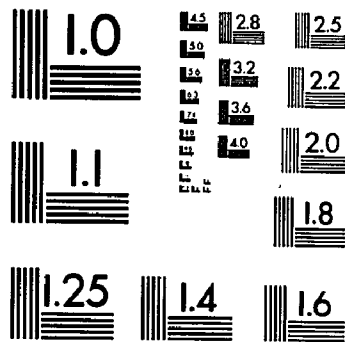


U·M·I



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

University Microfilms International
A Bell & Howell Information Company
300 N. Zeeb Road, Ann Arbor, Michigan 48106

INFORMATION TO USERS

This reproduction was made from a copy of a manuscript sent to us for publication and microfilming. While the most advanced technology has been used to photograph and reproduce this manuscript, the quality of the reproduction is heavily dependent upon the quality of the material submitted. Pages in any manuscript may have indistinct print. In all cases the best available copy has been filmed.

The following explanation of techniques is provided to help clarify notations which may appear on this reproduction.

1. Manuscripts may not always be complete. When it is not possible to obtain missing pages, a note appears to indicate this.
2. When copyrighted materials are removed from the manuscript, a note appears to indicate this.
3. Oversize materials (maps, drawings, and charts) are photographed by sectioning the original, beginning at the upper left hand corner and continuing from left to right in equal sections with small overlaps. Each oversize page is also filmed as one exposure and is available, for an additional charge, as a standard 35mm slide or in black and white paper format.*
4. Most photographs reproduce acceptably on positive microfilm or microfiche but lack clarity on xerographic copies made from the microfilm. For an additional charge, all photographs are available in black and white standard 35mm slide format.*

***For more information about black and white slides or enlarged paper reproductions, please contact the Dissertations Customer Services Department.**

U·M·I Dissertation
Information Service

University Microfilms International
A Bell & Howell Information Company
300 N. Zeeb Road, Ann Arbor, Michigan 48106

8617947

Friedman, Michael Allan

MODELING THE PENALTY COSTS OF SOFTWARE FAILURE

University of California, Irvine

PH.D. 1986

University
Microfilms
International 300 N. Zeeb Road, Ann Arbor, MI 48106

Copyright 1986

by

Friedman, Michael Allan

All Rights Reserved

PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark .

1. Glossy photographs or pages _____
2. Colored illustrations, paper or print _____
3. Photographs with dark background _____
4. Illustrations are poor copy _____
5. Pages with black marks, not original copy _____
6. Print shows through as there is text on both sides of page _____
7. Indistinct, broken or small print on several pages
8. Print exceeds margin requirements _____
9. Tightly bound copy with print lost in spine _____
10. Computer printout pages with indistinct print _____
11. Page(s) _____ lacking when material received, and not available from school or author.
12. Page(s) _____ seem to be missing in numbering only as text follows.
13. Two pages numbered _____. Text follows.
14. Curling and wrinkled pages _____
15. Dissertation contains pages with print at a slant, filmed as received _____
16. Other _____

University
Microfilms
International

UNIVERSITY OF CALIFORNIA

Irvine

MODELING THE PENALTY COSTS
OF SOFTWARE FAILURE

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in Information and Computer Science

by

Michael Allan Friedman

Committee in charge:


Professor Nancy G. Leveson, Chair

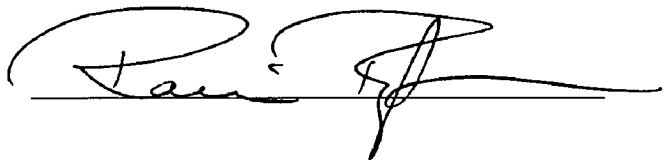
Professor Rami R. Razouk

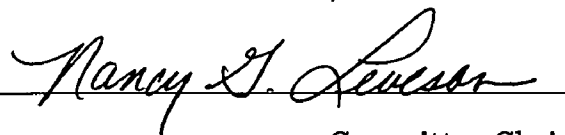
Professor Roger E. Whitney

1986

The dissertation of Michael Allan Friedman is approved,
and is acceptable in quality and form for
publication on microfilm:







Committee Chair

University of California, Irvine

1986

© 1986

Michael Allan Friedman

ALL RIGHTS RESERVED

Dedication

To my parents.

Contents

List of Tables	vii
List of Figures	viii
Acknowledgements	x
Curriculum Vitae	xi
Abstract	xii
Chapter 1: INTRODUCTION.	1
Background.	1
Statement of the Problem	4
Method of Research.	5
Contributions to the Field.	5
Synopsis of the Dissertation.	6
Chapter 2: PRELIMINARY CONCEPTS	8
Probability Concepts	8
Reliability Concepts	16
Software Reliability Concepts.	22
Software Reliability and Hardware.	32
Chapter 3: REVIEW OF RELATED RESEARCH.	34
Time-Domain Models.	35
Bayesian Models.	50
Markov Models	58
Data-Domain Models.	59
Phenomenological Models	60
Fault Seeding.	61
Discussion.	62
Chapter 4: DEVELOPMENT OF THE MODELING TECHNIQUE.	64
Statement of the Question.	64

Overview of the Modeling Technique	64
The Scenario	69
A Single Released Program Version	70
Delivered Failure Intensities	86
Failure-Counting Component: Variants	92
Failure Counting Process Estimation	97
Chapter 5: INCORPORATING PENALTY COSTS	99
Penalty Cost Component	99
Penalty Cost Distributions	103
Distribution of Aggregate Penalty Costs	103
Distributional Properties	107
Homogeneous Poisson Failure-Counting Component	108
Derivation by Severity Class	108
Pólya Variation	109
Some Examples	112
Use of Auxiliary Functions	119
Lattice Penalty-Cost Distribution	124
Penalty Cost Vectors	125
Chapter 6: A REAL-LIFE EXAMPLE	127
Failure Rate Estimation	130
Summary of How to Use the Modeling Technique	133
Chapter 7: SAMPLE SIZE DETERMINATION	135
Reporting Uncertainty	135
Sample Size for Determining Failure Intensity	141
Sample Size for Determining Aggregate Penalty Cost	142
Chapter 8: CONCLUSIONS	145
Summary	145
Future Research Directions	146

Conclusion	147
References	149
Appendix A: Data for Bayesian Failure Rate Updating Example	158
Appendix B: Data for Robust Program Example	170
Appendix C: Data for Space Shuttle Example	175
Appendix D: "Flow Diagrams"	186

List of Tables

Table	Page
1. Raw Data for Bayesian Updating Example	159
2. Best Estimate of Failure Rate	165
3. Raw Data for Robust Program Example	171
4. Raw Data for Space Shuttle Example	176
5. Aggregate Penalty Cost PMF	178
6. Aggregate Penalty Cost CDF	182

List of Figures

Figure	Page
1. Probability Distribution of a Discrete Random Variable	10
2. Probability Distribution of a Continuous Random Variable	11
3. Reliability Function and Cumulative Distribution Function	17
4. Typical Hardware Hazard Rate (Bathtub Curve)	19
5. Combinatorial Reliability	21
6. Fault Sources.	23
7. Classic Software Life Cycle Model.	26
8. Negative Exponential Distribution.	37
9. Rayleigh Distribution.	44
10. Weibull Distribution	46
11. Gamma Distribution	51
12. Pareto Distribution	53
13. Four-State Markov Model	57
14. Visualization of Time.	68
15. Two Random Failure Variables.	71
16. Software Failure Recurrence Times	72
17. The Counting Process $N(t)$	73
18. State Transitions of the Failure Counting Process	77
19. Poisson Probabilities	79
20. Failure-Counting Process Decomposition/Superposition	83
21. Two Bayesian Prior Distributions for λ	87
22. Best Estimate of Failure Rate	90
23. Negative Binomial Distribution	93
24. Poisson vs. Pólya Processes.	94
25. Forms of Penalty Cost Distribution X	101

26.	A Realization of Aggregate Penalty Cost Process	102
27.	The Aggregate Penalty Cost Process $\mathbf{Z}(t)$	104
28.	Normal Approximation to Poisson.	110
29.	Standard Normal Approximation	111
30.	Lognormal Distribution	114
31.	Comparison of Exponential, Lognormal, Pareto Densities	115
32.	Probability Generating Function of Poisson R.V.	120
33.	Weekly Test Effort	128
34.	Interval Coefficients.	136
35.	Student's t -Distribution	137
36.	Standard Normal <i>vs.</i> Student's t -Distribution	139
37.	Aggregate Penalty Cost pmf	181
38.	Aggregate Penalty Cost Cdf	185
39.	Typical Time-Domain Model	187
40.	Model with Penalty Costs	188

Acknowledgements

I am grateful to TRW, Inc., and Delta Resources, Inc., for financial assistance from their employee educational reimbursement programs, and for according me the flexibility being a student requires.

Raw space shuttle software failure/penalty cost data appearing in Appendix C: "Copyright 1983 International Business Machines Corporation. Reprinted with permission from the *IBM Systems Journal*, Vol. 22, No. 3, 1983"

Curriculum Vitae Michael Allan Friedman

1/14/58 Born Chicago, Illinois

1976-1977 Computer Studies Program, Northwestern University,
Evanston, Illinois

1977-1981 Programmer/Analyst, UCI Water Resources Laboratory

1980 BS (honors) in Information and Computer Science,
University of California, Irvine

1981-1984 Software Engineer/Task Leader, TRW Inc.; Torrance, CA

1984 MS in Information and Computer Science,
University of California, Irvine

1984+ Group Leader, Delta Resources Inc.; Anaheim, CA

1986 Ph.D. in Information and Computer Science,
University of California, Irvine.
Dissertation: "Modeling the Penalty Costs of
Software Failure."

HONORS

Illinois State Scholar

Phi Beta Kappa

Upsilon Pi Epsilon (National Honor Society for the Computing
Sciences)

Mensa Society

California State Graduate Fellow

Abstract of the Dissertation

MODELING THE PENALTY COSTS OF SOFTWARE FAILURE

by

Michael Allan Friedman

Doctor of Philosophy in Information and Computer Science

University of California, Irvine, 1986

Nancy G. Leveson, Chair

An improved software reliability modeling technique is presented that takes into account the penalty costs of software failures. "Penalty cost" is a quantification of the undesirable consequences of the failure, sometimes called a "severity rating." Mathematically the model is developed as a compound stochastic process with failure frequency and severity components. The main purpose of the technique is to probabilistically characterize the aggregate penalty cost to be incurred over a future time interval. Numerical techniques for facilitating calculations are provided, with special emphasis on two major classes of programs: "robust" programs and those with lattice penalty cost distributions. Use of the modeling technique is demonstrated with NASA space shuttle data. The relationship between accuracy and sample size is investigated.

CHAPTER 1

INTRODUCTION

Background

Reliability is a major consideration in the planning, design and operation of today's large, intricate hardware-software systems. It is especially important when computers lie at the heart of critical or vital real-time applications. Computer science, as a maturing discipline, is following the age-old trend of replacing art by science, and scientific method requires quantification as its first step. When software reliability is defined quantitatively, it can be specified, analyzed, measured and modeled. Informally, software reliability is the probability that a computer program will operate successfully for a specified period of time, under stated environmental conditions. Successful operation means that output (displays, hardcopies, commands, control, etc.) does not deviate beyond specified tolerances. Failure of the software may result in an unintended system state or course of action. A loss event could ensue in which property is damaged or destroyed, people are injured or killed, monetary costs are incurred. A quantitative measure of the loss is called the *penalty cost* of that failure.

Software reliability assessment, part of the broader area of software quality assessment[MOHA73], is still in its embryonic stages. In the early seventies, IBM's admission[HOPK70] that each release of its OS/360-370 operating systems contained over 1,000 faults ("bugs") shocked many people. Ironically, a follow-up study of error logs showed that the number was usually closer to 11,000[GILB74]. For persons newly acquainted with computer systems,

“[i]t is indeed a surprising thing to discover that products called software, created by reputable companies and sold or rented—for a reasonable or unreasonable price—will inevitably contain, not just a few errors, but masses of them[GILB74].”

As a practical matter, 100% reliability is unattainable; most software reliability models suggest that the effort to uncover each succeeding fault grows exponentially. And even if sufficient debugging resources were available, it is problematical to ascertain whether 100% reliability has in fact been achieved, since the two known ways of showing the absence of faults are exhaustive testing and formal verification. Exhaustive testing, whether functional (all possible input streams) or structural (all possible program paths), gives rise to a combinatorial explosion of cases. Formal verification of sizable real-time systems is likewise not currently within the realm of feasibility.

Given, then, that computer programs will contain faults, the practical goal of software development must be to achieve a high degree of software reliability that leaves an acceptable level of risk of software failure. “Acceptable level” presupposes that software reliability can be communicated and assessed. To communicate about software reliability requires agreed-upon figures of merit, and assessment of software reliability requires demonstrably valid techniques for obtaining the values of these metrics. Current techniques are designed to serve three principal functions [HECH77]:

1. Measurement—evaluation of a program’s reliability as it executes in its actual operating environment.
2. Estimation—extrapolation from a program’s failure history in a test environment.
3. Prediction—correlation of static program characteristics to failure behavior.

Once a reliability goal is established, reliability models can be employed to assist project management in planning to achieve that goal. For example, models

can help determine the resources required to meet the goal, and inform when it has been achieved. Considered more broadly, software reliability is not only a measure of system effectiveness but also bears on the total cost of system acquisition and ownership throughout the life cycle. Building-in greater reliability generally costs more up front but can save maintenance and penalty costs later down the line. Software reliability analysis provides information to allow tradeoffs to be made with other product characteristics and can help evaluate the effects of competing software development technologies.

Sound software engineering practices can go a long way in hindering the introduction of faults into the software. Unfortunately, advances in software development technology have lagged behind the advances in computer hardware that permit the construction of systems of great size and complexity. It has been suggested that the use of fault tolerance techniques might allow critical systems to recover from software failures in real time[KIM84].

Anecdotal news accounts of lost space probes, false military alerts, and persistent overbillings have pointed up the computer's knack for amplifying human error. During the late '60s, problems with software reliability were diagnosed as a symptom of a pervasive "software crisis" [DIJK68]. Software reliability has recently been thrust into the public eye when it was named as the Achilles' heel in the realization of proposed comprehensive strategic defense systems[LIN85]. The trends of more and more computing power per dollar, and greater and greater miniaturization, have resulted in ever-increasing application-areas for computers. Modern society is letting itself become dependent on computers in important areas like national defense, transportation, energy, medicine, law, finance and manufacturing. As computers are given more and more "responsibility," they are put into positions where incorrect operation can result in significant consequences. Computers by themselves can do no harm, but when they are embedded in systems their output is endowed with the power to inform or materially affect the real world. Examples:

- 1.) **Physical systems:** Misdirected mechanical, chemical, electromagnetic, thermodynamic, nuclear and other energies can result in damage to property and living organisms. Transportation, power utilities, manufacturing and weapons systems especially can involve powerful energies.
- 2.) **Biological systems:** Failure of lifesaving or life-support systems may result in immediate loss of life or limb. Failure in medical information systems and test equipment can more subtly endanger life and health. Computers can form a crucial part of artificial organs, prosthetic and other “bionic” devices.
- 3.) **Commercial systems:** Computers failing in accounting, banking, billing, financial analysis, management information, market research systems, and so on, could result in monetary loss to firms and individuals.

In summary, when computers are entrusted with the “power” or “authority” to perform useful work, the potential for serious undesired consequences exists should software failure occur.

Statement of the Problem

Something is missing in current software reliability models. They merely count software failures and do not take into account the differing severities of the failures. However, as Goodenough[GOOD79] and others have pointed out, software reliability must concern itself with both the frequency and severity of software failure. Presently, “An error in the tenth digit of a calculation is judged as harshly as a crash-causing bug[BEIZ84].” Failure can cause effects ranging from aesthetic affront or minor irritation to grave effects on the system and its environment. Even where loss is purely financial, penalty cost information might allow software engineers to relate system quality to economics and capital investment.

What is needed, then, is a new, expanded type of software reliability model that can weight software failures according to the magnitudes of their undesired

consequences. Such a model would be capable of probabilistically characterizing the aggregate penalty cost to be incurred in a future time interval.

Software reliability, as currently measured, is independent of *software safety*. However, incorporating failure severities can help unite these two areas [LEVE82].

Method of Research

The theoretical basis of the modeling technique comes from modern probability calculus, especially the theory of random functions. The theory of random functions has advanced rapidly since the pioneering work of Kolmogorov [KOLM31] and Feller [FELL36]. The subject is systematically treated in [CHUN60], [DOOB53], [FELL57, FELL66] and [HARR63]. Random functions (stochastic processes and random sequences) have been shown to be of great unifying power and have been applied in many diverse areas. By viewing the phenomenon of software failure in this context and carefully delineating assumptions, a family of models was developed that includes existing software reliability models as special cases. This family of models improves upon those existing models by taking into account the consequences of software failure.

Contributions to the Field

The research described in this work has contributed to the field of software reliability by providing a new modeling technique that extends the province of software reliability models to include penalty costs. Before this technique was developed, software reliability models could only deal with the frequency of failures. One of the major criticisms of the existing models has been their inability to distinguish among failures with differing degrees of severity of consequence.

A nice feature of the technique is that it is compatible with existing software reliability models and serves to enhance their utility. The technique models the phenomena of software failure in a more comprehensive way than the others because

of the additional dimension of penalty cost. Methods for measuring and predicting software reliability, such as this one, are useful to several software development roleplayers:

Software engineering researchers can use the methods to demonstrate the effectiveness and efficiency of proposed software development technologies. The reliability growth to be realized from the techniques can be studied and compared with competing technologies.

Project managers can use measurement and prediction methods to get a handle on when reliability goals will be met and what additional effort needs to be expended to meet those goals. The managers can use the researchers' results to help decide whether and how extensively to deploy competing software development technologies in pursuit of the goals.

A procuring organization can state in a requirements specification or contract a maximum acceptable yearly aggregate penalty cost or similar metric, and the developing organization will be able to the software has indeed met the required level of reliability.

Users can evaluate the penalty-cost reliability of operational systems. Total life-cycle costs of a piece of software can be predicted more accurately by including penalty cost estimates.

Systems safety personnel can use penalty-cost predictions to help decide when software is sufficiently safe to be released.

Synopsis of the Dissertation

After covering some preliminary definitions, concepts and results from probability and reliability theory, the dissertation surveys the literature relating to software reliability modeling. Here the main software reliability models are chronicled and compared, culminating in an assessment of the state of the art. The succeeding chapter begins by technically summarizing the main features of the

model and its principal formulae. Then the new modeling technique is carefully developed from a brief series of reasonable assumptions about failure occurrence and severity. Concepts from modern probability calculus and the theory of random functions are employed. A Bayesian technique for updating the failure rate is presented along with an example using real project data. Variants of the basic model are discussed. Convenient numerical techniques are covered for some major classes of computer programs. An example is given using failure data from a “robust” program. The use of auxiliary functions as a way of simplifying computations is discussed. An application of the modeling technique is demonstrated using failure data from NASA space shuttle software. The relation between sample size and accuracy of the technique is analyzed.

The main body of the dissertation ends with a summary, a discussion of conclusions, and suggestions for future research. In the appendices appear the raw data for all the examples, and detailed results from the space shuttle example. Illustrations appear throughout the dissertation to assist in visualizing the main ideas.

CHAPTER 2

PRELIMINARY CONCEPTS

Software reliability modeling techniques are grounded in probability calculus and reliability theory. It will be helpful to review the basic definitions and notions that will be drawn upon in the remainder of this work.

Probability Concepts

A *statistical experiment* can be real or purely conceptual and is one in which “a complex natural background leads to a chance outcome [HAST75].” A fundamental concept is that of a *probability space*, denoted by the triple $(\Omega, \mathcal{F}, \mathcal{P})$. The probability space serves to specify the basic objects on which probability calculations will be performed. The first component is the *sample space* Ω , which consists of the collectively exhaustive and mutually exclusive outcomes ω of the experiment. The second component is a σ -field \mathcal{F} —a nonempty collection of subsets (events) that is closed under the operations of complement, union, and infinite union. The third component is a *probability measure* \mathcal{P} that maps each event into the unit interval.

In this work, intervals on the real line will be specified by means of the following notational conventions: For real numbers a and b with $a < b$,

$$(a, b) \equiv \{x | a < x < b\};$$

$$[a, b] \equiv \{x | a \leq x \leq b\};$$

$$(a, b] \equiv \{x | a < x \leq b\};$$

$$[a, b) = \{x | a \leq x < b\}.$$

Thus, a parenthesis indicates that the endpoint is to be excluded from the interval, and a square bracket indicates that it is to be included. The “unit interval” would be notated $[0, 1]$.

The function \mathcal{P} must be such that

- 1) $\mathcal{P}(\Omega) = 1$ (standardization),
- 2) if A_1, A_2, \dots are disjoint events, then $\mathcal{P}(\cup_i A_i) = \sum_i \mathcal{P}(A_i)$ (additivity), and
- 3) $0 \leq \mathcal{P}(A) \leq 1$ (nonnegativity).

The notation $\Pr\{B\}$ means $\mathcal{P}(\{\omega \in \Omega : B \text{ holds}\})$.

Random Variables

A random variable $\mathbf{X}(\omega)$ is a real-valued function defined over the sample space Ω with the proviso that for every real number x , the set $\{\omega : \mathbf{X}(\omega) \leq x\} \in \mathcal{F}$. It is customary in most contexts to suppress the functional form and simply refer to the random variable as \mathbf{X} . In this work random variables will appear in boldface. Informally, a random variable can be understood as a numerical representation of the outcome of the statistical experiment.

Conditional Probabilities

The notation $\mathcal{P}(Y|X)$ denotes the probability of the event Y *given that the event X has occurred* and satisfies

$$\mathcal{P}(Y|X) = \frac{\mathcal{P}(X \cap Y)}{\mathcal{P}(X)}.$$

Two events are said to be statistically independent (“s-independent”) if $\mathcal{P}(X \cap Y) = \mathcal{P}(X) \cdot \mathcal{P}(Y)$.

Probability Distributions

The probability distribution of a random variable \mathbf{X} is usually represented by its *cumulative distribution function* (Cdf)

$$\text{Cdf}\{\mathbf{X}\} \equiv F(x) \equiv \Pr\{\mathbf{X} \leq x\}.$$

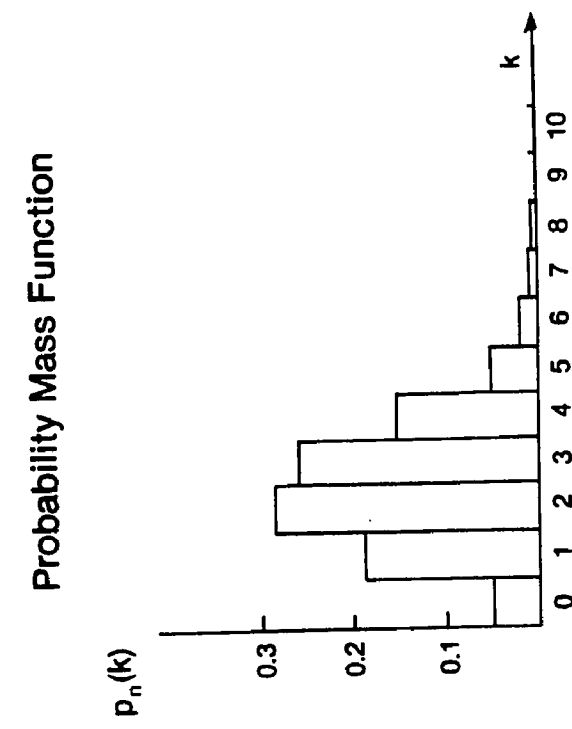
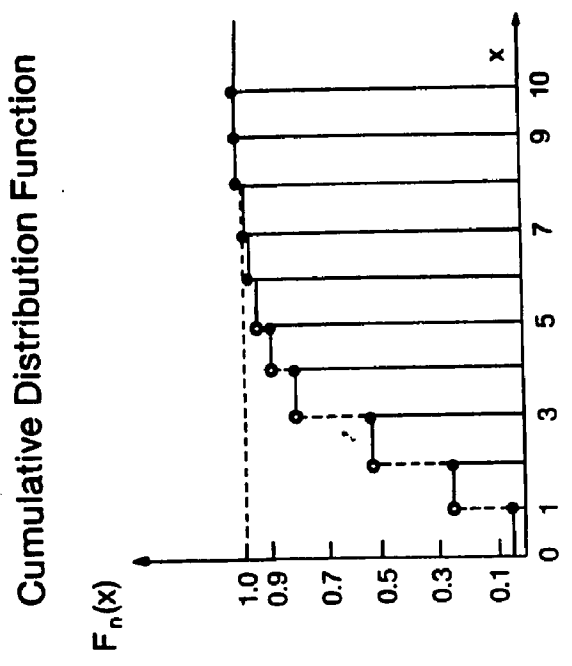
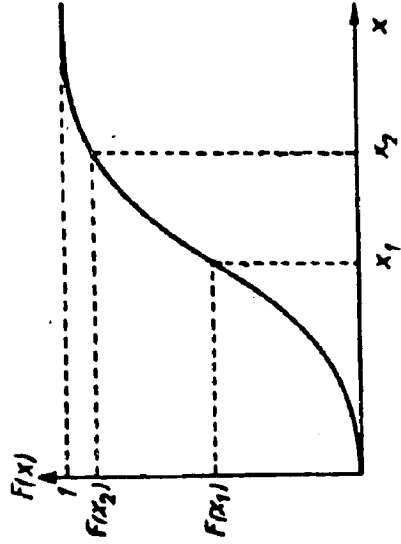
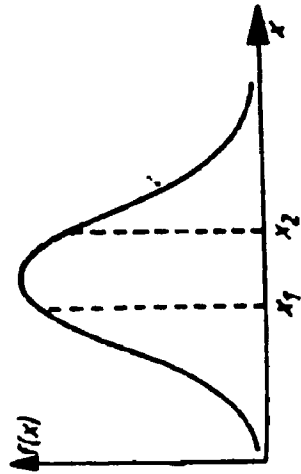


FIGURE 1
PROBABILITY DISTRIBUTION OF A DISCRETE RANDOM VARIABLE



Cdf



pdf

FIGURE 2
PROBABILITY DISTRIBUTION OF A CONTINUOUS RANDOM VARIABLE

It gives the probability that \mathbf{X} will not exceed the value x . The form with the random variable in curly braces is the official IEEE notation for the function. The function exhibits these basic properties:

- 1) It is nondecreasing,
- 2) $\lim_{x \rightarrow -\infty} F(x) = F(-\infty) = 0$,
- 3) $\lim_{x \rightarrow +\infty} F(x) = F(+\infty) = 1$,
- 4) $F(x)$ is continuous from the right: $\lim_{x \rightarrow x_0} F(x) = F(x_0)$, $x > x_0$.

Alternatively, if \mathbf{X} is a discrete random variable (can only take on a countable set of values), it can be represented by its *probability mass function* (pmf)

$$\text{pmf}\{\mathbf{X}\} \equiv p_x \equiv \Pr\{\mathbf{X} = x\}.$$

Events having nonzero probabilities are represented as points. (In this case $F(x)$ is a step function.) Figure 1 shows the probability mass function and cumulative distribution function of a typical discrete random variable. If \mathbf{X} is continuous (can take on any real number in an interval), it can be represented by its *probability density function* (pdf)

$$\text{pdf}\{\mathbf{X}\} \equiv f(x) \equiv \lim_{\Delta x \rightarrow 0^+} \frac{\Pr\{x < \mathbf{X} < x + \Delta x\}}{\Delta x} = \frac{d\text{Cdf}\{\mathbf{X}\}}{dx},$$

where this derivative exists. The pdf exhibits these fundamental properties:

- 1) It is nonnegative: $f(x) \geq 0$.
- 2) The total probability is unity: $\int_{-\infty}^{\infty} f(x) dx = 1$.
- 3) The definite integral gives probabilities: $\Pr\{a \leq \mathbf{X} \leq b\} = \int_a^b f(x) dx$. [Either or both inequalities can be strict (“<”) since the probabilities at single points are zero.]

Events are represented as sets of intervals on the real line. Figure 2 shows the probability density function and cumulative distribution function of a typical

continuous random variable. The pdf, pmf and Cdf should be presumed zero for unstated values of their arguments.

Descriptive Statistics

The *s*-expected value (also called the *expectation* or *mean*) of a continuous random variable \mathbf{X} is defined to be

$$E\{\mathbf{X}\} = \int_{-\infty}^{\infty} x f(x) dx$$

if $\int_{-\infty}^{\infty} |x| f(x) dx < \infty$. This corresponds to the center of gravity of the distribution.

When \mathbf{X} is discrete, the *s*-expected value is

$$E\{\mathbf{X}\} = \sum_i x_i \Pr\{\mathbf{X} = x_i\}$$

if $\sum_i |x_i| \Pr\{\mathbf{X} = x_i\} < \infty$.

The *variance* is defined to be

$$\text{Var}\{\mathbf{X}\} \equiv E\{[\mathbf{X} - E\{\mathbf{X}\}]^2\} = E\{\mathbf{X}^2\} - [E\{\mathbf{X}\}]^2.$$

It gives a measure of dispersion—how concentrated the distribution is about the center of gravity. The *standard deviation* is the square root of the variance. More generally, the *moments about the origin* are

$$\mu'_r = E\{\mathbf{X}^r\}, \quad r = 1, 2, \dots$$

and the *moments about the mean* are

$$\mu_r = E\{(\mathbf{X} - \mu)^r\}, \quad r = 1, 2, \dots$$

The *median* is the “middlemost value,” that value of M for which

$$\int_{-\infty}^M f(x) dx = \int_M^{\infty} f(x) dx = \frac{1}{2}.$$

The *mode* is a “most likely value”—a value of x for which

$$\frac{df(x)}{dx} = 0, \frac{d^2f(x)}{dx^2} < 0.$$

Some distributions may have more than one mode.

Random Functions

A *random function* $\underline{\mathbf{X}} = \{\mathbf{X}(t, \omega); t \in T, \omega \in \Omega\}$ is a family of random variables defined on the same probability space and parameterized by an independent deterministic variable t that takes on values from an *index set* T . If the indexing parameter t varies along a continuum, $\underline{\mathbf{X}}$ is termed a *stochastic process*; if t can take on only discrete values, $\underline{\mathbf{X}}$ is called a *random sequence*. The values that a particular random variable $\mathbf{X}(t, \omega)$ can assume comprise the *state space* of the random function. When the index set T is the real line \mathcal{R} and is thought of as “time,” random functions can be used to describe systems that evolve in time according to probabilistic laws. For each fixed $t \in T$, $\mathbf{X}(t, \omega)$ will be a real-valued random variable that is a function of ω . Contrarily, for each fixed $\omega \in \Omega$, $\mathbf{X}(t, \omega)$ will be a function of time called a *realization* of the random function. The set of all such time functions is called the *ensemble* of the random function.

Etymologically, the word *stochastic* comes from the Greek *στοχάζεσθαι*, meaning “to shoot with a bow at a target.” The picture is this: The trajectories of the arrows are at least partially random, and some will achieve the preferred outcome (hit the target). To abstract, there are 1) a stream of events that is not exactly predictable (has a random component) and 2) a nonrandom selection process that classifies the outcomes [BATE79].

Convolution

Let $\mathbf{Z} = \mathbf{X} + \mathbf{Y}$, where \mathbf{X} , \mathbf{Y} , and \mathbf{Z} are defined on the same probability space.

The pdf of \mathbf{Z} can be computed from the pdf's of \mathbf{X} and \mathbf{Y} by

$$\begin{aligned} f_Z(z) &= f_X(x) * f_Y(y) \\ &= \int_{-\infty}^{\infty} f_X(z-y) f_Y(y) dy \\ &= \int_{-\infty}^{\infty} f_X(x) f_Y(z-x) dx, \end{aligned}$$

where “*” is called the *convolution operator*.

Laplace Transform

The mathematical constant

$$e = \lim_{n \rightarrow \infty} \left(1 + \frac{1}{n}\right)^n = 2.7182818 \dots$$

is the base of the system of natural logarithms. The exponential function $f(x) = e^x$ has a growth rate that at any point is directly proportional to the function's magnitude. To enhance readability the notational variant “exp(x)” will be employed in this work.

The real function $f(x)$ is said to be *Laplace transformable* provided that the integral

$$\int_0^{\infty} |f(x)| \exp(-kx) dx$$

converges for some real k . The *Laplace transform* of $f(x)$ is defined as

$$\mathcal{L}_r\{f(x)\} \equiv \int_0^{\infty} \exp(-rx) f(x) dx,$$

where r is a complex variable. To recover $f(x)$, the inverse Laplace transform (inversion integral) is used:

$$f(x) = \frac{1}{2\pi i} \int_{c-i\infty}^{c+i\infty} \exp(rx) \mathcal{L}_r\{f(x)\} dr,$$

where $i = \sqrt{-1}$. The convolution of the pdf's of two nonnegative random variables \mathbf{X} and \mathbf{Y} can be easily computed by

$$f_{\mathbf{Z}}(z) = \mathcal{L}_{\mathbf{Z}}^{-1}\{\mathcal{L}_r\{f_{\mathbf{X}}(x)\}\mathcal{L}_r\{f_{\mathbf{Y}}(y)\}\}, \quad z \geq 0$$

where $\mathcal{L}_{\mathbf{Z}}^{-1}$ is the inverse Laplace transform operator [COOK81].

Gamma Function

Several probability laws that will be discussed in this paper make use of the widely tabulated “complete gamma function” defined by

$$\Gamma(n) = \int_0^{\infty} x^{n-1} \exp(-x) dx.$$

It has these properties:

- 1.) $\Gamma(0) = 1$,
- 2.) $\Gamma(1/2) = \sqrt{\pi}$, and
- 3.) $\Gamma(n) = (n-1)\Gamma(n-1)$.

If n is an integer, $\Gamma(n) = (n-1)!$.

Reliability Concepts

In reliability studies, the fundamental variable concerning an individual is a binary variable representing either survival or failure. (“Individual” in a technological context generally refers a man-made system or component, but reliability theory is also often applied in studies of the mortality of living organisms, where it masquerades as part of actuarial science or biometrics.) Generally one is interested in the proportion of survival within groups of individuals under given conditions. Proportions are productively represented in terms of probabilities, for then the highly developed concepts and methodologies of probability theory can be mustered.

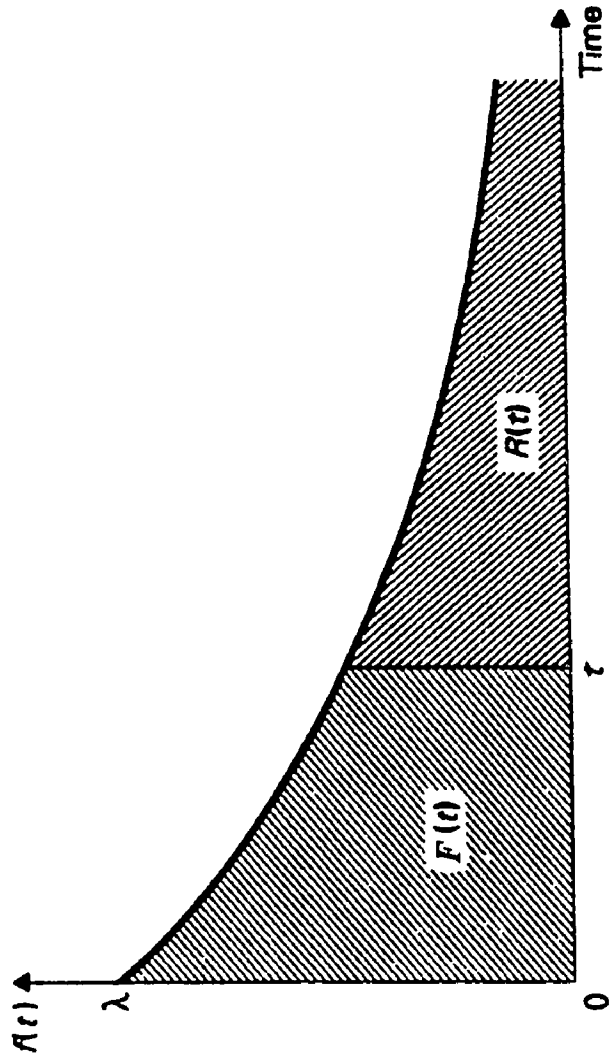


FIGURE 3
RELIABILITY FUNCTION $R(t)$ AND
CUMULATIVE DISTRIBUTION $F(t)$ FUNCTION

Since the useful operating time of a system cannot be predicted exactly, one can only state what the probability is that the system will not fail prior to a specified time t . Let the continuous random variable \mathbf{T} represent the time-to-failure of a system. $\text{Cdf}\{\mathbf{T}\}$ gives the probability of failure by time t , i.e.,

$$\text{Cdf}\{\mathbf{T}\} = \Pr\{\mathbf{T} \leq t\} = \int_0^t f(u) du.$$

Often it will be more natural or convenient to work with the function complementary to $\text{Cdf}\{\mathbf{T}\}$: The *survivor* or *reliability function*[COX62]

$$\text{Sf}\{\mathbf{T}\} \equiv R(t) \equiv \Pr\{\mathbf{T} > t\} = 1 - \text{Cdf}\{\mathbf{T}\}$$

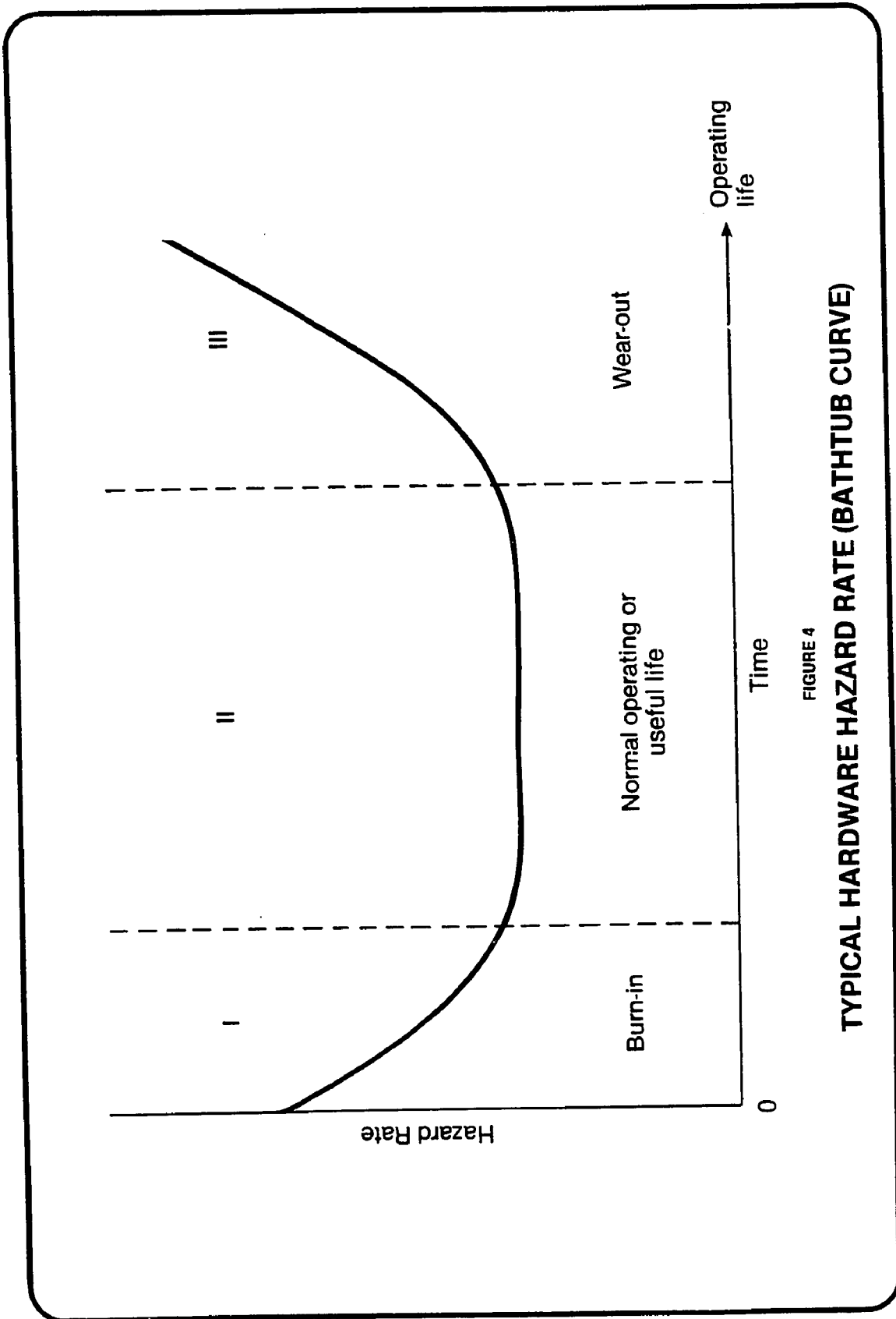
gives the probability that the system will operate without failure for at least the time interval $(0, t]$. Figure 3 shows the areas representing the cumulative distribution function $F(t)$ and the reliability function $R(t)$ for a typical time-to-failure pdf. The system is assumed to be operable at time $t = 0$; thus $R(0) = 1$. Eventually the system is presumed to fail: $R(\infty) = 0$. Between these two extremes, $R(t)$ is a monotone nonincreasing function of t . For negative values of t , $R(t) = 1$ for convenience, but negative time is meaningless. The first and most basic task of reliability theory is to derive and investigate the reliability function.

The *hazard rate function* is the conditional rate of failures per unit time, given that the system has survived to at least time t . Mathematically, the hazard rate is expressed as the limit of the ratio of the conditional probability of failure during the next small increment of time, given that a failure has not occurred, and the length of that time increment, as the length approaches zero:

$$\text{hr}\{\mathbf{T}\} \equiv \lim_{\Delta t \rightarrow 0^+} \frac{\Pr\{t < \mathbf{T} < t + \Delta t | \mathbf{T} > t\}}{\Delta t}.$$

In reliability modeling, this function is usually called the (*conditional*) *failure rate* and is traditionally symbolized as $\lambda(t)$. Note that $\lambda(t)$ is not a pdf since

$$\lambda(t) = f(t | \mathbf{T} \geq t) = \int_0^\infty \lambda(x) dx = \infty.$$



Any positive function $\lambda(x)$ for which

$$\lim_{t \rightarrow \infty} \int_0^t \lambda(x) dx$$

is unbounded can be used as a failure rate [MURT72]. The *average failure rate* is defined as

$$\bar{\lambda} = \frac{\text{failures}}{\text{time}},$$

the simple ratio of failures over time. The average failure rate over an infinitesimally small unit of time is just the instantaneous failure rate $\lambda(t)$. It is possible to define the failure acceleration, etc. using higher-order derivatives. Figure 4 shows the typical hazard rate through time for a hardware component or system. The first phase is the infant mortality phenomenon; the second phase is the period when the main peril is random stresses; and the final phase is the wearout period.

Failure time distributions can be broken into intervals of decreasing failure rate (DFR), increasing failure rate (IFR), and constant failure rate (CFR). Note that $f(t)\Delta t$ is the *unconditional* probability that failure will occur in the time interval $(t, t + \Delta t]$, and $\lambda(t)\Delta t$ is the *conditional* probability that a system surviving to age t will fail in the interval $(t, t + \Delta t]$. The distinction can be illustrated by considering a newborn baby. The probability that it will die between ages 107 and 107.5 is very small, but the probability that it will die between those ages, given that it has lived to age 107, is appreciably higher.

The hazard rate, probability density function and cumulative distribution function of \mathbf{T} each uniquely determine the other two functions. Letting the *cumulative hazard function*

$$\text{HR}\{\mathbf{T}\} = \int_0^t \lambda(x) dx,$$

the following identities hold for $t \geq 0$: $\text{Sf}\{\mathbf{T}\} = \exp(-\text{HR}\{\mathbf{T}\})$, $\text{pdf}\{\mathbf{T}\} = \text{hr}\{\mathbf{T}\} \exp(-\text{HR}\{\mathbf{T}\})$, and $\text{hr}\{\mathbf{T}\} = \text{pdf}\{\mathbf{T}\}/\text{Sf}\{\mathbf{T}\}$. The probability density function, the cumulative distribution function and the failure rate are thus completely

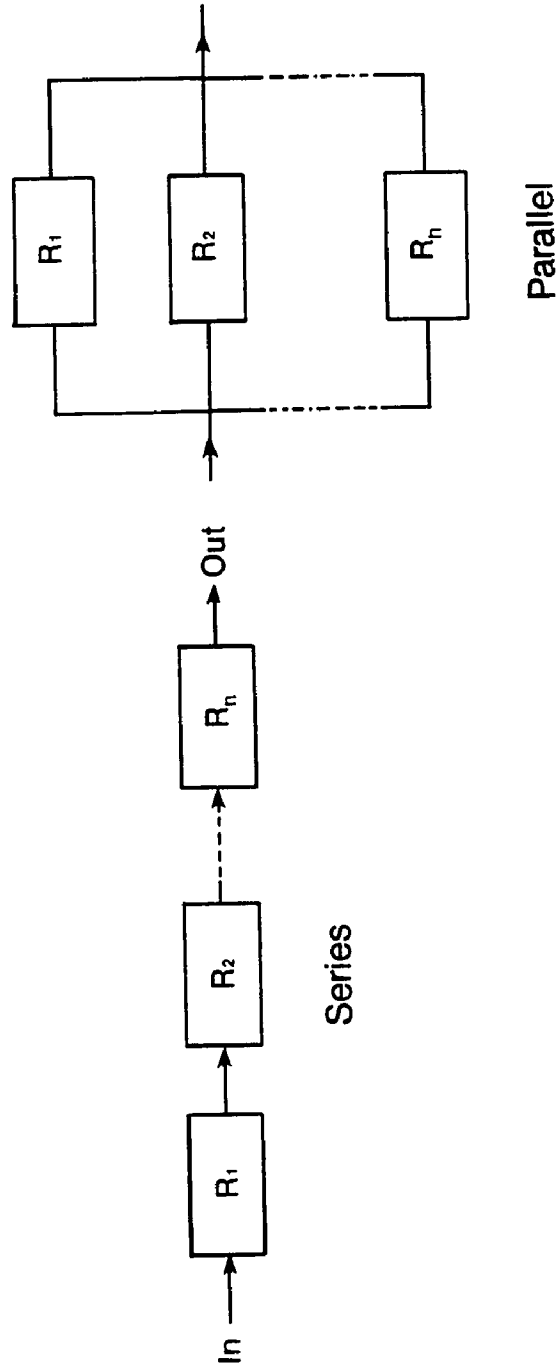


FIGURE 5
COMBINATORIAL RELIABILITY

equivalent mathematically. The choice of which to use when is a question of convenience.

The *mean time to failure* (MTTF) is the s -expected value of \mathbf{T} :

$$\text{MTTF} = E\{\mathbf{T}\} = \int_0^{\infty} t \text{pdf}\{\mathbf{T}\} dt.$$

If $\lim_{t \rightarrow \infty} tR(t) = 0$, then it can be shown through integration by parts that

$$\text{MTTF} = \int_0^{\infty} R(t) dt.$$

For nonredundant systems, the MTTF can be easily estimated by dividing the time during which the system is operational by the number of reported failures.

The reliability of a given component or system is usually unknown until failure data is collected in accordance with a *sampling scheme*.

Combinatorial Reliability

When k components are connected in series, the total reliability is given by

$$R(t) = R_1(t)R_2(t) \cdots R_k(t).$$

Had these components been connected in parallel, the total reliability would be given by

$$R(t) = 1 - [1 - R_1(t)][1 - R_2(t)] \cdots [1 - R_k(t)].$$

Figure 5 shows block diagrams of the basic parallel and series system configurations.

Software Reliability Concepts

Consider the case where the system under consideration is a running computer program. The program will purport to achieve some desired mapping from an input space to an output space. This desired mapping may fail to be realized on certain input cases because the program is in some sense an imperfect realization of that

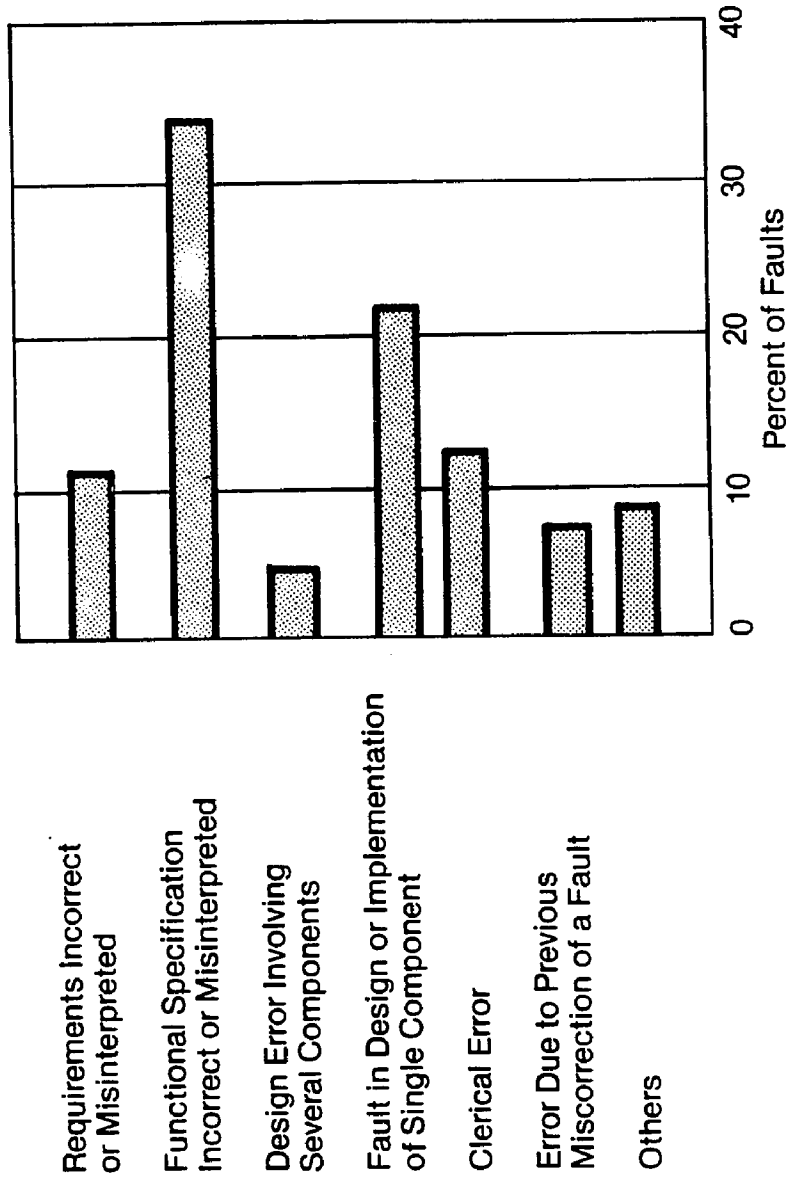


FIGURE 6

FAULT SOURCES

mapping. Suppose that a test oracle is available that will tell for any input case whether the program will fail to achieve its intended function. Let the oracle's answer be expressed by the binary function ϵ , whereby $\epsilon(i) = 0$ indicates success and $\epsilon(i) = 1$ indicates failure for input case i .

The input cases are presumed distributed in such a way that $p(i)$ is the probability of case i being presented as input to the program. Since $p(\cdot)$ is a pmf, it must be the case that $\sum_i p(i) = 1$.

The probability of a software failure on a single run is then

$$\lambda_n = \sum_i \epsilon(i)p(i).$$

If r is the number of input cases the program is presented with per unit time, then the failure rate will be given by $\lambda_n r$ [KOPE79]. The sample space for the experiment is the set of all possible operational scenarios.

A *fault* is an error or omission in the specification, design or coding of a computer program. Figure 6, based on a study described in [RAMA84], shows a breakdown of common fault sources. (The meanings of nomenclature like *error*, *fault*, *mistake*, *defect*, etc. differ widely in the literature. Some authors distinguish between the human action causing a fault and its manifestation.) A *software failure* is an unacceptable deviation from a program's expected behavior, attributable to a fault. Note that the "expected behavior" of the program is something in the user's head; the requirements specification that he agreed to may be wrong, or may have been misinterpreted by the development organization. A troublesome problem: There may be no reliable reference for the correct behavior. "More likely, we have the program result and another result that may be as error-prone as the program result ([e.g.] a hand calculation) [LEAT83]." On the other hand, "if the results of all processes have to be known beforehand, there would hardly be any purpose in writing a computer program. [WIRT73]." The reliable-reference problem has been circumvented in software reliability theory by assuming the existence of the test

oracle, but certainly has practical implications. Defining a failure as a deviation from the program specification can lead to the *reductio ad absurdum* quipped by Gilb:

“After all, [a failure] is only [a failure] in relation to the program documentation. This is a well-known trick for increasing reliability—no documentation—who can complain when they have no description of the system’s capabilities[GILB74]?”

A software failure can result from either a *conformance deviation* or a *performance deviation*. Conformance deviation results when the output itself is erroneous. Performance deviation results when the program produces the correct answer or response but expends too many resources (typically time) in doing so.

The behavior will be manifest in the program’s output, in the form of visual displays, printed hardcopies, control of and commands to peripheral devices, and so on. During testing, additional output such as dumps and traces might be forced, in order to provide more insight into the program’s operation. The output is examined to determine if it lies within some acceptable tolerance of the requirements. The determination that a failure has occurred implies the detection of a fault. A diagnostic procedure then ensues to find the fault. While in a nontechnical sense the word *fault* has the connotation of a minor, isolated flaw, in actuality anywhere from one to perhaps thousands of lines of code might be implicated. The earlier in the life cycle that the error originated, the more lines of code that generally will be affected.

Failures in a computer system traditionally fall into the categories of software failures, hardware failures, and “liveware” failures (operator error). It is easy to come up with situations, for instance firmware (program or microprogram code burned into ROM [read-only memory]), where the distinctions could become hazy. More on the relation between software and hardware failure modes appears later in this chapter.

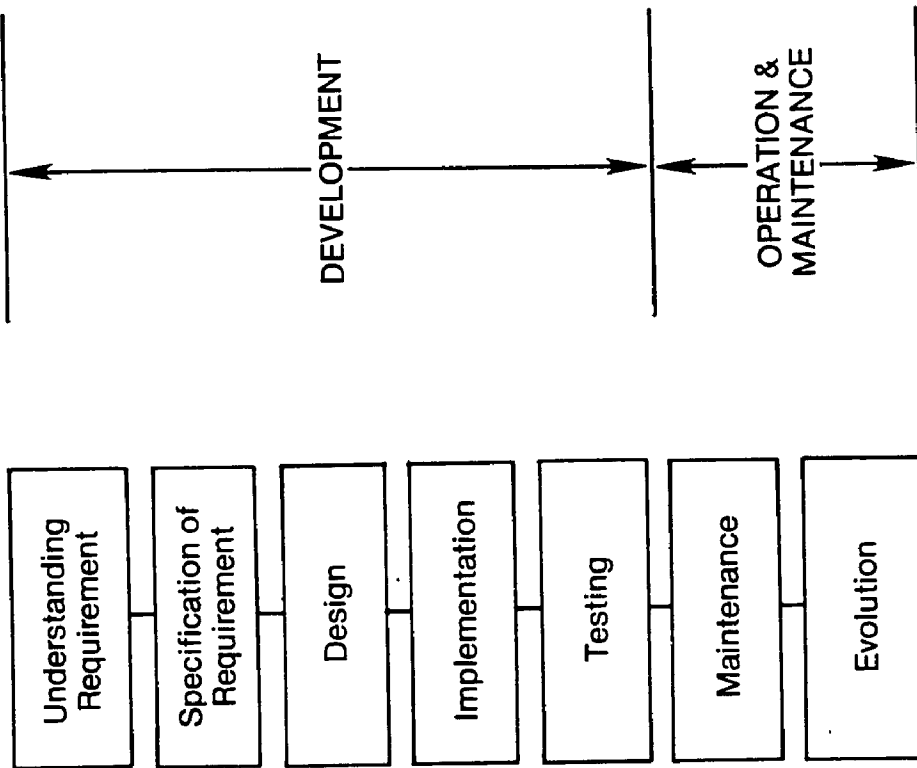


FIGURE 7

CLASSIC SOFTWARE LIFE CYCLE MODEL

The *penalty cost* of an individual software failure is a quantification of the undesired consequences of that failure. The *aggregate penalty cost* over a time interval is the sum of the penalty costs arising from every software failure that occurred during that interval.

For software, MTTF is a controversial measure, since if there is a nonzero probability of no faults in the software, the MTTF will be infinite[LITT75B]. (Most hardware, at least, will eventually wear out.) MTTF is “the cornerstone of current reliability theory; it is specified, predicted and demonstrated[GOLD81],” but its theoretical difficulty has prompted a move toward the use of positional measures such as percentiles. The value τ_i such that $\Pr\{\mathbf{T} \leq \tau_i\} = .01i$ is called the *i*th percentile of the time-to-failure distribution. The value $\tilde{t} \equiv \tau_{50}$ is the *median time to failure*.

Figure 7 (after [RAMA84]) shows the classic software life cycle model, which is generally assumed for the software reliability models discussed here. The *requirements phase* begins with an analysis of what the system is to accomplish and culminates in a specification document that expresses the desired behavioral attributes of the program from a nonprocedural viewpoint. It is in this phase that performance and reliability goals are generally set. The specification will serve as somewhat of a contract between the procuring and developing organizations. In the *design phase* the solution is devised and organized. The architectural framework of the software is established by allocating the functionality among modules whose interfaces are well defined. In the final stages of the design the procedural detail is fleshed out using graphical, tabular and textual tools. In the *coding phase* the algorithms and data structures are implemented in a programming language appropriate for the target machine.

In the *test phase* each module is first tested as a unit, and then the modules are synthesized into subsystems, which are further integrated into the whole hardware-software system. It is in this integration testing stage that failures can be recorded

for fitting to time-domain software reliability models. The final stage of testing is validation testing, during which the system's behavior is compared with the expectations of the requirements specification. The software may be sent out to a separate organization for independent verification and validation (IV&V).

At some point the software will be deemed acceptable by the customer and go into operational use. During this *maintenance phase* software failures may still occur, some of which may result in downtime. The failure times here in this phase are termed *retrospective failure data*. Often the software will need to be adapted to new environments (e.g., operating system releases, updated hardware). The end-user may request enhancements once he has had some experience with the software. Some of this maintenance activity may result in recursions of the full life cycle.

Phase completions (milestones) are marked by formal incremental reviews and the delivery of formalized products (baselines).

It is important to distinguish between two kinds of time considered in software reliability modeling. Debugging time (τ) occurs prior to the release of a version of a program. During this testing phase, the program is run until a software failure occurs. Then the program is modified in an attempt to repair the fault causing the failure. In some models the repair is assumed to take place immediately. In other models, fault isolation and correction activities occur concurrently with further test runs. In models with *perfect debugging*, faults are isolated and corrected instantaneously. In models with *imperfect debugging*, repair activity may fail to repair the fault and may even generate new faults.

Eventually, testing ceases and the latest program version is released. Cumulative operating time in the operational environment is denoted t and is what is used in expressing reliability.

The *availability* $A(t)$ is the probability that the system is operational at time t . The *mean time between failures* (MTBF) is equal to

$$\text{MTBF} = \text{MTTF} + \text{MTTR},$$

where MTTR is *mean time to repair*. The *steady state availability* is defined as

$$A = \lim_{t \rightarrow \infty} A(t),$$

which can be approximated by

$$\hat{A} = \frac{\text{uptime}}{\text{uptime} + \text{downtime}} = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}.$$

The difference between $R(t)$ and $A(t)$ can be clarified this way: $R(t)$ is the probability that no failures occur prior to time t ; $A(t)$ is the probability that all failures have been repaired by time t .

Software reliability models can be split into two main categories: *classical* and *Bayesian*. Classical models assign probabilities according to a relative frequency interpretation. Under this interpretation an experiment is performed (or imagined to be performed) repeatedly under identical conditions. If an event E is observed to occur m times in n trials, then the probability of E is assigned $\Pr\{E\} = \lim_{n \rightarrow \infty} m/n$.

The classical view of reliability [BILL83] considers a large fixed number N_0 of similar components. They are operated simultaneously starting from time $t = 0$. Let $N_f(t)$ be the number of components that have failed by time t . Then $N_s(t) = N_0 - N_f(t)$ are still operating. At any time t , the reliability function will be given by

$$R(t) = \frac{N_s(t)}{N_0} = \frac{N_0 - N_f(t)}{N_0} = 1 - \frac{N_f(t)}{N_0}.$$

Thus the *unreliability* (Cdf) is

$$F(t) = \frac{N_f(t)}{N_0}.$$

Differentiating $F(t)$ gives the pdf

$$f(t) = \frac{1}{N_0} \frac{dN_f(t)}{dt}.$$

To get the conditional failure rate $\lambda(t)$ it is necessary to substitute $N_s(t)$ for N_0 , since the number of components exposed to failure drops over time.

Puzzlement over how software reliability fits in with these frequency-based statistics (each piece of software is unique) has led some researchers to prefer a Bayesian approach. In Bayesian statistics, initial probabilities are allowed to have a purely subjective basis until they are iteratively updated in light of actual empirical data. Probability is viewed as a mathematical expression of a person's degree of belief concerning a certain proposition. The probability scale from 0 to 1 is merely a means of calibrating subjective attitudes and has no relation to an imagined infinite series of repeated experiments.

Prior distributions are used to represent what is known about the parameters before experimental evidence (sample data) is available. Difficulty can arise, though, in finding the proper prior distribution that conforms to vague prior knowledge about a parameter.

Bayesian statistics takes its name from the English Presbyterian minister and mathematician Thomas R. Bayes (1702–1761). Rev. Bayes developed a theorem called *Bayes' Rule*: Suppose that the events A_1, A_2, \dots, A_n form a partition of a sample space. Let B be some other event. For any j , a joint probability can be expressed as

$$\mathcal{P}(A_j \cap B) = \mathcal{P}(A_j) \times \mathcal{P}(B|A_j)$$

or by reversing the two events, as

$$\mathcal{P}(B \cap A_j) = \mathcal{P}(B) \times \mathcal{P}(A_j|B)$$

where $\mathcal{P}(B)$ is nonzero. Because $\mathcal{P}(A_j \cap B) = \mathcal{P}(B \cap A_j)$, it must now be the case that

$$\mathcal{P}(A_j) \times \mathcal{P}(B|A_j) = \mathcal{P}(B) \times \mathcal{P}(A_j|B).$$

Solving for $\mathcal{P}(A_j|B)$ gives

$$\mathcal{P}(A_j|B) = \frac{\mathcal{P}(A_j) \times \mathcal{P}(B|A_j)}{\mathcal{P}(B)}.$$

Since the A_j are mutually exclusive,

$$\mathcal{P}(B) = \sum_{i=1}^n \mathcal{P}(A_i) \times \mathcal{P}(B|A_i)$$

and thus

$$\mathcal{P}(A_j|B) = \frac{\mathcal{P}(A_j) \times \mathcal{P}(B|A_j)}{\sum_{i=1}^n \mathcal{P}(A_i) \times \mathcal{P}(B|A_i)}.$$

This formula is what is called Bayes' Rule. For a continuous distribution the denominator is replaced by the corresponding integral. Use of a special distributional form called a *conjugate prior* results in greater tractability.

This rather simple equality, based on sound mathematical reasoning, took on a new importance some two hundred years after its discovery. Some statisticians claimed that the prior probability $\mathcal{P}(A_j)$ could be based on subjective judgment and historical data.

For example, the software can be assumed to exhibit a certain reliability function or failure rate function. As the program is tested it leaves behind a failure history. Certain events might cause the reliability or failure rate to be considered improved:

- 1.) Correction of faults that have caused failures (especially the first-found high-failure-rate faults)
- 2.) "Long" interfailure times

Reliability will be considered worsened when

- 1.) New faults are introduced during repair activity
- 2.) “Short” interfailure times occur

The Review of Related Research in the next chapter will cover the major software reliability models, whether arising from classical or Bayesian foundations.

Software Reliability and Hardware

Microelectronic digital hardware is starting to take on more and more the characteristics of software vis-à-vis reliability. There are several reasons for this:

First, a correctly manufactured, *hermetically sealed* semiconductor device is not subject to wearout[OCON81]. The presence of the wearout failure mode has been one of the major reasons for the schism between hardware and software reliability principles.

Software and these semiconductor devices experience failures from the same three failure modes[SHOO83]:

- 1.) Poor-quality fabrication
- 2.) Design error
- 3.) Overload

Examples of “poor-quality fabrication” for software would be typographical errors, configuration management problems, and incompatibilities with operating system or hardware environment. “Design error” would include wrong algorithm or incorrect data structure. “Overload” will occur when real-time or multi-user systems have demands place upon them that exceed their designed capacities.

Second, as the circuitry in microelectronic devices becomes more and more complex, it has become impractical or impossible to perform 100% testing in a reasonable period of time. Just like software.

Third, often an integral part of the device is a permanently stored program or microprogram ("firmware"). Any nontrivial program there is almost certain to contain bugs. The techniques of software reliability modeling would apply directly to that code.

The import of all this is that the results of software reliability research, including the type of modeling techniques developed in this work, will find applicability in areas of hardware reliability as well.

CHAPTER 3

REVIEW OF RELATED RESEARCH

In order to sketch the historical development of software development and portray the state of the art, the major literature in the area will be surveyed. The survey will provide the context and motivation for the research presented in this work.

Mathematical reliability models originated in World War II Germany after frustration with the first ten of Wernher Von Braun's V-1 missiles, all of which either blew up on the launchpad or plopped into the English Channel[HENL81]. In recent years a number of promising software reliability models have been proposed. Each mathematically summarizes a set of assumptions the author has made about the phenomenon of software failure. The goal is to devise an accurate predictive model. Most of the models described here have been empirically verified to one degree or another, but none has emerged as clearly superior or ready for commercial use.

Software reliability models are an outgrowth of hardware reliability modeling. From a user's point of view, whether a system failure was caused by software or by hardware is unimportant: The disruption is still the same. This is why the U.S. Department of Defense has placed software under the "reliability umbrella."

The extensive body of literature and techniques developed for hardware reliability analysis, and the acknowledged success of that field, have seduced many researchers into attempting to apply hardware modeling methods to software. Controversy persists. Certainly there are differences between the nature of software and hardware. Neither hardware nor software failure times can be exactly predicted in

point of time, but for different reasons. For hardware, failure is caused by accumulated random stresses and wearout. Software is deterministic in that certain input will *always* result in a failure. Simple redundancy of the same code will do nothing to increase the reliability of software. On the other hand, with proper quality control feedback, a fault will not recur once discovered. The development process is not deterministic: Human fallibility leaves the program littered with faults, the whereabouts of which are initially unknown. Thus for software the faults are always there, and the passage of time serves to *reveal* them through testing and use. For hardware the fault is often *generated* over time, when stresses or wearout have accumulated to the breaking point[SCHN79]. Of course, software and hardware alike can suffer from inherent design failures, transient overload conditions and poor-quality fabrication.

As pointed out in the previous chapter, complex digital hardware, such as microcoded processors, exhibit failure behavior closer to that of software and are amenable to software reliability techniques.

Software reliability models generally employ named theoretical probability laws because such distributions are completely summarized by a small number of parameters (usually 1-2); there is no need to operate with long tables of observed frequencies. Also, theoretical distributions exhibit convenient, well-known properties that facilitate mathematical manipulation and analysis and allow statistical inferences to be made.

Hudson's[HUDS67] view of fault correction as a Markov "death" process was the first significant paper on the subject of software reliability modeling.

Time-Domain Models

Early researchers made the simple observation that as faults were removed from a program, the failure rate went down. They then went about postulating a relationship between the number of faults remaining in a program and the program's

failure rate. They tried to stay as close as possible to the familiar probability distributions of hardware reliability analysis.

In order to compare existing software reliability models, it is helpful to employ a variant of Govil's unifying notation [GOVI84]. Let τ denote the cumulative debugging time. Let t denote the operational time interval from the present, projected on the basis of no further fault correction. As functions of cumulative debugging time τ and operational time interval t , the failure rate will be denoted $\lambda(\tau, t)$, and the reliability will be denoted $R(\tau, t)$. The s -expected value of the total number of faults detected and corrected by debugging time τ will be given by the *renewal function* $N(\tau)$. The s -expected value of the initial number of errors present in the software will be denoted $N(\infty) = \lim_{\tau \rightarrow \infty} N(\tau)$. The function $g(t)$ denotes a function of operating time t that may be equal to 1 in some models.

Knowledge of the functions $N(\tau)$ and $g(t)$ will be sufficient to determine these figures of merit:

Failure rate—

$$\lambda(\tau, t) = \frac{dN(\tau)}{d\tau} \cdot g(t);$$

Reliability function—

$$R(\tau, t) = \exp \left[- \int_0^t \lambda(\tau, t) dt \right];$$

Mean time to failure—

$$\text{MTTF}(\tau) = \int_0^{\infty} R(\tau, t) dt.$$

Additionally, the special case of the mean time to *first* failure is defined as

$$T_0 = \text{MTTF}(\infty) = \int_0^{\infty} R(0, t) dt.$$

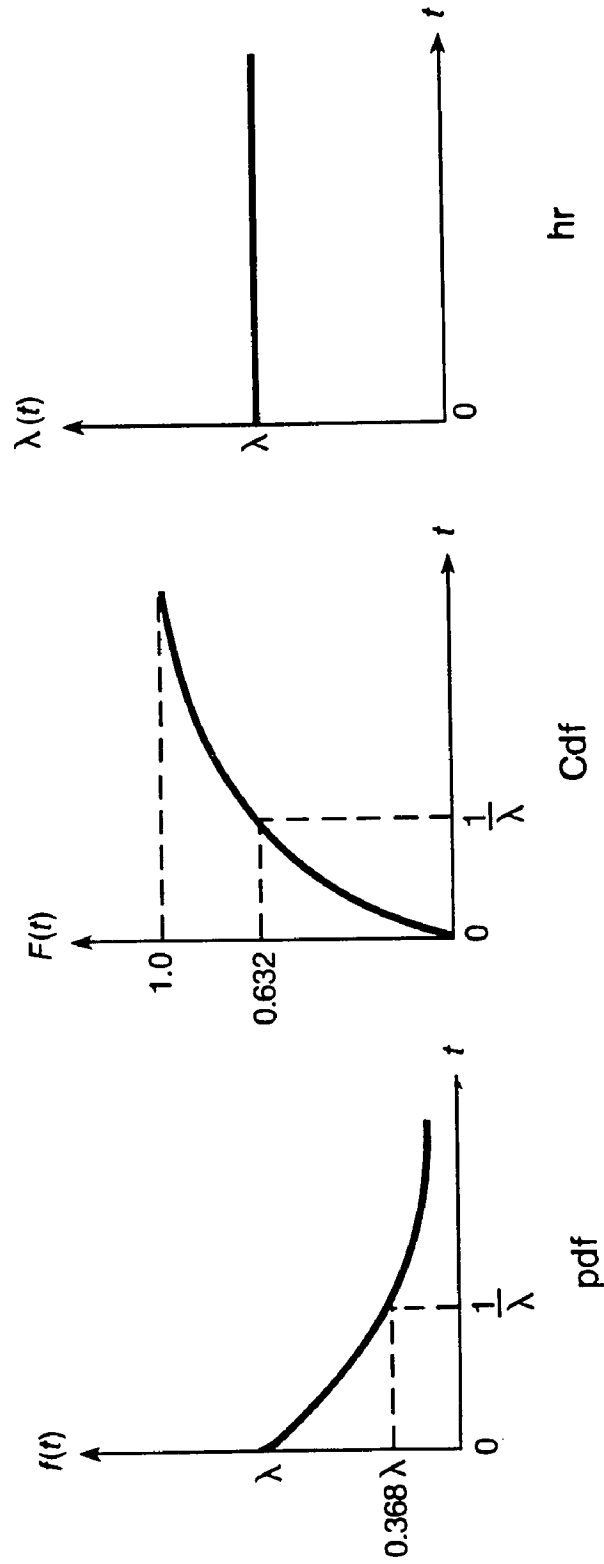


FIGURE 8
 NEGATIVE EXPONENTIAL DISTRIBUTION
 (JELINSKY-MORANDA, SHOOMAN, MUSA MODELS)

Jelinsky and Moranda[JELI73] developed a model based on the following assumptions:

1. The failure rate at any instant is linearly proportional to the number of faults in the program.
2. The interfailure times are mutually independent random variables.
3. All faults in the program are equally likely to cause failure.
4. The fault causing a failure is instantaneously corrected without introducing any new faults.

The resulting model is called the Jelinsky-Moranda De-Eutrophication Model, drawing an analogy to remedial pond management. Assumption #1 implies that a particular version of the program (frozen code) will exhibit a constant failure rate. A negative exponential distribution of interfailure times is then dictated by the fact that it is the only distribution with a constant failure rate[BARL75]. Figure 8 illustrates the characteristic shape of the negative exponential probability law. Since the failure rate is not a function of operational time, $g(t) = 1$, and thus the piecewise constant failure rate is formulated as

$$\lambda(\tau, t) = \frac{dN(\tau)}{d\tau} = K[N(\infty) - N(\tau)],$$

where K is a proportionality constant.

Integration gives

$$N(\tau) = N(\infty)[1 - \exp(-K\tau)],$$

and so the failure rate is

$$\lambda(\tau, t) = KN(\infty) \exp(-K\tau).$$

The reliability function is obtained from a further integration as

$$R(\tau, t) = \exp[-tKN(\infty) \exp(-K\tau)]$$

and the mean time to failure is

$$\text{MTTF}(\tau) = T_0 \exp(K\tau)$$

where

$$T_0 = \text{MTTF}(0) = \frac{1}{KN(\infty)}.$$

The probability density function is

$$f(\tau, t) = -\frac{\partial R(\tau, t)}{\partial t} = KN(\infty) \exp(-K\tau) \exp[-tKN(\infty) \exp(-K\tau)].$$

The number of errors remaining in the software after debugging time τ is

$$N(\infty) = \exp(-K\tau).$$

In terms of failures, the MTTF between the i th and $(i + 1)$ st software failures can be expressed as

$$\text{MTTF}_i = \frac{1}{K[N(\infty) - i]}.$$

(The variable $i = 1, 2, \dots$ is called the *failure sequence number*.)

Thayer, Lipow and Nelson [THAY78] give an extension of the Jelinsky-Moranda Model in which multiple faults can be detected in a time interval. Correction of the faults is considered to have been accomplished, not necessarily immediately, but at some time during that same time interval.

Shooman's model [SHOO73, SHOO75] came out about the same time as the Jelinsky-Moranda model. He assumes that the number of machine-level instructions in the program is a constant I . He then works with the "normalized" initial fault content

$$\varepsilon_0 = \frac{N(\infty)}{I}$$

and the normalized number of faults corrected by debugging time τ :

$$\varepsilon_c(\tau) = \frac{N(\tau)}{I}.$$

The normalized number of residual faults is then

$$\varepsilon_r(\tau) = \varepsilon_0 - \varepsilon_c(\tau) = \frac{N(\infty) - N(\tau)}{I}.$$

With again $g(t) = 1$, the failure rate is formulated

$$z(\tau, t) = \frac{dN(\tau)}{d\tau} = K \left[\frac{N(\infty) - N(\tau)}{I} \right]$$

where K is in this case a “bulk” proportionality constant that Shooman says can be estimated from

$$K \simeq \frac{\# \text{ of catastrophic faults detected}}{\text{total } \# \text{ of faults detected}}.$$

After integration,

$$N(\tau) = N(\infty)[1 - \exp(-K\tau/I)]$$

and so

$$\lambda(\tau, t) = \frac{K}{I} N(\infty) \exp(-K\tau).$$

The reliability function is

$$R(\tau, t) = \exp \left[-\frac{K}{I} N(\infty) \exp(-K\tau)t \right]$$

and the mean time to failure is

$$\text{MTTF}(\tau) = \frac{I \exp(K\tau)}{KN(\infty)}$$

where

$$T_0 = \frac{I}{KN(\infty)}.$$

The probability density function is

$$f(\tau, t) = \frac{KN(\infty)}{I} \exp(-K\tau) \exp \left[-\frac{K}{I} N(\infty) \exp(-K\tau)t \right].$$

Another difference between the Jelinsky-Moranda and Shooman models, which comes into play during parameter estimation, is that in the Jelinsky-Moranda model, debugging time is measured in calendar months, whereas Shooman defines debugging time in terms of effort (person-months).

Musa[MUSA75] introduced the idea of using execution (CPU) time as a more appropriate unit of exposure to “failure stress” than calendar time. He provided an elaborate companion model that relates execution to calendar time in light of resource constraints such as failure-identification personnel, failure-correction personnel, and the residence time of the program in the computer. Musa has collected a great deal of data[MUSA79A] in order to investigate the descriptive realism of his model and evaluate its predictive ability. The basic model is very similar to that of Jelinsky-Moranda and Shooman. Once again, $g(t) = 1$. The failure rate is

$$\lambda(\tau, t) = \frac{N(\tau)}{d\tau} = Kf[N(\infty) - N(\tau)],$$

where K is a proportionality constant (the “fault exposure ratio”) and f is the “linear execution frequency.” The fault exposure ratio relates “fault velocity” and failure rate. Fault velocity is the rate at which faults would be encountered by the CPU if the program were linearly executed. The fault exposure ratio thus “represents the fraction of time that the ‘passage’ results in a failure[MUSA84],” accounting for loops and branching and varying machine states. Typically, K is $1.5-3 \times 10^{-7}$. The linear execution frequency is the average instruction execution rate divided by the number of machine-level instructions in the program. Deriving the formulas,

$$N(\tau) = N(\infty)[1 - \exp(-fK\tau)]$$

$$\lambda(\tau, t) = KfN(\infty) \exp(-fK\tau)$$

$$R(\tau, t) = \exp[-KfN(\infty) \exp(-fK\tau)t]$$

$$\text{MTTF}(\tau) = T_0 \exp(fK\tau)$$

$$T_0 = \frac{1}{KfN(\infty)}$$

$$f(\tau, t) = KfN(\infty) \exp(fK\tau) \exp[-KfN(\infty) \exp(-fK\tau t)].$$

The constant K is decomposed into the product CB . C is the “testing compression factor” that is supposed to account for the difference between the testing and operational milieux. (Jelinsky-Moranda and Shooman had implicitly assumed that the program was tested using a simulator that emulates the program’s operating environment.) The failure rate will be affected by the workload of the system, and the testing environment is usually much more “stressful.” Musa has empirically estimated the testing compression factor to average around 10.9. B is the “fault reduction factor,” a proportionality constant that relates the fault correction rate to the failure rate. Typically B is $3-8 \times 10^{-3}$.

The number of faults detected and corrected by time τ can be re-expressed as

$$N(\tau) = N(\infty) \left[1 - \exp\left(-\frac{C\tau}{M_0T_0}\right) \right],$$

where M_0 is the total number of faults occurring during the program’s maintained life (the period in which faults are corrected). The relationships

$$N(\infty) = BM_0$$

and

$$N(\tau) = BM(\tau)$$

hold, where $M(\tau)$ is the number of failures experienced, given by

$$M(\tau) = M_0 \left[1 - \exp\left(-\frac{C\tau}{M_0T_0}\right) \right].$$

The additional number of failures that must be experienced to meet an MTTF objective T_F is

$$\Delta M(\tau) = M_0T_0 \left[\frac{1}{\text{MTTF}(\tau)} - \frac{1}{T_F} \right]$$

and the additional execution time required is

$$\Delta\tau = \frac{M_0 T_0}{C} \ln \left(\frac{T_F}{\text{MTTF}(\tau)} \right)$$

A compact expression for the reliability function is

$$R(\tau, t) = \exp \left(-\frac{t}{\text{MTTF}(\tau)} \right).$$

Musa claims that

“Most of the assumptions that were made in deriving the model have been validated. . . . There is no evidence to indicate any class of software to which the model would not apply. This model has been tested and its validity examined much more thoroughly than any other proposed software reliability model[MUSA84].”

Moranda has proposed a variation[MORA75] of the original Jelinsky-Moranda model that makes the additional assumption that the failure rates between successive failures form a geometric progression. With $g(t) = 1$, the failure rate is

$$\lambda(\tau, t) = \frac{N(\tau)}{d\tau} = DK^{-N(\tau)}.$$

And then

$$N(\tau) = \frac{\ln(1 + D\tau \ln K)}{\ln K}$$

$$\lambda(\tau, t) = \frac{D}{1 + D\tau \ln K}$$

$$R(\tau, t) = \exp \left(-\frac{D\tau}{1 + D\tau \ln K} \right)$$

$$\text{MTTF}(\tau) = \tau \ln K + \frac{1}{D}$$

$$T_0 = \frac{1}{D}$$

$$f(\tau, t) = \left(\frac{D}{1 + D\tau \ln K} \right) \exp \left(-\frac{D}{1 + D\tau \ln K} t \right),$$

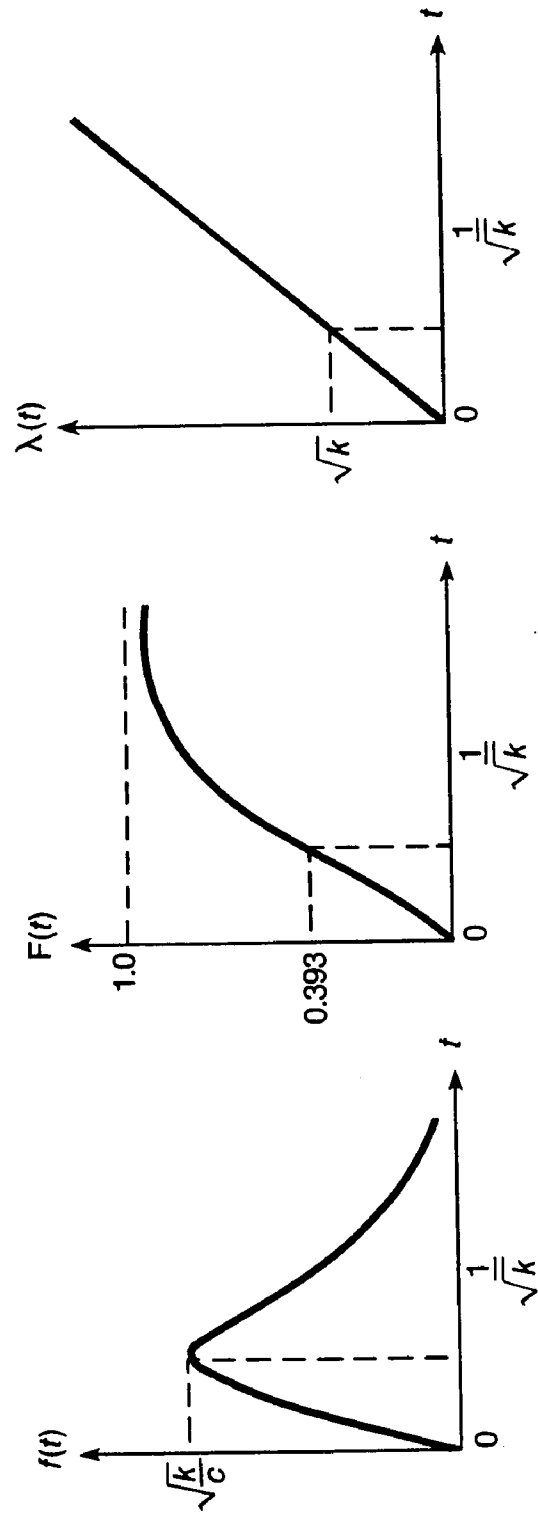


FIGURE 9
RAYLEIGH DISTRIBUTION (SCHICK-WOLVERTON MODEL)

where D is a constant. Unfortunately the number of remaining faults cannot be predicted as in the case of the ordinary Jelinsky-Moranda Model.

Lipow[LIP074] has suggested a modification to the Jelinsky-Moranda Geometric model to allow multiple faults in an operational time period.

Schneidewind[SCHN75] recommended an empirical approach that tries various curves in an effort to fit the project's data. It was he who underscored the importance of differentiating between operating time t and debugging time τ .

Schick and Wolverton[SCH178] presented a model based on a failure rate that discretely decreases at each failure but increases linearly during the debugging time between failures. For this model, $g(t) = t$. (Because of ambiguous notation, some early papers interpreted the model as in effect having $g(t) = \tau$ or $g(t) = \tau_{i-1}$)

The failure rate is

$$\lambda(\tau, t) = \frac{dN(\tau)}{d\tau} = K[N(\infty) - N(\tau)].$$

After integration,

$$N(\tau) = N(\infty)[1 - \exp(-K\tau)].$$

Then

$$\lambda(\tau, t) = KN(\infty) \exp(-K\tau)t$$

$$R(\tau, t) = \exp\left[-\frac{KN(\infty) \exp(-K\tau)t^2}{2}\right]$$

$$\text{MTTF}(\tau) = \left[\frac{\pi}{2KN(\infty)}\right]^{1/2} \exp(K\tau/2)$$

$$T_0 = [\pi/2KN(\infty)]^{1/2}.$$

The number of faults remaining after debugging time τ will be

$$N(\infty) \exp(-N\tau).$$

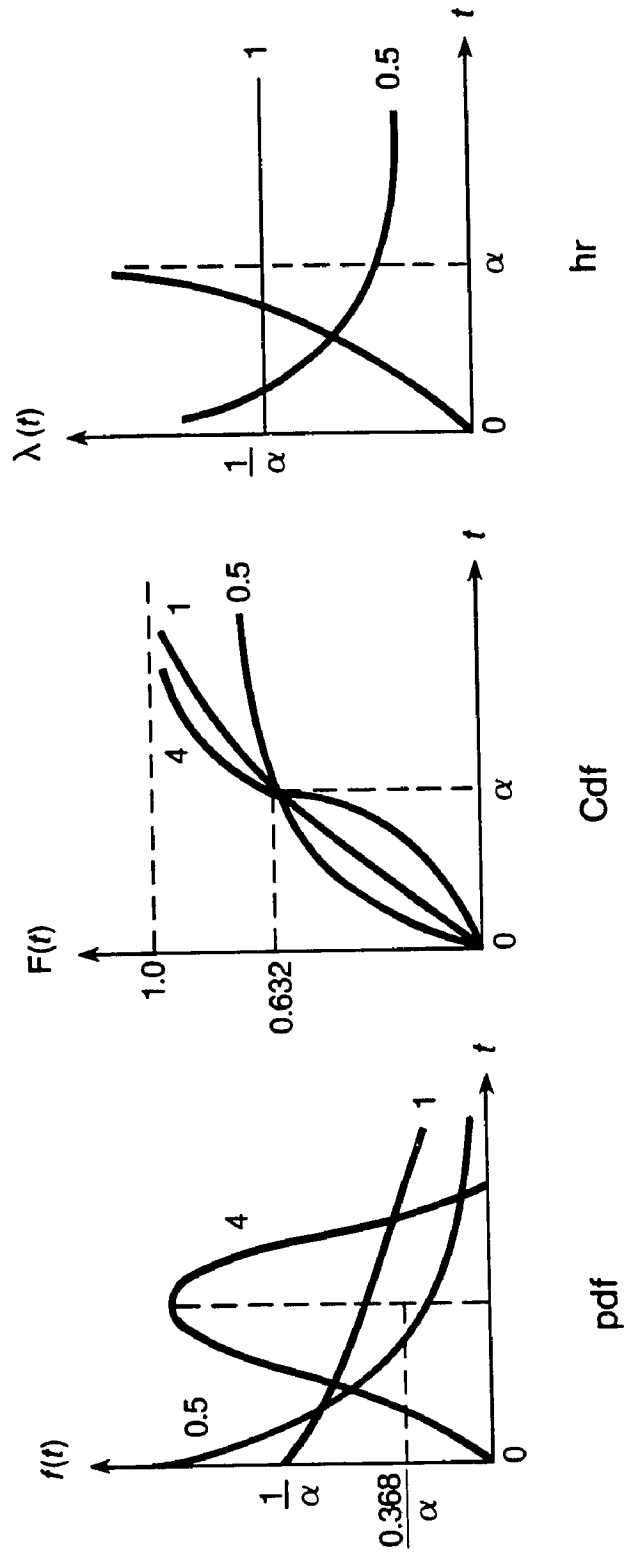


FIGURE 10
WEIBULL DISTRIBUTION (WAGONER MODEL)

The probability density function is

$$f(\tau, t) = KN(\infty) \exp(-K\tau)t \exp(-KN(\infty) \exp(-K\tau)t^2/2).$$

This results in a Rayleigh distribution for each interfailure interval, as opposed to the negative exponential distribution arising in the basic Jelinsky-Moranda, Shooman and Musa models. Figure 9 shows the characteristic shape of the Rayleigh distribution standard pdf

$$f(t) = kt \exp\left[-\frac{kt^2}{2}\right], \quad t \geq 0.$$

The early Wagoner model[WAG073] in effect generalizes on the Schick-Wolverton model by assuming that the failure rate is proportional to the product of the number of remaining faults times a positive power of the debugging time between successive failures. For this model $g(t) = t^{\beta-1}$. The failure rate is then

$$\lambda(\tau, t) = \frac{dN(\tau)}{d\tau} = K[N(\infty) - N(\tau)].$$

Thus

$$N(\tau) = N(\infty)[1 - \exp(-K\tau)]$$

$$\lambda(\tau, t) = KN(\infty) \exp(-K\tau)t^{\beta-1}$$

$$R(\tau, t) = \exp\left[-\frac{K}{\beta}N(\infty) \exp(-K\tau)t^\beta\right]$$

$$\text{MTTF}(\tau) = T_0 \exp(K\tau/\beta)$$

$$T_0 = \frac{\tau \left(1 + \frac{1}{\beta}\right)}{\left[\frac{K}{\beta}N(\infty)\right]^{1/\beta}}$$

$$f(\tau, t) = KN(\infty) \exp(-K\tau)t^{\beta-1} \exp\left[-\frac{K}{\beta}N(\infty) \exp(-K\tau)t^\beta\right].$$

The Wagoner model leads to a Weibull distribution for the interfailure times. The Weibull distribution has found many engineering applications since first described

[WEIB39] by its namesake in 1939. The Weibull distribution is one of the few common distributions that exhibit no specific characteristic shape, so it affords great adaptability in fitting experimental data. The special case $\beta = 2$ reduces to the Schick-Wolverton model, and the special case $\beta = 1$ reduces to the Jelinsky-Moranda model. Figure 10 shows the shapes of some typical Weibull distributions with standard pdf

$$f(t) = \frac{\beta t^{\beta-1}}{\alpha^\beta} \exp \left[- \left(\frac{t}{\alpha} \right)^\beta \right], \quad t \geq 0, \beta > 0, \alpha > 0.$$

In the Goel-Okumoto nonhomogeneous Poisson process (NHPP) model [GOEL79], the time to a given failure is assumed to be dependent on the preceding failure. Also, the inherent number of faults is regarded as a random variable rather than as a fixed parameter. This formulation allows for imperfect debugging, in that the fault causing a failure might not be immediately corrected. Each fault is assumed to have an equal probability of discovery.

If the failure rate is treated as a nonrandom function $\lambda(\tau)$ of time τ , the result is a (temporally) *nonhomogeneous Poisson process* (NHPP). Let $\mathbf{N}(\tau)$ be a random variable denoting the total number of faults detected by time τ . The mean value function is $N(\tau)$, as before.

The assumption is made that the s -expected number of software failures in the “small” interval $(\tau, \tau + \Delta\tau)$ is essentially proportional to the remaining number of faults at time τ , that is:

$$N(\tau + \Delta\tau) - N(\tau) = b[N(\infty) - N(\tau)]\Delta\tau + o(\Delta\tau)$$

where $o(\Delta\tau)/\Delta\tau \rightarrow 0$ as $\Delta\tau \rightarrow 0$ and b is a constant of proportionality. By allowing $\Delta\tau$ to approach zero, a differential equation

$$N'(\tau) = N(\infty)b - bN(\tau)$$

is obtained. Solving the equation under the boundary condition $N(0) = 0$ yields the mean value function

$$N(\tau) = N(\infty)(1 - \exp(-b\tau)).$$

The failure rate as a deterministic, continuously nonincreasing function of time is then

$$\lambda(t) \equiv N'(\tau) = N(\infty)b \exp(-b\tau).$$

The reliability given that the last failure occurred at time s is

$$R(t|s) = \exp(-[N(t+s) - N(t)]) = \exp(-a\{\exp(-bt) - \exp[-b(t+s)]\}).$$

The s -expected number of remaining faults at time τ is

$$E\{N(\infty) - N(\tau)\} = N(\infty) \exp(-b\tau).$$

$\mathbf{N}(\tau)$ is Poisson-distributed with mean $N(\tau)$, and $\mathbf{N}(\infty)$ is Poisson-distributed with mean $N(\infty)$. The number of remaining faults $\bar{\mathbf{N}}(\tau) = \mathbf{N}(\infty) - \mathbf{N}(\tau)$ has s -expected value

$$E\{\bar{\mathbf{N}}(\tau)\} = N(\infty) \exp(-b\tau).$$

A useful aspect of the Goel-Okumoto model is that parameter estimation can be performed on the basis of either of two forms of historical data:

- 1.) interfailure times, or
- 2.) number of failures within specified intervals.

In the failure-count version, the number of failures in nonoverlapping intervals are considered independent. That is, Goel and Okumoto make the assumption that $\mathbf{N}(t)$ has s -independent increments and is Poisson distributed with mean value function $m(t)$:

$$\Pr\{\mathbf{N}(t) = y\} = \frac{[m(t)]^y}{y!} \exp[-m(t)], \quad y = 0, 1, 2, \dots$$

In the time-between-failure version, as was covered, the time between the $(i - 1)$ st and i th failure is dependent on the time to the $(i - 1)$ st failure.

To contrast the earlier models with the Goel-Okumoto model: In the early models each interfailure period was governed by a distinct Poisson process, whereas here the whole debugging phase (“ τ ” space) is governed by a single nonhomogeneous Poisson process. At the expense of conformance to our intuitive desire for a piecewise constant failure rate, Goel and Okumoto achieve the analytical simplicity of a smooth failure rate curve. Now they can complicate the model someplace else: by considered the inherent number of faults in the program to be a random variable as opposed to a fixed but unknown parameter.

Bayesian Models

Littlewood questioned the theoretical validity of bug-counting models [LITT79A][LITT80] and substituted a Bayesian approach that views reliability as a measure of strength of belief that a program will operate successfully[LITT73].

In the Littlewood-Verral model[LITT73], the phenomenon of software failure is considered to arise from two distinct sources of randomness:

- (1) The program—the input data sets on which it will fail are unknown, and
- (2) The input—the particular input data sets the operational program will be presented with are unknown. A certain proportion of the input data sets will result in failure and the other data sets will not.

The randomness in the input data is modeled by assuming that the interfailure intervals \mathbf{T}_i have conditional distribution

$$\text{pdf}\{\mathbf{T}_i|\lambda_i\} = \lambda_i \exp(-\lambda_i T_i)$$

where \mathbf{T}_i denotes the time interval between the $(i - 1)$ st and i th software failures. The interfailure execution times are thus s -independent, negative exponentially distributed random variables. In this assumption, Littlewood and Verral are following

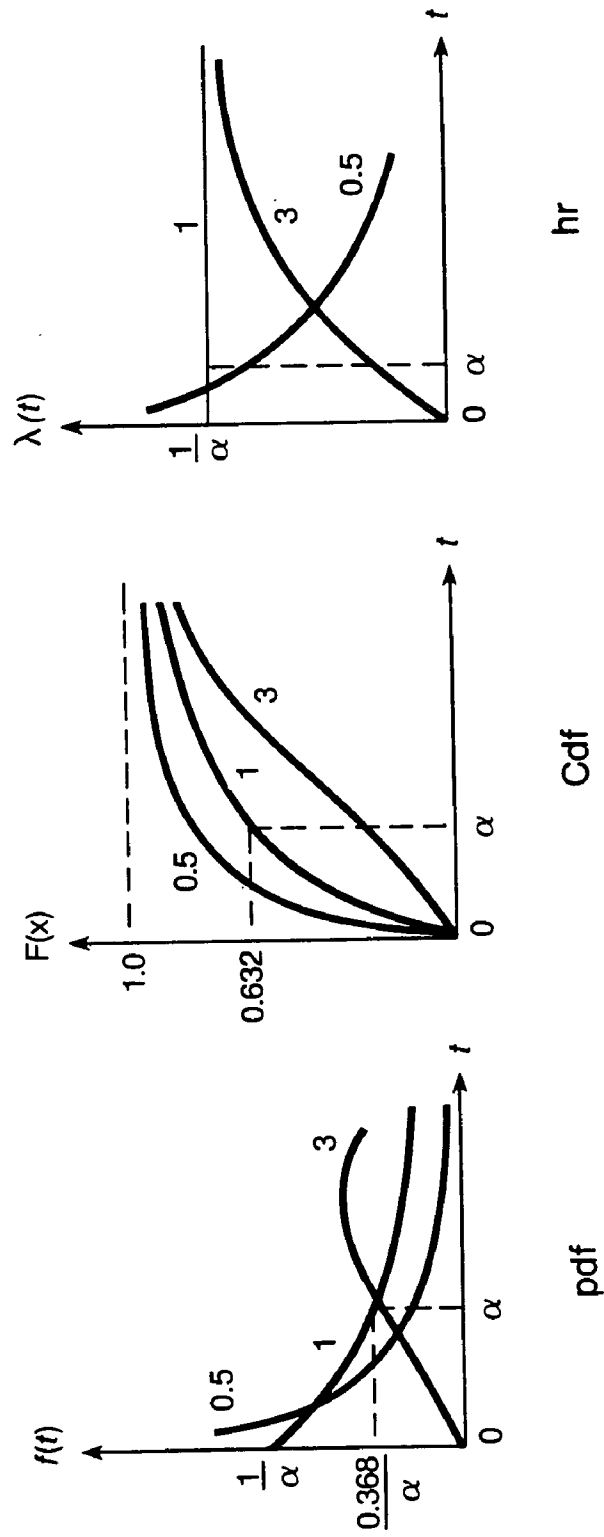


FIGURE 11
GAMMA DISTRIBUTION FOR VARIOUS VALUES OF β (LITTLEWOOD MODEL)

in the footsteps of Jelinsky-Moranda and Shooman. However, in those models the failure rate drops deterministically after each failure; in the Littlewood-Verral model, it is only required that $\lambda_i < \lambda_{i-1}$ in a *probabilistic* sense. As in the Goel-Okumoto model, then, debugging is considered to be imperfect.

Littlewood and Verral view the piecewise failure rates as a sequence $\{\lambda_i\}$ of s -independent random variables. For the sake of mathematical tractability, each is assumed to exhibit a gamma distribution:

$$\text{pdf}\{\lambda_i|\psi(i), \alpha\} = \frac{[\psi(i)]^\alpha \lambda_i^{\alpha-1} \exp[-\psi(i)\lambda_i]}{\Gamma(\alpha)}$$

where the $\psi(i)$, an increasing function of i , reflect reliability growth attributable to repair activities. Figure 11 illustrates the shapes of some typical gamma distributions with standard pdf

$$f(t) = \frac{t^{\beta-1}}{\alpha^\beta \Gamma(\beta)} \exp\left(-\frac{t}{\alpha}\right), \quad t \geq 0, \alpha > 0, \beta > 0.$$

The assumption was made in earlier models that a fault is immediately removed upon occurrence of the failure it caused. More realistically though, a lag occurs between the time a failure occurs and the time its underlying fault is discovered and removed. Not only that, but the repair activity itself may generate new faults. Evidence strongly suggests that most faults originate in the requirements and design phases[LIP079]. A software failure caused by a requirements misunderstanding or design deficiency may require extensive reworking of the program, and this reiteration of the original development process would likely produce new bugs. In fact, “the majority of software failures during operation can be traced to the unforeseen side effects of postrelease patches[Sch182].”

The initial number of faults, the speed with which failure-causing faults are repaired, and the degree of debugging “perfection” (lack of new-fault generation) are all in some sense indicative of the “quality” of the developing organization. The rate of $\psi(\cdot)$'s increase is directly related to this “quality” and inversely related to

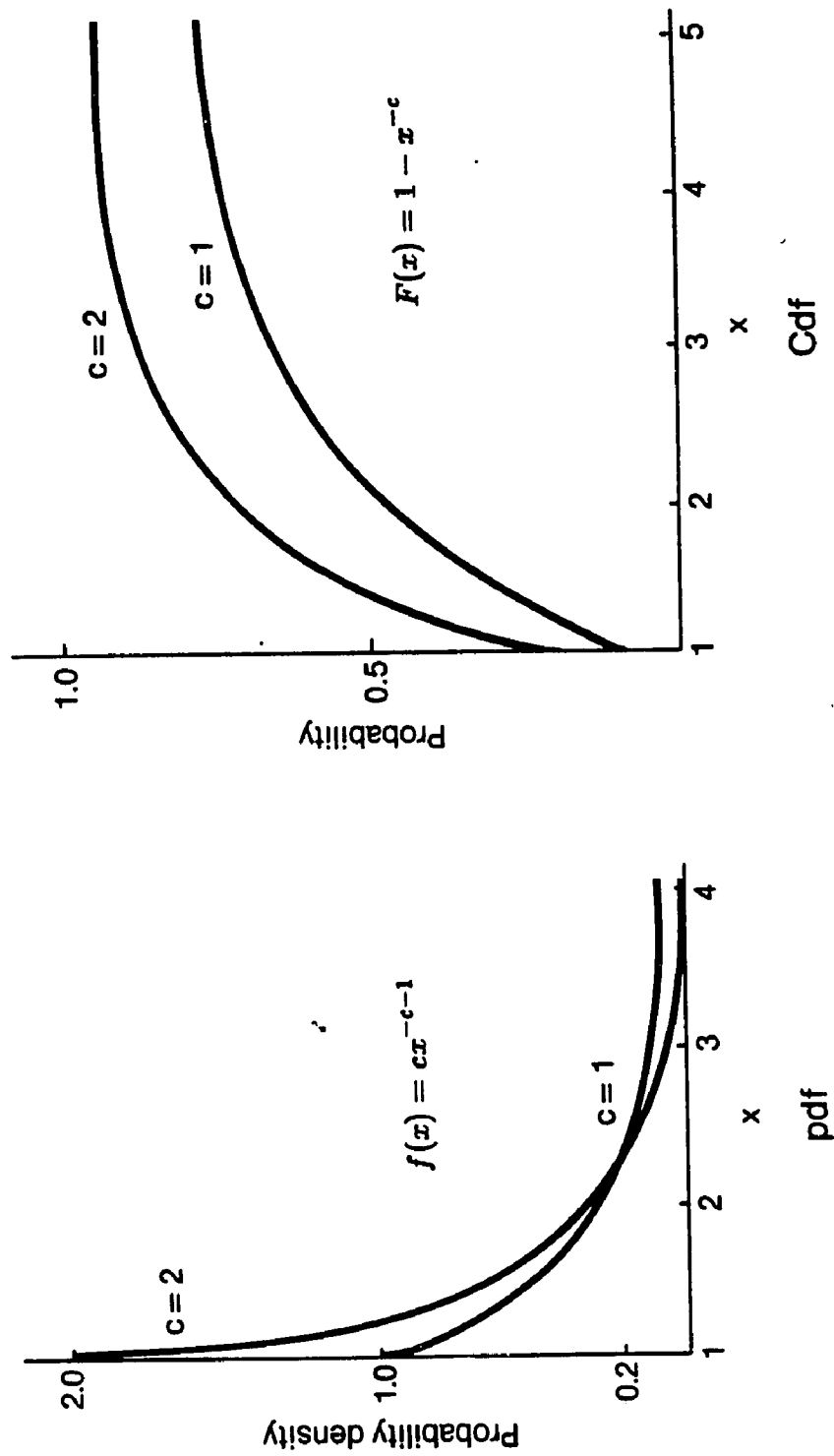


FIGURE 12
PARETO DISTRIBUTION (LITTLEWOOD-VERRAL MODEL)

programming task difficulty. For any particular project, Littlewood recommends that various families of growth functions be tried in an effort to find the best fit. The two sources of randomness are combined via Bayesian techniques.

Two parametric families of growth functions that Littlewood himself has suggested are the linear form

$$\psi(i) = \beta_1 + \beta_2 i$$

and

$$\psi(i) = \beta_1 + \beta_2 i^2.$$

Musa has suggested [MUSA84] the rational function

$$\psi(i) = \frac{M_0 T_0 \alpha}{M_0 - i}$$

where M_0 is the s -expected number of failures during the maintained life of the software and T_0 is the initial MTTF. In any case, Littlewood's model leads to the reliability function

$$R(t) = \int_0^\infty \exp[-\lambda_i t] \text{gamma}(\alpha, \psi(i)) d\lambda_i$$

where i is the number of failures that have already occurred. Using the relation

$$\lambda(t) = -\frac{1}{R(t)} \left[\frac{dR(t)}{dt} \right],$$

the failure rate after the i th software failure is

$$\lambda(t) = \frac{\alpha}{t + \psi(i)}, \quad t_i \leq t \leq t_{i+1}.$$

The unconditional distribution of \mathbf{T}_i is a Pareto distribution. Figure 12 illustrates the shapes of some typical Pareto distributions. The Pareto distribution is a positively skewed distribution that is known for its long tail that approaches zero more slowly than other popular distributions. Instead of MTTF, Littlewood uses

time-to-failure percentiles. As mentioned before, the possibility of a fault-free program implies that the MTTF may not exist. It so happens that the Pareto distribution with parameters $r, A > 0$ defined by the pdf

$$f(x) = \begin{cases} rA^r \frac{1}{x^{r+1}} & \text{for } x \geq A \\ 0 & \text{otherwise} \end{cases}$$

possesses a finite n th moment if and only if $n < r$, further explaining Littlewood's disparagement of MTTF.

Littlewood and Verrall first presented their model with an example using simulated data[LITT73]. Later, Littlewood used some of Musa's own data and demonstrated that the Littlewood-Verrall model gave a superior fit[LITT78].

One of the obvious inadequacies of the early models was that they related the failure rate (proportion of inputs resulting in failure) to the *number* of faults present in the program version. Certainly some fault-containing code would be encountered more frequently than other fault-containing code. For example, code in a loop will generally be executed more times than straight-line code. In Littlewood's Bayesian Differential Model[LITT79A], each fault exhibits its own failure rate ν_j . The overall program hazard rate when $(i - 1)$ failures have occurred is

$$\lambda_i = \nu_1 + \nu_2 + \cdots + \nu_{(N-i+1)}$$

where n represents the number of detected and corrected faults up to now. The λ_i 's are independent and identically distributed with a gamma pdf. In the previous models, the failure rate only fell after a fault was corrected. (A pseudofailure can be considered to occur at the end of testing, but this merely puts a floor under the reliability[MUSA84].) One is reminded of the hardware reliability adage, "The greater the reliability of a device the more difficult it is to determine its reliability [JORD83]." In this model, long periods without failure can also cause the failure rate to decline. Since faults are not showing up, the ones that are left, if any, must

in toto possess a low failure rate. From an intuitive point of a view, this innovation seems paradoxical: Frozen code should exhibit a constant failure rate. But no one can deny that people's confidence in the reliability of a program increases in the absence of failures. The resolution of the paradox may be that the actual failure rate is indeed constant, but a person's opinion about what its value is varies.

Letting W denote the event that a particular fault ν was not eliminated by elapsed time $\tau = \sum_0^{i-1} t_j$, the pdf for ν given that the fault was not eliminated by elapsed time τ is obtained from Bayes' theorem as

$$\text{pdf}\{\nu|W\} = \frac{\text{Pr}\{W|\nu\} \cdot f(\nu)}{\int_0^\infty \text{Pr}\{W|\nu\} \cdot f(\nu) d\nu}$$

where $f(\nu)$ denotes the pdf of ν at time zero. Under the assumption that $f(\nu)$ is gamma(α, β), $\nu > 0$, $\text{pdf}\{\nu|W\}$ is gamma($\alpha, \beta + \tau$). Furthermore, letting $R = N - i + 1$ be the number of remaining faults, $\text{pdf}\{\lambda_i\}$ is gamma($R\alpha, \beta + \tau$). The first parameter is familiar from straight fault-counting models; however, corrections early on in testing reduce the failure rate more than those later on. The second parameter reflects the subjective belief that long-surviving faults most likely have a low failure rate. That is, those faults that contribute greatest to the program failure rate would tend to be discovered first.

The distribution of interfailure times is

$$\begin{aligned} \text{pdf}\{\mathbf{T}_i|\alpha, \beta\} &\equiv \text{pdf}\{\mathbf{T}_i|\alpha, \beta, t_1, t_2, \dots, t_{i-1}\} \\ &= \int_0^\infty \text{pdf}\{\mathbf{T}_i|\lambda_i\} \text{pdf}\{\lambda_i|\alpha, \beta, t_1, t_2, \dots, t_{i-1}\} d\lambda_i \\ &= \frac{R\alpha(\beta + \tau)^{R\alpha}}{(\beta + \tau + t_i)^{R\alpha+1}}, \end{aligned}$$

a Pareto distribution again.

It might be mentioned that an implicit assumption of the preceding models is that all software failures are observed. What if the failure detection process is imperfect? The reliable-reference problem alluded to earlier or other problems could cause a failure not to be detected.

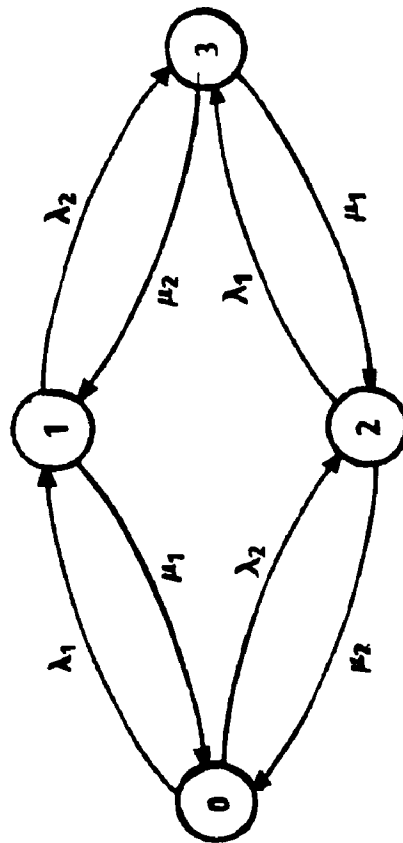


FIGURE 13
FOUR—STATE MARKOV MODEL

Recently Littlewood has developed an adaptive method[LITT85] that allows a time-domain software reliability model to learn from its past mistakes. If a model shows a systematic bias (optimistic or pessimistic), the method can correct that. If the model is just noisy (frequent swings in the predictions) the method will not help.

The Thompson-Chelson Bayesian model[THOM79][THOM80] concerns itself with developing acceptance-rejection criteria for programs. The producing and consuming organizations cooperate in a quality-surveillance program which protects the consumer against delivery of an inferior software product and protects the producer against rejection of acceptable software. Rejection means to have the program tested more. The model uses a gamma-distributed prior for the piecewise-constant failure rate random variable. A second prior is for the probability that at least one fault resides in the program. Other software reliability implicitly assume this probability to be equal to one. The posterior failure rate Cdf is denoted $F(\lambda|H)$, where H is historical failure data. The program is accepted if $F(\lambda|H) \geq 1 - \alpha$, where α is the significance level.

Markov Models

Markov models picture the system as going through a sequence of states. For example these states can be “up” and “down” in order to address repair time [TRIV74][TRIV75]. In Markov models, the probability of a given state transition depends only on the present state. The transitions occur at fixed time intervals. In semi-Markov models, a random amount of time passes between transition times. Costes, Landrault and Laprie[COST78] are able to calculate availability figures with their semi-Markov model. Another use of Markov models assigns a state to each module to study the “macro” reliability when the reliability of the individual modules is known[LITT75A][LITT80][CHEU80].

A couple of Markov models purport to figure in penalty costs [CHEU80][LITT80], but these costs are merely assumed to be known constants. Figure 13 shows a four-state Markov model for a system with failure rates λ_i and repair rates μ_i .

Data-Domain Models

In data-domain models (first described in [NELS73]) a computer program corresponds to computable function, F , whose domain is the set $E = \{E_i, i = 1, 2, \dots, N\}$. E consists of all possible sets of input data values. Each E_i is a complete set of data values needed to run the program. The output of the program is the function value $F(E_i)$. Because of the presence of faults, the computer program will actually implement the function F' . F' is a partial function, since the presence of program faults may prevent the program from terminating [MCGE82]. The sets of input values on which F' produces a software failure (F' differing from F) is E_e . Letting N_e be the number of input data sets E_i in E_e , the probability that a single run will result in a software failure is

$$p = \frac{N_e}{N}.$$

Reliability is the probability of no failure:

$$R = 1 - p = 1 - \frac{N_e}{N}.$$

If p_i ($i = 1, 2, \dots, N$) is the probability that the selected input data set is E_i , and Y_i is 0 if E_i is successful and 1 if E_i results in a software failure, then

$$p = \sum_{i=1}^N p_i Y_i.$$

The reliability (probability of no failure in n runs) is

$$R(n) = R^n = (1 - p)^n.$$

Different applications and users will tend to have different $\{p_i\}$ distributions [CHEU80], resulting in different reliability values from the same software.

Phenomenological Models

The time-domain models estimate their parameters primarily on the basis of observed interfailure times—from the *dynamic* failure behavior of the program. Assuming that the differences between the testing and operational environment can be accounted for, is interfailure-time information “pithy” enough to sufficiently characterize those parameters? It may be that the observations that can be made of a running program are insufficient to construct a good model. At the other extreme are the phenomenological models, such as Software Science [HALS77], in which the *static* features of the program code (such as size, complexity, and structure) are analyzed in an attempt to estimate reliability. The estimated number of faults in a program is given by

$$B = K \left(\frac{V}{EO} \right)$$

where K is a proportionality constant and $EO \approx 3000$ is the average number of mental discriminations between faults. Halstead bases the 3000 figure on psychological research suggesting that the human brain can handle five “chunks” of data at a time. V is the program volume defined as

$$V = N \log_2 n.$$

N is the program length consisting of the total number of operator and operand occurrences, and n is the total number of distinct operators and operands. Validation efforts have produced inconclusive results. Another metric is McCabe’s cyclomatic complexity. The value of $V(G)$ is the number of enclosed areas on the plane of the program graph, plus one for the unbounded area outside the graph. It is equivalent to determining the graph theoretic “basis set.” In graph theory, the cyclomatic

number is

$$V(G) = e - n + p$$

where n is the number of nodes, e is the number of edges, and p is the number of connected components. McCabe's cyclomatic complexity—the number of linearly independent program paths—through the program graph—is $V(G) + p$. [MCCA76]

Thayer and Lipow derived through multiple regression techniques the empirical relationship

$$p = 0.0598b$$

where p is the number of modifications required to correct development-phase failures and b is the number of branches in the software. Thayer and Lipow had started out with nine predictor variables, such as the number of procedure calls, loop complexity, and number of input/output statements, but in the end eliminated all but one as insignificant.

Fault Seeding

Mills[MILL72] proposed a “fault-seeding” model that works by injecting known bugs into a program to see what proportion of them are discovered. The number of real bugs found is then presumed to be in the same proportion to the total number of real bugs that were present. If y is the number of injected faults, u is the number of indigenous faults found so far, and v is the number of induced faults found so far, the maximum likelihood estimator for x , the original number of indigenous faults, is

$$x = \left\lfloor \frac{yu}{v} \right\rfloor.$$

Variations of this approach include having two independent testing teams debug the same program and using a count of the common faults.

Discussion

The current state of the art of software reliability modeling is vulnerable to criticism in several respects:

- 1.) Validity—No model has been subjected to the kind of validation that would inspire confidence in its use on a widespread commercial scale. Only recently has a great deal of thought been put into what constitutes a fair validation test with which to compare the competing models[LITT85].
- 2.) Environment—Dissimilarity between the testing and operational environments is only addressed in Musa's model, and he tries to account for the time differential, but time is only one aspect of the dissimilarity.
- 3.) Workload—Another problem in existing models is that they do not take into account the effect of workload—the rate at which transactions are presented to the system—on the failure rate.
- 4.) Immutability of code—In existing models, immutability of the code is assumed, but as time goes on the size of program will change, and last-minute requirements changes may turn a program into a patchwork-quilt that diminishes debugging efficiency. That is, a “law of increasing entropy” may exist for programs.
- 5.) Variation in testing intensity—Testing intensity is in reality not uniform; it oscillates up and down through development. (It especially increases near delivery deadlines.) Also,

“The models assume path homogeneity—that is, data are entered randomly and such data uniformly exercise all code. This is in direct contradiction to the reality that the statistically significant paths cover a small percentage (say under 10%) of the code [BEI784].”

- 6.) Equal weighting of all failures—The lack of attention to penalty cost is another criticism that has been leveled at the current models and is the

crux of the research presented in the chapters to follow. Software reliability has usually been defined in terms of failure occurrence as opposed to failure effect. Since the effect of a failure can be so minor as to be ignorable or so major as to kill human beings, it seems silly to count them equally. Failure effect is probably more in the mind of a user when he says “software reliability” than is mere failure occurrence.

The research described in the remainder of this dissertation involves a new software reliability modeling technique that squarely addresses criticism #6.

CHAPTER 4

DEVELOPMENT OF THE MODELING TECHNIQUE

Now a new modeling technique will be presented. This chapter will provide an overview of the technique and develop its failure-counting component; the next chapter will develop the penalty-cost component and tie the two components together.

Describing empirical phenomena mathematically necessarily involves making some simplifying assumptions [CHIA68]. Framing the problem in terms of random functions will provide insight and useful answers. The failure-counting component of the model is built from a set of simple, abstract assumptions about the nature of software failure occurrence.

Statement of the Question

Given sufficient historical data, for example the results of a suitable simulation run, probabilistically characterize the aggregate penalty cost that a system will incur from software failure during operational time interval $(0, t]$.

Overview of the Modeling Technique

A model is the formalized expression of a quantitative understanding of a situation. The presence of uncertainty in the phenomenon of software failure and its associated penalty cost is acknowledged by the presence of random variables. The modeling technique developed here makes some assumptions regarding the phenomenon and uses probabilistic techniques to yield a distribution of future values, conditioned by a knowledge of past values. Prediction is made on the basis of an *informative experiment*, in which a sufficiently large number of software

failures and their concomitant penalty costs have been observed. Additionally, prior information may be available concerning unknown distributional parameters. This prior information can place plausibilities or probabilities on the various possible values of these parameters.

Upon the occurrence of a software failure (in simulation or real life) the consequences of the failure must be quantitatively appraised, by subjective or objective (possibly automated) means. The approach supposes a function that measures the "loss," the *penalty cost* that has been incurred. The primary random variables are the interfailure times and the individual penalty costs. A number of associated random variables are of interest:

- 1.) The incidence of software failures in a specified period of time.
- 2.) The failure times.
- 3.) The time to the n th failure.
- 4.) The failure recurrence times, backward and forward.
- 5.) The penalty cost of an individual failure.
- 6.) The aggregate penalty cost over a time interval.

While many of the general results covered here can be used for arbitrary probability distributions, the adoption of a few simple assumptions about the phenomenon of software failure/penalty cost will lead to special mathematical forms that will facilitate computation and analysis. These distributions will each be completely characterized by a succinct set of parameters whose interpretation will depend on the individual distribution. The parameters will always be positive.

Technical Summary

Since the details of the modeling technique are to be found distributed throughout two chapters, the model is concisely summarized here.

Parameters

The parameters of the model are

- 1.) Selected structure of failure-counting component (class of stochastic process that probabilistically characterizes failure occurrence times, failure frequency and interfailure times).
- 2.) Selected structure of penalty-cost component.

Inputs

The two inputs to the model are

- 1.) Failure rate function (constant rate λ , time function $\lambda(t)$ or random variable λ).
- 2.) Penalty cost distribution (empirical or theoretical)

Outputs

The model will provide

- 1.) A probabilistic characterization of the aggregate penalty cost that will be incurred over a future time interval by a released software system. This characterization includes probability density/mass function and various statistics such as quantiles, moments, etc.
- 2.) As special cases (penalty costs all equal to 1), mean-time-to-failure and other figures of merit already obtainable from current reliability models.

Description

The penalty costs associated with the software failures are the s -independent random variables

$$X_1, X_2, \dots$$

An individual penalty cost is probabilistically characterized by the cumulative distribution function $S(x)$. If the counting process $\mathbf{N}(t)$ gives the cumulative

number of software failures that have occurred in time interval $[0, t)$, the *aggregate penalty cost* for that interval is the compound stochastic process

$$\mathbf{Z}(t) = \mathbf{X}_1 + \mathbf{X}_2 + \cdots + \mathbf{X}_{\mathbf{N}(t)}.$$

It is assumed that penalty cost is s -independent of $\mathbf{N}(t)$. The aggregate penalty cost $\mathbf{Z}(t)$ is probabilistically characterized by the cumulative distribution function

$$F(z, t) = \sum_{n=0}^{\infty} p_n(t) S^{n*}(z)$$

where t is operating time projected into the future on the basis of no further fault correction; S^{n*} is the n th convolution of $S(\cdot)$ with itself, and $p_n(t)$ is the probability mass function for n failures during time interval $[0, t)$ (i.e., $p_n(t) = \Pr\{\mathbf{N}(t) = n\}$).

The probability mass function $p_n(t)$ is a property of the *failure-counting component* of the model. This component models software failure occurrence in time as a stochastic process. The nature of software occurrence lends itself to being productively modeled as a Poisson process. Variants arise according to the form of the failure rate function. A piecewise-constant failure rate makes for a temporally homogeneous Poisson process, with negative exponentially distributed interfailure times. In this case,

$$p_n(t) = \frac{(\lambda t)^n \exp(-\lambda t)}{n!}.$$

This variant conforms to the Jelinsky-Moranda, Shooman and Musa models. A Bayesian updating technique can be used to revise the estimated failure rate as empirical data comes in.

If the failure rate is considered a nonrandom function of time, a nonhomogeneous Poisson process results, conforming to the Goel-Okumoto NHPP, Schick-Wolverton and Wagner models. In this case,

$$p_n(t) = \frac{\int_0^t \lambda(u) du}{n!} \exp \left[- \int_0^t \lambda(u) du \right], \quad n = 0, 1, 2, \dots$$

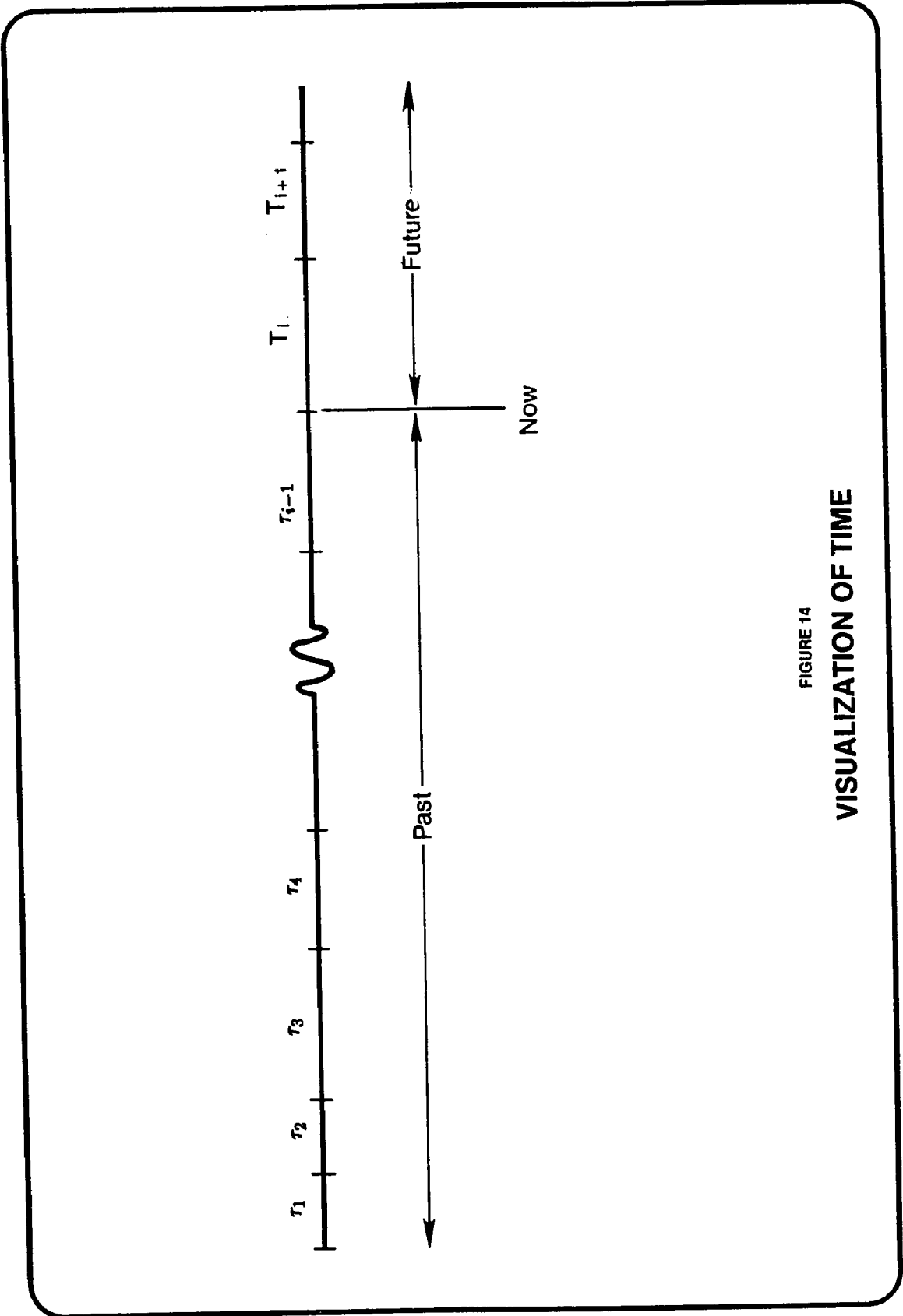


FIGURE 14

VISUALIZATION OF TIME

A gamma-distributed failure rate, with parameters α and β , forms a Pólya process that is compatible with the Littlewood models:

$$p_n(t) = \binom{n + \alpha - 1}{n} \left(\frac{t}{\beta + t} \right)^n \left(\frac{\beta}{\beta + t} \right)^\alpha.$$

Additional current and future software reliability models are accommodated through the weighted Poisson process variant, the most general:

$$p_n(t) = \frac{1}{n!} \int_0^\infty \exp(-\lambda t) (\lambda t)^n f_{t|\lambda}(\lambda) d\lambda.$$

The cumulative distribution function $S(x)$ probabilistically characterizes the individual penalty cost \mathbf{X} and can be empirical or theoretical; continuous, discrete or mixed. Certain forms for $S(x)$ result in simple recurrence relations that facilitate calculations.

Now the failure-counting component will be developed in detail.

The Scenario

A computer program begins running at time epoch τ_0 , measured in CPU seconds since an arbitrary time origin. At time epoch $\tau_{(1)}$, the first software failure occurs. The waiting time to this first failure, in CPU seconds, is $\tau_1 = \tau_{(1)} - \tau_0$. At this point debugging personnel will isolate and remove the fault that caused the failure. The time spent on this repair activity will be ignored when calculating interfailure times. (This is not to say that the repair time is irrelevant when modeling availability, imperfect fault correction, or when assessing penalty cost.) Then the program is restarted and runs until the second software failure occurs, this at time epoch $\tau_{(2)}$. The second interfailure interval is $\tau_{(2)} - \tau_{(1)}$ CPU seconds long. This test-repair cycle will repeat itself until the program is released.

Suppose that i software failures have already been observed. The time epochs $\tau_{(1)}, \tau_{(2)}, \dots, \tau_{(i)}$ are known. However, the time epochs of the software failures yet-to-come are anybody's guess. This uncertainty is reflected by the use of the random

variables $\mathbf{T}_{(i+1)}, \mathbf{T}_{(i+2)}, \dots$ for the future failure time epochs and $\mathbf{T}_{i+1}, \mathbf{T}_{i+2}, \dots$ for the future interfailure intervals. Figure 14, after[LITT85], shows how the time notation relates to the concepts of past, present and future.

To view the situation as a stochastic process, one sees pictures of random input data being presented to the program. From the program's perspective the input is randomly generated, in the sense that the program cannot exactly predict the input data sets. (If they were exactly predictable in content and order there would be no need for input data. [Reference tables read from a file for convenience do not count.]) Some of the input data sets will result in "failure" and the others in "success." Furthermore, the computer may be embedded in a system or environment where a *loss event* is possible (probability greater than zero). In severe cases the computer, in failing, is in a position to inflict damage or injury to proximate property or people. The amount of loss that will be incurred is generally unpredictable in advance, although it will be aggravated by the presence of *hazards*. While the occurrence time and severity of an individual software failure is a random event, propositions about software failures in the aggregate can be assigned probabilities through modeling techniques such as those presented here.

A Single Released Program Version

Since software failure is a phenomenon characterized by highly localized events distributed randomly in a time continuum, it will prove fruitful to view the evolution of failure occurrence through time as a *point process*, a type of stochastic process. Suppose that operation of a single program version begins at time t_0 and is observed for a subsequent T seconds. The continuum space of the point process is an interval $\{t : t_0 \leq t \leq t_0 + T\}$ of the real line. Each element t represents an instant of time during the observation period. A realization of the point process is a sequence of failure time epochs $\{t_{(1)}, t_{(2)}, \dots, t_{(n)}\}$, where n is the total number of software

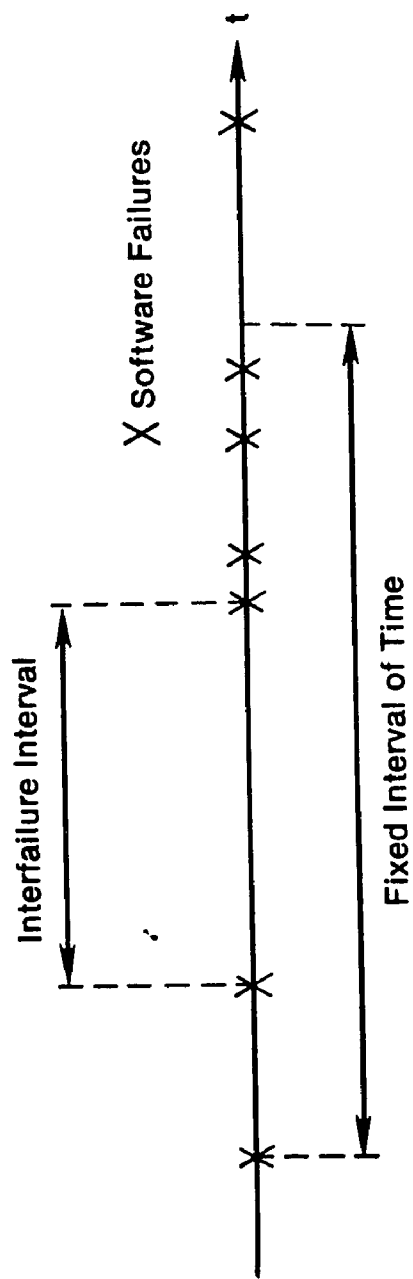


FIGURE 15
TWO RANDOM FAILURE VARIABLES

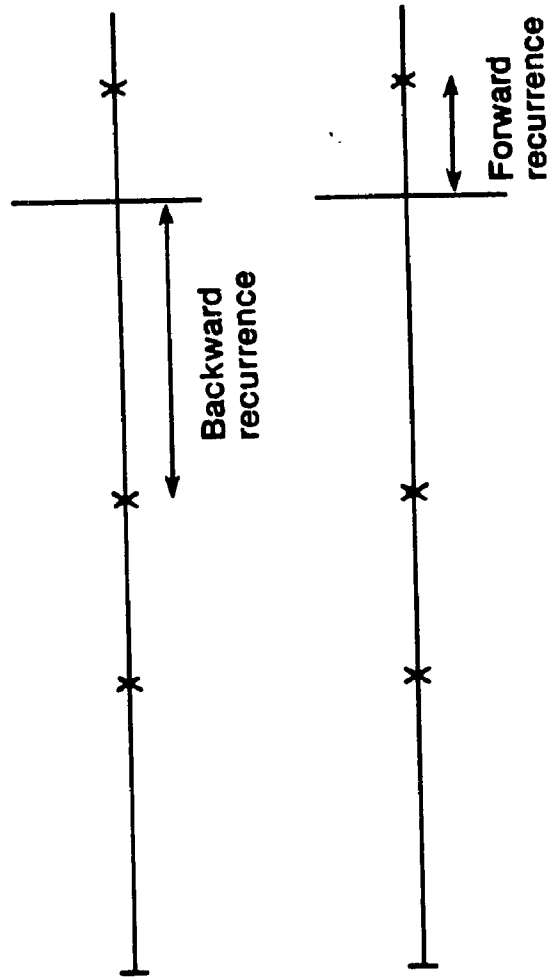


FIGURE 16
SOFTWARE FAILURE RECURRENCE TIMES

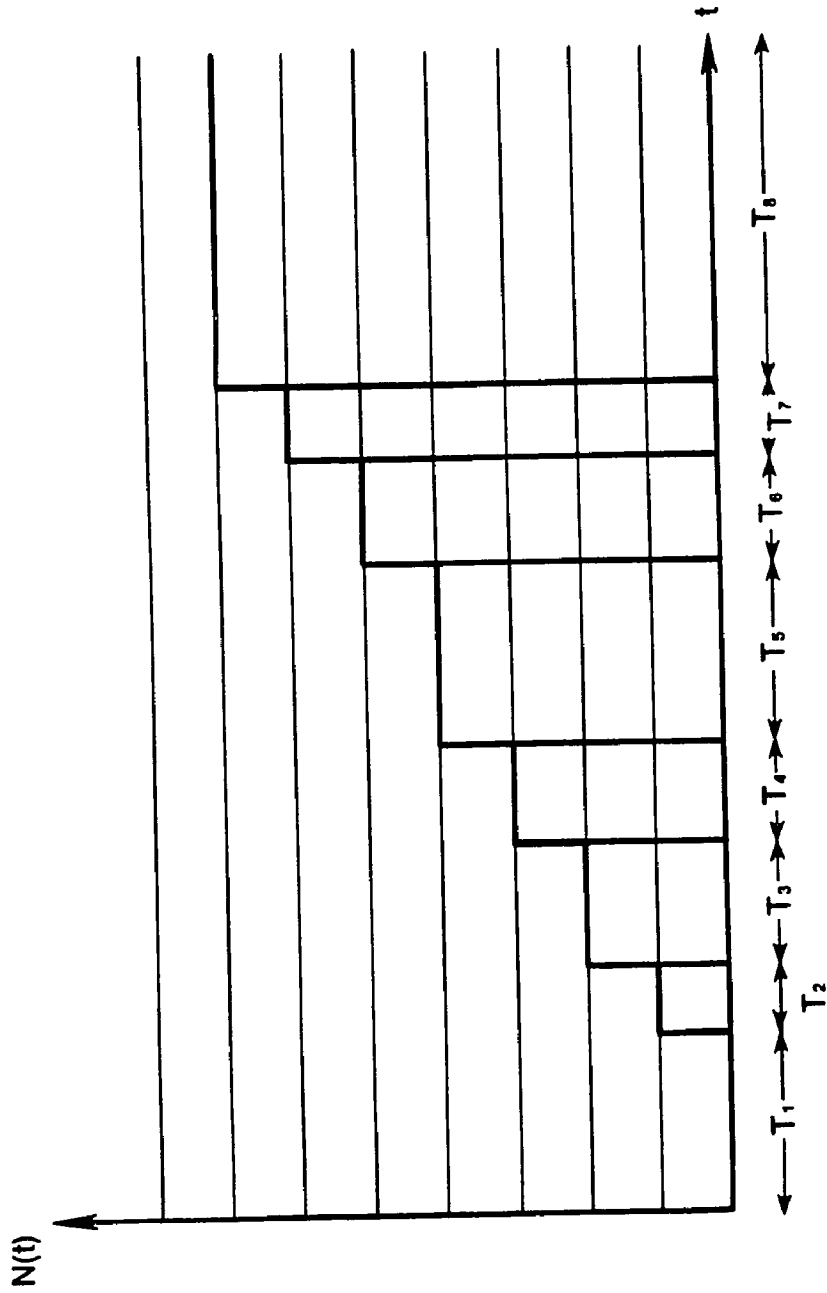


FIGURE 17
THE COUNTING PROCESS $N(t)$

failures occurring during the observation period $[t_0, t_0 + T]$. The failure times are ordered such that $t_0 \leq t_{(1)} < t_{(2)} < \dots < t_{(n)} < t_0 + T$.

Several features of the failure point process are of particular interest: the numbers of failures occurring in fixed time intervals, the interfailure times, and the forward/backward recurrence times. Figure 15 shows a time-line illustration of the two major random variables of interest in the failure-counting component of the model. Figure 16 shows the meaning of forward and backward software recurrence times from an arbitrary time origin.

The Software Failure Counting Process

To probabilistically characterize the underlying probability laws governing the failure process it is necessary to consider

- 1.) The probability distribution of \mathbf{T}_1 ,
- 2.) The probability distribution of \mathbf{T}_2 for $\{\mathbf{T}_1 = t_1 \text{ and } \mathbf{T}_2 = t_2\}$,

etc.

Unfortunately this process is too general to work with[GIRA66]. So suppose for a single program version that the interfailure times $\{\mathbf{T}_1, \mathbf{T}_2, \dots\}$ form a sequence of mutually s -independent nonnegative identically distributed random variables. This assumption suggests itself since after each failure the identical program version is restarted: The process is "self-renewing." Assume the \mathbf{T}_i are continuous random variables with common Cdf $F(t)$. Let $F^{(k)}(t)$ denote the Cdf of $\mathbf{T}_{(k)}$. Because $\mathbf{T}_{(k)}$ is the sum of mutually s -independent and identically distributed random variables, $F^{(k)}$ is computed as the k -fold convolution of F with itself. For notational convenience, define

$$F^{(0)}(t) = \begin{cases} 1, & t \geq 0 \\ 0, & t < 0. \end{cases}$$

To study the statistical characteristics of the numbers of failures occurring in fixed time intervals, it is helpful to introduce another stochastic process: the *counting*

process associated with the failure point process. Consider the nonnegative integer-valued stochastic process $\{\mathbf{N}(t, \omega), t \geq 0, \omega \in \Omega\}$. The counting process $\mathbf{N}(t, \omega)$ is thus for each outcome ω in the sample space Ω a piecewise-constant function of t that will be denoted by $\mathbf{N}(t)$. The random variable $\mathbf{N}(t)$ gives the cumulative number of software failures that have occurred during the time interval $[0, t)$. Figure 17 illustrates the realization of a typical software failure counting process $\mathbf{N}(t)$. Jumps occur at the time epochs $\mathbf{T}_{(1)}, \mathbf{T}_{(2)}, \dots, \mathbf{T}_{(n)}$. At the beginning ($t = 0$), the counting process $\mathbf{N}(t)$ will have the value zero. After the first software failure, $\mathbf{N}(t)$ will have the value one; after the second, two. And so on. More formally,

$$\mathbf{N}(t) = \sup_k \{t_{(k)} \leq t\}$$

with the proviso that $\mathbf{N}(t) = 0$ if $t_{(1)} > t$. If $u < v$, then $\mathbf{N}(v) - \mathbf{N}(u)$ yields the number of software failures that have occurred during the time interval $(u, v]$. It is desired to find

$$p_n(t) \equiv \Pr\{\mathbf{N}(t) = n\},$$

the probability of n software failures occurring during time interval $(0, t)$. Since $\mathbf{N}(t) = n$ if and only if $\mathbf{T}_{(n)} \leq t < \mathbf{T}_{(n+1)}$,

$$\begin{aligned} \Pr\{\mathbf{N}(t) = n\} &= \Pr\{\mathbf{T}_{(n)} \leq t < \mathbf{T}_{(n+1)}\} \\ &= \Pr\{\mathbf{T}_{(n)} \leq t\} - \Pr\{\mathbf{T}_{(n+1)} \leq t\} \\ &= F^{(n)}(t) - F^{(n+1)}(t). \end{aligned}$$

The *renewal function* $N(t) \equiv E\{\mathbf{N}(t)\}$, met in the related research survey, is computed by the recurrence

$$N(t) = F(t) + \int_0^t N(t-x)f(x) dx.$$

The *renewal density* $m(t)$ is the s -expected number of software failures in a narrow interval near t :

$$\begin{aligned} m(t) &= \frac{dN(t)}{dt} \\ &= \sum_{n=1}^{\infty} f^{(n)}(t) = f(t) + \int_0^t m(t-x)f(x) dx \\ &= f(t) + m(t) * f(t) \end{aligned}$$

with $\lim_{t \rightarrow \infty} m(t) = 1/E\{\mathbf{T}\}$ [ROSS70]. Let the Laplace transform of $f(x)$ be denoted $\mathcal{L}_f(s)$:

$$\mathcal{L}_f(s) = \int_0^{\infty} \exp(-sx) f(x) dx$$

and the Laplace transform of $m(x)$ as $\mathcal{L}_m(s)$:

$$\mathcal{L}_m(s) = \int_0^{\infty} \exp(-sx) m(x) dx.$$

Since the Laplace transform of the convolution of two functions is the product of their Laplace transforms,

$$\mathcal{L}_m(s) = \mathcal{L}_f(s) + \mathcal{L}_m(s)\mathcal{L}_f(s)$$

or

$$\mathcal{L}_m(s) = \frac{\mathcal{L}_f(s)}{1 - \mathcal{L}_f(s)}$$

and

$$\mathcal{L}_f(s) = \frac{\mathcal{L}_m(s)}{1 + \mathcal{L}_m(s)}.$$

This process is still too general to manipulate fruitfully, so the software failure counting process $\mathbf{N}(t)$ itself will be assumed to satisfy the following reasonable conditions (Poissonian assumptions leading via standard calculations to a temporally homogeneous Poisson distribution):

- 1.) The numbers of software failures occurring in disjoint time intervals are statistically independent;

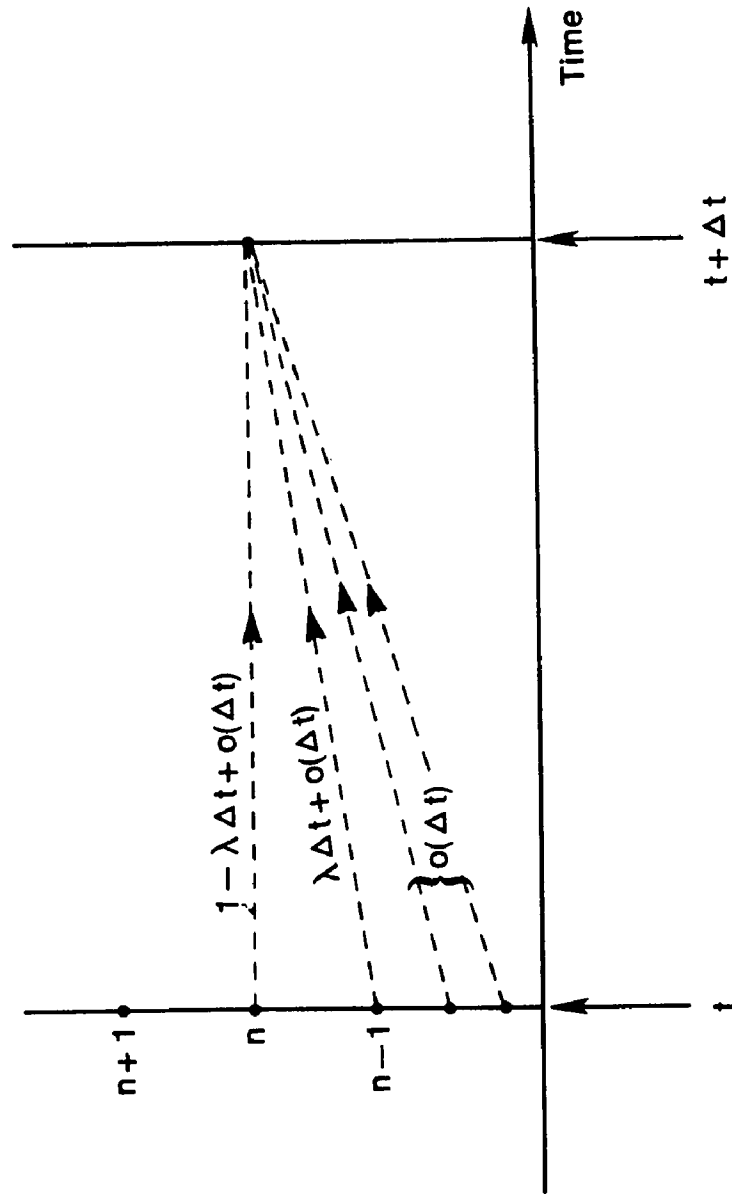


FIGURE 18

STATE TRANSITIONS OF FAILURE-COUNTING PROCESS

- 2.) The probability of exactly one software failure occurring during the “small” time interval $(t, t + \Delta t)$ is proportional to the length of that interval:

$$\Pr\{\mathbf{N}(t + \Delta t) - \mathbf{N}(t) = 1\} = \lambda\Delta t + o(\Delta t)$$

where λ is the *failure rate*—the average number of software failures per unit time;

- 3.) The probability of multiple software failures occurring during the interval $(t, t + \Delta t)$ is negligible with respect to the length of that interval:

$$\Pr\{\mathbf{N}(t + \Delta t) - \mathbf{N}(t) > 1\} = o(\Delta t);$$

and

- 4.) Counting begins at time 0:

$$\Pr\{\mathbf{N}(0) = 0\} = 1.$$

(A function is described as $o(h)$ if $\lim_{h \rightarrow 0} f(h)/h = 0$.) Figure 18 illustrates the probabilities associated with the various state transitions of the failure-counting process $\mathbf{N}(t)$. Condition (1) means that the number of failures in nonoverlapping intervals are *s*-independent random variables, regardless of the size of the intervals or their distance from each other. Basically, these additional assumptions rule out “chain reactions” among software failures; “trends” such as increasing, decreasing or oscillating; and multiple simultaneous failures. Note that the exclusion of trends applies to frozen code only. Reliability growth will result in a decreasing failure rate in the long run as faults are removed from successive versions of the program. The parameter λ always has the dimension $[\text{time}]^{-1}$. The probability $p_0(t)$ of zero software failures in the time interval $(0, t)$ can be derived from these assumptions as follows: The probability $p_0(t + dt)$ of zero software failures in the time interval $(0, t + dt)$ equals the probability of zero software failures in the time interval $(0, t)$

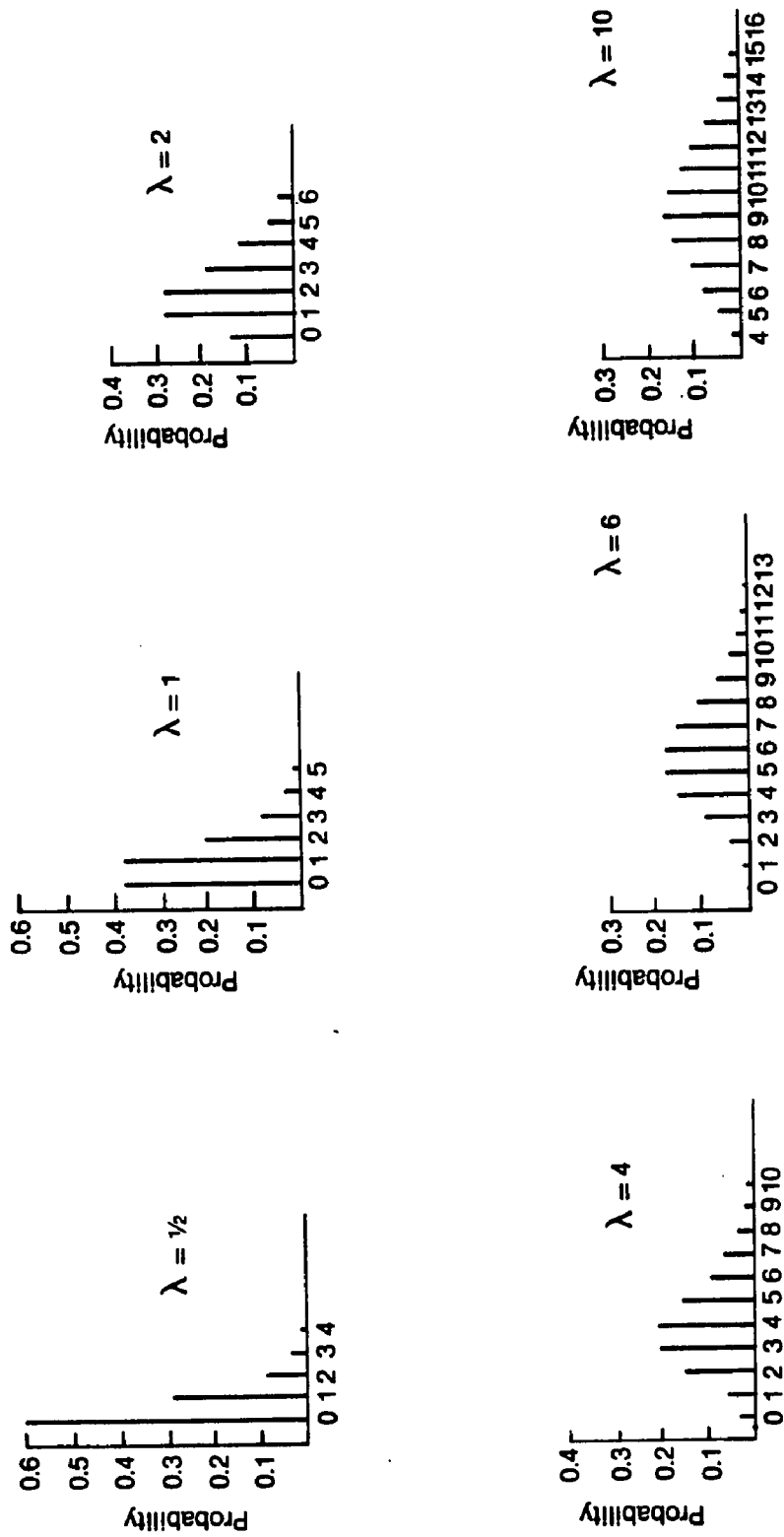


FIGURE 19
POISSON PROBABILITIES ($t=1$)

times the probability of zero software failures in the time interval $(t, \Delta t)$, since both these events must occur. Since by assumption (1) these two events are s -independent, their probabilities are multiplied to obtain the probability $p_0(t + dt)$. From assumptions (2) and (3), we know that

$$p_0(t + \Delta t) = p_0(t)(1 - \lambda\Delta t).$$

Simplifying, we get

$$\frac{p_0(t + \Delta t) - p_0(t)}{\Delta t} = -\lambda p_0(t).$$

As Δt approaches zero as a limit, we are left with

$$\frac{dp_0(t)}{dt} = -\lambda p_0(t).$$

Integrating both sides so as to solve this differential equation yields

$$\ln p_0(t) = -\lambda t + C.$$

At the boundary condition $t = 0$, the system is operating; hence $t = 0$, $p_0(0) = 1$, $\ln p_0(t) = 0$, and the constant of integration C is seen to be zero, so

$$p_0(t) = \exp(-\lambda t).$$

The next step is to determine the more general probability $p_n(t)$ of n software failures in time interval $(0, t)$. The probability $p_n(t + dt)$ can occur in two ways: n software failures in time interval $(0, t)$ followed by zero software failures in time interval $(t, t + dt)$; or $n - 1$ software failures in time interval $(0, t)$ followed by one software failure in time interval $(t, t + dt)$. Any other combination requires more than one software failure to take place during $(t, t + dt)$, an event all but prohibited by assumption (3). Thus

$$\begin{aligned} p_n(t + dt) &= p_n(t)(1 - \lambda dt) + p_{n-1}(t)(\lambda dt) \\ &= p_n(t) - \lambda dt[p_n(t) - p_{n-1}] \end{aligned}$$

This recurrence can be rewritten

$$p_n(t) = \frac{(\lambda t)^n \exp(-\lambda t)}{n!}.$$

Figure 19 illustrates the probability mass functions for various values of λ when $t = 1$. When $n = 0$, the formula gives $\exp(-\lambda t)$, which agrees with our previous result.

The s -expected value can be computed as

$$\begin{aligned} E\{\mathbf{N}(t)\} &= \sum_{n=0}^{\infty} n \frac{(\lambda t)^n \exp(-\lambda t)}{n!} \\ &= \sum_{n=1}^{\infty} n \frac{(\lambda t)^n \exp(-\lambda t)}{n!} \\ &= \lambda t \sum_{n=1}^{\infty} \frac{(\lambda t)^{n-1} \exp(-\lambda t)}{(n-1)!} \\ &= \lambda t. \end{aligned}$$

The last simplification is made possible by the requirement that the probabilities for all n must add up to one.

This type of counting process would come under the classification of a *temporally homogeneous Poisson process*.

Besides probabilistically characterizing the cumulative number of software failures, another objective of the failure-counting component of the model is to probabilistically characterize the future interfailure intervals $\mathbf{T}_i, \mathbf{T}_{i+1}, \mathbf{T}_{i+2}, \dots$, given a sequence of observed past interfailure intervals $t_1, t_2, t_3, \dots, t_{i-1}$. When a program version is released, its code becomes frozen. Thus it makes sense to speak of a constant failure rate $\lambda(t) = \lambda$ during an operational period. The value of λ cannot be deduced theoretically but must be determined empirically through the use of statistical techniques. Since the time origin is arbitrary, the preceding formula will apply to the number of software failures falling in any intervals of a given length

$t > 0$. and the Cdf will be

$$\text{Cdf}\{\mathbf{N}(t)|\lambda\} = \sum_{i=0}^{\lfloor n \rfloor} \exp(-\lambda t) \frac{(\lambda t)^i}{i!}, \quad n = 0, 1, 2, \dots$$

The expected value is

$$\text{E}\{\mathbf{N}(t)\} = \sum_{j=0}^{\infty} j \frac{(\lambda t)^j}{j!} \exp(-\lambda t) = t \lambda \exp(-\lambda t) \left(1 + \lambda t + \frac{(\lambda t)^2}{2!} + \frac{(\lambda t)^3}{3!} + \dots \right).$$

The infinite series has value $\exp(\lambda t)$, so $\text{E}\{\mathbf{N}(t)\} = \lambda t$.

The mean of the distribution is $\mu = \text{E}\{\mathbf{N}(t)\} = \lambda t$ and, interestingly, the variance $\text{Var}\{\mathbf{N}(t)\} = \lambda t$ also. The *failure intensity* $\lambda > 0$ indicates the average number of failures in a unit time interval. Failure occurrence can be seen to depend on both the length of time exposure t and the failure intensity λ .

Interfailure Times

The constant failure rate leads to an interfailure time random variable \mathbf{T} that is negative exponentially distributed because the event of zero software failures in time t is equivalent to the event that the waiting time to the first event is greater than t .

$$\text{Pr}\{\mathbf{T} > t\} = \text{Pr}\{\text{zero failures in time } t\} = p_0(t) = \exp(-\lambda t).$$

The cumulative distribution function is

$$\text{Cdf}\{\mathbf{T}\} = \text{Pr}\{\mathbf{T} \leq t\} = 1 - \exp(-\lambda t).$$

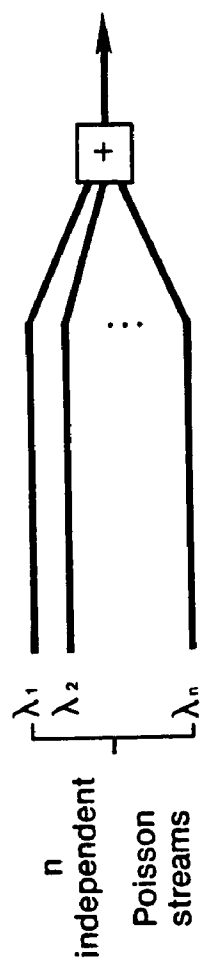
And the probability density function is

$$\text{pdf}\{\mathbf{T}\} = \frac{d\text{Cdf}\{\mathbf{T}\}}{dt} = \lambda \exp(-\lambda t).$$

The reliability function is

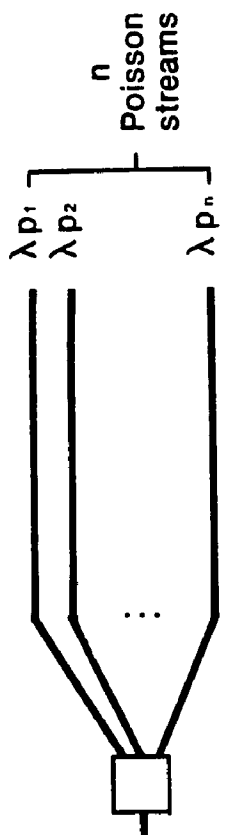
$$R(t) = 1 - \text{Cdf}\{\mathbf{T}\} = \exp(-\lambda t).$$

Pooled stream rate
$$\lambda = \sum_{i=1}^n \lambda_i$$



Superposition of
Failure-Counting Processes

Poisson stream
with rate
 λ



Decomposition of
Failure-Counting Process

FIGURE 20
DECOMPOSITION OF FAILURE-COUNTING PROCESS

Thus the Poisson probability law and the negative exponential probability law both describe the same failure process but from the perspectives of two different random variables, \mathbf{T} and $\mathbf{N}(t)$. The negative exponential distribution can also be demonstrated through a limit argument: At each “instant” $\Delta t, 2\Delta t, 3\Delta t, \dots$, a software failure can either occur or not occur. Suppose the probability of failure at each instant is p . Then

$$\Pr\{\mathbf{T}_1 \leq k\Delta t\} = 1 - (1 - p)^k$$

because the event “ $\mathbf{T} \leq k\Delta t$ ” means that a software failure occurred at at least one of the k instants. Now let $\Delta t \rightarrow 0$. As the size of the moments shrink, so too must p , but in such a way that the failure rate $\lambda = p/(\Delta t)$ remains fixed. In the limit as $k \rightarrow \infty, \Delta t \rightarrow 0$ and $k\Delta t \rightarrow t \geq 0$, the formula becomes

$$\begin{aligned} \Pr\{\mathbf{T}_1 \leq t\} &= \lim_{\Delta t \rightarrow 0} \left[1 - (1 - \lambda\Delta t)^{\frac{t}{\Delta t}} \right] \\ &= 1 - \exp(-\lambda t), \end{aligned}$$

the negative exponential Cdf. See[FELL68, BUSH55].

A theoretical argument for the use of this type of Poisson process is that the superposition of the points of many s -independent sparse (not necessarily Poisson) point processes converges to a Poisson process[CINL72]. In fact, “It has been demonstrated by many authors that the number of arbitrary renewal processes needed to achieve the Poisson process approximately, is only about five or six [MURT74].” Likewise a Poisson process can be decomposed into Poisson processes. This decomposition/superposition property reflects the modular structure of software: *The failure law for the overall program must be the same as the failure rate governing each module.* Figure 20 illustrates the decomposition and superposition of Poisson processes. Information theoretic arguments regarding entropy can also be adduced[GROS74].

Software reliability researcher Musa has performed and reviewed a number of studies, and his conclusion is, “Empirical evidence indicates that the [failure interval] independence and [negative] exponential distribution assumptions are soundly based[MUSA84].” Even Littlewood, while criticizing many aspects of current models, concedes the validity of the conditional negative exponential pdf above.

Distributional Properties

For any positive numbers a and b , the negative exponential distribution obeys the *memoryless* property

$$\begin{aligned}\Pr\{\mathbf{T} > a + b | \mathbf{T} > b\} &\equiv \frac{\Pr\{\mathbf{T} > a + b\}}{\Pr\{\mathbf{T} > b\}} \\ &= \frac{\exp[-\lambda(a + b)]}{\exp(-\lambda b)} \\ &= \exp(-\lambda a) = \Pr\{\mathbf{T} > a\}.\end{aligned}$$

This implies that the forward and backward recurrence times are both distributed identically to the interfailure times.

The parameterization $\theta = \lambda^{-1}$ gives

$$\text{pdf}\{\mathbf{T}|\theta\} = \theta^{-1} \exp(-t/\theta), \quad t \geq 0.$$

The Cdf is

$$\text{Cdf}\{\mathbf{T}|\lambda\} = \int_0^t \lambda \exp(-\lambda x) dx = 1 - \exp(-\lambda t), \quad t \geq 0.$$

The mean of \mathbf{T} (the MTTF) is

$$\text{MTTF} = \int_0^{\infty} \text{Sf}\{\mathbf{T}|\lambda\} dt = \int_0^{\infty} \exp(-\lambda t) dt = \frac{\exp(-\lambda t)}{-\lambda} \Big|_0^{\infty} = 1/\lambda = \theta.$$

The variance is $\text{Var}\{\mathbf{T}\} = \theta^2$. The Laplace transform is $\mathcal{L}_r(t) = \lambda/(r + \lambda)$.

The i th time-to-failure percentile is

$$\tau_i = \frac{\ln(1 - .01i)}{-\lambda}.$$

The median time-to-failure is

$$\tilde{t} = -\frac{\ln 0.5}{\lambda}.$$

The probability distribution of the time epoch $\mathbf{T}_{(n)} = \mathbf{T}_1 + \mathbf{T}_2 + \cdots + \mathbf{T}_n$ can be obtained as follows: Let $F_n(t) = \Pr\{\mathbf{T}_{(n)} \leq t\}$. Then $1 - F_n(t) = \Pr\{\mathbf{T}_{(n)} > t\} = \Pr\{\mathbf{N}(t) \leq n\}$; consequently

$$1 - F_n(t) = \sum_{k=0}^{n-1} \frac{(\lambda t)^k \exp(-\lambda t)}{k!}.$$

Differentiation with respect to t produces

$$f(t) = \frac{\lambda}{(n-1)!} (\lambda t)^{n-1} \exp(-\lambda t), \quad t \geq 0.$$

This probability law would be classified as a special n -stage Erlang distribution and always arises from the sum of n s -independent and identically distributed negative exponential random variables.

Given that $n > 1$ software failures have occurred in an interval $[0, t]$, the n times $t_{(1)} < t_{(2)} < \cdots < t_{(n)}$ at which failures occurred (when unordered) are uniformly distributed[FELL68].

The distribution is bimodal when $m_1 = \lambda t - 1$ is an integer, in which case the modes are at $m_1 = \lambda t - 1$ and $m_2 = m_1 + 1 = \lambda t$. The distribution is unimodal when $m_1 = \lambda t - 1$ is nonintegral; in this case the mode is $m = \lfloor \lambda t \rfloor$.

Delivered Failure Intensities

The failure intensity during an operational period will depend on the testing activity that took place prior to release.

So far the analysis has centered on the characteristics of a single program version. The program version was considered to exhibit some particular failure rate λ . As the program evolves as the result of fault correction, the failure rate will

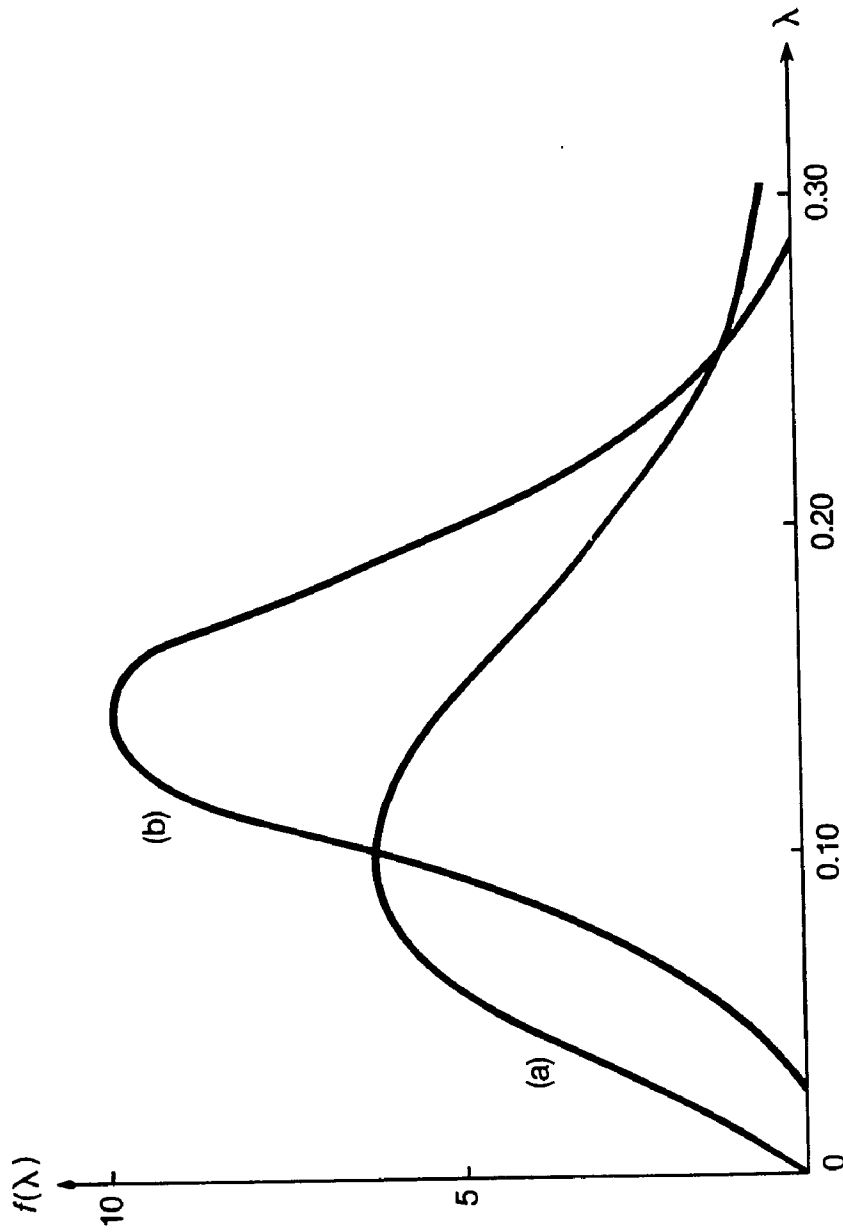


FIGURE 21

TWO BAYESIAN PRIOR DISTRIBUTIONS FOR λ
 (mean = 0.125; Var $\{ (b) \} = \frac{1}{3}$ Var $\{ (a) \}$)

vary. Each version will have its own failure rate. In the long run, the failure rate should drop, as the hoped-for phenomenon of *reliability growth* occurs.

The *failure intensity* λ_i is the failure rate of the i th program version. This program version reflects the correction of $(i - 1)$ faults. The failure intensity can be estimated on the basis of the interfailure times $\tau_1, \tau_2, \dots, \tau_{i-1}$ or the failure epochs $\tau_{(1)}, \tau_{(2)}, \dots, \tau_{(i-1)}$ or by information about the static properties of the program. The estimation can be subjective. The estimation can come from empirically studying the failure behavior (i.e., statistically). The conditional probability density function of the i th interfailure interval is

$$\text{pdf}\{\mathbf{T}_i|\lambda_i\} = \lambda_i \exp(-\lambda_i t_i)$$

The failure-counting component of the model is compatible with existing software reliability models because it can borrow their failure intensity formulations. In these models the failure rate is computed from the history of interfailure times or failure-occurrence times, plus sometimes historical or subjective information. Because the failure-counting component does not concern itself with how the failure intensity is arrived at, it escapes the controversy directed at those formulations. As the formulations improve, the failure-counting component will also.

As simple examples, under the Jelinsky-Moranda formulation the failure intensities would be

$$\lambda_i = K[N(\infty) - (i - 1)],$$

and under the Shooman formulation they would be

$$\lambda_i = K \left[\frac{N(\infty) - (i - 1)}{I} \right],$$

where $N(\infty)$ is the initial number of faults in the program (constant), i is the number of faults detected and corrected, I is the number of machine-level instructions in the program (constant), and K is a proportionality constant.

Bayesian Updating of Failure Intensity

Regardless of whether the failure intensity estimate for a program version is obtained from a software reliability model or some other means, once the program version goes into operational use, empirical data will become available. A chronological log of software failures can be kept, and this information can be used to update the best estimate of the actual failure intensity. (Remember, program version = frozen code.) The uncertainty about the actual value of the underlying failure rate λ can be reflected by employing an assumed gamma prior distribution. Figure 21 shows the shapes of two Bayesian prior distributions for the failure rate λ . These two gamma distributions differ only in their standard deviations, reflecting the amount of uncertainty present.

$$\Pr\{\mathbf{A} = \lambda\} = \frac{\beta}{\Gamma(\alpha)} \exp(-\beta\lambda)(\beta\lambda)^{\alpha-1} d\lambda, \quad 0 \leq \lambda < \infty.$$

This distribution is convenient because it ranges over the entire positive axis $(0, \infty]$. The mean α/β can be set to the latest best estimate. The degree of uncertainty is reflected by the variance α/β^2 . Just one gamma distribution has a preassigned mean and variance. The smaller the variance, the greater the s-reliability of the estimate. The distribution of the number of software failures will follow the Poisson law with mean λt :

$$\begin{aligned} \Pr\{\mathbf{N}(t) = n | \lambda < \mathbf{A} \leq \lambda + d\lambda\} \\ = \frac{\exp(-\lambda t)(\lambda t)^n}{n!} \end{aligned}$$

By Bayes' rule,

$$\begin{aligned} \Pr\{\lambda \leq \mathbf{A} \leq \lambda + d\lambda | \mathbf{N}(t) = n\} \\ = \frac{\Pr\{\lambda < \mathbf{A} \leq \lambda + d\lambda\} \Pr\{\mathbf{N}(t) = n | \lambda \leq \mathbf{A} \leq \lambda + d\lambda\}}{\int_{\lambda=0}^{\infty} \Pr\{\lambda < \mathbf{A} \leq \lambda + d\lambda\} \Pr\{\mathbf{N}(t) = n | \lambda < \mathbf{A} \leq \lambda + d\lambda\}}, \end{aligned}$$

obtaining

$$\Pr\{\lambda < \mathbf{A} \leq \lambda + d\lambda\} = \frac{B}{\Gamma(A)} \exp(-B\lambda)(B\lambda)^{A-1} d\lambda, \quad 0 \leq \lambda < \infty$$

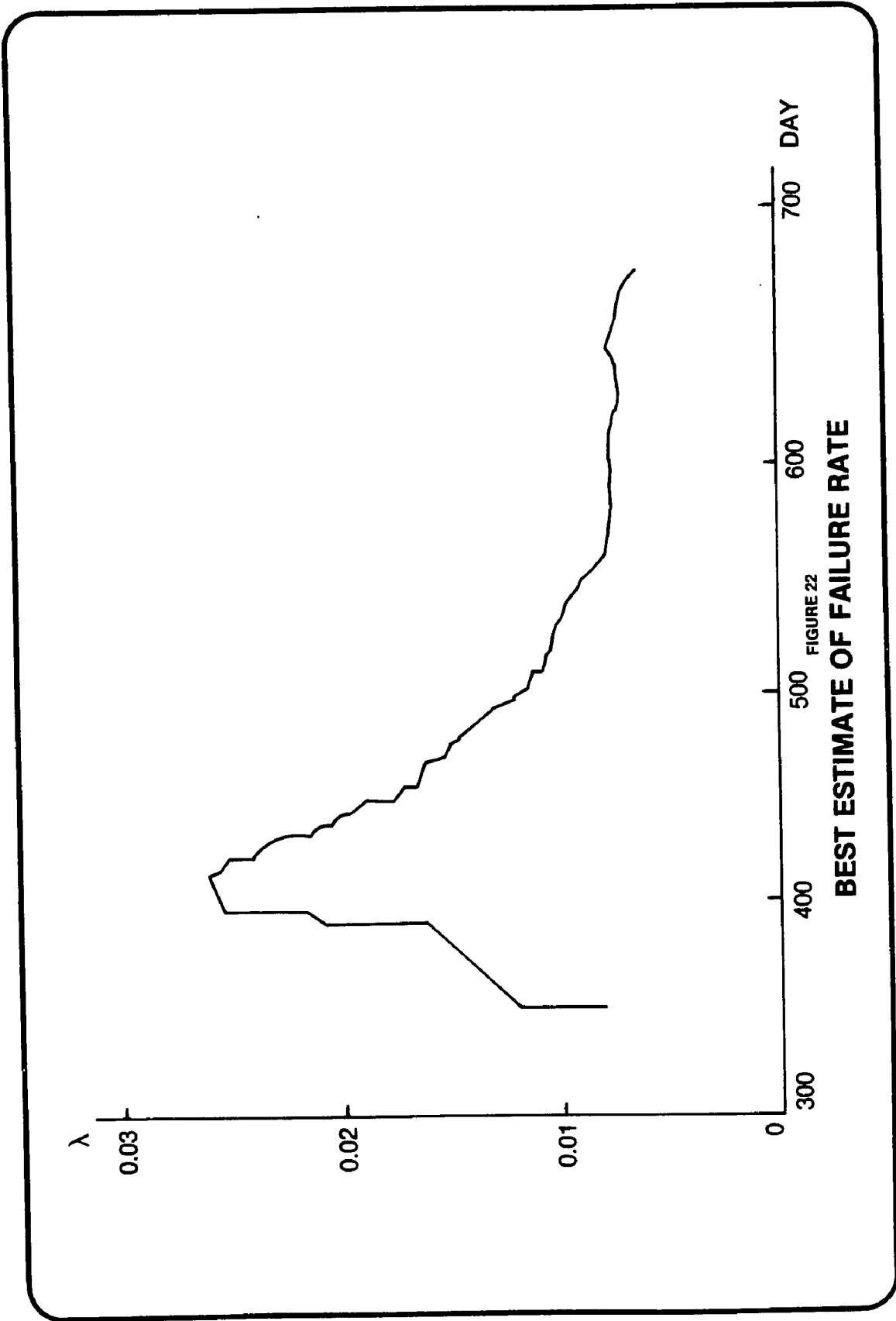


FIGURE 22

BEST ESTIMATE OF FAILURE RATE

where

$$A = \alpha + n$$

and

$$B = \beta + t.$$

All this ink has led to a simple recurrence: The best estimate of the underlying failure rate λ is updated after each failure from α/β to $(\alpha + n)/(\beta + t)$, with the distribution remaining gamma.

Example

The purpose of this example is to demonstrate the Bayesian failure-rate updating technique using data from a real project. The system from which the Appendix A data[RADA81] was gathered consisted of a pair of real-time interrupt-driven command and control (C²) programs totaling 24,000 lines of assembly source code, and a time critical mathematical program consisting of 39,000 FORTRAN source lines and 2,000 assembly-language source lines. Development spanned a 3-1/2-year period. The development practices included top-down program design, use of a modified programmer team concept, and use of a program support library.

The major software tools used for IV& V were an interpretive computer simulator (ICS) that emulated the on-board flight computer at the machine-instruction level, and a six-degrees-of-freedom environment simulator that modeled mechanical, aerodynamic, gravitational, and other physical effects.

Some 249 failures occurred, each resulting in an anomaly report. Severities were distributed as 21% high ("3"), 48% medium ("2"), and 30% low ("1"). Figure 22 shows a plot of the running best estimate of the failure rate. The original estimate was $\lambda = 1.75$, and the latest revised best estimate turned out to be about $\lambda = 1.6$. The recurrence rule is: Let

$$\lambda_i = \frac{\alpha_i}{\beta_i},$$

then

$$\lambda_{i+1} = \frac{\alpha_i + 1}{\beta_i + t_{i+1}},$$

where t_i is the i th interfailure time.

Failure-Counting Component: Variants

The basic failure-counting component assumes the failure rate λ to be constant for a single program version. The formulation of the failure can be conveniently borrowed from homogeneous-Poisson-process-based models such as the Jelinsky-Moranda, Shooman, Schick-Wolverton and Musa models. The failure-counting component can also accommodate a failure rate borrowed from other major time-domain models. For the Littlewood models, a gamma-distributed λ is used. For the Goel-Okumoto NHPP, Schick-Wolverton and Wagner models, a time-dependent failure rate function $\lambda(\tau)$ is used.

Mixing via Gamma Structure Function

In the Bayesian updating technique, the failure rate parameter λ had a fixed but unknown value. Quite another idea is to let λ itself be a gamma-distributed random variable with pdf

$$f(\lambda) = \begin{cases} \frac{\beta^\alpha}{\Gamma(\alpha)} \lambda^{\alpha-1} \exp(-\beta\lambda), & \lambda > 0, \alpha > 0, \beta > 0 \\ 0, & \lambda \leq 0. \end{cases}$$

Starting out with

$$p_n(t) = \int_0^\infty p_{n|\lambda}(t) f(\lambda) d\lambda$$

we have

$$\begin{aligned} p_n(t) &= \int_0^\infty \frac{\exp(-\lambda t) (\lambda t)^n}{n!} f(\lambda) d\lambda \\ &= \frac{\beta^\alpha t^n}{\Gamma(\alpha) n!} \int_0^\infty \lambda^{n+\alpha-1} \exp[-\lambda(\beta + t)] d\lambda \\ &= \frac{\Gamma(n + \alpha)}{n! \Gamma(\alpha)} \beta^\alpha t^n (\beta + t)^{-(n+\alpha)}, \quad n = 0, 1, \dots \end{aligned}$$

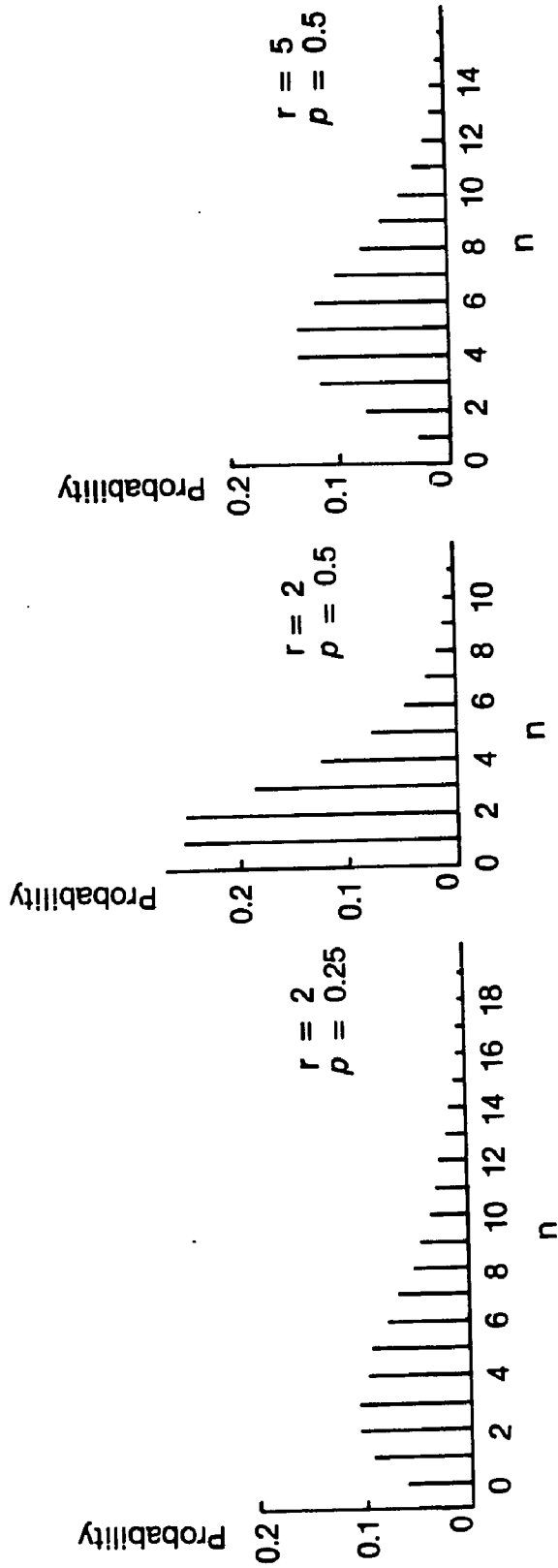


FIGURE 23

NEGATIVE BINOMIAL pmf

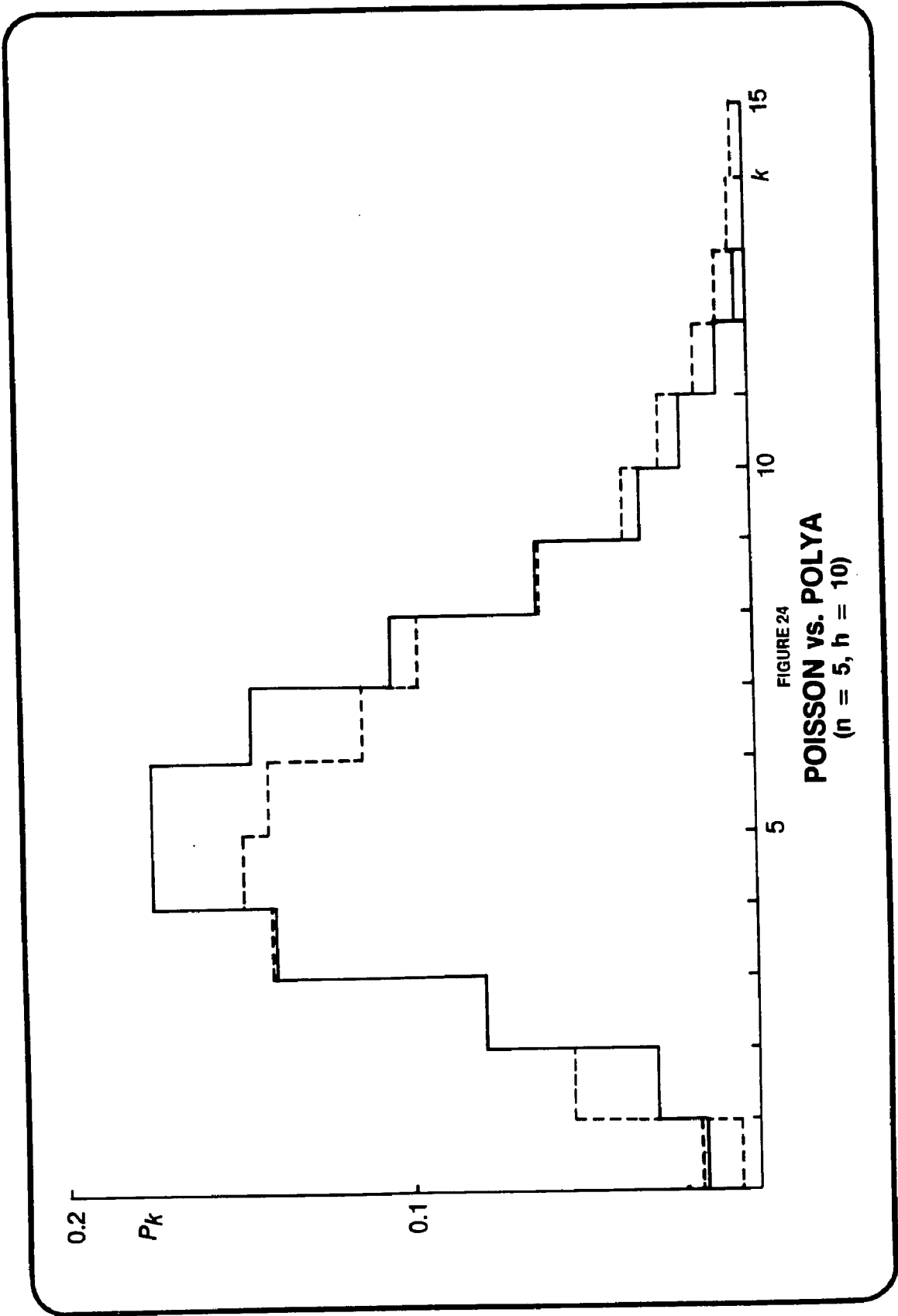


FIGURE 24
POISSON vs. POLYA
($n = 5, h = 10$)

Using the recurrence relation $\Gamma(\alpha + 1) = \alpha\Gamma(\alpha)$ and the expansion

$$\binom{x}{n} = \frac{(x-1) \cdots (x-n+1)}{n!},$$

the pmf can be rewritten

$$p_n(t) = \binom{n+\alpha-1}{n} \left(\frac{t}{\beta+t}\right)^n \left(\frac{\beta}{\beta+t}\right)^\alpha.$$

This turns out to be for each t a negative binomial distribution with parameters

$$h = \alpha, p = \frac{\beta}{\beta+t}$$

The mean of this distribution is

$$\mu = \mathbf{E}\{\mathbf{N}(t)\} = t \frac{\alpha}{\beta}$$

and the variance is

$$\sigma^2 = \mathbf{Var}\{\mathbf{N}(t)\} = t \frac{\alpha}{\beta} \left(1 + \frac{t}{\beta}\right).$$

This new process would come under the classification of a Pólya process and is distinguished from the Poisson process in that its variance exceeds its mean. Recall that for a Poisson process the mean and variance are equal. Figure 23 shows the shapes of typical negative binomial distributions. Figure 24 shows a comparison between the realizations of a Poisson process and a Pólya process. The form of the latter distribution is

$$p_k(n) = \binom{n+k-1}{k} \left(\frac{n}{n+h}\right)^n \left(\frac{h}{n+h}\right)^k,$$

which reduces to the Poisson formula when $n \rightarrow \infty$.

This variation conforms to the Littewood models.

Temporal Nonhomogeneity

Another variant is to assume as in the Goel-Okumoto NHPP model that $\lambda(\tau)$ is a bounded, measurable (nonrandom) function of time, with the probability of

software failure in the “small” time interval $(\tau, \tau + \Delta\tau)$ being $\lambda(\tau) d\tau + o(\Delta\tau)$. This results in

$$p_n(t) = \frac{\int_0^\tau \lambda(u) du}{n!} \exp \left[- \int_0^\tau \lambda(u) du \right], \quad n = 0, 1, 2, \dots$$

By the time scale transformation

$$\tau' = \int_0^\tau \lambda(u) du,$$

this distribution is reduced to the simple case

$$p_n(\tau) = (n!)^{-1} \tau'^n \exp(-\tau), \quad n = 0, 1, 2, \dots,$$

where the lengths of time intervals are measured by the s -expected numbers of software failures in those intervals.

This variation conforms to the Goel-Okumoto NHPP, Schick-Wolverton and Wagner models.

Arbitrary Failure Rate Distribution

A very general case is where λ is a random variable but is arbitrarily distributed. When λ has an arbitrary distribution, the probability of observing n software failures is the integral

$$p_n(t) = \frac{1}{n!} \int_0^\infty \exp(-\lambda t) (\lambda t)^n f_{t|\lambda}(\lambda) d\lambda.$$

Note that the pdf for this “weighted” Poisson process must be reinterpreted as a conditional distribution. Interesting applications of this type of distribution can be found in [BATE52], [LUND40], [CHIANG66].

This generalization is used for all other current and future models in which λ is considered a random variable. Unlike the specific case in which λ is gamma-distributed, though, a closed-form analytical solution of the integral might not exist, and numerical quadrature would have to be resorted to.

Failure Counting Process Estimation

In addition to a probabilistic model, a complete *prediction system*[LIT85] also requires a statistical inference procedure for estimating the unknown parameters in the model, and a prediction procedure. Poor performance of some of the early software reliability models might be improved upon through use of more sophisticated parameter estimation methods[FORM77].

An *estimator* is a function of a sample that furnishes an estimate of a distributional parameter. The principal of *maximum likelihood*, which is employed in all the examples in this dissertation, provides a powerful method for obtaining estimators. Under general conditions these estimators are known to exhibit very desirable properties. The method selects those parameter values that result in a probability density/mass function that is most likely to have produced the observed data.

Let the observed failure times be characterized by the random variables $\mathbf{T}_1, \mathbf{T}_2, \dots, \mathbf{T}_n$. Let the actual values be $\tau_1, \tau_2, \dots, \tau_n$. The probability that $\mathbf{T}_i = \tau_i$ is dependent on the value of a parameter θ and so its probability mass function will be written $f(\tau_i; \theta)$. Because the \mathbf{T}_i 's are mutually *s*-independent, their joint pmf can be written

$$L(\tau_1, \tau_2, \dots, \tau_n; \theta) = \prod_{i=1}^n f(\tau_i; \theta).$$

The function L is called the *likelihood function*. To find the value of θ for which L is maximized, the partial derivative $\partial L / \partial \theta$ is taken and set to zero. Often at this point the logarithm of L is first taken to simplify the differentiation. The root of the equation then found by means of an iterative computational procedure, or graphically.

That value of θ for which L (or $\log L$) attains its highest value, is called the *maximum likelihood estimator* (MLE) for the parameter θ .

For instance, the Jelinsky-Moranda failure rate formulation, as introduced in the Survey of Related Research, looks like this:

$$\lambda(t, \lambda) = KN(\infty) \exp(-K\tau).$$

The two unknown parameters are the inherent number of errors $N(\infty)$ and the proportionality constant K . When the principle of maximum likelihood is applied to the parameters in that failure rate formulation, the following equation [JELI71] is obtained for finding an estimate $\hat{N}(\infty)$ of the initial fault content $N(\infty)$:

$$\sum_{i=1}^n \frac{1}{\hat{N}(\infty) - (i-1)} = \frac{n \sum_{i=1}^n \tau_i}{\hat{N}(\infty) \sum_{i=1}^n \tau_i - \sum_{i=1}^n (i-1)\tau_i}$$

The maximum likelihood estimator \hat{K} for proportionality constant K is then obtained as

$$\hat{K} = \frac{n}{\hat{N}(\infty) \sum_{i=1}^n \tau_i - \sum_{i=1}^n (i-1)\tau_i}$$

The failure rate λ_i can now be estimated as

$$\hat{\lambda}_i = \hat{K}[\hat{N}(\infty) - (i-1)].$$

The estimated MTTF is $1/\hat{\lambda}_i$.

CHAPTER 5

INCORPORATING PENALTY COSTS

Now that the failure-counting component of the model is complete, attention will now turn to the penalty cost component and how it will be incorporated into the model. Both the failure-counting component and penalty-cost component concern themselves with uncertainty: For the former, the uncertainty is regarding the frequency and occurrence-times of software failure; for the latter the uncertainty is regarding the severity of individual software failures.

Penalty Cost Component

Penalty cost is a quantitative measure of the severity of loss resulting from a software failure. For the purposes of mathematical modeling, penalty cost can be treated simply as a number. The loss can generally be decomposed into several possible categories[BROW80]:

- 1.) Damage to, destruction of property;
- 2.) Injury, fatality of persons;
- 3.) Community, environmental damage;
- 4.) Financial effects (e.g., legal liability, lost revenue);
- 5.) Psychological effects;
- 6.) Resources, effort to isolate and correct the fault.

It may seem distasteful or even immoral to put a value on human life and suffering (as for example is done in ANSI Standard Z16.1[AMER67]). If some of these categories are deemed inconvertible to a common measurement, the penalty cost can be recorded as a finite-dimensional random vector rather than a scalar,

as will be shown. Efforts have been made to produce a taxonomical classification scheme and rating scale for the effects of software failures[DANA74].

The assumption will be made that future penalty costs will be described by the same random variable as past penalty costs. That is, causal influences in the future will be the same as those in the past: The random sequence is assumed *stationary*, which means that samples from a sequence of sufficiently long duration are statistically representative of the whole.

Let the random variable \mathbf{X}_j be the penalty cost associated with the software failure occurring at time $\mathbf{T}_{(j)}$. The penalty costs arising from the software failures at times $\mathbf{T}_{(1)}, \mathbf{T}_{(2)}, \dots$ form a random sequence

$$\{\mathbf{X}_i, i = 1, 2, \dots\}$$

wherein $\mathbf{X}_1, \mathbf{X}_2, \dots$ are *s*-independent and identically distributed nonnegative random variables. The \mathbf{X}_j 's are not all zero with probability one and follow the common cumulative distribution function

$$S(x) \equiv \text{Cdf}\{\mathbf{X}\}.$$

The existence of $S(x)$ is axiomatic and is supposed *s*-independent of the time the failure occurs and any information about previous failures. Thus $\mathbf{N}(t)$ and the penalty-cost sequence $\{\mathbf{X}_i\}$ are *s*-independent of each other. For concreteness, suppose that $S(x)$ is such that for some k its k th convolution has an absolutely continuous component (that is, at least one moment exists) and that $E\{\mathbf{X}_i\} > 0$.

The outcome in a particular case is a sequence of ordered pairs of software failure time epochs and associated penalty costs. Each realization $\omega = \{(t_{(j)}, x_j), j = 1, 2, \dots\}$ represents the actual development of the failure history in a particular case. The *aggregate penalty cost* process $\{\mathbf{Z}(t), 0 \leq t < \infty\}$, totals the sum of the penalty costs incurred over the interval $(0, t)$:

$$\mathbf{Z}(t) = \mathbf{X}_1 + \mathbf{X}_2 + \dots + \mathbf{X}_{\mathbf{N}(t)}, \quad t \geq 0.$$

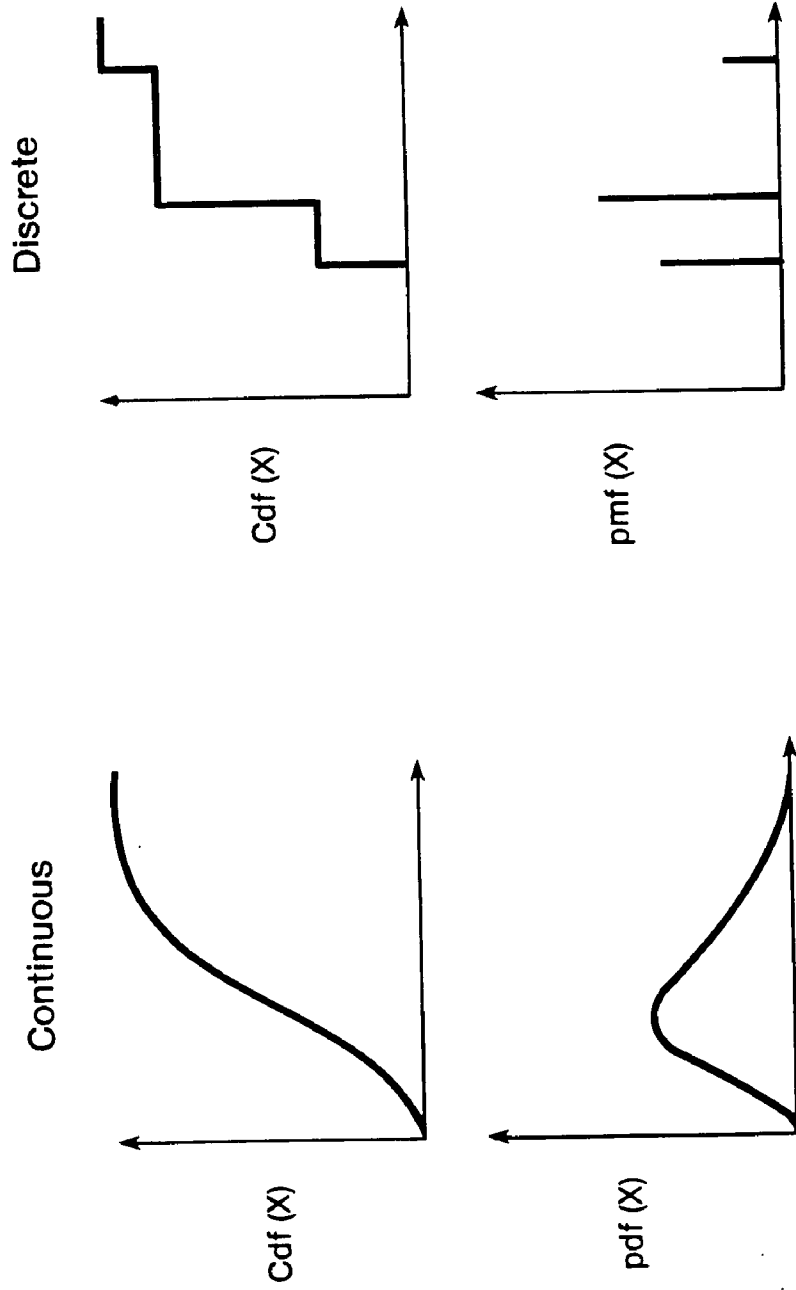


FIGURE 25
FORMS OF PENALTY COST DISTRIBUTION X

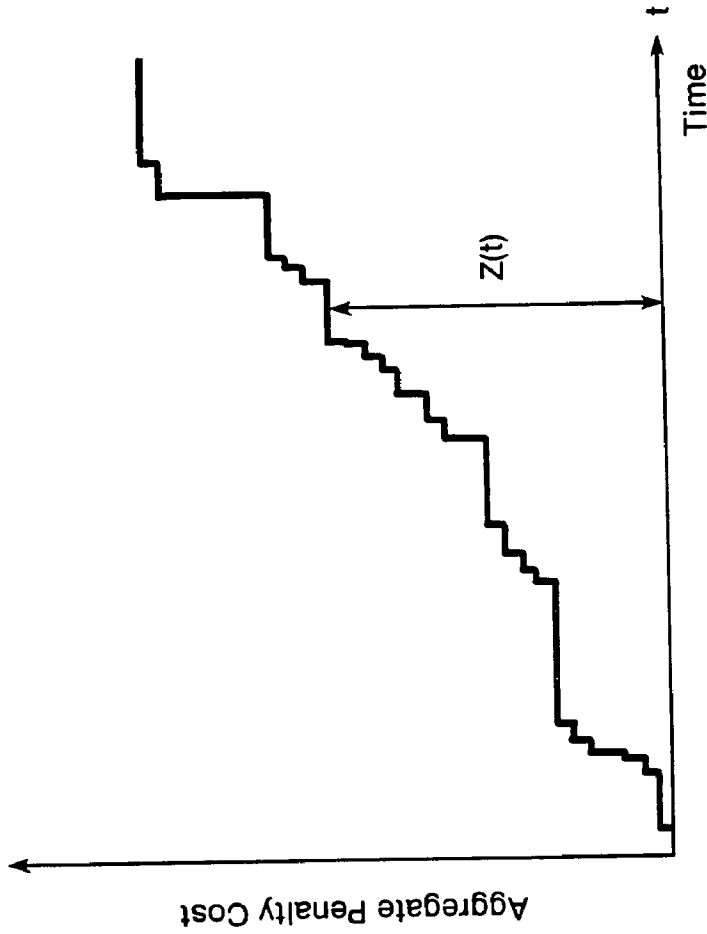


FIGURE 26

A REALIZATION OF AGGREGATE PENALTY COST PROCESS

In words, $\mathbf{Z}(t)$ is a random sum of random variables. The process will be associated with well-defined probabilities induced by the measures of the variables ω . The restricted functional space of $\mathbf{Z}(t)$ consists of step-functions with discontinuity points $t_{(j)}$. The process $\mathbf{Z}(t)$ describes the evolution of cumulative failure effects, recognizing both the random number of failures $\mathbf{N}(t)$ in time and the random nature of the penalty costs.

Penalty Cost Distributions

The penalty cost distribution can be either discrete or continuous, or even a mixture of both (see Figure 25). For penalty costs values based on low-resolution severity ratings, use of an *empirical distribution*[HAYS84] suggests itself. An empirical distribution simply assigns a probability mass of $1/n$ to each of the n sample points. A problem: If the scale is open-ended then the right tail of the distribution is not modeled. For example if the greatest severity so far observed is 9, we have no way of assigning a probability to severities of 10, 11, 12 and so on. Furthermore, even if the scale is closed-ended, for example 1–15, no natural method of interpolation between data points exists: If there were a lot of 4's and a lot of 6's but no 5's, it is not clear what probability should be assigned to 5 (zero seems a bit harsh).

Use of continuous theoretical distributions such as normal, negative exponential, gamma and Pareto can avoid these problems and do offer the advantage of well-developed statistical inference techniques.

Distribution of Aggregate Penalty Costs

Figure 26 shows the realization of a typical aggregate penalty cost process $\mathbf{Z}(t)$. Two approaches will be considered: deterministic and stochastic. The deterministic approach is rather simplistic, while the stochastic approach is much more comprehensive.

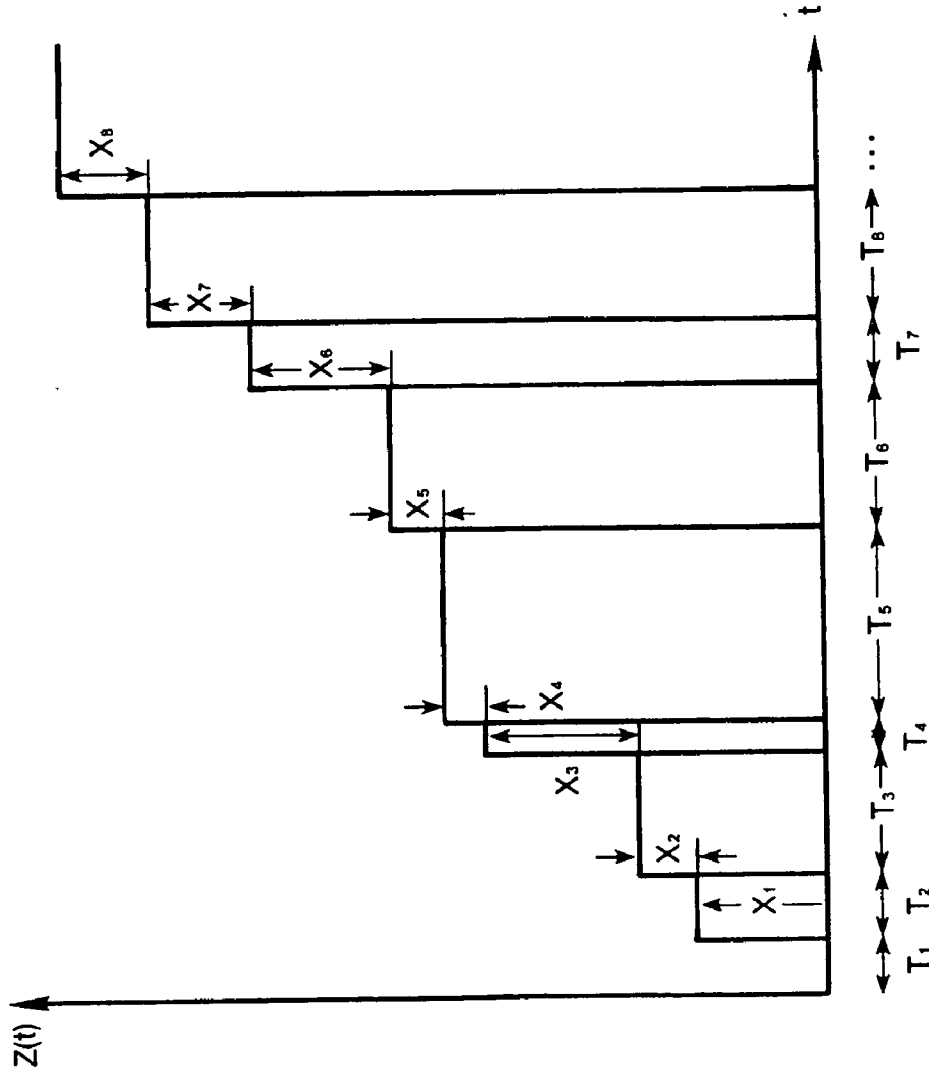


FIGURE 27

THE AGGREGATE PENALTY COST PROCESS $Z(t)$

Deterministic Approach

The frequency of software failure can be summarized by

$$\text{Average Frequency} = \frac{\text{Total number of software failures}}{\text{Observed time exposure}},$$

and the severity by

$$\text{Average Severity} = \frac{\text{Aggregate penalty cost}}{\text{Total number of software failures}}.$$

Combining the two averages yields

$$\text{Avg. Penalty Cost} = \text{Avg. Frequency} \times \text{Avg. Severity} \times \text{Future Time Exposure}.$$

Or, in symbols,

$$E\{\mathbf{Z}(t)\} = E\left\{\frac{\mathbf{N}(\tau)}{\tau}\right\} \cdot E\{\mathbf{X}\} \cdot t,$$

where τ is the cumulative debugging time and t is future operational time. This average aggregate penalty cost can be considered the “deterministic” solution to the problem, in the sense that it yields a single answer. However, the average aggregate penalty cost is in reality only a mean about which the actual aggregate penalty cost figure will vary. The actual figure could range all the way from zero to some very high number. There could be, for example, two aggregate penalty cost distributions, both of which have the same average, yet one situation may be “riskier” because the actual values would tend to be more widely dispersed about that average value. (There is an old story about a statistician who drowned in water of average depth three feet. . . .) So the average value, while informative, by far does not tell the whole story: It is necessary to work with the full distribution. From the distribution a number of other features of the random variable can be computed to supplement the average, such as variance, skewness, kurtosis, higher moments, percentiles, etc. An additional risk also exists: Due to sampling error, the average value may not equal the true mean of the penalty cost distribution.

Stochastic Approach

The deterministic approach replaced random variables by their average values. This machination abstracted out the stochasticity. Now we will take the more comprehensive approach of deriving the actual distribution of the aggregate penalty cost. It is desired to find the probability distribution of the aggregate penalty cost. The two layers of stochastic variation—failure occurrence and penalty cost—constitute a compound stochastic process. The Cdf of the aggregate penalty cost will be denoted $F(z, t)$. Figure 27 illustrates the aggregate penalty cost process $\mathbf{Z}(t)$. The Cdf can be obtained by exhaustively enumerating all possibilities leading to the occurrence of the event $\{\mathbf{Z}(t) \leq z\}$. The event can occur in the following mutually exclusive ways (in the manner of, e.g., [CRAM54]):

- (i) In the time interval no software failure occurs;
- (ii) 1 software failure occurs and its penalty cost is $\leq z$;
- (iii) 2 software failures occur and the total penalty cost is $\leq z$;
- (iv) 3 software failures occur and the total penalty cost is $\leq z$;

and so on.

Let $S_n(z)$ be the conditional aggregate penalty cost Cdf given that n software failures have occurred:

$$S_n(z) = \text{Cdf}\{\mathbf{Z}(t) | \mathbf{N}(t) = n\}.$$

From the multiplication and addition rules of probability, it follows that if

$$E\{\mathbf{Z}(t)\} < \infty,$$

then

$$F(z, t) = \sum_{n=0}^{\infty} \text{Pr}\{\mathbf{N}(t) = n\} S_n(z).$$

According to probability calculus, the Cdf of the sum of mutually s -independent random variables is the convolution of their respective Cdf's. $S_n(z)$ is thus the n th convolution— n being a fixed number—of $S(z)$ with itself:

$$S_n(z) = \int_0^z S_{n-1}(z-w) dS(w) = S^{(n-1)*} * S(z) = S^{n*}(z), \quad n > 1,$$

and the formula for the Cdf of the aggregate penalty cost, after conditioning on the values of $\mathbf{N}(t)$, becomes

$$F(z, t) = \sum_{n=0}^{\infty} \Pr\{\mathbf{N}(t) = n\} S^{n*}(z).$$

The basis of the recurrence is

$$S^{0*}(z) = \begin{cases} 0, & z < 0 \\ 1, & z \geq 0. \end{cases}$$

Also, $S^{1*}(z) = S(z)$.

Distributional Properties

The s -expected value of the aggregate penalty cost $\mathbf{Z}(t)$ for the time interval $(0, t)$ will be

$$\mathbf{E}\{\mathbf{Z}(t)\} = \mathbf{E}\{\mathbf{N}(t)\}\mathbf{E}\{\mathbf{X}\},$$

because of the s -independence of the failure occurrence and penalty costs. The variance is easily shown to be

$$\text{Var}\{\mathbf{Z}(t)\} = \mathbf{E}\{\mathbf{N}(t)\}\text{Var}\{\mathbf{X}\} + \text{Var}\{\mathbf{N}(t)\}[\mathbf{E}\{\mathbf{X}\}]^2.$$

Now we can combine a specifically distributed failure-counting component with the penalty-cost component.

Homogeneous Poisson Failure-Counting Component

The development of the failure-counting component in the previous chapter emphasized the temporally homogeneous Poisson process. In this case, the aggregate penalty cost distribution

$$F(z, t) = \sum_{n=0}^{\infty} \Pr\{\mathbf{N}(t) = n\} S^{n*}(z)$$

will be

$$F(z, t) = \sum_{n=0}^{\infty} \exp(-\lambda t) \frac{(\lambda t)^n}{n!} S^{n*}(z).$$

From the previous results it follows that the mean of this distribution is

$$E\{\mathbf{Z}(t)\} = \lambda t E\{\mathbf{X}\}.$$

The variance is

$$\text{Var}\{\mathbf{Z}(t)\} = \lambda t (\text{Var}\{\mathbf{X}\} + [E\{\mathbf{X}\}]^2) = \lambda t E\{\mathbf{X}^2\}$$

because of the identity $\text{Var}\{\mathbf{X}\} \equiv E\{\mathbf{X}^2\} - [E\{\mathbf{X}\}]^2$ [MEND81].

Derivation by Severity Class

When penalty cost is measured in discrete units, the aggregate penalty-cost distribution admits of an alternate derivation. Instead of software failures occurring randomly in time, suppose that “penalty cost units” occur randomly in time. Recall that the probability of more than one software failure event during the time interval $(0, t)$ was assumed to be $o(\Delta t)$. However, we will make no corresponding assumption for penalty cost units. Let

$$\begin{aligned} & \Pr\{i\text{-unit aggregate penalty cost in } (t, t + \Delta t)\} \\ &= \lambda_i \Delta t + o(\Delta t), \quad i = 1, 2, \dots, m \end{aligned}$$

with

$$\sum_{i=1}^m \lambda_i = \lambda.$$

A software failure is considered to have occurred whenever one or more simultaneous penalty cost units appear. The number of simultaneous penalty cost units at that moment is the penalty cost of that software failure. Each individual stream of occurrence of the same penalty cost batch size i forms its own temporally homogeneous Poisson process with failure rate parameter λ_i . The aggregate penalty process is now

$$\mathbf{Z}(t) = \sum_{i=1}^m i \mathbf{N}_i(t),$$

with probability mass function

$$\begin{aligned} p_r(t) &= \Pr\{r \text{ penalty cost units in } (0, t)\} \\ &= \sum_{i=0}^r \exp(-\lambda t) \frac{(\lambda t)^i}{i!} c_r^{(i)} \end{aligned}$$

where $c_r^{(i)} = \Pr\{i \text{ occurrences give grand total of } r\}$, the probability associated with the i -fold convolution of the batch-size probabilities $\{\lambda_i/\lambda\}$. This is the aggregate penalty cost desired.

Thus for discrete penalty costs, the aggregate penalty cost process can be viewed either as a single stream with a varying batch size (the original derivation), or as the sum of multiple constant-batch-size streams; both produce the same aggregate penalty cost distribution.

Pólya Variation

In the case where the failure rate is gamma-distributed, the incidence of software failures was shown to be negative binomially distributed. The distribution of aggregate penalty cost in this variation will be

$$F(z, t) = \sum_{k=0}^{\infty} \binom{k + \alpha - 1}{k} \left(\frac{t}{\beta + t}\right)^k \left(\frac{\beta}{\beta + t}\right)^{\alpha} S^{k*}(z).$$

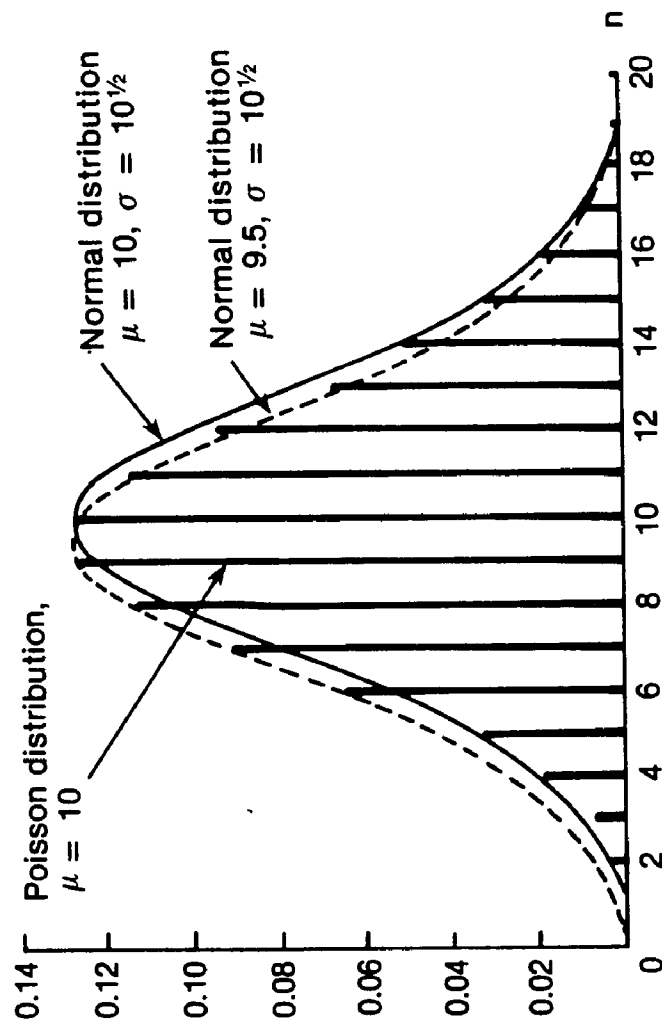


FIGURE 28

NORMAL APPROXIMATION TO POISSON

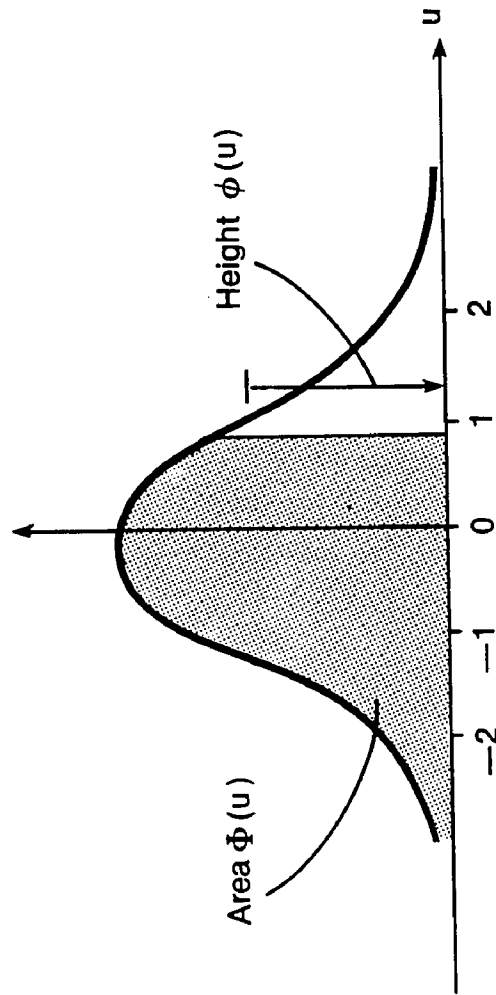


FIGURE 29
STANDARD NORMAL APPROXIMATION

The mean will be

$$E\{\mathbf{Z}(t)\} = t \frac{\alpha}{\beta} E\{\mathbf{X}\}.$$

The variance will be

$$\text{Var}\{\mathbf{Z}(t)\} = t \frac{\alpha}{\beta} E\{\mathbf{X}^2\} + \frac{\left(t \frac{\alpha}{\beta}\right)^2 (E\{X\})^2}{\beta}.$$

The Pólya variation is the one that conforms to the Littlewood models' failure rate formulations.

Some Examples

Simple Poisson/Unit Penalty Costs

Suppose that $\mathbf{N}(t)$ has a Poisson distribution with time unit chosen such that $\lambda = 1$. As a very simple example, the case in which the penalty cost for each software failure is 1, that is,

$$S(x) = \begin{cases} 0, & x < 1 \\ 1, & x \geq 1. \end{cases}$$

will be worked out. In this case

$$S^{2*}(z) = \int_0^z S^{1*}(z-t) dS(t) = S^{1*}(z-1) = \begin{cases} 0, & z < 2 \\ 1, & z \geq 2. \end{cases}$$

and then for $n = 3, 4, \dots$,

$$S^{n*}(z) = \begin{cases} 0, & z < n \\ 1, & z \geq n. \end{cases}$$

$$F(z, t) = \sum_{k=0}^{\infty} \frac{\exp(-t)t^k}{k!} = \sum_{n=0}^{\lfloor z \rfloor} \frac{\exp(-t)t^n}{n!},$$

which will be recognized as the Cdf of the failure-counting process $\mathbf{N}(t)$. The model has been reduced to that of its failure-counting component. If λt is reasonably large (> 10), the normal approximation

$$\Pr\{\mathbf{N}(t) \leq n\} \simeq \Phi[(n - \lambda t)/\sqrt{\lambda t}]$$

can be used. The symbol “ Φ ” stands for the Cdf of the standard normal distribution, a limiting distribution discussed in more detail in the next chapter. Figure 28 illustrates the normal approximation to the Poisson distribution.

Figure lastfigure shows the Cdf and pdf of the standard normal distribution.

Simple Poisson/Simple Exponential Penalty Costs

Suppose that the penalty costs are distributed according to a negative exponential distribution with penalty cost unit chosen such that the intensity 1. That is,

$$S(x) = \begin{cases} 0, & x < 0 \\ 1 - \exp(-x), & x \geq 0. \end{cases}$$

Assume that the interfailure time is chosen such that the failure rate $\lambda = 1$. Then

$$\begin{aligned} S^{2*}(z) &= \int_0^\infty S^{1*}(z-t)S(t) dt = \int_0^z \{1 - \exp[-(x-t)]\} \exp(-t) dt \\ &= 1 - \sum_{n=0}^1 \frac{\exp(-x)x^n}{n!} = \int_0^x \frac{\exp(-t)t}{1!} dt. \end{aligned}$$

By the principle of mathematical induction,

$$S^{n*}(z) = \int_0^z \frac{\exp(-t)t^{n-1}}{(n-1)!} dt,$$

which is the standard incomplete gamma function $\Gamma(z, n)$ [BEEK84]. The value can be approximated by the formula[KHAM65]

$$\Gamma(z, n) = \frac{z^n \exp(-z)}{\Gamma(n)} \sum_{j=0}^{\infty} \frac{z^j}{n(n+1) \cdots (n+j)}.$$

(When $z \geq 280$, the asymptotic complete gamma function $\Gamma(n)$, met before, can be substituted.)

“Robust” Programs

Kopetz has defined[KOPE79] a robust program as one in which the penalty cost C of a software failure is inversely related to its probability of occurrence p :

$$C \sim \frac{1}{p}$$

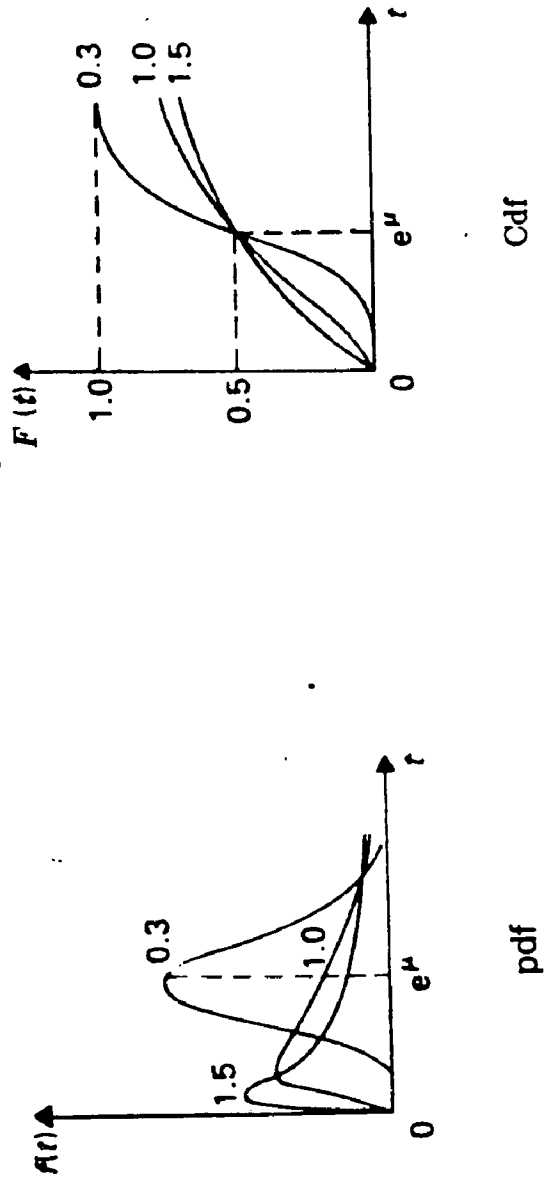


FIGURE 30
LOGNORMAL DISTRIBUTION

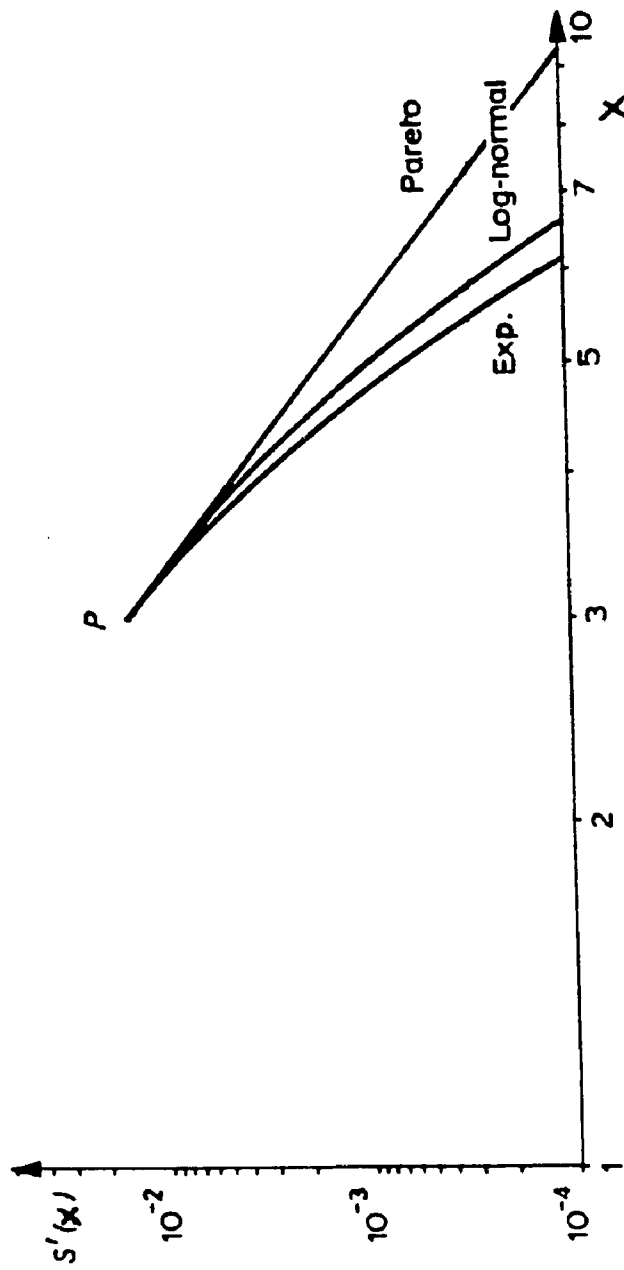


FIGURE 31
COMPARISON OF EXPONENTIAL, LOGNORMAL,
PARETO DENSITIES(DOUBLE LOG SCALE)

The simplest type of probability distribution that provides this property is the negative exponential

$$S(z) = 1 - \exp(-cz), \quad z \geq 0.$$

This results in a recurrence relation for calculating $S^{k*}(z)$:

$$\begin{aligned} S^{k*}(z) &= 1 - \exp(-cz) \sum_{i=0}^{k-1} \frac{(cz)^i}{i!} \\ &= S^{(k-1)*}(z) - \exp(-cz)(cz)^{k-1}/(k-1)! \end{aligned}$$

An even more compact form is again in terms of the standard incomplete gamma function:

$$S^{k*}(z) = \Gamma(cz, k).$$

The main objection to the use of the negative exponential distribution is that its right tail vanishes rapidly. This can be remedied through use of heavier-tailed distributions such as the Pareto (which converges very slowly) and the lognormal (in between negative exponential and Pareto). Figure 30 shows the shape of the lognormal distribution with standard pdf

$$f(x) = \frac{1}{t\sigma\sqrt{2\pi}} \exp\left[-\frac{(\ln t - \mu)^2}{2\sigma^2}\right], \quad t \geq 0$$

The random variable \mathbf{X} has a lognormal distribution if $\mathbf{Y} = \ln \mathbf{X}$ has a normal distribution. In the pdf, μ and σ are the parameters (mean and standard deviation) of $\ln \mathbf{X}$. The exponential and lognormal distributions would tend to underestimate the risks of excessive penalty costs, while the Pareto[BEAR84] models “dangerous” systems. Figure 31, after[BEAR84], shows a comparison of the negative exponential, lognormal and Pareto probability densities on a double logarithmic scale.

Example: Parameter Estimation for Robust Program

Here is an example of parameter estimation from a hypothetical robust program. In this example the failure times are due to Musa[MUSA79B], and the penalty

costs are negative exponential random deviates. The required parameters $N(\infty)$ and K will be estimated using the technique of maximum likelihood, described in the previous chapter. The key equation for calculating the estimate $\hat{N}(\infty)$ is

$$\frac{n \sum_{i=1}^n \tau_i}{\hat{N}(\infty) \sum_{i=1}^n \tau_i - \sum_{i=1}^n (i-1)\tau_i} = \sum_{i=1}^n \frac{1}{\hat{N}(\infty) - (i-1)}.$$

From the failure data detailed in Appendix B,

$$\sum_{i=1}^{136} \tau_i = 88,683$$

and

$$\sum_{i=1}^{136} (i-1)\tau_i = 8,694,899$$

yielding

$$\frac{(136)(88,683)}{\hat{N}(\infty)(88,683) - 8,694,899} = \sum_{i=1}^{136} \frac{1}{\hat{N}(\infty) - (i-1)}$$

or

$$\begin{aligned} & \frac{12,060,888}{88,683\hat{N}(\infty) - 8,694,899} \\ &= \frac{1}{\hat{N}(\infty)} + \frac{1}{\hat{N}(\infty) - 1} + \cdots + \frac{1}{\hat{N}(\infty) - 136}. \end{aligned}$$

This equation was solved on a computer and found to be satisfied for $\hat{N}(\infty) = 142$.

For this value, the left-hand side is 3.094053 and the right-hand side is 3.086560.

The maximum likelihood estimator of K is

$$\hat{K} = \frac{n}{\hat{N}(\infty) \sum_{i=1}^n \tau_i - \sum_{i=1}^n (i-1)\tau_i}.$$

Substituting the numbers gives

$$\frac{136}{142(88,683) - 8694899} = 0.000035.$$

The failure rate is

$$\hat{\lambda} = [\hat{N}(\infty) - (n-1)]\hat{K}$$

$$\begin{aligned}
&= [142 - (136 - 1)]0.000035 \\
&= 0.000245.
\end{aligned}$$

The required parameter λ for the failure-counting component has now been estimated. Now we turn to the penalty cost component. The distribution in that component is

$$S(x) = 1 - \exp(-cx).$$

The unknown parameter is c . Since this is a negative exponential distribution, c is the intensity—the reciprocal of the mean. For the penalty cost, the mean can be estimated by the arithmetic average of the penalty cost values $\{x_i\}$.

$$E\{\mathbf{X}\} = \frac{\sum_{i=1}^{136} x_i}{136} = 27.769853$$

and the parameter c to $S(x)$ is

$$c = \frac{1}{E\{\mathbf{X}\}} = 0.036010.$$

Now the aggregate penalty cost cumulative distribution function can be computed from the formula

$$\begin{aligned}
F(z, t) &= \sum_{n=0}^{\infty} \exp(-\lambda t) \frac{(\lambda t)^n}{n!} S^{n*}(z) \\
&= \sum_{n=0}^{\infty} \exp(-\lambda t) \frac{(\lambda t)^n}{n!} \Gamma(cz, n).
\end{aligned}$$

The summation need not go up to infinity, of course, as the series will converge. The result would be a table of Cdf values, from which pdf values could also be calculated. Examples of Cdf and pdf values appear in the worked-out example in the next chapter.

Use of Auxiliary Functions

Computations can be simplified by using certain operational functions of probability calculus—the *probability generating function* and the *characteristic function*.

Probability Generating Functions

The probability generating function (pgf) for the integer-valued random variable \mathbf{X} is given by

$$h(s) = E\{s^{\mathbf{X}}\} = \sum_{k=0}^{\infty} \Pr\{\mathbf{X} = k\} s^k, \quad |s| < 1.$$

Suppose that the penalty costs $\mathbf{X}_1, \mathbf{X}_2, \dots$ are discrete random variables with common pmf $\{q_j\}$. Then the pmf of the aggregate penalty cost

$$\mathbf{Z}(t) = \mathbf{X}_1 + \mathbf{X}_2 + \dots + \mathbf{X}_{\mathbf{N}(t)}$$

is given by

$$r_j = \Pr\{\mathbf{Z}(t) = j\} = \sum_{n=0}^{\infty} \Pr\{\mathbf{Z}(t) = j | \mathbf{N}(t) = n\} \Pr\{\mathbf{N}(t) = n\}.$$

Since the \mathbf{X}_k and $\mathbf{N}(t)$ are mutually s -independent,

$$r_j = \sum_{n=0}^{\infty} \Pr\{\mathbf{N}(t) = n\} \Pr\{\mathbf{Z}(t) = j\}$$

The distribution of $\mathbf{Z}(t)$ is given by the n -fold convolution of $\{q_j\}$ with itself. Thus

$$\{r_j\} = \sum_{n=0}^{\infty} p_n \{q_j\}^{n*}.$$

The pgf of $\mathbf{Z}(t)$ is

$$\begin{aligned} h_{\mathbf{Z}}(s) &= \sum_{j=0}^{\infty} \sum_{n=0}^{\infty} p_n \{q_j\}^{n*} s^j \\ &= \sum_{n=0}^{\infty} p_n \sum_{j=0}^{\infty} \{q_j\}^{n*} s^j \\ &= \sum_{n=0}^{\infty} p_n h_{\mathbf{X}}(s) = h_{\mathbf{N}(t)}[h_{\mathbf{X}}(s)] \end{aligned}$$

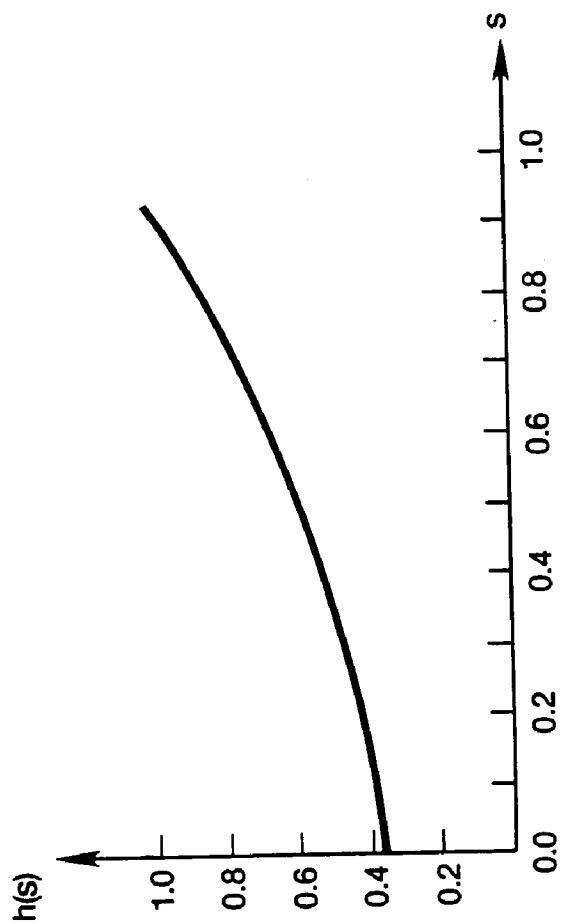


FIGURE 32
PROBABILITY GENERATING FUNCTION OF POISSON R.V. ($\lambda = 1$)

After multiplying out and collecting terms, the probability r_j is found as the coefficient of s^j in h_Z .

The pgf for the Poisson distribution with mean λt is

$$\begin{aligned} h_{\mathbf{N}(t)}(s) &= \sum_{j=0}^{\infty} \frac{\exp(-\lambda t)(\lambda t)^j}{j!} s^j \\ &= \exp(\lambda t) \sum_{j=0}^{\infty} \frac{(\lambda t s)^j}{j!} \\ &= \exp(-\lambda t) \exp(\lambda t s) \\ &= \exp[\lambda t(s - 1)]. \end{aligned}$$

Figure 32 shows the shape of the probability-generating function for a Poisson random variable. It then follows that

$$h_Z(s) = \exp\{\lambda t[h_X(s) - 1]\}.$$

In the Pólya case

$$h_{\mathbf{N}(t)}(s) = \left[1 - \frac{\alpha^t}{\beta}(s - 1) \right]^{-\beta}$$

and

$$h_Z(s) = \left[1 - \frac{\alpha^t}{\beta}(h_X(s) - 1) \right]^{-\beta}.$$

If the bracketed expression is negative, the pgf may not exist.

Example: Equispaced, Equiprobable Penalty Costs

When the possible penalty cost values are equally spaced, e.g., 1, 2, 3, 4, 5, 6, 7, and they have equal probability of being realized (in this example, 1/7), the problem is similar to a classic dice problem tackled by De Moivre. He was interested

“To find how many Chances there are upon any number of Dice, each of them of the same number of Faces, to throw any given number of points.[DEMO1756]”

That is, he wanted to find the probability distribution of the total number of spots that would appear after throwing a certain number of dice. De Moivre used a property of probability-generating functions (pgf's) to arrive at his answer.

Let each penalty cost \mathbf{X}_i take on values in the range $1, 2, \dots, a$ and let the pmf

$$p_k = \Pr\{\mathbf{X} = k\} = 1/a, \quad k = 1, 2, \dots$$

The pgf for X is defined as

$$\begin{aligned} \text{pgf}\{\mathbf{X}\} &\equiv \mathbb{E}\{s^{\mathbf{X}}\} = \sum_{k=1}^a p_k s^k \\ &= (s + s^2 + \dots + s^a)/a = \frac{s}{a} \left(\frac{1 - s^a}{1 - s} \right). \end{aligned}$$

The pgf exists for $|s| < 1$. It uniquely determines the distribution of the random variable \mathbf{X} [FISZ63].

Suppose \mathbf{X} and \mathbf{Y} are s -independent random variables. Then for any s , $s^{\mathbf{X}}$ and $s^{\mathbf{Y}}$ are also s -independent random variables. This means that

$$h_{\mathbf{X}+\mathbf{Y}}(s) = \mathbb{E}\{s^{\mathbf{X}} s^{\mathbf{Y}}\} = \mathbb{E}\{s^{\mathbf{X}}\} \mathbb{E}\{s^{\mathbf{Y}}\}$$

and so

$$h_{\mathbf{X}+\mathbf{Y}}(s) = h_{\mathbf{X}}(s) \cdot h_{\mathbf{Y}}(s),$$

an important property of pgf's. Let the aggregate penalty cost $\mathbf{Z}_n = \mathbf{X}_1 + \mathbf{X}_2 + \dots + \mathbf{X}_n$ have probability generating function $h_{\mathbf{Z}}(s)$. In common with the Laplace transform, the pgf of the sum of a set of mutually s -independent random variables is the product of their respective pgf's. In particular, the pgf of the sum of n s -independent and identically distributed random variables is their common pgf raised to the n th power. Thus

$$h_{\mathbf{Z}_n} = (h_{\mathbf{X}})^n = \left[\frac{s(1 - s^a)}{a(1 - s)} \right]^n, \quad |s| < 1.$$

The probability that $\mathbf{Z}_n = j$ is found as the coefficient of s^j in $h_{\mathbf{Z}}$, by definition of pgf. By the binomial expansion,

$$\Pr\{\mathbf{Z}_n = j\} = \sum_{k=0}^{\infty} \frac{(-1)^k}{a^n} \binom{n}{k} \binom{j - ak - 1}{n - 1}.$$

The aggregate penalty cost will be

$$\Pr\{\mathbf{Z}_n = j\} = \sum_{n=0}^j \Pr\{\mathbf{N}(t) = n\} = \sum_{k=0}^{\infty} \frac{(-1)^k}{a^n} \binom{n}{k} \binom{j - ak - 1}{n - 1}$$

with the convention that $\binom{n}{k} = 0$ for $k > n$.

Characteristic Functions

The pgf is useful only for discrete penalty cost distributions. For continuous distributions, the *characteristic function* (cf) exhibits analogous properties. The characteristic function of the continuous random variable \mathbf{X} is defined as

$$\begin{aligned} \Psi_{\mathbf{X}}(\omega) &= \mathbb{E}\{\exp(j\omega\mathbf{X})\} \\ &= \mathbb{E}\{\cos \omega\mathbf{X}\} + j\mathbb{E}\{\sin \mathbf{X}\} \\ &= \int_{-\infty}^{\infty} f_{\mathbf{X}}(x) \exp(j\omega\mathbf{X}) dx \end{aligned}$$

where j is the imaginary unit $\sqrt{-1}$. Let $\Psi_{\mathbf{X}}(\omega)$ be the common characteristic function of the penalty cost random variables \mathbf{X}_k . Then

$$\Psi_{\mathbf{Z}}(\omega) = \mathbb{E}_{\mathbf{N}(t)}\{\mathbb{E}_{\mathbf{X}}\{\exp(j\omega\mathbf{Z})|\mathbf{N}(t)\}\}$$

with

$$\begin{aligned} \mathbb{E}_{\mathbf{X}}\{\exp(j\omega\mathbf{Z})|\mathbf{N}(t) = n\} &= \mathbb{E}_{\mathbf{X}}\{\exp[j\omega(\mathbf{X}_1 + \mathbf{X}_2 + \cdots + \mathbf{X}_n)|\mathbf{N}(t) = n]\} \\ &= [\Psi_{\mathbf{X}}(\omega)]^n. \end{aligned}$$

Thus

$$\begin{aligned} \Psi_{\mathbf{Z}}(\omega) &= \mathbb{E}_{\mathbf{N}(t)}\{[\Psi_{\mathbf{X}}(\omega)]^{\mathbf{N}(t)}\} \\ &= \sum_{k=0}^{\infty} \Pr\{\mathbf{N}(t) = k\} [\Psi_{\mathbf{X}}(\omega)]^k, \\ &= h_{\mathbf{N}(t)}[\Psi_{\mathbf{X}}(\omega)] \end{aligned}$$

where

$$h_{\mathbf{N}(t)}(s) = \sum_{k=0}^{\infty} \Pr\{\mathbf{N}(t) = k\} s^k.$$

Lattice Penalty-Cost Distribution

When the penalty cost values form a lattice (discrete, equidistant values), the required numerical computations can be expressed by means of a recurrence formula. The formula can be derived by means of techniques [ADEL66, BEAR84] developed in operations research.

In a lattice distribution, all possible penalty-cost values would be expressed as multiples of the base unit x_1 :

$$x_i = ix_1, \quad i = 1, 2, \dots, s.$$

Let $\{q_i\}$ be the pmf for penalty cost \mathbf{X} . The value q_i gives the probability that the aggregate penalty cost will have the multiplier i :

$$q_i = \Pr\{\mathbf{X} = i\}$$

This pmf is assumed to satisfy

$$q_s > 0$$

and

$$q_i = 0, \quad i > s.$$

Instead of the Cdf $F(z, t)$, the pmf $f(z, t)$ will be employed:

$$f(z, t) = \sum_{n=0}^{\infty} \Pr\{\mathbf{N}(t) = n\} q_x^{n*} \quad \text{for } x = 0, 1, 2, \dots$$

The convolution can be expressed as

$$q_x^{k*} = \Pr\{\mathbf{X}_1 + \mathbf{X}_2 + \dots + \mathbf{X}_k = x\},$$

giving rise to the recurrence

$$f(z, t) = \sum_{\min(z, s)} (i\lambda t/z) q_i f(z - i, t)$$

with basis

$$f(0, t) = \exp(-\lambda t).$$

From the pdf, the CDF can be computed as

$$F(z, t) = \sum_{\nu}^{\lfloor z \rfloor} f(\nu, t).$$

Penalty Cost Vectors

So far, penalty cost has been reckoned as a scalar quantity. A generalization is to allow each penalty cost to form a finite-dimensional random vector \vec{X}_i of real numbers. Penalty cost vectors would be used when the penalty costs contain multiple inconvertible parts—when associating a single, monolithic penalty cost value with a software failure would result in “mixing apples and oranges.” That way, for example, one need not have to assign loss of life or limb an equivalent monetary value. The aggregate penalty cost would be defined by the vector sum of all penalty costs arising in $[0, t)$:

$$\vec{Z}(t) = \sum_{i=0}^{N(t)} \vec{X}_i,$$

where $\vec{X}_0 = \mathbf{0}$. The basic vector operations for random vectors are precisely those for m -tuples of real numbers. If the random vector \vec{v} is the m -tuple $(\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m)$ and the random vector \vec{w} is the m -tuple $(\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_m)$, then the sum $\vec{v} + \vec{w}$ is the random vector defined by

$$\vec{v} + \vec{w} = (\mathbf{v}_1 + \mathbf{w}_1, \mathbf{v}_2 + \mathbf{w}_2, \dots, \mathbf{v}_m + \mathbf{w}_m).$$

The s -expected value of the aggregate penalty is $E\{\vec{\mathbf{X}}\}\lambda t$, where

$$E\{\vec{\mathbf{X}}\} = (E\{\mathbf{X}_1\}, E\{\mathbf{X}_2\}, \dots, E\{\mathbf{X}_m\}),$$

the vector of s -expected values. (Here \mathbf{X}_i is the i th component of the vector $\vec{\mathbf{X}}$, not the penalty cost of the i th failure.)

CHAPTER 6

A REAL-LIFE EXAMPLE

The kind of data required to demonstrate the modeling technique was hard to come by: Since the technique has only now come into existence, no one knew to collect the data. The software reliability databases at the Rome Air Development Center did not have the right type of data, nor did other major reliability research centers. Fortunately it was discovered that the IBM Corporation did have a collection of usable data that they were kind enough to make available.

The project from which the data came is NASA's space shuttle. Some background: The reusable Space Shuttle Transportation System is the product of a cost-benefit approach to space operations, now that the U.S. has been freed from the expensive urgency of the moon race. The shuttle's long-range function is to serve as a transport for defense experimentation and provide a workshop for development of manufacturing and large-scale commercial operations in space. For example, the shuttle can rendezvous with satellites to service them or bring them back to earth. For military purposes, the shuttle can conduct manned reconnaissance sorties and deploy "spy" satellites into geostationary orbits. Ultimately the shuttle, with its very large payload capacity, will be instrumental in construction of a permanent space station and similar large structures[GATL81].

The space shuttle vehicle consists of a winged orbiter, two recoverable solid-rocket boosters and an external fuel tank. Lift-off thrust comes from the orbiter's three main liquid-propellant engines plus the boosters. After two minutes the boosters separate, their earthward fall slowed by parachutes. After eight minutes the orbital main engines shut down, and the jettisoned external tank burns up as it re-enters the atmosphere. A short burn of its two small Orbital Maneuvering

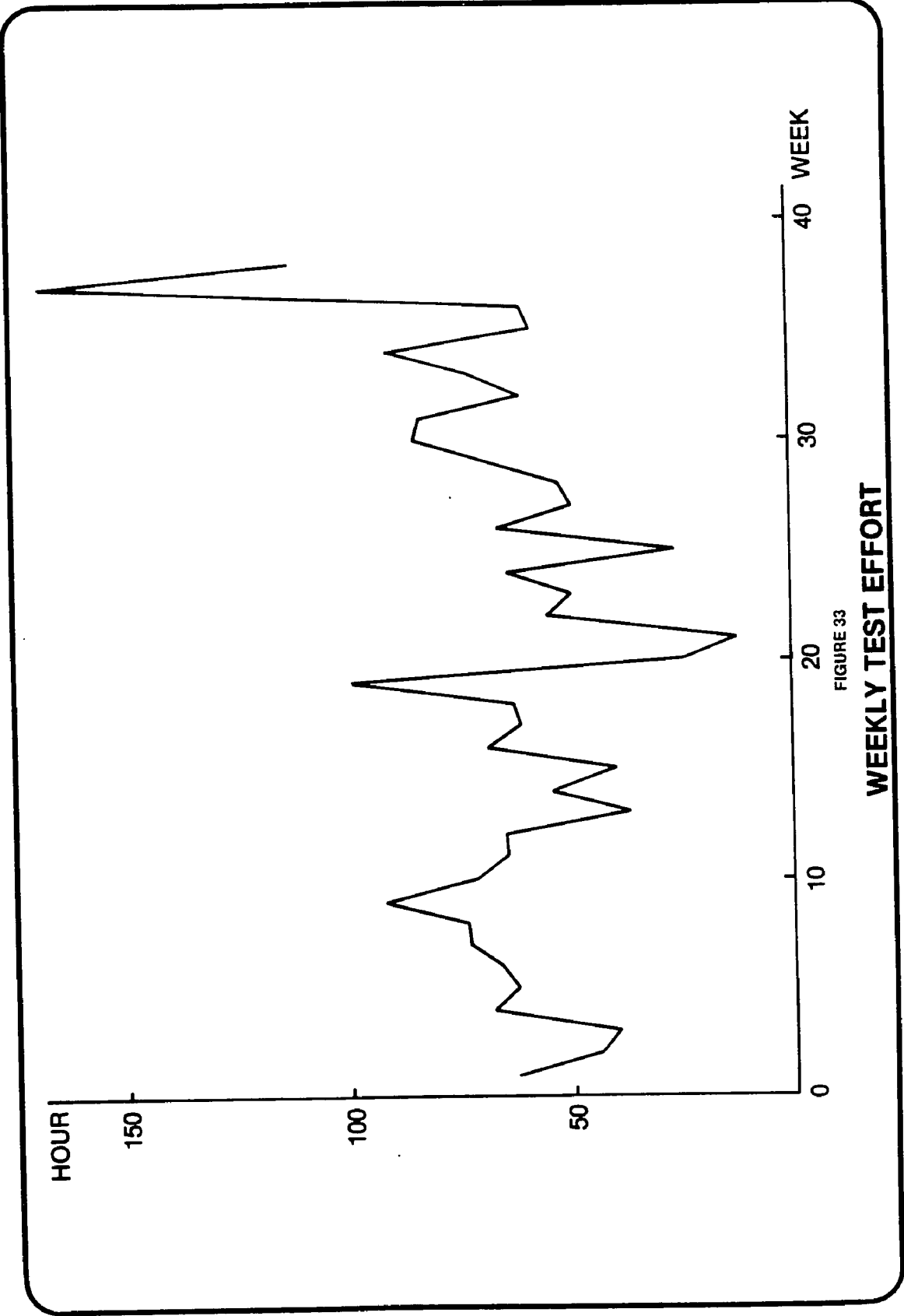


FIGURE 33
WEEKLY TEST EFFORT

System (OMS) engines causes the spacecraft to enter orbit. When the orbiter is ready to return to earth, it turns around, fires its OMS engines to decelerate, and then glides to an airplane-like landing.

The first orbital test flight—Space Transportation System (STS)-1—took place 12–14 April 1981; STS-2, 12–14 November 1981; STS-3, 22–30 March 1982; and STS-4, 22 June–4 July 1982. Following these four orbital test flights, operational flights commenced in November 1982. After 24 successful launches, the shuttle *Columbia* exploded shortly after takeoff, destroying the vehicle and its crew, and setting back the program for an indefinite period of time.

The software studied was developed under contract to the National Aeronautics and Space Administration's Johnson Space Center (NASA/JSC) in Houston. The application is the shuttle's Ground Processing System. This system, with over 500,000 lines of source code, is one of the largest real-time systems ever developed. The system receives high-speed telemetry data from the shuttle to verify and augment the functions performed by the on-board computers. Additionally, the system must respond to interactive commands from ground-based flight control personnel to perform trajectory and abort predictions, and to control the worldwide ground tracking network[SPEC84].

After validation by the developing organization, the software was forwarded to a separate organization for independent verification and validation (IV&V). Software failures that occurred during IV&V, operational mission simulations, and the actual missions, were documented in a series of *anomaly reports*.

The anomaly reports are summarized in the tables that appear in Appendix C. The 38th week corresponds to the STS-3 flight itself. The last entry is from the STS-4 flight. Figure 33 plots the weekly test effort in hours.

Since the data is from a single project, this example does not serve to demonstrate validity of the modeling technique. Rather, the data is used to show a

worked-out numerical example of how the technique would be employed in a real-world situation. The sample data was divided into two sets: the *fit set* (weeks 1–38) and the *check set*. (the STS-4 flight). The fit set was used to determine the failure rate parameter of the failure-counting component of the model as well as an empirical penalty-cost distribution. From this information the modeling technique is applied to probabilistically characterize the aggregate penalty cost to be incurred on the STS-4 mission. The check set contains the actual failure and penalty cost experience on STS-4. Because of the scarcity of “critical” failures, the “critical” and “major” categories, distinct in the fit set, were lumped together in the prediction set. This convention was established by the suppliers of the data.

Failure Rate Estimation

The first thing to note about the failure data in Appendix C is that the failure counts are summarized on a calendar week basis. The more usual form of software failure data is a list of the times at which each failure occurred. But no matter: The Goel-Okumoto Nonhomogeneous Poisson Process (NHPP) model was specially designed to work well with this type of batched data. We will use the Goel-Okumoto formulation to predict the failure rate λ for the STS-4 operational software.

The key formulas of the Goel-Okumoto NHPP model, as covered in the Survey of Related Research, are the mean value function and the failure rate. The mean value function is described by

$$N(\tau) = N(\infty)[1 - \exp(-b\tau)]$$

where $N(\infty)$ is the expected number of faults that would be uncovered if testing went on forever, and b is a proportionality constant determining the fault discovery rate. The mean value function $N(t)$ gives the expected number of failures that would occur up to time t . The number of failures up to time t is the random

function $N(t)$. The corresponding instantaneous failure rate is

$$\lambda(t) = N'(t) = N(\infty)b \exp(-bt).$$

In order to determine $\lambda(t)$, we must first obtain estimates of the parameters $N(\infty)$ and b . The maximum likelihood estimators for $N(\infty)$ and b were computed by the following reasoning:

Under the nonhomogeneous Poisson process assumption, the probability of w failures being observed up to time t is given by the probability mass function

$$\Pr\{N(t) = w\} = \frac{[N(t)]^w}{z!} \exp[-N(t)].$$

Let $\dots, w_i, w_{i+1}, \dots$ be the number of failures that have been observed up to times $\dots, t_i, t_{i+1}, \dots$ respectively. The conditional probability that $N(t_{i+1}) = w_{i+1}$ given that $N(t_i) = w_i$ is given by the conditional probability mass function

$$\begin{aligned} & \Pr\{N(t_{i+1}) = w_{i+1} | N(t_i) = w_i\} \\ &= \Pr\{N(t_{i+1}) - N(t_i) = w_{i+1} - w_i\} \\ &= \frac{[N(t_{i+1}) - N(t_i)]^{w_{i+1} - w_i}}{(z_{i+1} - z_i)!} \exp\{-[N(t_{i+1}) - N(t_i)]\}. \end{aligned}$$

The joint probability mass function for the observed data pairs (t_i, w_i) , $(t_2, w_2), \dots, (t_n, z_n)$ is thus

$$\begin{aligned} L &= \Pr\{N(0) = 0, N(t_1) = w_1, \dots, N(t_n) = w_n\} \\ &= \prod_{i=1}^n \frac{[N(t_i) - N(t_{i-1})]^{w_i - w_{i-1}}}{(z_i - z_{i-1})!} \exp\{-[N(t_i) - N(t_{i-1})]\}. \end{aligned}$$

The estimates for $N(\infty)$ and b will come from maximizing the likelihood function. It will prove convenient to work with the logarithm of the likelihood function L :

$$\ln L = \sum_{i=1}^n (z_i - z_{i-1}) \ln[N(t_i) - N(t_{i-1})] - \sum_{i=1}^n \ln[(z_i - z_{i-1})!] - N(t_n)$$

where $N(t_i) = N(\infty)(1 - \exp(-bt_i))$. Now the $\ln L$ function is maximized by taking derivatives with respect to the variables $N(\infty)$ and b , and setting them equal to zero. Two equations are obtained:

$$N(\infty) = \frac{w_n}{1 - \exp(-bt_n)}$$

and

$$N(\infty)t_n \exp(-bt_n) = \sum_{i=1}^n (w_i - w_{i-1}) \frac{t_i \exp(-bt_i) - t_{i-1} \exp(-bt_{i-1})}{\exp(-bt_i) - 1 - \exp(-bt_i)}$$

Combining the two yields

$$\frac{t_n \exp(-bt_n)}{1 - \exp(-bt_n)} = (w_i - w_{i-1}) \frac{t_i \exp(-bt_i) - t_{i-1} \exp(-bt_{i-1})}{\exp(-bt_{i-1})}$$

This final equation can be numerically solved to furnish the value of b . Back-substitution into the previous equations provides $N(\infty)$ and then $\lambda(t)$.

Using the software failure data in Appendix C, obtained from testing and the operational STS-3 mission, the maximum likelihood estimators for the Goel-Okumoto model parameters were computed to be

$$N(\infty) = 597.887$$

and

$$b = 0.20988 \times 10^{-3}.$$

At the beginning of STS-4, $t = 2456.9$ hours of testing had been completed. The predicted failure rate at the beginning of STS-4 was $\lambda(2456.9) \approx 0.075$. In actuality the average failure rate turned out to be 0.07 (14 failures/200 hours). So one failure less than predicted actually occurred.

The potential penalty cost values form a lattice because they are discrete and equidistant. Using the empirical distribution and a severity rating of 3 for critical/major and 2 for minor, the aggregate penalty cost pmf and Cdf were

computed via the lattice method. The cumulative distribution function $F(z, t)$ for the aggregate penalty cost $\mathbf{Z}(t)$ was obtained by the summation

$$F(z, t) = \sum_{\nu=0}^{\lfloor z \rfloor} f(\nu, t),$$

where z is the aggregate penalty cost value for which the probability is desired and the probability mass function $f(z, t)$ is given by the exact recursion formula

$$f(z, t) = \sum_{i=i}^{\min(z, s)} (i\lambda t/z) q_i f(z - i, t).$$

The upper limit of the penalty cost range is s (3) and the values for q_i are the percentages of the failures in the fit set that incurred the penalty cost value i .

The cumulative distribution function and probability mass function are tabulated and plotted in Appendix C. The predicted median aggregate penalty cost is seen to be about 36, while the actual value was $3 \times 5 + 9 \times 2 = 33$.

Summary of How to Use the Modeling Technique

The failure and penalty cost data can come from operational experience or from the results of simulation runs. Such simulation would model the target computer, the external environment, and the demands placed on the system. Penalty cost assessment after a failure would take place through subjective or automated examination of the failure's effects. Modeling of the external environment would in most cases have to be more extensive than if only failure occurrence was to be detected. A simulation could not, of course be expected to model all possible objects that could be adversely affected by a failure, so penalty cost values would have to be qualified as having arisen in the context of a particular simulation.

The failure times and penalty costs are used to statistically infer the values of unknown parameters in the failure-counting and penalty-cost components. The particular parameters will depend upon the chosen structures of those components.

Then the cumulative distribution function for the aggregate penalty cost is obtained and solved by means of the appropriate numerical technique. The resulting Cdf can be used to find the probability mass/density function and figures of merit and interest such as percentiles, mean, variance, and so forth.

CHAPTER 7

SAMPLE SIZE DETERMINATION

Attention now turns to the design of the informative experiment. The observation of software failures and penalty costs can be terminated at a preassigned time point (*truncation*) or after a preassigned number of software failures have occurred (*censorship*). The Law of Large Numbers argues that the larger the number of observed software failures the more closely they will tend to be representative of the full stochastic process (“population”). The sample size used in determining the parameters of the model should be large enough so that inferences drawn from the sample will have no greater probability of being wrong than a pre-specified cap. It is possible to develop such a sample size figure statistically. The larger the sample size the better, but a point of diminishing returns will be reached at which it will not be cost effective to go on. Accuracy and sample size analysis depends on a sample size large enough so that sample statistics have distributions that are approximated by limiting distributions.

Reporting Uncertainty

An *estimate* is a numerical value obtained from a sample that is assigned to a population parameter. Point estimates (single values) of a population parameter (e.g., the mean) are of little use unless accompanied by accuracy measures. A *confidence interval* is an interval computed from sample values that has a specified probability $1 - \alpha$ of containing the true value of an unknown parameter. The confidence probability is most popularly selected to be 95%. This figure means that in 95% of the cases the constructed interval can be expected to include the true value and in 5% of the cases it can be expected not to. This represents odds

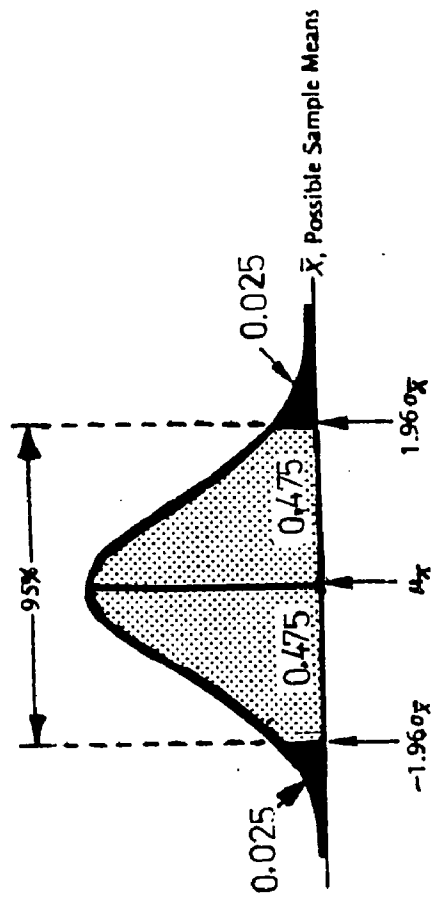


FIGURE 34
INTERVAL COEFFICIENT, 95% CONFIDENCE INTERVAL

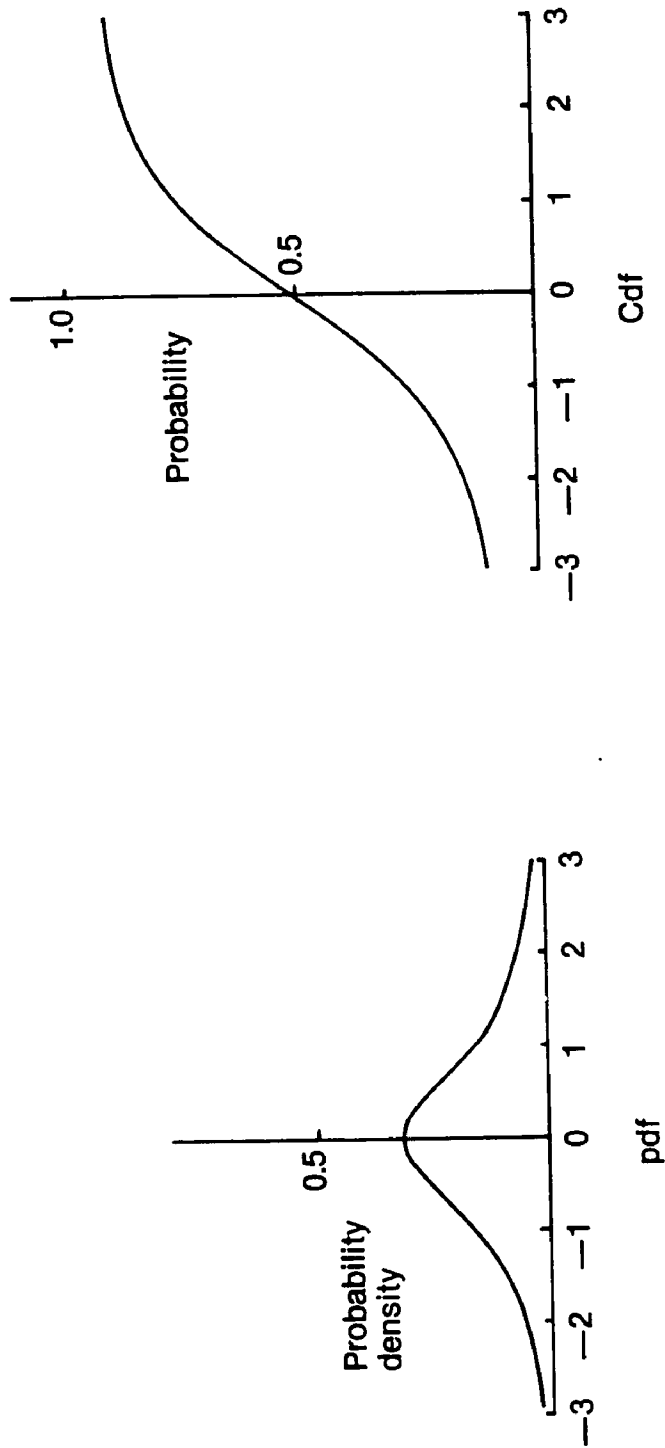


FIGURE 35
STUDENT'S t -DISTRIBUTION (DF = 1)

of 19 to 1 of being right. The endpoints of the interval are the *confidence limits* ℓ and u , both functions of the observations.

A general way to obtain a confidence interval for the failure rate λ of a single program version is to employ the Central Limit Theorem. According to the Theorem, the standardized sample mean

$$\frac{\bar{x} - \mu}{\left(\frac{\sigma}{\sqrt{n}}\right)}$$

of a sequence of s -independent and identically distributed random variables

$$X_1, X_2, \dots, X_n$$

with $E\{X_i\} = \mu < \infty$, $\text{Var}\{X_i\} = \sigma^2 < \infty$, converges to the standard normal distribution as $n \rightarrow \infty$. Figure 34 shows the 95% confidence interval in this case. (For practical purposes when $n > 30$.) When n is small or σ is not known, the statistic is distributed according to a Student's t -distribution with $n - 1$ degrees of freedom. Figure 35 shows the shape of a typical Student's t -distribution. Each choice of the degrees of freedom ν determines a distinct but similar density curve. This distribution is bell-shaped but flatter than the normal.

Recall that the failure rate $\lambda = 1/\text{MTTF}$ so confidence limits for the sample mean (MTTF) can be obtained and then their reciprocals taken to produce confidence limits for the failure rate.

For a large sample, the confidence interval for μ is

$$\bar{x} \pm Z_{\alpha/2} \left(\frac{\sigma}{\sqrt{n}}\right)$$

and for a small sample it is

$$\bar{x} \pm t_{\alpha/2} \left(\frac{\sigma}{\sqrt{n}}\right).$$

The score $Z_{\alpha/2}$ is that value of z for which $\Pr\{-z \leq Z \leq z\} = 1 - \alpha$, where Z is a standard normal random variable. Here are the values of $Z_{\alpha/2}$ for some common

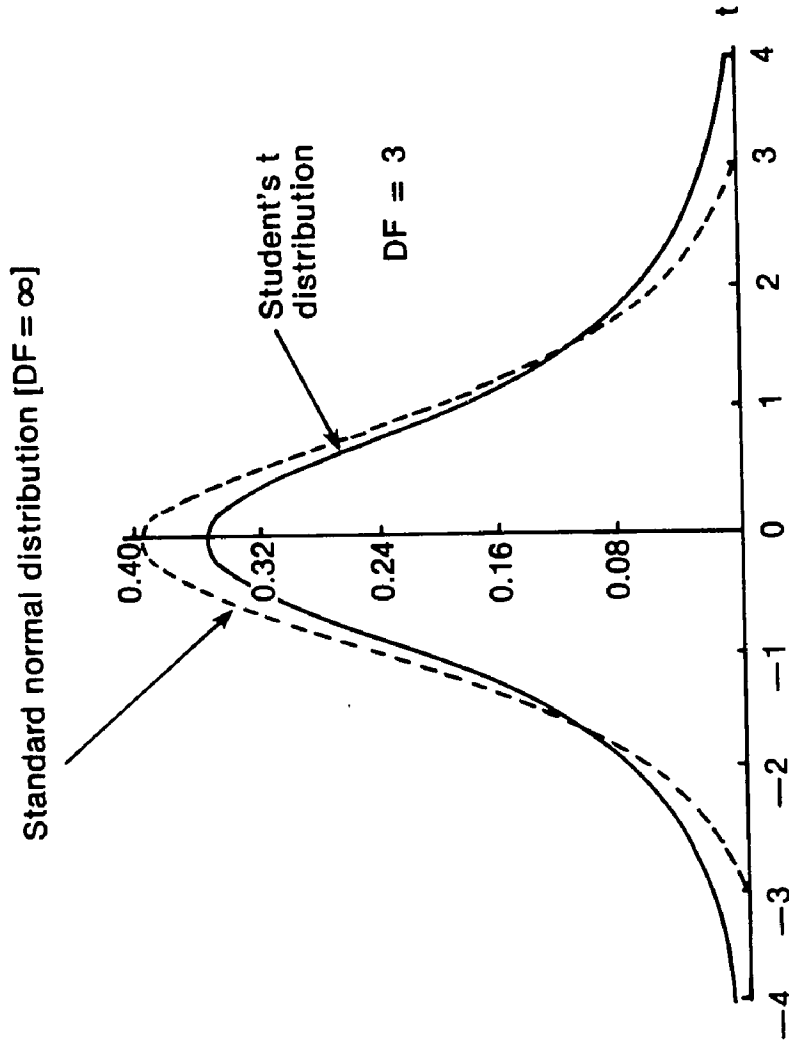


FIGURE 36

STANDARD NORMAL vs. STUDENT'S t-DISTRIBUTION

confidence levels:

$$1 - \alpha = 99.9\% \implies Z_{\alpha/2} \approx 3.291$$

$$1 - \alpha = 99\% \implies Z_{\alpha/2} \approx 2.576$$

$$1 - \alpha = 95\% \implies Z_{\alpha/2} \approx 1.960$$

$$1 - \alpha = 90\% \implies Z_{\alpha/2} \approx 1.645$$

$$1 - \alpha = 80\% \implies Z_{\alpha/2} \approx 1.282$$

$$1 - \alpha = 50\% \implies Z_{\alpha/2} \approx 0.674.$$

The standard normal distribution is given by the Cdf

$$\Phi(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^z \exp(u^2/2) du.$$

It has mean, median and mode equal to zero, and variance equal to one.

Similarly, $t_{\alpha/2}$ is that value of t for which $\Pr\{-t \leq \mathcal{T} \leq t\} = 1 - \alpha$, where \mathcal{T} is a t -distributed random variable. The Student's t -distribution with ν degrees of freedom is given by Cdf

$$F(t) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\pi\nu}\Gamma(\nu/2)\left(1 + \frac{t^2}{\nu}\right)^{(\nu+1)/2}} dx.$$

(Note that the “ t ” here has nothing to do with time.) For neither of these two distributions is it possible to write down a simple closed expression for the Cdf for any choice of parameter values, but the Cdf's have been computed to great accuracy. It will always be true that $t_{\alpha/2} > Z_{\alpha/2}$; t -confidence intervals are longer than their standard normal counterparts. The standard normal and Student's t -distributions are ubiquitously tabulated in the appendices of statistics textbooks. Figure 36 compares the standard normal and Student's t -distribution.

Sample Size for Determining Failure Intensity

From the confidence interval formula, the *tolerable error* (half-width of the confidence interval) is

$$\epsilon = \frac{Z_{\alpha/2}\sigma}{\sqrt{n}}$$

in the large-sample case. Solving for n gives

$$n = \left(\frac{Z_{\alpha/2}\sigma}{\epsilon} \right)^2,$$

the required sample size to achieve the desired error bound and confidence level. Since the population standard deviation σ would generally not be known a priori, a small pilot sample could be undertaken and its sample standard deviation s used.

Thus

$$n = \left(\frac{Z_{\alpha/2}s}{\epsilon} \right)^2.$$

In the small sample case

$$n = \left(\frac{t_{\alpha/2}s}{\epsilon} \right)^2.$$

This estimate is *consistent* in that the larger the sample size the closer the sample mean comes to the population mean. It is *unbiased* in that the s -expected value of the estimate equals the population parameter. It is *sufficient* in that it utilizes all occurrence information from the set of sample data. It is more *efficient* than the median in that it has less variability for a given sample size. Sample size depends on the confidence level and degree of precision that will be required for the results and is affected by the variability and spread of the sampled data. The greater the degree of confidence or precision desired, the larger must be the sample size. Of course, time and cost constraints must also be considered. Note that halving the error bound results in a four-fold increase in the required sample size. The discussion here has centered on data from a single program version. In most cases the failure rate for a program version will be estimated using the failure rate formulation from an existing software reliability model. The properties of the estimators and the

required sample sizes will vary. Some authors do discuss these issues, but this is an often neglected area.

Sample Size for Determining Aggregate Penalty Cost

One of the keys to the usability of the modeling technique so far presented is the sample size required. Assume that the time unit is chosen such that $t = 1$. Assuming the homogeneous-Poisson-based distribution of software failures—as developed in the chapter on the failure-counting component—let the random variable \mathbf{N} represent the number of software failures during this time period. Then

$$E\{\mathbf{N}\} = \text{Var}\{\mathbf{N}\} = n.$$

The s -expected value and variance of the penalty cost incurred by the i th software failure are given by

$$E\{\mathbf{X}_i\} = m$$

$$\text{Var}\{\mathbf{X}_i\} = \sigma^2.$$

The aggregate penalty cost random variable

$$\mathbf{Z} = \mathbf{X}_1 + \mathbf{X}_2 + \cdots + \mathbf{X}_N$$

has expected value $E\{\mathbf{Z}\} = z = nm$, and variance $\text{Var}\{\mathbf{Z}\} = n(\sigma^2 + m^2)$, as derived previously. It is desired to determine the sample size required to produce a $100P\%$ confidence interval of range $100k\%$ about the true value. Symbolically,

$$\Pr \left\{ \left(1 - \frac{k}{2}\right) z < \mathbf{Z} < \left(1 + \frac{k}{2}\right) z \right\} = 1 - \alpha$$

or

$$\Pr \left\{ \frac{-kz}{\sqrt{\text{Var}\{\mathbf{Z}\}}} < \frac{\mathbf{Z} - z}{\sqrt{\text{Var}\{\mathbf{Z}\}}} < \frac{kz}{\sqrt{\text{Var}\{\mathbf{Z}\}}} \right\} = 1 - \alpha.$$

According to the Central Limit Theorem, $(\mathbf{Z} - z)/\sqrt{\text{Var}\{\mathbf{Z}\}}$ will be distributed as a standard normal variable when the sample is reasonably large. Thus

$$\frac{kz}{\sqrt{\text{Var}\{\mathbf{Z}\}}} = \Phi\left(\frac{1 - \alpha}{2}\right),$$

where $\Psi(\cdot)$ is the Cdf of the standard normal distribution. When nm is substituted for z , and $n(\sigma^2 + m^2)$ for $\text{Var}\{\mathbf{Z}\}$, this rearranges to

$$n_{\text{MIN}} = \left(\frac{Z_{\alpha/2}}{k}\right)^2 [1 + (\sigma/m)^2].$$

At the 95% confidence level, for example, a sample size of 96 software failures would be sufficient to furnish a figure accurate to within $\pm 10\%$. A sample size of 384 would provide it to within $\pm 5\%$. If an a priori estimate \hat{z}_0 of the s -expected aggregate penalty cost is available, the sample size can be less than n_0 with good results. This initial estimate can be thought of as arising from a Bayesian prior distribution. Let \hat{z}_E be an estimate of the s -expected aggregate penalty cost based on sampling data. The best estimate of the s -expected aggregate penalty cost might be put in the form of a weighted average

$$\hat{z} = (1 - \xi)\hat{z}_E + \xi\hat{z}_0, \quad 0 < \xi \leq 1.$$

The sample sizes given previously arise in the special case where the breaking constant $\xi = 1$. When the sample size n is less than n_0 , however, ξ will be less than unity.

It is desired to find the values of the coefficients ξ and $(1 - \xi)$. If ξ is restricted to a value small enough to limit random fluctuations, for example by the restriction

$$\text{Pr}\{\xi z - kz < \xi \mathbf{Z} < \xi z + kz\} = P$$

implying

$$\text{Pr}\left\{\frac{-kz}{\xi\sqrt{\text{Var}\{\mathbf{Z}\}}} < \frac{\mathbf{Z} - z}{\sqrt{\text{Var}\{\mathbf{Z}\}}} < \frac{kz}{\xi\sqrt{\text{Var}\{\mathbf{Z}\}}}\right\} = P.$$

By the Central Limit Theorem again, the statistic $(\mathbf{Z} - z)/\sqrt{\text{Var}\{\mathbf{Z}\}}$ can be approximated by the standard normal distribution and so

$$\frac{kz}{\xi\sqrt{\text{Var}\{\mathbf{Z}\}}} = y,$$

where, as before, y is such that $\Psi(y) - \Psi(-y) = P$. Substituting $z = mn$, $\sqrt{n_0} = y/k$ and $n_{\text{MIN}} = n_0[1 + (\sigma/m)^2]$ yields, after simplifying,

$$\xi = \sqrt{n/n_{\text{MIN}}}.$$

See[HOSS83, BENJ77].

CHAPTER 8

CONCLUSIONS

Summary

The dissertation has discussed research into a method for incorporating penalty costs into software reliability models. First, related research was surveyed to portray the history and state of the art of software reliability modeling. Then the method for incorporating penalty costs was axiomatically developed from assumptions about software failure and its concomitant penalty cost. Numerical techniques were presented for simplifying the required calculations, including examples. The paper culminates in a demonstration of the use of a model constructed along the lines of the method.

The main contribution of the research is the modeling method and the class of models the method generates. The research does not attempt to address all the criticisms that have been directed at the state of the art in software reliability modeling; It does, however, represent a significant addition to the repertoire of methods available in constructing software reliability models.

The dissertation presents the modeling technique in the manner that has become traditional for new software reliability models: theory, model construction, parameter estimation and a demonstration based on data from a real-world project.

The modeling method views penalty costs as a secondary stochastic process that feeds into a primary stochastic process. Both processes fulfill a simple set of postulates. The technical features of the model can be summarized by its key assumptions:

- 1) Software failures are ordered, highly localized events occurring randomly in time.
- 2) The system is restarted (“renewed”) after every software failure; consequently the interfailure times are mutually s -independent.
- 3) The probability of failure in a small interval of time is proportional to the length of that interval multiplied by the failure rate.
- 4) The failure rate is considered either constant with a gamma-distributed prior (producing a type of temporally homogeneous Poisson process), actually gamma-distributed (producing a type of Pólya process) or arbitrary.
- 5) Penalty costs are mutually s -independent, nonnegative random variables.
- 6) Penalty cost amounts are s -independent of the failure-counting process.
- 7) The individual penalty cost distribution can be empirical or theoretical. Usually it falls into one of the categories for which simplified numerical techniques are available: lattice distribution (recurrence technique), discrete distribution (probability-generating function technique), continuous distribution (characteristic function technique), “robust” program (incomplete gamma function technique).

In most cases the aggregate penalty-cost cost cumulative distribution function and probability density/mass function are easily computed. The distribution can then be plotted or tabulated; or it can be summarized through descriptive statistics such as moments, percentiles, etc.

Future Research Directions

Often with new types of quantitative models there is a shortage of suitable historical data. Since the model did not exist in the past, no one knew to collect the data needed for the model. Early software reliability modelers faced this problem. Efforts at the Rome Air Development Center and elsewhere have made

progress in making available databases of software failure information. Unfortunately, while failure times have been recorded for a variety of projects, failure severity data is much harder to come by. So one obvious research area is in finding out how to increase the quantity and quality of historical failure collection and dissemination. An aggravating factor is the psychological and political resistance to “self-incrimination.”

Improvements in the modeling technique itself might come in several areas:

- 1) As new software reliability models appear that improve on other aspects of the state of the art, the technique can be applied to them.
- 2) Empirical studies of severity data can suggest specific penalty-cost distributions for various classes of programs. This could result in a richer class of models and simpler calculations.
- 3) Penalty costs can possibly be related to static program features—if only to provide prior distributions for parameters.
- 4) Other penalty cost measures besides “aggregate penalty cost incurred over a future interval” could be developed that might be more informative in certain applications.
- 5) The construction of simulators that accurately model penalty cost could be studied, since for many applications simulation may be the only feasible way to acquire the historical data for parameter estimation.

Conclusion

Software reliability modeling is becoming increasingly important as computerized systems become bigger and more complex, and as these systems are given the responsibility of controlling powerful physical forces, vital biological processes, and trustful financial accounting. Bugs in computer programs can cause adverse effects on the well-being of individuals and their environments. A quantification of

the undesired consequences of a software failure is termed its penalty cost. Current software reliability models simply count failures and are deficient in taking into account the penalty costs of software failures. A new technique, grounded in the theory of random functions, provides a way for existing and compatible future software reliability models to probabilistically characterize the aggregate penalty cost incurred over a future time interval.

The modeling technique presented here should prove helpful in specifying penalty-cost-based reliability goals, measuring and predicting penalty-cost-based reliability, and evaluating competing software development technologies.

Digital hardware is starting to take on the reliability characteristics of software, promising an even greater role for software reliability modeling techniques.

REFERENCES

- [ADEL66] ADELSON, R. M. Compound Poisson distributions. *Operations Research Quarterly* 17 (1966), 30–36.
- [AMER67] *Method of Recording and Measuring Work Injury Experience*, ANSI Z16.1, American National Standards Institute, New York, 1967.
- [BARL75] BARLOW, R. E., & PROSCHAN, F. *Statistical Theory of Reliability and Life Testing: Probability Models*, Holt, Rinehart and Winston, New York, 1975.
- [BATE52] BATES, G. E., & NEYMAN, J. Contribution to the theory of accident proneness I. An optimistic model of the correlation between light and severe accidents. *University of California Publications in Statistics 1* (1952), 215–254, University of California Press, Berkeley.
- [BATE79] BATESON, G. *Mind and Nature: A Necessary Unity*, Bantam Books, New York, 1979.
- [BEAR84] BEARD, R. E., PENTIKÄINEN, T., & PESONEN, E. *Risk theory*, Chapman and Hall, New York, 1984.
- [BEEK84] BEEKMAN, J. A. Risk convolution calculations. In *Premium Calculation in Insurance*, de Vylder, F. et al., Ed., D. Reidel Publishing, Dordrecht, Holland, pp. 19–30.
- [BEIZ84] BEIZER, B. *Software System Testing and Quality Assurance*, Van Nostrand Reinhold, New York, 1984, p. 293.
- [BENJ77] BENJAMIN, B. *General Insurance*, Heinemann, 1977, pp. 162–165.
- [BILL83] BILLINTON, R., & ALLAN, R. N. *Reliability Evaluation of Engineering Systems: Concepts and Techniques*, Plenum Press, New York, 1983, pp. 128–129.
- [BROW80] BROWNING, R. L. *The Loss Rate Concept in Safety Engineering*, Marcel Dekker, Inc., New York, 1980, pp. 1–13.
- [BUSH55] BUSH, R. R., & MOSTELLER, F. *Stochastic Models for Learning*, Wiley, New York, 1955.

- [CHEU80] CHEUNG, R. C. A user-oriented software reliability model. *IEEE Transactions on Software Engineering SE-6*, 2 (March, 1980), 118–125, IEEE, New York.
- [CHIA68] CHIANG, C. L. *Introduction to Stochastic Processes in Biostatistics*, John Wiley & Sons, New York, 1968.
- [CHUN60] CHUNG, K. L. *Markov Chains with Stationary Transition Probabilities*, Springer-Verlag, Berlin, 1960.
- [CINL72] ÇINLAR, E. Superposition of point processes. In *Statistical Analysis: Theory and Applications*, Lewis, P. A. W., Ed., Wiley, New York, 1972, pp. 549–606.
- [COOK81] COOK, I. D, JR. *The H-Function and Probability Density Functions of Certain Algebraic Combinations of Independent Random Variables with H-Function Probability Distributions*, Ph.D. Dissertation, The University of Texas at Austin, 1981.
- [COST78] COSTES, A., LANDRAULT, C., & LAPRIE, J. C. Reliability and availability models for maintained systems featuring hardware failures and design faults. *IEEE Transactions on Computing C-27* (June, 1978), 548–560.
- [COX62] COX, D.R. *Renewal Theory*, Methuen, London, 1962.
- [CRAM54] CRAMÉR, H. On some questions connected with mathematical risk. *University of California Publications in Statistics 2* (1954), 99–124.
- [DANA74] DANA, J. A., & BLIZZARD J. D. *The Development of a Software Error Theory to Classify and Detect Software Errors*, Logicon Report HR-74012, 1974.
- [DEMO1756] DE MOIVRE, A. *The Doctrine of Chances*, Millar, London, 1718,1738,1756.
- [DOOB53] DOOB, J. L. *Stochastic Processes*, Wiley, New York, 1953.
- [FELL57] FELLER, W. *An Introduction to Probability Theory and Its Applications*, Vol. 1 (2nd ed.), Wiley, New York, 1957.

- [FELL66] FELLER, W. *An Introduction to Probability Theory and Its Applications*, Vol. 2, Wiley, New York, 1966.
- [FELL68] FELLER, W. *An Introduction to Probability Theory and Its Applications*, (3rd ed.), Wiley, New York, 1968.
- [FISZ63] FISZ, M. *Probability Theory and Mathematical Statistics*, Wiley, New York, 1963.
- [FORM77] FORMAN, E. H., & SINGPURWALLA, W. D. An empirical stopping rule for debugging and testing computer software. *Journal of the American Statistical Association* 72 (December, 1977), 750-757.
- [GATL81] GATLAND, K. W. *The Illustrated Encyclopedia of Space Technology*, Harmony Books, New York, 1981.
- [GILB74] GILB, T. *Reliable EDP Application Design*, Petrocelli Books, New York, 1974.
- [GIRA66] GIRAULT, M. *Stochastic Processes*, Springer-Verlag, New York, 1966.
- [GOEL79] GOEL, A. L., & OKUMOTO, K. Time dependent error-detection-rate model for software and other performance measures. *IEEE Transactions on Reliability R-28* (August, 1979), 206-211, IEEE, New York.
- [GOLD81] GOLDBERG, H. *Extending the Limits of Reliability Theory*, John Wiley & Sons, New York, p. 2.
- [GOOD79] GOODENOUGH, J. A survey of program testing issues. In *Research Directions in Software Technology*, Wegner, P., Ed., MIT Press, 1979, pp. 316-340.
- [GOVI84] GOVIL, K. K. Incorporation of execution time concept in several software reliability models. *Reliability Engineering* 7 (1984), 235-249.
- [GROS74] GROSS, D., & HARRIS, C. M. *Fundamentals of Queueing Theory*, New York, 1974, p. 34.

- [HALS77] HALSTEAD, M. *Elements of software science*, Elsevier-North-Holland, New York, 1977.
- [HARR63] HARRIS, T. E. *The Theory of Branching Processes*, Springer-Verlag, Berlin, 1963.
- [HAST75] HASTINGS, N. A. J., & PEACOCK, J. B. *Statistical Distributions*, John Wiley & Sons, New York, 1975.
- [HAYS84] HAYS, R. V., & KLUGMAN, S. A. *Loss Distributions*, John Wiley & Sons, 1984.
- [HECH77] HECHT, H. Measurement, estimation, and prediction of software reliability. In *Software Engineering Technology*, Infotech International, Maidenhead, Berkshire, England, Vol. 2, 1977, pp. 209–224.
- [HENL81] HENLEY, E. J., & KUMAMOTO, H. *Reliability Engineering and Risk Assessment*, Prentice Hall, Inc., Englewood Cliffs, NJ, 1981, p. 1.
- [HOPK70] HOPKINS, J. D. *Software Engineering Techniques*, NATO, 1970.
- [HOSS83] HOSSACK, I. B., POLLARD, J. H., & ZEHNWIRTH, B. *Introductory Statistics with Applications to General Insurance*, Cambridge University Press, Cambridge, 1983.
- [HUDS67] HUDSON, G. R. *Program errors as a birth and death process*, Report SP-3011, System Development Corp., 1967.
- [JELI71] JELINSKY, Z., & MORANDA, P. B. Software reliability research. *Conference on Statistical Methods for the Evaluation of Computer Systems Performance* (November, 1971), Brown University, Providence, RI.
- [JELI73] JELINSKY, Z., & MORANDA, P. B. Application of a probability-based model to a code reading experiment. *1973 IEEE Symposium on Computer Software Reliability*, IEEE Cat. No. 73CH0741-9CSR, 78–80, IEEE, New York.
- [JORD83] JORDAN, J. R., & WHITTINGTON, H. W. Reliability of microelectronic devices. In *The Impact of Microelectronics Technology*, Jack, M. A., Ed., 1983, p. 98.

- [KHAM65] KHAMIS, S. H., & RUDERT, W. *Tables of the Incomplete Gamma Function Ratio*, von Liebig, Darmstadt, West Germany, 1965.
- [KIM84] KIM, K. H. Software fault tolerance. In *Handbook of Software Engineering*, Vick, C. R., & Ramamoorthy, C. V., Ed., Van Nostrand Reinhold, New York, 1984, pp. 437-455.
- [KOLM31] KOLMOGOROV, A. M. Über die analitischen Methoden in der Wahrscheinlichkeitsrechnung. *Mathematische Annln.* 104 (1931), 415-458.
- [KOPE79] KOPETZ, H. *Software reliability*, Springer-Verlag, New York, 1979.
- [LIN85] LIN, H. Forum. *Technological Review* 88, 5 (July, 1985), 16-18.
- [LEAT83] LEATHRUM, J. F. *Foundations of Software Design*, Reston Publishing, Inc., Reston, VA, 1983.
- [LEVE82] LEVESON, N. G. Software safety. *ACM SIGSOFT Software Engineering Notes* 7, 2 (April, 1982), 21-24.
- [LIPO74] LIPOW, M. *Some Variations of a Model for Software Time-to-Failure*, TRW Systems Group, Redondo Beach, CA, 1974, pp. 9-21.
- [LIPO79] LIPOW, M. Prediction of software errors. *Journal of Systems and Software*, 1 (1979), 71-75.
- [LITT73] LITTLEWOOD, B., & VERRALL, J.L. A Bayesian reliability growth model for computer software. *Journal of the Royal Statistical Society (series C), Applied Statistics* 22, 3 (1973), 146-152.
- [LITT75A] LITTLEWOOD, B. A reliability model for Markov structured software. *IEEE Conference on Reliable Software* (1975), 204-207, IEEE, New York.
- [LITT75B] LITTLEWOOD, B. Correspondence. *IEEE Transactions on Reliability* 24 1 (April, 1975), p. 82.
- [LITT78] LITTLEWOOD, B. Validation of a software reliability model. *2nd Software Life Cycle Management Workshop*, August 1978, p. 146-152.

- [LITT79A] LITTLEWOOD, B. A Bayesian differential model for software reliability. *Workshop on Quantitative Software Models*, IEEE Cat. No. TH0067-9 (1979), 170–181, IEEE, New York.
- [LITT80] LITTLEWOOD, B. Theories of software reliability: How good are they and how can they be improved?. *IEEE Transactions on Software Engineering SE-6*, 5 (September, 1980), IEEE, New York.
- [LITT85] LITTLEWOOD, B., GHALY, A.A.A., & CHAN, P.Y. Evaluation of competing software reliability predictions. *CSR Research Report* (March, 1985), Centre for Software Reliability, The City University, Northampton Square, London.
- [LUND40] LUNDBERG, O. *On Random Processes and Their Applications to Sickness and Accident Statistics*, Uppsala, 1940.
- [MCCA76] MCCABE, T. A software complexity measure. *IEEE Transactions on Software Engineering 2* (December, 1976), 308–320.
- [MCGE82] MCGETTRICK, A. D. *Program Verification using Ada*, University Press, Cambridge, England, 1982, pp. 11–12.
- [MEND81] MENDENHALL, *et al.* *Mathematical Statistics with Applications*, 2nd ed., Duxbury Press, Boston, 1981, p. 101.
- [MILL72] MILLS, H. D. *On the statistical evaluation of computer programs*, IBM Federal Systems Div. Report FSC-72-6015, Gaithersburg, MD, 1972.
- [MOHA73] MOHANLY, S.N. Models and measurements for quality assessment of software. *ACM Computing Surveys 11* (September, 1973), 250–273.
- [MORA75] MORANDA, P. *Probability Based Models for the Failures during Burn-In Phase*, Joint National Meeting ORSA/TIMS, Las Vegas, 1975.
- [MURT72] MURTY, V. K., & SWARTZ, B. *Annotated Bibliography on Cumulative Fatigue Damage and Structural Reliability Models*, ARL Report, ARL 72-0161, United States Air Force, Wright-Patterson AFB, OH, 1972.

- [MURT74] MURTY, V. K. *The General Point Process: Applications to Structural Fatigue, Bioscience, and Medical Research*, Addison-Wesley Publishing, Reading, MA, 1974, p. 239.
- [MUSA75] MUSA, J.D. A Theory of software reliability and its applications. *IEEE Transactions on Software Engineering SE-1* (September, 1975), 312–327.
- [MUSA79A] MUSA, J. D. Validation of execution-time theory of software reliability. *IEEE Transactions on Reliability R-28*, 3 (August, 1979), 181–191.
- [MUSA79B] MUSA, J. D. *Software Reliability Data*, Report available from Data and Analysis Center for Software (DACS), Rome Air Development Center, Griffiss AFB, NY, 1979.
- [MUSA84] MUSA, J.D. Software reliability. In *Handbook of Software Engineering*, Vick C.R., & Ramamoorthy, C.V., Ed., Van Nostrand Reinhold, New York, 1984.
- [NELS73] NELSON, E. C. *A Statistical Basis for Software Reliability Assessment*, TRW, Redondo Beach, CA, 1973.
- [OCON81] O'CONNOR, P. D. T. *Practical Reliability Engineering*, Heyden & Son, London, 1981.
- [RADA81] RADATZ, J. W. *Analysis of IV& V Data*, Report RiSED-81319, Logicon Inc., San Pedro, CA, 1981.
- [RAMA84] RAMAMOORTHY, *et al.* Software engineering: problems and perspectives. *Computer* 17, 10 (October, 1984), 191–209, IEEE Computer Society, Los Alamitos, CA.
- [ROSS70] ROSS, S. *Applied Probability Models with Optimization Applications*, Holden-Day, San Francisco, 1970.
- [SCH178] SCHICK, G., & WOLVERTON, R. An analysis of competing software reliability models. *IEEE Transactions on Software Engineering SE-4* (March, 1978), 104–120, IEEE, New York.
- [SCH182] SCHINDLER, M. Software testing—a scarce art struggles to become a science. *Electronic Design* (July 22, 1982), 85–102.

- [SCHN75] SCHNEIDEWIND, N. F. Analysis of error processes in software. *Proceedings of the International Conference on Reliable Software*, 337-346, IEEE, New York.
- [SCHN79] SCHNEIDEWIND, N. F. The applicability of hardware reliability principles to computer software. In *Software Quality Management*, Cooper, J. D., & Fisher, M. J., Ed., Petrocelli Books, New York, 1979.
- [SHOO73] SHOOMAN, M. L. Operational testing and software reliability estimation during program development. *1973 IEEE Symposium on Computer Software Reliability*, IEEE Cat. No. 73CH0741-9CSR (1973), 51-57, IEEE, New York.
- [SHOO75] SHOOMAN, M. L. Software reliability: Measurement and models. *Proceedings of the 1975 Annual Reliability and Maintainability Symposium*, IEEE Cat. No. 75CH0918-3RQC (1975), 485-491, IEEE, New York.
- [SHOO83] SHOOMAN, M. *Software Engineering: Design, Reliability and Management*, McGraw-Hill, New York, 1983, pp. 297-298.
- [SPEC84] SPECTOR, A., & GIFFORD, D. Case study: the space shuttle primary computer system. *Communications of the ACM* 27, 9 (September, 1984), 872-900.
- [THAY78] THAYER, T.A., LIPOW, M., & NELSON, E.C. *Software Reliability*, North-Holland, New York, 1978.
- [THOM79] THOMPSON, W. E., & CHELSON, P. O. Software reliability testing for embedded computer systems. *Workshop on Quantitative Software Models*, IEEE Cat. No. TH0067-9 (October, 1979), 201-208.
- [THOM80] THOMPSON, W. E., & CHELSON, P. O. On the specification and testing of software reliability. *Proceedings of the 1980 Annual Reliability and Maintainability Symposium*, IEEE Cat. No. 80CH1513-R, 379-383, IEEE, New York.
- [TRIV74] TRIVEDI, A., & SHOOMAN, M. *A Markov Model for the Evaluation of Computer Software Performance*, Research report, EE/EP 74-0111-EER110, under contract to ONR and RADC, Polytechnic Institute of New York, Brooklyn, NY, 1974.

- [TRIV75] TRIVEDI, A., & SHOOMAN, M. A many-state Markov model for the estimation and prediction of computer software performance parameters. *Proceedings of the 1975 International Conference on Reliable Software*, IEEE Cat. No. 75CH0940-7CSR, 208-220, IEEE, New York.
- [WAGO73] WAGONER, W. L. *The Final Report on a Software Reliability Measurement Study*, Report No. TOR-0074 (4112)-1, The Aerospace Corporation, 1973.
- [WEIB39] WEIBULL, W. A statistical theory of the strength of material. *Ing. Vetenskaps Akad. Handl 151* (1939), 72-93.
- [WIRT73] WIRTH, N. *Systematic Programming: An Introduction*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1973, p. 16.

Appendix A

Data for Bayesian Failure Rate Updating Example

Contained in this appendix is the failure data used in the example of the Bayesian technique for failure rate updating. The data concerns a single program version (frozen code). The "DATE" column gives the number of days since the start of program testing. The "DELTA DATE" column gives the number of days since the preceding date. The "PENALTY COST" column gives the severity rating of the data and is not important for this particular application. Each row documents a software failure that occurred.

The example starts out with an assumed failure rate and successively revises the estimate as more and more historical data is revealed and taken into account.

The second table, with column headers "DAY" and "LAMBDA," shows the running best estimate of the failure rate for each successive day. The data points in this table are what is plotted in Figure 22.

Table 1: *Raw Data for Bayesian Updating Example*

DATE	DELTA DATE	PENALTY COST	DATE	DELTA DATE	PENALTY COST
0000	0000	3	0231	0000	1
0159	0159	1	0235	0004	3
0159	0000	1	0236	0001	2
0159	0000	1	0236	0000	2
0159	0000	1	0238	0002	2
0159	0000	1	0238	0000	2
0159	0000	1	0238	0000	2
0159	0000	1	0238	0000	2
0159	0000	1	0241	0003	1
0164	0005	1	0244	0003	1
0164	0000	1	0245	0001	2
0164	0000	1	0245	0000	2
0164	0000	1	0246	0001	1
0206	0042	2	0246	0000	1
0206	0000	2	0251	0005	1
0206	0000	2	0251	0000	1
0218	0012	3	0262	0011	3
0227	0009	3	0264	0002	2
0229	0002	1	0270	0006	3
0229	0000	1	0278	0008	2
0229	0000	1	0281	0003	2
0229	0000	1	0294	0013	3
0229	0000	2	0294	0000	1
0229	0000	2	0294	0000	1
0229	0000	2	0294	0000	1
0231	0002	3	0294	0000	1
0231	0000	3	0294	0000	1

DATE	DELTA DATE	PENALTY COST	DATE	DELTA DATE	PENALTY COST
0294	0000	1	0348	0000	2
0294	0000	1	0367	0019	2
0294	0000	1	0389	0022	1
0294	0000	2	0389	0000	2
0294	0000	2	0395	0006	1
0300	0006	1	0395	0000	2
0301	0001	2	0395	0000	2
0301	0000	2	0396	0001	1
0304	0003	3	0409	0013	1
0304	0000	1	0409	0000	2
0305	0001	2	0409	0000	2
0305	0000	2	0411	0002	3
0305	0000	2	0419	0008	3
0307	0002	2	0419	0000	3
0307	0000	2	0419	0000	2
0308	0001	1	0424	0005	2
0318	0010	3	0427	0003	3
0319	0001	2	0428	0001	3
0322	0003	2	0428	0000	3
0326	0004	3	0430	0002	1
0333	0007	3	0430	0000	1
0333	0000	1	0430	0000	2
0334	0001	3	0433	0003	2
0334	0000	2	0434	0001	2
0342	0008	1	0435	0001	2
0347	0005	2	0440	0005	1
0348	0001	2	0446	0006	1

DATE	DELTA DATE	PENALTY COST	DATE	DELTA DATE	PENALTY COST
0446	0000	1	0489	0000	2
0446	0000	1	0489	0000	2
0446	0000	1	0493	0004	2
0447	0001	1	0493	0000	2
0454	0007	3	0495	0002	2
0455	0001	3	0496	0001	2
0455	0000	3	0502	0006	1
0468	0013	3	0503	0001	1
0468	0000	3	0503	0000	2
0468	0000	1	0503	0000	2
0468	0000	2	0503	0000	2
0473	0005	2	0503	0000	2
0474	0001	2	0508	0005	3
0475	0001	3	0508	0000	2
0475	0000	2	0510	0002	1
0476	0001	2	0511	0001	2
0480	0004	3	0511	0000	2
0486	0006	2	0512	0001	1
0486	0000	2	0514	0002	2
0487	0001	2	0515	0001	3
0487	0000	2	0515	0000	1
0488	0001	2	0515	0000	2
0489	0001	2	0515	0000	2
0489	0000	2	0515	0000	2
0489	0000	2	0521	0006	1
0489	0000	2	0521	0000	2
0489	0000	2	0522	0001	3
0489	0000	2	0524	0002	1

DATE	DELTA DATE	PENALTY COST	DATE	DELTA DATE	PENALTY COST
0528	0004	3	0549	0000	2
0529	0001	1	0549	0000	2
0529	0000	1	0552	0003	3
0529	0000	2	0553	0001	2
0532	0003	3	0553	0000	2
0532	0000	2	0553	0000	2
0535	0003	2	0556	0003	1
0535	0000	2	0556	0000	1
0535	0000	2	0556	0000	1
0535	0000	2	0563	0007	2
0535	0000	2	0563	0000	2
0535	0000	2	0563	0000	2
0535	0000	2	0567	0004	3
0536	0001	1	0567	0000	2
0536	0000	1	0567	0000	2
0536	0000	1	0571	0004	2
0538	0002	3	0573	0002	3
0538	0000	1	0574	0001	1
0539	0001	2	0578	0004	3
0542	0003	1	0578	0000	2
0542	0000	1	0578	0000	2
0544	0002	3	0579	0001	2
0545	0001	3	0579	0000	2
0546	0001	1	0591	0012	3
0546	0000	2	0592	0001	2
0547	0001	3	0592	0000	1
0549	0002	2	0602	0010	1

DATE	DELTA DATE	PENALTY COST	DATE	DELTA DATE	PENALTY COST
0609	0007	1	0685	0000	2
0629	0020	2	0686	0001	3
0630	0001	1	0686	0000	1
0635	0005	2	0686	0000	1
0635	0000	2	0686	0000	2
0636	0001	2	0686	0000	2
0637	0001	1			
0644	0007	1			
0647	0003	2			
0649	0002	3			
0662	0013	3			
0662	0000	1			
0662	0000	1			
0662	0000	1			
0662	0000	1			
0662	0000	2			
0662	0000	1			
0664	0002	1			
0665	0001	1			
0670	0005	1			
0671	0001	1			
0671	0000	1			
0671	0000	1			
0671	0000	1			
0671	0000	1			
0671	0000	1			
0676	0005	1			
0685	0009	1			

Table 2: *Best Estimate of Failure Rate*

DAY	LAMBDA	DAY	LAMBDA
0347	0.008000	0440	0.019491
0348	0.008991	0446	0.019176
0348	0.010967	0446	0.018871
0367	0.013659	0446	0.018579
0389	0.016529	0446	0.018299
0389	0.019247	0447	0.018032
0395	0.021497	0454	0.017773
0395	0.023211	0455	0.017522
0395	0.024431	0455	0.017281
0396	0.025238	0468	0.017043
0409	0.025599	0468	0.016811
0409	0.025659	0468	0.016584
0409	0.025527	0468	0.016365
0411	0.025268	0473	0.016150
0419	0.024895	0474	0.015941
0419	0.024465	0475	0.015739
0419	0.024012	0475	0.015542
0424	0.023541	0475	0.015352
0427	0.023064	0476	0.015168
0428	0.022592	0480	0.014989
0428	0.022136	0486	0.014814
0430	0.021698	0486	0.014645
0430	0.021280	0487	0.014479
0430	0.020885	0487	0.014319
0433	0.020510	0488	0.014163
0434	0.020154	0489	0.014011
0435	0.019816	0489	0.013864

DAY	LAMEDA	DAY	LAMEDA
0489	0.013722	0522	0.011017
0489	0.013583	0524	0.010945
0489	0.013449	0528	0.010874
0489	0.013320	0529	0.010804
0493	0.013194	0529	0.010736
0493	0.013072	0529	0.010669
0495	0.012953	0532	0.010603
0496	0.012837	0532	0.010538
0502	0.012725	0535	0.010474
0503	0.012614	0535	0.010411
0503	0.012507	0535	0.010350
0503	0.012402	0535	0.010289
0503	0.012300	0535	0.010229
0503	0.012200	0535	0.010171
0508	0.012103	0535	0.010113
0508	0.012008	0536	0.010057
0510	0.011915	0536	0.010001
0511	0.011824	0536	0.009947
0511	0.011735	0538	0.009893
0512	0.011648	0538	0.009841
0514	0.011563	0539	0.009789
0515	0.011480	0542	0.009738
0515	0.011399	0542	0.009688
0515	0.011319	0544	0.009639
0515	0.011241	0545	0.009590
0521	0.011165	0546	0.009542
0521	0.011090	0546	0.009495

DAY	LAMBDA	DAY	LAMBDA
0547	0.009449	0602	0.008403
0549	0.009403	0609	0.008369
0549	0.009358	0629	0.008336
0549	0.009314	0630	0.008302
0552	0.009270	0635	0.008269
0553	0.009227	0635	0.008235
0553	0.009185	0636	0.008202
0553	0.009143	0637	0.008169
0556	0.009102	0644	0.008136
0556	0.009061	0647	0.008102
0556	0.009021	0649	0.008069
0563	0.008981	0662	0.008036
0563	0.008942	0662	0.008003
0563	0.008903	0662	0.007970
0567	0.008865	0662	0.007937
0567	0.008827	0662	0.007905
0567	0.008790	0662	0.007872
0571	0.008753	0662	0.007839
0573	0.008717	0664	0.007807
0574	0.008680	0665	0.007774
0578	0.008645	0670	0.007742
0578	0.008609	0671	0.007710
0579	0.008574	0671	0.007678
0579	0.008539	0671	0.007646
0591	0.008505	0671	0.007615
0592	0.008471	0671	0.007583
0592	0.008437	0676	0.007552

DAY	LAMBDA
0685	0.007521
0685	0.007489
0686	0.007459
0686	0.007428
0686	0.007397
0686	0.007367
0686	0.007337

Appendix B

Data for Robust Program Example

The data in this appendix is used in the example showing the application of the model to a hypothetical robust program. The data concerns a program in which the fault causing a failure is removed once discovered. Thus each row refers to a different program version (unlike the data in Appendix A). The "SECONDS" column gives cumulative CPU seconds (ignoring repair activity time). The "PENALTY COST" column is the severity rating of the failure.

The example borrows its failure-rate formulation from the Jelinsky-Moranda model. The unknown parameters are statistically estimated via the method of maximum likelihood.

Table 3: *Raw Data for Robust Program Example*

SECONDS	PENALTY COST	SECONDS	PENALTY COST
3.	2.6	1.146.	12.4
30.	53.6	600.	5.3
113.	22.2	15.	3.4
81.	44.3	36.	12.6
115.	16.3	4.	47.7
9.	19.5	0.	13.7
2.	31.6	8.	10.6
91.	7.0	227.	2.4
112.	13.5	65.	37.6
15.	33.5	176.	11.1
138.	45.4	58.	18.5
50.	10.7	457.	6.9
77.	4.9	300.	123.6
24.	23.7	97.	47.8
108.	8.8	263.	3.4
88.	4.4	452.	54.2
670.	12.5	255.	42.7
120.	41.2	197.	1.6
26.	40.5	193.	42.3
114.	0.7	6.	10.7
325.	0.4	79.	32.0
55.	8.7	816.	63.8
242.	3.6	1351.	27.7
68.	38.4	148.	12.4
422.	28.5	21.	9.0
180.	31.3	233.	11.2
10.	4.5	134.	0.9

SECONDS	PENALTY COST	SECONDS	PENALTY COST
357.	36.4	296.	39.7
193.	1.4	1755.	23.1
236.	6.7	1064.	58.5
31.	29.2	1783.	50.6
369.	5.3	860.	60.9
748.	13.7	983.	182.2
0.	9.1	707.	6.4
232.	75.1	33.	24.7
330.	30.2	868.	8.0
365.	41.2	724.	2.5
1222.	1.3	2323.	38.1
543.	33.0	2930.	7.9
10.	13.8	1461.	110.9
16.	22.7	843.	10.9
529.	1.8	12.	8.3
379.	11.9	261.	3.2
44.	56.1	1800.	43.4
129.	58.4	865.	22.8
810.	27.7	1435.	0.2
290.	97.5	30.	17.0
300.	2.9	143.	22.1
529.	30.4	109.	11.2
281.	159.9	0.	18.1
160.	6.5	3110.	19.3
828.	35.6	1247.	25.3
1011.	15.7	943.	35.0
445.	25.1	700.	11.6

SECONDS PENALTY COST

4116. 54.4

SECONDS PENALTY COST

875.	3.8
245.	42.0
729.	18.2
1897.	1.0
447.	4.4
386.	62.7
446.	1.2
122.	1.4
990.	13.1
949.	3.1
1082.	25.2
22.	44.4
75.	53.4
482.	18.5
5509.	14.3
100.	31.4
10.	155.5
1071.	37.1
371.	1.9
790.	30.5
6150.	15.5
3321.	40.7
1045.	11.5
648.	24.4
5485.	61.5
1160.	29.6
1864.	37.7

Appendix C

Data for Space Shuttle Example

The data in this appendix summarizes the data used in the space shuttle example. The "WEEK" column gives the week of testing the row refers to. The "TEST HOURS" column tells how many hours were spent in testing during that week. The "CRITICAL FAILURES," "MAJOR FAILURES" and "MINOR FAILURES" columns give the number of software failures that fell into each severity class. The last row refers to the STS-4 mission itself, and so represents operational as opposed to test phase data. Note that, as in Appendix B, fault correction is taking place: The code is not frozen.

Also contained in this appendix are plots and tabulations of the aggregate penalty cost probability mass function and cumulative distribution function obtained from application of the model. Figure 38 shows the aggregate penalty cost probability mass function. Figure 39 shows the aggregate penalty cost cumulative distribution function.

Table 4: *Raw Data for Space Shuttle Example*

WEEK	TEST HOURS	CRITICAL FAILURES	MAJOR FAILURES	MINOR FAILURES	WEEK	TEST HOURS	CRITICAL FAILURES	MAJOR FAILURES	MINOR FAILURES
1	62.5	0	6	9	28	52.0	0	2	2
2	44.0	0	2	4	29	70.0	0	1	3
3	40.0	0	1	7	30	84.5	0	2	6
4	68.0	1	1	6	31	83.0	1	2	3
5	62.0	0	3	5	32	60.0	1	0	1
6	66.0	0	1	3	33	72.5	0	2	1
7	73.0	0	2	2	34	90.0	0	2	4
8	73.5	0	3	5	35	58.0	0	3	3
9	92.0	0	2	4	36	60.0	0	1	2
10	71.4	0	0	2	37	168.0	1	2	11
11	64.5	0	3	4	38	111.5	0	1	9
12	64.7	0	1	7	STS-4	200.0	(0)	5	9
13	36.0	0	3	0					
14	54.0	0	0	5					
15	39.5	0	2	3					
16	68.0	0	5	3					
17	61.0	0	5	3					
18	62.6	0	2	4					
19	98.7	0	2	10					
20	25.0	0	2	3					
21	12.0	0	1	1					
22	55.0	0	3	2					
23	49.0	0	2	4					
24	64.0	0	4	5					
25	26.0	0	1	0					
26	66.0	0	2	2					
27	49.0	0	2	0					

Table 5: *Aggregate Penalty Cost PMF*

AGGREGATE PENALTY COST PME (con't)

AGGREGATE PENALTY COST PME

f(0)	=	0.000001
f(1)	=	0.000000
f(2)	=	0.000006
f(3)	=	0.000011
f(4)	=	0.000014
f(5)	=	0.000052
f(6)	=	0.000069
f(7)	=	0.000126
f(8)	=	0.000254
f(9)	=	0.000340
f(10)	=	0.000579
f(11)	=	0.000907
f(12)	=	0.001214
f(13)	=	0.001850
f(14)	=	0.002545
f(15)	=	0.003327
f(16)	=	0.004586
f(17)	=	0.005840
f(18)	=	0.007338
f(19)	=	0.009334
f(20)	=	0.011243
f(21)	=	0.013508
f(22)	=	0.016120
f(23)	=	0.018558
f(24)	=	0.021323
f(25)	=	0.024163
f(26)	=	0.025731
f(27)	=	0.029454
f(28)	=	0.031958
f(29)	=	0.034096
f(30)	=	0.036154
f(31)	=	0.037779
f(32)	=	0.038979
f(33)	=	0.039914
f(34)	=	0.040338
f(35)	=	0.040345
f(36)	=	0.040020
f(37)	=	0.039240
f(38)	=	0.038129
f(39)	=	0.036737
f(40)	=	0.035036
f(41)	=	0.033139
f(42)	=	0.031085
f(43)	=	0.028894
f(44)	=	0.026650
f(45)	=	0.024385
f(46)	=	0.022129
f(47)	=	0.019936
f(48)	=	0.017824
f(49)	=	0.015816
f(50)	=	0.013937
f(51)	=	0.012193
f(52)	=	0.010593
f(53)	=	0.009142

AGGREGATE PENALTY COST FME (con't)

f(81) = 0.000014
 f(82) = 0.000010
 f(83) = 0.000008
 f(84) = 0.000006
 f(85) = 0.000004
 f(86) = 0.000003
 f(87) = 0.000002
 f(88) = 0.000002
 f(89) = 0.000001
 f(90) = 0.000001
 f(91) = 0.000001

AGGREGATE PENALTY COST FME (con't)

f(54) = 0.007837
 f(55) = 0.006674
 f(56) = 0.005647
 f(57) = 0.004749
 f(58) = 0.003968
 f(59) = 0.003296
 f(60) = 0.002721
 f(61) = 0.002233
 f(62) = 0.001822
 f(63) = 0.001479
 f(64) = 0.001193
 f(65) = 0.000957
 f(66) = 0.000764
 f(67) = 0.000606
 f(68) = 0.000479
 f(69) = 0.000376
 f(70) = 0.000294
 f(71) = 0.000229
 f(72) = 0.000177
 f(73) = 0.000136
 f(74) = 0.000104
 f(75) = 0.000080
 f(76) = 0.000060
 f(77) = 0.000046
 f(78) = 0.000034
 f(79) = 0.000026
 f(80) = 0.000019

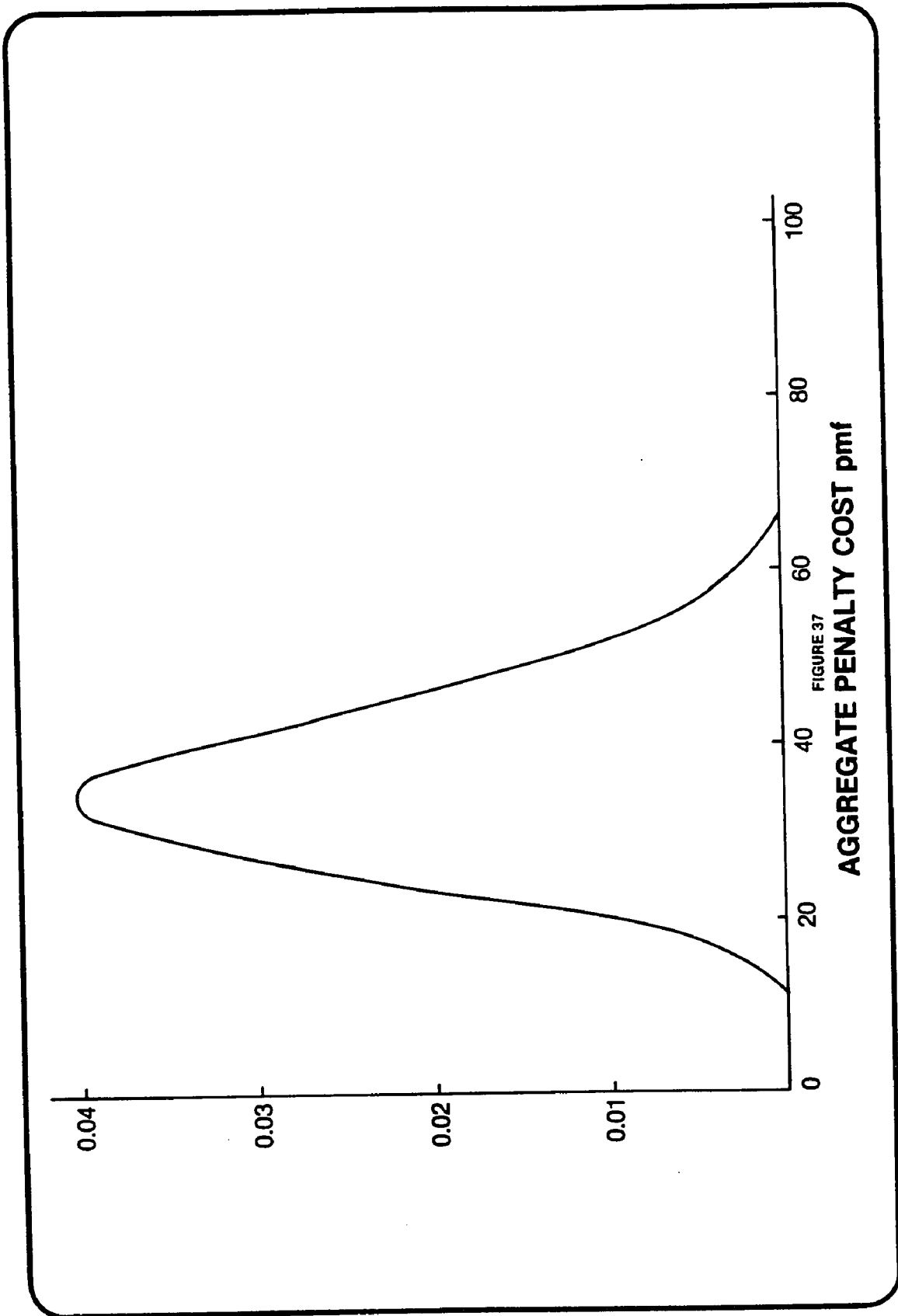


FIGURE 37

Table 6: *Aggregate Penalty Cost CDF*

AGGREGATE PENALTY COST CDF (con't)

F (0) = 0.000000
 F (1) = 0.000001
 F (2) = 0.000001
 F (3) = 0.000007
 F (4) = 0.000018
 F (5) = 0.000032
 F (6) = 0.000084
 F (7) = 0.000153
 F (8) = 0.000279
 F (9) = 0.000533
 F (10) = 0.000872
 F (11) = 0.001451
 F (12) = 0.002358
 F (13) = 0.003571
 F (14) = 0.005421
 F (15) = 0.007966
 F (16) = 0.011294
 F (17) = 0.015880
 F (18) = 0.021720
 F (19) = 0.029058
 F (20) = 0.038392
 F (21) = 0.049635
 F (22) = 0.063142
 F (23) = 0.079263
 F (24) = 0.097820
 F (25) = 0.119143
 F (26) = 0.143306
 F (27) = 0.170037
 F (28) = 0.199492
 F (29) = 0.231449
 F (30) = 0.265545
 F (31) = 0.301699
 F (32) = 0.339478
 F (33) = 0.378457
 F (34) = 0.418371
 F (35) = 0.458708
 F (36) = 0.499053
 F (37) = 0.539073
 F (38) = 0.578314
 F (39) = 0.616442
 F (40) = 0.653180
 F (41) = 0.688216
 F (42) = 0.721354
 F (43) = 0.752440
 F (44) = 0.781333
 F (45) = 0.807983
 F (46) = 0.832368
 F (47) = 0.854498
 F (48) = 0.874434
 F (49) = 0.892259
 F (50) = 0.908075
 F (51) = 0.922012
 F (52) = 0.934205
 F (53) = 0.944798

AGGREGATE PENALTY COST CDF

F (0) = 0.000000
 F (1) = 0.000001
 F (2) = 0.000001
 F (3) = 0.000007
 F (4) = 0.000018
 F (5) = 0.000032
 F (6) = 0.000084
 F (7) = 0.000153
 F (8) = 0.000279
 F (9) = 0.000533
 F (10) = 0.000872
 F (11) = 0.001451
 F (12) = 0.002358
 F (13) = 0.003571
 F (14) = 0.005421
 F (15) = 0.007966
 F (16) = 0.011294
 F (17) = 0.015880
 F (18) = 0.021720
 F (19) = 0.029058
 F (20) = 0.038392
 F (21) = 0.049635
 F (22) = 0.063142
 F (23) = 0.079263
 F (24) = 0.097820
 F (25) = 0.119143
 F (26) = 0.143306

AGGREGATE PENALTY COST CDF (con't)

F (54) = 0.953941
 F (55) = 0.961777
 F (56) = 0.968451
 F (57) = 0.974099
 F (58) = 0.978847
 F (59) = 0.982815
 F (60) = 0.986111
 F (61) = 0.988832
 F (62) = 0.991066
 F (63) = 0.992888
 F (64) = 0.994367
 F (65) = 0.995560
 F (66) = 0.996517
 F (67) = 0.997281
 F (68) = 0.997888
 F (69) = 0.998367
 F (70) = 0.998743
 F (71) = 0.999037
 F (72) = 0.999265
 F (73) = 0.999442
 F (74) = 0.999578
 F (75) = 0.999683
 F (76) = 0.999762
 F (77) = 0.999823
 F (78) = 0.999868
 F (79) = 0.999902
 F (80) = 0.999928

AGGREGATE PENALTY COST CDF

F (81) = 0.999947
 F (82) = 0.999961
 F (83) = 0.999972
 F (84) = 0.999980
 F (85) = 0.999985
 F (86) = 0.999989
 F (87) = 0.999992
 F (88) = 0.999995
 F (89) = 0.999996
 F (90) = 0.999997
 F (91) = 0.999998

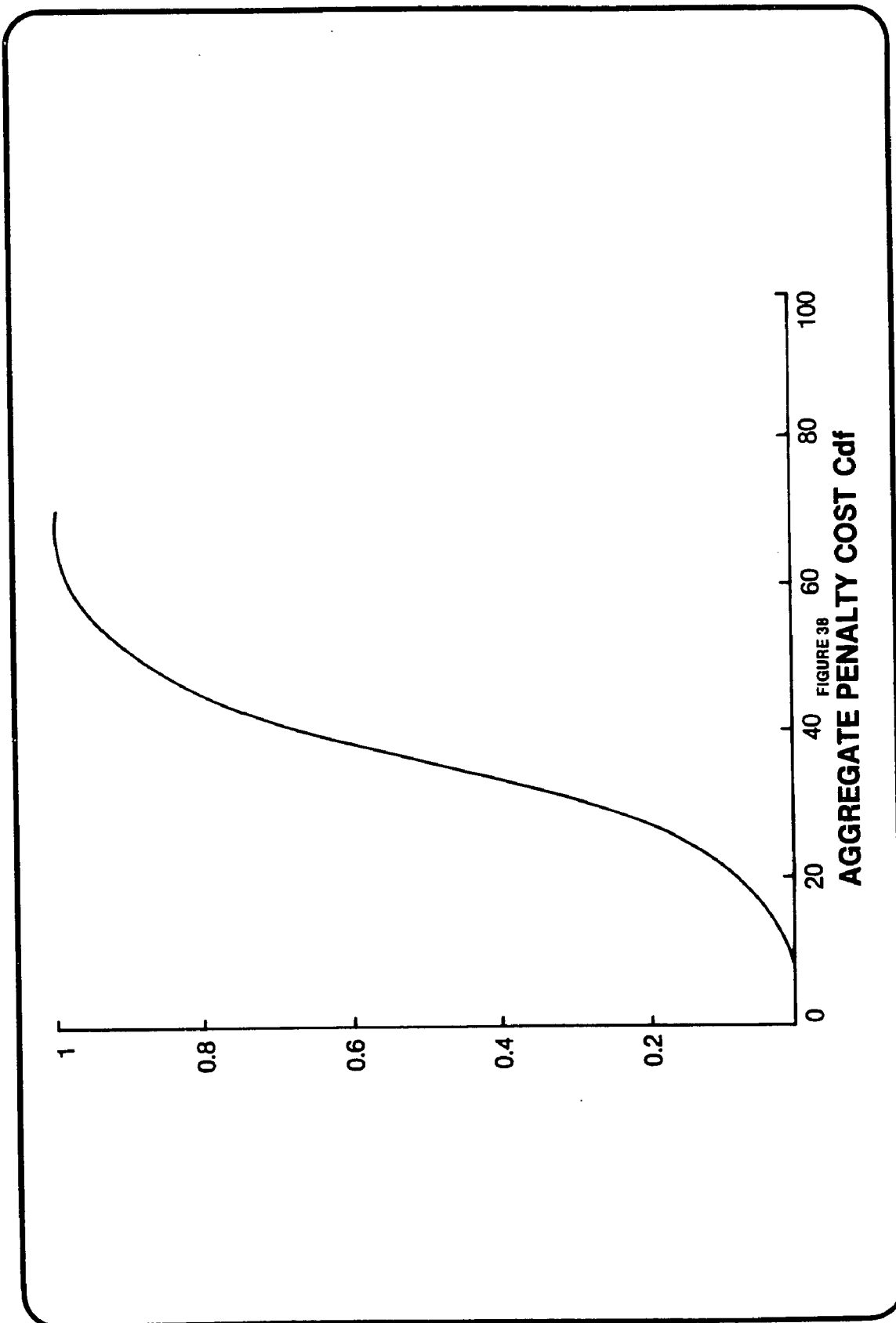


FIGURE 3B

Appendix D

“Flow Diagrams”

Figure 39 shows a typical time-domain model. The debugging times τ are used to statistically infer the values of the unknown parameters in the failure rate formulation. The failure rate $\lambda(t)$ might be a constant, a deterministic time-dependent function, or a random variable. The failure rate feeds into a stochastic process that probabilistically characterizes the operational interfailure times. From the probability distribution of times-to-failure a number of useful figures of merit such as MTTF and percentiles are derived.

Figure 40 show the same typical time-domain model augmented by penalty costs. The penalty cost values x are used to estimate the parameters of an underlying individual penalty cost distribution. The original stochastic process and the penalty cost distribution are combined to create a compound stochastic process. From the resulting aggregate penalty cost distribution, figures of merit such as mean aggregate penalty cost and percentiles are derived.

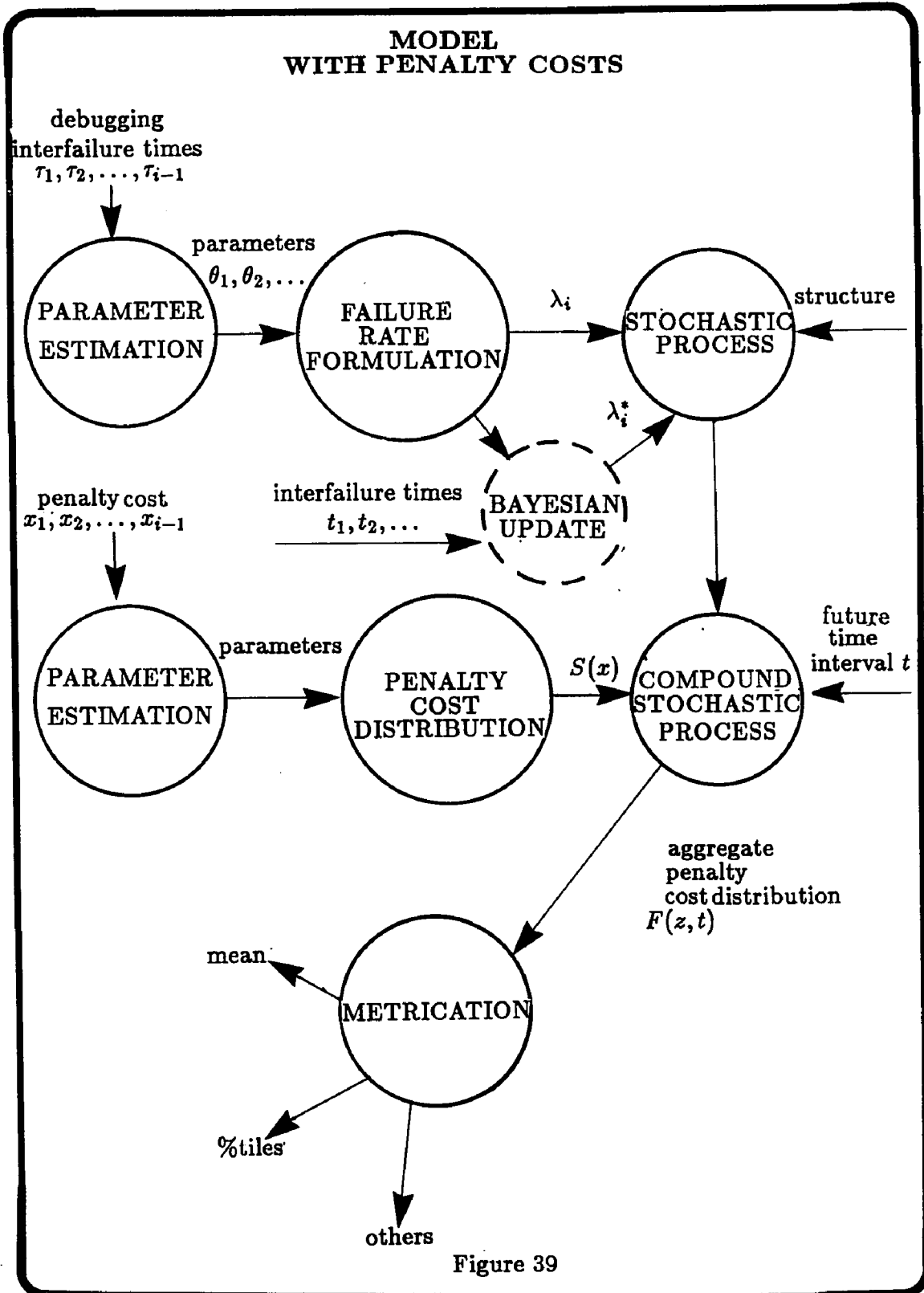


Figure 39

TYPICAL TIME-DOMAIN MODEL

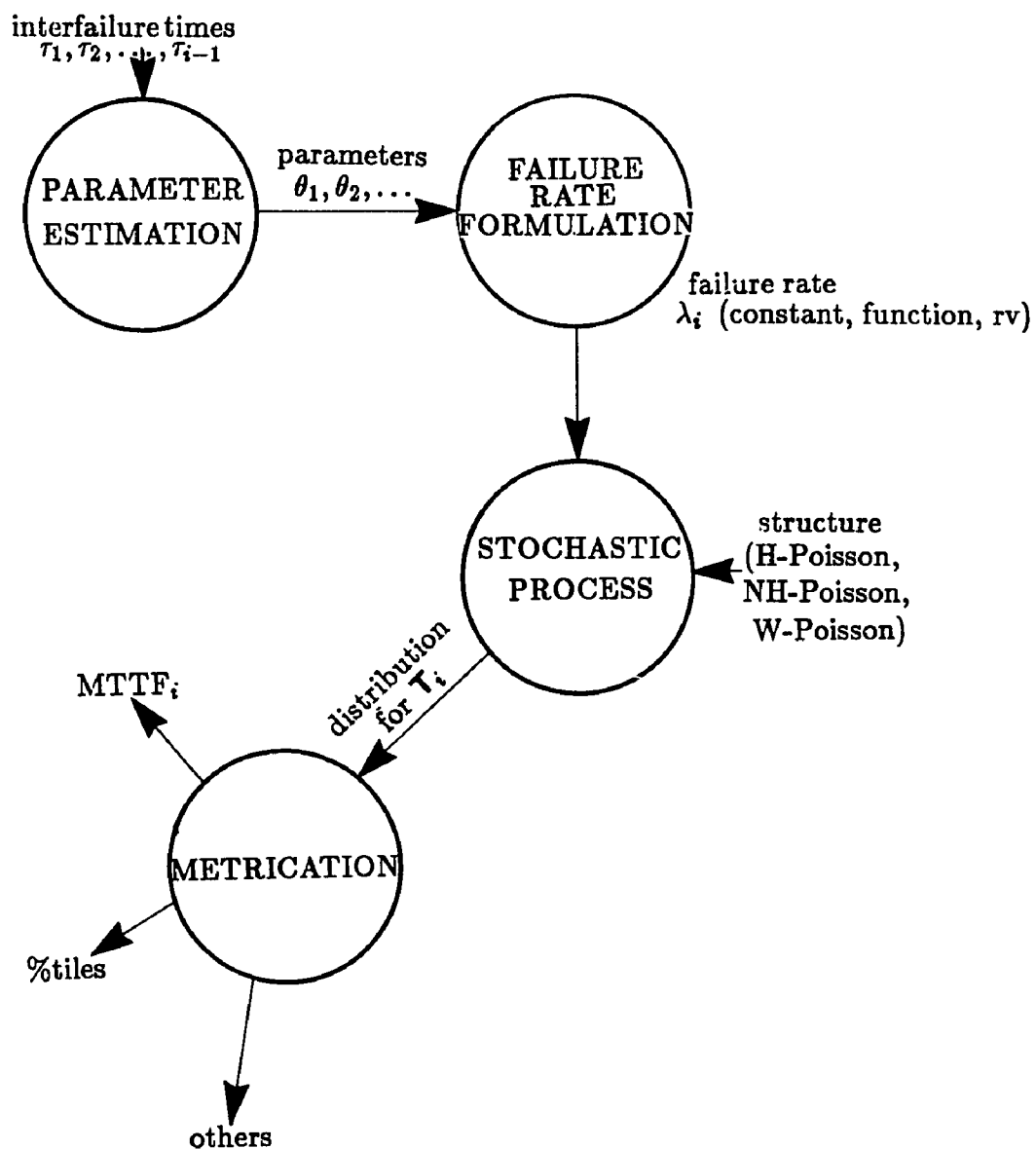


Figure 40