Jelinski and Moranda in [4], is depicted in Figure 3-1. From (2.21), the MTTF during t_i is

$$\text{MTTF} = \frac{1}{\phi(N - i + 1)}. \quad (3.2)$$

From (2.20), the reliability for future elapsed time t' during t_i is

$$R(t') = e^{-\phi(N-i+1)t'}. \quad (3.3)$$

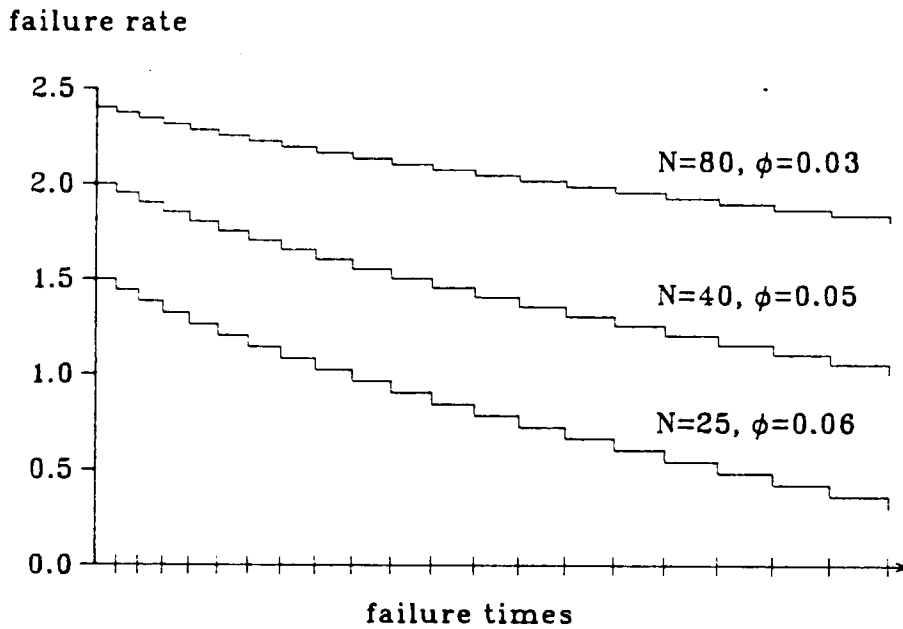


Figure 3-1: Jelinski-Moranda model failure rate process.

3.1.3 Parameter Estimation

If the failure rate λ is known at a certain time, then other reliability measures can be determined at that time. In order to compute an estimate for λ , estimates for N and ϕ must be computed. Methods to estimate parameters for software reliability models include the maximum likelihood method and the least squares method. Jelinski and Moranda use the maximum likelihood method to estimate N and ϕ .

The following discussion of the maximum likelihood method is a summary of material presented by Kreyszig [32]. Let X be a random variable whose probability density function depends upon certain parameters; in this case, N and ϕ . For a corresponding sample of n independent values $x_1, \dots, x_n$, the probability that a sample of size n consists of precisely those n values is

$$L = f(x_1)f(x_2)\dots f(x_n). \quad (3.4)$$

L is called the likelihood function. The x_i values are times between failures in the Jelinski-Moranda model.

In order to maximize L , approximations are chosen for N and ϕ for which L is as large as possible. If L is a differentiable function of N and ϕ , then necessary conditions for L to have a maximum are

$$\frac{\partial L}{\partial N} = 0, \quad \frac{\partial L}{\partial \phi} = 0. \quad (3.5)$$

For the Jelinski-Moranda model, the probability density function is given by

$$f(t_i) = \phi(N - i + 1)e^{-\phi(N-i+1)t_i} \quad (3.6)$$

from (3.3), (2.12), and (2.10). The likelihood function is then

$$L = \prod_{i=1}^n \phi(N - i + 1)e^{-\phi(N-i+1)t_i}, \quad (3.7)$$

where the t_i values are n observed times between failures. The logarithm of

L is maximized in order to change the product to a sum and to eliminate the exponential from the expression. The logarithm of the likelihood function is

$$\ln L = \sum_{i=1}^n \ln [\phi(N - i + 1)] - \sum_{i=1}^n \phi(N - i + 1)t_i. \quad (3.8)$$

The partial derivatives of the logarithm of the likelihood function are given by

$$\frac{\partial \ln L}{\partial N} = \sum_{i=1}^n \frac{1}{(N - i + 1)} - \phi \sum_{i=1}^n t_i \quad (3.9)$$

and

$$\frac{\partial \ln L}{\partial \phi} = \frac{n}{\phi} - \sum_{i=1}^n (N - i + 1)t_i. \quad (3.10)$$

The right hand sides of (3.9) and (3.10) are set equal to zero in order to maximize (3.8). From (3.10), this process yields

$$\phi = \frac{n}{\sum_{i=1}^n (N - i + 1)t_i}. \quad (3.11)$$

This value of ϕ is substituted into the right hand side of (3.9). When the right hand side of (3.9) is set equal to zero, the resulting equation is

$$\sum_{i=1}^n \frac{1}{N - i + 1} - \frac{n}{N - c} = 0, \quad (3.12)$$

where

$$c = \frac{\sum_{i=1}^n (i - 1)t_i}{\sum_{i=1}^n t_i}. \quad (3.13)$$

Since an analytical solution for N is not available for (3.12), it is solved numerically. The resulting estimate for N is substituted into (3.11) to determine the estimate for ϕ .

The Newton-Raphson method is used in the implementation of the Jelinski-Moranda model to solve (3.12) for N . Once estimates for N , and therefore ϕ , are computed at a given time, then failure rate, MTTF, and reliability estimates are easily computed at that time.

3.2 The Geometric De-Eutrophication Model

The geometric de-eutrophication model was proposed by Moranda as a variation of the Jelinski-Moranda model. The geometric de-eutrophication model relaxes the Jelinski-Moranda model assumptions that faults are equally likely to cause a failure and that the failure rate at any time is proportional to the current fault content of a software system.

3.2.1 Model Assumptions

The following assumptions form the basis of the geometric de-eutrophication model:

1. each of the infinite number of faults in a software system will cause a failure after a time which is distributed exponentially;

2. software faults are independent of each other;
3. the failure rate decreases geometrically after each fault is detected and corrected;
4. a detected fault is corrected with certainty in a negligible amount of time and no new faults are introduced.

The second and fourth assumptions are identical to Jelinski-Moranda model assumptions and were discussed in Section 3.1.1.

The assumption that the number of faults in a software system is infinite is based primarily upon large software systems. Moranda writes in [13] that "certainly this is so if there is an attempt to mirror in software all of the continuum of eventualities which occur in complex dynamic situations." However, a software system of finite length would seem to have a finite, although possibly very large, number of faults. Two important implications of this assumption are that it is difficult to assert that no faults remain in a software system and that the number of faults remaining in a software system has less significance than the failure rate of the software system.

The assumption that the failure rate decreases geometrically after each fault is detected and corrected takes into account that, on the average, faults detected early result in a greater reduction in the failure rate than faults detected later. As discussed in Section 3.1.1, this assumption is generally considered to be more

accurate than the Jelinski-Moranda model assumption that the failure rate is proportional to the current fault content of a software system.

3.2.2 Model Development

The initial failure rate in the geometric de-eutrophication model is denoted by D . The failure rate is D until the first fault is detected since an exponential distribution of faults is assumed. Since it is assumed that the failure rate decreases geometrically after each fault is detected, the failure rate decreases to Dk after the first fault is detected and decreases geometrically by an amount k after each successive fault is detected. The parameter k is in the range $0 < k < 1$, so that the successive failure rates form a converging geometric series.

In general, the failure rate during t_i , the time interval between failures $(i-1)$ and i , is given by

$$\lambda(t_i) = Dk^{i-1}. \quad (3.14)$$

This process is depicted in Figure 3-2. From (2.21), the MTTF during t_i is

$$\text{MTTF} = \frac{1}{Dk^{i-1}}. \quad (3.15)$$

From (2.20), the reliability for future elapsed time t' during t_i is

$$R(t') = e^{-Dk^{i-1}t'}. \quad (3.16)$$

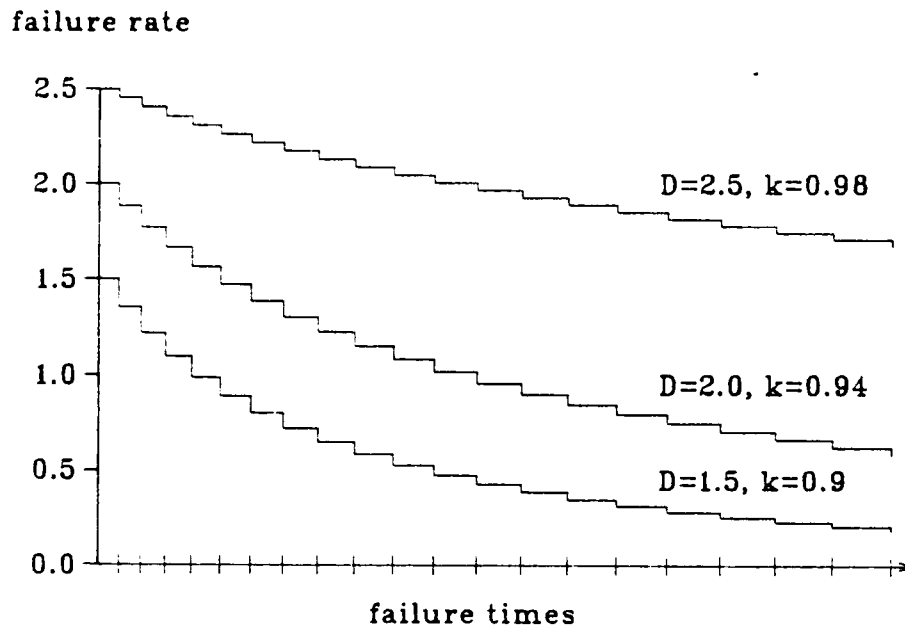


Figure 3-2: Geometric de-eutrophication model failure rate process.

3.2.3 Parameter Estimation

If the failure rate λ is known at a certain time, then other reliability measures can be determined at that time. In order to compute an estimate for λ , estimates for D and k must be computed. Moranda uses the maximum likelihood method to estimate D and k for the geometric de-eutrophication model.

For the geometric de-eutrophication model, the probability density function is given by

$$f(t_i) = Dk^{i-1}e^{-Dk^{i-1}t_i} \quad (3.17)$$

from (3.16), (2.12), and (2.10). The logarithm of the likelihood function is

$$\ln L = \sum_{i=1}^n \ln(Dk^{i-1}) - \sum_{i=1}^n Dk^{i-1}t_i, \quad (3.18)$$

where the t_i values are n observed times between failures. The partial derivatives of the logarithm of the likelihood function are given by

$$\frac{\partial \ln L}{\partial D} = \frac{n}{D} - \sum_{i=1}^n k^{i-1}t_i \quad (3.19)$$

and

$$\frac{\partial \ln L}{\partial k} = \frac{1}{k} \sum_{i=1}^n (i-1) - D \sum_{i=1}^n (i-1)k^{i-2}t_i. \quad (3.20)$$

The right hand sides of (3.19) and (3.20) are set equal to zero in order to maximize (3.18). From (3.19), this process yields

$$D = \frac{n}{\sum_{i=1}^n k^{i-1}t_i}. \quad (3.21)$$

This value of D is substituted into the right hand side of (3.20). When the right hand side of (3.20) is set equal to zero, the resulting equation is

$$\frac{1}{k} \sum_{i=1}^n (i-1) - \frac{n}{\sum_{i=1}^n k^{i-1} t_i} \sum_{i=1}^n (i-1) k^{i-2} t_i = 0. \quad (3.22)$$

Since an analytical solution for k is not available for (3.22), it is solved numerically. The resulting estimate for k is substituted into (3.21) to determine the estimate for D .

The Newton-Raphson method is used in the implementation of the geometric de-eutrophication model to solve (3.22) for k . Once estimates for k , and therefore D , are computed at a given time, then failure rate, MTTF, and reliability estimates are easily computed at that time.

3.3 The Schick-Wolverton Model

The Schick-Wolverton model proposes an alternative to the Jelinski-Moranda model assumption that the failure rate at any time is proportional to the current fault content of a software system. It is assumed in the development of the Schick-Wolverton model that the failure rate at any time is proportional to the current fault content of a software system and to the elapsed time since the last fault was detected. Although this may seem to be a minor variation to the Jelinski-Moranda model, it will be shown that it has a large effect on the formulation of the mathematical expressions that are used to compute reliability

measures and that it also has an effect on the assumed distribution of faults. The effect of this extension to the Jelinski-Moranda model illustrates a common problem in software reliability modeling; a minor extension to a particular model often results in a major increase in the complexity of the extended model.

3.3.1 Model Assumptions

The following assumptions form the basis of the Schick-Wolverton model:

1. the time between the occurrences of failures caused by each of the N faults in a software system has a Rayleigh distribution;
2. software faults are independent of each other and are equally likely to cause a failure;
3. the failure rate at any time is proportional to the current fault content of a software system and to the elapsed time since the last fault was detected;
4. a detected fault is corrected with certainty in a negligible amount of time and no new faults are introduced.

The second and fourth assumptions are identical to Jelinski-Moranda model assumptions and were discussed in Section 3.1.1.

The assumption that the time between the occurrences of failures has a Rayleigh distribution is dictated by the third assumption. This will be shown in the next section.

The assumption that the failure rate at any time is proportional to the current fault content of a software system and to the elapsed time since the last fault was detected is probably less realistic than the Jelinski-Moranda model assumption that the failure rate at any time is proportional only to the current fault content of a software system. Under the Schick-Wolverton model assumption, the failure rate is zero when a failure occurs, but the software is not performing reliably when a failure occurs. Additionally, as the elapsed time since the last failure increases, the failure rate increases linearly. However, since software does not deteriorate with age, the software is performing more reliably as the elapsed time since the last failure increases. In neither the case of no elapsed time since the last failure nor the case of increasing elapsed time since the last failure does the failure rate reflect intuitive perceptions of the reliability of a software system. This Schick-Wolverton model assumption would seem to be more applicable to hardware than to software.

3.3.2 Model Development

The key Schick-Wolverton model assumption is that the failure rate is proportional to the current fault content of a software system and to the elapsed time since the last fault was detected. The initial fault content of a software system is denoted by N and the proportionality constant is denoted by ϕ . The elapsed time since failure $(i - 1)$ is denoted by t_i . The failure rate is initially zero

and is $\phi N t_1$ until the first fault is detected. The failure rate increases linearly with slope ϕN as t_1 increases. When the first fault is detected, the failure rate returns to zero. The failure rate is then $\phi(N - 1)t_2$ until the second fault is detected. The failure rate increases linearly with slope $\phi(N - 1)$ between the detection of the first and second faults as t_2 increases. The failure rate returns to zero when each successive fault is detected and increases linearly with a slope that decreases by an amount ϕ between the detection of each successive fault as t_i increases.

In general, the failure rate at t_i , the elapsed time since the last fault was detected, is given by

$$\lambda(t_i) = \phi(N - i + 1)t_i. \quad (3.23)$$

This process is depicted in Figure 3-3. The reliability for future elapsed time t' at t_i is

$$R(t') = e^{-\phi(N-i+1)\frac{t'^2}{2}}. \quad (3.24)$$

Note that the Schick-Wolverton model assumptions force the derived reliability model to have a form different than the exponential distribution given in (2.20). The reliability model given in (3.24) is known as the Rayleigh distribution.

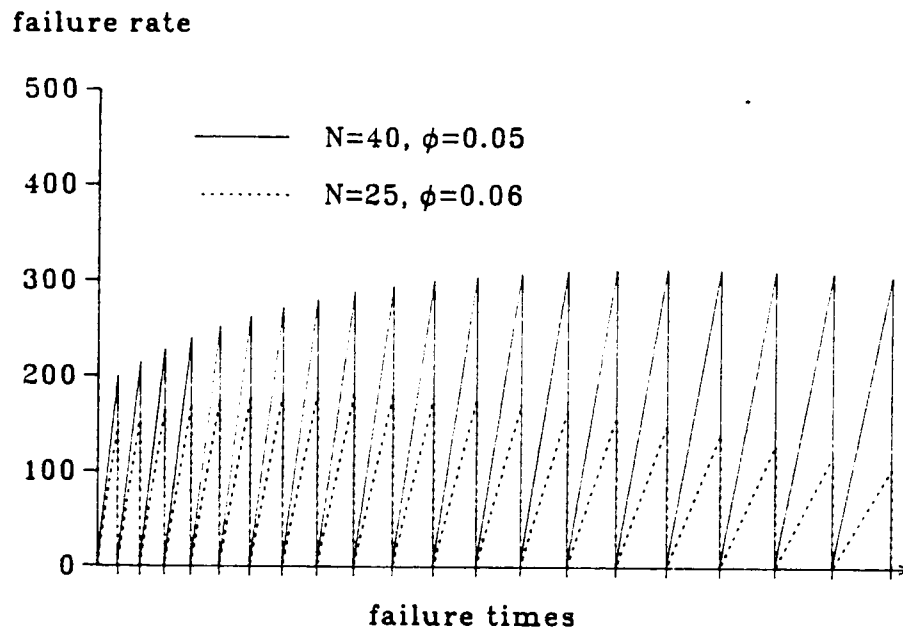


Figure 3-3: Schick-Wolverton model failure rate process.

Since failures are not distributed exponentially, the MTTF at t_i cannot be expressed as the reciprocal of the failure rate. Sukert [33] gives the MTTF at t_i as

$$\text{MTTF} = \left[\frac{\pi}{2\phi(N - i + 1)} \right]^{1/2}. \quad (3.25)$$

3.3.3 Parameter Estimation

If the failure rate λ is known at a certain time, then other reliability measures can be determined at that time. In order to compute an estimate for λ , estimates for N and ϕ must be computed. Schick and Wolverson use the maximum likelihood method to estimate N and ϕ .

For the Schick-Wolverson model, the probability density function is given by

$$f(t_i) = \phi(N - i + 1)t_i e^{-\phi(N-i+1)\frac{t_i^2}{2}} \quad (3.26)$$

from (3.24), (2.12), and (2.10). The logarithm of the likelihood function is

$$\ln L = \sum_{i=1}^n \ln [\phi(N - i + 1)t_i] - \sum_{i=1}^n \phi(N - i + 1)\frac{t_i^2}{2}, \quad (3.27)$$

where the t_i values are n observed times between failures. The partial derivatives of the logarithm of the likelihood function are given by

$$\frac{\partial \ln L}{\partial N} = \sum_{i=1}^n \frac{1}{N - i + 1} - \frac{\phi}{2} \sum_{i=1}^n t_i^2 \quad (3.28)$$

and

$$\frac{\partial \ln L}{\partial \phi} = \frac{n}{\phi} - \frac{1}{2} \sum_{i=1}^n (N - i + 1)t_i^2. \quad (3.29)$$

The right hand sides of (3.28) and (3.29) are set equal to zero in order to maximize (3.27). From (3.29), this process yields

$$\phi = \frac{2n}{\sum_{i=1}^n (N - i + 1)t_i^2}. \quad (3.30)$$

This value of ϕ is substituted into the right hand side of (3.28). When the right hand side of (3.28) is set equal to zero, the resulting equation is

$$\sum_{i=1}^n \frac{1}{N - i + 1} - \frac{n}{N - c} = 0, \quad (3.31)$$

where

$$c = \frac{\sum_{i=1}^n (i - 1)t_i^2}{\sum_{i=1}^n t_i^2}. \quad (3.32)$$

Equation (3.31) is close to (3.12) for the Jelinski-Moranda model. The only difference is that the t_i values are squared in the formulation of a software system model constant, whereas the t_i values are not squared in the formulation of the Jelinski-Moranda model constant. Since an analytical solution for N is not available for (3.31), it is solved numerically. The resulting estimate for N is substituted into (3.30) to determine the estimate for ϕ .

The Newton-Raphson method is used in the implementation of the Schick-Wolverton model to solve (3.31) for N . Once estimates for N , and therefore ϕ , are computed at a given time, then failure rate, MTTF, and reliability estimates are easily computed at that time.

3.4 The Goel-Okumoto Imperfect Debugging Model

The Goel-Okumoto imperfect debugging model proposes an alternative to the Jelinski-Moranda model assumption that a detected fault is corrected with certainty. It is assumed in the development of the imperfect debugging model that a detected fault is corrected with probability p and that a detected fault is not corrected with probability $q = 1 - p$.

3.4.1 Model Assumptions

The following assumptions form the basis of the Goel-Okumoto imperfect debugging model:

1. each of the N faults in a software system will cause a failure after a time which is distributed exponentially;
2. software faults are independent of each other and are equally likely to cause a failure;
3. the failure rate at any time is proportional to the current fault content of a software system;
4. a detected fault is corrected with probability p in a negligible amount of time and a detected fault is not corrected with probability $q = 1 - p$;
5. no new faults are introduced during the correction process.

The first, second, third, and fifth assumptions are identical to Jelinski-Moranda model assumptions and were discussed in Section 3.1.1.

The assumption that a detected fault may not be corrected is more accurate than the Jelinski-Moranda model assumption that a detected fault is corrected with certainty. Obviously, the nature of faults will vary widely and it is no more realistic to assume that all faults are corrected with certainty than it is to assume that a software system initially contains no faults. However, the assumption that no new faults are introduced during the correction process remains overly optimistic, and the assumption that all faults are equally likely to be corrected is inaccurate.

Computation of the perfect debugging probability p requires that a recurrent fault be distinguished from a previously undetected fault. The assumption that a recurrent fault is distinguished from a previously undetected fault may also be overly optimistic. The ability to distinguish a recurrent fault from a previously undetected fault clearly has a large effect on the accuracy of the estimate for p .

3.4.2 Model Development

As is the case for the Jelinski-Moranda model, the initial fault content of a software system is denoted by N and the proportionality constant is denoted by ϕ in the imperfect debugging model. The failure rate is originally ϕN and

remains constant at ϕN until the first fault is detected since an exponential distribution of faults is assumed.

After the first fault is detected, the failure rate decreases to $\phi(N - 1)$ with probability p and remains ϕN with probability $q = 1 - p$. After each successive fault is detected, the failure rate decreases by an amount ϕ with probability p and remains unchanged with probability q . A trivial example of this process is presented in Figure 3-4 for a software system for which two faults remain to be detected and an attempt is made to correct each of the two faults.

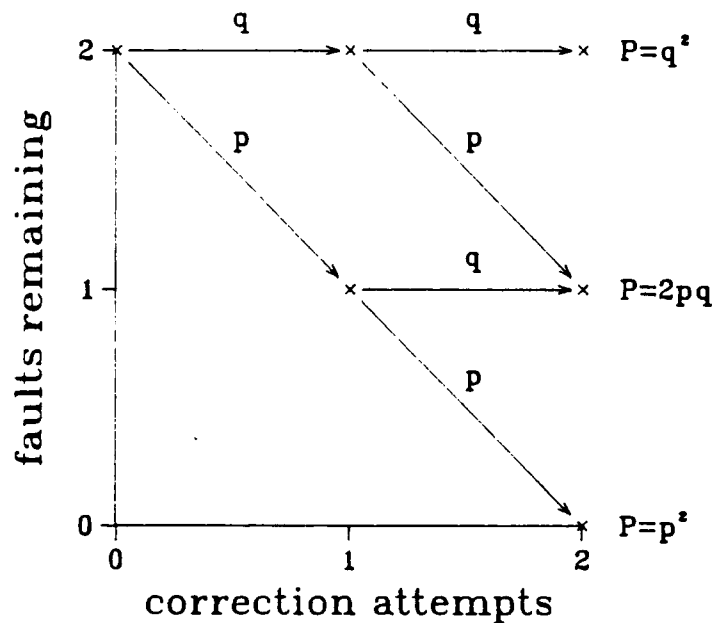


Figure 3-4: Illustration of the imperfect debugging process.

For this example, the probability that both faults are corrected is p^2 . For the general case where N faults have been detected, the probability that every fault is corrected is p^N . Note that if p is 1, then every fault is corrected with certainty and the imperfect debugging model reduces to the Jelinski-Moranda model.

The probability p^N is of interest for two reasons. First, the accuracy of the probability p^N is dependent upon the accuracy of the estimate for p , illustrating the importance of an accurate estimate for p . For example, for a software system for which N is 90, if p is 0.95, then the probability that every fault is corrected is less than 1 percent according to the imperfect debugging model. If p is 0.99, then the probability that every fault is corrected jumps to greater than 40 percent according to the imperfect debugging model. Clearly, small differences in the estimate for p may result in large differences in the probability that every fault is corrected.

Second, even for values of p approaching 1, the probability that every fault is corrected decreases as N increases. Therefore, it is likely that faults are remaining in a software system after an attempt is made to correct each of the N faults.

It will be shown in Chapter 4 that the expected number of failures at time t is given by

$$m(t) = a(1 - e^{-bt}), \quad (3.33)$$

where a is the expected number of software faults to eventually be detected, including recurrent faults resulting from imperfect debugging, and b is a proportionality constant.

Then,

$$a = \frac{N}{p} \quad (3.34)$$

and

$$b = p\phi. \quad (3.35)$$

Note that, on the average, the number of faults is reduced by p when a fault is detected. However, according to the model assumptions, when a fault is detected, it is either corrected or not corrected. Therefore, reducing the number of faults by p between any two successive faults is not accurate.

The failure rate at any time is the derivative of the number of failures with respect to time. From (3.33),

$$\lambda(t) = N\phi e^{-p\phi t}. \quad (3.36)$$

The failure rate process is illustrated in Figure 3-5. From (2.21), the MTTF is

$$\text{MTTF} = \frac{1}{N\phi e^{-p\phi t}}. \quad (3.37)$$

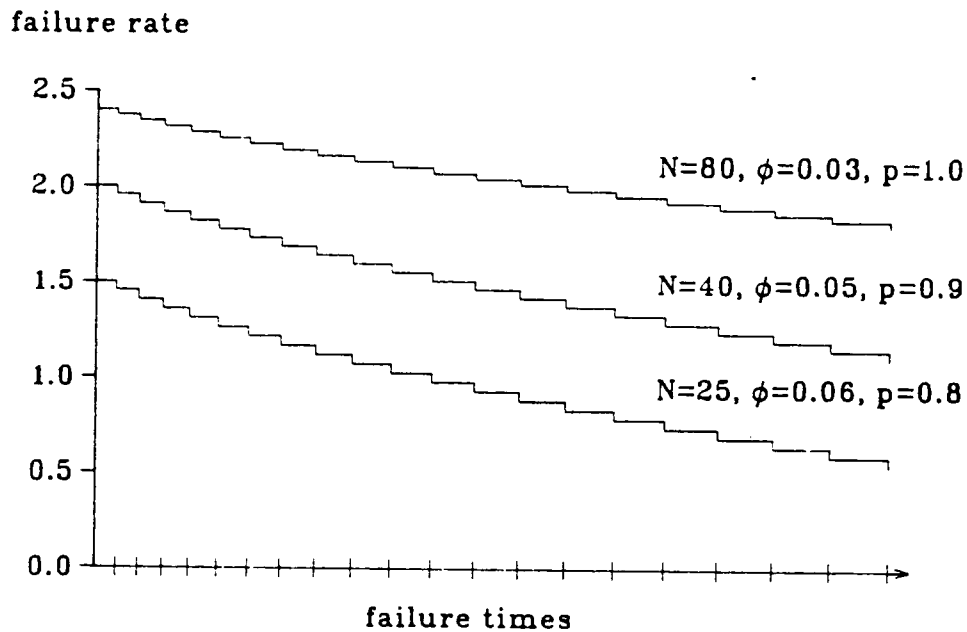


Figure 3-5: Imperfect debugging model failure rate process.

Goel and Okumoto estimate the reliability for future elapsed time t' during the time interval between failures $(i - 1)$ and i by

$$R(t') = e^{-\phi[N-p(i-1)]t'}. \quad (3.38)$$

Note that if p is 1, then (3.38) reduces to the Jelinski-Moranda model reliability estimate given by (3.3).

3.4.3 Parameter Estimation

If N , ϕ , and p are known at a certain time, then other reliability measures can be determined at that time. Goel and Okumoto use nonlinear regression analysis to estimate a and b . Estimates for N and ϕ are then determined from the estimates for a , b , and p .

An estimate for p may be determined using one of three methods. One method is to determine p from previous similar software system projects. This method results in an estimate for p whose accuracy is dependent upon the degree of similarity between the software system projects and the accuracy of the estimates for p for the previous projects.

Two other methods for estimating p require a recurrent fault to be distinguished from a previously undetected fault. Let u_i be the total number of faults detected at t_i , the time to i failures, and let v_i be the number of recurrent faults detected at t_i . An estimate for p is then

$$p = 1 - \frac{v_n}{n}, \quad (3.39)$$

where n failures are observed. A weighted estimate for p that takes into account when recurrent faults are detected during the fault correction process is given by

$$p = 1 - \frac{\sum_{i=1}^n v_i / u_i}{n}. \quad (3.40)$$

Recurrent faults detected early in the fault correction process reduce the estimate for p more than recurrent faults detected later in the fault correction process if an estimate for p is determined using (3.40).

The least squares method is used to determine estimates for a and b . The number of failures observed at t_i is

$$x_i = i. \quad (3.41)$$

The regression equation is then

$$x_i = a(1 - e^{-bt_i}) + \epsilon_i, \quad (3.42)$$

where ϵ_i is the distance from the point (x_i, t_i) to the model.

Estimates for a and b are determined by using a SLATEC Mathematical Subroutine Library [34] nonlinear regression analysis routine to minimize the sum of the ϵ_i^2 terms in the implementation of the Goel-Okumoto imperfect debugging model. Once estimates for a , b , and p , and therefore N and ϕ , are determined at a given time, then failure rate, MTTF, and reliability estimates are easily computed at that time.

3.5 The Littlewood Bayesian Differential Debugging Model

The Littlewood Bayesian differential debugging model proposes two alternatives to the Jelinski-Moranda model. The Littlewood Bayesian differential

debugging model relaxes the Jelinski-Moranda model assumptions that faults are equally likely to cause a failure and that the failure rate at any time is proportional to the current fault content of a software system. Additionally, the Littlewood Bayesian differential debugging model allows the failure rate to change during the interval between failures. Bayes' theorem is used to describe how the user or tester perceives the failure rate of a software system changes even when no changes are made to the software system. In particular, confidence in the software system increases as time passes without a failure.

3.5.1 Model Assumptions

The following assumptions form the basis of the Littlewood Bayesian differential debugging model:

1. each of the N faults in a software system will cause a failure after a time which is distributed exponentially;
2. software faults are independent of each other;
3. a detected fault is corrected with certainty in a negligible amount of time and no new faults are introduced;
4. the failure rate of faults remaining in the software system, denoted by Φ , are independently and identically distributed random variables;

5. after n faults are detected and corrected, the failure rate of the software system is

$$\Lambda = \sum_{i=1}^{N-n} \Phi_i; \quad (3.43)$$

6. when debugging of the software system begins, the distribution of the failure rates associated with each fault is assumed to be the gamma distribution with parameters α and β .

The first, second, and third assumptions are identical to Jelinski-Moranda model assumptions and were discussed in Section 3.1.1. The fourth, fifth, and sixth assumptions are closely related. The assumption that failure rates associated with faults are independent of each other is consistent with the assumption that faults are independent of each other. Additionally, Martz and Waller write in [28] that the gamma distribution is "often used as a prior distribution in Bayesian reliability analysis" because of the flexibility of the parameters α and β .

3.5.2 Model Development

Bayes' theorem provides a means of repeatedly updating reliability measures as additional failure data are observed for a software system. Initially, a given distribution of failure rates is assumed. This distribution is called the prior distribution. Over a given time period, additional failure data are observed.

These failure data change how the user or tester perceives the distribution of failure rates for the software system. Therefore, an updated distribution of failure rates is formed from the prior distribution and the additional observed failure data. This updated distribution is called the posterior distribution. The posterior distribution then becomes the new prior distribution, additional failure data are again observed, and a new posterior distribution is determined. The prior and posterior distributions may be updated repeatedly in this manner as additional failure data are observed.

Bayes' theorem is formally stated as follows. If X is a continuous random variable whose probability density function depends upon θ so that the conditional probability density function of X , given θ , is $f(x|\theta)$, and if the prior probability density function of Θ is $g(\theta)$, then for every x such that $f(x) > 0$ exists, the posterior probability density function of Θ , given $X = x$, is

$$g(\theta|x) = \frac{f(x|\theta)g(\theta)}{f(x)}, \quad (3.44)$$

where

$$f(x) = \int f(x|\theta)g(\theta) d\theta. \quad (3.45)$$

Prior and posterior probability density functions are used to represent beliefs concerning the likelihood of various values of Θ before and after the observation of values of X .

From the fourth and fifth assumptions, when n faults have been detected and corrected at total elapsed time τ , the failure rate of the software system is

$$\lambda = \sum_{i=1}^{N-n} \phi_i, \quad (3.46)$$

where λ and ϕ values are realizations of the random variables Λ and Φ , respectively. If the ϕ values are known, then the current reliability of the software system is also known since an exponential distribution of failures is assumed. However, the ϕ values are not known. The uncertainty about the ϕ values is modeled by treating the failure rate of each of the $(N - n)$ remaining faults in the software system as a random variable with a probability density function. The failure rate of the software system is then a random variable

$$\Lambda = \sum_{i=1}^{N-n} \Phi_i \quad (3.47)$$

with the probability density function

$$f(t|\Lambda = \lambda) = \lambda e^{-\lambda t}, \quad (3.48)$$

where t is future elapsed time.

Bayes' theorem is now used to describe how the uncertainty of the failure rate associated with each of the $(N - n)$ faults remaining in the software system

changes over a time period beginning at a given time. From (3.44), the failure rate of one of the $(N - n)$ faults remaining in the software system has the probability density function

$$f(\phi|\text{no failure in } (0, \tau)) = \frac{P(\text{no failure in } (0, \tau)|\Phi = \phi)f(\phi)}{\int P(\text{no failure in } (0, \tau)|\Phi = \phi)f(\phi) d\phi}. \quad (3.49)$$

Equation (3.49) may be rewritten as

$$f(\phi|\text{no failure in } (0, \tau)) = \frac{e^{-\phi\tau}f(\phi)}{\int e^{-\phi\tau}f(\phi) d\phi}. \quad (3.50)$$

From the sixth assumption, the probability density function of the failure rate of this fault at time τ is

$$f(\phi) = \frac{(\beta + \tau)^\alpha \phi^{\alpha-1} e^{-(\beta+\tau)\phi}}{\Gamma(\alpha)}, \quad (3.51)$$

where $\Gamma(\alpha)$ is the gamma function of α . From the fourth and fifth assumptions, the failure rate of the software system, Λ , is the gamma distribution with parameters $(N - n)\alpha$ and $(\beta + \tau)$. From (3.45), (3.48), and (3.51),

$$f(t) = \int_0^\infty \frac{\lambda e^{-\lambda t} (\beta + \tau)^{(N-n)\alpha} \lambda^{(N-n)\alpha-1} e^{-(\beta+\tau)\lambda}}{\Gamma[(N - n)\alpha]} d\lambda. \quad (3.52)$$

Equation (3.52) may be rewritten as

$$f(t) = \int_0^\infty \frac{[(\beta + \tau)\lambda]^{(N-n)\alpha} e^{-(\beta+\tau+t)\lambda}}{\Gamma[(N - n)\alpha]} d\lambda. \quad (3.53)$$

Substituting $y = \lambda(\beta + \tau + t)$ into (3.53),

$$f(t) = \int_0^\infty \frac{\left[\frac{y(\beta+\tau)}{\beta+\tau+t}\right]^{(N-n)\alpha} e^{-y}}{\Gamma[(N-n)\alpha]} \frac{dy}{\beta + \tau + t}. \quad (3.54)$$

Simplifying,

$$f(t) = \frac{(\beta + \tau)^{(N-n)\alpha} \int_0^\infty y^{(N-n)\alpha} e^{-y} dy}{(\beta + \tau + t)^{(N-n)\alpha+1} \Gamma[(N-n)\alpha]}. \quad (3.55)$$

Since

$$\int_0^\infty y^{(N-n)\alpha} e^{-y} dy = \Gamma[(N-n)\alpha + 1], \quad (3.56)$$

equation (3.55) may be rewritten as

$$f(t) = \frac{(N-n)\alpha(\beta + \tau)^{(N-n)\alpha}}{(\beta + \tau + t)^{(N-n)\alpha+1}} \quad (3.57)$$

From (3.57), (2.10), and (2.12), the reliability for future elapsed time t at time τ is

$$R(t) = \left(\frac{\beta + \tau}{\beta + \tau + t} \right)^{(N-n)\alpha}. \quad (3.58)$$

From (2.13) and (3.58), the failure rate function of (3.57) is

$$\lambda(t) = \frac{(N-n)\alpha}{\beta + \tau + t}. \quad (3.59)$$

As time passes without a failure, confidence in the software system increases and the failure rate is perceived to be decreasing. When a failure occurs, confidence

in the software system decreases and the failure rate is perceived to be increasing. When the fault causing the failure is removed, confidence in the software system again increases and the failure rate is again perceived to be decreasing. This process is illustrated in Figure 3-6. Note that as long as the decrease in confidence in the software system resulting from the occurrence of a failure is smaller than the increase in confidence in the software system resulting from the removal of the fault causing the failure, the failure rate of the software system is perceived to be generally decreasing as time passes.

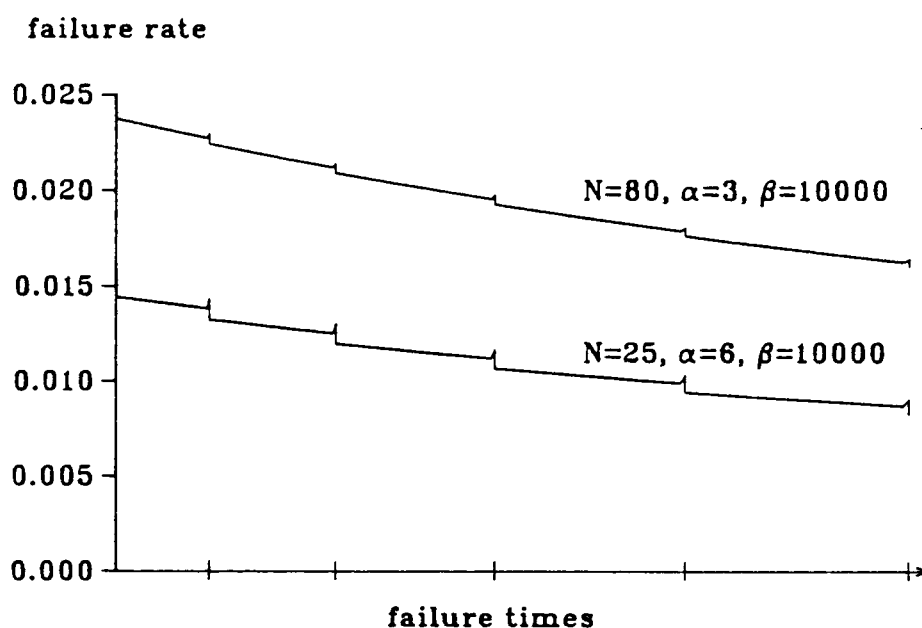


Figure 3-6: Littlewood model failure rate process.

Littlewood writes in several papers that while the number of faults predicted to eventually occur in a software system may be of interest to a software manager, the most important reliability measure is the distribution of future times between failures [17,18,30,35]. Additionally, MTTF may be infinite at certain times for models that predict the total number of faults to eventually occur in a software system. Therefore, MTTF predictions are not given for the Littlewood model.

In addition to the current failure rate and reliability, the failure rate after i additional failures may also be calculated. The failure rate after i additional failures is given by

$$\lambda = \frac{(N - n - i)\alpha}{\beta + \tau} \prod_{j=1}^i \frac{(N - n - i + j)\alpha}{(N - n - i + j)\alpha + 1}. \quad (3.60)$$

Note that the value of α determines the degree to which the failure rate after predicted failures differs from the current failure rate given by (3.59). In particular, if α is large compared to $(N - n - i + j)$, then the failure rate at each predicted failure will be only slightly smaller than the failure rate at the previous failure.

3.5.3 Parameter Estimation

If N , α , and β are known at a certain time, then other reliability measures can be determined at that time. Littlewood uses the maximum likelihood method to estimate N , α , and β .

For the Littlewood Bayesian differential debugging model, the probability density function during the time interval between failures $(i - 1)$ and i is

$$f(t) = \frac{(N - i + 1)\alpha(\beta + \tau)^{(N-i+1)\alpha}}{(\beta + \tau + t)^{(N-i+1)\alpha+1}} \quad (3.61)$$

from (3.57). The logarithm of the likelihood function is

$$\begin{aligned} \log L = \sum_{i=1}^n & \left[\log(N - i + 1) + \log \alpha + (N - i + 1)\alpha \log(\beta + \tau) \right. \\ & \left. - [(N - i + 1)\alpha + 1] \log(\beta + \tau + t) \right]. \end{aligned} \quad (3.62)$$

The partial derivatives of the logarithm of the likelihood function are given by

$$\frac{\partial \log L}{\partial N} = \sum_{i=1}^n \left[\frac{1}{(N - i + 1)} + \alpha \log \left(\frac{\beta + \tau}{\beta + \tau + t} \right) \right] \quad (3.63)$$

and

$$\frac{\partial \log L}{\partial \alpha} = \frac{n}{\alpha} + \sum_{i=1}^n \left[(N - i + 1) \log \left(\frac{\beta + \tau}{\beta + \tau + t} \right) \right] \quad (3.64)$$

and

$$\frac{\partial \log L}{\partial \beta} = \sum_{i=1}^n \left[\frac{(N - i + 1)\alpha}{\beta + \tau} - \frac{(N - i + 1)\alpha + 1}{\beta + \tau + t} \right]. \quad (3.65)$$

The right hand sides of (3.63), (3.64), and (3.65) are set equal to zero in order to maximize (3.62). From (3.64), this process yields

$$\alpha = -\frac{n}{\sum_{i=1}^n \left[(N - i + 1) \log \left(\frac{\beta + \tau}{\beta + \tau + t} \right) \right]}. \quad (3.66)$$

This value of α is substituted into the right hand sides of (3.63) and (3.65).

The resulting equations form a system of two unknowns in two nonlinear equations. Since an analytical solution for the system of equations is not available, a solution is obtained numerically. Estimates for N and β are determined by using a SLATEC Mathematical Subroutine Library routine to solve the equations for N and β in the implementation of the Littlewood Bayesian differential debugging model. The resulting estimates for N and β are substituted into (3.64) to determine the estimate for α . Once estimates for N , α , and β are computed at a given time, then failure rate and reliability estimates are easily computed at that time.

3.6 The Certification Model

The certification model proposed by Currit, Dyer, and Mills is equivalent in form to the geometric de-eutrophication model. Its significance lies in statistical inference used to produce certified reliability measure estimates for a software system. Such statistical inferences may be applied because it is assumed that the software system is developed under statistical control and because standard

linear regression is used to estimate the parameters which, in turn, determine estimates of reliability measures.

3.6.1 Model Assumptions

The following assumptions form the basis of the certification model:

1. each of the faults in a software system will cause a failure after a time which is distributed exponentially;
2. software faults are independent of each other;
3. the MTTF increases geometrically after each fault is detected and corrected;
4. the software system is tested under statistical control.

Additionally, a parameter to model the introduction of failures during the correction process is introduced in [19]. However, no basis is given for its estimation therein. Therefore, a fifth assumption made here for the certification model is that a detected fault is corrected with certainty in a negligible amount of time and no new faults are introduced.

The first, second, and fifth assumptions are identical to Jelinski-Moranda model assumptions and were discussed in Section 3.1.1. The third assumption is equivalent to the third geometric de-eutrophication model assumption, which was discussed in Section 3.2.1.

The assumption that statistical testing is used for software system testing is not unreasonable, but statistical testing of a software system appears to be inefficient compared to selective testing. However, if a software development method such as the cleanroom method described in [20] is used and the software system being tested is assumed to be of high quality, then statistical testing will be far more effective than selective testing. Statistical testing tends to uncover faults in the same order as their seriousness, while selective testing uncovers faults without regard to their seriousness [19].

3.6.2 Model Development

The initial MTTF of a software system is denoted by M in the certification model. The MTTF remains constant at M until the first fault is detected since an exponential distribution of faults is assumed. Since it is assumed that the MTTF increases geometrically after each fault is detected, the MTTF increases to Mr after the first fault is detected and increases geometrically by an amount r after each successive fault is detected. The parameter r is the average fractional improvement to the MTTF resulting from the correction of a detected fault.

In general, the MTTF during t_i , the time interval between failures $(i - 1)$ and i , is given by

$$\text{MTTF}(t_i) = Mr^{i-1}. \quad (3.67)$$

From (2.21), the failure rate during t_i is

$$\lambda(t_i) = \frac{1}{Mr^{i-1}}. \quad (3.68)$$

To remain consistent with plots previously presented in this chapter, the failure rate process is depicted in Figure 3-7. From (2.20), the reliability for future elapsed time t' during t_i is

$$R(t') = e^{-\frac{t'}{Mr^{i-1}}}. \quad (3.69)$$

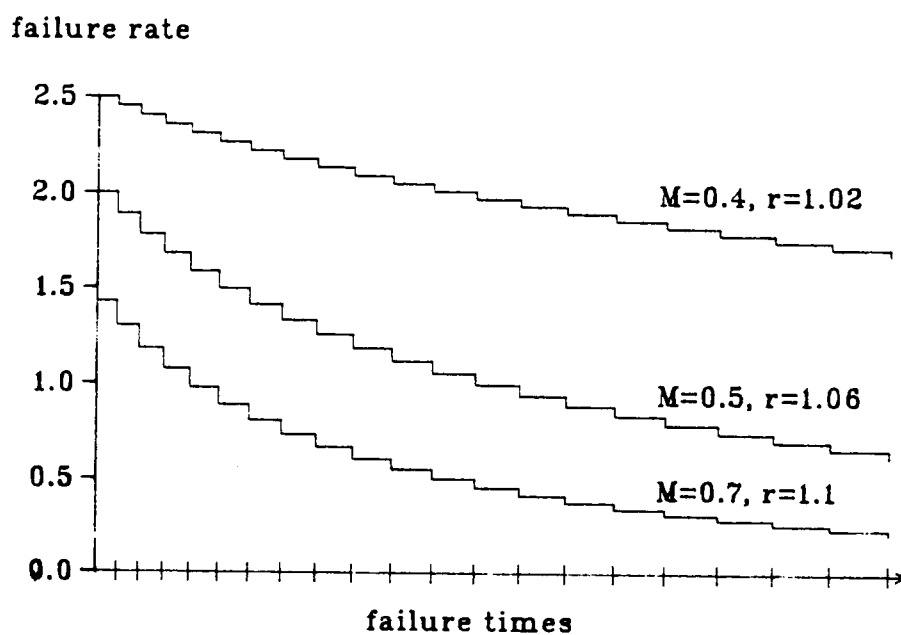


Figure 3-7: Certification model failure rate process.

Note that the certification model does not attempt to predict the initial fault content of a software system. The following rationale for not predicting the initial fault content of a software system is given in [19]:

Not only is the number of defects academic to users, there is no real management reason for it to be either high or low. If the number is high the defect removal process can be seen to be working, but the number of defects remaining may be high, as well. If the number is low, it is not obvious whether the defect removal process has broken down or there are few defects to find.

3.6.3 Parameter Estimation

If M and r are known at a certain time, then other reliability measures can be determined at that time. Currit, Dyer, and Mills use the least squares method to determine estimates for M and r .

From (3.67), the time between failures $(i - 1)$ and i is estimated by the equation

$$t_i = Mr^{i-1}. \quad (3.70)$$

Taking the logarithm of each side of (3.70),

$$\ln t_i = \ln M + (i - 1) \ln r. \quad (3.71)$$

Equation (3.71) may be used to perform standard linear regression analysis with

$$y_i = \ln t_i \quad (3.72)$$

and

$$x_i = i - 1, \quad (3.73)$$

where the t_i values are n observed times between failures. The regression analysis yields estimates for $\ln M$ and $\ln r$, but M and r may be computed once $\ln M$ and $\ln r$ are known.

The standard linear regression equations are used in the implementation of the certification model to determine estimates for $\ln M$ and $\ln r$ from (3.71). The estimate for M is computed as $\exp(\ln M)$ and the estimate for r is computed as $\exp(\ln r)$. Once estimates for M and r are computed at a given time, estimates for the MTTF, failure rate, and reliability at that time are easily computed.

3.7 Example Reliability Measure Calculations

For the purpose of illustration, reliability measures computed using the Jelinski-Moranda model, the geometric de-eutrophication model, the Schick-Wolverton model, the Goel-Okumoto imperfect debugging model, the Littlewood Bayesian differential debugging model, and the certification model are now presented in graphical form. Reliability measures are computed for a failure data set that consists of 136 times between failures, in CPU seconds, for software performing the function of a real time command and control system. This failure data set was originally analyzed by Musa in [5] and has since been

widely used for analysis of software reliability models. It is commonly referred to as the "Musa failure data."

The Musa failure data are listed in tabular form in Table 3-1. The Musa failure data are presented in graphical form in Figure 3-8. The number of detected faults is plotted against the elapsed time for the detected faults. Since the times between failures generally increase as the number of detected faults increases, the Musa failure data set exhibits general reliability growth.

Table 3-1: Musa failure data times (CPU seconds).

Failures	Times									
1-10	3	30	113	81	115	9	2	91	112	15
11-20	138	50	77	24	108	88	670	120	26	114
21-30	325	55	242	68	422	180	10	1146	600	15
31-40	36	4	0	8	227	65	176	58	457	300
41-50	97	263	452	255	197	193	6	79	816	1351
51-60	148	21	233	134	357	193	236	31	369	748
61-70	0	232	330	365	1222	543	10	16	529	379
71-80	44	129	810	290	300	529	281	160	828	1011
81-90	445	296	1755	1064	1783	860	983	707	33	868
91-100	724	2323	2930	1461	843	12	261	1800	865	1435
101-110	30	143	109	0	3310	1247	943	700	875	245
111-120	729	1897	447	386	446	122	990	948	1082	22
121-130	75	482	5509	100	10	1071	371	790	6150	3321
131-136	1045	648	5485	1160	1864	4116				

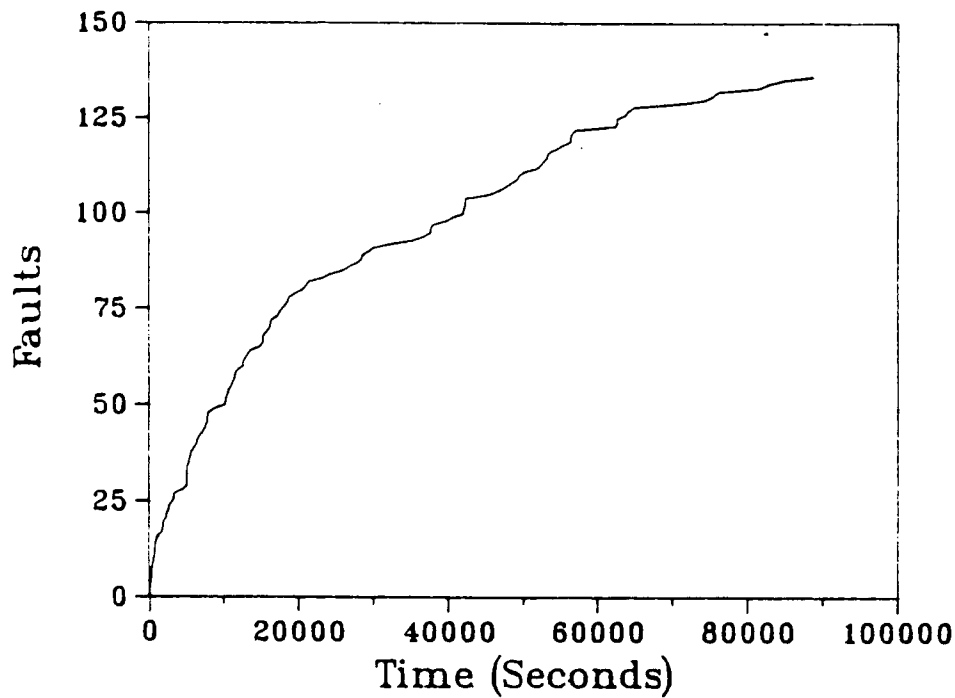


Figure 3-8: Musa failure data distribution.

Reliability measures are computed for each of the six times-between-failures models at time-values ranging from 2 percent of the total reported time to 100 percent of the total reported time in increments of 2 percent. Therefore, there are 50 time-values at which reliability measures are computed for each of the six models. At each value, only the failure data observed up to that time are used in the reliability measure calculations. Plots are presented for

two reliability measures, the failure rate and the percentage of faults detected. The percentage of faults detected is defined to be the number of detected faults divided by the total number of faults predicted to eventually occur. For the geometric de-eutrophication model, the percentage of faults detected during the time interval between failures $(i - 1)$ and i is given by $1 - k^{i-1}$, where k is the converging geometric series term. Descriptions of the plots illustrating reliability measures computed for the Musa failure data using the models presented in this chapter are given in Table 3-2.

Table 3-2: Illustrated reliability measures, Figures 3-9 through 3-19.

Figure	Model	Reliability Measure
3-9	Jelinski-Moranda	failure rates
3-10	Jelinski-Moranda	percentages of faults detected
3-11	geometric de-eutrophication	failure rates
3-12	geometric de-eutrophication	percentages of faults detected
3-13	Schick-Wolverton	failure rates
3-14	Schick-Wolverton	percentages of faults detected
3-15	imperfect debugging ($p = 0.95$)	failure rates
3-16	imperfect debugging ($p = 0.95$)	percentages of faults detected
3-17	Littlewood	failure rates
3-18	Littlewood	percentages of faults detected
3-19	certification	failure rates

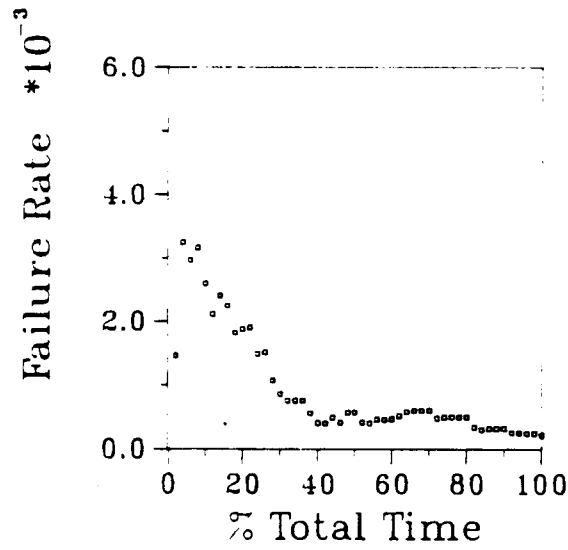


Figure 3-9: Failure rates, Jelinski-Moranda model.

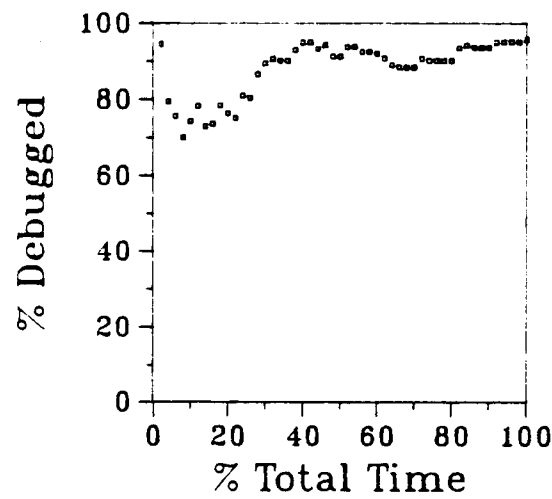


Figure 3-10: Percentages of faults detected, Jelinski-Moranda model.

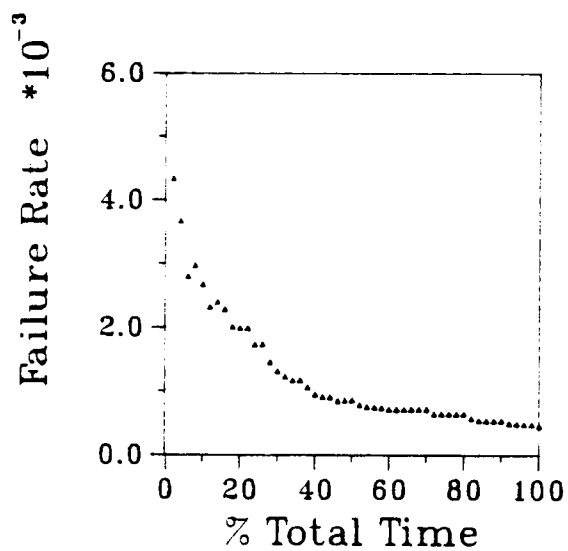


Figure 3-11: Failure rates, geometric de-eutrophication model.

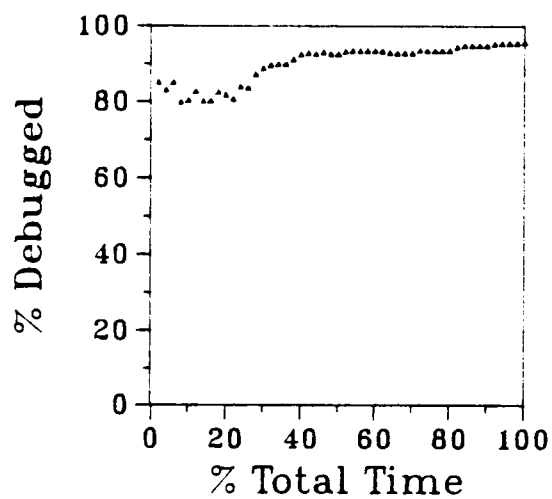


Figure 3-12: Percentages of faults detected, geometric de-eutrophication model.

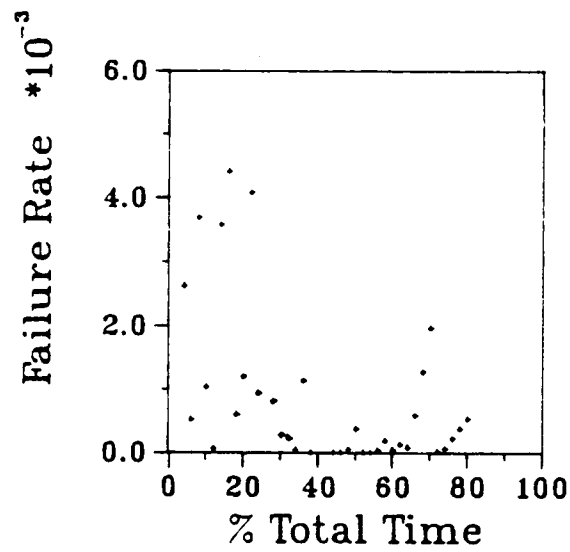


Figure 3-13: Failure rates, Schick-Wolverton model.

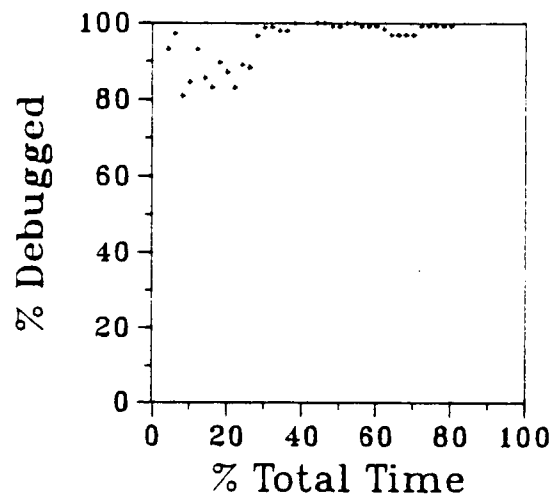


Figure 3-14: Percentages of faults detected, Schick-Wolverton model.

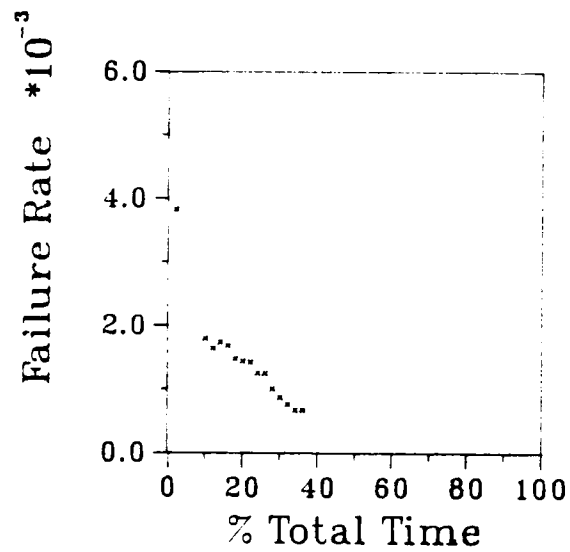


Figure 3-15: Failure rates, imperfect debugging model.

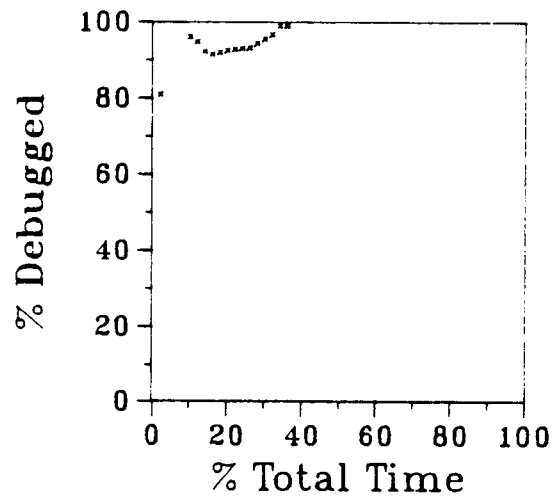


Figure 3-16: Percentages of faults detected, imperfect debugging model.

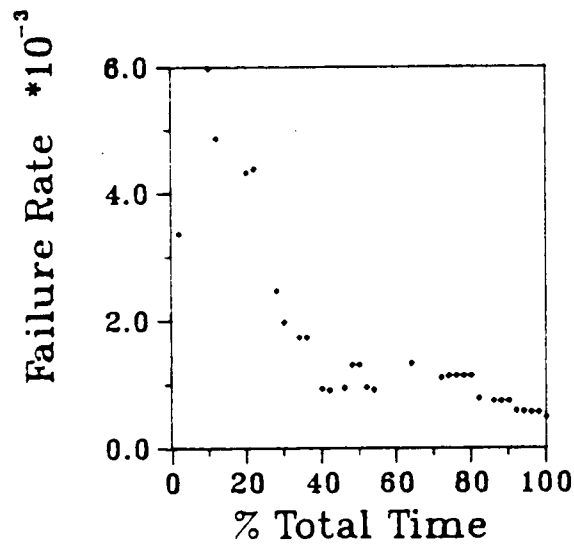


Figure 3-17: Failure rates, Littlewood model.

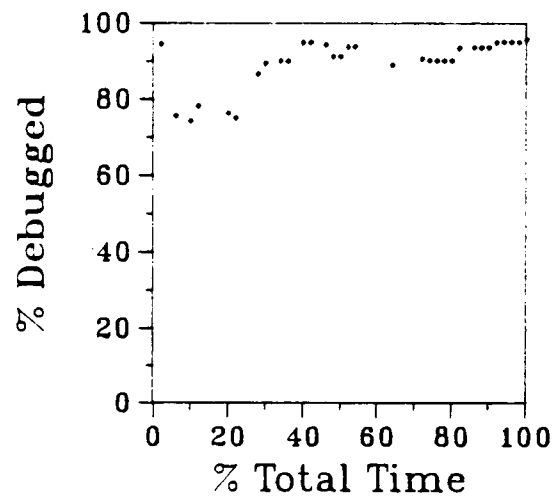


Figure 3-18: Percentages of faults detected, Littlewood model.

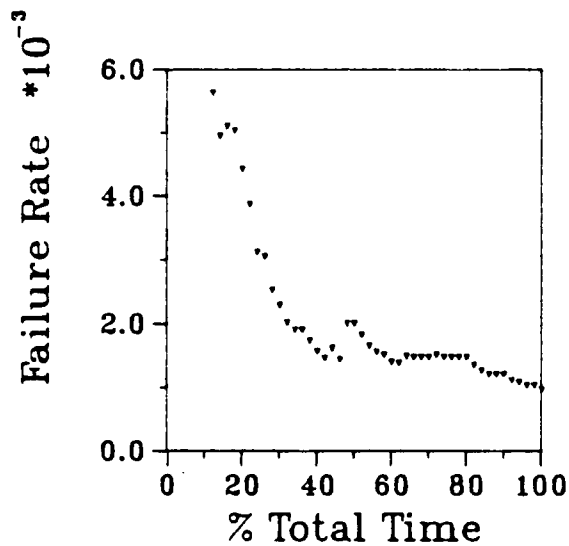


Figure 3-19: Failure rates, certification model.

Note that there are certain times at which reliability measures could not be computed for the Schick-Wolverton, imperfect debugging, and Littlewood models using the Musa failure data because numerical solutions were not obtainable for estimated parameters at such time-values. Several papers [3,30,36,37,38] investigate conditions for the Jelinski-Moranda model under which the distribution of observed failure times dictates that the maximum likelihood method cannot yield a solution for estimated parameters.

However, the inability to obtain a numerical solution for estimated parameters under certain conditions is obviously not limited to the Jelinski-Moranda model. In practice, failure data distributions exist for which each of the models

using the maximum likelihood method or nonlinear regression analysis to estimate parameters will fail to yield a numerical solution. When a numerical solution is not obtainable for estimated parameters at a given time, reliability measures cannot be computed at that point. Therefore, the ability of a model to obtain a numerical solution for estimated parameters, especially using failure data sets that exhibit general reliability growth, should be considered when selecting a model.

The plots presented in this section serve only to illustrate basic reliability measures that can be computed using times-between-failures software reliability models. Strengths and weaknesses of the models presented here cannot be completely determined from predictions of reliability measures for one software failure data set. Therefore, more exhaustive comparisons of the models are presented in Chapter 5 for a variety of failure data distributions.

CHAPTER 4

FAILURE-COUNT MODELS

This chapter presents an analysis of the three failure-count software reliability models being studied. They are:

1. the Goel-Okumoto nonhomogeneous Poisson process model;
2. the Musa-Okumoto logarithmic Poisson execution time model;
3. the Musa execution time model.

Assumptions underlying each of the models presented in this chapter are analyzed to determine their accuracy in modeling the behavior of software. The mathematical expressions that are used to compute reliability measures for each of the models are derived from the assumptions that form the basis of the respective models. These mathematical expressions contain parameters (such as the total number of faults in a software system) that are required for the computation of the various reliability measures. The methods used to estimate such parameters in the implementations of the models are given. For illustration purposes, reliability measure predictions are presented for each of the models using a software failure data set that is widely used in the literature for analysis of software reliability models.

4.1 The Goel-Okumoto Nonhomogeneous Poisson Process Model

The Poisson distribution is frequently used to model the number of occurrences of random phenomena [28]. The values of a Poisson random variable are the nonnegative integers. For a random variable $N(t)$, the sample space is $\{0, 1, 2, \dots\}$. The probability function of a Poisson random variable is given by

$$P(N(t) = y) = \frac{m^y e^{-m}}{y!}, \quad y = 0, 1, 2, \dots \quad (4.1)$$

where m is the mean of the Poisson distribution. Goel and Okumoto propose a model in which $N(t)$, the number of software failures observed at time t , is modeled as a Poisson process with a time-dependent failure rate. This process is also referred to as a nonhomogeneous Poisson process (NHPP).

4.1.1 Model Assumptions

The following assumptions form the basis of the Goel-Okumoto NHPP model:

1. the time between failures $(i-1)$ and i is dependent upon the time to failure $(i-1)$;
2. the counting process time intervals are independent of each other;
3. the number of failures in the time interval $(t, t + \Delta t]$ is proportional to the number of remaining faults at time t ;

4. a detected fault is corrected with certainty in a negligible amount of time and no new faults are introduced.

The fourth assumption is identical to a Jelinski-Moranda model assumption and was discussed in Section 3.1.1.

The accuracy of the assumption that times between failures are independent of each other was discussed in Section 3.1.1. For software, the assumption that the time between failures $(i - 1)$ and i is dependent upon the time to failure $(i - 1)$ is more realistic than the assumption that times between failures are independent of each other.

A counting process time interval is a time interval during which the number of failures occurring may be counted. The assumption that counting process time intervals are independent of each other implies that for time intervals $t_1, t_2, \dots, t_n$, the n random variables $N(t_1), N(t_2) - N(t_1), \dots, N(t_n) - N(t_{n-1})$ are independent of each other. Since the numbers of failures in time intervals are dependent upon the times between failures, this assumption is analogous to the times-between-failures models assumption that times between failures are independent of each other. This assumption appears to be inconsistent with the assumption that the time between failures $(i - 1)$ and i is dependent upon the time to failure $(i - 1)$.

The assumption that the number of failures in the time interval $(t, t + \Delta t]$ is proportional to the number of remaining faults at time t is analogous to the times-between-failures models assumption that the failure rate at any time is

proportional to the current fault content of a software system. The accuracy of the assumption that the failure rate at any time is proportional to the current fault content of a software system was also discussed in Section 3.1.1.

4.1.2 Model Development

Let the random variable $N(t)$ be a counting process representing the cumulative number of software failures at time t . Therefore, the initial fault content of a software system is treated as a random variable by the Goel-Okumoto NHPP model as opposed to being treated as an unknown, fixed constant by the majority of the times-between-failures models. Let $m(t)$ be the expected number of failures at time t . The function $m(t)$ is called the mean value function of the $N(t)$ process [21]. Then

$$m(0) = 0, \quad m(\infty) = a, \quad (4.2)$$

where a is the expected number of software faults to eventually be detected.

The assumption that the number of failures in the time interval $(t, t + \Delta t]$ is proportional to the number of remaining faults at time t implies that

$$m(t + \Delta t) - m(t) = b[a - m(t)] \Delta t + O(\Delta t), \quad (4.3)$$

where $O(\Delta t)/\Delta t$ approaches zero as Δt approaches zero and b is a proportionality constant. Equation (4.3) may be rewritten as

$$\frac{m(t + \Delta t) - m(t)}{\Delta t} = b[a - m(t)] + \frac{O(\Delta t)}{\Delta t}. \quad (4.4)$$

Taking the limit as Δt approaches zero of both sides of (4.4) results in the equation

$$m'(t) = ab - bm(t). \quad (4.5)$$

Solving (4.5) with the initial condition $m(0) = 0$ yields

$$m(t) = a(1 - e^{-bt}). \quad (4.6)$$

The failure intensity function, which is the fault detection rate or failure rate, is

$$\lambda(t) = m'(t) = abe^{-bt}. \quad (4.7)$$

The failure intensity process is depicted in Figure 4-1.

In addition to the expected number of faults to eventually be detected and the failure intensity at time t , reliability at time t can also be calculated. Let t_{i-1} be the time to $(i - 1)$ failures. From [22], the reliability for future elapsed time t' is given by

$$R(t') = e^{-[m(t') - m(t_{i-1})]}. \quad (4.8)$$

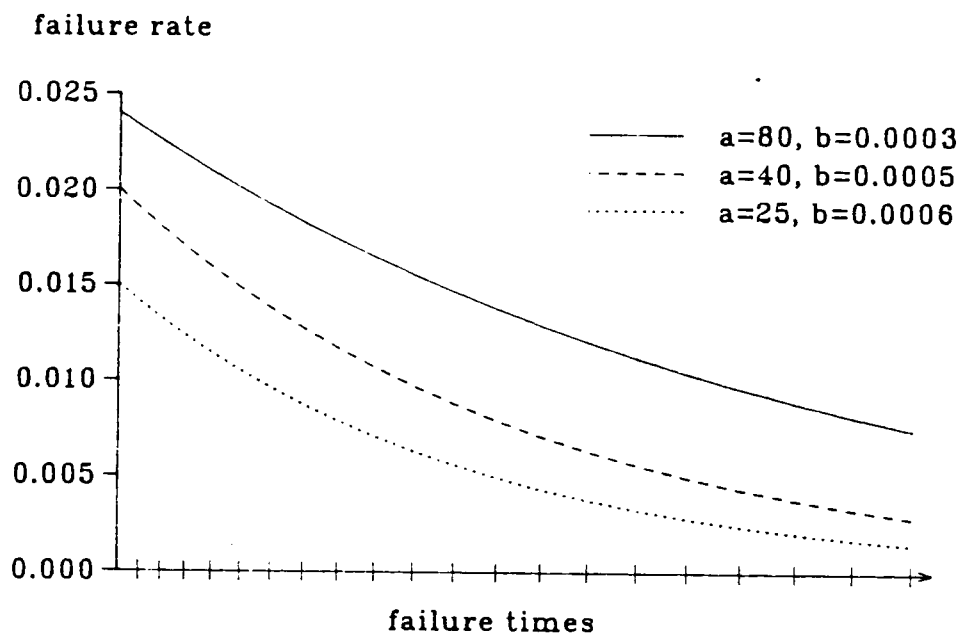


Figure 4-1: Goel-Okumoto NHPP model failure intensity process.

Substituting $a(1 - e^{-bt})$ for $m(t)$ in (4.8),

$$R(t') = \exp \left[-a(e^{-bt_{i-1}} - e^{-bt'}) \right]. \quad (4.9)$$

Therefore, the reliability at the time of failure i is dependent upon the time of failure $(i - 1)$.

4.1.3 Parameter Estimation

If a and b are known at a certain time, then other reliability measures can be determined at that time. Goel and Okumoto use the maximum likelihood method to estimate a and b .

For the Goel-Okumoto NHPP model, the probability density function is given by

$$f(t_i) = abe^{-bt_i} e^{-[m(t_i) - m(t_{i-1})]} \quad (4.10)$$

from (4.9), (4.8), (2.12), and (2.10). The logarithm of the likelihood function is

$$\ln L = \sum_{i=1}^n \ln(ab) - \sum_{i=1}^n bt_i - a(1 - e^{-bt_n}), \quad (4.11)$$

where there are n observed failures. The partial derivatives of the logarithm of the likelihood function are given by

$$\frac{\partial \ln L}{\partial a} = \frac{n}{a} - (1 - e^{-bt_n}) \quad (4.12)$$

and

$$\frac{\partial \ln L}{\partial b} = \frac{n}{b} - at_n e^{-bt_n} - \sum_{i=1}^n t_i. \quad (4.13)$$

The right hand sides of (4.12) and (4.13) are set equal to zero in order to maximize (4.11). From (4.12), this process yields

$$a = \frac{n}{1 - e^{-bt_n}}. \quad (4.14)$$

This value of a is substituted into the right hand side of (4.13). When the right hand side of (4.13) is set equal to zero, the resulting equation is

$$b - \frac{n}{t_n(\frac{n}{1 - e^{-bt_n}})e^{-bt_n} + \sum_{i=1}^n t_i} = 0. \quad (4.15)$$

Since an analytical solution for b is not available for (4.15), it is solved numerically. In order to facilitate the numerical solution for b , (4.15) is rewritten as

$$e^{-bt_n}(bt_n n - b \sum_{i=1}^n t_i + n) + b \sum_{i=1}^n t_i - n = 0. \quad (4.16)$$

Equation (4.16) is then solved numerically for b , and the resulting estimate for b is substituted into (4.14) to determine the estimate for a .

The Newton-Raphson method is used in the implementation of the Goel-Okumoto NHPP model to solve (4.16) for b . Once estimates for b , and therefore a , are computed at a given time, then failure intensity and reliability estimates are easily computed at that time.

4.2 The Musa-Okumoto Logarithmic NHPP Execution Time Model

Musa and Okumoto propose a model in which $N(\tau)$, the number of software failures observed at execution time τ , is modeled as a Poisson process with a time-dependent failure rate. The Musa-Okumoto model specifies that the time component of the model is execution time.

Musa and Okumoto relate execution time τ to calendar time t in [22]. The failure intensity function is exponential in execution time and has various forms in calendar time depending upon available personnel and computing resources. The calendar time component of the Musa-Okumoto model will not be considered further due to difficulties in accurately estimating resource utilization parameters over the life of a software system project.

When software failures are reported in terms of calendar time, the actual amount of execution time is not generally known. Therefore, reporting software failures in terms of execution time generally results in more accurate reliability measure predictions than reporting software failures in terms of calendar time. However, reporting software failures in terms of execution time requires more detailed monitoring of a software system than does reporting software failures in terms of calendar time.

4.2.1 Model Assumptions

The following assumptions form the basis of the Musa-Okumoto logarithmic NHPP model:

1. the time between failures $(i-1)$ and i is dependent upon the time to failure $(i-1)$;
2. the counting process time intervals are independent of each other;
3. no failures have been observed at time $\tau = 0$;
4. the failure intensity λ decreases exponentially with the expected number of failures observed;
5. the failure intensity decreases geometrically after each fault is detected and corrected;
6. a detected fault is corrected with certainty in a negligible amount of time and no new faults are introduced;
7. for a small value $\Delta\tau$, the probability of one failure in the time interval $(\tau, \tau + \Delta\tau]$ is $\lambda(\tau)\Delta\tau + O(\Delta\tau)$ and the probability of more than one failure in the time interval $(\tau, \tau + \Delta\tau]$ is $O(\Delta\tau)$, where $O(\Delta\tau)/\Delta\tau$ approaches zero as $\Delta\tau$ approaches zero.

The first and second assumptions are identical to Goel-Okumoto NHPP model assumptions and were discussed in Section 4.1.1. The third assumption always

holds true. It has been shown in Chapter 2 that an exponential distribution of failures is a reasonable assumption. The fifth and sixth assumptions are identical to geometric de-eutrophication model assumptions and were discussed in Section 3.1.1. The seventh assumption generally holds true and is used to show that the Musa-Okumoto model is a Poisson process model.

4.2.2 Model Development

Given the assumption that the failure intensity λ decreases exponentially with the expected number of failures observed, if the initial failure intensity and the rate of reduction in the failure intensity per failure are denoted by λ_0 and θ , respectively, then the assumption may be written as

$$\lambda(\tau) = \lambda_0 e^{-\theta m(\tau)}, \quad (4.17)$$

where $m(\tau)$ is the mean value function of the $N(\tau)$ process. Equation (4.17) results in a model in which faults detected early in the fault correction process reduce the failure intensity more than faults detected later in the fault correction process. A larger value of θ causes the failure intensity to decrease at a faster rate than does a smaller value of θ . The failure intensity process is illustrated in Figure 4-2.

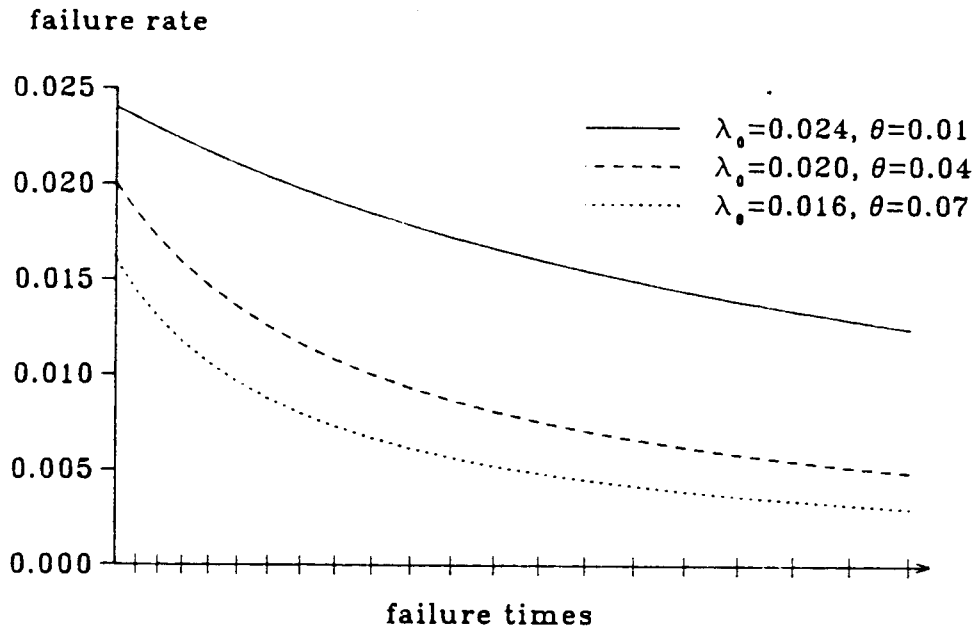


Figure 4-2: Musa-Okumoto model failure intensity process.

From (4.7), for the Musa-Okumoto model,

$$m'(\tau) = \lambda_0 e^{-\theta m(\tau)}. \quad (4.18)$$

Solving (4.18) for $m(\tau)$ with the initial condition $m(0) = 0$ results in a mean value function of

$$m(\tau) = \frac{1}{\theta} \ln(\lambda_0 \theta \tau + 1). \quad (4.19)$$

Also from (4.7), the failure intensity function is then

$$\lambda(\tau) = \frac{\lambda_0}{\lambda_0 \theta \tau + 1}. \quad (4.20)$$

Using the third and seventh assumptions, it can be shown that the random variable $N(\tau)$ is a Poisson process. Let τ_{i-1} be the time to $(i - 1)$ failures. The reliability for future elapsed time τ' is given by

$$R(\tau') = e^{-[m(\tau') - m(\tau_{i-1})]}. \quad (4.21)$$

From (4.19),

$$R(\tau') = e^{-[\frac{1}{\theta} \ln(\lambda_0 \theta \tau' + 1) - \frac{1}{\theta} \ln(\lambda_0 \theta \tau_{i-1} + 1)]}. \quad (4.22)$$

Equation (4.22) may be rewritten as

$$R(\tau') = \left[\frac{\lambda_0 \theta \tau_{i-1} + 1}{\lambda_0 \theta \tau' + 1} \right]^{\frac{1}{\theta}}. \quad (4.23)$$

As is the case with the Goel-Okumoto NHPP model, the reliability at the time of failure i is dependent upon the time of failure $(i - 1)$.

Additionally, the execution time $\Delta\tau$ required to achieve a decrease in the failure rate from the failure rate at the current time τ_C to a failure rate goal λ_G can also be calculated. From (4.20),

$$\lambda_G = \frac{\lambda_0}{\lambda_0 \theta \tau_G + 1}. \quad (4.24)$$

Rewriting (4.24),

$$\Delta\tau = \frac{\lambda_0 - \lambda_G}{\lambda_0\theta\lambda_G} - \tau_C. \quad (4.25)$$

Note that the Musa-Okumoto model does not attempt to predict the initial fault content of a software system. Musa and Okumoto provide the following rationale for not predicting the initial fault content of a software system in [22]:

Faults can only be defined by their discovery, e.g., through failures; there are so many possible implementations of a program that faults cannot be defined with respect to a standard program. But this means that the fault as a unit of measure is not constant as execution proceeds, because the extent of comprehension of the underlying defect at a failure can vary. The actual number of faults can only be counted accurately as time of execution becomes very large and understanding of what happened is reasonably complete.

4.2.3 Parameter Estimation

If λ_0 and θ are known at a certain time, then other reliability measures can be determined at that time. Okumoto proposes a method in [23] for estimating λ_0 and θ using linear regression analysis that is based upon partitioning failures into groups.

The time interval $(0, \tau_n)$, where there are n observed failures, is partitioned into groups of k failures so that there are p subintervals. The parameter p is the largest integer of n/k . Therefore, there are k failures in each of the first $(p - 1)$ subintervals and there are $n - k(p - 1)$ failures in the last subinterval. The last

subinterval may contain more failures than each of the first $(p - 1)$ subintervals.

The failure intensity for subinterval i is estimated by

$$y_i = \begin{cases} \frac{k}{\tau_{ki} - \tau_{k(i-1)}} & \text{if } i < p, \\ \frac{n - k(p-1)}{\tau_n - \tau_{k(p-1)}} & \text{if } i = p. \end{cases} \quad (4.26)$$

The mean value function for subinterval i is estimated by

$$x_i = k(i - 1). \quad (4.27)$$

Grouping a small number of failures may result in large variations in the estimated failure intensities. Grouping a large number of failures may result in too much smoothing of the estimated failure intensities. Clearly, the optimum number of failures to be grouped by partitioning of the observed failure data will vary among failure data sets. Based upon extensive study, Okumoto recommends selecting $k = 5$ in general. Although selecting $k = 5$ may provide a reasonable starting point, each particular failure data set should be studied in order to determine if some other value of k is more appropriate based upon the distribution of observed failures.

The least squares method is used to determine estimates for λ_0 and θ . The failure intensity function given by (4.17) is fitted to the estimated failure intensities for the p time subintervals, resulting in the equation

$$y_i = \lambda(x_i)\epsilon_i, \quad (4.28)$$

where ϵ_i is the ratio of the point (x_i, y_i) to the model. Taking the logarithm of each side of (4.28),

$$\ln y_i = \ln \lambda(x_i) + \ln \epsilon_i. \quad (4.29)$$

Substituting (4.17) into (4.29) and simplifying,

$$\ln y_i = \ln \lambda_0 - \theta x_i + \ln \epsilon_i. \quad (4.30)$$

Equation (4.30) may be used to perform standard linear regression analysis to minimize the sum of the $(\ln \epsilon_i)^2$ terms. The regression analysis yields estimates for θ and $\ln \lambda_0$, but λ_0 can be computed once $\ln \lambda_0$ is known.

The standard linear regression equations are used in the implementation of the Musa-Okumoto logarithmic NHPP execution time model to compute estimates for $\ln \lambda_0$ and θ from (4.30). The estimate for λ_0 is computed as $\exp(\ln \lambda_0)$. Once estimates for λ_0 and θ are computed at a given time, estimates for the failure intensity, the reliability, and the execution time required to achieve a decrease in the failure rate from the current failure rate to a failure rate goal are easily computed.

4.3 The Musa Execution Time Model

Each of the models previously described has incorporated the assumption that no new faults are introduced during the fault correction process. The

Musa execution time model may be used to compute the standard reliability measures, and it also allows the introduction of new faults during the fault correction process to be modeled via a fault reduction factor parameter.

The Musa model, as does the Musa-Okumoto model, specifies that the time component of the model is execution time. As is the case with the Musa-Okumoto model, the calendar time component of the Musa model will not be considered further due to difficulties in accurately estimating resource utilization parameters over the life of a software system project.

4.3.1 Model Assumptions

The following assumptions form the basis of the Musa execution time model:

1. each of the N_0 faults in a software system will cause a failure after a time which is distributed exponentially;
2. software faults are independent of each other and are equally likely to cause a failure;
3. the failure rate at any time is proportional to the current fault content of a software system;
4. execution times between failures are large compared to the average instruction execution time;
5. testing for a software system is representative of its eventual use;

6. all failures are observed.

The first, second, and third assumptions are identical to Jelinski-Moranda model assumptions and were discussed in Section 3.1.1.

The assumption that execution times between failures are large compared to the average instruction execution time ensures that the set of instructions executed between failures forms a large random sample. Therefore, the average instruction execution rate of the random sample accurately models the overall instruction execution rate.

The assumption that testing for a software system is representative of its eventual use ensures that results of testing can be used to make meaningful predictions about the reliability of a software system in the use environment. This assumption does not imply that testing must be exhaustive, but clearly test input data sets should be representative of use input data sets. This assumption has been implied for the models previously discussed and is generally accurate.

The assumption that all failures are observed prevents reliability measure predictions from being overly optimistic. Recall from Chapter 1 that a failure is defined as any departure of program output from requirements. Therefore, failures may be subtle. Musa writes in [5] that "one is more likely to miss failures in the use period of the program than the test period." This assumption has also been implied for the models previously discussed.

4.3.2 Model Development

Let the execution frequency f be the average instruction execution rate divided by the number of instructions in a software system. Let K be a fault exposure proportionality constant. The initial fault content of a software system is denoted by N_0 and the number of failures observed at time τ is denoted by n .

The fault reduction factor B is defined as the average ratio of the reduction of faults to the rate of failure occurrence. Let m be the number of failures occurring in the process of correcting n faults and let M_0 be the number of failures required to correct the N_0 faults initially in a software system. Then,

$$m = \frac{n}{B} \quad (4.31)$$

and

$$M_0 = \frac{N_0}{B}. \quad (4.32)$$

For example, if 100 failures occur in the process of correcting 95 faults, then B is 0.95. Therefore, the fault reduction factor B allows the introduction of new faults to be modeled.

Additionally, Musa writes in [5] that it is a common practice to “directly plan test cases at the extremes of system performance and then randomly pick tests at intermediate points.” Failures are more likely to occur at the extremes of system

performance, but the majority of input data sets to a software system during its actual use do not approach system extremes. Therefore, Musa has incorporated a testing compression factor C into the model. The testing compression factor C is defined by Musa to be "the average ratio of the rate of detection of failures during test to that during use."

Unfortunately, C must be estimated from previous similar software system projects. Musa admits that the value used for C in validating the model with failure data sets in [5] had "low" accuracy. Since it is very difficult to determine an accurate estimate for C , it is best to be conservative and let C be 1. The development of the Musa model presented here assumes C is 1, effectively removing C from consideration in the model.

The assumption that the failure rate λ at any time is proportional to the current fault content of a software system implies that

$$\lambda(\tau) = BfKM_0 - BfKm. \quad (4.33)$$

The fault correction rate is equal to the failure rate. Therefore,

$$\frac{dm}{d\tau} = \lambda(\tau). \quad (4.34)$$

Combining (4.33) and (4.34) results in the differential equation

$$\frac{dm}{d\tau} + BfKm = BfKM_0. \quad (4.35)$$

Solving (4.35) with the initial condition $m = 0$ at $\tau = 0$ yields

$$m = M_0(1 - e^{-BfK\tau}). \quad (4.36)$$

Note the similarity between (4.36), the mean value function for the Goel-Okumoto NHPP model given by (4.6), and the mean value function for the Musa-Okumoto model given by (4.19).

Since an exponential distribution of failures is assumed, the MTTF is the reciprocal of the failure rate. Musa denotes MTTF by T ; this convention will be used in the remainder of the discussion of the Musa model. From (4.33), the initial MTTF of a software system before any testing is done is given by

$$T_0 = \frac{1}{BfKM_0}. \quad (4.37)$$

Substituting (4.36) into (4.33) and simplifying the resulting equation, the failure rate at time τ is

$$\lambda(\tau) = BfKM_0e^{-BfK\tau}. \quad (4.38)$$

Substituting (4.37) into (4.38),

$$\lambda(\tau) = \frac{1}{T_0} \exp\left(\frac{-\tau}{M_0T_0}\right). \quad (4.39)$$

The failure rate process is illustrated in Figure 4-3. From (2.21), the MTTF at time τ is

$$T = T_0 \exp \left(\frac{\tau}{M_0 T_0} \right). \quad (4.40)$$

From (2.20) and (2.21), the reliability for future elapsed time τ' at time τ is

$$R(\tau') = \exp \left(\frac{-\tau'}{T} \right). \quad (4.41)$$

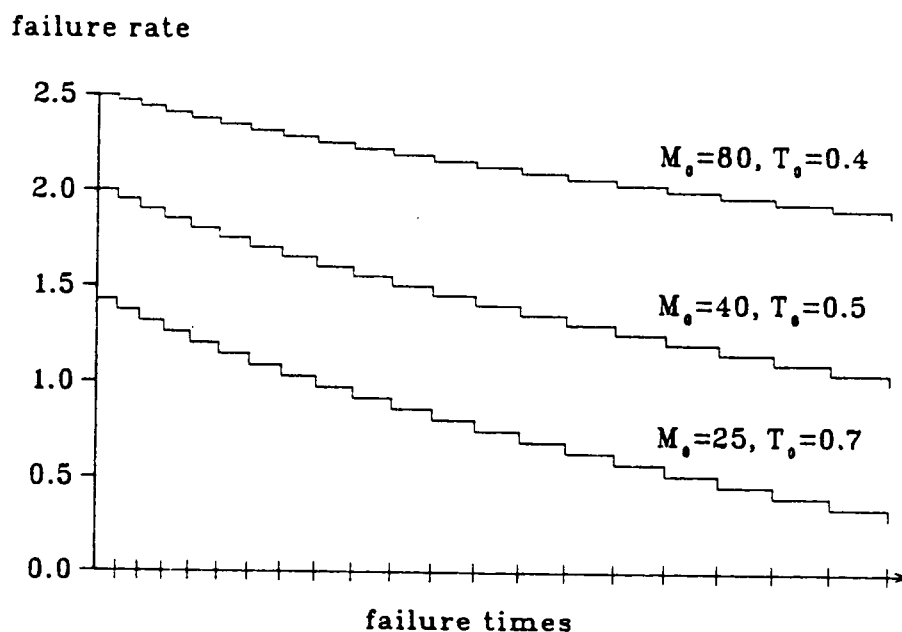


Figure 4-3: Musa model failure intensity process.

Additionally, the number of failures Δm that must be observed in the process of detecting and correcting faults in order to achieve a decrease in the failure rate from the current failure rate λ_C to a failure rate goal λ_G and the execution time $\Delta\tau$ required to achieve a decrease in the failure rate from λ_C to λ_G may also be computed. From (4.36) and (4.37),

$$\Delta m = M_0 \left(1 - \frac{\lambda_G}{\lambda_0} \right) - M_0 \left(1 - \frac{\lambda_C}{\lambda_0} \right), \quad (4.42)$$

where λ_0 is the initial failure rate. Equation (4.42) may be rewritten as

$$\Delta m = \frac{M_0}{\lambda_0} (\lambda_C - \lambda_G). \quad (4.43)$$

From (4.40) and (4.37),

$$\Delta\tau = \frac{M_0}{\lambda_0} \ln \left(\frac{\lambda_0}{\lambda_G} \right) - \frac{M_0}{\lambda_0} \ln \left(\frac{\lambda_0}{\lambda_C} \right). \quad (4.44)$$

Equation (4.44) may be rewritten as

$$\Delta\tau = \frac{M_0}{\lambda_0} \ln \left(\frac{\lambda_C}{\lambda_G} \right). \quad (4.45)$$

4.3.3 Parameter Estimation

If M_0 , T_0 , and B are known at a certain time, then other reliability measures can be determined at that time. Musa uses the maximum likelihood method to estimate M_0 and T_0 . Note that T_0 is estimated rather than f and K .

An estimate for B must be determined from previous similar software system projects. This reduces the possibility of effective use of B somewhat, but it does not make the model any less valid. Musa reports B values ranging from 0.94 to 1 for the four software system projects he studied in [5], with an average B value of 0.96. As was the case with the testing compression factor C , if there is no basis for estimation for B , then it is best to be conservative and let B be 1. However, doing so eliminates a major reason for using the Musa model rather than other models, because a value of 1 for B implies that no new faults are introduced during the fault correction process.

Since an exponential distribution of failures is assumed for the Musa model, the probability density function is

$$f(\tau_i) = \lambda e^{-\lambda \tau_i} \quad (4.46)$$

from (2.12) and (2.21), where τ_i values are m observed times between failures. Since $(i - 1)$ failures have been observed during τ_i and the failure rate at any time is proportional to the current fault content of a software system, the current failure rate is

$$\lambda(\tau_i) = BfKM_0 - BfK(i - 1) \quad (4.47)$$

from (4.33). Substituting T_0 into (4.47),

$$\lambda(\tau_i) = \frac{1}{T_0} - \frac{1}{T_0 M_0}(i - 1). \quad (4.48)$$

Simplifying further,

$$\lambda(\tau_i) = \frac{1}{T_0} \left(1 - \frac{i-1}{M_0}\right). \quad (4.49)$$

The probability density function for the Musa model is then

$$f(\tau_i) = \frac{1}{T_0} \left(1 - \frac{i-1}{M_0}\right) \exp \left[-\tau_i \frac{1}{T_0} \left(1 - \frac{i-1}{M_0}\right) \right]. \quad (4.50)$$

The logarithm of the likelihood function is

$$\ln L = \sum_{i=1}^m \ln \frac{1}{T_0} \left(1 - \frac{i-1}{M_0}\right) - \sum_{i=1}^m \frac{1}{T_0} \left(1 - \frac{i-1}{M_0}\right) \tau_i. \quad (4.51)$$

The partial derivatives of the logarithm of the likelihood function are given by

$$\frac{\partial \ln L}{\partial T_0} = \frac{1}{m} \sum_{i=1}^m \left(1 - \frac{i-1}{M_0}\right) \tau_i - T_0 \quad (4.52)$$

and

$$\frac{\partial \ln L}{\partial M_0} = \sum_{i=1}^m \frac{1}{M_0 - i + 1} - \frac{1}{M_0 T_0} \sum_{i=1}^m \tau_i. \quad (4.53)$$

The right hand sides of (4.52) and (4.53) are set equal to zero in order to maximize (4.51). The resulting equations are

$$\frac{1}{m} \sum_{i=1}^m \left(1 - \frac{i-1}{M_0}\right) \tau_i - T_0 = 0 \quad (4.54)$$

and

$$\sum_{i=1}^m \frac{1}{M_0 - i + 1} - \frac{1}{M_0 T_0} \sum_{i=1}^m \tau_i = 0. \quad (4.55)$$

Equations (4.54) and (4.55) form a system of two unknowns in two nonlinear equations. Since an analytical solution for the system of equations is not available, a solution is obtained numerically. Estimates for M_0 and T_0 are determined by using a SLATEC Mathematical Subroutine Library routine to solve (4.54) and (4.55) for M_0 and T_0 in the implementation of the Musa model. Once estimates for M_0 and T_0 are computed at a given time, then failure rate, MTTF, reliability, Δm , and $\Delta \tau$ estimates are easily computed at that time.

4.4 Example Reliability Measure Calculations

For the purpose of illustration, reliability measures computed using the Goel-Okumoto NHPP model, the Musa-Okumoto logarithmic NHPP model, and the Musa execution time model are now presented in graphical form. Reliability measures are computed using the Musa failure data previously described in Section 3.7. Reliability measures are computed for each of the three failure-count models at time-values ranging from 2 percent of the total reported time to 100 percent of the total reported time in increments of 2 percent. Therefore, there are 50 time-values at which reliability measures are computed for each of the three models. At each value, only the failure data observed up to that

time are used in the reliability measure calculations. Plots are presented for two reliability measures, the failure rate and the percentage of faults detected. The percentage of faults detected is defined to be the number of detected faults divided by the total number of faults predicted to eventually occur. Descriptions of the plots illustrating reliability measures computed for the Musa failure data using the models presented in this chapter are given in Table 4-1.

Table 4-1: Illustrated reliability measures, Figures 4-4 through 4-8.

Figure	Model	Reliability Measure
4-4	Goel-Okumoto NHPP	failure rates
4-5	Goel-Okumoto NHPP	percentages of faults detected
4-6	Musa ($B = 1$)	failure rates
4-7	Musa ($B = 1$)	percentages of faults detected
4-8	Musa-Okumoto	failure rates

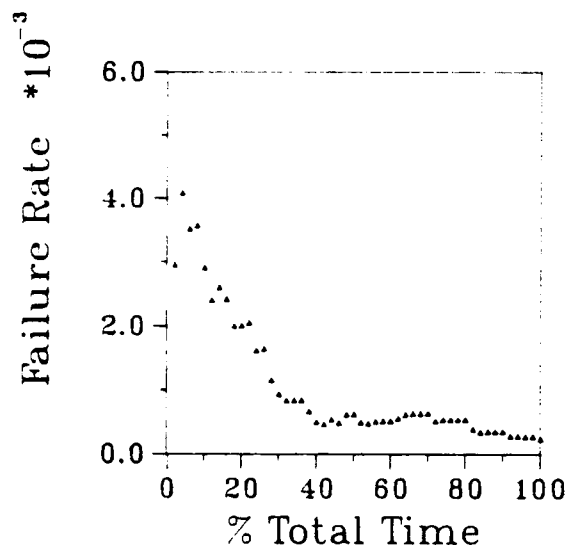


Figure 4-4: Failure rates, Goel-Okumoto NHPP model.

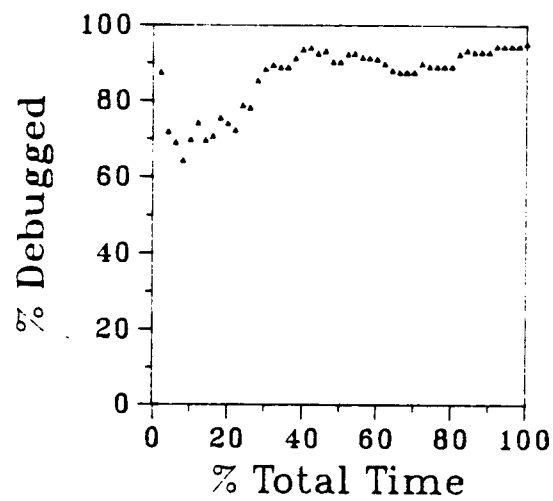


Figure 4-5: Percentages of faults detected, Goel-Okumoto NHPP model.

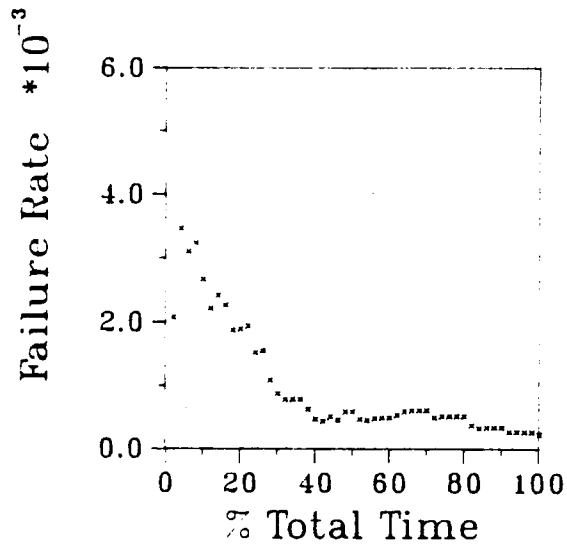


Figure 4-6: Failure rates, Musa model.

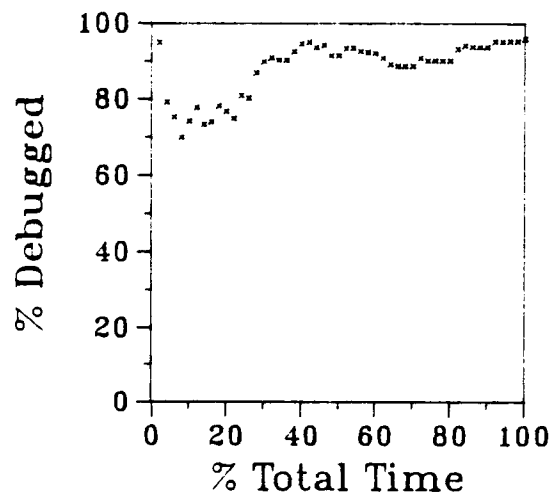


Figure 4-7: Percentages of faults detected, Musa model.

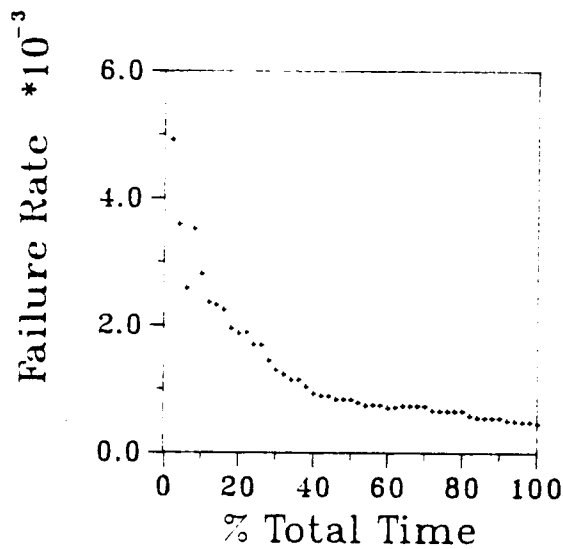


Figure 4-8: Failure rates, Musa-Okumoto model.

The numerical problems discussed in Section 3.7 did not occur for the Musa failure data using any of the three failure-count models. Numerical solutions were obtainable for estimated parameters at each of the 50 time-values for each of the models. This absence of numerical problems is an indication that the Musa failure data set exhibits general reliability growth and that these models are likely to converge to solutions for estimated parameters for failure data sets that exhibit general reliability growth.

The plots presented in this section serve only to illustrate basic reliability measures that can be computed using failure-count software reliability models. Strengths and weaknesses of the models presented here cannot be completely

determined from predictions of reliability measures for one software failure data set. Therefore, more exhaustive comparisons of the models are presented in Chapter 5 for a variety of failure data distributions.

CHAPTER 5

MODEL COMPARISONS AND RECOMMENDATIONS

It was stated in Chapter 1 that no single software reliability model has been shown to perform well in all contexts. Therefore, models must be compared.

This chapter presents comparison criteria to be considered when selecting a model to make reliability measure predictions from observed failure data. General recommendations are given based upon experience gained while performing these analyses. Comparisons of reliability measures predicted for various failure data distributions using the models described in Chapter 3 and Chapter 4 are then presented. The distributions for which predictions are compared are chosen to be representative of software systems of high, medium, and low quality. Finally, specific recommendations are given based upon those comparisons.

5.1 Model Comparison Criteria

The following criteria should be considered when comparing software reliability models:

1. matching the characteristics of the experimental data to model assumptions;
2. reliability measures for which predictions are desired;
3. predictive validity of the models;
4. the ability of the models to compute meaningful parameter estimates;
5. ease of understanding and ease of use of the models.

The primary goal of a software manager selecting a reliability model should be to match the distribution of observed failure data as well as possible to model assumptions. Clearly, the software manager must understand the nature of the models and the quality of their assumptions in order to select the best possible model for a particular experimental circumstance. If the distribution of observed failure data is not consistent with the assumptions of a particular model, then there is no basis for claiming validity of the resulting reliability measure predictions using that model, even if the other criteria are satisfied.

It is unlikely that times between failures comprising a failure data set will follow a monotonic pattern from beginning to end. Therefore, the distribution of the failure data should be determined using its general nature. Representing failure data graphically as in Figure 3-8 facilitates characterizing the data set.

The reliability measures for which predictions are desired must be considered. For example, if one wants a prediction of the number of failures to eventually

occur, then a model must be selected that has this capability; not all models will do this. However, certain criteria must be satisfied in order to assure a valid prediction.

Each model presented here predicts the failure rate and reliability at the time of the most recent failure and can extend to future predicted failures. The MTTF at the time of any failure, either observed or predicted, is obtained from the failure rate for each model.

Although such equations are not derived here, the number of failures predicted to occur in a future time interval $(t, t + \Delta t]$ can be computed for times-between-failures models and for failure-count models. For failure-count models, the number of failures predicted to occur in the interval $(t, t + \Delta t]$ is simply $m(t + \Delta t) - m(t)$, where $m(t)$ is the mean value function evaluated at time t . For times-between-failures models, the number of failures predicted to occur in the interval $(t, t + \Delta t]$ is determined by summing predicted MTTF values until Δt is exceeded. Table 5-1 presents a compilation of reliability measures predicted by each model.

The predictive validity of a model is the capability of the model to predict future failure behavior from observed failure data. The predictive validity of models can be determined by comparing MTTF predictions with observed times to the next failure. If the distribution of observed failure data matches the assumptions of more than one model equally well, then the model that exhibits the best predictive validity for that failure data should be chosen.

Table 5-1: Predicted reliability measures.

Model*	Number of Faults	λ , MTTF, R (current, future)	Failures in $(t, t + \Delta t]$	Required For Target λ	
				Time	Failures
J-M	x	x	x		
geom		x	x		
S-W	x	x	x		
i-d	x	x	x		
L	x	x	x		
cert		x	x		
G-O	x	x	x		
M-O		x	x	x	
Musa	x	x	x	x	x

* J-M = Jelinski-Moranda, geom = geometric de-eutrophication,
 S-W = Schick-Wolverton, i-d = imperfect debugging,
 L = Littlewood, cert = certification, G-O = Goel-Okumoto NHPP,
 M-O = Musa-Okumoto

The ability of a model to compute meaningful parameter estimates is primarily determined by the distribution of observed failure data. If the data closely matches the assumptions of a model, then the parameter estimates computed by that model are likely to be accurate, regardless of the model complexity and parameter estimation method. Additionally, all models are more likely to produce accurate parameter estimates for failure data sets that exhibit general reliability growth than for those that do not exhibit general reliability growth. Of course, if failure data reported for a software system does not exhibit general reliability growth, then software reliability models are not needed to determine that the software system lacks quality.

Since a software manager must understand models in order to compare them, ease of understanding is clearly important and should be considered. An extensive mathematical background should not be required in order to understand the nature of a model and its assumptions. Ease of use should also be considered. It should be simple and inexpensive to collect failure data to be used as input to a model and the model should be able to be implemented as a practical management tool [6]. None of the models described here has a clear advantage over the other models when considering ease of use, because a collection of times between failures is the only input required for most of them. The Musa model and the imperfect debugging model also require that an input parameter be estimated. If an accurate estimate cannot be assured, then a conservative estimate may be chosen. Choosing a conservative estimate will reduce either model to the Jelinski-Moranda model.

Table 5-2 lists the model class, the author's assessment of complexity, and the parameter estimation method used for each model.

Table 5-2: Model attributes.

Model	Class ¹	Complexity ²	Parameter Estimation
Jelinski-Moranda	TBF	1	maximum likelihood
geometric de-eutrophication	TBF	2	maximum likelihood
Schick-Wolverton	TBF	3	maximum likelihood
imperfect debugging	TBF	4	nonlinear regression
Littlewood	TBF	5	maximum likelihood
certification	TBF	1	linear regression
Goel-Okumoto NHPP	FC	2	maximum likelihood
Musa-Okumoto	FC	2	linear regression
Musa	FC	3	maximum likelihood

¹ TBF is times-between-failures, FC is failure-count

² 1 is the least complex, 5 is the most complex

5.2 General Recommendations

General recommendations for selecting a software reliability model from among the models described are now given. These recommendations are based upon the preceding analyses rather than upon detailed model comparisons. Therefore, these general recommendations are intended to provide a starting point in the selection of a software reliability model. Recommendations vary with circumstances. The software manager must determine which considerations are applicable.

Nine software reliability models were presented. The first recommendation is to eliminate the Schick-Wolverton model and the imperfect debugging model from consideration for reasons detailed in Chapter 3. The Schick-Wolverton model assumption that the failure rate at any time is proportional to the current fault content and to the elapsed time since the last fault was detected is not valid in use for software. The imperfect debugging model is an improvement over the Jelinski-Moranda model only if the perfect debugging probability can be accurately estimated. This probability is difficult to estimate, and small variations in the estimate may cause large variations in predicted reliability measures. As noted, a conservative estimate for this probability reduces the imperfect debugging model to the Jelinski-Moranda model.

The second recommendation is to consider models that incorporate the assumption that, on the average, faults detected early result in a greater reduction in the failure rate than faults detected later over models that incorporate the assumption that the failure rate at any time is proportional to the current

fault content. Models incorporating the former assumption are the geometric de-eutrophication model, the Littlewood Bayesian differential debugging model, the certification model, and the Musa-Okumoto model.

The relative merits of the maximum likelihood method and the least squares method for parameter estimation have been discussed in the literature [23,36]. Neither method has been shown to be clearly superior over a wide range of applications. However, if linear regression analysis is used to estimate parameters, then statistical inferences may be used to produce certified reliability measure estimates if the software system is developed under statistical control. Linear regression analysis is recommended for parameter estimation by the developers of the certification model and the Musa-Okumoto model.

If a decision must be made to either accept or reject a software system based upon reliability measure estimates after future failures, then it is desirable to predict reliability measures at the times of future predicted failures. Each model presented here has the ability to predict reliability measures at the times of future predicted failures.

The certification model possesses each of the desirable qualities mentioned above. However, neither the certification model, the geometric de-eutrophication model, nor the Musa-Okumoto model predicts the initial fault content of a software system for (excellent) reasons given in Chapter 3 and Chapter 4. If the initial fault content of a software system is desired, then the software manager must choose from among the Jelinski-Moranda model, the Littlewood Bayesian differential debugging model, the Goel-Okumoto NHPP model, and the Musa model.

Similarly, if the execution time required to achieve a decrease in the failure rate from the current failure rate to a target failure rate is desired, then the software manager must choose from among the Musa-Okumoto model and the Musa model. If the number of failures that must be observed in the process of detecting and correcting faults in order to achieve a decrease in the failure rate from the current failure rate to a target failure rate is desired, then the Musa model must be selected.

5.3 Model Comparisons

The models are now compared using published failure data sets chosen to be representative of failure data distributions for software systems of high, medium, and low quality. Model comparisons are presented graphically. The Schick-Wolverton and imperfect debugging models are not considered.

For each of the published failure data sets, two types of plots are used to illustrate predictive validity. The first plot is a comparison of the observed times to failures and the current MTTF values predicted by the models at the time of each reported failure. The second plot is a comparison of observed times to failures and future MTTF values predicted by the models at the time of a specified failure. At each failure for which MTTF values are predicted, the failure data up to and including that failure are used to determine current and future MTTF predictions.

5.3.1 High Quality Software Failure Data

Representative failure data for software systems of high quality was reported by Currit, Dyer, and Mills in [19]. This data was reported for a software system developed using the cleanroom method described in [20] and consists of 30 times between failures, in CPU seconds. The observed number of failures is relatively low, the total elapsed time is high, and general reliability growth is exhibited. This data set will be referred to as the "Mills failure data" and is presented in Table 5-3.

Table 5-3: Mills failure data times (CPU seconds).

Failures	Times									
1-10	85	85	479	965	469	385	796	277	927	340
11-20	405	150	277	503	694	620	4050	3064	522	1797
21-30	2329	3798	2775	2393	909	994	28212	14956	1971	5308

Figure 5-1 presents the times to failures reported for the Mills failure data and the current MTTF values predicted at the times of each failure by the seven models. Figure 5-2 presents the times to failures reported for the Mills failure data and future MTTF values predicted at the time of the twenty-fifth failure by the seven models. Predicted MTTF values that are plotted in Figure 5-1 and Figure 5-2 are presented in Table A-1, Appendix A, and Table A-2, Appendix A, respectively.

5.3.2 Medium Quality Software Failure Data

The Musa failure data presented in Chapter 3 is representative of software systems of medium quality. The observed number of failures is relatively high for the Musa failure data, but the total elapsed time is high and general reliability growth is exhibited. The Musa failure data set is presented in Table 3-1.

Figure 5-3 presents the times to failures reported for the one hundredth and subsequent failures for the Musa failure data and the current MTTF values predicted at the times of those failures by the seven models. Figure 5-4 presents the times to failures reported for the one hundredth and subsequent failures for the Musa failure data and future MTTF values predicted at the time of the one hundredth failure by the seven models. The times to the first 99 failures reported are excluded in order to reduce clutter in the plots. Predicted MTTF values that are plotted in Figure 5-3 and Figure 5-4 are presented in Table A-3, Appendix A, and Table A-4, Appendix A, respectively.

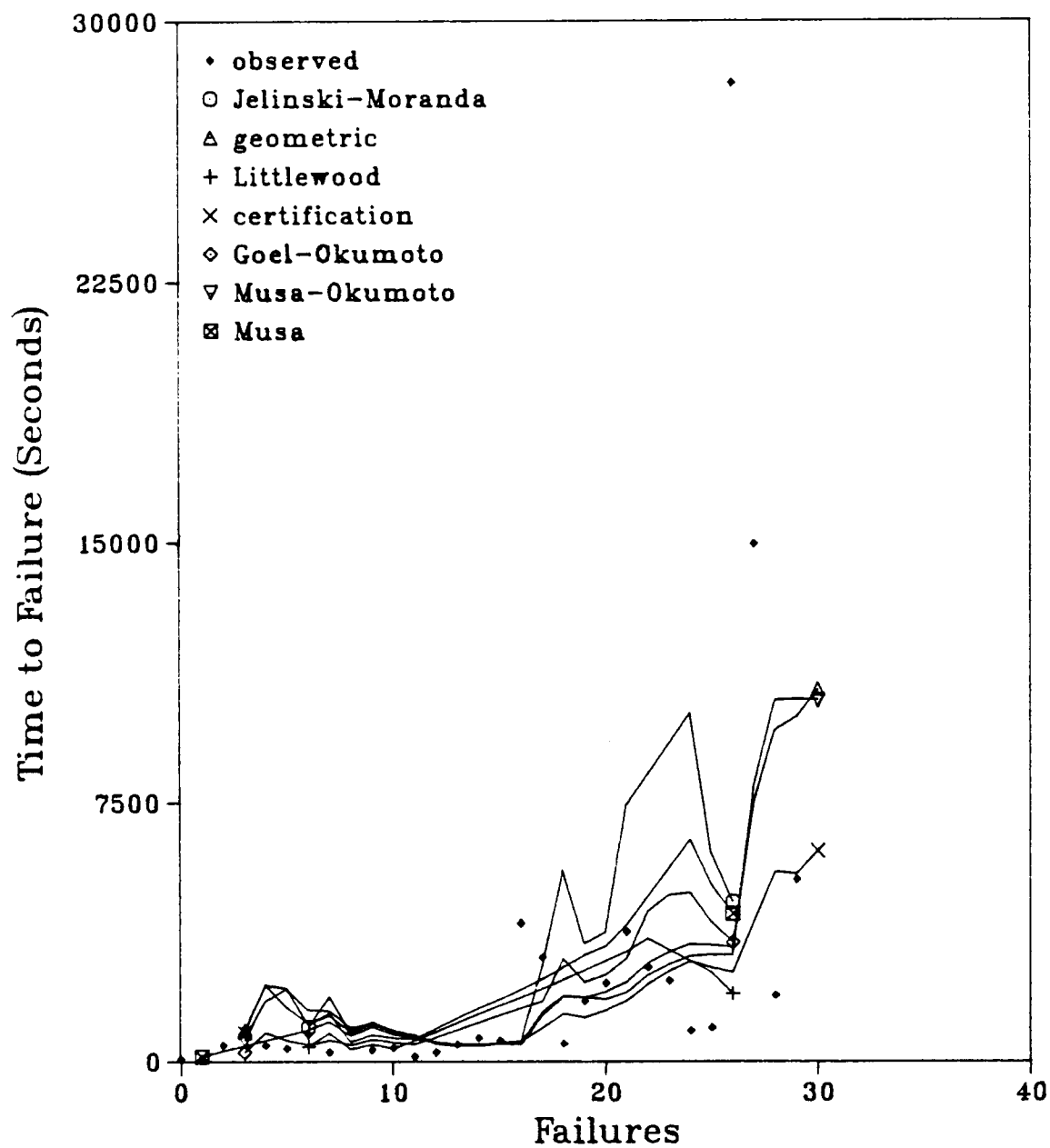


Figure 5-1: Current MTTF predictions, Mills failure data.

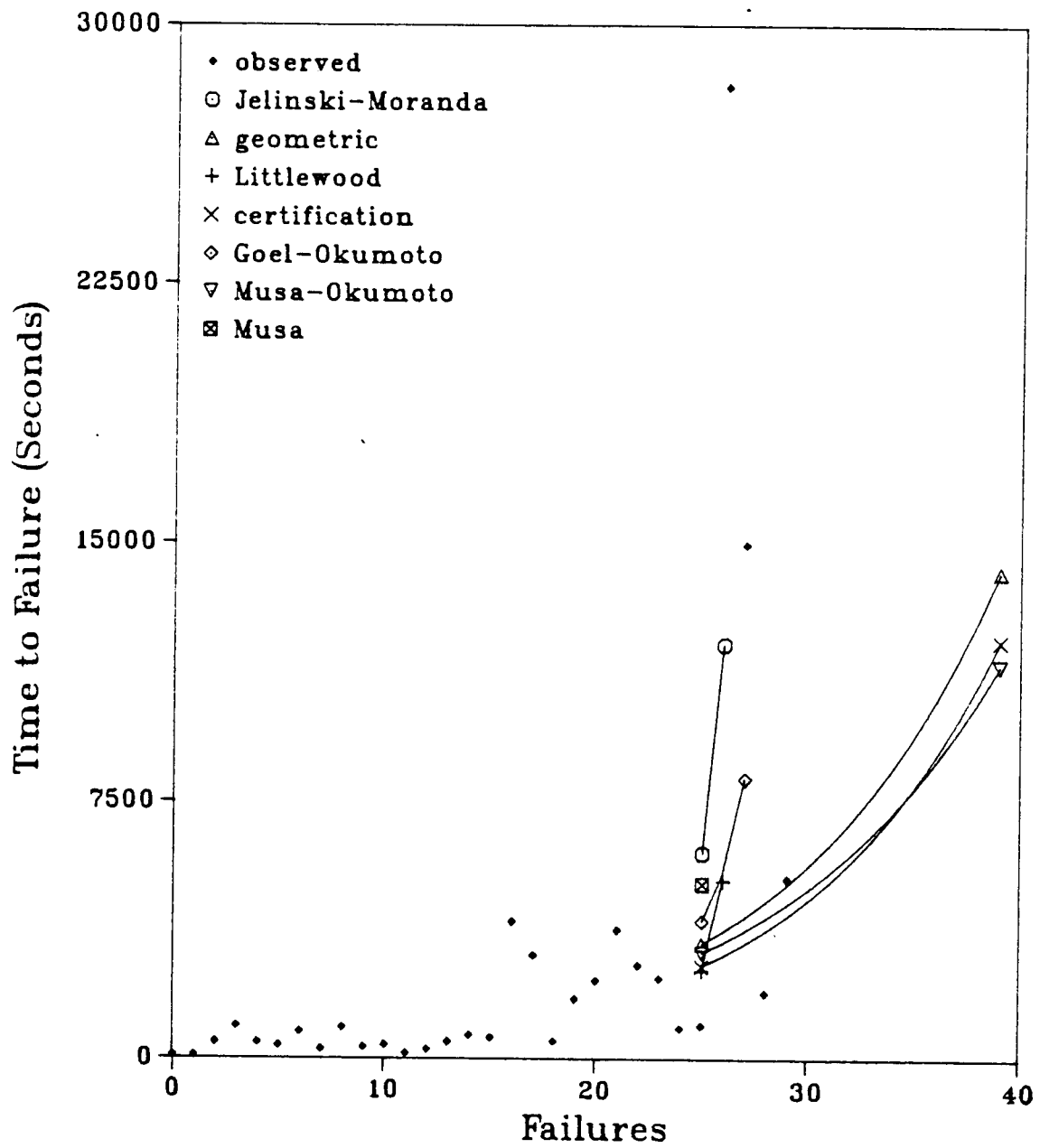


Figure 5-2: Future MTTF predictions, Mills failure data.

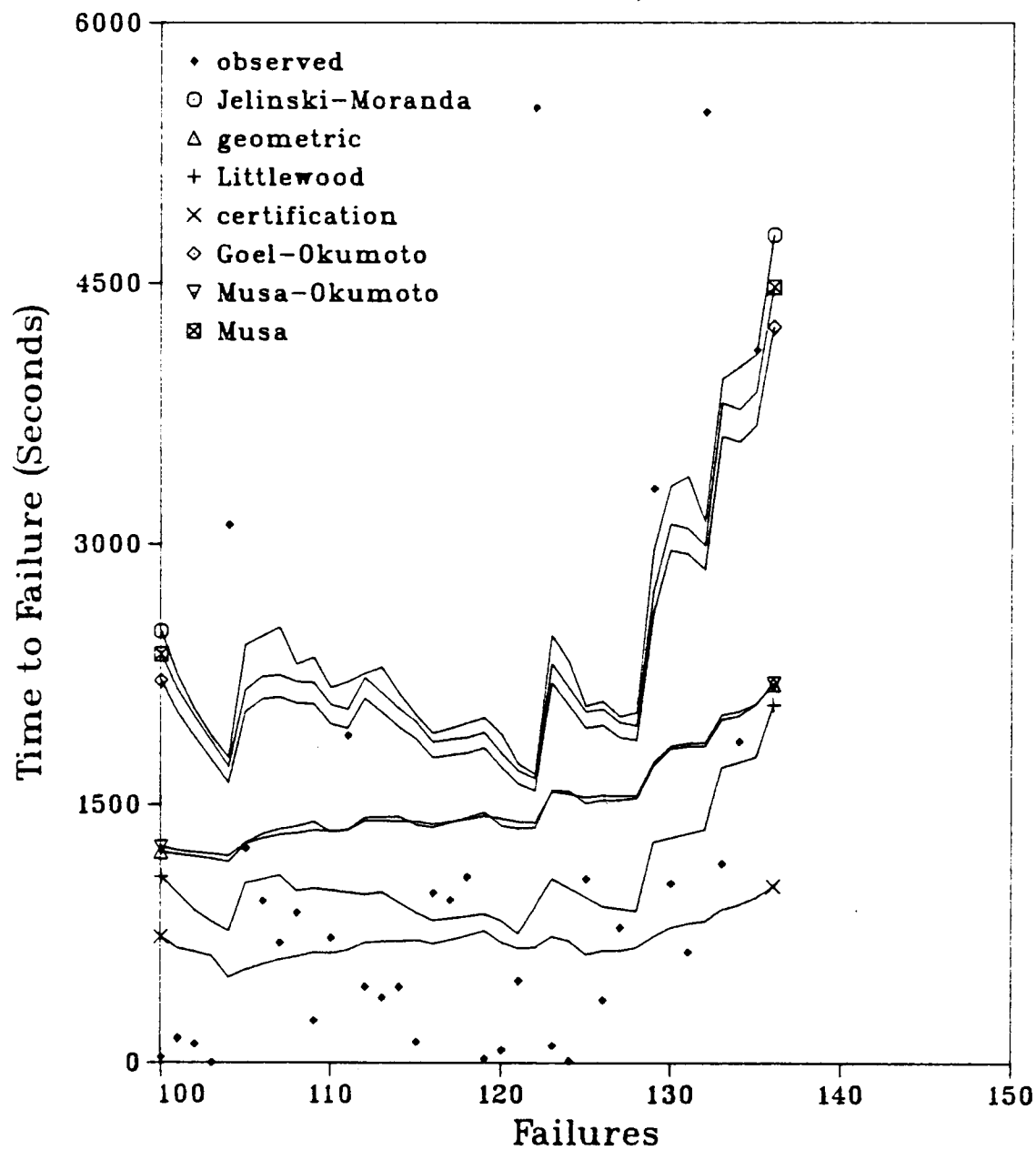


Figure 5-3: Current MTTF predictions, Musa failure data.

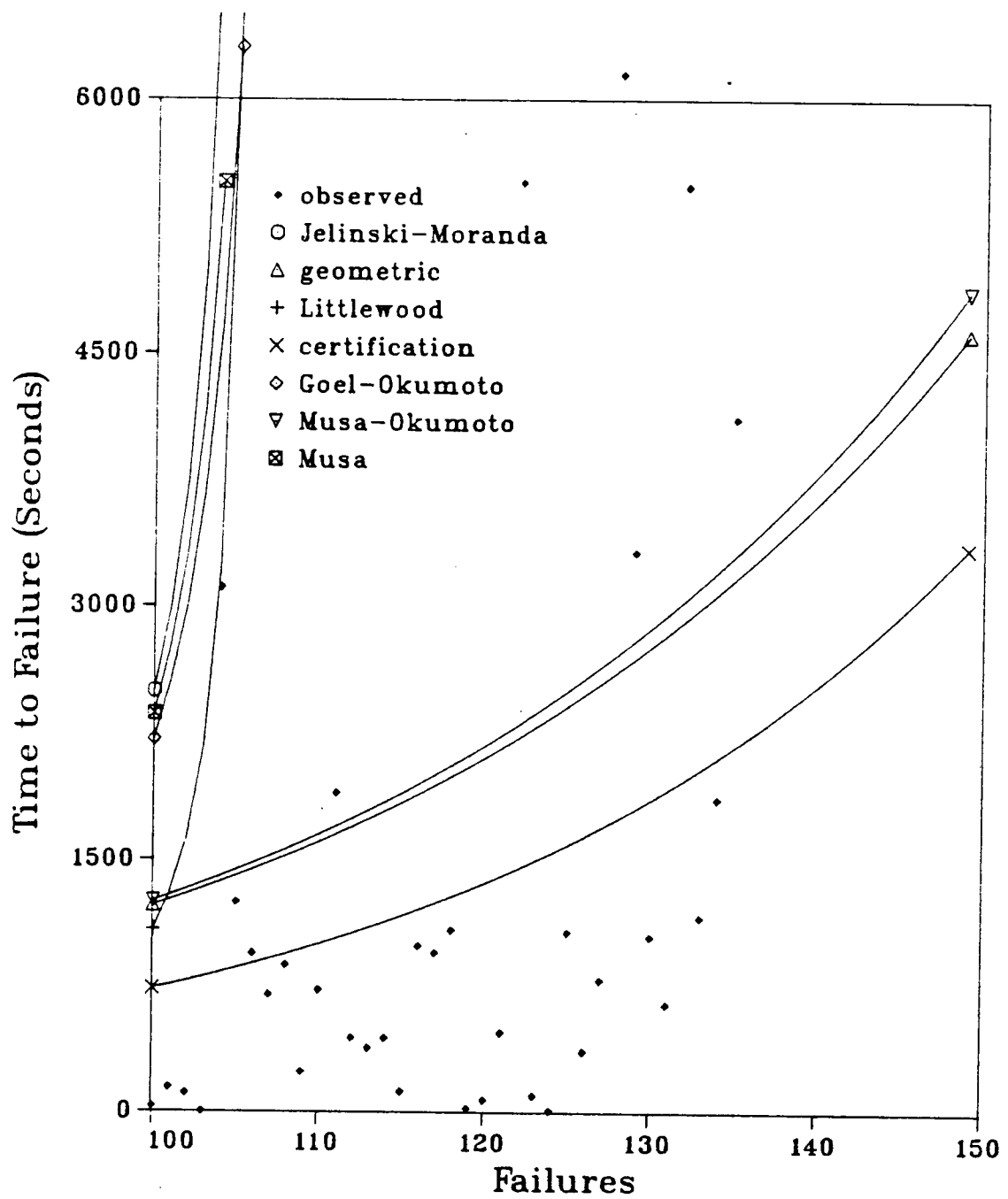


Figure 5-4: Future MTTF predictions, Musa failure data.

5.3.3 Low Quality Software Failure Data

A failure data set representative of software systems of low quality was reported by Abdel-Ghaly, Chan, and Littlewood in [24]. This data consists of 86 times between failures, in CPU seconds. The observed number of failures is relatively high and the total elapsed time is low. This failure data will be referred to as the "Abdel-Ghaly failure data" and is presented in Table 5-4.

Table 5-4: Abdel-Ghaly failure data times (CPU seconds).

Failures	Times							
1-8	4.79	2.66	2.77	5.54	10.34	9.49	6.93	5.97
9-16	1.17	1.70	1.17	12.74	4.69	11.74	6.93	19.08
17-24	1.35	2.77	5.96	7.57	4.37	22.30	4.37	3.40
25-32	4.05	5.75	2.77	3.63	5.22	6.13	2.77	13.00
33-40	8.21	2.13	16.20	16.01	2.98	8.74	6.18	26.40
41-48	0.05	1.49	10.34	24.41	4.60	5.65	11.19	4.37
49-56	9.27	44.62	7.14	1.81	14.85	7.57	31.54	21.15
57-64	8.84	20.37	14.81	5.59	4.90	5.93	17.69	0.85
65-72	28.36	2.13	18.66	4.90	14.87	43.22	14.18	10.23
73-80	54.90	15.20	32.81	27.16	21.75	35.05	7.25	19.63
81-86	39.79	10.90	2.45	11.94	9.94	39.02		

Figure 5-5 presents the times to failures reported for the fiftieth and subsequent failures for the Abdel-Ghaly failure data and the current MTTF values predicted at the times of those failures by the seven models. Figure 5-6 presents the times to failures reported for the fiftieth and subsequent failures for the Abdel-Ghaly failure data and future MTTF values predicted at the time of the fiftieth failure by six of the seven models. The Littlewood model predictions are not presented because a numerical solution was not obtained for the parameters used to estimate MTTF values at the time of the fiftieth failure. Predicted MTTF values that are plotted in Figure 5-5 and Figure 5-6 are presented in Table A-5, Appendix A, and Table A-6, Appendix A, respectively.

5.4 Specific Recommendations

Recommendations are now made for software systems of high, medium, and low quality based upon the comparisons presented in the previous section.

5.4.1 High Quality Software

Recommendations for software systems of high quality are based upon Figure 5-1 and Figure 5-2, which used the Mills failure data. It is expected that the use of software development techniques such as the cleanroom method will lead to a general improvement in software quality. Therefore, these recommendations are especially important.

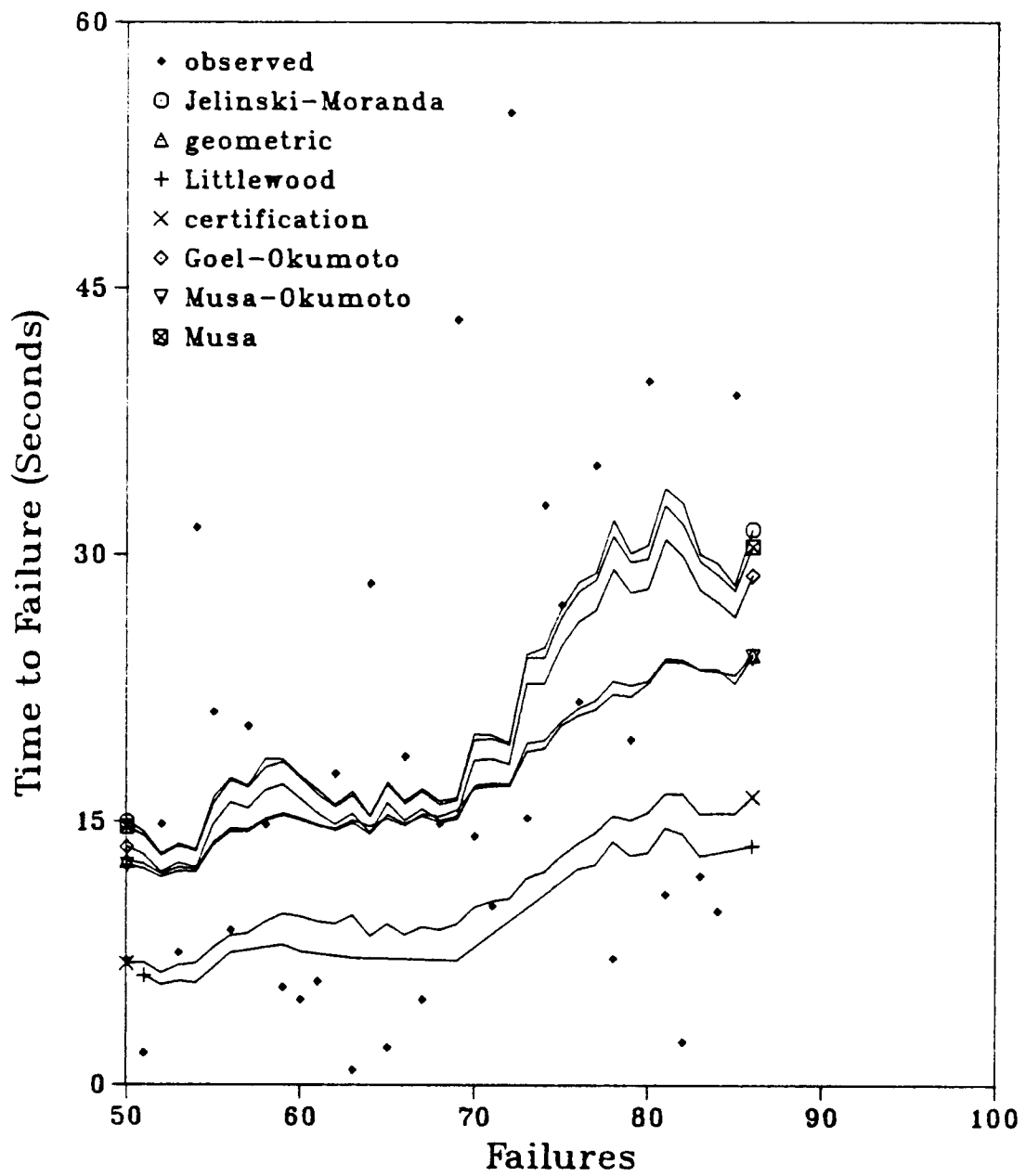


Figure 5-5: Current MTTF predictions, Abdel-Ghaly failure data.

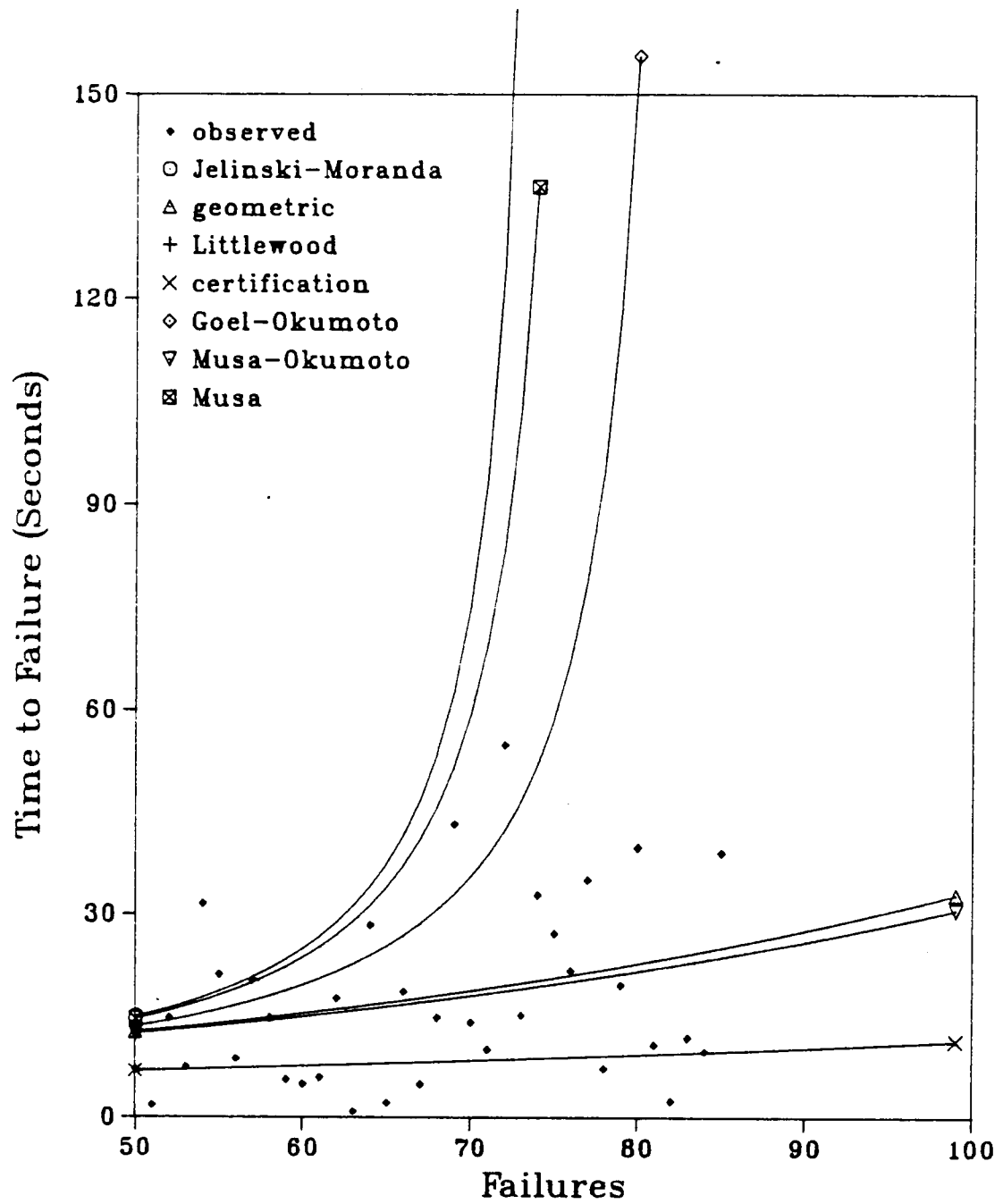


Figure 5-6: Future MTTF predictions, Abdel-Ghaly failure data.

Figure 5-1 shows that for the Mills failure data a numerical solution was not obtained by the Jelinski-Moranda model, the Littlewood model, the Goel-Okumoto NHPP model, or the Musa model for the parameters used to estimate current MTTF values once the twenty-seventh failure was observed. The time to the twenty-seventh failure is about seven times as large as the time to any previously observed failure. However, large times to failures are characteristic of software systems of high quality. Figure 5-1 also shows that the current MTTF values predicted by the certification model are more conservative and more accurate than the current MTTF values predicted by the geometric de-eutrophication model and the Musa-Okumoto model.

Figure 5-2 shows that at the time of the twenty-fifth failure, the Jelinski-Moranda model, the Littlewood model, the Goel-Okumoto NHPP model, and the Musa model each predict that fewer than 30 failures will be observed. Since 30 failures are reported, these predictions for the number of failures to eventually occur are clearly overly optimistic. Figure 5-2 also shows that at the time of the twenty-fifth failure, future MTTF values predicted by the geometric de-eutrophication model are slightly more optimistic than those predicted by the Musa-Okumoto model and the certification model.

Based upon the Mills failure data comparisons presented in Figure 5-1 and Figure 5-2, the certification model is recommended for making MTTF, failure rate, and reliability predictions for software systems of high quality.

5.4.2 Medium Quality Software

Recommendations for software systems of medium quality are based upon Musa failure data comparisons presented in Figure 5-3 and Figure 5-4. Figure 5-3 shows that the current MTTF values predicted by the Jelinski-Moranda model, the Goel-Okumoto NHPP model, and the Musa model are overly optimistic. Figure 5-3 also shows that the current MTTF values predicted by each of the four remaining models being compared can be considered reasonable. The geometric de-eutrophication model and the Musa-Okumoto model predictions are essentially the same and are the most optimistic, the certification model predictions are the most conservative, and the Littlewood model predictions fall in between.

Figure 5-4 shows that at the time of the one hundredth failure, the Jelinski-Moranda model, the Littlewood model, the Goel-Okumoto NHPP model, and the Musa model each predict that 106 or fewer failures will be observed. Since 136 failures are reported, these predictions are again clearly overly optimistic. Figure 5-4 also shows that at the time of the one hundredth failure, future MTTF values predicted by the certification model are more conservative and more accurate than those predicted by the geometric de-eutrophication model and the Musa-Okumoto model.

Based upon the Musa failure data comparisons presented in Figure 5-3 and Figure 5-4, the certification model is recommended for making MTTF, failure rate, and reliability predictions for software systems of medium quality.

5.4.3 Low Quality Software

Recommendations for software systems of low quality are based upon Abdel-Ghaly failure data comparisons that are presented in Figure 5-5 and Figure 5-6. Figure 5-5 shows that the current MTTF values predicted by the Littlewood model and the certification model are overly conservative. Figure 5-5 also shows that the current MTTF values predicted by each of the five remaining models can be considered reasonable. The Jelinski-Moranda model and the Musa model predictions are essentially the same and are the most optimistic, the geometric de-eutrophication model and the Musa-Okumoto model predictions are essentially the same and are the most conservative, and the Goel-Okumoto NHPP model predictions fall in between.

Figure 5-6 shows that at the time of the fiftieth failure, the Jelinski-Moranda model, the Goel-Okumoto NHPP model, and the Musa model each predict that between 70 and 80 failures will be observed. Since 86 failures are reported and, based upon the observed failure data, it is likely that even more failures will be observed, these predictions are again overly optimistic. Figure 5-6 also shows that at the time of the fiftieth failure, future MTTF values predicted by the geometric de-eutrophication model and the Musa-Okumoto model are essentially the same and are reasonably accurate.

It is not surprising that for the Abdel-Ghaly failure data, the certification model predictions are less accurate than predictions made by other models, because the certification model assumes that the software system is of high

quality. Note in Figure 5-6 that the certification model predicts at the time of the fiftieth failure that there is almost no improvement in future MTTF values.

Based upon the Abdel-Ghaly failure data comparisons that are presented in Figure 5-5 and Figure 5-6, both the geometric de-eutrophication model and the Musa-Okumoto model are recommended for making MTTF, failure rate, and reliability predictions for software systems of low quality.

5.4.4 Summary of Recommendations

The software reliability models recommended for making MTTF, failure rate, and reliability predictions for software systems of high, medium, and low quality are summarized in Table 5-5.

None of the recommended models predicts the number of failures to eventually occur in a software system. If the number of failures to eventually occur in a software system must be predicted, then it is recommended that the Goel-Okumoto NHPP model be used. Goel-Okumoto NHPP model predictions for the number of failures to eventually occur in a software system are more conservative than Jelinski-Moranda model, Littlewood model, and Musa model predictions, regardless of the quality of the software system. However, it should be remembered that Goel-Okumoto NHPP model predictions for the number of failures to eventually occur in a software system, although more conservative than other models, are likely to be overly optimistic.

Table 5-5: Model recommendations.

Software Quality	Recommended Model
High	certification
Medium	certification
Low	Musa-Okumoto, geometric de-eutrophication

CHAPTER 6

SUMMARY AND CONCLUSIONS

Classical reliability theory is fundamentally applicable and useful in the software environment, notwithstanding the observation that software does not deteriorate with age. Several models have been proposed that attempt to better characterize the failure behavior of software. Each software reliability model has advantages and disadvantages in terms of analytical complexity, predictive validity, required data, ease of understanding, and ease of use.

Nine software reliability models are presented. The assumptions underlying each of the models are presented and analyzed to determine their accuracy in modeling the behavior of software. The mathematical expressions that are used to compute reliability measures for each of the models are derived from the assumptions that form the basis of the respective models. These mathematical expressions contain parameters that are required for the computation of the various reliability measures. The methods used to estimate such parameters in the implementations of each of the models are given.

The performance of models has been compared using failure data sets reported in the literature to be representative of failure data distributions for software systems of high, medium, and low quality. Based upon these comparisons, the certification model is recommended for making MTTF, failure rate, and reliability predictions for software systems of high and medium quality. Both the

Musa-Okumoto model and the geometric de-eutrophication model are recommended for making MTTF, failure rate, and reliability predictions for software systems of low quality.

Predictions of the number of failures to eventually occur in a software system are consistently optimistic. Therefore, since it is difficult to obtain an accurate estimate for the number of failures to eventually occur in a software system, the reliability measures that should be emphasized when comparing models and when reporting reliability measure predictions are MTTF and failure rate.

If the number of failures to eventually occur in a software system must be predicted, then it is recommended that the Goel-Okumoto NHPP model be used. Goel-Okumoto NHPP model predictions for the number of failures to eventually occur in a software system are more conservative than those made by other models, regardless of the quality of the software system. However, the predictions of even this model are still optimistic.

LIST OF REFERENCES

LIST OF REFERENCES

1. Goel, Amrit L., "Software Reliability Models: Assumptions, Limitations, and Applicability," *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 12, December 1985, pp. 1411-1423.
2. Linger, R.C., H.D. Mills, and B.I. Witt, *Structured Programming: Theory and Practice*, Reading, Massachusetts, Addison-Wesley, 1979.
3. Forman, Ernest H. and Nozer D. Singpurwalla, "An Empirical Stopping Rule for Debugging and Testing Computer Software," *Journal of the American Statistical Association*, Vol. 72, No. 360, December 1977, pp. 750-757.
4. Jelinski, Z. and P. Moranda, "Software Reliability Research," *Statistical Computer Performance Evaluation*, W. Freiberger, Ed. New York: Academic, 1972, pp. 465-484.
5. Musa, John D., "A Theory of Software Reliability and Its Application," *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 3, September 1975, pp. 312-327.
6. Iannino, Anthony, John D. Musa, Kazuhira Okumoto, and Bev Littlewood, "Criteria for Software Reliability Model Comparisons," *IEEE Transactions on Software Engineering*, Vol. SE-10, No. 6, November 1984, pp. 687-691.
7. Musa, John D., "Validity of Execution-Time Theory of Software Reliability," *IEEE Transactions on Reliability*, Vol. R-28, No. 3, August 1979, pp. 181-191.
8. Myers, Glenford J., *Software Reliability: Principles and Practices*, New York, Wiley-Interscience, 1976.
9. Musa, John D., "The Measurement and Management of Software Reliability," *Proceedings of the IEEE*, Vol. 68, No. 9, September 1980, pp. 1131-1143.
10. Mills, Harlan D., "On the Statistical Validation of Computer Programs," IBM Federal Systems Division, Gaithersburg, Maryland, Report 72-6015, 1972.
11. Nelson, E.C., "Estimating Software Reliability From Test Data," *Microelectronics and Reliability*, Vol. 17 New York: Pergamon, 1978, pp. 67-74.

12. Ramamoorthy, C.V. and Farokh B. Bastani, "Software Reliability - Status and Perspectives," *IEEE Transactions on Software Engineering*, Vol. SE-8, No. 4, July 1982, pp. 354-371.
13. Moranda, P.B., "Prediction of Software Reliability During Debugging," *Proceedings Annual Reliability and Maintainability Symposium*, January 1975, pp. 327-332.
14. Schick, George J. and Ray W. Wolverton, "An Analysis of Competing Software Reliability Models," *IEEE Transactions on Software Engineering*, Vol. SE-4, No. 2, March 1978, pp. 104-120.
15. Goel, Amrit L. and Kazu Okumoto, "A Markovian model for reliability and other performance measures of software systems," *Proceedings National Computer Conference*, New York, Vol. 48, 1979, pp. 769-774.
16. Goel, Amrit L. and Kazu Okumoto, "An Analysis of Recurrent Software Errors in a Real-Time Control System," *Proceedings of the ACM*, 1978, pp. 496-501.
17. Littlewood, B., "A Bayesian Differential Debugging Model for Software Reliability," *Proceedings COMPSAC 1980*, 1980, pp. 511-519.
18. Littlewood, Bev, "Stochastic Reliability Growth: A Model for Fault Removal in Computer Programs and Hardware Designs," *IEEE Transactions on Reliability*, Vol. R-30, No. 4, October 1981, pp. 313-320.
19. Currit, P. Allen, Michael Dyer, and Harlan D. Mills, "Certifying the Reliability of Software," *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 1, January 1986, pp. 3-11.
20. Mills, Harlan D., Michael Dyer, and Richard C. Linger, "Cleanroom Software Engineering," *IEEE Software*, September 1987, pp. 19-25.
21. Goel, Amrit L. and Kazu Okumoto, "A Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures," *IEEE Transactions on Reliability*, Vol. R-28, No. 3, August 1979, pp. 206-211.
22. Musa, John D. and Kazu Okumoto, "A Logarithmic Poisson Execution Time Model for Software Reliability Measurement," *Proceedings 7th International Conference on Software Engineering*, March 1983, pp. 230-237.
23. Okumoto, Kazuhira, "A Statistical Method for Software Quality Control," *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 12, December 1985, pp. 1424-1430.

24. Abdel-Ghaly, Abdalla A., P.Y. Chan, and Bev Littlewood, "Evaluation of Competing Software Reliability Predictions," *IEEE Transactions on Software Engineering*, Vol. SE-10, No. 9, September 1986, pp. 950-967.
25. Kline, Melvin G., "Software & Hardware R&M: What Are the Differences?" *Proceedings Annual Reliability and Maintainability Symposium*, January 1980, pp. 179-185.
26. Apostol, Tom M., *Calculus*, Vol. 2, New York, John Wiley & Sons, 1969.
27. Barlow, Richard E. and Frank Proschan, *Statistical Theory of Reliability and Life Testing*, New York, Holt, Rinehart and Winston, 1975.
28. Martz, Harry F. and Ray A. Waller, *Bayesian Reliability Analysis*, New York, John Wiley & Sons, 1982.
29. Spreij, Peter, "Parameter Estimation for a Specific Software Reliability Model," *IEEE Transactions on Reliability*, Vol. R-34, No. 4, October 1985, pp. 323-328.
30. Littlewood, Bev, "Theories of Software Reliability: How Good Are They and How Can They Be Improved?" *IEEE Transactions on Software Engineering*, Vol. SE-6, No. 5, September 1980, pp. 489-500.
31. Private communication with Harlan D. Mills.
32. Kreyszig, Erwin, *Advanced Engineering Mathematics*, 2nd ed., New York, John Wiley & Sons, 1967.
33. Sukert, Alan N., "An Investigation of Software Reliability Models," *Proceedings Annual Reliability and Maintainability Symposium*, January 1977, pp. 478-484.
34. More, J.J., B.S. Garbow, and K.E. Hillstrom, *User Guide for MINPACK-1*, Argonne National Laboratory, ANL-80-74.
35. Littlewood, Bev, "How to Measure Software Reliability and How Not To," *IEEE Transactions on Reliability*, Vol. R-28, No. 2, June 1979, pp. 103-110.
36. Angus, John E., Ray E. Schafer, and Alan Sukert, "Software Reliability Model Validation," *Proceedings Annual Reliability and Maintainability Symposium*, January 1980, pp. 191-199.
37. Littlewood, Bev and John L. Verrall, "Likelihood Function of a Debugging Model for Computer Software Reliability," *IEEE Transactions on Reliability*, Vol. R-30, No. 2, June 1981, pp. 145-148.

38. Joe, H. and N. Reid, "On the Software Reliability Models of Jelinski-Moranda and Littlewood," *IEEE Transactions on Reliability*, Vol. R-34, No. 3, August 1985, pp. 216-218.

APPENDIX

Data Values Plotted, Figure 5-1 Through Figure 5-6

This appendix presents the data values that are plotted in the plots presented in Figure 5-1 through Figure 5-6. All times are given in units of CPU seconds.

The following abbreviations are used for the models:

- J-M – Jelinski-Moranda;
- geom – geometric de-eutrophication;
- L – Littlewood;
- cert – certification;
- G-O – Goel-Okumoto NHPP;
- M-O – Musa-Okumoto.

Table A-1: Data values plotted, Figure 5-1 (Mills failure data).

Failures	Observed Time to Next Failure	Current MTTF Predictions						
		J-M	geom	L	cert	G-O	M-O	Musa
0	85	—	—	—	—	—	—	—
1	85	—	—	—	—	—	—	140.1
2	479	—	—	—	—	—	—	—
3	965	—	917.7	—	852.4	270.3	624.8	—
4	469	—	2283.8	—	2290.8	859.2	1811.5	—
5	385	—	2154.4	—	1586.9	604.9	2183.8	—
6	796	1000.4	1507.0	434.5	1107.5	472.7	1064.2	937.6
7	277	1942.4	1478.1	843.6	1343.9	630.6	1419.6	1172.0
8	927	784.5	975.9	342.9	863.1	471.0	560.4	746.5
9	340	1175.8	1143.0	510.6	1104.9	654.3	791.0	1027.0
10	405	811.5	894.3	352.4	859.2	538.4	674.9	776.5
11	150	688.8	753.7	—	763.7	489.5	651.1	670.5
12	277	503.1	534.5	—	501.3	—	542.7	—
13	503	438.3	440.7	—	443.0	—	490.0	—
14	694	457.9	464.7	—	477.8	—	514.4	—
15	620	530.1	541.8	—	551.0	—	—	—
16	4050	569.9	576.4	—	598.0	—	494.5	—
17	3064	2774.2	1449.5	—	990.8	1784.2	1353.7	2438.1
18	522	5580.7	1966.4	2423.7	1396.6	3028.4	1947.0	—
19	1797	3451.6	1894.4	—	1270.1	2340.9	1908.1	3133.5
20	2329	3791.1	2082.9	—	1499.6	2563.3	1833.2	3389.9
21	3798	7443.0	2360.8	3232.4	1805.2	3033.0	2067.7	4000.0
22	2775	8323.5	2898.3	3614.9	2300.9	4403.6	2559.0	—
23	2393	9203.7	3218.5	—	2669.9	4865.2	2864.6	—
24	909	10094.0	3451.2	—	2950.6	4926.8	3101.9	6445.5
25	994	6027.2	3404.2	2617.6	2747.6	4063.1	3141.1	5135.1
26	28212	4660.6	3350.4	2024.1	2613.2	3499.7	3118.9	4314.5
27	14956	—	7500.2	—	4098.4	—	7980.9	—
28	1971	—	9604.9	—	5539.9	—	10462.0	—
29	5308	—	9980.5	—	5449.1	—	10489.0	—

Table A-2: Data values plotted, Figure 5-2 (Mills failure data).

Failures	Observed Time to Next Failure	Future MTTF Predictions After 25 Failures						
		J-M	geom	L	cert	G-O	M-O	Musa
25	994	6027.2	3404.2	2617.6	2747.6	4063.1	3141.1	5135.1
26	28212	12054.0	3769.4	5235.1	3055.7	5474.0	3445.2	—
27	14956	—	4173.7	—	3398.2	8179.1	3778.8	—
28	1971	—	4621.3	—	3779.1	—	4144.7	—
29	5308	—	5117.0	—	4202.8	—	4546.0	—
30	—	—	5665.8	—	4673.9	—	4986.2	—
31	—	—	6273.4	—	5197.9	—	5468.9	—
32	—	—	6946.3	—	5780.6	—	5998.5	—
33	—	—	7691.3	—	6428.6	—	6579.3	—
34	—	—	8516.3	—	7149.2	—	7216.3	—
35	—	—	9429.7	—	7950.7	—	7915.1	—
36	—	—	10441.0	—	8842.0	—	8681.5	—
37	—	—	11561.0	—	9833.2	—	9522.1	—
38	—	—	12801.0	—	10935.0	—	10444.0	—
39	—	—	14174.0	—	12161.0	—	11454.0	—

Table A-3: Data values plotted, Figure 5-3 (Musa failure data).

Failures	Observed Time to Next Failure	Current MTTF Predictions						
		J-M	geom	L	cert	G-O	M-O	Musa
100	30	2495.1	1226.3	1083.6	734.4	2211.5	1252.9	2361.4
101	143	2236.7	1210.8	—	667.4	2025.6	1233.8	2156.7
102	109	2042.5	1199.3	887.1	646.8	1885.0	1223.8	2002.2
103	0	1890.1	1185.9	820.9	620.8	1754.0	1214.2	1858.8
104	3310	1766.1	1167.4	767.0	498.2	1619.6	1200.7	1712.5
105	1247	2417.7	1275.8	1050.0	547.0	2034.2	1282.5	2158.6
106	943	2469.6	1307.9	—	578.5	2107.1	1330.6	2253.7
107	700	2518.9	1327.5	1093.9	604.0	2117.0	1359.6	2244.6
108	875	2301.1	1337.2	999.3	622.9	2079.6	1376.6	2202.4
109	245	2344.4	1353.4	1018.1	647.0	2075.8	1401.7	2196.7
110	729	2166.7	1344.6	—	640.7	1961.1	1339.2	2071.6
111	1897	2204.0	1354.2	—	660.1	1936.4	1356.1	2043.6
112	447	2252.9	1408.6	978.4	702.9	2109.3	1425.7	2227.9
113	386	2288.5	1407.0	993.9	709.8	2028.9	1429.2	2140.1
114	446	2137.4	1402.5	—	712.7	1945.1	1431.4	2048.8
115	122	2014.0	1400.0	874.7	719.1	1877.1	1381.0	1974.7
116	990	1908.0	1384.5	828.7	693.5	1767.8	1367.3	1856.8
117	948	1939.0	1401.8	—	719.2	1786.9	1398.4	1876.1
118	1082	1969.7	1417.2	855.4	743.9	1798.7	1425.8	1887.6
119	22	2001.7	1437.4	869.4	772.0	1828.0	1457.8	1917.7
120	75	1905.0	1417.2	827.3	702.8	1714.5	1375.6	1795.9
121	482	1731.3	1398.4	751.9	668.0	1618.2	1360.4	1692.7
122	5509	1672.0	1394.6	—	675.7	1577.3	1369.6	1648.5
123	100	2470.8	1576.2	1073.1	739.6	2198.3	1585.2	2308.0
124	10	2320.7	1559.4	—	709.5	2069.2	1574.8	2169.2
125	1071	1063.7	1538.8	—	632.9	1940.1	1505.8	2030.8
126	371	2094.7	1556.1	909.7	656.9	1958.4	1527.5	2049.2
127	790	2004.6	1547.8	—	658.8	1886.1	1524.2	1971.5
128	6150	2031.3	1554.2	882.2	676.3	1869.6	1538.0	1953.1
129	3321	2964.7	1747.9	1287.6	739.3	2597.7	1729.0	2725.9
130	1045	3338.4	1840.5	—	791.0	2965.7	1824.3	3117.0
131	648	3392.9	1856.3	—	815.4	2940.4	1836.8	3088.3
132	5485	3125.4	1858.5	1357.4	827.8	2849.7	1838.8	2989.9
133	1160	3953.6	2019.0	1717.0	895.7	3623.4	1993.1	3814.7
134	1864	4021.0	2038.3	1746.3	923.2	3587.9	2012.4	3774.5
135	4116	4094.5	2079.9	1778.2	964.3	3688.3	2078.8	3879.7

Table A-4: Data values plotted, Figure 5-4 (Musa failure data).

Failures	Observed Time to Next Failure	Future MTTF Predictions After 100 Failures						
		J-M	geom	L	cert	G-O	M-O	Musa
100	30	2495.1	1226.3	1083.6	734.4	2211.5	1252.9	2361.4
101	143	2994.1	1260.0	1300.3	747.5	2553.8	1288.1	2766.8
102	109	3742.6	1294.5	1625.4	781.3	3015.4	1324.2	3331.4
103	0	4990.2	1330.0	2167.2	805.9	3669.1	1361.4	4166.1
104	3310	7485.2	1366.5	3250.8	831.2	4658.4	1399.6	5510.0
105	1247	14970.0	1404.0	6501.6	857.3	6307.6	1438.9	—
106	943	—	1442.5	—	884.2	—	1479.2	—
107	700	—	1482.0	—	912.1	—	1520.8	—
108	875	—	1522.7	—	940.8	—	1563.4	—
109	245	—	1564.4	—	970.4	—	1607.3	—
110	729	—	1607.3	—	1000.9	—	1652.4	—
111	1897	—	1651.4	—	1032.3	—	1698.8	—
112	447	—	1696.7	—	1064.8	—	1746.5	—
113	386	—	1743.3	—	1098.3	—	1795.5	—
114	446	—	1791.1	—	1132.8	—	1845.9	—
115	122	—	1840.2	—	1168.4	—	1897.7	—
116	990	—	1890.7	—	1205.2	—	1951.0	—
117	948	—	1942.5	—	1243.1	—	2005.7	—
118	1082	—	1995.8	—	1282.1	—	2062.0	—
119	22	—	2050.5	—	1322.5	—	2119.9	—
120	75	—	2106.7	—	1364.0	—	2179.4	—
121	482	—	2164.5	—	1406.9	—	2240.5	—
122	5509	—	2223.9	—	1451.2	—	2303.4	—
123	100	—	2284.9	—	1496.8	—	2368.1	—
124	10	—	2347.5	—	1543.9	—	2434.5	—

Table A-4 (Continued)

Failures	Observed Time to Next Failure	Future MTTF Predictions After 100 Failures						
		J-M	geom	L	cert	G-O	M-O	Musa
125	1071	—	2411.9	—	1592.4	—	2502.9	—
126	371	—	2487.1	—	1642.5	—	2573.1	—
127	790	—	2546.0	—	1694.1	—	2645.3	—
128	6150	—	2615.9	—	1747.4	—	2719.6	—
129	3321	—	2687.6	—	1802.3	—	2795.9	—
130	1045	—	2761.3	—	1859.0	—	2874.4	—
131	648	—	2837.0	—	1917.5	—	2955.0	—
132	5485	—	2914.9	—	1977.8	—	3038.0	—
133	1160	—	2994.8	—	2039.9	—	3123.2	—
134	1864	—	3076.9	—	2104.1	—	3210.9	—
135	4116	—	3161.3	—	2170.3	—	3301.0	—
136	—	—	3248.0	—	2238.5	—	3393.7	—
137	—	—	3337.1	—	2308.9	—	3488.9	—
138	—	—	3428.6	—	2381.5	—	3586.8	—
139	—	—	3522.6	—	2456.4	—	3687.5	—
140	—	—	3619.3	—	2533.6	—	3791.0	—
141	—	—	3718.5	—	2613.3	—	3897.4	—
142	—	—	3820.5	—	2695.4	—	4006.8	—
143	—	—	3925.3	—	2780.2	—	4119.2	—
144	—	—	4032.9	—	2867.6	—	4234.8	—
145	—	—	4143.5	—	2957.8	—	4353.7	—
146	—	—	4257.2	—	3050.8	—	4475.9	—
147	—	—	4373.9	—	3146.7	—	4601.5	—
148	—	—	4493.9	—	3245.7	—	4730.6	—
149	—	—	4617.1	—	3347.7	—	4863.4	—

Table A-5: Data values plotted, Figure 5-5 (Abdel-Ghaly failure data).

Failures	Observed Time to Next Failure	Current MTTF Predictions						
		J-M	geom	L	cert	G-O	M-O	Musa
50	7.14	15.0	12.8	—	6.9	13.6	12.5	14.7
51	1.81	14.4	12.6	6.2	7.0	13.1	12.3	14.2
52	14.85	13.1	12.0	5.7	6.4	12.1	11.8	13.0
53	7.57	13.7	12.4	6.0	6.9	12.7	12.2	13.6
54	31.54	13.4	12.3	5.8	7.0	12.4	12.1	13.3
55	21.15	16.3	13.8	—	7.9	14.9	13.7	16.0
56	8.84	17.5	14.6	7.6	8.5	16.1	14.5	17.3
57	20.37	17.0	14.5	—	8.7	15.7	14.4	16.9
58	14.81	18.5	15.2	—	9.3	16.8	15.1	18.0
59	5.59	18.5	15.5	8.0	9.8	17.1	15.4	18.4
60	4.90	17.5	15.1	7.6	9.6	16.3	15.0	17.4
61	5.93	16.8	14.8	—	9.3	15.4	14.7	16.5
62	17.69	15.9	14.5	—	9.2	14.8	14.5	15.8
63	0.85	16.7	14.9	7.2	9.7	15.5	15.1	16.5
64	28.36	15.3	14.3	—	8.5	14.3	14.7	15.2
65	2.13	17.2	15.4	—	9.2	16.1	15.2	17.1
66	18.66	16.1	14.8	—	8.5	15.0	14.7	16.0
67	4.90	16.8	15.3	—	9.1	15.7	15.4	16.7
68	14.87	16.1	14.9	—	8.8	15.0	15.3	15.9
69	43.22	16.3	15.2	7.1	9.2	15.3	15.7	16.2
70	14.18	19.9	17.0	—	10.2	18.5	16.9	19.6
71	10.23	19.9	17.2	8.6	10.5	18.5	17.0	19.7
72	54.90	19.4	17.1	—	10.7	18.2	17.0	19.3
73	15.20	24.4	19.4	—	11.8	22.8	19.0	24.2
74	32.81	24.8	19.6	—	12.2	22.8	19.1	24.2
75	27.16	27.0	20.6	—	13.0	24.9	20.4	26.5
76	21.75	28.5	21.4	12.4	13.8	26.3	21.0	27.9
77	35.05	29.0	21.8	12.6	14.4	26.9	21.3	28.6
78	7.25	32.0	22.9	13.9	15.3	29.2	22.2	31.1
79	19.63	30.0	22.6	13.0	15.1	27.8	22.0	29.5
80	39.79	30.6	22.9	13.3	15.5	28.1	22.8	29.8
81	10.90	33.8	24.2	14.7	16.6	30.9	24.0	32.8
82	2.45	32.9	24.1	14.3	16.6	29.9	23.9	31.7
83	11.94	30.0	23.5	13.0	15.4	28.0	23.5	29.6
84	9.94	29.5	23.4	—	15.5	27.3	23.6	28.9
85	39.02	28.2	23.2	—	15.4	26.5	22.8	27.9

Table A-6: Data values plotted, Figure 5-6 (Abdel-Ghaly failure data).

Failures	Observed Time to Next Failure	Future MTTF Predictions After 50 Failures					
		J-M	geom	cert	G-O	M-O	Musa
50	7.14	15.0	12.7	6.9	13.6	12.5	14.7
51	1.81	15.6	13.0	7.0	14.0	12.7	15.3
52	14.85	16.3	13.3	7.1	14.4	13.0	15.9
53	7.57	17.0	13.5	7.1	14.9	13.2	16.6
54	31.54	17.8	13.8	7.2	15.5	13.5	17.3
55	21.15	18.7	14.0	7.3	16.0	13.7	18.1
56	8.84	19.7	14.3	7.4	16.7	14.0	19.0
57	20.37	20.8	14.6	7.4	17.3	14.2	20.0
58	14.81	22.0	14.9	7.5	18.0	14.5	21.1
59	5.59	23.4	15.2	7.6	18.8	14.8	22.3
60	4.90	25.0	15.5	7.7	19.7	15.0	23.7
61	5.93	26.7	15.8	7.8	20.6	15.3	25.2
62	17.69	28.8	16.1	7.8	21.6	15.6	27.0
63	0.85	31.2	16.4	7.9	22.7	15.9	28.9
64	28.36	34.0	16.7	8.0	24.0	16.2	31.3
65	2.13	37.4	17.0	8.1	25.3	16.5	34.0
66	18.66	41.6	17.4	8.2	26.9	16.8	37.2
67	4.90	46.8	17.7	8.3	28.6	17.1	41.0
68	14.87	53.5	18.1	8.3	30.6	17.4	45.7
69	43.22	62.4	18.4	8.4	32.9	17.7	51.6
70	14.18	74.9	18.8	8.5	35.5	18.0	59.2
71	10.23	93.6	19.1	8.6	38.6	18.4	69.2
72	54.90	124.8	19.5	8.7	42.4	18.7	83.1
73	15.20	187.2	19.9	8.8	46.6	19.1	103.6
74	32.81	374.3	20.3	8.9	51.9	19.4	136.3

Table A-6 (Continued)

Failures	Observed Time to Next Failure	Future MTTF Predictions After 50 Failures					
		J-M	geom	cert	G-O	M-O	Musa
75	27.16	—	20.7	9.0	58.6	19.8	—
76	21.75	—	21.1	9.1	67.3	20.1	—
77	35.05	—	21.5	9.1	78.7	20.5	—
78	7.25	—	21.9	9.2	94.6	20.9	—
79	19.63	—	22.4	9.3	118.1	21.3	—
80	39.79	—	22.8	9.4	155.6	21.7	—
81	10.90	—	23.2	9.5	—	22.1	—
82	2.45	—	23.7	9.6	—	22.5	—
83	11.94	—	24.2	9.7	—	22.9	—
84	9.94	—	24.6	9.8	—	23.3	—
85	39.02	—	25.1	9.9	—	23.7	—
86	—	—	25.6	10.0	—	24.2	—
87	—	—	26.1	10.1	—	24.6	—
88	—	—	26.6	10.2	—	25.1	—
89	—	—	27.1	10.4	—	25.5	—
90	—	—	27.7	10.5	—	26.0	—
91	—	—	28.2	10.6	—	26.5	—
92	—	—	28.8	10.7	—	27.0	—
93	—	—	29.3	10.8	—	27.5	—
94	—	—	29.9	10.9	—	28.0	—
95	—	—	30.5	11.0	—	28.5	—
96	—	—	31.1	11.1	—	29.0	—
97	—	—	31.7	11.2	—	29.6	—
98	—	—	32.3	11.4	—	30.1	—
99	—	—	32.9	11.5	—	30.7	—

VITA

Clifford Kimbrough Cobb was born in New Orleans, Louisiana on October 15, 1959. He graduated from Farragut High School in Concord, Tennessee in June 1977. The following September he entered the University of Tennessee, Knoxville, and in June 1981 he received the Bachelor of Arts degree with a major in Computer Science. Upon graduation, he accepted a position as a computing analyst with Union Carbide Corporation, Nuclear Division, operators of government installations for the United States Department of Energy in Oak Ridge, Tennessee.

Mr. Cobb entered The Graduate School of the University of Tennessee, Knoxville in September 1983. He received the Master of Science degree with a major in Computer Science in June 1988.

Mr. Cobb is a member of Phi Beta Kappa. He is currently employed as a computing specialist by Martin Marietta Energy Systems, Inc., operators of the Department of Energy installations in Oak Ridge since 1984.