

Springer Series in Reliability Engineering

**Hoang Pham**

# **System Software Reliability**



**Springer**

# Springer Series in Reliability Engineering

---

## **Series Editor**

Professor Hoang Pham  
Department of Industrial Engineering  
Rutgers  
The State University of New Jersey  
96 Frelinghuysen Road  
Piscataway, NJ 08854-8018  
USA

## **Other titles in this series**

*The Universal Generating Function in Reliability Analysis and Optimization*  
Gregory Levitin

*Warranty Management and Product Manufacture*  
D.N.P Murthy and Wallace R. Blischke

*Maintenance Theory of Reliability*  
Toshio Nakagawa

*Reliability and Optimal Maintenance*  
Hongzhou Wang and Hoang Pham

Hoang Pham

---

# System Software Reliability

With 63 Figures

Hoang Pham, PhD  
Department of Industrial Engineering  
Rutgers, The State University of New Jersey  
96 Freylinghuysen Road  
Piscataway, New Jersey 08854-8018  
USA

British Library Cataloguing in Publication Data

Pham, Hoang

System software reliability. - (Springer series in  
reliability engineering)

1.Systems software 2.Computer software - Reliability

I.Title

005.4'3

ISBN-10: 1852339500

Library of Congress Control Number: 2006924634

Springer Series in Reliability Engineering series ISSN 1614-7839

ISBN-10: 1-85233-950-0

e-ISBN 1-84628-295-0

Printed on acid-free paper

ISBN-13: 978-1-85233-950-0

© Springer-Verlag London Limited 2006

Apart from any fair dealing for the purposes of research or private study, or criticism or review, as permitted under the Copyright, Designs and Patents Act 1988, this publication may only be reproduced, stored or transmitted, in any form or by any means, with the prior permission in writing of the publishers, or in the case of reprographic reproduction in accordance with the terms of licences issued by the Copyright Licensing Agency. Enquiries concerning reproduction outside those terms should be sent to the publishers.

The use of registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant laws and regulations and therefore free for general use.

The publisher makes no representation, express or implied, with regard to the accuracy of the information contained in this book and cannot accept any legal responsibility or liability for any errors or omissions that may be made.

9 8 7 6 5 4 3 2 1

Springer Science+Business Media  
springer.com

*To my parents  
Phong Pham and Tam Huynh,  
for many reasons.*

---

## Preface

In today's technological world nearly everyone depends upon the continued functioning of a wide array of complex machinery and equipment for our everyday safety, security, mobility and economic welfare. We expect our electric appliances, hospital monitoring control, next-generation aircraft, data exchange systems, and aerospace applications to function wherever and whenever we need them. When they fail, the results can be catastrophic. As our society grows in complexity, so do the critical challenges in the area of system and software reliability engineering.

In general, system software reliability is the probability that the system will not fail for a specified period of time under specified conditions. The greatest problem facing the industry today is how to assess quantitatively system reliability characteristics.

The author published the book *Software Reliability* in 2000. Due to the critical challenges and complexity of modern embedded-software systems developed over the last five years, there has arisen an ever increasing attention of both public and professional communities to look for products with high reliability at reasonable costs. At the same time, the techniques, tools and models available in the last five years to system designers, engineers, and practitioners have continued to be developed by scientists and researchers at the same rate.

This book aims to present the state-of-the-art of system software reliability in theory and practice and recent research on this subject over the last five years. It is a textbook based mainly on the author's recent research and publications as well as experience of 20 years in this field. The topics covered are organized as follows.

Chapter 1 gives a brief introduction to system software reliability and basic terminologies used throughout the book. This chapter also identifies the literature available in the area of software reliability engineering. Chapter 2 discusses the concepts of system reliability engineering, systemability, various reliability aspects of systems with multiple failure modes, and the stochastic processes including the Markov process, renewal process, quasi-renewal process, and nonhomogeneous Poisson process.

Chapter 3 describes the theory of estimation, common estimation techniques and confidence interval estimates. Chapter 4 presents the basic concepts of software engineering assessment including software lifecycle, software development

process and its applications, software testing concepts, and data analysis. Chapter 5 discusses various groups of traditional software reliability models and methods for evaluating software reliability and other performance measures, such as software complexity and the number of remaining errors.

Chapter 6 comprehensively covers software reliability models for the failure phenomenon based on the nonhomogeneous Poisson process (NHPP). The generalized NHPP model, model selection and the software mean time between failures are also discussed. Chapter 7 discusses various software reliability models addressing testing coverage and fault removal.

Chapter 8 describes some recent studies on environmental factors and the impact of these factors on software reliability assessment. Several software reliability models that incorporate environmental factors are also discussed. Chapter 9 discusses calibrating software reliability modelling research on how to quantify the mismatch between the system test environment and the field environment.

Chapter 10 discusses some software cost models based on the NHPP addressing warranty issues and risk costs due to software failures. This chapter also discusses a generalized gain model under random field environments. Various optimal release policies of the software systems, that is when to conclude testing and release the software, are also presented.

Chapter 11 is devoted to the basic concepts of fault-tolerant software system modeling and other advanced techniques including self-checking schemes. The reliability analysis of fault-tolerant software schemes such as recover block, N-version programming, and hybrid fault-tolerant systems are presented. This chapter also discusses a triple-version programming reliability model with common failures. A brief mathematical reliability analysis of complex systems considering hardware and software interaction failures is also discussed.

A list of references for further reading and problems are included at the end of each chapter. Solutions to selected problems are provided towards the end of the book. Appendix 1 contains various distribution tables. Appendix 2 contains some useful Laplace transform functions. Appendix 3 provides a survey form which software engineers may adopt in order to obtain a better understanding of the software development activities and priorities.

The text is suitable for a one-semester graduate course on software reliability in Industrial Engineering, Systems Engineering, Operations Research, Computer Science and Engineering, Mathematics, Statistics, and Business Management. The book will also be a valuable reference tool for practitioners and managers in reliability engineering, software engineering, statistics, safety engineering, and for researchers in the field. It is also intended that the individual, after having utilized this book, will be thoroughly prepared to pursue advanced studies in software reliability engineering and research in this field.

I have used the first seven chapters as supplementary reading for a three-day industrial seminar on system software reliability and, in addition, have included the material from Chapter 8 through 10 for a five-day seminar. These short sessions, three-day and five-day, serve as both an introduction and an account of advances in software reliability discipline, and can be easily tailored by the instructors to be specific for an industry or organization. Similarly, one can use the first



three chapters, together with Chapter 6 and Chapter 11, for a two-day seminar on system reliability engineering.

I am grateful to the students of the Department of Industrial and Systems Engineering at Rutgers University who have used preliminary versions of this book during the past two years and have provided numerous comments and suggestions. Thanks also go to many colleagues from universities and industry for their useful suggestions.

Anthony Doyle, Senior Editor, and Kate Brown at Springer-Verlag deserve significant praise for their patience and understanding when deadlines were missed and for their assistance in finalizing the camera-ready version.

Finally and most importantly, I want to thank my other-half, Michelle, and my sons, Hoang Jr. and David, for their love, patience, understanding, and support. It is to them that this book is dedicated.

Hoang Pham  
Rutgers University, New Jersey, USA  
December 2005

---

# Contents

<b>1</b>	<b>Introduction</b>	1
1.1	The Need for System Software Reliability	1
1.2	Software-related Problems	3
1.3	Software Reliability Engineering	5
1.4	Future Problems in the Twenty-first Century	5
1.5	Further Reading	6
1.6	Problems	7
<b>2</b>	<b>System Reliability Concepts</b>	9
2.1	Reliability Measures	10
2.2	Common Distribution Functions	16
2.3	A Generalized Systemability Function	32
2.3.1	Systemability Definition	33
2.3.2	Systemability Calculations	33
2.4	System Reliability with Multiple Failure Modes	41
2.4.1	Reliability Calculations	42
2.4.2	An Application of Systems with Multiple Failure Modes	48
2.5	Markov Processes	50
2.6	Counting Processes	62
2.6.1	Poisson Processes	63
2.6.2	Renewal Processes	64
2.6.3	Quasi-renewal Processes	66
2.6.4	Non-homogeneous Poisson Processes	69
2.7	Further Reading	71
2.8	Problems	71
<b>3</b>	<b>Theory of Estimation</b>	77
3.1	Point Estimation	77
3.2	Maximum Likelihood Estimation Method	79
3.3	Maximum Likelihood Estimation with Censored Data	86
3.3.1	Parameter Estimate with Multiple-censored Data	86

3.3.2	Confidence Intervals of Estimates .....	88
3.3.3	Applications.....	89
3.4	Statistical Change-point Estimation Methods.....	91
3.4.1	Application: A Software Model with a Change Point .....	95
3.5	Goodness of Fit Techniques .....	96
3.5.1	Chi-squared Test.....	96
3.5.2	Kolmogorov-Smirnov $d$ Test.....	98
3.6	Least Squared Estimation .....	98
3.7	Interval Estimation .....	100
3.7.1	Confidence Intervals for the Normal Parameters.....	100
3.7.2	Confidence Intervals for the Exponential Parameters.....	102
3.7.3	Confidence Intervals for the Binomial Parameters .....	104
3.7.4	Confidence Intervals for the Poisson Parameters .....	106
3.8	Non-parametric Tolerance Limits.....	106
3.9	Sequential Sampling .....	107
3.10	Bayesian Methods.....	113
3.11	Further Reading .....	118
3.12	Problems .....	119
<b>4</b>	<b>Software Development Lifecycle and Data Analysis .....</b>	<b>121</b>
4.1	Introduction .....	121
4.2	Software vs Hardware Reliability.....	122
4.3	Software Reliability and Testing Concepts.....	124
4.4	Software Lifecycle .....	127
4.5	Software Development Process and its Applications .....	132
4.5.1	Analytic Hierarchy Process .....	133
4.5.2	Evaluation of Software Development Process.....	133
4.6	Software Verification and Validation.....	134
4.7	Data Analysis.....	135
4.8	Failure Data Sets.....	136
4.9	Further Reading .....	150
4.10	Problems .....	150
<b>5</b>	<b>Software Reliability Modeling.....</b>	<b>153</b>
5.1	Introduction .....	153
5.2	Halstead's Software Metric .....	154
5.3	McCabe's Cyclomatic Complexity Metric .....	157
5.4	Error Seeding Models .....	159
5.5	Failure Rate Models.....	164
5.6	Curve Fitting Models.....	169
5.7	Reliability Growth Models .....	171
5.8	Markov Structure Models.....	172
5.9	Time Series Models .....	174
5.10	Non-homogeneous Poisson Process Models .....	175
5.11	Further Reading .....	176
5.12	Problems .....	176

<b>6</b>	<b>Imperfect-debugging Models</b>	179
6.1	Introduction	179
6.2	Parameter Estimation	180
6.3	Model Selection	181
6.4	NHPP Exponential Models	183
6.5	NHPP S-shaped Models	188
6.6	NHPP Imperfect Debugging Models	192
6.7	NHPP Imperfect Debugging S-shaped Models	194
6.7.1	A Generalized Imperfect-debugging Fault-detection Model	195
6.8	Applications	203
6.9	Imperfect Debugging vs Perfect Debugging	211
6.10	Mean Time Between Failures for NHPP	213
6.11	Further Reading	216
6.12	Problems	216
<b>7</b>	<b>Testing Coverage and Removal Models</b>	219
7.1	Introduction	219
7.2	Testing Coverage Models	219
7.3	Testing Coverage and Imperfect Debugging	222
7.4	Fault Removal Efficiency Model	224
7.5	Model Implementations	231
7.6	Imperfect Debugging Model with Multiple Failure Types	247
7.6.1	A Constant Fault Detection Rate	248
7.6.2	Fault Detection Time-dependent Rate	251
7.7	Further Reading	255
7.8	Problems	256
<b>8</b>	<b>Software Reliability Models with Environmental Factors</b>	257
8.1	Introduction	257
8.2	Data Analysis	257
8.2.1	Survey Analysis	258
8.2.2	Statistical Methods	261
8.3	Exploratory Analysis of Environmental Factors	263
8.4	Further Exploratory Analysis	266
8.5	A Generalized Model with Environmental Factors	272
8.6	Environmental Parameter Estimation	275
8.7	Enhanced Proportional Hazard Jelinski-Moranda (EPJM) Model	276
8.8	Applications	279
8.9	Further Reading	292
8.10	Problems	292
<b>9</b>	<b>Calibrating Software Reliability Models</b>	293
9.1	Introduction	293
9.2	Calibration Factor Approach	294
9.3	Model Application	295
9.4	Calibrating Models with Random Field Environments	296
9.4.1	A Generalized Random Field Environmental Model	298

9.4.2 RFE Reliability Models .....	301
9.4.3 Applications.....	303
9.5 Further Reading .....	313
9.6 Problems .....	313
<b>10 Optimal Release Policies .....</b>	<b>315</b>
10.1 Introduction .....	315
10.2 A Software Cost Model with Risk Factor.....	316
10.3 Cost Model with Testing Coverage .....	319
10.4 A Generalized Software Cost Model .....	323
10.5 Cost Model with Multiple Failure Errors .....	326
10.6 Gain Model with Random Field Environments .....	332
10.6.1 Model Formulation .....	334
10.6.2 Applications.....	338
10.7 Other Cost Models.....	342
10.8 Further Reading .....	343
10.9 Problems .....	343
<b>11 Complex Fault-tolerant System Reliability Modeling.....</b>	<b>347</b>
11.1 Introduction .....	347
11.2 Basic Fault-tolerant Software Techniques .....	348
11.2.1 Recovery Block Scheme.....	349
11.2.2 N-version Programming .....	350
11.3 Other Advanced Techniques.....	352
11.3.1 Self-checking Duplex Scheme.....	352
11.3.2 Hybrid Fault-tolerant Scheme.....	353
11.3.3 Reduction of Common-cause Failures.....	355
11.4 Triple-version Programming Model with Common Failures .....	357
11.4.1 Modeling Assumptions.....	360
11.4.2 TVP Reliability Function.....	364
11.4.3 Numerical Example .....	366
11.5 Complex-system Reliability Modeling .....	375
11.5.1 System Considerations.....	375
11.5.2 Reliability Modeling .....	377
11.6 Application Example .....	383
11.7 Further Reading .....	385
11.8 Problems .....	386
<b>Appendix 1: Distribution Tables.....</b>	<b>389</b>
<b>Appendix 2: Laplace Transform.....</b>	<b>395</b>
<b>Appendix 3: Survey of Factors that Affect Software Reliability.....</b>	<b>399</b>
<b>References .....</b>	<b>407</b>
<b>Glossary.....</b>	<b>423</b>
<b>Solutions to Selected Problems.....</b>	<b>429</b>
<b>Index .....</b>	<b>437</b>

## Introduction

### 1.1 The Need for System Software Reliability

Today, almost everyone in the world is directly or indirectly affected by computer systems. Computers are used in diverse areas for various applications including air traffic control, nuclear reactors, aircraft, real-time sensor networks, industrial process control, automotive mechanical and safety control, and hospital health care, affecting many millions of people.

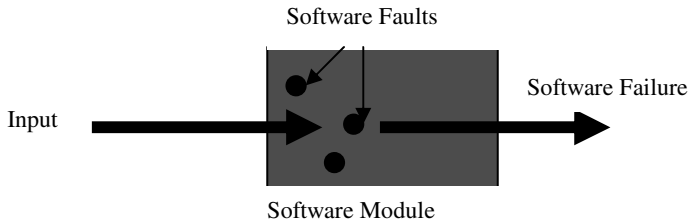
An application of computer systems to the hospital health care is the monitoring of heart patients. In hospitals so equipped, sensors that detect electrical signals associated with heart activity are attached to the patient's heart area. The signals from these sensors are transmitted along wires to a computer programmed to analyze such data. If the incoming data indicate that the patient is doing well, the computer generated no output. If the data indicate the onset of serious conditions, the computer signals an alarm at the nursing station indicating which patient needs human care and the kind of help most apt to be useful.

As the functionality of computer operations becomes more essential and yet more complicated and critical applications increase in size and complexity, there is a great need for looking at ways to quantify and predict the reliability of computer systems in various complex operating environments (Pham 2005c). Faults, especially with logic, in software design thus become more subtle. Usually logic errors in the software are not hard to fix but diagnosing logic bugs is the most challenging for many reasons. The fault again is usually subtle.

Let us look at an example. A man wants to withdraw \$50 at an automatic transfer machine (ATM) from a checking account held jointly with his wife. Almost simultaneously, at another machine, his wife also begins the deposit of \$500. Both the husband's and the wife's ATM read the account balance of \$100 from the memory at the bank's central computer. While the first ATM (husband's machine) subtracts the withdrawal, the second ATM adds the deposit. Because withdrawals often take slightly longer to process than deposits, the wife's ATM records a new balance of \$600 before her husband's transaction is complete. His ATM, obviously not knowing that the old balance has been changed and in fact

increased, records a wrong balance of \$50, instead of the new balance which should be \$550!

Let us define the terms such as “software error”, “fault” and “failure” (IEEE Std. 610.12,1990). An error is a mental mistake made by the programmer or designer. A fault is the manifestation of that error in the code. A software failure is defined as the occurrence of an incorrect output as a result of an input value that is received with respect to the specification. Figure 1.1 demonstrates how a software fault triggered by a specific input leads to software failure.

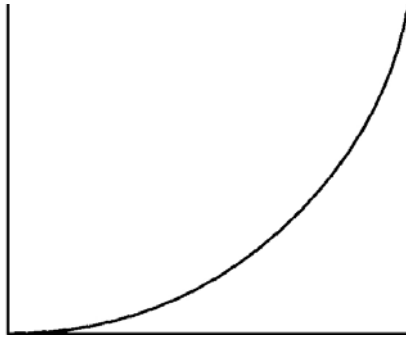


**Figure 1.1.** Relationship between software fault and software failure

A computer system consists of two major components: hardware and software. Although extensive research has been carried out on hardware reliability, the growing importance of recent software in complex applications dictates that the major focus has shifted to system software reliability and cost analysis. Software reliability is different from hardware reliability in the sense that software does not wear out or burn out. The software itself does not fail unless flaws within the software result in a failure in its dependent system.

In recent years, the cost of developing software and the penalty cost of software failure have become a major expense in the whole system (Pham 1992a). 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, and/or monetary costs are incurred. A quantitative measure of loss is called the risk cost of failure (Pham 1996). In other words, risk cost is a quantitative measure of the severity of loss resulting from a software failure. A research study has shown that professional programmers average six software defects for every 1000 lines of code (LOC) written. At that rate, a typical commercial software application of 350000 LOCs may contain over 2000 programming errors including memory-related errors, memory leaks, language-specific errors, errors calling third-party libraries, extra compilation errors, standard library errors, *etc.*

As software projects become larger, the rate of software defects increases geometrically (see Figure 1.2). Table 1.1 shows the defect rates of several software applications per 100 LOC. Locating software faults is extremely difficult and costly. A study conducted by Microsoft showed that it takes about 12 programming hours to locate and correct a software defect. At this rate, it can take more than 24000 hours (or 11.4 man-years) to debug a program of 350000 LOC at a cost of over US\$1 million.



**Figure 1.2.** The rate of software defect changes

**Table 1.1.** Defect rates of several software applications

Application	Number of systems	Fault density (per 100 LOC)
Airborne	8	1.28
Strategic	18	0.66
Tactical	6	1.00
Process control	2	0.18
Production	9	1.30
Developmental	2	0.40

## 1.2 Software-related Problems

Software errors have caused spectacular failures, some with dire consequences, such as the following examples. On 31 March 1986, a Mexicana Airlines Boeing 727 airliner crashed into a mountain because the software system did not correctly negotiate the mountain position. Between March and June 1986, the massive Therac-25 radiation therapy machines in Marietta, Georgia; Boston, Massachusetts; and Tyler, Texas overdosed cancer patients due to flaws in the computer program controlling the highly automated devices. On 26 June 1988, Air France's new A320 model, just delivered two days before, crashed into the trees at an air show near Mulhouse in France due to computer software failure while performing a low-level pass. Three passengers were killed.

During the period 2-4 November 1988, a computer virus infected software at universities and defense research centers in the United States causing system failures. On 10 December 1990, Space Shuttle Columbia was forced to land early due to computer software problems. On 17 September 1991, a power outage at the AT&T switching facility in New York City interrupted service to 10 million telephone users for 9 hours. The problem was due to the deletion of three bits of



code in a software upgrade and failure to test the software before its installation in the public network.

The Patriot missile systems were built to intercept the Scud missiles but a bug in the Patriot software processing their clock times caused them to fail to intercept the target. The clocks were originally supposed to be reset frequently but as they were in one place for more than 100 hours the software failed, causing the missiles to miss their Scud targets. During the 1991 Gulf War this software problem resulted in the failure of the Patriot missile system to track an Iraqi Scud missile causing the deaths of 28 American soldiers. On 14 August 2003, a blackout that crippled most of the Northeast corridor of the United State and parts of Canada resulted from a software failure at FirstEnergy Corporation which may have contributed significantly to the outage.

In 1992, the London Ambulance service switched to a voice and computer control system which logged all its activities. However, when the traffic to the computer increased the software could not cope and slowed down; as a consequence it lost track of the ambulances. On 26 October 1992, the computer-aided dispatch system of the Ambulance Service in London, which handles more than 5000 requests daily in the transportation of patients in critical condition, failed after installation. This led to serious consequences for many critical patients.

A recent inquiry revealed that a software design error and insufficient software testing caused an explosion that ended the maiden flight of the European Space Agency's (ESA) Ariane 5 rocket, less than 40 seconds after lift-off on 4 June 1996. The problems occurred in the flight control system and were caused by a few lines of Ada code containing three unprotected variables. One of these variables pertained to the rocket launcher's horizontal velocity. A problem occurred when the ESA used the software for the inertial-reference flight-control system in the Ariane 5, similar to the one used in the Ariane 4. The Ariane 5 has a high initial acceleration and a trajectory that leads to a horizontal velocity acceleration rate five times greater than that found in Ariane 4. Upon lift-off, the Ariane 5's horizontal velocity exceeded a limit that was set by the old software in the backup inertial-reference system's computer. This stopped the primary and backup inertial-reference system's computers, causing the rocket to veer off course and explode. The ESA report revealed that officials did not conduct a pre-flight test of the Ariane 5's inertial-reference system, which would have located the fault. The companies involved in this project had assumed that the same inertial-reference-system software would work in both Ariane 4 and Ariane 5. The ESA estimates that corrective measures will amount to US\$362 million (Pham 2000a).

Generally, software faults are more insidious and much more difficult to handle than physical defects. In theory, software can be error-free, and unlike hardware, does not degrade or wear out but it does deteriorate. The deterioration here, however, is not a function of time. Rather, it is a function of the results of changes made to the software during maintenance, through correcting latent defects, modifying the code to changing requirements and specifications, environments and applications, or improving software performance. All design faults are present from the time the software is installed in the computer. In principle, these faults could be removed completely, but in reality the goal of

perfect software remains elusive (Friedman and Voas 1995). Computer programs, which vary for fairly critical applications between hundreds and millions of lines of code, can make the wrong decision because the particular inputs that triggered the problem were not tested and corrected during the testing phase. Such inputs may even have been misunderstood or unanticipated by the designer who either correctly programmed the wrong interpretation or failed to identify the problem. These situations and other such events have made it apparent that we must determine the reliability of the software systems before putting them into operation.

### 1.3 Software Reliability Engineering

Research on software reliability engineering has been conducted during the past three decades and numerous statistical models have been proposed for estimating software reliability (Pham 1999a, 2000a). Most existing models for predicting software reliability are based purely on the observation of software product failures where they require a considerable amount of failure data to obtain an accurate reliability prediction. Some other research efforts recently have developed reliability models addressing fault coverage, testing coverage, and imperfect debugging processes.

In contrast, not many software practitioners, developers, or users utilize these models to evaluate software system reliability as they do not know how to select and apply them. A survey conducted in the late 1990s by the American Society for Quality reported that only 4% of the participants responded positively when asked if they could use a software reliability model.

Many researchers are currently pursuing the development of statistical models, based on nonhomogeneous Poisson process, semi quasi renewal, time series, that can be used to evaluate the reliability of real-world software systems. To develop an application-practice software reliability model and be able to make sound judgments when using the model, one needs to understand how software is produced and tested, the types of errors, and how errors are introduced. Environmental factors can help us justify the usefulness of the model and its applicability in a user environment. In other words, these models would be valuable if practitioners, software developers and users could use the information about the software development process, incorporating the environmental factors, thus giving greater confidence in estimates based on small numbers of failure data.

### 1.4 Future Problems in the Twenty-first Century

In the early 1970s, when computers were first used in the business world, storage space was at a premium and the use of a two-digit convention to represent the year seemed appropriate. For example, a date such as 20 April 1997 is typically represented in software as YY/MM/DD, or 97/04/20, and 1 January 2000 will look like 00/01/01 on our computers, but at that time, the year 2000 was a long way off.

However, the time arrived, causing the year 2000 to be a major software concern of the twenty-first century. It was called the Year 2000 Problem, Y2K Problem or the Millennium Bug. The Year 2000 problem involved either or all of the following: (i) the year 2000 represented as a two-digit number causes failures in arithmetic, comparisons, and input/output to databases or files when manipulating date data, (ii) using an incorrect algorithm to recognize leap years for years divisible by 400, and (iii) system date data types that may roll over and fail due to the storage register becoming full.

Incorrect software programs will assume that the maximum value of a year field is "99" and will roll systems over to the year 1900 instead of 2000, resulting in negative date calculations and the creation of many overnight centenarians. Incorrect leap year calculations will therefore incorrectly assume that the year 2000 has only 365 days instead of 366. The Year 2000 Problem was widespread several years ago. It indeed affected hardware, embedded firmware, languages and compilers, operating systems, nuclear power plants, air-traffic control, security services, database-management systems, communications systems, transaction processing systems, banking systems and medical services.

In 1997, United States federal officials estimated the cost of analysis and modification of the Year 2000 Problem to be US\$2 per line. As an estimated 15 billion lines of code have to be changed to cope with the problem, the work may cost up to US\$30 billion and the worldwide cost may be US\$600 billion. This estimate, however, reflects only conversion costs and may not include the cost of replacing hardware, testing and upgrading the systems.

The year 2000 was, however, not the only date dangerous to software applications. Over the next 50 years, at least 100 million applications around the world will need modification because of formatting problems with dates or related data. The total cost of the remedies in the United States could exceed US\$5 trillion. These problems include: the "nine" end-of-file, the global positioning system's date roll over; the social insecurity identity numbers, the date on which Unix and C libraries rollover and some date-like patterns used for data purposes. In the early twenty-first century, the nine digits assigned to the United States social security numbers and the ten digits allotted to long-distance telephone numbers will no longer suffice for American citizens and required phone lines, respectively. Changes to these numbering systems will also affect software. There is a solution to such problems but it will need a global agreement. A good starting point would be to examine all known date and data-like problems, and how computers and application software basically handle dates.

## 1.5 Further Reading

There are numerous papers published on this subject – *System Software Reliability* – in the last three decades. This section only provides a brief list of recent papers, books, and reviewed papers published in the last 10 years. Interested readers are referred to recent reviewed articles by Pham (1999a, 2003b).

The book *Software Reliability* by Hoang Pham (Springer, 2000), *Software-Reliability-Engineered Testing Practice* by J. D. Musa (McGraw-Hill,

New York, 1997), *Software Assessment: Reliability, Safety, Testability* by M.A. Friedman and J.M. Voas (Wiley, New York, 1995), are recent good textbooks for readers including students, practitioners and researchers.

The *Handbook of Reliability Engineering* edited by Hoang Pham (Springer, 2003), *Handbook of Software Reliability Engineering* edited by M. R. Lyu (IEEE Computer Society Press, Los Angeles, 1996), and *Software Reliability and Testing* by H. Pham (IEEE Computer Society Press, Los Angeles, 1995) are edited books containing many interesting results in which the readers may find useful.

Many research and tutorial papers on software reliability have been published in the *IEEE Transactions on Software Engineering*, *IEEE Transactions on Reliability*, and *IEEE Software Magazine*. Several special issues on software reliability are of practical interest:

Special issues of the *International Journal of Reliability, Quality and Safety Engineering* on Software Reliability, vol. 4(3), September 1997; on Software Reliability Model and Applications, vol. 6(1), March 1999; Special issue of the *IEEE Computer* on System Testing and Reliability, November, 1996

Several other journals occasionally publish papers on the subject:

*Journal of Systems and Software*

*International Journal of Reliability, Quality and Safety Engineering*

*Reliability Engineering and System Safety Journal*

*Microelectronics and Reliability - An International Journal*

*IIE Transactions on Quality and Reliability Engineering*.

There are also a great number of proceedings of international conferences where many interesting papers on software reliability and testing can be found, for example:

*IEEE Annual Reliability and Maintainability Symposium*

*ISSAT International Conference on Reliability and Quality in Design*

*IEEE International Computer Software and Applications Conference*

*IEEE International Conference on Software Engineering*

*IEEE International Symposium on Software Reliability Engineering*

This list is by no means exhaustive, but it will provide readers with a basic knowledge of software reliability.

## 1.6 Problems

1. What is the difference between reliability and availability?
2. List the differences between hardware failures and software failures.
3. Provide examples of software failures, software defects and software faults.

## System Reliability Concepts

The analysis of the reliability of a system must be based on precisely defined concepts. Since it is readily accepted that a population of supposedly identical systems, operating under similar conditions, fall at different points in time, then a failure phenomenon can only be described in probabilistic terms. Thus, the fundamental definitions of reliability must depend on concepts from probability theory. This chapter describes the concepts of system reliability engineering. These concepts provide the basis for quantifying the reliability of a system. They allow precise comparisons between systems or provide a logical basis for improvement in a failure rate. Various examples reinforce the definitions as presented in Section 2.1. Section 2.2 examines common distribution functions useful in reliability engineering. Several distribution models are discussed and the resulting hazard functions are derived. Section 2.3 describes a new concept of systemability. Several systemability functions of various system configurations such as series, parallel, and k-out-of-n, are presented. Section 2.4 discusses various reliability aspects of systems with multiple failure modes. Stochastic processes including Markov process, Poisson process, renewal process, quasi-renewal process, and nonhomogeneous Poisson process are discussed in Sections 2.5 and 2.6.

In general, a system may be required to perform various functions, each of which may have a different reliability. In addition, at different times, the system may have a different probability of successfully performing the required function under stated conditions. The term failure means that the system is not capable of performing a function when required. The term *capable* used here is to define if the system is capable of performing the required function. However, the term *capable* is unclear and only various degrees of capability can be defined.

## 2.1 Reliability Measures

The reliability definitions given in the literature vary between different practitioners as well as researchers. The generally accepted definition is as follows.

**Definition 2.1:** *Reliability* is the probability of success or the probability that the system will perform its intended function under specified design limits.

More specific, reliability is the probability that a product or part will operate properly for a specified period of time (design life) under the design operating conditions (such as temperature, volt, *etc.*) without failure. In other words, reliability may be used as a measure of the system's success in providing its function properly. Reliability is one of the quality characteristics that consumers require from the manufacturer of products.

Mathematically, reliability  $R(t)$  is the probability that a system will be successful in the interval from time 0 to time  $t$ :

$$R(t) = P(T > t) \quad t \geq 0 \quad (2.1)$$

where  $T$  is a random variable denoting the time-to-failure or failure time.

*Unreliability*  $F(t)$ , a measure of failure, is defined as the probability that the system will fail by time  $t$ :

$$F(t) = P(T \leq t) \quad \text{for } t \geq 0$$

In other words,  $F(t)$  is the failure distribution function. If the time-to-failure random variable  $T$  has a density function  $f(t)$ , then

$$R(t) = \int_t^{\infty} f(s) ds$$

or, equivalently,

$$f(t) = -\frac{d}{dt}[R(t)]$$

The density function can be mathematically described in terms of  $T$ :

$$\lim_{\Delta t \rightarrow 0} P(t < T \leq t + \Delta t)$$

This can be interpreted as the probability that the failure time  $T$  will occur between the operating time  $t$  and the next interval of operation,  $t + \Delta t$ .

Consider a new and successfully tested system that operates well when put into service at time  $t = 0$ . The system becomes less likely to remain successful as the time interval increases. The probability of success for an infinite time interval, of course, is zero.

Thus, the system functions at a probability of one and eventually decreases to a probability of zero. Clearly, reliability is a function of mission time. For example, one can say that the reliability of the system is 0.995 for a mission time of 24 hours. However, a statement such as the reliability of the system is 0.995 is meaningless because the time interval is unknown.

*Example 2.1:* A computer system has an exponential failure time density function

$$f(t) = \frac{1}{9,000} e^{-\frac{t}{9,000}} \quad t \geq 0$$

What is the probability that the system will fail after the warranty (six months or 4380 hours) and before the end of the first year (one year or 8760 hours)?

*Solution:* From equation (2.1) we obtain

$$\begin{aligned} P(4380 < T \leq 8760) &= \int_{4380}^{8760} \frac{1}{9000} e^{-\frac{t}{9000}} dt \\ &= 0.237 \end{aligned}$$

This indicates that the probability of failure during the interval from six months to one year is 23.7%.

If the time to failure is described by an exponential failure time density function, then

$$f(t) = \frac{1}{\theta} e^{-\frac{t}{\theta}} \quad t \geq 0, \theta > 0 \quad (2.2)$$

and this will lead to the reliability function

$$R(t) = \int_t^{\infty} \frac{1}{\theta} e^{-\frac{s}{\theta}} ds = e^{-\frac{t}{\theta}} \quad t \geq 0 \quad (2.3)$$

Consider the Weibull distribution where the failure time density function is given by

$$f(t) = \frac{\beta t^{\beta-1}}{\theta^{\beta}} e^{-\left(\frac{t}{\theta}\right)^{\beta}} \quad t \geq 0, \theta > 0, \beta > 0$$

Then the reliability function is

$$R(t) = e^{-\left(\frac{t}{\theta}\right)^{\beta}} \quad t \geq 0$$

Thus, given a particular failure time density function or failure time distribution function, the reliability function can be obtained directly. Section 2.2 provides further insight for specific distributions.

### System Mean Time to Failure

Suppose that the reliability function for a system is given by  $R(t)$ . The expected failure time during which a component is expected to perform successfully, or the system mean time to failure (MTTF), is given by

$$MTTF = \int_0^{\infty} t f(t) dt \quad (2.4)$$

Substituting

$$f(t) = -\frac{d}{dt}[R(t)]$$

into equation (2.4) and performing integration by parts, we obtain

$$\begin{aligned}
 MTTF &= - \int_0^{\infty} t d[R(t)] \\
 &= [-tR(t)] \Big|_0^{\infty} + \int_0^{\infty} R(t) dt
 \end{aligned}
 \tag{2.5}$$

The first term on the right-hand side of equation (2.5) equals zero at both limits, since the system must fail after a finite amount of operating time. Therefore, we must have  $tR(t) \rightarrow 0$  as  $t \rightarrow \infty$ . This leaves the second term, which equals

$$MTTF = \int_0^{\infty} R(t) dt
 \tag{2.6}$$

Thus, MTTF is the definite integral evaluation of the reliability function. In general, if  $\lambda(t)$  is defined as the failure rate function, then, by definition, MTTF is not equal to  $1/\lambda(t)$ .

The MTTF should be used when the failure time distribution function is specified because the reliability level implied by the MTTF depends on the underlying failure time distribution. Although the MTTF measure is one of the most widely used reliability calculations, it is also one of the most misused calculations. It has been misinterpreted as “guaranteed minimum lifetime”. Consider the results as given in Table 2.1 for a twelve-component life duration test.

**Table 2.1.** Results of a twelve-component life duration test

Component	Time to failure (hours)
1	4510
2	3690
3	3550
4	5280
5	2595
6	3690
7	920
8	3890
9	4320
10	4770
11	3955
12	2750

Using a basic averaging technique, the component MTTF of 3660 hours was estimated. However, one of the components failed after 920 hours. Therefore, it is important to note that the system MTTF denotes the average time to failure. It is neither the failure time that could be expected 50% of the time, nor is it the guaranteed minimum time of system failure.

A careful examination of equation (2.6) will show that two failure distributions can have the same MTTF and yet produce different reliability levels. This is illustrated in a case where the MTTFs are equal, but with normal and exponential



failure distributions. The normal failure distribution is symmetrical about its mean, thus

$$R(MTTF) = P(Z \geq 0) = 0.5$$

where  $Z$  is a standard normal random variable. When we compute for the exponential failure distribution using equation (2.3), recognizing that  $\theta = MTTF$ , the reliability at the MTTF is

$$R(MTTF) = e^{-\frac{MTTF}{MTTF}} = 0.368$$

Clearly, the reliability in the case of the exponential distribution is about 74% of that for the normal failure distribution with the same MTTF.

### Failure Rate Function

The probability of a system failure in a given time interval  $[t_1, t_2]$  can be expressed in terms of the reliability function as

$$\begin{aligned} \int_{t_1}^{t_2} f(t) dt &= \int_{t_1}^{\infty} f(t) dt - \int_{t_2}^{\infty} f(t) dt \\ &= R(t_1) - R(t_2) \end{aligned}$$

or in terms of the failure distribution function (or the unreliability function) as

$$\begin{aligned} \int_{t_1}^{t_2} f(t) dt &= \int_{-\infty}^{t_2} f(t) dt - \int_{-\infty}^{t_1} f(t) dt \\ &= F(t_2) - F(t_1) \end{aligned}$$

The rate at which failures occur in a certain time interval  $[t_1, t_2]$  is called the *failure rate*. It is defined as the probability that a failure per unit time occurs in the interval, given that a failure has not occurred prior to  $t_1$ , the beginning of the interval. Thus, the failure rate is

$$\frac{R(t_1) - R(t_2)}{(t_2 - t_1)R(t_1)}$$

Note that the failure rate is a function of time. If we redefine the interval as  $[t, t + \Delta t]$ , the above expression becomes

$$\frac{R(t) - R(t + \Delta t)}{\Delta t R(t)}$$

The rate in the above definitions is expressed as failures per unit time, when in reality, the time units might be in terms of miles, hours, etc. The *hazard function* is defined as the limit of the failure rate as the interval approaches zero. Thus, the hazard function  $h(t)$  is the instantaneous failure rate, and is defined by

$$\begin{aligned} h(t) &= \lim_{\Delta t \rightarrow 0} \frac{R(t) - R(t + \Delta t)}{\Delta t R(t)} \\ &= \frac{1}{R(t)} \left[ -\frac{d}{dt} R(t) \right] \\ &= \frac{f(t)}{R(t)} \end{aligned} \tag{2.7}$$

The quantity  $h(t)dt$  represents the probability that a device of age  $t$  will fail in the small interval of time  $t$  to  $(t+dt)$ . The importance of the hazard function is that it indicates the change in the failure rate over the life of a population of components by plotting their hazard functions on a single axis. For example, two designs may provide the same reliability at a specific point in time, but the failure rates up to this point in time can differ.

The death rate, in statistical theory, is analogous to the failure rate as the force of mortality is analogous to the hazard function. Therefore, the hazard function or hazard rate or failure rate function is the ratio of the probability density function (pdf) to the reliability function.

### Maintainability

When a system fails to perform satisfactorily, repair is normally carried out to locate and correct the fault. The system is restored to operational effectiveness by making an adjustment or by replacing a component.

Maintainability is defined as the probability that a failed system will be restored to specified conditions within a given period of time when maintenance is performed according to prescribed procedures and resources. In other words, maintainability is the probability of isolating and repairing a fault in a system within a given time. Maintainability engineers must work with system designers to ensure that the system product can be maintained by the customer efficiently and cost effectively. This function requires the analysis of part removal, replacement, tear-down, and build-up of the product in order to determine the required time to carry out the operation, the necessary skill, the type of support equipment and the documentation.

Let  $T$  denote the random variable of the time to repair or the total downtime. If the repair time  $T$  has a repair time density function  $g(t)$ , then the maintainability,  $V(t)$ , is defined as the probability that the failed system will be back in service by time  $t$ , i.e.,

$$V(t) = P(T \leq t) = \int_0^t g(s)ds$$

For example, if  $g(t) = \mu e^{-\mu t}$  where  $\mu > 0$  is a constant repair rate, then

$$V(t) = 1 - e^{-\mu t}$$

which represents the exponential form of the maintainability function.

An important measure often used in maintenance studies is the mean time to repair (MTTR) or the mean downtime. MTTR is the expected value of the random variable repair time, not failure time, and is given by

$$MTTR = \int_0^{\infty} t g(t) dt$$

When the distribution has a repair time density given by  $g(t) = \mu e^{-\mu t}$ , then, from the above equation,  $MTTR = 1/\mu$ . When the repair time  $T$  has the log normal density function  $g(t)$ , and the density function is given by

$$g(t) = \frac{1}{\sqrt{2\pi} \sigma t} e^{-\frac{(\ln t - \mu)^2}{2\sigma^2}} \quad t > 0$$

then it can be shown that

$$MTTR = m e^{\frac{\sigma^2}{2}}$$

where  $m$  denotes the median of the log normal distribution.

In order to design and manufacture a maintainable system, it is necessary to predict the MTTR for various fault conditions that could occur in the system. This is generally based on past experiences of designers and the expertise available to handle repair work.

The system repair time consists of two separate intervals: passive repair time and active repair time. Passive repair time is mainly determined by the time taken by service engineers to travel to the customer site. In many cases, the cost of travel time exceeds the cost of the actual repair. Active repair time is directly affected by the system design and is listed as follows:

1. The time between the occurrence of a failure and the system user becoming aware that it has occurred.
2. The time needed to detect a fault and isolate the replaceable component(s).
3. The time needed to replace the faulty component(s).
4. The time needed to verify that the fault has been corrected and the system is fully operational.

The active repair time can be improved significantly by designing the system in such a way that faults may be quickly detected and isolated. As more complex systems are designed, it becomes more difficult to isolate the faults.

### Availability

Reliability is a measure that requires system success for an entire mission time. No failures or repairs are allowed. Space missions and aircraft flights are examples of systems where failures or repairs are not allowed. Availability is a measure that allows for a system to repair when failure occurs.

The availability of a system is defined as the probability that the system is successful at time  $t$ . Mathematically,

$$\begin{aligned} \text{Availability} &= \frac{\text{System up time}}{\text{System up time} + \text{System down time}} \\ &= \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}} \end{aligned}$$

Availability is a measure of success used primarily for repairable systems. For non-repairable systems, availability,  $A(t)$ , equals reliability,  $R(t)$ . In repairable systems,  $A(t)$  will be equal to or greater than  $R(t)$ .

The mean time between failures (MTBF) is an important measure in repairable systems. This implies that the system has failed and has been repaired. Like MTTF

and MTTR, MTBF is an expected value of the random variable time between failures. Mathematically,

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

The term MTBF has been widely misused. In practice, MTTR is much smaller than MTTF, which is approximately equal to MTBF. MTBF is often incorrectly substituted for MTTF, which applies to both repairable systems and non-repairable systems. If the MTTR can be reduced, availability will increase, and the system will be more economical.

A system where faults are rapidly diagnosed is more desirable than a system that has a lower failure rate but where the cause of a failure takes longer to detect, resulting in a lengthy system downtime. When the system being tested is renewed through maintenance and repairs,  $E(T)$  is also known as MTBF.

## 2.2 Common Distribution Functions

This section presents some of the common distribution functions and several hazard models that have applications in reliability engineering (Pham 2000a).

### Binomial Distribution

The binomial distribution is one of the most widely used discrete random variable distributions in reliability and quality inspection. It has applications in reliability engineering, *e.g.*, when one is dealing with a situation in which an event is either a success or a failure.

The pdf of the distribution is given by

$$P(X = x) = \binom{n}{x} p^x (1-p)^{n-x} \quad x = 0, 1, 2, \dots, n$$

$$\binom{n}{x} = \frac{n!}{x!(n-x)!}$$

where  $n$  = number of trials;  $x$  = number of successes;  $p$  = single trial probability of success.

The reliability function,  $R(k)$ , (*i.e.*, at least  $k$  out of  $n$  items are good) is given by

$$R(k) = \sum_{x=k}^n \binom{n}{x} p^x (1-p)^{n-x}$$

**Example 2.2:** Suppose in the production of lightbulbs, 90% are good. In a random sample of 20 lightbulbs, what is the probability of obtaining at least 18 good lightbulbs?

**Solution:** The probability of obtaining 18 or more good lightbulbs in the sample of 20 is

$$\begin{aligned}
 R(18) &= \sum_{x=18}^{20} \binom{20}{18} (0.9)^x (0.1)^{20-x} \\
 &= 0.667
 \end{aligned}$$

### Poisson Distribution

Although the Poisson distribution can be used in a manner similar to the binomial distribution, it is used to deal with events in which the sample size is unknown. This is also a discrete random variable distribution whose pdf is given by

$$P(X = x) = \frac{(\lambda t)^x e^{-\lambda t}}{x!} \quad \text{for } x = 0, 1, 2, \dots$$

where  $\lambda$  = constant failure rate,  $x$  = is the number of events. In other words,  $P(X = x)$  is the probability of exactly  $x$  failures occurring in time  $t$ . Therefore, the reliability Poisson distribution,  $R(k)$  (the probability of  $k$  or fewer failures) is given by

$$R(k) = \sum_{x=0}^k \frac{(\lambda t)^x e^{-\lambda t}}{x!}$$

This distribution can be used to determine the number of spares required for the reliability of standby redundant systems during a given mission.

### Exponential Distribution

Exponential distribution plays an essential role in reliability engineering because it has a constant failure rate. This distribution has been used to model the lifetime of electronic and electrical components and systems. This distribution is appropriate when a used component that has not failed is as good as a new component - a rather restrictive assumption. Therefore, it must be used diplomatically since numerous applications exist where the restriction of the memoryless property may not apply. For this distribution, we have reproduced equations (2.2) and (2.3), respectively:

$$\begin{aligned}
 f(t) &= \frac{1}{\theta} e^{-\frac{t}{\theta}} = \lambda e^{-\lambda t}, \quad t \geq 0 \\
 R(t) &= e^{-\frac{t}{\theta}} = e^{-\lambda t}, \quad t \geq 0
 \end{aligned}$$

where  $\theta = 1/\lambda > 0$  is an MTTF's parameter and  $\lambda \geq 0$  is a constant failure rate.

The hazard function or failure rate for the exponential density function is constant, i.e.,

$$h(t) = \frac{f(t)}{R(t)} = \frac{\frac{1}{\theta} e^{-\frac{t}{\theta}}}{e^{-\frac{t}{\theta}}} = \frac{1}{\theta} = \lambda$$

The failure rate for this distribution is  $\lambda$ , a constant, which is the main reason for this widely used distribution. Because of its constant failure rate property, the exponential is an excellent model for the long flat "intrinsic failure" portion of the

bathtub curve. Since most parts and systems spend most of their lifetimes in this portion of the bathtub curve, this justifies frequent use of the exponential (when early failures or wear out is not a concern). The exponential model works well for inter-arrival times. When these events trigger failures, the exponential lifetime model can be used.

We will now discuss some properties of the exponential distribution that are useful in understanding its characteristics, when and where it can be applied.

**Property 2.1:** (*Memoryless property*) The exponential distribution is the only continuous distribution satisfying

$$P\{T \geq t\} = P\{T \geq t + s \mid T \geq s\} \quad \text{for } t > 0, s > 0 \quad (2.8)$$

This result indicates that the conditional reliability function for the lifetime of a component that has survived to time  $s$  is identical to that of a new component. This term is the so-called "used-as-good-as-new" assumption.

The lifetime of a fuse in an electrical distribution system may be assumed to have an exponential distribution. It will fail when there is a power surge causing the fuse to burn out. Assuming that the fuse does not undergo any degradation over time and that power surges that cause failure are likely to occur equally over time, then use of the exponential lifetime distribution is appropriate, and a used fuse that has not failed is as good as new.

**Property 2.2:** If  $T_1, T_2, \dots, T_n$ , are independently and identically distributed exponential random variables (RVs) with a constant failure rate  $\lambda$ , then

$$2\lambda \sum_{i=1}^n T_i \sim \chi^2(2n)$$

where  $\chi^2(2n)$  is a chi-squared distribution with degrees of freedom  $2n$ . This result is useful for establishing a confidence interval for  $\lambda$ .

*Example 2.3:* A manufacturer performs an operational life test on ceramic capacitors and finds they exhibit constant failure rate with a value of  $3 \times 10^{-8}$  failure per hour. What is the reliability of a capacitor at  $10^4$  hours?

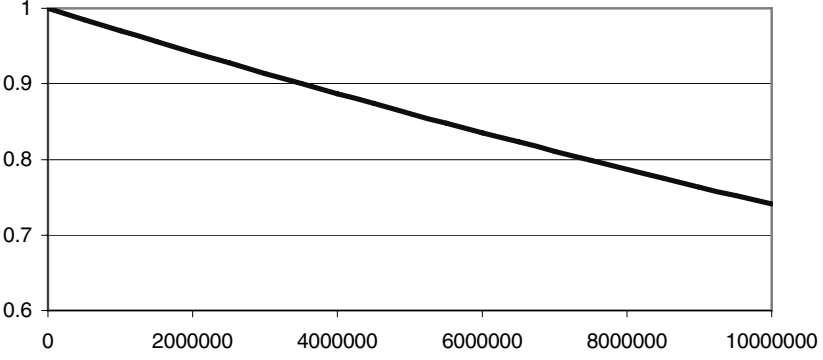
*Solution:* The reliability of a capacitor at  $10^4$  hour is

$$R(t) = e^{-\lambda t} = e^{-3 \times 10^{-8} t} = e^{-3 \times 10^{-4}} = 0.9997$$

The resulting reliability plot is shown in Figure 2.1.

### Normal Distribution

Normal distribution plays an important role in classical statistics owing to the *Central Limit Theorem*. In reliability engineering, the normal distribution primarily applies to measurements of product susceptibility and external stress. This two-parameter distribution is used to describe systems in which a failure results due to some wearout effect for many mechanical systems.



**Figure 2.1.** Reliability function vs time

The normal distribution takes the well-known bell shape. This distribution is symmetrical about the mean and the spread is measured by variance. The larger the value, the flatter the distribution. The pdf is given by

$$f(t) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2} \quad -\infty < t < \infty$$

where  $\mu$  is the mean value and  $\sigma$  is the standard deviation. The cumulative distribution function (cdf) is

$$F(t) = \int_{-\infty}^t \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{s-\mu}{\sigma}\right)^2} ds$$

The reliability function is

$$R(t) = \int_t^{\infty} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{s-\mu}{\sigma}\right)^2} ds$$

There is no closed form solution for the above equation. However, tables for the standard normal density function are readily available (see Table A1.1 in Appendix 1) and can be used to find probabilities for any normal distribution. If

$$Z = \frac{T - \mu}{\sigma}$$

is substituted into the normal pdf, we obtain

$$f(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}} \quad -\infty < Z < \infty$$

This is a so-called standard normal pdf, with a mean value of 0 and a standard deviation of 1. The standardized cdf is given by

$$\Phi(t) = \int_{-\infty}^t \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}s^2} ds \quad (2.9)$$

where  $\Phi$  is a standard normal distribution function. Thus, for a normal random variable  $T$ , with mean  $\mu$  and standard deviation  $\sigma$ ,

$$P(T \leq t) = P\left(Z \leq \frac{t - \mu}{\sigma}\right) = \Phi\left(\frac{t - \mu}{\sigma}\right)$$

where  $\Phi$  yields the relationship necessary if standard normal tables are to be used. The hazard function for a normal distribution is a monotonically increasing function of  $t$ . This can be easily shown by proving that  $h'(t) \geq 0$  for all  $t$ . Since

$$h(t) = \frac{f(t)}{R(t)}$$

then (see Problem 15)

$$h'(t) = \frac{R(t)f'(t) + f^2(t)}{R^2(t)} \geq 0 \quad (2.10)$$

One can try this proof by employing the basic definition of a normal density function  $f$ .

*Example 2.4:* A component has a normal distribution of failure times with  $\mu = 2000$  hours and  $\sigma = 100$  hours. Find the reliability of the component and the hazard function at 1900 hours.

*Solution:* The reliability function is related to the standard normal deviate  $z$  by

$$R(t) = P\left(Z > \frac{t - \mu}{\sigma}\right)$$

where the distribution function for  $Z$  is given by equation (2.9). For this particular application,

$$\begin{aligned} R(1900) &= P\left(Z > \frac{1900 - 2000}{100}\right) \\ &= P(z > -1) \end{aligned}$$

From the standard normal table in Table A1.1 in Appendix 1, we obtain

$$R(1900) = 1 - \Phi(-1) = 0.8413.$$

The value of the hazard function is found from the relationship

$$h(t) = \frac{f(t)}{R(t)} = \frac{\Phi\left(z = \frac{t - \mu}{\sigma}\right)}{\sigma R(t)}$$

where  $\phi$  is a pdf of standard normal density. Here

$$\begin{aligned} h(1900) &= \frac{\Phi(-1.0)}{\sigma R(t)} = \frac{0.1587}{100(0.8413)} \\ &= 0.0019 \text{ failures/cycle} \end{aligned}$$

*Example 2.5:* A part has a normal distribution of failure times with  $\mu = 40000$  cycles and  $\sigma = 2000$  cycles. Find the reliability of the part at 38000 cycles.



*Solution:* The reliability at 38000 cycles

$$\begin{aligned}
 R(38000) &= P\left(z > \frac{38000 - 40000}{2000}\right) \\
 &= P(z > -1.0) \\
 &= \Phi(1.0) = 0.8413
 \end{aligned}$$

The resulting reliability plot is shown in Figure 2.2.

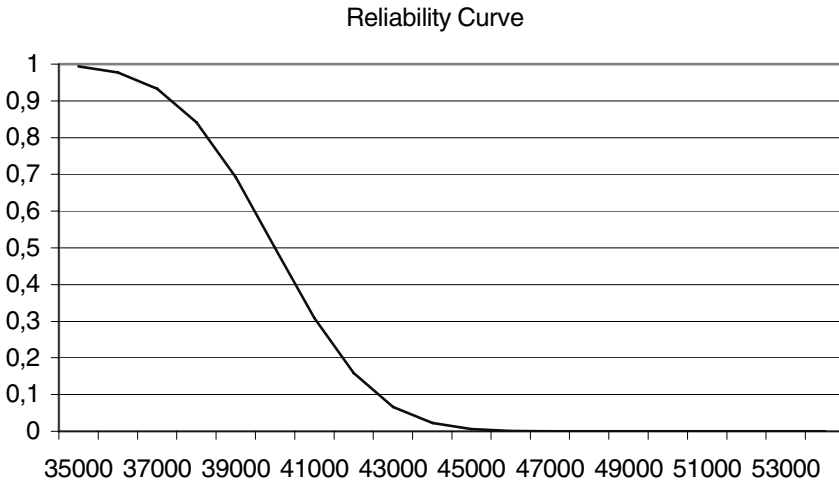
The normal distribution is flexible enough to make it a very useful empirical model. It can be theoretically derived under assumptions matching many failure mechanisms. Some of these are corrosion, migration, crack growth, and in general, failures resulting from chemical reactions or processes. That does not mean that the normal is always the correct model for these mechanisms, but it does perhaps explain why it has been empirically successful in so many of these cases.

### Log Normal Distribution

The log normal lifetime distribution is a very flexible model that can empirically fit many types of failure data. This distribution, with its applications in maintainability engineering, is able to model failure probabilities of repairable systems and to model the uncertainty in failure rate information. The log normal density function is given by

$$f(t) = \frac{1}{\sigma t \sqrt{2\pi}} e^{-\frac{1}{2} \left( \frac{\ln t - \mu}{\sigma} \right)^2} \quad t \geq 0 \quad (2.11)$$

where  $\mu$  and  $\sigma$  are parameters such that  $-\infty < \mu < \infty$ , and  $\sigma > 0$ . Note that  $\mu$  and  $\sigma$  are not the mean and standard deviations of the distribution.



**Figure 2.2.** Normal reliability plot vs time

The relationship to the normal (just take natural logarithms of all the data and time points and you have “normal” data) makes it easy to work with many good software analysis programs available to treat normal data.

Mathematically, if a random variable  $X$  is defined as  $X = \ln T$ , then  $X$  is normally distributed with a mean of  $\mu$  and a variance of  $\sigma^2$ . That is,

$$E(X) = E(\ln T) = \mu$$

and

$$V(X) = V(\ln T) = \sigma^2.$$

Since  $T = e^X$ , the mean of the log normal distribution can be found by using the normal distribution. Consider that

$$E(T) = E(e^X) = \int_{-\infty}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} e^{\left[x - \frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right]} dx$$

and by rearrangement of the exponent, this integral becomes

$$E(T) = e^{\mu + \frac{\sigma^2}{2}} \int_{-\infty}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2\sigma^2}[x-(\mu+\sigma^2)]^2} dx$$

Thus, the mean of the log normal distribution is

$$E(T) = e^{\mu + \frac{\sigma^2}{2}}$$

Proceeding in a similar manner,

$$E(T^2) = E(e^{2X}) = e^{2(\mu + \sigma^2)}$$

thus, the variance for the log normal is

$$V(T) = e^{2\mu + \sigma^2} (e^{\sigma^2} - 1)$$

The cumulative distribution function for the log normal is

$$F(t) = \int_0^t \frac{1}{\sigma s\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\ln s - \mu}{\sigma}\right)^2} ds$$

and this can be related to the standard normal deviate  $Z$  by

$$\begin{aligned} F(t) &= P[T \leq t] = P(\ln T \leq \ln t) \\ &= P\left[Z \leq \frac{\ln t - \mu}{\sigma}\right] \end{aligned}$$

Therefore, the reliability function is given by

$$R(t) = P\left[Z > \frac{\ln t - \mu}{\sigma}\right] \quad (2.12)$$

and the hazard function would be

$$h(t) = \frac{f(t)}{R(t)} = \frac{\Phi\left(\frac{\ln t - \mu}{\sigma}\right)}{\sigma t R(t)}$$

where  $\Phi$  is a cdf of standard normal density.

*Example 2.6:* The failure time of a certain component is log normal distributed with  $\mu = 5$  and  $\sigma = 1$ . Find the reliability of the component and the hazard rate for a life of 50 time units.

*Solution:* Substituting the numerical values of  $\mu$ ,  $\sigma$ , and  $t$  into equation (2.12), we compute

$$\begin{aligned} R(50) &= P\left[Z > \frac{\ln 50 - 5}{1}\right] = P[Z > -1.09] \\ &= 0.8621 \end{aligned}$$

Similarly, the hazard function is given by

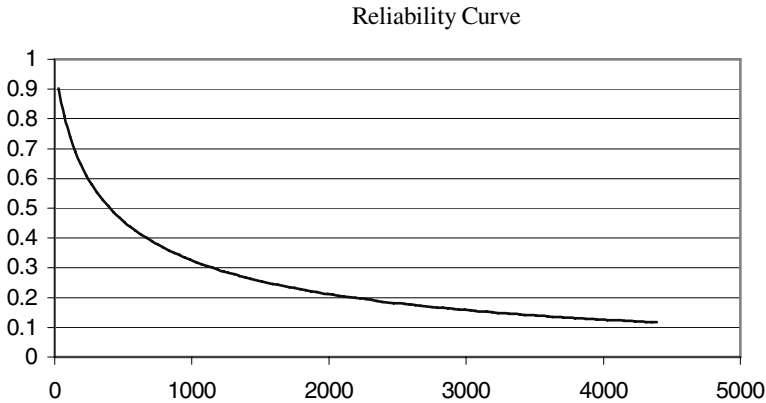
$$h(50) = \frac{\Phi\left(\frac{\ln 50 - 5}{1}\right)}{50(1)(0.8621)} = 0.032 \text{ failures/unit.}$$

Thus, values for the log normal distribution are easily computed by using the standard normal tables.

*Example 2.7:* The failure time of a part is log normal distributed with  $\mu = 6$  and  $\sigma = 2$ . Find the part reliability for a life of 200 time units.

*Solution:* The reliability for the part of 200 time units is

$$\begin{aligned} R(200) &= P\left(Z > \frac{\ln 200 - 6}{2}\right) = P(Z > -0.35) \\ &= 0.6368 \end{aligned}$$



**Figure 2.3.** Log normal reliability plot vs time

The log normal lifetime model, like the normal, is flexible enough to make it a very useful empirical model. Figure 2.3 shows the reliability of the log normal vs time. It can be theoretically derived under assumptions matching many failure

mechanisms. Some of these are: corrosion and crack growth, and in general, failures resulting from chemical reactions or processes.

### Weibull Distribution

The exponential distribution is often limited in applicability owing to the memoryless property. The Weibull distribution (Weibull 1951) is a generalization of the exponential distribution and is commonly used to represent fatigue life, ball bearing life, and vacuum tube life. The Weibull distribution is extremely flexible and appropriate for modeling component lifetimes with fluctuating hazard rate functions and for representing various types of engineering applications. The three-parameters probability density function is

$$f(t) = \frac{\beta(t-\gamma)^{\beta-1}}{\theta^\beta} e^{-\left(\frac{t-\gamma}{\theta}\right)^\beta} \quad t \geq \gamma \geq 0$$

where  $\theta$  and  $\beta$  are known as the scale and shape parameters, respectively, and  $\gamma$  is known as the location parameter. These parameters are always positive. By using different parameters, this distribution can follow the exponential distribution, the normal distribution, *etc.* It is clear that, for  $t \geq \gamma$ , the reliability function  $R(t)$  is

$$R(t) = e^{-\left(\frac{t-\gamma}{\theta}\right)^\beta} \quad \text{for } t > \gamma > 0, \beta > 0, \theta > 0 \quad (2.13)$$

hence,

$$h(t) = \frac{\beta(t-\gamma)^{\beta-1}}{\theta^\beta} \quad t > \gamma > 0, \beta > 0, \theta > 0 \quad (2.14)$$

It can be shown that the hazard function is decreasing for  $\beta < 1$ , increasing for  $\beta > 1$ , and constant when  $\beta = 1$ .

**Example 2.8:** The failure time of a certain component has a Weibull distribution with  $\beta = 4$ ,  $\theta = 2000$ , and  $\gamma = 1000$ . Find the reliability of the component and the hazard rate for an operating time of 1500 hours.

**Solution:** A direct substitution into equation (2.13) yields

$$R(1500) = e^{-\left(\frac{1500-1000}{2000}\right)^4} = 0.996$$

Using equation (2.14), the desired hazard function is given by

$$\begin{aligned} h(1500) &= \frac{4(1500-1000)^{4-1}}{(2000)^4} \\ &= 3.13 \times 10^{-5} \text{ failures/hour} \end{aligned}$$

Note that the Rayleigh and exponential distributions are special cases of the Weibull distribution at  $\beta = 2$ ,  $\gamma = 0$ , and  $\beta = 1$ ,  $\gamma = 0$ , respectively. For example, when  $\beta = 1$  and  $\gamma = 0$ , the reliability of the Weibull distribution function in equation (2.13) reduces to

$$R(t) = e^{-\frac{t}{\theta}}$$

and the hazard function given in equation (2.14) reduces to  $1/\theta$ , a constant. Thus, the exponential is a special case of the Weibull distribution. Similarly, when  $\gamma = 0$  and  $\beta = 2$ , the Weibull probability density function becomes the Rayleigh density function. That is

$$f(t) = \frac{2}{\theta} t e^{-\frac{t^2}{\theta}} \quad \text{for } \theta > 0, t \geq 0$$

### Other Forms of Weibull Distributions

The Weibull distribution again is widely used in engineering applications. It was originally proposed for representing the distribution of the breaking strength of materials. The Weibull model is very flexible and also has theoretical justification in many applications as a purely empirical model. Another form of Weibull probability density function is, for example,

$$f(x) = \lambda \gamma x^{\gamma-1} e^{-\lambda x^\gamma} \quad (2.15)$$

When  $\gamma=2$ , the density function becomes a Rayleigh distribution.

It can easily be shown that the mean, variance and reliability of the above Weibull distribution are, respectively, as follows:

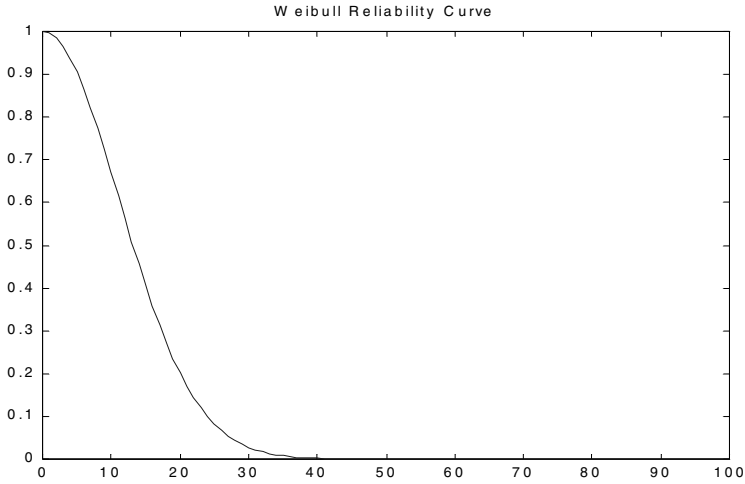
$$\begin{aligned} \text{Mean} &= \lambda^{\frac{1}{\gamma}} \Gamma\left(1 + \frac{1}{\gamma}\right) \\ \text{Variance} &= \lambda^{\frac{2}{\gamma}} \left( \Gamma\left(1 + \frac{2}{\gamma}\right) - \left( \Gamma\left(1 + \frac{1}{\gamma}\right) \right)^2 \right) \\ \text{Reliability} &= e^{-\lambda x^\gamma} \end{aligned} \quad (2.16)$$

*Example 2.9:* The time to failure of a part has a Weibull distribution with  $\frac{1}{\lambda} = 250$  (measured in  $10^5$  cycles) and  $\gamma=2$ . Find the part reliability at  $10^6$  cycles.

*Solution:* The part reliability at  $10^6$  cycles is

$$R(10^6) = e^{-(10^6)^2 / 250} = 0.6703$$

The resulting reliability function is shown in Figure 2.4.



**Figure 2.4.** Weibull reliability function vs time

### Gamma Distribution

Gamma distribution can be used as a failure probability function for components whose distribution is skewed. The failure density function for a gamma distribution is

$$f(t) = \frac{t^{\alpha-1}}{\beta^\alpha \Gamma(\alpha)} e^{-\frac{t}{\beta}} \quad t \geq 0, \quad \alpha, \beta > 0 \quad (2.17)$$

where  $\alpha$  is the shape parameter and  $\beta$  is the scale parameter. Hence,

$$R(t) = \int_t^\infty \frac{1}{\beta^\alpha \Gamma(\alpha)} s^{\alpha-1} e^{-\frac{s}{\beta}} ds$$

If  $\alpha$  is an integer, it can be shown by successive integration by parts that

$$R(t) = e^{-\frac{t}{\beta}} \sum_{i=0}^{\alpha-1} \frac{(\frac{t}{\beta})^i}{i!} \quad (2.18)$$

and

$$h(t) = \frac{f(t)}{R(t)} = \frac{\frac{1}{\beta^\alpha \Gamma(\alpha)} t^{\alpha-1} e^{-\frac{t}{\beta}}}{e^{-\frac{t}{\beta}} \sum_{i=0}^{\alpha-1} \frac{(\frac{t}{\beta})^i}{i!}}$$

The gamma density function has shapes that are very similar to the Weibull distribution. At  $\alpha = 1$ , the gamma distribution becomes the exponential distribution with the constant failure rate  $1/\beta$ . The gamma distribution can also be used to model the time to the  $n^{\text{th}}$  failure of a system if the underlying failure distribution is exponential. Thus, if  $X_i$  is exponentially distributed with parameter  $\theta = 1/\beta$ , then  $T = X_1 + X_2 + \dots + X_n$ , is gamma distributed with parameters  $\beta$  and  $n$ .

*Example 2.10:* The time to failure of a component has a gamma distribution with  $\alpha = 3$  and  $\beta = 5$ . Determine the reliability of the component and the hazard rate at 10 time-units.

*Solution:* Using equation (2.18), we compute

$$R(10) = e^{-\frac{10}{5}} \sum_{i=0}^2 \frac{\left(\frac{10}{5}\right)^i}{i!} = 0.6767$$

From equation (2.17), we obtain

$$h(10) = \frac{f(10)}{R(10)} = \frac{0.054}{0.6767} = 0.798 \text{ failures/unit time}$$

The other form of the gamma probability density function can be written as follows:

$$f(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x} \quad \text{for } x > 0 \quad (2.19)$$

This pdf is characterized by two parameters: shape parameter  $\alpha$  and scale parameter  $\beta$ . When  $0 < \alpha < 1$ , the failure rate monotonically decreases; when  $\alpha > 1$ , the failure rate monotonically increase; when  $\alpha = 1$  the failure rate is constant.

The mean, variance and reliability of the density function in equation (2.19) are, respectively,

$$\begin{aligned} \text{Mean (MTTF)} &= \frac{\alpha}{\beta} \\ \text{Variance} &= \frac{\alpha}{\beta^2} \\ \text{Reliability} &= \int_t^\infty \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x} dx \end{aligned}$$

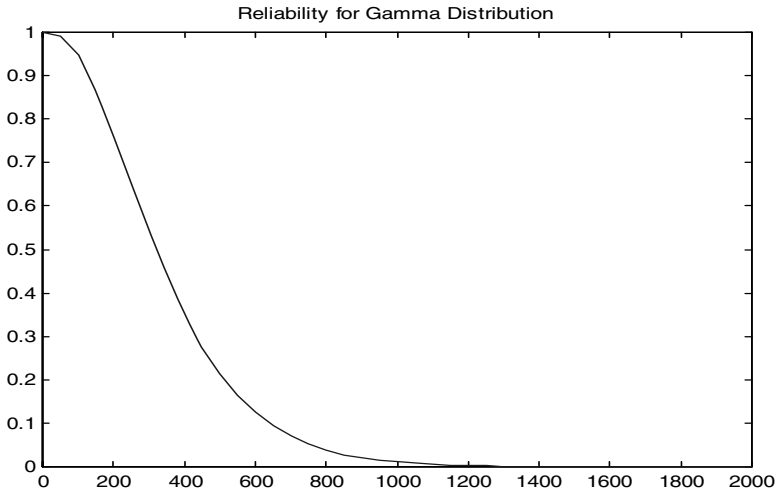
*Example 2.11:* A mechanical system time to failure is gamma distribution with  $\alpha=3$  and  $1/\beta=120$ . Find the system reliability at 280 hours.

*Solution:* The system reliability at 280 hours is given by

$$R(280) = e^{-\frac{280}{120}} \sum_{k=0}^2 \frac{\left(\frac{280}{120}\right)^k}{k!} = 0.85119$$

and the resulting reliability plot is shown in Figure 2.5.

The gamma model is a flexible lifetime model that may offer a good fit to some sets of failure data. It is not, however, widely used as a lifetime distribution model for common failure mechanisms. A common use of the gamma lifetime model occurs in Bayesian reliability applications.



**Figure 2.5.** Gamma reliability function vs time

### Beta Distribution

The two-parameter Beta density function,  $f(t)$ , is given by

$$f(t) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} t^{\alpha-1} (1-t)^{\beta-1} \quad 0 < t < 1, \alpha > 0, \beta > 0$$

where  $\alpha$  and  $\beta$  are the distribution parameters. This two-parameter distribution is commonly used in many reliability engineering applications.

### Pareto Distribution

The Pareto distribution was originally developed to model income in a population. Phenomena such as city population size, stock price fluctuations, and personal incomes have distributions with very long right tails. The probability density function of the Pareto distribution is given by

$$f(t) = \frac{\alpha k^{\alpha}}{t^{\alpha+1}} \quad k \leq t \leq \infty$$

The mean, variance and reliability of the Pareto distribution are, respectively,

$$\text{Mean} = k / (\alpha - 1) \quad \text{for } \alpha > 1$$

$$\text{Variance} = \frac{\alpha k^2}{(\alpha - 1)^2 (\alpha - 2)} \quad \text{for } \alpha > 2$$

$$\text{Reliability} = \left( \frac{k}{t} \right)^{\alpha}$$

The Pareto and log normal distributions have been commonly used to model the population size and economical incomes. The Pareto is used to fit the tail of the distribution, and the log normal is used to fit the rest of the distribution.



### Rayleigh Distribution

The Rayleigh function is a flexible lifetime distribution that can apply to many degradation process failure modes. The Rayleigh probability density function is

$$f(t) = \frac{t}{\sigma^2} e^{\left(\frac{-t^2}{2\sigma^2}\right)} \quad (2.20)$$

The mean, variance, and reliability of Rayleigh function are, respectively,

$$\begin{aligned} \text{Mean} &= \sigma \left( \frac{\pi}{2} \right)^{\frac{1}{2}} \\ \text{Variance} &= \left( 2 - \frac{\pi}{2} \right) \sigma^2 \\ \text{Reliability} &= e^{\frac{-\sigma^2}{2}} \end{aligned}$$

*Example 2.12:* Rolling resistance is a measure of the energy lost by a tire under load when it resists the force opposing its direction of travel. In a typical car, traveling at 60 miles per hour, about 20% of the engine power is used to overcome the rolling resistance of the tires.

A tire manufacturer introduces a new material that, when added to the tire rubber compound, significantly improves the tire rolling resistance but increases the wear rate of the tire tread. Analysis of a laboratory test of 150 tires shows that the failure rate of the new tire linearly increases with time (hours). It is expressed as

$$h(t) = 0.5 \times 10^{-8} t$$

Find the reliability of the tire at one year.

*Solution:* The reliability of the tire after one year (8760 hours) of use is

$$R(1_{\text{year}}) = e^{\frac{0.5}{2} \times 10^{-8} \times (8760)^2} = 0.8254$$

Figure 2.6 shows the resulting reliability function.

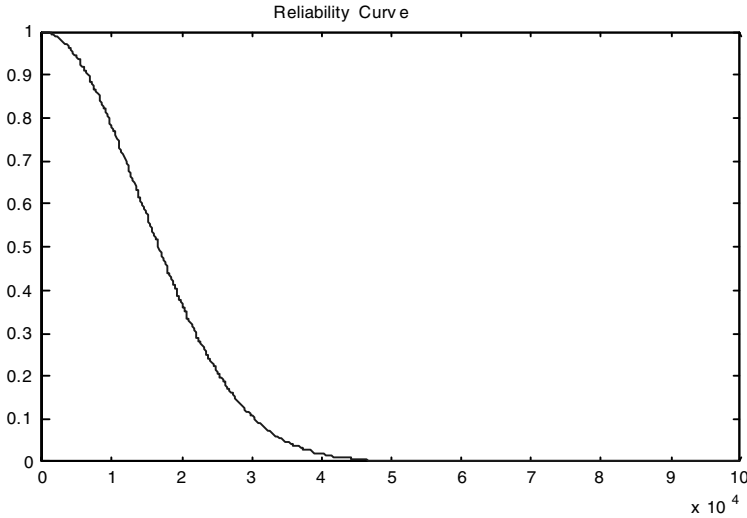


Figure 2.6. Rayleigh reliability function vs time

### Vtub-shaped Hazard Rate Distribution

Pham (2002a) recently developed a two-parameter lifetime distribution with a Vtub-shaped hazard rate, called *Pham distribution* - also known as *Loglog distribution*.

Note that the loglog distribution with Vtub-shaped and Weibull distribution with bathtub-shaped failure rates are not the same. As for the bathtub-shaped, after the infant mortality period, the useful life of the system begins. During its useful life, the system fails as a constant rate. This period is then followed by a wear out period during which the system starts slowly and increases with the onset of wear out. For the Vtub-shaped, after the infant mortality period, the system starts to experience at a relatively low increasing rate, but this is not constant, and then increases with failures due to aging.

The Pham probability density function is given as follows (Pham 2002a):

$$f(t) = \alpha \ln a \, t^{\alpha-1} a^{t^\alpha} e^{1-a^{t^\alpha}} \quad \text{for } t > 0, a > 0, \alpha > 0. \quad (2.21)$$

The Pham distribution and reliability functions are

$$F(t) = \int_0^t f(x) dx = 1 - e^{1-a^{t^\alpha}}$$

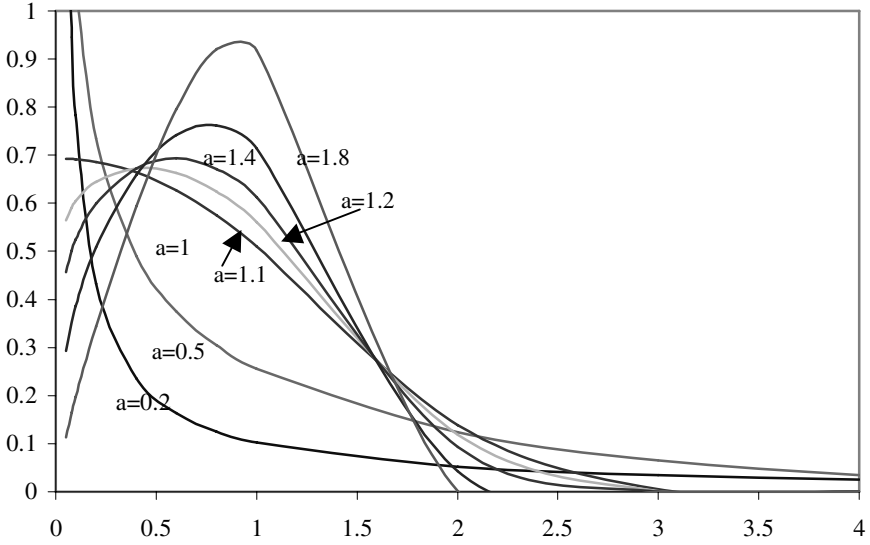
and

$$R(t) = e^{1-a^{t^\alpha}} \quad (2.22)$$

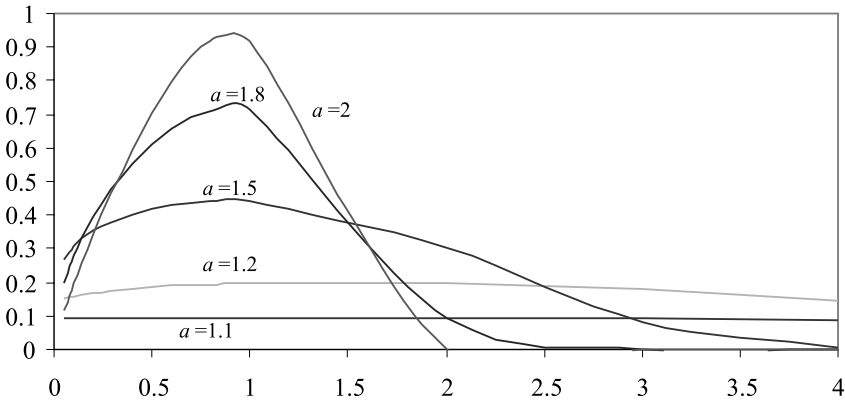
respectively. The corresponding failure rate of the Pham distribution is given by

$$h(t) = \alpha \ln(a) \, t^{\alpha-1} a^{t^\alpha} \quad (2.23)$$

Figures 2.7 and 2.8 describe the density function and failure rate function for various values of  $a$  and  $\alpha$ .



**Figure 2.7.** Probability density function for various values  $\alpha$  with  $a=2$



**Figure 2.8.** Probability density function for various values  $a$  with  $\alpha = 1.5$

### Two-Parameter Hazard Rate Function

This is a two-parameter function that can have increasing and decreasing hazard rates. The hazard rate,  $h(t)$ , the reliability function,  $R(t)$ , and the pdf are, respectively, given as follows

$$h(t) = \frac{n\lambda t^{n-1}}{\lambda t^n + 1} \quad \text{for } n \geq 1, \lambda > 0, t \geq 0 \quad (2.24)$$

$$R(t) = e^{-\ln(\lambda t^n + 1)} \quad (2.25)$$

and

$$f(t) = \frac{n\lambda t^{n-1}}{\lambda t^n + 1} e^{-\ln(\lambda t^n + 1)} \quad n \geq 1, \lambda > 0, t \geq 0 \quad (2.26)$$

where  $n$  = shape parameter;  $\lambda$  = scale parameter

### Three-Parameter Hazard Rate Function

This is a three-parameter distribution that can have increasing and decreasing hazard rates. The hazard rate,  $h(t)$ , is given as

$$h(t) = \frac{\lambda(b+1)[\ln(\lambda t + \alpha)]^b}{(\lambda t + \alpha)} \quad b \geq 0, \lambda > 0, \alpha \geq 0, t \geq 0 \quad (2.27)$$

The reliability function  $R(t)$  for  $\alpha = 1$  is

$$R(t) = e^{-(\ln(\lambda t + \alpha))^{b+1}}$$

The probability density function  $f(t)$  is

$$f(t) = e^{-[\ln(\lambda t + \alpha)]^{b+1}} \frac{\lambda(b+1)[\ln(\lambda t + \alpha)]^b}{(\lambda t + \alpha)} \quad (2.28)$$

where  $b$  = shape parameter,  $\lambda$  = scale parameter, and  $\alpha$  = location parameter.

## 2.3 A Generalized Systemability Function

The traditional reliability definitions and its calculations have commonly been carried out through the failure rate function within a controlled laboratory-test environment. In other words, such reliability functions are applied to the failure testing data and then utilized to make predictions on the reliability of the system used in the field. The underlying assumption for such calculation is that the field environments and the testing environments are the same.

By definition, a mathematical reliability function is the probability that a system will be successful in the interval from time 0 to time  $t$ , given by

$$R(t) = \int_t^\infty f(s)ds = e^{-\int_0^t h(s)ds} \quad (2.29)$$

where  $f(s)$  and  $h(s)$  are, respectively, the failure time density and failure rate function.

The operating environments are, however, often unknown and yet different due to the uncertainties of environments in the field (Pham and Xie 2003). A new look at how reliability researchers can take account of the randomness of the field environments into mathematical reliability modeling covering system failure in the field is great interest.

Pham (2005a) recently developed a new mathematical function called *systemability*, considering the uncertainty of the operational environments in the function for predicting the reliability of systems.

*Notation*

$h_i(t)$	$i^{\text{th}}$ component hazard rate function
$R_i(t)$	$i^{\text{th}}$ component reliability function
$\lambda_i$	Intensity parameter of Weibull distribution for $i^{\text{th}}$ component
$\underline{\lambda}$	$\underline{\lambda} = (\lambda_1, \lambda_2, \lambda_3, \dots, \lambda_n)$ .
$\gamma_i$	Shape parameter of Weibull distribution for $i^{\text{th}}$ component
$\underline{\gamma}$	$\underline{\gamma} = (\gamma_1, \gamma_2, \gamma_3, \dots, \gamma_n)$ .
$\eta$	A common environment factor
$G(\eta)$	Cumulative distribution function of $\eta$
$\alpha$	Shape parameter of Gamma distribution
$\beta$	Scale parameter of Gamma distribution

**2.3.1 Systemability Definition**

This section discusses a definition of systemability function.

**Definition 2.2 (Pham 2005a):** *Systemability* is defined as the probability that the system will perform its intended function for a specified mission time under the random operational environments.

In a mathematical form, the *systemability* function is given by

$$R_s(t) = \int_{\eta} e^{-\eta \int_0^t h(s) ds} dG(\eta) \quad (2.30)$$

where  $\eta$  is a random variable that represents the system operational environments with a distribution function  $G$ .

This new function captures the uncertainty of complex operational environments of systems in terms of the system failure rate. It also would reflect the reliability estimation of the system in the field.

If we assume that  $\eta$  has a gamma distribution with parameters  $\alpha$  and  $\beta$ , i.e.,  $\eta \sim \text{gamma}(\alpha, \beta)$  where the pdf of  $\eta$  is given by

$$f_{\eta}(x) = \frac{\beta^{\alpha} x^{\alpha-1} e^{-\beta x}}{\Gamma(\alpha)} \quad \text{for } \alpha, \beta > 0; x \geq 0 \quad (2.31)$$

then the systemability function of the system in equation (2.30), using the Laplace transform (see Appendix 2), is given by

$$R_s(t) = \left[ \frac{\beta}{\beta + \int_0^t h(s) ds} \right]^{\alpha} \quad (2.32)$$

### 2.3.2 Systemability Calculations

This subsection presents several systemability results and variances of some system configurations such as series, parallel, and  $k$ -out-of- $n$  systems (Pham 2005a). Consider the following assumptions:

1. A system consists of  $n$  independent components where the system is subject to a random operational environment  $\eta$ .
2.  $i^{\text{th}}$  component lifetime is assumed to follow the Weibull density function, *i.e.*

$$\text{Component hazard rate } h_i(t) = \lambda_i \gamma_i t^{\gamma_i - 1} \quad (2.33)$$

$$\text{Component reliability } R_i(t) = e^{-\lambda_i t^{\gamma_i}} \quad t > 0 \quad (2.34)$$

Given common environment factor  $\eta \sim \text{gamma}(\alpha, \beta)$ , the systemability functions for different system structures can be obtained as follows.

#### Series System Configuration

In a series system, all components must operate successfully if the system is to function. The conditional reliability function of series systems subject to an actual operational random environment  $\eta$  is given by

$$R_{\text{Series}}(t \mid \underline{\lambda}, \underline{\gamma}) = e^{\left(-\eta \sum_{i=1}^n \lambda_i t^{\gamma_i}\right)} \quad (2.35)$$

The series systemability is given as follows

$$R_{\text{Series}}(t \mid \underline{\lambda}, \underline{\gamma}) = \int_{\eta} \exp\left(-\eta \sum_{i=1}^n \lambda_i t^{\gamma_i}\right) dG(\eta) = \left[ \frac{\beta}{\beta + \sum_{i=1}^n \lambda_i t^{\gamma_i}} \right]^{\alpha} \quad (2.36)$$

The variance of a general function  $R(t)$  is given by

$$\text{Var}[R(t)] = E[R^2(t)] - (E[R(t)])^2 \quad (2.37)$$

Given  $\eta \sim \text{gamma}(\alpha, \beta)$ , the variance of systemability for any system structure can be easily obtained. Therefore, the variance of series systemability is given by

$$\begin{aligned} \text{Var}[R_{\text{Series}}(t \mid \underline{\lambda}, \underline{\gamma})] &= \int_{\eta} \exp\left(-\eta \left(2 \sum_{i=1}^n \lambda_i t^{\gamma_i}\right)\right) dG(\eta) - \\ &\quad \left( \int_{\eta} \exp\left(-\eta \sum_{i=1}^n \lambda_i t^{\gamma_i}\right) dG(\eta) \right)^2 \end{aligned} \quad (2.38)$$

or

$$\text{Var}[R_{\text{Series}}(t \mid \underline{\lambda}, \underline{\gamma})] = \left[ \frac{\beta}{\beta + 2 \sum_{i=1}^n \lambda_i t^{\gamma_i}} \right]^{\alpha} - \left[ \frac{\beta}{\beta + \sum_{i=1}^n \lambda_i t^{\gamma_i}} \right]^{2\alpha} \quad (2.39)$$

### Parallel System Configuration

A parallel system is a system that is not considered to have failed unless all components have failed. The conditional reliability function of parallel systems subject to the uncertainty operational environment  $\eta$  is given by

$$\begin{aligned}
 R_{Parallel}(t | \underline{\eta}, \underline{\lambda}, \underline{\gamma}) = & \exp(-\eta \lambda_{t_1} t^{\gamma_{t_1}}) - \sum_{\substack{i_1, i_2=1 \\ i_1 \neq i_2}}^n \exp(-\eta (\lambda_{t_1} t^{\gamma_{t_1}} + \lambda_{t_2} t^{\gamma_{t_2}})) + \\
 & \sum_{\substack{i_1, i_2, i_3=1 \\ i_1 \neq i_2 \neq i_3}}^n \exp(-\eta (\lambda_{t_1} t^{\gamma_{t_1}} + \lambda_{t_2} t^{\gamma_{t_2}} + \lambda_{t_3} t^{\gamma_{t_3}})) - \\
 & \dots \\
 & + (-1)^{n-1} \exp\left(-\eta \sum_{i=1}^n \lambda_{t_i} t^{\gamma_{t_i}}\right)
 \end{aligned} \tag{2.40}$$

Hence, the parallel systemability is given by

$$\begin{aligned}
 R_{parallel}(t | \underline{\lambda}, \underline{\gamma}) = & \sum_{i=1}^n \left[ \frac{\beta}{\beta + \lambda_{t_i} t^{\gamma_{t_i}}} \right]^\alpha - \sum_{\substack{i_1, i_2=1 \\ i_1 \neq i_2}}^n \left[ \frac{\beta}{\beta + \lambda_{t_{i_1}} t^{\gamma_{t_{i_1}}} + \lambda_{t_{i_2}} t^{\gamma_{t_{i_2}}}} \right]^\alpha + \\
 & \sum_{\substack{i_1, i_2, i_3=1 \\ i_1 \neq i_2 \neq i_3}}^n \left[ \frac{\beta}{\beta + \lambda_{t_{i_1}} t^{\gamma_{t_{i_1}}} + \lambda_{t_{i_2}} t^{\gamma_{t_{i_2}}} + \lambda_{t_{i_3}} t^{\gamma_{t_{i_3}}}} \right]^\alpha - \\
 & \dots \\
 & + (-1)^{n-1} \left[ \frac{\beta}{\beta + \sum_{i=1}^n \lambda_{t_i} t^{\gamma_{t_i}}} \right]^\alpha
 \end{aligned} \tag{2.41}$$

or

$$R_{parallel}(t | \underline{\lambda}, \underline{\gamma}) = \sum_{k=1}^n (-1)^{k-1} \sum_{\substack{i_1, i_2, \dots, i_k=1 \\ i_1 \neq i_2 \dots \neq i_k}}^n \left[ \frac{\beta}{\beta + \sum_{j=i_1, \dots, i_k} \lambda_{t_j} t^{\gamma_{t_j}}} \right]^\alpha \tag{2.42}$$

To simplify the calculation of a general n-component parallel system, we only consider here a parallel system consisting of two components. It is easy to see that the second-order moments of the systemability function can be written as

$$\begin{aligned}
 E[R_{Parallel}^2(t | \underline{\lambda}, \underline{\gamma})] = & \int_{\eta} (e^{-2\eta \lambda_{t_1} t^{\gamma_{t_1}}} + e^{-2\eta \lambda_{t_2} t^{\gamma_{t_2}}} + e^{-2\eta (\lambda_{t_1} t^{\gamma_{t_1}} + \lambda_{t_2} t^{\gamma_{t_2}})} \\
 & + e^{-\eta (\lambda_{t_1} t^{\gamma_{t_1}} + \lambda_{t_2} t^{\gamma_{t_2}})} - e^{-\eta (2\lambda_{t_1} t^{\gamma_{t_1}} + \lambda_{t_2} t^{\gamma_{t_2}})} - e^{-\eta (\lambda_{t_1} t^{\gamma_{t_1}} + 2\lambda_{t_2} t^{\gamma_{t_2}})}) d G(\eta)
 \end{aligned}$$

The variance of series systemability of a two-component parallel system is given by

$$\begin{aligned}
 \text{Var}[R_{\text{parallel}}(t | \underline{\lambda}, \underline{\gamma})] = & \left[ \frac{\beta}{\beta + 2\lambda_1 t^{\gamma_1}} \right]^\alpha + \left[ \frac{\beta}{\beta + 2\lambda_2 t^{\gamma_2}} \right]^\alpha + \\
 & \left[ \frac{\beta}{\beta + 2\lambda_1 t^{\gamma_1} + 2\lambda_2 t^{\gamma_2}} \right]^\alpha + \left[ \frac{\beta}{\beta + \lambda_1 t^{\gamma_1} + \lambda_2 t^{\gamma_2}} \right]^\alpha - \\
 & \left[ \frac{\beta}{\beta + 2\lambda_1 t^{\gamma_1} + \lambda_2 t^{\gamma_2}} \right]^\alpha - \left[ \frac{\beta}{\beta + \lambda_1 t^{\gamma_1} + 2\lambda_2 t^{\gamma_2}} \right]^\alpha - \\
 & \left[ \left[ \frac{\beta}{\beta + \lambda_1 t^{\gamma_1}} \right]^\alpha + \left[ \frac{\beta}{\beta + \lambda_2 t^{\gamma_2}} \right]^\alpha - \left[ \frac{\beta}{\beta + \lambda_1 t^{\gamma_1} + \lambda_2 t^{\gamma_2}} \right]^\alpha \right]^2
 \end{aligned} \tag{2.43}$$

### ***k-out-of-n* System Configuration**

In a *k-out-of-n* configuration, the system will operate if at least *k* out of *n* components are operating. To simplify the complexity of the systemability function, we assume that all the components in the *k-out-of-n* systems are identical. Therefore, for a given common environment  $\eta$ , the conditional reliability function of a component is given by

$$R(t | \eta, \lambda, \gamma) = e^{-\eta \lambda t^\gamma} \tag{2.44}$$

The conditional reliability function of *k-out-of-n* systems subject to the uncertainty operational environment  $\eta$  can be obtained as follows:

$$R_{k\text{-out-of-}n}(t | \eta, \lambda, \gamma) = \sum_{j=k}^n \binom{n}{j} e^{-\eta j \lambda t^\gamma} (1 - e^{-\eta \lambda t^\gamma})^{(n-j)} \tag{2.45}$$

Note that

$$(1 - e^{-\eta \lambda t^\gamma})^{(n-j)} = \sum_{l=0}^{n-j} \binom{n-j}{l} (-e^{-\eta \lambda t^\gamma})^l$$

The conditional reliability function of *k-out-of-n* systems, from equation (2.45), can be rewritten as

$$R_{k\text{-out-of-}n}(t | \eta, \lambda, \gamma) = \sum_{j=k}^n \binom{n}{j} \sum_{l=0}^{n-j} \binom{n-j}{l} (-1)^l e^{-\eta(j+l)\lambda t^\gamma} \tag{2.46}$$

Then if  $\eta \sim \text{gamma}(\alpha, \beta)$  then the *k-out-of-n* systemability is given by

$$R_{(T_1, \dots, T_n)}(t | \lambda, \gamma) = \sum_{j=k}^n \binom{n}{j} \sum_{l=0}^{n-j} \binom{n-j}{l} (-1)^l \left[ \frac{\beta}{\beta + \lambda(j+l)t^\gamma} \right]^\alpha \tag{2.47}$$

It can be easily shown that

$$R_{k\text{-out-of-}n}^2(t | \eta, \lambda, \gamma) = \sum_{i=k}^n \binom{n}{i} \sum_{j=k}^n \binom{n}{j} e^{-\eta(i+j)\lambda t^\gamma} (1 - e^{-\eta \lambda t^\gamma})^{(2n-i-j)} \tag{2.48}$$



Since

$$(1 - e^{-\eta \lambda t^\gamma})^{(2n-i-j)} = \sum_{l=0}^{2n-i-j} \binom{2n-i-j}{l} (-e^{-\eta \lambda t^\gamma})^l \quad (2.49)$$

we can rewrite equation (2.48), after several simplifications, as follows

$$R_{k-out-of-n}^2(t | \eta, \lambda, \gamma) = \sum_{i=k}^n \binom{n}{i} \sum_{j=k}^n \binom{n}{j} (-1)^l \sum_{l=0}^{2n-i-j} \binom{2n-i-j}{l} e^{-\eta(i+j+l)\lambda t^\gamma} \quad (2.50)$$

Therefore, the variance of  $k$ -out-of- $n$  system systemability function is given by

$$\begin{aligned} Var(R_{k/n}(t | \lambda, \gamma)) &= \int_{\eta} R_{k/n}^2(t | \eta, \lambda, \gamma) dG(\eta) - \left[ \int_{\eta} R_{k/n}(t | \eta, \lambda, \gamma) dG(\eta) \right]^2 \\ &= \sum_{i=k}^n \binom{n}{i} \sum_{j=k}^n \binom{n}{j} \sum_{l=0}^{2n-i-j} \binom{2n-i-j}{l} (-1)^l \left( \frac{\beta}{\beta + (i+j+l)\lambda t^\gamma} \right)^2 \\ &\quad - \left( \sum_{j=k}^n \binom{n}{j} \sum_{l=0}^{n-j} \binom{n-j}{l} (-1)^l \left( \frac{\beta}{\beta + (j+l)\lambda t^\gamma} \right)^2 \right)^2 \end{aligned} \quad (2.51)$$

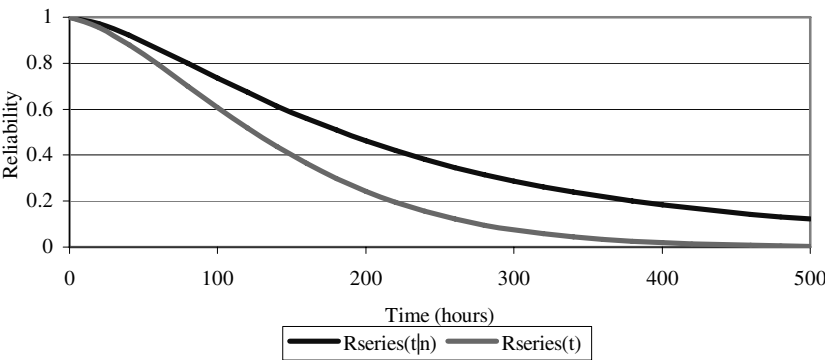
*Example 2.13:* Consider a  $k$ -out-of- $n$  system where  $\lambda = 0.0001$ ,  $\gamma = 1.5$ ,  $n = 5$ , and  $\eta \sim \text{gamma}(\alpha, \beta)$ . Calculate the systemability of various  $k$ -out-of- $n$  system configurations.

*Solution:* The systemability of generalized  $k$ -out-of-5 system configurations is given as follows:

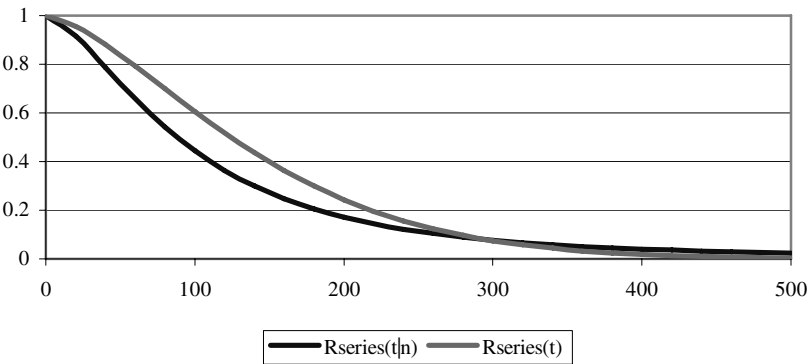
$$R_{k-out-of-n}(t | \lambda, \gamma) = \sum_{j=k}^5 \binom{5}{j} \sum_{l=0}^{5-j} \binom{5-j}{l} (-1)^l \left[ \frac{\beta}{\beta + \lambda(j+l)t^\gamma} \right]^\alpha \quad (2.52)$$

Figures 2.9 and 2.10 show the reliability function (conventional reliability function) and systemability function (equation 2.52) of a series system (here  $k=5$ ) for  $\alpha = 2, \beta = 3$  and for  $\alpha = 2, \beta = 1$ , respectively.

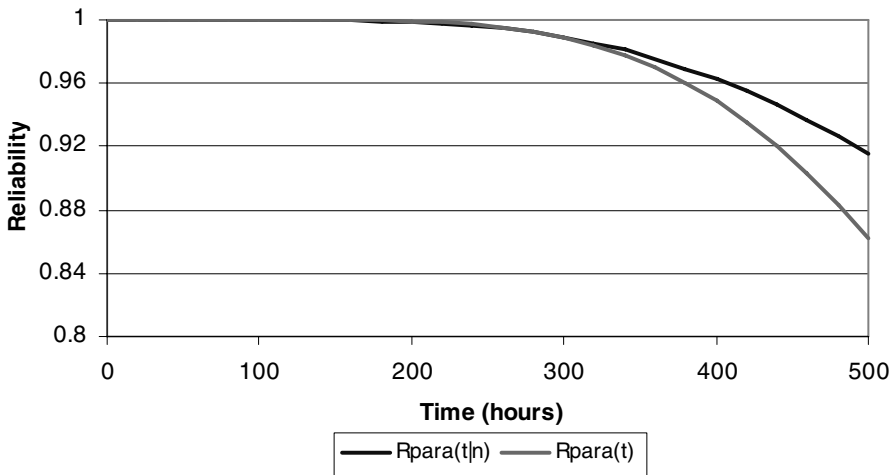
Figures 2.11 and 2.12 show the reliability and systemability functions of a parallel system (here  $k=1$ ) for  $\alpha = 2, \beta = 3$  and for  $\alpha = 2, \beta = 1$ , respectively. Similarly, Figures 2.13 and 2.14 show the reliability and systemability functions of a 3-out-of-5 system for  $\alpha = 2, \beta = 3$  and for  $\alpha = 2, \beta = 1$ , respectively.



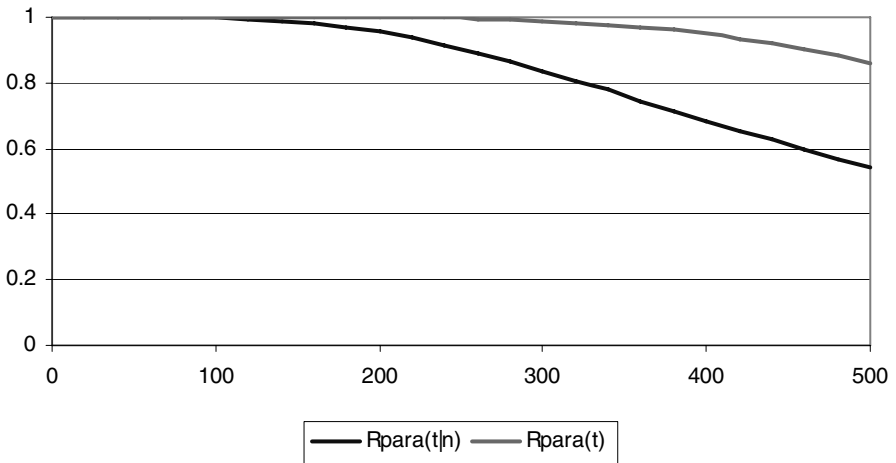
**Figure 2.9.** Comparisons of series system reliability vs systemability functions for  $\alpha=2$  and  $\beta=3$



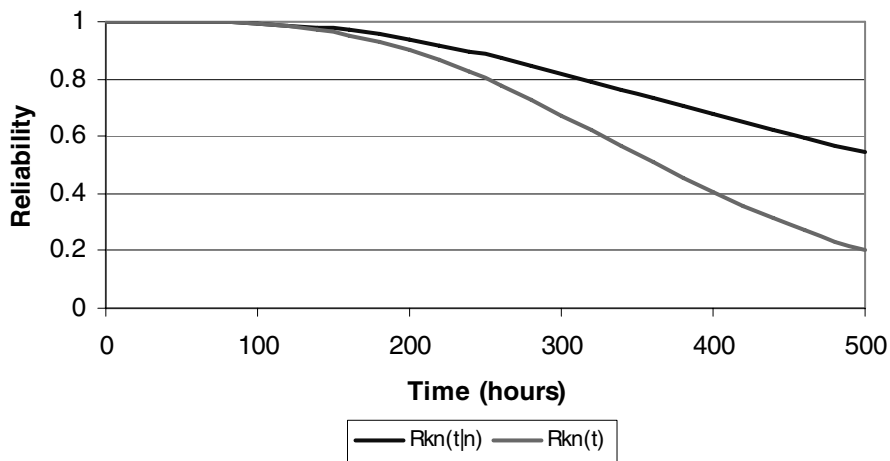
**Figure 2.10.** Comparisons of series system reliability vs. systemability functions for  $\alpha=2$  and  $\beta=1$



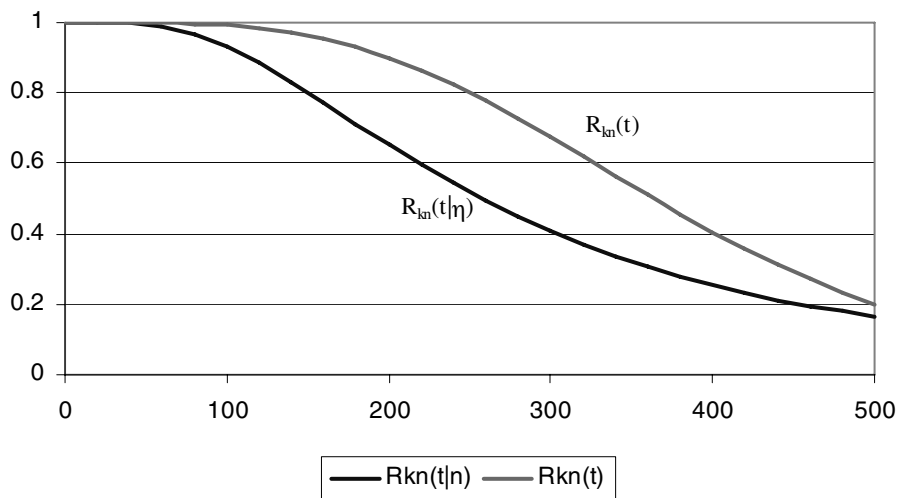
**Figure 2.11.** Comparisons of parallel system reliability vs systemability function for  $\alpha = 2$  and  $\beta = 3$



**Figure 2.12.** Comparisons of parallel system reliability vs Systemability functions for  $\alpha = 2$  and  $\beta = 1$



**Figure 2.13.** Comparisons of  $k$ -out-of- $n$  system reliability vs. systemability functions for  $\alpha=2$  and  $\beta=3$



**Figure 2.14.** Comparisons of  $k$ -out-of- $n$  system reliability vs. systemability functions for  $\alpha=2$  and  $\beta=1$

### Variance of Systemability Calculations

Assume  $\lambda = 0.00001$ ,  $\gamma = 1.5$ ,  $n = 3$ ,  $k = 2$ , and  $\eta \sim \text{gamma}(\alpha, \beta)$ , Figures 2.15 and 2.16 shows the systemability and its confidence intervals of a 2-out-of-3 system (Pham 1993) for  $\alpha=2, \beta=1$  and  $\alpha=2, \beta=2$ , respectively.

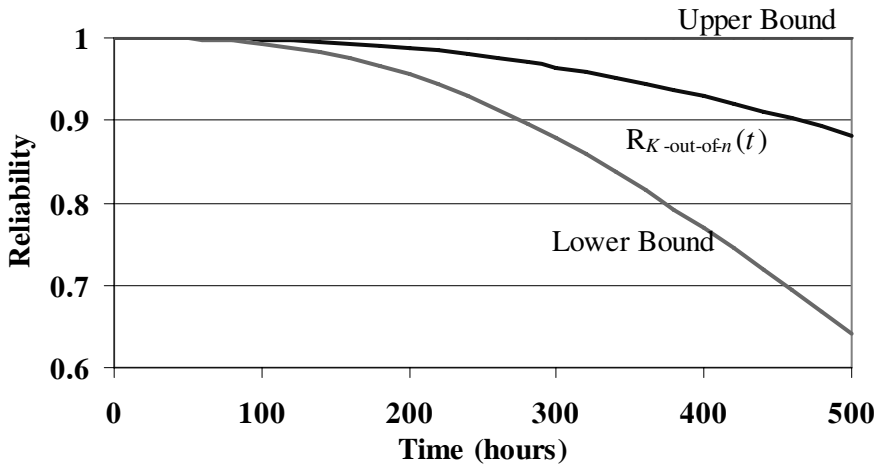


Figure 2.15. A 2-out-of-3 systemability and its 95% confidence interval where  $\alpha = 2$ ,  $\beta = 1$

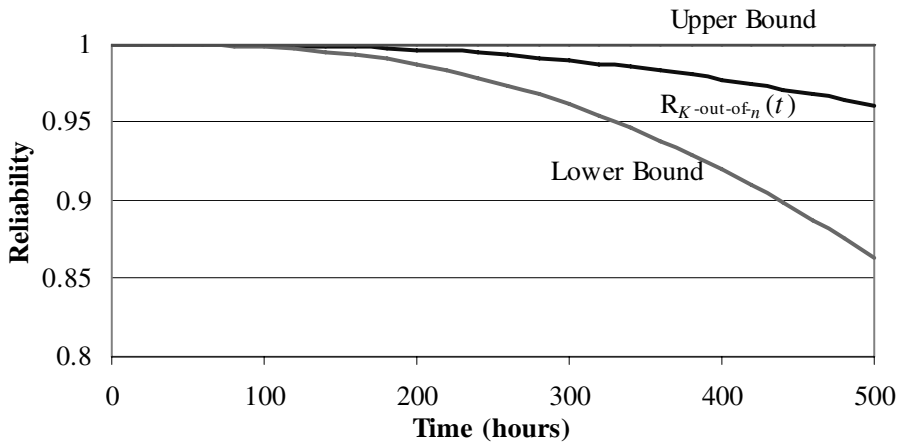


Figure 2.16. A 2-out-of-3 systemability and its 95% confidence interval ( $\alpha = 2$ ,  $\beta = 2$ )

## 2.4 System Reliability with Multiple Failure Modes

This section discusses various reliability and optimization aspects of systems subject to multiple types of failure. It is assumed that the system component states are statistically independent and identically distributed. Networks of relays, diode circuits, fluid flow valves, *etc.* are a few examples of systems having components subject to failure in either open or closed modes.

The designations “closed mode” and “short mode” both appear in this section, and we will use the two terms interchangeably. Redundancy can be used to enhance the reliability of a system without any change in the reliability of the

individual components that form the system. However, in a two-failure mode problem, redundancy may either increase or decrease the system's reliability. Therefore, adding components to the system may not increase the system reliability.

The reliability of a system subject to two kinds of failure is calculated as follows (Malon 1989):

$$\begin{aligned} \text{System reliability} &= \Pr\{\text{system works in both modes}\} \\ &= \Pr\{\text{system works in open mode}\} - \Pr\{\text{system fails in} \\ &\quad \text{closed mode}\} + \Pr\{\text{system fails in both modes}\} \end{aligned} \quad (2.53)$$

When the open- and closed-mode failure structures are dual of one another, *i.e.*  $\Pr\{\text{system fails in both modes}\} = 0$ , then the system reliability given by equation (2.53) becomes

$$\begin{aligned} \text{System reliability} &= 1 - \Pr\{\text{system fails in open mode}\} \\ &\quad - \Pr\{\text{system fails in closed mode}\} \end{aligned} \quad (2.54)$$

#### Notation

$q_0$	The open-mode failure probability of each component ( $p_0 = 1 - q_0$ )
$q_s$	The short-mode failure probability of each component ( $p_s = 1 - q_s$ )
$\lfloor x \rfloor$	The largest integer not exceeding $x$
*	Implies an optimal value

### 2.4.1 Reliability Calculations

#### The Series System

Consider a series system consisting of  $n$  components. In this series system, any one component failing in an open mode causes system failure in open mode whereas all components of the system must malfunction in short mode for the system to fail in closed mode.

The probabilities of system fails in open mode and fails in short mode are

$$F_0(n) = 1 - (1 - q_0)^n$$

and

$$F_s(n) = q_s^n$$

respectively. From equation (2.54), the system reliability is

$$R_s(n) = (1 - q_0)^n - q_s^n \quad (2.55)$$

where  $n$  is the number of identical and independent components. In a series arrangement, reliability with respect to closed system failure increases with the number of components, whereas reliability with respect to open system failure decreases.

**Theorem 2.1:** Let  $q_0$  and  $q_s$  be fixed. There exists an optimum number of components, say  $n^*$ , that maximizes the system reliability. If we define

$$n_0 = \frac{\log\left(\frac{q_0}{1-q_s}\right)}{\log\left(\frac{q_s}{1-q_0}\right)}$$

then the system reliability,  $R_s(n^*)$ , is maximum for

$$n^* = \begin{cases} \lfloor n_0 \rfloor + 1 & \text{if } n_0 \text{ is not an integer} \\ n_0 \text{ or } n_0 + 1 & \text{if } n_0 \text{ is an integer} \end{cases} \quad (2.56)$$

*Proof:* The proof is left as an exercise for the reader (see Problem 2.17).

*Example 2.14:* A switch has two failure modes: fail-open and fail-short. The probability of switch open-circuit failure and short-circuit failure are 0.1 and 0.2 respectively. A system consists of  $n$  switches wired in series. That is, given  $q_0 = 0.1$  and  $q_s = 0.2$ . Then

$$n_0 = \frac{\log\left(\frac{0.1}{1-0.2}\right)}{\log\left(\frac{0.2}{1-0.1}\right)} = 1.4$$

Thus,  $n^* = \lfloor 1.4 \rfloor + 1 = 2$ . Therefore, when  $n^* = 2$  the system reliability  $R_s(n) = 0.77$  is maximized.

### The Parallel System

Consider a parallel system consisting of  $n$  components. For a parallel configuration, all the components must fail in open mode or at least one component must malfunction in short mode to cause the system to fail completely. The system reliability is given by

$$R_p(n) = (1 - q_s)^n - q_0^n \quad (2.57)$$

where  $n$  is the number of components connected in parallel. In this case,  $(1 - q_s)^n$  represents the probability that no components fail in short mode, and  $q_0^n$  represents the probability that all components fail in open mode.

**Theorem 2.2:** If we define

$$n_0 = \frac{\log\left(\frac{q_s}{1-q_0}\right)}{\log\left(\frac{q_0}{1-q_s}\right)} \quad (2.58)$$

then the system reliability  $R_p(n^*)$  is maximum for

$$n^* = \begin{cases} \lfloor n_0 \rfloor + 1 & \text{if } n_0 \text{ is not an integer} \\ n_0 \text{ or } n_0 + 1 & \text{if } n_0 \text{ is an integer.} \end{cases} \quad (2.59)$$

*Proof:* The proof is left as an exercise for the reader (see Problem 2.18).

It is observed that, for any range of  $q_0$  and  $q_s$ , the optimal number of parallel components that maximizes the system reliability is one, if  $q_s > q_0$  (see Problem 2.19). For most other practical values of  $q_0$  and  $q_s$  the optimal number turns out to be two. In general, the optimal value of parallel components can be easily obtained using equation (2.58).

### The Parallel-Series System

Consider a system of components arranged so that there are  $m$  subsystems operating in parallel, each subsystem consisting of  $n$  identical components in series. Such an arrangement is called a parallel-series arrangement. The components are subject to two types of failure: failure in open mode and failure in short mode.

The systems are characterized by the following properties:

1. The system consists of  $m$  subsystems, each subsystem containing  $n$  i.i.d. components.
2. A component is either good, failed open, or failed short. Failed components can never become good, and there are no transitions between the open and short failure modes.
3. The system can be (a) good, (b) failed open (at least one component in each subsystem fails open), or (c) failed short (all the components in any subsystem fail short).
4. The unconditional probabilities of component failure in open and short modes are known and are constrained:  $q_o, q_s > 0$ ;  $q_o + q_s < 1$ .

The probabilities of a system failing in open mode and failing in short mode are given by

$$F_o(m) = [1 - (1 - q_o)^n]^m \quad (2.60)$$

$$\text{and} \quad F_s(m) = 1 - (1 - (q_s)^n)^m \quad (2.61)$$

respectively. The system reliability is

$$R_{ps}(n, m) = (1 - q_s^n)^m - [1 - (1 - q_o)^n]^m \quad (2.62)$$

An interesting example in Barlow and Proschan (1965) shows that there exists no pair  $n, m$  maximizing system reliability, since  $R_{ps}$  is made arbitrarily close to one by appropriate choice of  $m$  and  $n$ . To see this, let

$$a = \frac{\log q_s - \log(1 - q_o)}{\log q_s + \log(1 - q_o)} \quad M_n = q_s^{-n/(1+a)} \quad m_n = \lfloor M_n \rfloor$$



For given  $n$ , take  $m = m_n$ ; then one can rewrite equation (2.62) as:

$$R_{ps}(n, m_n) = (1 - q_s^n)^{m_n} - [1 - (1 - q_0)^n]^{m_n}$$

A straightforward computation yields

$$\lim_{n \rightarrow \infty} R_{ps}(n, m_n) = \lim_{n \rightarrow \infty} \{ (1 - q_s^n)^{m_n} - [1 - (1 - q_0)^n]^{m_n} \} = 1$$

For fixed  $n$ ,  $q_0$ , and  $q_s$ , one can determine the value of  $m$  that maximizes  $R_{ps}$ .

**Theorem 2.3 (Barlow and Proschan 1965):** Let  $n$ ,  $q_0$ , and  $q_s$  be fixed. The maximum value of  $R_{ps}(m)$  is attained at  $m^* = \lfloor m_0 \rfloor + 1$ , where

$$m_0 = \frac{n(\log p_0 - \log q_s)}{\log(1 - q_s^n) + \log(1 - p_0^n)} \quad (2.63)$$

If  $m_0$  is an integer, then  $m_0$  and  $m_0 + 1$  both maximize  $R_{ps}(m)$ .

*Proof:* The proof is left as an exercise for the reader (see Problem 20).

### The Series-Parallel System

The series-parallel structure is the dual of the parallel-series structure. We consider a system of components arranged so that there are  $m$  subsystems operating in series, each subsystem consisting of  $n$  identical components in parallel. Such an arrangement is called a series-parallel arrangement.

Failure in open mode of all the components in any subsystem makes the system unresponsive. Failure in closed (short) mode of a single component in each subsystem also makes the system unresponsive. The probabilities of system failure in open and short mode are given by

$$F_0(m) = 1 - (1 - q_0^n)^m \quad (2.64)$$

and

$$F_s(m) = [1 - (1 - q_s)^n]^m \quad (2.65)$$

respectively. The system reliability is

$$R(m) = (1 - q_0^n)^m - [1 - (1 - q_s)^n]^m \quad (2.66)$$

where  $m$  is the number of identical subsystems in series and  $n$  is the number of identical components in each parallel subsystem.

Barlow and Proschan (1965) show that there exists no pair  $(m, n)$  maximizing system reliability. For fixed  $m$ ,  $q_0$ , and  $q_s$  however, one can determine the value of  $n$  that maximizes the system reliability.

**Theorem 2.4 (Barlow and Proschan 1965):** Let  $n$ ,  $q_0$ , and  $q_s$  be fixed. The maximum value of  $R(m)$  is attained at  $m^* = \lfloor m_0 \rfloor + 1$ , where

$$m_0 = \frac{n(\log p_s - \log q_0)}{\log(1 - q_0^n) - \log(1 - p_s^n)} \quad (2.67)$$

If  $m_0$  is an integer, then  $m_0$  and  $m_0 + 1$  both maximize  $R(m)$ .

*Proof:* (see Problem 21).

### The $k$ -out-of- $n$ Systems

Consider a  $k$ -out-of- $n$  system consisting of  $n$  identical and independent components that can be either good or failed. The components are subject to two types of failure: failure in open mode and failure in closed mode. The  $k$  out of  $n$  system can fail when  $k$  or more components fail in closed mode or when  $(n - k + 1)$  or more components fail in open mode.

Applications of  $k$ -out-of- $n$  systems can be found in the areas of target detection, communication, and safety monitoring systems, and, particularly, in the area of human organizations. The following is an example in the area of human organizations (Nordmann and Pham 1999).

Consider a committee with  $n$  members who must decide to accept or reject innovation-oriented projects. The projects are of two types: "good" and "bad". It is assumed that the communication among the members is limited, and each member will make a yes-no decision on each project. A committee member can make two types of error: the error of accepting a bad project and the error of rejecting a good project. The committee will accept a project when  $k$  or more members accept it, and will reject a project when  $(n - k + 1)$  or more members reject it.

Thus, the two types of potential error of the committee are: (1) the acceptance of a bad project (which occurs when  $k$  or more members make the error of accepting a bad project); (2) the rejection of a good project (which occurs when  $(n - k + 1)$  or more members make the error of rejecting a good project).

This section determines the optimal  $k$  or  $n$  that maximizes the system reliability. We also study the effect of the system's parameters on the optimal  $k$  or  $n$ . The system fails in closed mode if and only if at least  $k$  of its  $n$  components fail in closed mode, and we obtain

$$F_s(k, n) = \sum_{i=k}^n \binom{n}{i} q_s^i p_s^{n-i} = 1 - \sum_{i=0}^{k-1} \binom{n}{i} q_s^i p_s^{n-i} \quad (2.68)$$

The system fails in open mode if and only if at least  $(n - k + 1)$  of its  $n$  components fail in open mode, that is:

$$F_0(k, n) = \sum_{i=n-k+1}^n \binom{n}{i} q_0^i p_0^{n-i} = \sum_{i=0}^{k-1} \binom{n}{i} p_0^i q_0^{n-i} \quad (2.69)$$

The system reliability is given by

$$R(k, n) = 1 - F_0(k, n) - F_s(k, n) = \sum_{i=0}^{k-1} \binom{n}{i} q_s^i p_s^{n-i} - \sum_{i=0}^{k-1} \binom{n}{i} p_0^i q_0^{n-i} \quad (2.70)$$

For a given  $k$ , we can find the optimum value of  $n$ , say  $n^*$ , that maximizes the system reliability.

**Theorem 2.5 (Pham 1989a):** For fixed  $k$ ,  $q_0$ , and  $q_s$ , the maximum value of  $R(k, n)$  is attained at  $n^* = \lfloor n_0 \rfloor$  where

$$n_0 = k \left[ 1 + \frac{\log \left( \frac{1-q_0}{q_s} \right)}{\log \left( \frac{1-q_s}{q_0} \right)} \right] \quad (2.71)$$

If  $n_0$  is an integer, both  $n_0$  and  $n_0 + 1$  maximize  $R(k, n)$ .

*Proof:* The proof is left as an exercise for the reader (see Problem 22).

This result shows that when  $n_0$  is an integer, both  $n^*-1$  and  $n^*$  maximize the system reliability  $R(k, n)$ . In such cases, the lower value will provide the more economical optimal configuration for the system. If  $q_0 = q_s$  the system reliability  $R(k, n)$  is maximized when  $n = 2k$  or  $2k-1$ . In this case, the optimum value of  $n$  does not depend on the value of  $q_0$  and  $q_s$  and the best choice for a decision voter is a majority voter; this system is also called a majority system (Pham, 1989a).

From Theorem 2.5, we understand that the optimal system size  $n^*$  depends on the various parameters  $q_0$  and  $q_s$ . It can be shown the optimal value  $n^*$  is an increasing function of  $q_0$  and a decreasing function of  $q_s$  (see Problem 23). Intuitively, these results state that when  $q_s$  increases it is desirable to reduce the number of components in the system as close to the value of threshold level  $k$  as possible. On the other hand, when  $q_0$  increases, the system reliability will be improved if the number of components increases.

**Theorem 2.6 (Ben-Dov 1980):** For fixed  $n$ ,  $q_0$ , and  $q_s$ , it is straightforward to see that the maximum value of  $R(k, n)$  is attained at  $k^* = \lfloor k_0 \rfloor + 1$ , where

$$k_0 = n \frac{\log \left( \frac{q_0}{p_s} \right)}{\log \left( \frac{q_s q_0}{p_s p_0} \right)} \quad (2.72)$$

If  $k_0$  is an integer, both  $k_0$  and  $k_0 + 1$  maximize  $R(k, n)$ .

*Proof:* The proof is left as an exercise for the reader (see Problem 24).

We now discuss how these two values,  $k^*$  and  $n^*$ , are related to one another. Define  $\alpha$  by

$$\alpha = \frac{\log \left( \frac{q_0}{p_s} \right)}{\log \left( \frac{q_s q_0}{p_s p_0} \right)} \quad (2.73)$$

then, for a given  $n$ , the optimal threshold  $k$  is given by  $k^* = \lceil n\alpha \rceil$  and for a given  $k$  the optimal  $n$  is  $n^* = \lfloor k / \alpha \rfloor$ . For any given  $q_0$  and  $q_s$ , we can easily show that (see Problem 25)

$$q_s < \alpha < p_0 \quad (2.74)$$

Therefore, we can obtain the following bounds for the optimal value of the threshold  $k$ :

$$nq_s < k^* < np_0 \quad (2.75)$$

This result shows that for given values of  $q_0$  and  $q_s$ , an upper bound for the optimal threshold  $k^*$  is the expected number of components working in open mode, and a lower bound for the optimal threshold  $k^*$  is the expected number of components failing in closed mode.

#### 2.4.2 An Application of Systems with Multiple Failure Modes

In many critical applications of digital systems, fault tolerance has been an essential architectural attribute for achieving high reliability. Several techniques can achieve fault tolerance using redundant hardware (Mathur and De Sousa 1975) or software (Pham 1985).

Typical forms of redundant hardware structures for fault-tolerant systems are of two types: fault masking and standby. Masking redundancy is achieved by implementing the functions so that they are inherently error correcting, *e.g.* triple-modular redundancy (TMR), N-modular redundancy (NMR), and self-purging redundancy. In standby redundancy, spare units are switched into the system when working units break down. Mathur and De Sousa (1975) have analyzed, in detail, hardware redundancy in the design of fault-tolerant digital systems. Redundant software structures for fault-tolerant systems based on the acceptance tests have been proposed by Homing *et al.* (1974).

This section presents a fault-tolerant architecture to increase the reliability of a special class of digital systems in communication (Pham and Upadhyaya 1989b). In this system, a monitor and a switch are associated with each redundant unit. The switches and monitors can fail. The monitors have two failure modes: failure to accept a correct result, and failure to reject an incorrect result. The scheme can be used in communication systems to improve their reliability.

Consider a digital circuit module designed to process the incoming messages in a communication system. This module consists of two units: a converter to process the messages, and a monitor to analyze the messages for their accuracy. For example, the converter could be decoding or unpacking circuitry, whereas the monitor could be checker circuitry (Lala 1985).

To guarantee a high reliability of operation at the receiver end,  $n$  converters are arranged in "parallel". All, except converter  $n$ , have a monitor to determine if the output of the converter is correct. If the output of a converter is not correct, the output is cancelled and a switch is changed so that the original input message is sent to the next converter. The architecture of such a system has been proposed by Pham and Upadhyaya (1989b). Systems of this kind have useful applications in communication and network control systems and in the analysis of fault-tolerant software systems.

We assume that a switch is never connected to the next converter without a signal from the monitor, and the probability that it is connected when a signal arrives is  $p_s$ . We next present a general expression for the reliability of the system

consisting of  $n$  non-identical converters arranged in "parallel". Let us define the following notation, events, and assumptions.

#### Notation

$p_i^c$	$\Pr\{\text{converter } i \text{ works}\}$
$p_i^s$	$\Pr\{\text{switch } i \text{ is connected to converter } (i + 1) \text{ when a signal arrives}\}$
$p_i^{m1}$	$\Pr\{\text{monitor } i \text{ works when converter } i \text{ works}\} = \Pr\{\text{not sending a signal to the switch when converter } i \text{ works}\}$
$p_i^{m2}$	$\Pr\{i \text{ monitor works when converter } i \text{ has failed}\} = \Pr\{\text{sending a signal to the switch when converter } i \text{ has failed}\}$
$R_{n-k}^k$	Reliability of the remaining system of size $(n-k)$ given that the first $k$ switches work
$R_n$	Reliability of the system consisting of $n$ converters

The events are:

$C_i^w, C_i^f$	Converter $i$ works, fails
$M_i^w, M_i^f$	Monitor $i$ works, fails
$S_i^w, S_i^f$	Switch $i$ works, fails
$W$	System works

The assumptions are:

1. The system, the switches, and the converters are two-state: good or failed.
2. The module (converter, monitor, or switch) states are mutually statistically independent.
3. The monitors have three states: good, failed in mode 1, failed in mode 2.
4. The modules are not identical.

The reliability of the system is defined as the probability of obtaining the correctly processed message at the output. To derive a general expression for the reliability of the system, we use an adapted form of the total probability theorem as translated into the language of reliability.

Let  $A$  denote the event that a system performs as desired. Let  $X_i$  and  $X_j$  be the event that a component  $X$  (e.g. converter, monitor, or switch) is good or failed respectively. Then

$$\Pr\{\text{system works}\} = \Pr\{\text{system works when unit } X \text{ is good}\} \times \Pr\{\text{unit } X \text{ is good}\} \\ + \Pr\{\text{system works when unit } X \text{ fails}\} \times \Pr\{\text{unit } X \text{ is failed}\}$$

The above equation provides a convenient way of calculating the reliability of complex systems. Notice that  $R_1 = p_1^c$  and for  $n \geq 2$ , the reliability of the system can be calculated as follows:

$$R_n = \Pr\{W | C_1^w \text{ and } M_1^w\} \Pr\{C_1^w \text{ and } M_1^w\} + \Pr\{W | C_1^w \text{ and } M_1^f\} \\ \Pr\{C_1^w \text{ and } M_1^f\} + \Pr\{W | C_1^f \text{ and } M_1^w\} \Pr\{C_1^f \text{ and } M_1^w\}$$

$$+ \Pr\{W | C_1^f \text{ and } M_1^f\} \Pr\{C_1^f \text{ and } M_1^f\}$$

In order for the system to operate when the first converter works and the first monitor fails, the first switch must work and the remaining system of size  $n-1$  must work:

$$\Pr\{W | C_1^w \text{ and } M_1^f\} = p_1^s R_{n-1}^1$$

Similarly,

$$\Pr\{W | C_1^f \text{ and } M_1^w\} = p_1^s R_{n-1}^1$$

then

$$R_n = p_1^c p_1^{m_1} + [p_1^c (1 - p_1^{m_1}) + (1 - p_1^c) p_1^{m_2}] p_1^s R_{n-1}^1$$

The reliability of the system consisting of  $n$  non-identical converters can be rewritten as:

$$R_n = \sum_{i=1}^{n-1} p_i^c p_i^{m_1} \pi_{i-1} + \pi_{n-1} p_n^c \quad \text{for } n > 1 \quad (2.76)$$

and  $R_1 = p_1^c$  where

$$\pi_k^j = \prod_{i=j}^k A_i \quad \text{for } k \geq 1$$

$$\pi_k \equiv \pi_k^1 \quad \text{for all } k \text{ and } \pi_0 = 1$$

and

$$A_i \equiv [p_i^c (1 - p_i^{m_1}) + (1 - p_i^c) p_i^{m_2}] \quad \text{for all } i = 1, 2, \dots, n$$

Assume that all the converters, monitors, and switches have the same reliability, that is

$$p_i^c = p^c, \quad p_i^{m_1} = p^{m_1}, \quad p_i^{m_2} = p^{m_2}, \quad p_i^s = p^s \quad \text{for all } i$$

then we obtain a closed form expression for the reliability of system as follows:

$$R_n = \frac{p^c p^{m_1}}{1 - A} (1 - A^{n-1}) + p^c A^{n-1} \quad (2.77)$$

where

$$A = [p^c (1 - p^{m_1}) + (1 - p^c) p^{m_2}] p^s$$

## 2.5 Markov Processes

Stochastic processes are used for the description of a systems operation over time. There are two main types of stochastic processes: continuous and discrete. The complex continuous process is a process describing a system transition from state to state. The simplest process that will be discussed here is a Markov process. Given the current state of the process, its future behavior does not depend on the past. In Section 2.6 we will discuss the discrete stochastic process. As an introduction to the Markov process, let us examine the following example.

*Example 2.15:* Consider a parallel system consisting of two components. From a reliability point of view, the states of the system can be described by

*State 1:* Full operation (both components operating)

*State 2:* One component operating - one component failed

*State 3:* Both components failed

Define

$$P_i(t) = P[X(t) = i] = P[\text{system is in state } i \text{ at time } t]$$

and

$$P_i(t + dt) = P[X(t + dt) = i] = P[\text{system is in state } i \text{ at time } t + dt].$$

Define a random variable  $X(t)$  which can assume the values 1, 2, or 3 corresponding to the above-mentioned states. Since  $X(t)$  is a random variable, one can discuss  $P[X(t) = 1]$ ,  $P[X(t) = 2]$  or conditional probability,  $P[X(t_1) = 2 \mid X(t_0) = 1]$ . Again,  $X(t)$  is defined as a function of time  $t$ , the last stated conditional probability,  $P[X(t_1) = 2 \mid X(t_0) = 1]$ , can be interpreted as the probability of being in state 2 at time  $t_1$ , given that the system was in state 1 at time  $t_0$ . In this example, the "stage space" is discrete, *i.e.*, 1, 2, 3, *etc.*, and the parameter space (time) is continuous. The simple process described above is called a stochastic process, *i.e.*, a process which develops in time (or space) in accordance with some probabilistic (stochastic) laws. There are many types of stochastic processes. In this section, the emphasis will be on Markov processes which are a special type of stochastic process.

**Definition 2.3:** Let  $t_0 < t_1 < \dots < t_n$ . If

$$\begin{aligned} P[X(t_n) = A_n \mid X(t_{n-1}) = A_{n-1}, X(t_{n-2}) = A_{n-2}, \dots, X(t_0) = A_0] \\ = P[X(t_n) = A_n \mid X(t_{n-1}) = A_{n-1}] \end{aligned}$$

then the process is called a Markov process.

Given the present state of the process, its future behavior does not depend on past information of the process.

The essential characteristic of a Markov process is that it is a process that has no memory; its future is determined by the present and not the past. If, in addition to having no memory, the process is such that it depends only on the difference  $(t+dt)-t = dt$  and not the value of  $t$ , *i.e.*,  $P[X(t+dt) = j \mid X(t) = i]$  is independent of  $t$ , then the process is Markov with stationary transition probabilities or homogeneous in time. This is the same property noted in exponential event times, and referring back to the graphical representation of  $X(t)$ , the times between state changes would in fact be exponential if the process has stationary transition probabilities.

Thus, a Markov process which is time homogeneous can be described as a process where events have exponential occurrence times. The random variable of the process is  $X(t)$ , the state variable rather than the time to failure as in the exponential failure density. To see the types of processes that can be described, a review of the exponential distribution and its properties will be made. Recall that, if  $X_1, X_2, \dots, X_n$ , are independent random variables, each with exponential density

and a mean equal to  $1/\lambda_i$  then  $\min \{ X_1, X_2, \dots, X_n \}$  has an exponential density with mean  $(\sum \lambda_i)^{-1}$ .

The significance of the property is as follows:

1. The failure behavior of the simultaneous operation of components can be characterized by an exponential density with a mean equal to the reciprocal of the sum of the failure rates.
2. The joint failure/repair behavior of a system where components are operating and/or undergoing repair can be characterized by an exponential density with a mean equal to the reciprocal of the sum of the failure and repair rates.
3. The failure/repair behavior of a system such as 2 above, but further complicated by active and dormant operating states and sensing and switching, can be characterized by an exponential density.

The above property means that almost all reliability and availability models can be characterized by a time homogeneous Markov process if the various failure times and repair times are exponential. The notation for the Markov process is  $\{X(t), t \geq 0\}$ , where  $X(t)$  is discrete (state space) and  $t$  is continuous (parameter space). By convention, this type of Markov process is called a continuous parameter Markov chain.

From a reliability/availability viewpoint, there are two types of Markov processes. These are defined as follows:

1. *Absorbing Process*: Contains what is called an "absorbing state" which is a state from which the system can never leave once it has entered, e.g., a failure which aborts a flight or a mission.
2. *Ergodic Process*: Contains no absorbing states such that  $X(t)$  can move around indefinitely, e.g., the operation of a ground power plant where failure only temporarily disrupts the operation.

Pham (2000a) page 265, presents a summary of the processes to be considered broken down by absorbing and ergodic categories. Both reliability and availability can be described in terms of the probability of the process or system being in defined "up" states, e.g., states 1 and 2 in the initial example. Likewise, the mean time between failures (MTBF) can be described as the total time in the "up" states before proceeding to the absorbing state or failure state.

Define the incremental transition probability as

$$P_{ij}(dt) = P[X(t+dt) = j \mid X(t) = i]$$

This is the probability that the process (random variable  $X(t)$ ) will go to state  $i$  during the increment  $t$  to  $(t+dt)$ , given that it was in state  $i$  at time  $t$ . Since we are dealing with time homogeneous Markov processes, i.e., exponential failure and repair times, the incremental transition probabilities can be derived from an analysis of the exponential hazard function. In Section 2.1, it was shown that the hazard function for the exponential with mean  $1/\lambda$  was just  $\lambda$ . This means that the limiting (as  $dt \rightarrow 0$ ) conditional probability of an event occurrence between  $t$  and  $t + dt$ , given that an event had not occurred at time  $t$ , is just  $\lambda$ , i.e.,



$$h(t) = \lim_{dt \rightarrow 0} \frac{P[t < X < t + dt \mid X > t]}{dt} = \lambda$$

The equivalent statement for the random variable  $X(t)$  is

$$h(t)dt = P[X(t + dt) = j \mid X(t) = i] = \lambda dt$$

Now,  $h(t)dt$  is in fact the incremental transition probability, thus the  $P_{ij}(dt)$  can be stated in terms of the basic failure and/or repair rates.

Returning to Example 2.15, a state transition can be easily constructed showing the incremental transition probabilities for process between all possible states (see Figure.B.4, Pham 2000a)

State 1: Both components operating

State 2: One component up - one component down

State 3: Both components down (absorbing state)

The loops indicate the probability of remaining in the present state during the  $dt$  increment

$$P_{11}(dt) = 1 - 2\lambda dt$$

$$P_{12}(dt) = 2\lambda dt$$

$$P_{13}(dt) = 0$$

$$P_{21}(dt) = 0$$

$$P_{22}(dt) = 1 - \lambda dt$$

$$P_{23}(dt) = \lambda dt$$

$$P_{31}(dt) = 0$$

$$P_{32}(dt) = 0$$

$$P_{33}(dt) = 1$$

The zeros on  $P_{ij}$ ,  $i > j$ , denote that the process cannot go backwards, *i.e.*, this is not a repair process. The zero on  $P_{13}$  denotes that in a process of this type, the probability of more than one event (*e.g.*, failure, repair, *etc.*) in the incremental time period  $dt$  approaches zero as  $dt$  approaches zero.

Except for the initial conditions of the process, *i.e.*, the state in which the process starts, the process is completely specified by the incremental transition probabilities. The reason for the latter is that the assumption of exponential event (failure or repair) times allows the process to be characterized at any time  $t$  since it depends only on what happens between  $t$  and  $(t + dt)$ . The incremental transition probabilities can be arranged into a matrix in a way which depicts all possible statewide movements. Thus, for the parallel configurations,

$$[p_{ij}(dt)] = \begin{bmatrix} 1 & 2 & 3 \\ 1 - 2\lambda dt & 2\lambda dt & 0 \\ 0 & 1 - \lambda dt & \lambda dt \\ 0 & 0 & 1 \end{bmatrix}$$

for  $i, j = 1, 2$ , or  $3$ . The matrix  $[P_{ij}(dt)]$  is called the incremental, one-step transition matrix. It is a stochastic matrix, *i.e.*, the rows sum to 1.0. As mentioned earlier, this matrix along with the initial conditions completely describes the process.

Now,  $[P_{ij}(dt)]$  gives the probabilities for either remaining or moving to all the various states during the interval  $t$  to  $t + dt$ , hence,

$$P_1(t + dt) = (1 - 2\lambda dt)P_1(t)$$

$$P_2(t + dt) = 2\lambda dt P_1(t)(1 - \lambda dt)P_2(t)$$

$$P_3(t + dt) = \lambda dt P_2(t) + P_3(t)$$

By algebraic manipulation, we have

$$\begin{aligned}\frac{[P_1(t + dt) - P_1(t)]}{dt} &= -2\lambda P_1(t) \\ \frac{[P_2(t + dt) - P_2(t)]}{dt} &= 2\lambda P_1(t) - \lambda P_2(t) \\ \frac{[P_3(t + dt) - P_3(t)]}{dt} &= \lambda P_2(t)\end{aligned}$$

Taking limits of both sides as  $dt \rightarrow 0$ , we obtain

$$\begin{aligned}P_1'(t) &= -2\lambda P_1(t) \\ P_2'(t) &= 2\lambda P_1(t) - \lambda P_2(t) \\ P_3'(t) &= \lambda P_2(t)\end{aligned}$$

The above system of linear first-order differential equations can be easily solved for  $P_1(t)$  and  $P_2(t)$ , and therefore, the reliability of the configuration can be obtained:

$$R(t) = \sum_{i=1}^2 P_i(t)$$

Actually, there is no need to solve all three equations, but only the first two as  $P_3(t)$  does not appear and also  $P_3(t) = 1 - P_1(t) - P_2(t)$ . The system of linear, first-order differential equations can be solved by various means including both manual and machine methods. For purposes here, the manual methods employing the Laplace transform (see Appendix 2) will be used.

$$L[P_i(t)] = \int_0^{\infty} e^{-st} P_i(t) dt = f_i(s)$$

$$L[P_i'(t)] = \int_0^{\infty} e^{-st} P_i'(t) dt = s f_i(s) - P_i(0)$$

The use of the Laplace transform will allow transformation of the system of linear, first-order differential equations into a system of linear algebraic equations which can easily be solved, and by means of the inverse transforms, solutions of  $P_i(t)$  can be determined.

Returning to the example, the initial condition of the parallel configuration is assumed to be “full-up” such that

$$P_1(t=0) = 1, \quad P_2(t=0) = 0, \quad P_3(t=0) = 0$$

transforming the equations for  $P_1'(t)$  and  $P_2'(t)$  gives

$$\begin{aligned}sf_1(s) - P_1(t)|_{t=0} &= -2\lambda f_1(s) \\ sf_2(s) - P_2(t)|_{t=0} &= 2\lambda f_1(s) - \lambda f_2(s)\end{aligned}$$

Evaluating  $P_1(t)$  and  $P_2(t)$  at  $t = 0$  gives

$$\begin{aligned}sf_1(s) - 1 &= -2\lambda f_1(s) \\ sf_2(s) - 0 &= 2\lambda f_1(s) - \lambda f_2(s)\end{aligned}$$

from which we obtain

$$\begin{aligned}(s + 2\lambda)f_1(s) &= 1 \\ -2\lambda f_1(s) + (s + \lambda)f_2(s) &= 0\end{aligned}$$

Solving the above equations for  $f_1(s)$  and  $f_2(s)$ , we have

$$f_1(s) = \frac{1}{(s + 2\lambda)}$$

$$f_2(s) = \frac{2\lambda}{[(s + 2\lambda)(s + \lambda)]}$$

From Appendix 2 of the inverse Laplace transforms,

$$P_1(t) = e^{-2\lambda t}$$

$$P_2(t) = 2e^{-\lambda t} - 2e^{-2\lambda t}$$

$$R(t) = P_1(t) + P_2(t) = 2e^{-\lambda t} - e^{-2\lambda t}$$

The example given above is that of a simple absorbing process where we are concerned about reliability. If repair capability in the form of a repair rate  $\mu$  were added to the model, the methodology would remain the same with only the final result changing. With a repair rate  $\mu$  added to the parallel configuration, the incremental transition matrix would be

$$[P_{ij}(dt)] = \begin{bmatrix} 1 - 2\lambda dt & 2\lambda dt & 0 \\ \mu dt & 1 - (\lambda + \mu)dt & \lambda dt \\ 0 & 0 & 1 \end{bmatrix}$$

The differential equations would become

$$P_1'(t) = -2\lambda P_1(t) + \mu P_2(t)$$

$$P_2'(t) = 2\lambda P_1(t) - (\lambda + \mu)P_2(t)$$

and the transformed equations would become

$$(s + 2\lambda)f_1(s) - \mu f_2(s) = 1$$

$$-2\lambda f_1(s) + (s + \lambda + \mu)f_2(s) = 0$$

Hence, we obtain

$$f_1(s) = \frac{(s + \lambda + \mu)}{(s - s_1)(s - s_2)}$$

$$f_2(s) = \frac{2\lambda}{(s - s_1)(s - s_2)}$$

where

$$s_1 = \frac{-(3\lambda + \mu) + \sqrt{(3\lambda + \mu)^2 - 8\lambda^2}}{2}$$

$$s_2 = \frac{-(3\lambda + \mu) - \sqrt{(3\lambda + \mu)^2 - 8\lambda^2}}{2}$$

From Appendix 2, we obtain

$$P_1(t) = \frac{(s_1 + \lambda + \mu)e^{-s_1 t}}{(s_1 - s_2)} + \frac{(s_2 + \lambda + \mu)e^{-s_2 t}}{(s_2 - s_1)}$$

$$P_2(t) = \frac{2\lambda e^{-s_1 t}}{(s_1 - s_2)} + \frac{2\lambda e^{-s_2 t}}{(s_2 - s_1)}$$

Thus, the reliability of two-component in a parallel system is given by

$$R(t) = P_1(t) + P_2(t)$$

$$= \frac{(s_1 + 3\lambda + \mu)e^{-s_1 t} - (s_2 + 3\lambda + \mu)e^{-s_2 t}}{(s_1 - s_2)} \quad (2.78)$$

### System Mean Time Between Failures

Another parameter of interest in absorbing Markov processes is the mean time between failures (MTBF). Recalling the previous example of a parallel configuration with repair, the differential equations  $P_1'(t)$  and  $P_2'(t)$  describing the process were

$$P_1'(t) = -2\lambda P_1(t) + \mu P_2(t)$$

$$P_2'(t) = 2\lambda P_1(t) - (\lambda + \mu)P_2(t)$$

Integrating both sides of the above equations yields

$$\int_0^{\infty} P_1'(t) dt = -2\lambda \int_0^{\infty} P_1(t) dt + \mu \int_0^{\infty} P_2(t) dt$$

$$\int_0^{\infty} P_2'(t) dt = 2\lambda \int_0^{\infty} P_1(t) dt - (\lambda + \mu) \int_0^{\infty} P_2(t) dt$$

From Section 2.1,

$$\int_0^{\infty} R(t) dt = MTBF$$

Similarly,

$$\int_0^{\infty} P_1(t) dt = \text{mean time spent in state 1, and}$$

$$\int_0^{\infty} P_2(t) dt = \text{mean time spent in state 2}$$

Designating these mean times as  $T_1$  and  $T_2$ , respectively, we have

$$P_1(t) dt \Big|_0^{\infty} = -2\lambda T_1 + \mu T_2$$

$$P_2(t) dt \Big|_0^{\infty} = 2\lambda T_1 - (\lambda + \mu) T_2$$

But  $P_1(t) = 0$  as  $t \rightarrow \infty$  and  $P_1(t) = 1$  for  $t = 0$ . Likewise,  $P_2(t) = 0$  as  $t \rightarrow \infty$  and  $P_2(t) = 0$  for  $t = 0$ . Thus,

$$\begin{aligned}-1 &= -2\lambda T_1 + \mu T_2 \\ 0 &= 2\lambda T_1 - (\lambda + \mu)T_2\end{aligned}$$

or, equivalently,

$$\begin{bmatrix} -1 \\ 0 \end{bmatrix} = \begin{bmatrix} -2\lambda & \mu \\ 2\lambda & -(\lambda + \mu) \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \end{bmatrix}$$

Therefore,

$$\begin{aligned}T_1 &= \frac{(\lambda + \mu)}{2\lambda^2} & T_2 &= \frac{1}{\lambda} \\ MTBF &= T_1 + T_2 = \frac{(\lambda + \mu)}{2\lambda^2} + \frac{1}{\lambda} = \frac{(3\lambda + \mu)}{2\lambda^2}\end{aligned}$$

The MTBF for non-maintenance processes is developed exactly the same way as just shown. What remains under absorbing processes is the case for availability for maintained systems. The difference between reliability and availability for absorbing processes is somewhat subtle. A good example is that of a communication system where, if such a system failed temporarily, the mission would continue, but, if it failed permanently, the mission would be aborted. Consider the following cold-standby configuration consisting of two units: one main unit and one spare unit (Pham 2000a):

- State 1:* Main unit operating - spare OK
- State 2:* Main unit out - restoration underway
- State 3:* Spare unit installed and operating
- State 4:* Permanent failure (no spare available)

The incremental transition matrix is given by (see Figure B.8 in Pham 2000a, for a detailed state transition diagram)

$$[P_{ij}(dt)] = \begin{bmatrix} 1 - \lambda dt & \lambda dt & 0 & 0 \\ 0 & 1 - \mu dt & \mu dt & 0 \\ 0 & 0 & 1 - \lambda dt & \lambda dt \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We obtain

$$\begin{aligned}P_1'(t) &= -\lambda P_1(t) \\ P_2'(t) &= \lambda P_1(t) - \mu P_2(t) \\ P_3'(t) &= \mu P_2(t) - \lambda P_3(t)\end{aligned}$$

Using the Laplace transform, we obtain

$$\begin{aligned}sf_1(s) - 1 &= -\lambda f_1(s) \\ sf_2(s) &= \lambda f_1(s) - \mu f_2(s) \\ sf_3(s) &= \mu f_2(s) - \lambda f_3(s)\end{aligned}$$

After simplifications,

$$f_1(s) = \frac{1}{(s + \lambda)}$$

$$f_2(s) = \frac{\lambda}{[(s + \lambda)(s + \mu)]}$$

$$f_3(s) = \frac{\lambda\mu}{[(s + \lambda)^2(s + \mu)]}$$

Therefore, the probability of full-up performance,  $P_1(t)$ , is given by

$$P_1(t) = e^{-\lambda t}$$

Similarly, the probability of the system being down and under repair,  $P_2(t)$ , is

$$P_2(t) = \left[ \frac{\lambda}{(\lambda - \mu)} \right] (e^{-\mu t} - e^{-\lambda t})$$

and the probability of the system being full-up but no spare available,  $P_3(t)$ , is

$$P_3(t) = \left[ \frac{\lambda\mu}{(\lambda - \mu)^2} \right] [e^{-\mu t} - e^{-\lambda t} - (\lambda - \mu)te^{-\lambda t}]$$

Hence, the point availability,  $A(t)$ , is given by

$$A(t) = P_1(t) + P_3(t)$$

If average or interval availability is required, this is achieved by

$$\left( \frac{1}{t} \right) \int_0^T A(t) dt = \left( \frac{1}{t} \right) \int_0^T [P_1(t) + P_3(t)] dt \quad (2.79)$$

where  $T$  is the interval of concern.

With the above example, cases of the absorbing process (both maintained and non-maintained) have been covered insofar as "manual" methods are concerned. In general, the methodology for treatment of absorbing Markov processes can be "packaged" in a fairly simplified form by utilizing matrix notation. Thus, for example, if the incremental transition matrix is defined as follows:

$$[P_{ij}(dt)] = \begin{bmatrix} 1 - 2\lambda dt & 2\lambda dt & 0 \\ \mu dt & 1 - (\lambda + \mu)dt & \lambda dt \\ 0 & 0 & 1 \end{bmatrix}$$

then if the  $dt$ s are dropped and the last row and the last column are deleted, the remainder is designated as the matrix  $T$ :

$$[T] = \begin{bmatrix} 1 - 2\lambda & 2\lambda \\ \mu & 1 - (\lambda + \mu) \end{bmatrix}$$

Define  $[Q] = [T]' - [I]$ , where  $[T]'$  is the transposition of  $[T]$  and  $[I]$  is the unity matrix:

$$[Q] = \begin{bmatrix} 1-2\lambda & \mu \\ 2\lambda & 1-(\lambda+\mu) \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ = \begin{bmatrix} -2\lambda & \mu \\ 2\lambda & -(\lambda+\mu) \end{bmatrix}$$

Further define  $[P(t)]$  and  $[P'(t)]$  as column vectors such that

$$[P_1(t)] = \begin{bmatrix} P_1(t) \\ P_2(t) \end{bmatrix}, \quad [P'(t)] = \begin{bmatrix} P'_1(t) \\ P'_2(t) \end{bmatrix}$$

then

$$[P'(t)] = [Q][P(t)]$$

At the above point, solution of the system of differential equations will produce solutions to  $P_1(t)$  and  $P_2(t)$ . If the MTBF is desired, integration of both sides of the system produces

$$\begin{bmatrix} -1 \\ 0 \end{bmatrix} = [Q] \begin{bmatrix} T_1 \\ T_2 \end{bmatrix} \\ \begin{bmatrix} -1 \\ 0 \end{bmatrix} = \begin{bmatrix} -2\lambda & \mu \\ 2\lambda & -(\lambda+\mu) \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \end{bmatrix} \quad \text{or} \\ [Q]^{-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} T_1 \\ T_2 \end{bmatrix}$$

where  $[Q]^{-1}$  is the inverse of  $[Q]$  and the MTBF is given by

$$\text{MTBF} = T_1 + T_2 = \frac{3\lambda + \mu}{(2\lambda)^2}$$

In the more general MTBF case,

$$[Q]^{-1} \begin{bmatrix} -1 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix} = \begin{bmatrix} T_1 \\ T_2 \\ \cdot \\ \cdot \\ \cdot \\ T_{n-1} \end{bmatrix} \quad \text{where} \quad \sum_{i=1}^{n-1} T_i = \text{MTBF}$$

and  $(n - 1)$  is the number of non-absorbing states.

For the reliability/availability case, utilizing the Laplace transform, the system of linear, first-order differential equations is transformed to

$$\begin{aligned}
s \begin{bmatrix} f_1(s) \\ f_2(s) \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \end{bmatrix} &= [Q] \begin{bmatrix} f_1(s) \\ f_2(s) \end{bmatrix} \\
[sI - Q] \begin{bmatrix} f_1(s) \\ f_2(s) \end{bmatrix} &= \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\
\begin{bmatrix} f_1(s) \\ f_2(s) \end{bmatrix} &= [sI - Q]^{-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\
L^{-1} \begin{bmatrix} f_1(s) \\ f_2(s) \end{bmatrix} &= L^{-1} \{ [sI - Q]^{-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \} \\
\begin{bmatrix} p_1(s) \\ p_2(s) \end{bmatrix} &= L^{-1} \{ [sI - Q]^{-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \}
\end{aligned}$$

Generalization of the latter to the case of  $(n-1)$  non-absorbing states is straightforward.

Ergodic processes, as opposed to absorbing processes, do not have any absorbing states, and hence, movement between states can go on indefinitely. For the latter reason, availability (point, steady-state, or interval) is the only meaningful measure. As an example for ergodic processes, a ground-based power unit configured in parallel will be selected.

The parallel units are identical, each with exponential failure and repair times with means  $1/\lambda$  and  $1/\mu$ , respectively (Pham 2000a). Assume a two-repairmen capability if required (both units down), then

*State 1:* Full-up (both units operating)

*State 2:* One unit down and under repair (other unit up)

*State 3:* Both units down and under repair

It should be noted that, as in the case of failure events, two or more repairs cannot be made in the  $dt$  interval.

$$[P_{ij}(dt)] = \begin{bmatrix} 1 - 2\lambda dt & 2\lambda dt & 0 \\ \mu dt & 1 - (\lambda + \mu)dt & \lambda dt \\ 0 & 2\mu dt & 1 - 2\mu dt \end{bmatrix}$$

*Case I: Point Availability - Ergodic Process.* For an ergodic process, as  $t \rightarrow \infty$  the availability settles down to a constant level. Point availability gives a measure of things before the "settling down" and reflects the initial conditions on the process. Solution of the point availability is similar to the case for absorbing processes except that the last row and column of the transition matrix must be retained and entered into the system of equations. For example, the system of differential equations becomes



$$\begin{bmatrix} P_1'(t) \\ P_2'(t) \\ P_3'(t) \end{bmatrix} = \begin{bmatrix} -2\lambda & \mu & 0 \\ 2\lambda & -(\lambda + \mu) & 2\mu \\ 0 & \lambda & -2\mu \end{bmatrix} \begin{bmatrix} P_1(t) \\ P_2(t) \\ P_3(t) \end{bmatrix}$$

Similar to the absorbing case, the method of the Laplace transform can be used to solve for  $P_1(t)$ ,  $P_2(t)$ , and  $P_3(t)$ , with the point availability,  $A(t)$ , given by

$$A(t) = P_1(t) + P_2(t)$$

*Case II: Interval Availability - Ergodic Process.* This is the same as the absorbing case with integration over time period  $T$  of interest. The interval availability,  $A(T)$ , is

$$A(T) = \frac{1}{T} \int_0^T A(t) dt \quad (2.80)$$

*Case III: Steady State Availability - Ergodic Process.* Here the process is examined as  $t \rightarrow \infty$  with complete “washout” of the initial conditions. Letting  $t \rightarrow \infty$  the system of differential equations can be transformed to linear algebraic equations. Thus,

$$\lim_{t \rightarrow \infty} \begin{bmatrix} P_1'(t) \\ P_2'(t) \\ P_3'(t) \end{bmatrix} = \lim_{t \rightarrow \infty} \begin{bmatrix} -2\lambda & \mu & 0 \\ 2\lambda & -(\lambda + \mu) & 2\mu \\ 0 & \lambda & -2\mu \end{bmatrix} \begin{bmatrix} P_1(t) \\ P_2(t) \\ P_3(t) \end{bmatrix}$$

As  $t \rightarrow \infty$ ,  $P_i(t) \rightarrow \text{constant}$  and  $P_i'(t) \rightarrow 0$ . This leads to an unsolvable system, namely

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -2\lambda & \mu & 0 \\ 2\lambda & -(\lambda + \mu) & 2\mu \\ 0 & \lambda & -2\mu \end{bmatrix} \begin{bmatrix} P_1(t) \\ P_2(t) \\ P_3(t) \end{bmatrix}$$

To avoid the above difficulty, an additional equation is introduced:

$$\sum_{i=1}^3 P_i(t) = 1$$

With the introduction of the new equation, one of the original equations is deleted and a new system is formed:

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ -2\lambda & \mu & 0 \\ 2\lambda & -(\lambda + \mu) & 2\mu \end{bmatrix} \begin{bmatrix} P_1(t) \\ P_2(t) \\ P_3(t) \end{bmatrix}$$

or, equivalently,

$$\begin{bmatrix} P_1(t) \\ P_2(t) \\ P_3(t) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ -2\lambda & \mu & 0 \\ 2\lambda & -(\lambda + \mu) & 2\mu \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

We now obtain the following results:

$$P_1(t) = \frac{\mu^2}{(\mu + \lambda)^2}$$

$$P_2(t) = \frac{2\lambda\mu}{(\mu + \lambda)^2}$$

and

$$\begin{aligned} P_3(t) &= 1 - P_1(t) - P_2(t) \\ &= \frac{\lambda^2}{(\mu + \lambda)^2} \end{aligned}$$

Therefore, the steady state availability,  $A(\infty)$ , is given by

$$\begin{aligned} A_3(\infty) &= P_1(t) + P_2(t) \\ &= \frac{\mu(\mu + 2\lambda)}{(\mu + \lambda)^2} \end{aligned}$$

Note that Markov methods can also be employed where failure or repair times are not exponential, but can be represented as the sum of exponential times with identical means (Erlang distribution or Gamma distribution with integer valued shape parameters). Basically, the method involves the introduction of "dummy" states which are of no particular interest in themselves, but serve the purpose of changing the hazard function from constant to increasing.

## 2.6 Counting Processes

Among discrete stochastic processes, counting processes in reliability engineering are widely used to describe the appearance of events in time, *e.g.*, failures, number of perfect repairs, *etc.* The simplest counting process is a Poisson process. The Poisson process plays a special role to many applications in reliability (Pham 2000a). A classic example of such an application is the decay of uranium. Radioactive particles from nuclear material strike a certain target in accordance with a Poisson process of some fixed intensity. A well-known counting process is the so-called renewal process. This process is described as a sequence of events, the intervals between which are independent and identically distributed random variables. In reliability theory, this type of mathematical model is used to describe the number of occurrences of an event in the time interval. In this section we also discuss the quasi-renewal process and the non-homogeneous Poisson process.

A non-negative, integer-valued stochastic process,  $N(t)$ , is called a counting process if  $N(t)$  represents the total number of occurrences of the event in the time interval  $[0, t]$  and satisfies these two properties:

1. If  $t_1 < t_2$ , then  $N(t_1) \leq N(t_2)$
2. If  $t_1 < t_2$ , then  $N(t_2) - N(t_1)$  is the number of occurrences of the event in the interval  $[t_1, t_2]$

For example, if  $N(t)$  equals the number of persons who have entered a restaurant at or prior to time  $t$ , then  $N(t)$  is a counting process in which an event occurs whenever a person enters the restaurant.

### 2.6.1 Poisson Processes

One of the most important counting processes is the Poisson process.

**Definition 2.4:** A counting process,  $N(t)$ , is said to be a Poisson process with intensity  $\lambda$  if

1. The failure process,  $N(t)$ , has stationary independent increments
2. The number of failures in any time interval of length  $s$  has a Poisson distribution with mean  $\lambda s$ , that is,

$$P\{N(t+s) - N(t) = n\} = \frac{e^{-\lambda s} (\lambda s)^n}{n!} \quad n = 0, 1, 2, \dots \quad (2.81)$$

3. The initial condition is  $N(0) = 0$

This model is also called a homogeneous Poisson process indicating that the failure rate  $\lambda$  does not depend on time  $t$ . In other words, the number of failures occurring during the time interval  $(t, t+s)$  does not depend on the current time  $t$  but only the length of time interval  $s$ . A counting process is said to possess independent increments if the number of events in disjoint time intervals are independent.

For a stochastic process with independent increments, the auto-covariance function is

$$\text{Cov}[X(t_1), X(t_2)] = \begin{cases} \text{Var}[N(t_1+s) - N(t_2)] & \text{for } 0 < t_2 - t_1 < s \\ 0 & \text{otherwise} \end{cases}$$

where

$$X(t) = N(t+s) - N(t).$$

If  $X(t)$  is Poisson distributed, then the variance of the Poisson distribution is

$$\text{Cov}[X(t_1), X(t_2)] = \begin{cases} \lambda[s - (t_2 - t_1)] & \text{for } 0 < t_2 - t_1 < s \\ 0 & \text{otherwise} \end{cases}$$

This result shows that the Poisson increment process is covariance stationary. We now present several properties of the Poisson process.

**Property 2.3:** The sum of independent Poisson processes,  $N_1(t)$ ,  $N_2(t)$ , ...,  $N_k(t)$ , with mean values  $\lambda_1 t$ ,  $\lambda_2 t$ , ...,  $\lambda_k t$  respectively, is also a Poisson process with mean  $\left( \sum_{i=1}^k \lambda_i \right) t$ . In other words, the sum of the independent Poisson processes is also a Poisson process with a mean that is equal to the sum of the individual Poisson process' mean.

*Proof:* The proof is left as an exercise for the reader (see Problem 26).

**Property 2.4:** The difference of two independent Poisson processes,  $N_1(t)$ , and  $N_2(t)$ , with mean  $\lambda_1 t$  and  $\lambda_2 t$ , respectively, is not a Poisson process. Instead, it has the probability mass function

$$P[N_1(t) - N_2(t) = k] = e^{-(\lambda_1 + \lambda_2)t} \left( \frac{\lambda_1}{\lambda_2} \right)^{\frac{k}{2}} I_k(2\sqrt{\lambda_1 \lambda_2} t) \quad (2.82)$$

where  $I_k(\cdot)$  is a modified Bessel function of order  $k$  (Handbook 1980).

*Proof:* The proof is left as an exercise for the reader (see Problem 27).

**Property 2.5:** If the Poisson process,  $N(t)$ , with mean  $\lambda t$ , is filtered such that every occurrence of the event is not completely counted, then the process has a constant probability  $p$  of being counted. The result of this process is a Poisson process with mean  $\lambda p t$  [ ].

**Property 2.6:** Let  $N(t)$  be a Poisson process and  $Y_n$  a family of independent and identically distributed random variables which are also independent of  $N(t)$ . A stochastic process  $X(t)$  is said to be a compound Poisson process if it can be represented as

$$X(t) = \sum_{i=1}^{N(t)} Y_i$$

## 2.6.2 Renewal Processes

A renewal process is a more general case of the Poisson process in which the inter-arrival times of the process or the time between failures do not necessarily follow the exponential distribution. For convenience, we will call the occurrence of an event a renewal, the inter-arrival time the renewal period, and the waiting time the renewal time.

**Definition 2.5:** A counting process  $N(t)$  that represents the total number of occurrences of an event in the time interval  $(0, t]$  is called a renewal process, if the time between failures are independent and identically distributed random variables.

The probability that there are exactly  $n$  failures occurring by time  $t$  can be written as

$$P\{N(t) = n\} = P\{N(t) \geq n\} - P\{N(t) > n\} \quad (2.83)$$

Note that the times between the failures are  $T_1, T_2, \dots, T_n$  so the failures occurring at time  $W_k$  are

$$W_k = \sum_{i=1}^k T_i$$

and

$$T_k = W_k - W_{k-1}$$

Thus,

$$\begin{aligned}
 P\{N(t) = n\} &= P\{N(t) \geq n\} - P\{N(t) > n\} \\
 &= P\{W_n \leq t\} - P\{W_{n+1} \leq t\} \\
 &= F_n(t) - F_{n+1}(t)
 \end{aligned}$$

where  $F_n(t)$  is the cumulative distribution function for the time of the  $n$ th failure and  $n = 0, 1, 2, \dots$ .

*Example 2.16:* Consider a software testing model for which the time to find an error during the testing phase has an exponential distribution with a failure rate of  $X$ . It can be shown that the time of the  $n$ th failure follows the gamma distribution with parameters  $k$  and  $n$  with probability density function. From equation (2.83) we obtain

$$\begin{aligned}
 P\{N(t) = n\} &= P\{N(t) \leq n\} - P\{N(t) \leq n-1\} \\
 &= \sum_{k=0}^n \frac{(\lambda t)^k}{k!} e^{-\lambda t} - \sum_{k=0}^{n-1} \frac{(\lambda t)^k}{k!} e^{-\lambda t} \\
 &= \frac{(\lambda t)^n}{n!} e^{-\lambda t} \quad \text{for } n = 0, 1, 2, \dots
 \end{aligned}$$

Several important properties of the renewal function are given below.

**Property 2.7:** The mean value function of the renewal process, denoted by  $m(t)$ , is equal to the sum of the distribution function of all renewal times, that is,

$$\begin{aligned}
 m(t) &= E[N(t)] \\
 &= \sum_{n=1}^{\infty} F_n(t)
 \end{aligned}$$

*Proof:* The renewal function can be obtained as

$$\begin{aligned}
 m(t) &= E[N(t)] \\
 &= \sum_{n=1}^{\infty} n P\{N(t) = n\} \\
 &= \sum_{n=1}^{\infty} n [F_n(t) - F_{n+1}(t)] \\
 &= \sum_{n=1}^{\infty} F_n(t)
 \end{aligned}$$

The mean value function of the renewal process is also called the renewal function.

**Property 2.8:** The renewal function,  $m(t)$ , satisfies the following equation:

$$m(t) = F_a(t) + \int_0^t m(t-s) dF_a(s) \quad (2.84)$$

where  $F_a(t)$  is the distribution function of the inter-arrival time or the renewal period. The proof is left as an exercise for the reader (see Problem 28).

In general, let  $y(t)$  be an unknown function to be evaluated and  $x(t)$  be any non-negative and integrable function associated with the renewal process. Assume that  $F_a(t)$  is the distribution function of the renewal period. We can then obtain the following result.

**Property 2.9:** Let the renewal equation be

$$y(t) = x(t) + \int_0^t y(t-s) dF_a(s) \quad (2.85)$$

then its solution is given by

$$y(t) = x(t) + \int_0^t x(t-s) dm(s)$$

where  $m(t)$  is the mean value function of the renewal process.

The proof of the above property can be easily derived using the Laplace transform. It is also noted that the integral equation given in Property 2.8 is a special case of Property 2.9.

*Example 2.17:* Let  $x(t) = a$ . Thus, in Property 2.9, the solution  $y(t)$  is given by

$$\begin{aligned} y(t) &= x(t) + \int_0^t x(t-s) dm(s) \\ &= a + \int_0^t a dm(s) \\ &= a(1 + E[N(t)]) \end{aligned}$$

### 2.6.3 Quasi-renewal Processes

In this section, a general renewal process, namely, the quasi-renewal process, is discussed. Let  $\{N(t), t > 0\}$  be a counting process and let  $X_n$  be the time between the  $(n-1)^{\text{th}}$  and the  $n^{\text{th}}$  event of this process,  $n \geq 1$ .

**Definition 2.6 (Wang and Pham 1996a):** If the sequence of non-negative random variables  $\{X_1, X_2, \dots\}$  is independent and

$$X_i = \alpha X_{i-1} \quad (2.86)$$

for  $i \geq 2$  where  $\alpha > 0$  is a constant, then the counting process  $\{N(t), t \geq 0\}$  is said to be a quasi-renewal process with parameter  $\alpha$  and the first inter-arrival time  $X_1$ .

When  $\alpha = 1$ , this process becomes the ordinary renewal process as discussed in Section 2.6.2. This quasi-renewal process can be used to model reliability growth processes in software testing phases and hardware burn-in stages for  $\alpha > 1$ , and in hardware maintenance processes when  $\alpha \leq 1$ .

Assume that the probability density function, cumulative distribution function, survival function, and failure rate of random variable  $X_1$  are  $f_1(x)$ ,  $F_1(x)$ ,  $s_1(x)$ , and  $r_1(x)$ , respectively. Then the pdf, cdf, survival function, failure rate of  $X_n$  for  $n = 1, 2, 3, \dots$  is respectively given below (Wang and Pham 1996a):

$$\begin{aligned} f_n(x) &= \frac{1}{\alpha^{n-1}} f_1\left(\frac{1}{\alpha^{n-1}} x\right) \\ F_n(x) &= F_1\left(\frac{1}{\alpha^{n-1}} x\right) \\ s_n(x) &= s_1\left(\frac{1}{\alpha^{n-1}} x\right) \\ r_n(x) &= \frac{1}{\alpha^{n-1}} r_1\left(\frac{1}{\alpha^{n-1}} x\right) \end{aligned}$$

Similarly, the mean and variance of  $X_n$  is given as

$$\begin{aligned} E(X_n) &= \alpha^{n-1} E(X_1) \\ \text{Var}(X_n) &= \alpha^{2n-2} \text{Var}(X_1) \end{aligned}$$

Because of the non-negativity of  $X_1$  and the fact that  $X_1$  is not identically 0, we obtain

$$E(X_1) = \mu_1 \neq 0$$

**Proposition 2.1 (Wang and Pham 1996a):** The shape parameters of  $X_n$  are the same for  $n = 1, 2, 3, \dots$  for a quasi-renewal process if  $X_1$  follows the gamma, Weibull, or log normal distribution.

This means that after "renewal", the shape parameters of the inter-arrival time will not change. In software reliability, the assumption that the software debugging process does not change the error-free distribution type seems reasonable. Thus, the error-free times of software during the debugging phase modeled by a quasi-renewal process will have the same shape parameters. In this sense, a quasi-renewal process is suitable to model the software reliability growth. It is worthwhile to note that

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{E(X_1 + X_2 + \dots + X_n)}{n} &= \lim_{n \rightarrow \infty} \frac{\mu_1(1 - \alpha^n)}{(1 - \alpha)n} \\ &= 0 \quad \text{if } \alpha < 1 \\ &= \infty \quad \text{if } \alpha > 1 \end{aligned}$$

Therefore, if the inter-arrival time represents the error-free time of a software system, then the average error-free time approaches infinity when its debugging process is occurring for a long debugging time.

### *Distribution of $N(t)$*

Consider a quasi-renewal process with parameter  $\alpha$  and the first inter-arrival time  $X_1$ . Clearly, the total number of renewals,  $N(t)$ , that has occurred up to time  $t$  and the arrival time of the  $n$ th renewal,  $SS_n$ , has the following relationship:

$$N(t) \geq n \text{ if and only if } SS_n \leq t$$

that is,  $N(t)$  is at least  $n$  if and only if the  $n$ th renewal occurs prior to time  $t$ . It is easily seen that

$$SS_n = \sum_{i=1}^n X_i = \sum_{i=1}^n \alpha^{i-1} X_1 \quad \text{for } n \geq 1 \quad (2.87)$$

Here,  $SS_0 = 0$ . Thus, we have

$$\begin{aligned} P\{N(t) = n\} &= P\{N(t) = n\} - P\{N(t) \geq n+1\} \\ &= P\{SS_n \leq t\} - P\{SS_{n+1} \leq t\} \\ &= G_n(t) - G_{n+1}(t) \end{aligned}$$

where  $G_n(t)$  is the convolution of the inter-arrival times  $F_1, F_2, F_3, \dots, F_n$ . In other words,

$$G_n(t) = P\{F_1 + F_2 + \dots + F_n \leq t\}$$

If the mean value of  $N(t)$  is defined as the renewal function  $m(t)$ , then,

$$\begin{aligned} m(t) &= E[N(t)] \\ &= \sum_{n=1}^{\infty} P\{N(t) \geq n\} \\ &= \sum_{n=1}^{\infty} P\{SS_n \leq t\} \\ &= \sum_{n=1}^{\infty} G_n(t) \end{aligned}$$

The derivative of  $m(t)$  is known as the renewal density

$$\lambda(t) = m'(t)$$

In renewal theory, random variables representing the inter-arrival distributions only assume non-negative values, and the Laplace transform of its distribution  $F_1(t)$  is defined by

$$\mathcal{L}\{F_1(s)\} = \int_0^{\infty} e^{-sx} dF_1(x)$$

Therefore,

$$\mathcal{L}F_n(s) = \int_0^{\infty} e^{-\alpha^{n-1}st} dF_1(t) = \mathcal{L}F_1(\alpha^{n-1}s)$$



and

$$\begin{aligned}\mathcal{L}m_n(s) &= \sum_{n=1}^{\infty} \mathcal{L}G_n(s) \\ &= \sum_{n=1}^{\infty} \mathcal{L}F_1(s) \mathcal{L}F_1(\alpha s) \cdots \mathcal{L}F_1(\alpha^{n-1}s)\end{aligned}$$

Since there is a one-to-one correspondence between distribution functions and its Laplace transform, it follows that

**Proposition 2.2 (Wang and Pham 1996a):** The first inter-arrival distribution of a quasi-renewal process uniquely determines its renewal function.

If the inter-arrival time represents the error-free time (time to first failure), a quasi-renewal process can be used to model reliability growth for both software and hardware.

Suppose that all faults of software have the same chance of being detected. If the inter-arrival time of a quasi-renewal process represents the error-free time of a software system, then the expected number of software faults in the time interval  $[0, t]$  can be defined by the renewal function,  $m(t)$ , with parameter  $\alpha > 1$ . Denoted by  $m_r(t)$ , the number of remaining software faults at time  $t$ , it follows that

$$m_r(t) = m(T_c) - m(t)$$

where  $m(T_c)$  is the number of faults that will eventually be detected through a software lifecycle  $T_c$ .

## 2.6.4 Non-homogeneous Poisson Processes

The non-homogeneous Poisson process model (NHPP) that represents the number of failures experienced up to time  $t$  is a non-homogeneous Poisson process  $\{N(t), t \geq 0\}$ . The main issue in the NHPP model is to determine an appropriate mean value function to denote the expected number of failures experienced up to a certain time.

With different assumptions, the model will end up with different functional forms of the mean value function. Note that in a renewal process, the exponential assumption for the inter-arrival time between failures is relaxed, and in the NHPP, the stationary assumption is relaxed.

The NHPP model is based on the following assumptions:

- The failure process has an independent increment, *i.e.*, the number of failures during the time interval  $(t, t + s)$  depends on the current time  $t$  and the length of time interval  $s$ , and does not depend on the past history of the process.
- The failure rate of the process is given by

$$\begin{aligned}P\{\text{exactly one failure in } (t, t + \Delta t)\} &= P\{N(t + \Delta t) - N(t) = 1\} \\ &= \lambda(t)\Delta t + o(\Delta t)\end{aligned}$$

where  $\lambda(t)$  is the intensity function.

- During a small interval  $\Delta t$ , the probability of more than one failure is negligible, that is,

$$P\{\text{two or more failure in } (t, t + \Delta t)\} = o(\Delta t)$$

- The initial condition is  $N(0) = 0$ .

On the basis of these assumptions, the probability of exactly  $n$  failures occurring during the time interval  $(0, t)$  for the NHPP is given by

$$\Pr\{N(t) = n\} = \frac{[m(t)]^n}{n!} e^{-m(t)} \quad n = 0, 1, 2, \dots \quad (2.88)$$

where  $m(t) = E[N(t)] = \int_0^t \lambda(s) ds$  and  $\lambda(t)$  is the intensity function. It can be easily shown that the mean value function  $m(t)$  is non-decreasing.

### Reliability Function

The reliability  $R(t)$ , defined as the probability that there are no failures in the time interval  $(0, t)$ , is given by

$$\begin{aligned} R(t) &= P\{N(t) = 0\} \\ &= e^{-m(t)} \end{aligned}$$

In general, the reliability  $R(x|t)$ , the probability that there are no failures in the interval  $(t, t + x)$ , is given by

$$\begin{aligned} R(x|t) &= P\{N(t+x) - N(t) = 0\} \\ &= e^{-[m(t+x) - m(t)]} \end{aligned}$$

and its density is given by

$$f(x) = \lambda(t+x) e^{-[m(t+x) - m(t)]}$$

where

$$\lambda(x) = \frac{\partial}{\partial x} [m(x)]$$

The variance of the NHPP can be obtained as follows:

$$\text{Var}[N(t)] = \int_0^t \lambda(s) ds$$

and the auto-correlation function is given by

$$\begin{aligned} \text{Cor}[s] &= E[N(t)]E[N(t+s) - N(t)] + E[N^2(t)] \\ &= \int_0^t \lambda(s) ds \int_0^{t+s} \lambda(s) ds + \int_0^t \lambda(s) ds \\ &= \int_0^t \lambda(s) ds \left[ 1 + \int_0^{t+s} \lambda(s) ds \right] \end{aligned}$$

*Example 2.18:* Assume that the intensity  $\lambda$  is a random variable with the pdf  $f(\lambda)$ . Then the probability of exactly  $n$  failures occurring during the time interval  $(0, t)$  is given by

$$P\{N(t) = n\} = \int_0^{\infty} e^{-\lambda t} \frac{(\lambda t)^n}{n!} f(\lambda) d\lambda$$

It can be shown that if the pdf  $f(\lambda)$  is given as the following gamma density function with parameters  $k$  and  $m$ ,

$$f(\lambda) = \frac{1}{\Gamma(m)} k^m \lambda^{m-1} e^{-k\lambda} \quad \text{for } \lambda \geq 0$$

then

$$P\{N(t) = n\} = \binom{n+m-1}{n} p^m q^n \quad n = 0, 1, 2, \dots$$

is also called a negative binomial density function, where

$$p = \frac{k}{t+k} \quad \text{and} \quad q = \frac{t}{t+k} = 1 - p$$

## 2.7 Further Reading

The reader interested in a deeper understanding of advanced probability theory and stochastic processes should note the following highly recommended books:

Devore, J.L., *Probability and Statistics for Engineering and the Sciences*, 3rd edition, Brooks/Cole Pub. Co., Pacific Grove, 1991.

Gnedenko, BV and I.A. Ushakov, *Probabilistic Reliability Engineering*, Wiley, New York, 1995.

Feller, W., *An Introduction to Probability Theory and Its Applications*, 3rd edition, Wiley, New York, 1994.

## 2.8 Problems

1. Assume that the hazard rate,  $h(t)$ , has a positive derivative. Show that the hazard distribution

$$H(t) = \int_0^t h(x) dx$$

is strictly convex.

2. An operating unit is supported by  $(n-1)$  identical units on cold standby. When it fails, a unit from standby takes its place. The system fails if all  $n$  units fail.

Assume that units on standby cannot fail and the lifetime of each unit follows the exponential distribution with failure rate  $\lambda$ .

- (a) What is the distribution of the system lifetime?
  - (b) Determine the reliability of the standby system for a mission of 100 hours when  $\lambda = 0.0001$  per hour and  $n = 5$ .
3. Assume that there is some latent deterioration process occurring in the system. During the interval  $[0, a-h]$  the deterioration is comparatively small so that the shocks do not cause system failure. During a relatively short time interval  $[a-h, a]$ , the deterioration progresses rapidly and makes the system susceptible to shocks. Assume that the appearance of each shock follows the exponential distribution with failure rate  $\lambda$ . What is the distribution of the system lifetime?

4. Consider a series system of  $n$  Weibull components. The corresponding lifetimes  $T_1, T_2, \dots, T_n$  are assumed to be independent with pdf

$$f(t) = \begin{cases} \lambda_i^\beta \beta t^{\beta-1} e^{-(\lambda_i t)^\beta} & \text{for } t \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

where  $\lambda > 0$  and  $\beta > 0$  are the scale and shape parameters, respectively.

- (a) Show that the lifetime of a series system has the Weibull distribution with pdf

$$f_s(t) = \begin{cases} \left( \sum_{i=1}^n \lambda_i^\beta \right) \beta t^{\beta-1} e^{-\left( \sum_{i=1}^n \lambda_i^\beta \right) t^\beta} & \text{for } t \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

- (b) Find the reliability of this series system.

5. Consider the pdf of a random variable that is equally likely to take on any value *only* in the interval from  $a$  to  $b$ .

- (a) Show that this pdf is given by

$$f(t) = \begin{cases} \frac{1}{b-a} & \text{for } a < t < b \\ 0 & \text{otherwise} \end{cases}$$

- (b) Derive the corresponding reliability function  $R(t)$  and failure rate  $h(t)$ .  
(c) Think of an example where such a distribution function would be of interest in reliability application.

6. The failure rate function, denoted by  $h(t)$ , is defined as

$$h(t) = -\frac{d}{dt} \ln[R(t)]$$

Show that the constant failure rate function implies an exponential distribution.

7. One thousand new streetlights are installed in Saigon city. Assume that the lifetimes of these streetlights follow the normal distribution. The average life of these lamps is estimated at 980 burning-hours with a standard deviation of 100 hours.
  - (a) What is the expected number of lights that will fail during the first 800 burning-hours?
  - (b) What is the expected number of lights that will fail between 900 and 1100 burning-hours?
  - (c) After how many burning-hours would 10% of the lamps be expected to fail?
8. A fax machine with constant failure rate  $\lambda$  will survive for a period of 720 hours without failure, with probability 0.80.
  - (a) Determine the failure rate  $\lambda$ .
  - (b) Determine the probability that the machine, which is functioning after 600 hours, will still function after 800 hours.
  - (c) Find the probability that the machine will fail within 900 hours, given that the machine was functioning at 720 hours.
9. The time to failure  $T$  of a unit is assumed to have a log normal distribution with pdf

$$f(t) = \frac{1}{\sqrt{2\pi}} \frac{1}{\sigma t} e^{-\frac{(\ln t - \mu)^2}{2\sigma^2}} \quad t > 0$$

Show that the failure rate function is unimodal.

10. A diode may fail due to either open or short failure modes. Assume that the time to failure  $T_o$  caused by open mode is exponentially distributed with pdf

$$f_o(t) = \lambda_o e^{-\lambda_o t} \quad t \geq 0$$

and the time to failure  $T_s$  caused by short mode has the pdf

$$f_s(t) = \lambda_s e^{-\lambda_s t} \quad t \geq 0$$

The pdf for the time to failure  $T$  of the diode is given by

$$f(t) = p f_o(t) + (1 - p) f_s(t) \quad t \geq 0$$

- (a) Explain the meaning of  $p$  in the above pdf function.
  - (b) Derive the reliability function  $R(t)$  and failure rate function  $h(t)$  for the time to failure  $T$  of the diode.
  - (c) Show that the diode with pdf  $f(t)$  has a decreasing failure rate (DFR).
11. A diesel is known to have an operating life (in hours) that fits the following pdf:

$$f(t) = \frac{2a}{(t+b)^2} \quad t \geq 0$$

The average operating life of the diesel has been estimated to be 8760 hours.

- (a) Determine  $a$  and  $b$ .
- (b) Determine the probability that the diesel will not fail during the first 6000 operating-hours.
- (c) If the manufacturer wants no more than 10% of the diesels returned for warranty service, how long should the warranty be?

**12.** The failure rate for a hydraulic component

$$h(t) = \frac{t}{t+1} \quad t > 0$$

where  $t$  is in years.

- (a) Determine the reliability function  $R(t)$ .
  - (b) Determine the MTTF of the component.
- 13.** A 18-month guarantee is given based on the assumption that no more than 5% of new cars will be returned.
- (a) The time to failure  $T$  of a car has a constant failure rate. What is the maximum failure rate that can be tolerated?
  - (b) Determine the probability that a new car will fail within three years assuming that the car was functioning at 18 months.

**14.** Show that if

$$R_1(t) \geq R_2(t) \quad \text{for all } t$$

where  $R_i(t)$  is the system reliability of the structure  $i$ , then MTTF of the system structure 1 is always  $\geq$  MTTF of the system structure 2.

**15.** Prove equation (2.10)

**16.** Show that the reliability function of Pham distribution (see equation 2.21) is given as in equation (2.22).

**17.** Prove Theorem 2.1.

**18.** Prove Theorem 2.2.

**19.** Show that for any range of  $q_0$  and  $q_s$ , if  $q_s > q_0$ , the optimal number of parallel components that maximizes the system reliability is one.

**20.** Prove Theorem 2.3.

**21.** Prove Theorem 2.4.

**22.** Prove Theorem 2.5.

- 23.** Show that the optimal value  $n^*$  in Theorem 2.5 is an increasing function of  $q_0$  and a decreasing function of  $q_s$ .
- 24.** Prove Theorem 2.6.
- 25.** For any given  $q_0$  and  $q_s$ , show that  $q_s < \alpha < p_0$  where  $\alpha$  is given in equation (2.73).
- 26.** Prove Property 2.3.
- 27.** Prove Property 2.4.
- 28.** Prove Property 2.8.
- 29.** Events occur according to an NHPP in which the mean value function is
- $$m(t) = t^3 + 3t^2 + 6t \quad t > 0.$$

What is the probability that  $n$  events occur between times  $t = 10$  and  $t = 15$ ?

## Theory of Estimation

---

### 3.1 Point Estimation

The problem of point estimation is that of estimating the parameters of a population, *e.g.*,  $\lambda$  or  $\theta$  from an exponential,  $\mu$  and  $\sigma^2$  from a normal, *etc.* It is assumed that the population distribution by type is known, but the distribution parameters are unknown and they have to be estimated by using collected failure data. This chapter is devoted to the theory of estimation and discusses several common estimation techniques such as maximum likelihood, least squared, and Bayesian methods. We also discuss the confidence interval estimates and tolerance limit estimates. For example, assume  $n$  independent samples from the exponential density  $f(x; \lambda) = \lambda e^{-\lambda x}$  for  $x > 0$  and  $\lambda > 0$ , then the joint probability density function (pdf) or sample density (for short) is given by

$$f(x_1, \lambda) \cdot f(x_2, \lambda) \cdots f(x_n, \lambda) = \lambda^n e^{-\lambda \sum_{i=1}^n x_i} \quad (3.1)$$

The problem here is to find a "good" point estimate of  $\lambda$  which is denoted by  $\hat{\lambda}$ . In other words, we shall find a function  $h(X_1, X_2, \dots, X_n)$  such that, if  $x_1, x_2, \dots, x_n$  are the observed experimental values of  $X_1, X_2, \dots, X_n$ , then the value  $h(x_1, x_2, \dots, x_n)$  will be a good point estimate of  $\lambda$ . By "good" we mean the following properties shall be implied:

- Unbiasedness
- Consistency
- Efficiency (*i.e.*, minimum variance)
- Sufficiency

In other words, if  $\hat{\lambda}$  is a good point estimate of  $\lambda$ , then one can select the function  $h(X_1, X_2, \dots, X_n)$  such that  $h(X_1, X_2, \dots, X_n)$  is not only an unbiased estimator of  $\lambda$  but



also the variance of  $h(X_1, X_2, \dots, X_n)$  is a minimum. We will now present the following definitions.

**Definition 3.1:** For a given positive integer  $n$ , the statistic  $Y = h(X_1, X_2, \dots, X_n)$  is called an unbiased estimator of the parameter  $\theta$  if the expectation of  $Y$  is equal to a parameter  $\theta$ , that is,

$$E(Y) = \theta \quad (3.2)$$

**Definition 3.2:** The statistic  $Y$  is called a consistent estimator of the parameter  $\theta$  if  $Y$  converges stochastically to a parameter  $\theta$  as  $n$  approaches infinity. If  $\epsilon$  is an arbitrarily small positive number when  $Y$  is consistent, then

$$\lim_{n \rightarrow \infty} P(|Y - \theta| \leq \epsilon) = 1 \quad (3.3)$$

**Definition 3.3:** The statistic  $Y$  will be called the minimum variance unbiased estimator of the parameter  $\theta$  if  $Y$  is unbiased and the variance of  $Y$  is less than or equal to the variance of every other unbiased estimator of  $\theta$ . An estimator that has the property of minimum variance in large samples is said to be efficient.

**Definition 3.4:** The statistic  $Y$  is said to be sufficient for  $\theta$  if the conditional distribution of  $X$ , given  $Y = y$ , is independent of  $\theta$ .

Definition 3.3 is useful in finding a lower bound on the variance of all unbiased estimators. We now establish a lower bound inequality known as the Cramér-Rao inequality.

**Theorem 3.1 (Cramér-Rao inequality):** Let  $X_1, X_2, \dots, X_n$  denote a random sample from a distribution with pdf  $f(x; \theta)$  for  $\theta_1 < \theta < \theta_2$ , where  $\theta_1$  and  $\theta_2$  are known. Let  $Y = h(X_1, X_2, \dots, X_n)$  be an unbiased estimator of  $\theta$ . The lower bound inequality on the variance of  $Y$ ,  $\text{Var}(Y)$ , is given by

$$\text{Var}(Y) \geq \frac{1}{nE \left\{ \left[ \frac{\partial \ln f(x; \theta)}{\partial \theta} \right]^2 \right\}} \quad (3.4)$$

**Theorem 3.2:** An estimator  $\hat{\theta}$  is said to be asymptotically efficient if  $\sqrt{n}\hat{\theta}$  has a variance that approaches the Cramér-Rao lower bound for large  $n$ , that is,

$$\lim_{n \rightarrow \infty} \text{Var}(\sqrt{n}\hat{\theta}) = \frac{1}{nE \left\{ \left[ \frac{\partial \ln f(x; \theta)}{\partial \theta} \right]^2 \right\}}. \quad (3.5)$$

We now discuss some basic methods of parameter estimation.

### 3.2 Maximum Likelihood Estimation Method

The method of maximum likelihood estimation (MLE) is one of the most useful techniques for deriving point estimators. As a lead-in to this method, a simple example will be considered. The assumption that the sample is representative of the population will be exercised both in the example and later discussions.

*Example 3.1:* Consider a sequence of 25 Bernoulli trials (binomial situation) where each trial results in either success or failure. From the 25 trials, 6 failures and 19 successes result. Let  $p$  be the probability of success, and  $1-p$  the probability of failure. Find the estimator of  $p$ ,  $\hat{p}$ , which maximizes that particular outcome.

The sample density function can be written as

$$g(19) = \binom{25}{19} p^{19} (1-p)^6$$

The maximum of  $g(19)$  occurs when

$$p = \hat{p} = \frac{19}{25}$$

so that

$$g\left(19 \mid p = \frac{19}{25}\right) \geq g\left(19 \mid p \neq \frac{19}{25}\right)$$

Now  $g(19)$  is the probability or "likelihood" of 6 failures in a sequence of 25 trials.

Select  $p = \hat{p} = \frac{19}{25}$  as the probability or likelihood maximum value and, hence,  $\hat{p}$

is referred to as the maximum likelihood estimate. The reason for maximizing  $g(19)$  is that the sample contained six failures, and hence, if it is representative of the population, it is desired to find an estimate which maximizes this sample result. Just as  $g(19)$  was a particular sample estimate, in general, one deals with a sample density:

$$f(x_1, x_2, \dots, x_n) = f(x_1; \theta) f(x_2; \theta) \dots f(x_n; \theta) \quad (3.6)$$

where  $x_1, x_2, \dots, x_n$  are random, independent observation from a population with density function  $f(x)$ . For the general case, it is desired to find an estimate or estimates,  $\hat{\theta}_1, \hat{\theta}_2, \dots, \hat{\theta}_m$  (if such exist) where

$$f(x_1, x_2, \dots, x_n; \theta_1, \theta_2, \dots, \theta_m) > f(x_1, x_2, \dots, x_n; \theta'_1, \theta'_2, \dots, \theta'_m)$$

Notation  $\theta'_1, \theta'_2, \dots, \theta'_n$  refers to any other estimates different than  $\hat{\theta}_1, \hat{\theta}_2, \dots, \hat{\theta}_m$ .

Let us now discuss the method of MLE. Consider a random sample  $X_1, X_2, \dots, X_n$  from a distribution having pdf  $f(x; \theta)$ . This distribution has a vector  $\theta = (\theta_1, \theta_2, \dots, \theta_m)$  of unknown parameters associated with it, where  $m$  is the number of unknown parameters. Assuming that the random variables are independent, then

the likelihood function,  $L(X; \theta)$ , is the product of the probability density function evaluated at each sample point:

$$L(X, \theta) = \prod_{i=1}^n f(X_i; \theta) \quad (3.7)$$

where  $X = (X_1, X_2, \dots, X_n)$ . The maximum likelihood estimator  $\hat{\theta}$  is found by maximizing  $L(X; \theta)$  with respect to  $\theta$ . In practice, it is often easier to maximize  $\ln[L(X; \theta)]$  to find the vector of MLEs, which is valid because the logarithm function is monotonic. The log likelihood function is given by

$$\ln L(X, \theta) = \sum_{i=1}^n \ln f(X_i; \theta) \quad (3.8)$$

and is asymptotically normally distributed since it consists of the sum of  $n$  independent variables and the implication of the central limit theorem. Since  $L(X; \theta)$  is a joint probability density function for  $X_1, X_2, \dots, X_n$ , it must integrate equal to 1, that is,

$$\int_0^\infty \int_0^\infty \dots \int_0^\infty L(X; \theta) dX = 1$$

Assuming that the likelihood is continuous, the partial derivative of the left-hand side with respect to one of the parameters,  $\theta_i$ , yields

$$\begin{aligned} \frac{\partial}{\partial \theta_i} \int_0^\infty \int_0^\infty \dots \int_0^\infty L(X; \theta) dX &= \int_0^\infty \int_0^\infty \dots \int_0^\infty \frac{\partial}{\partial \theta_i} L(X; \theta) dX \\ &= \int_0^\infty \int_0^\infty \dots \int_0^\infty \frac{\partial \log L(X; \theta)}{\partial \theta_i} L(X; \theta) dX \\ &= E \left[ \frac{\partial \log L(X; \theta)}{\partial \theta_i} \right] \\ &= E[U_i(\theta)] \quad \text{for } i = 1, 2, \dots, m \end{aligned} \quad (3.9)$$

where  $U(\theta) = (U_1(\theta), U_2(\theta), \dots, U_m(\theta))'$  is often called the score vector and the vector  $U(\theta)$  has components

$$U_i(\theta) = \frac{\partial [\log L(X; \theta)]}{\partial \theta_i} \quad \text{for } i = 1, 2, \dots, m \quad (3.10)$$

which, when equated to zero and solved, yields the MLE vector  $\theta$ .

Suppose that we can obtain a non-trivial function of  $X_1, X_2, \dots, X_n$ , say  $h(X_1, X_2, \dots, X_n)$ , such that, when  $\theta$  is replaced by  $h(X_1, X_2, \dots, X_n)$ , the likelihood function  $L$  will achieve a maximum. In other words,

$$L(X, h(X)) \geq L(X, \theta)$$

for every  $\theta$ . The statistic  $h(X_1, X_2, \dots, X_n)$  is called a maximum likelihood estimator of  $\theta$  and will be denoted as

$$\hat{\theta} = h(x_1, x_2, \dots, x_n) \quad (3.11)$$

The observed value of  $\hat{\theta}$  is called the MLE of  $\theta$ . In general, the mechanics for obtaining the MLE can be obtained as follows:

- Step 1. Find the joint density function  $L(X, \theta)$
- Step 2. Take the natural log of the density  $\ln L$
- Step 3. Take the partial derivatives of  $\ln L$  with respect to each parameter
- Step 4. Set partial derivatives to “zero”
- Step 5. Solve for parameter(s)

*Example 3.2:* Let  $X_1, X_2, \dots, X_n$  be a random sample from the exponential distribution with pdf

$$f(x; \lambda) = \lambda e^{-\lambda x} \quad x > 0, \lambda > 0 \quad (3.12)$$

The joint pdf of  $X_1, X_2, \dots, X_n$ , is given by

$$L(X, \lambda) = \lambda^n e^{-\lambda \sum_{i=1}^n x_i}$$

and

$$\ln L(X, \lambda) = n \ln \lambda - \lambda \sum_{i=1}^n x_i$$

The function  $\ln L$  can be maximized by setting the first derivative of  $\ln L$ , with respect to  $\lambda$ , equal to zero and solving the resulting equation for  $\lambda$ . Therefore,

$$\frac{\partial \ln L}{\partial \lambda} = \frac{n}{\lambda} - \sum_{i=1}^n x_i = 0$$

This implies that

$$\hat{\lambda} = \frac{n}{\sum_{i=1}^n x_i} \quad (3.13)$$

The observed value of  $\hat{\lambda}$  is the maximum likelihood estimate of  $\lambda$ .

*Example 3.3:* In an exponential censored case, the non-conditional joint pdf that  $r$  items have failed is given by

$$f(x_1, x_2, \dots, x_r) = \lambda^r e^{-\lambda \sum_{i=1}^n x_i} \quad (r \text{ failed items}) \quad (3.14)$$

and the probability distribution that  $(n-r)$  items will survive is

$$P(X_{r+1} > t_1, X_{r+2} > t_2, \dots, X_n > t_{n-r}) = e^{-\lambda \sum_{j=1}^{n-r} t_j}$$

Thus, the joint density function is

$$\begin{aligned} L(X, \lambda) &= f(x_1, x_2, \dots, x_r) P(X_{r+1} > t_1, \dots, X_n > t_{n-r}) \\ &= \frac{n!}{(n-r)!} \lambda^r e^{-\lambda(\sum_{i=1}^r x_i + \sum_{j=1}^{n-r} t_j)} \end{aligned}$$

Let

$$T = \sum_{i=1}^r x_i + \sum_{j=1}^{n-r} t_j \quad (3.15)$$

then

$$\ln L = \ln \left( \frac{n!}{(n-r)!} \right) + r \ln \lambda - \lambda T$$

and

$$\frac{\partial \ln L}{\partial \lambda} = \frac{r}{\lambda} - T = 0$$

Hence,

$$\hat{\lambda} = \frac{r}{T} \quad (3.16)$$

Note that with the exponential, regardless of the censoring type or lack of censoring, the MLE of  $\lambda$  is the number of failures divided by the total operating time.

*Example 3.4:* Let  $X_1, X_2, \dots, X_n$  represent a random sample from the distribution with pdf

$$f(x; \theta) = e^{-(x-\theta)} \quad \text{for } \theta \leq x \leq \infty \quad \text{and } -\infty < \theta < \infty \quad (3.17)$$

The likelihood function is given by

$$\begin{aligned} L(\theta; X) &= \prod_{i=1}^n f(x_i; \theta) \quad \text{for } \theta \leq x_i \leq \infty \quad \text{all } i \\ &= \prod_{i=1}^n e^{-(x_i - \theta)} = e^{-\sum_{i=1}^n x_i + n\theta} \end{aligned}$$

For fixed values of  $x_1, x_2, \dots, x_n$ , we wish to find that value of  $\theta$  which maximizes  $L(\theta, X)$ . Here we cannot use the techniques of calculus to maximize  $L(\theta, X)$ . Note that  $L(\theta, X)$  is largest when  $\theta$  is as large as possible. However, the largest value of  $\theta$  is equal to the smallest value of  $X_i$  in the sample. Thus,

$$\hat{\theta} = \min\{X_i\} \quad 1 \leq i \leq n.$$

*Example 3.5:* Let  $X_1, X_2, \dots, X_n$ , denote a random sample from the normal distribution  $N(\mu, \sigma^2)$ . Then the likelihood function is given by

$$L(X, \mu, \sigma^2) = \left(\frac{1}{2\pi}\right)^{\frac{n}{2}} \frac{1}{\sigma^n} e^{-\frac{1}{2\sigma^2} \sum_{i=1}^n (x_i - \mu)^2}$$

and

$$\ln L = -\frac{n}{2} \log(2\pi) - \frac{n}{2} \log \sigma^2 - \frac{1}{2\sigma^2} \sum_{i=1}^n (x_i - \mu)^2$$

Thus we have

$$\begin{aligned} \frac{\partial \ln L}{\partial \mu} &= \frac{1}{\sigma^2} \sum_{i=1}^n (x_i - \mu) = 0 \\ \frac{\partial \ln L}{\partial \sigma^2} &= -\frac{n}{2\sigma^2} - \frac{1}{2\sigma^4} \sum_{i=1}^n (x_i - \mu)^2 = 0 \end{aligned}$$

Solving the two equations simultaneously, we obtain

$$\begin{aligned} \hat{\mu} &= \frac{\sum_{i=1}^n x_i}{n} \\ \hat{\sigma}^2 &= \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2 \end{aligned} \quad (3.18)$$

Note that the MLEs, if they exist, are both sufficient and efficient estimates. They also have an additional property called invariance, *i.e.*, for an MLE of  $\theta$ , then  $\mu(\theta)$  is the MLE of  $\mu(\theta)$ . However, they are not necessarily unbiased, *i.e.*,  $E(\hat{\theta}) = \theta$ . The point in fact is  $\sigma^2$ :

$$E(\hat{\sigma}^2) = \left(\frac{n-1}{n}\right) \sigma^2 \neq \sigma^2$$

Therefore, for small  $n$ ,  $\sigma^2$  is usually adjusted for its bias and the best estimate of  $\sigma^2$  is

$$\hat{\sigma}^2 = \left(\frac{1}{n-1}\right) \sum_{i=1}^n (x_i - \bar{x})^2$$

Sometimes it is difficult, if not impossible, to obtain maximum likelihood estimators in a closed form, and therefore numerical methods must be used to maximize the likelihood function. For illustration see the following example.

*Example 3.6:* Suppose that  $X_1, X_2, \dots, X_n$  is a random sample from the Weibull distribution with pdf

$$f(x, \alpha, \lambda) = \alpha \lambda x^{\alpha-1} e^{-\lambda x^\alpha} \quad (3.19)$$

The likelihood function is

$$L(X, \alpha, \lambda) = \alpha^n \lambda^n \prod_{i=1}^n x_i^{\alpha-1} e^{-\lambda \sum_{i=1}^n x_i^{\alpha}}$$

Then

$$\begin{aligned} \ln L &= n \log \alpha + n \log \lambda + (\alpha - 1) \sum_{i=1}^n \log x_i - \lambda \sum_{i=1}^n x_i^{\alpha} \\ \frac{\partial \ln L}{\partial \alpha} &= \frac{n}{\alpha} + \sum_{i=1}^n \log x_i - \lambda \sum_{i=1}^n x_i^{\alpha} \log x_i = 0 \\ \frac{\partial \ln L}{\partial \lambda} &= \frac{n}{\lambda} - \sum_{i=1}^n x_i^{\alpha} = 0 \end{aligned}$$

As noted, solutions of the above two equations for  $\alpha$  and  $\lambda$  are extremely difficult and require either graphical or numerical methods.

*Example 3.7:* Let  $X_1, X_2, \dots, X_n$  be a random sample from the gamma distribution with pdf

$$f(x, \lambda, \alpha) = \frac{\lambda^{(\alpha+1)} x^{\alpha} e^{-\lambda x}}{(\alpha!)^n} \quad (3.20)$$

then the likelihood function and log of the likelihood function, respectively, are

$$\begin{aligned} L(X, \lambda, \alpha) &= \frac{\lambda^{n(\alpha+1)} \prod_{i=1}^n x_i^{\alpha} e^{-\lambda \sum_{i=1}^n x_i}}{(\alpha!)^n} \\ \ln L &= n(\alpha+1) \log \lambda + \alpha \sum_{i=1}^n \log x_i - \lambda \sum_{i=1}^n x_i - n \log(\alpha!) \end{aligned}$$

Taking the partial derivatives, we obtain

$$\begin{aligned} \frac{\partial \ln L}{\partial \alpha} &= n \log \lambda + \sum_{i=1}^n \log x_i - n \frac{\partial}{\partial \alpha} [\log \alpha!] = 0 \\ \frac{\partial \ln L}{\partial \lambda} &= \frac{n(\alpha+1)}{\lambda} - \sum_{i=1}^n x_i = 0 \end{aligned} \quad (3.21)$$

The solutions of the two equations at equation (3.21) for  $\alpha$  and  $\lambda$  are extremely difficult and require either graphical or numerical methods.

*Example 3.8:* Let  $t_1, t_2, \dots, t_n$  be failure times of a random variable having the Pham distribution, also known as loglog distribution (Pham 2002a), with two parameters  $a$  and  $\alpha$  as follows (see also equation (2.21), Chapter 2):

$$f(t) = \alpha \cdot \ln a \cdot t^{\alpha-1} \cdot a^{t^{\alpha}} \cdot e^{1-a^{\alpha}} \quad \text{for } t > 0, \alpha > 0, a > 1 \quad (3.22)$$

From Chapter 2, equation (2.22), the Pham cumulative distribution function is given as follows:

$$F(t) = 1 - e^{1-a^t} \quad (3.23)$$

We now estimate the values of  $a$  and  $\alpha$  using the MLE method. From equation (3.22), the likelihood function is

$$\begin{aligned} L(a, \alpha) &= \prod_{i=1}^n \alpha \ln a \cdot t_i^{\alpha-1} e^{1-a^{t_i}} a^{t_i^\alpha} \\ &= \alpha^n (\ln a)^n \left( \prod_{i=1}^n t_i \right)^{\alpha-1} a^{\sum_{i=1}^n t_i^\alpha} e^{n - \sum_{i=1}^n a^{t_i}} \end{aligned}$$

The log likelihood function is

$$\begin{aligned} \log L(a, \alpha) &= n \log \alpha + n \ln(\ln a) + (\alpha - 1) \left( \sum_{i=1}^n \ln t_i \right) \\ &\quad + \ln a \cdot \sum_{i=1}^n t_i^\alpha + n - \sum_{i=1}^n a^{t_i} \end{aligned} \quad (3.24)$$

The first derivatives of the log likelihood function with respect to  $a$  and  $\alpha$  are, respectively,

$$\frac{\partial}{\partial a} \log L(a, \alpha) = \frac{n}{a \ln a} + \frac{1}{a} \cdot \sum_{i=1}^n t_i^\alpha - \sum_{i=1}^n t_i^\alpha a^{t_i-1} \quad (3.25)$$

and

$$\begin{aligned} \frac{\partial}{\partial \alpha} \log L(a, \alpha) &= \frac{n}{\alpha} + \sum_{i=1}^n \ln t_i + \ln a \cdot \sum_{i=1}^n \ln t_i \cdot t_i^\alpha \\ &\quad - \sum_{i=1}^n t_i^\alpha \cdot a^{t_i} \cdot \ln a \cdot \ln t_i \end{aligned} \quad (3.26)$$

Setting equations (3.25) and (3.26) equal to zero, we can obtain the MLE of  $a$  and  $\alpha$  by solving the following simultaneous equations:

$$\frac{n}{\ln a} + \sum_{i=1}^n t_i^\alpha - \sum_{i=1}^n t_i^\alpha a^{t_i} = 0$$

$$\frac{n}{\alpha} + \sum_{i=1}^n \ln t_i + \ln a \cdot \sum_{i=1}^n \ln t_i \cdot t_i^\alpha (1 - a^{t_i}) = 0$$

After rearrangements, we obtain

$$\ln a \sum_{i=1}^n t_i^\alpha (a^{t_i} - 1) = n$$

$$\ln a \cdot \sum_{i=1}^n \ln t_i \cdot t_i^\alpha \cdot (a^{t_i} - 1) - \frac{n}{\alpha} = \sum_{i=1}^n \ln t_i$$



### 3.3 Maximum Likelihood Estimation with Censored Data

Censored data arises when an individual's life length is known to occur only in a certain period of time. In other words, a censored observation contains only partial information about the random variable of interest. In this section, we consider two types of censoring. The first type is called Type-I censoring where the event is observed only if it occurs prior to some pre-specified time. The second type is Type-II censoring in which the study continues until the failure of the first  $r$  units (or components), where  $r$  is some predetermined integer ( $r < n$ ).

Examples of Type-II censoring are often used in testing of equipment life. Here items are put on test at the same time, and the test is terminated when  $r$  of the  $n$  items have failed. Such an experiment may, however, save time and resources because it could take a very long time for all items to fail. Both Type-I and Type-II censoring arise in many reliability applications.

For example, there is a batch of transistors or tubes; we put them all on test at  $t=0$ , and record their times to failure. Some transistors may take a long time to burn out, and we will not want to wait that long to end the experiment. Therefore, we might stop the experiment at a pre-specified time  $t_c$ , in which case we have Type-I censoring, or we might not know beforehand what value of the fixed censoring time is good so we decide to wait until a pre-specified number of units have failed,  $r$ , of all the transistors has burned out, in which case we have Type-II censoring.

Censoring times may vary from individual to individual or from application to application. We now discuss a generalized censoring times case, call a multiple-censored data.

#### 3.3.1 Parameter Estimate with Multiple-censored Data

The likelihood function for the multiple-censored data is given by

$$L = f(t_{1,f}, \dots, t_{r,f}, t_{1,s}, \dots, t_{m,s}) = C \prod_{i=1}^r f(t_{i,f}) \prod_{j=1}^m [1 - F(t_{j,s})] \quad (3.27)$$

where  $C$  is a constant,  $f(\cdot)$  is the density function and  $F(\cdot)$  is the distribution function. There are  $r$  failure at times  $t_{1,f}, \dots, t_{r,f}$  and  $m$  units with censoring times  $t_{1,s}, \dots, t_{m,s}$ .

Note that this includes Type-I censoring by simply setting  $t_{i,f} = t_{i,n}$  and  $t_{j,s} = t_0$  in the likelihood function in equation (3.27). Also, the likelihood function for Type-II censoring is similar to Type-I censoring except  $t_{j,s} = t_r$  in equation (3.27). In other words, the likelihood function for the first  $r$  observations from a sample size  $n$  drawn from the model in both Type-I and Type-II censoring is given by

$$L = f(t_{1,n}, \dots, t_{r,n}) = C \prod_{i=1}^r f(t_{i,n}) [1 - F(t_*)]^{n-r} \quad (3.28)$$

where  $t_* = t_0$ , the time of cessation of the test for Type-I censoring and  $t_* = t_r$ , the time of the  $r^{\text{th}}$  failure for Type-II censoring.

*Example 3.9:* Consider a two-parameter probability density distribution with multiple-censored data and distribution function with failure rate bathtub shape, as given by (Chen 2000):

$$f(t) = \lambda \beta t^{\beta-1} \exp[\lambda t^\beta + \lambda(1 - e^{t^\beta})] \quad t, \lambda, \beta > 0 \quad (3.29)$$

and

$$F(t) = 1 - \exp[\lambda(1 - e^{t^\beta})] \quad t, \lambda, \beta > 0 \quad (3.30)$$

respectively.

Substituting the functions  $f(t)$  and  $F(t)$  in equations (3.29) and (3.30) into equation (3.28), we obtain the logarithm of the likelihood function:

$$\begin{aligned} \ln L = \ln C + r \ln \lambda + r \ln \beta + \sum_{i=1}^r (\beta - 1) \ln t_i \\ + (m + r) \lambda + \sum_{i=1}^r t_i^\beta - \left[ \sum_{i=1}^r \lambda e^{t_i^\beta} + \sum_{j=1}^m \lambda e^{t_j^\beta} \right] \end{aligned} \quad (3.31)$$

The function  $\ln L$  can be maximized by setting the partial derivative of  $\ln L$  with respect to  $\lambda$  and  $\beta$ , equal to zero and solving the resulting equations simultaneously for  $\lambda$  and  $\beta$ . Therefore, we obtain

$$\begin{aligned} \frac{\partial \ln L}{\partial \lambda} &= \frac{r}{\lambda} + (m + r) - \sum_{i=1}^r e^{t_i^\beta} - \sum_{j=1}^m e^{t_j^\beta} \equiv 0 \\ \frac{\partial \ln L}{\partial \beta} &= \frac{r}{\beta} + \sum_{i=1}^r \ln t_i + \sum_{i=1}^r t_i^\beta \ln t_i \\ &\quad - \lambda \left[ \sum_{i=1}^r e^{t_i^\beta} t_i^\beta \ln t_i + \sum_{j=1}^m e^{t_j^\beta} t_j^\beta \ln t_j \right] \equiv 0 \end{aligned} \quad (3.32)$$

This implies that

$$\hat{\lambda} = \frac{r}{\left( \sum_{i=1}^r e^{t_i^{\hat{\beta}}} + \sum_{j=1}^m e^{t_j^{\hat{\beta}}} \right) - m - r} \quad (3.33)$$

and  $\hat{\beta}$  is the solution of

$$\begin{aligned} \frac{r}{\hat{\beta}} + \sum_{i=1}^r \ln t_i + \sum_{i=1}^r t_i^{\hat{\beta}} \ln t_i = \\ \frac{r}{\left( \sum_{i=1}^r e^{t_i^{\hat{\beta}}} + \sum_{j=1}^m e^{t_j^{\hat{\beta}}} \right) - m - r} \left[ \sum_{i=1}^r e^{t_i^{\hat{\beta}}} t_i^{\hat{\beta}} \ln t_i + \sum_{j=1}^m e^{t_j^{\hat{\beta}}} t_j^{\hat{\beta}} \ln t_j \right] \end{aligned} \quad (3.34)$$

We now discuss two special cases as follows.

*Case I: Type-I or Type-II Censoring Data.*

From equation (3.28), the likelihood function for the first  $r$  observations from a sample size  $n$  drawn from the model in both Type-I and Type-II censoring is

$$L = f(t_{1,n}, \dots, t_{r,n}) = C \prod_{i=1}^r f(t_{i,n}) [1 - F(t_*)]^{n-r}$$

where  $t_* = t_0$ , the time of cessation of the test for Type-I censoring and  $t_* = t_r$ , the time of the  $r^{\text{th}}$  failure for Type-II censoring equation (3.33) and (3.34) become

$$\hat{\lambda} = \frac{r}{\sum_{i=1}^r e^{t_i^{\hat{\beta}}} + (n-r)e^{t_r^{\hat{\beta}}} - n} \quad (3.35)$$

$$\begin{aligned} \frac{r}{\hat{\beta}} + \sum_{i=1}^r \ln t_i + \sum_{i=1}^r t_i^{\hat{\beta}} \ln t_i = \\ \frac{r}{\sum_{i=1}^r e^{t_i^{\hat{\beta}}} + (n-r)e^{t_r^{\hat{\beta}}} - n} \left[ \sum_{i=1}^r e^{t_i^{\hat{\beta}}} t_i^{\hat{\beta}} \ln t_i + \sum_{j=1}^m e^{t_j^{\hat{\beta}}} t_j^{\hat{\beta}} \ln t_j \right] \end{aligned} \quad (3.36)$$

*Case II: Complete Censored Data.*

Simply replace  $r$  with  $n$  in equations (3.33) and (3.34) and ignore the  $t_j$  portions. The maximum likelihood equations for the  $\lambda$  and  $\beta$  are given by

$$\hat{\lambda} = \frac{n}{\sum_{i=1}^n e^{t_i^{\hat{\beta}}} - n} \quad (3.37)$$

$$\frac{n}{\hat{\beta}} + \sum_{i=1}^n \ln t_i + \sum_{i=1}^n t_i^{\hat{\beta}} \ln t_i = \frac{n}{\sum_{i=1}^n e^{t_i^{\hat{\beta}}} - n} \times \sum_{i=1}^n e^{t_i^{\hat{\beta}}} t_i^{\hat{\beta}} \ln t_i \quad (3.38)$$

**3.3.2 Confidence Intervals of Estimates**

The asymptotic variance-covariance matrix of the parameters ( $\lambda$  and  $\beta$ ) is obtained by inverting the Fisher information matrix

$$I_{ij} = E \left[ -\frac{\partial^2 L}{\partial \theta_i \partial \theta_j} \right], \quad i, j = 1, 2 \quad (3.39)$$

where  $\theta_1, \theta_2 = \lambda$  or  $\beta$  (Nelson 1990). This leads to

$$\begin{bmatrix} \text{Var}(\hat{\lambda}) & \text{Cov}(\hat{\lambda}, \hat{\beta}) \\ \text{Cov}(\hat{\lambda}, \hat{\beta}) & \text{Var}(\hat{\beta}) \end{bmatrix} = \begin{bmatrix} E\left(-\frac{\partial^2 \ln L}{\partial^2 \lambda} \Big|_{\hat{\lambda}, \hat{\beta}}\right) & E\left(-\frac{\partial^2 \ln L}{\partial \lambda \partial \beta} \Big|_{\hat{\lambda}, \hat{\beta}}\right) \\ E\left(-\frac{\partial^2 \ln L}{\partial \beta \partial \lambda} \Big|_{\hat{\lambda}, \hat{\beta}}\right) & E\left(-\frac{\partial^2 \ln L}{\partial^2 \beta} \Big|_{\hat{\lambda}, \hat{\beta}}\right) \end{bmatrix}^{-1} \quad (3.40)$$

We can obtain an approximate  $(1 - \alpha)100\%$  confidence intervals on parameter  $\lambda$  and  $\beta$  based on the asymptotic normality of the MLEs (Nelson 1990) as follows:

$$\hat{\lambda} \pm Z_{\alpha/2} \sqrt{\text{Var}(\hat{\lambda})} \quad \text{and} \quad \hat{\beta} \pm Z_{\alpha/2} \sqrt{\text{Var}(\hat{\beta})} \quad (3.41)$$

where  $Z_{\alpha/2}$  is upper percentile of standard normal distribution.

### 3.3.3 Applications

Consider a helicopter main rotor blade part code xxx-015-001-107 based on the system database collected from October 1995 to September 1999 (Pham 2002a). The data set is shown in Table 3.1. In this application, we consider several distribution functions including Weibull, lognormal, normal, and loglog distribution functions.

From Example 3.8, the loglog pdf (see equation 3.22) with parameters  $a$  and  $\alpha$  is

$$f(t) = \alpha \cdot \ln a \cdot t^{\alpha-1} \cdot a^{t^\alpha} \cdot e^{1-a^\alpha} \quad \text{for } t > 0, \alpha > 0, a > 1$$

and its corresponding log likelihood function (see equation 3.24) is

$$\begin{aligned} \log L(a, \alpha) = & n \log \alpha + n \ln(\ln a) + (\alpha - 1) \left( \sum_{i=1}^n \ln t_i \right) \\ & + \ln a \cdot \sum_{i=1}^n t_i^\alpha + n - \sum_{i=1}^n a^{t_i^\alpha} \end{aligned}$$

We next determine the confidence intervals for parameter estimates  $a$  and  $\alpha$ . For the log-likelihood function given in equation (3.24), we can obtain the Fisher

information matrix  $H$  as  $H = \begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix}$  where  $h_{11} = E \left[ -\frac{\partial^2 \log L}{\partial a^2} \right]$

$$h_{12} = h_{21} = E \left[ -\frac{\partial^2 \log L}{\partial a \partial \alpha} \right]$$

$$h_{22} = E \left[ -\frac{\partial^2 \log L}{\partial \alpha^2} \right]$$

The variance matrix,  $V$ , can be obtained as follows:

$$V = [H]^{-1} = \begin{bmatrix} v_{11} & v_{12} \\ v_{21} & v_{22} \end{bmatrix} \quad (3.42)$$

The variances of  $a$  and  $\alpha$  are

$$\text{Var}(a) = v_{11} \quad \text{Var}(\alpha) = v_{22}$$

One can approximately obtain the  $100(1-\beta)\%$  confidence intervals for  $a$  and  $\alpha$  based on the normal distribution as  $[\hat{a} - z_{\beta/2} \sqrt{v_{11}}, \hat{a} + z_{\beta/2} \sqrt{v_{11}}]$  and  $[\hat{\alpha} - z_{\beta/2} \sqrt{v_{22}}, \hat{\alpha} + z_{\beta/2} \sqrt{v_{22}}]$ , respectively, where  $v_{ij}$  is given in equation (3.42) and  $z_{\beta}$  is  $(1-\beta/2)100\%$  of the standard normal distribution. After we obtain  $\hat{a}$  and  $\hat{\alpha}$ , the MLE of reliability function can be computed as

$$\hat{R}(t) = e^{1-\hat{a}t^{\hat{\alpha}}}$$

Let us define a partial derivative vector for reliability  $R(t)$  as

$$v[R(t)] = \begin{bmatrix} \frac{\partial R(t)}{\partial a} & \frac{\partial R(t)}{\partial \alpha} \end{bmatrix}$$

then the variance of  $R(t)$  can be obtained as follows:

$$\text{Var}[R(t)] = v[R(t)] \cdot V \cdot (v[R(t)])^T \quad (3.43)$$

where  $V$  is given in equation (3.42).

One can approximately obtain the  $(1-\beta)100\%$  confidence interval for  $R(t)$  as

$$[\hat{R}(t) - z_{\beta/2} \sqrt{\text{Var}[R(t)]}, \hat{R}(t) + z_{\beta/2} \sqrt{\text{Var}[R(t)]}].$$

The MLE parameter estimations of Pham distribution using the data set in Table 3.1 are given as follows:

$$\begin{aligned} \hat{\alpha} &= 1.1075 & \text{Var}[\hat{\alpha}] &= 0.0162 \\ 95\% \text{ CI for } \hat{\alpha} &: [0.8577, 1.3573] \end{aligned}$$

$$\begin{aligned} \hat{a} &= 1.0002 & \text{Var}[\hat{a}] &= 2.782 e^{-8} \\ 95\% \text{ CI for } a &: [0.9998, 1.0005] \end{aligned}$$

$$\begin{aligned} \text{MTTF} &= 1608.324 \\ \text{MRL}(t=\text{MTTF}) &= 950.475 \end{aligned}$$

Substituting  $a = 1.0002$  and  $\alpha = 1.1075$  into the equation for  $R(t)$  yields the results in Table 3.2. Figure 3.1 shows the reliability comparisons between the normal

model, the lognormal model, the Weibull model and the loglog model for the main rotor blade data set.

**Table 3.1.** Main rotor blade data (hour)

1634.3	2094.3	3318.2
1100.5	2166.2	2317.3
1100.5	2956.2	1081.3
819.9	795.5	1953.5
1398.3	795.5	2418.5
1181	204.5	1485.1
128.7	204.5	2663.7
1193.6	1723.2	1778.3
254.1	403.2	1778.3
3078.5	2898.5	2943.6
3078.5	2869.1	2260
3078.5	26.5	2299.2
26.5	26.5	1655
26.5	3180.6	1683.1
3265.9	644.1	1683.1
254.1	1898.5	2751.4
2888.3	3318.2	
2080.2	1940.1	

### 3.4 Statistical Change-point Estimation Methods

The change-point problem has been widely studied in reliability applications such as biological sciences, survival analysis, and environmental statistics.

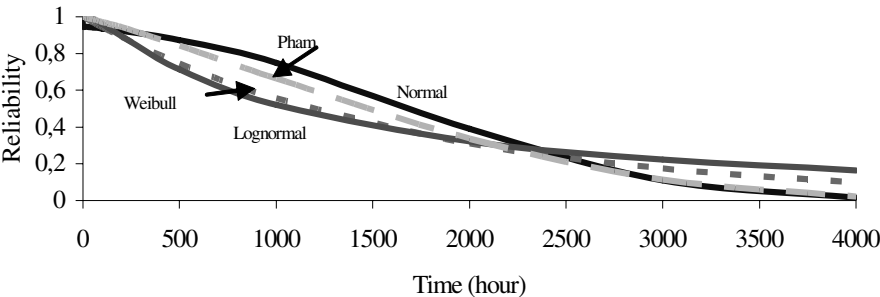
Assume there is a sequence of random variables  $X_1, X_2, \dots, X_n$ , that represent the inter-failure times and exists an index change-point  $\tau$ , such that  $X_1, X_2, \dots, X_\tau$  have a common distribution  $F$  with density function  $f(t)$  and  $X_{\tau+1}, X_{\tau+2}, \dots, X_n$  have the distribution  $G$  with density function  $g(t)$ , where  $F \neq G$ . Consider the following assumptions:

1. There is a finite unknown number of units,  $N$ , to put under the test.
2. At the beginning, all of the units have the same lifetime distribution  $F$ . After  $\tau$  failures are observed, the remaining  $(N - \tau)$  items have the distribution  $G$ . The change-point  $\tau$  is assumed unknown.

- 3. The sequence  $\{X_1, X_2, \dots, X_r\}$  is statistically independent of the sequence  $\{X_{r+1}, X_{r+2}, \dots, X_n\}$ .
- 4. The lifetime test is performed according to the Type-II censoring plan in which the number of failures,  $n$ , is pre-determined.

**Table 3.2.** Reliability of a main rotor blade for various mission time  $t$  (hour)

Time	Reliability	Time	Reliability
10	0.9974	450	0.8274
20	0.9945	500	0.8063
30	0.9914	600	0.7637
40	0.9881	700	0.7209
50	0.9848	800	0.6781
100	0.9672	900	0.6354
150	0.9486	1000	0.5931
200	0.9294	1500	0.3939
250	0.9096	2000	0.2292
300	0.8895	2500	0.1122
350	0.8690	3000	0.0436
400	0.8483	3500	0.0125



**Figure 3.1.** Reliability comparisons for a main rotor blade data set

Note that in hardware reliability testing, the total number of units to put on the test  $N$  can be determined in advance. But in software, the parameter  $N$  can be defined as the initial number of faults, and therefore it makes more sense for it to be an unknown parameter. Let  $T_1, T_2, \dots, T_n$  be the arrival times of sequential failures.

Then

$$\begin{aligned} T_1 &= X_1 \\ T_2 &= X_1 + X_2 \\ &\vdots \\ T_n &= X_1 + X_2 + \dots X_n \end{aligned} \quad (3.44)$$

The failure times  $T_1, T_2, \dots, T_\tau$  are the first  $\tau$  order statistics of a sample of size  $N$  from the distribution  $F$ . The failure times  $T_{\tau+1}, T_{\tau+2}, \dots, T_n$  are the first  $(n-\tau)$  order statistics of a sample of size  $(N-\tau)$  from the distribution  $G$ .

*Example 3.10:* The Weibull change-point model of given life time distributions  $F$  and  $G$  with parameters  $(\lambda_1, \beta_1)$  and  $(\lambda_2, \beta_2)$ , respectively, can be expressed as follows:

$$F(t) = 1 - \exp(-\lambda_1 t^{\beta_1}) \quad (3.45)$$

$$G(t) = 1 - \exp(-\lambda_2 t^{\beta_2}). \quad (3.46)$$

Assume that the distributions belong to parametric families  $\{F(t|\theta_1), \theta_1 \in \Theta_1\}$  and  $\{G(t|\theta_2), \theta_2 \in \Theta_2\}$ . Assume  $T_1, T_2, \dots, T_\tau$  are the first  $\tau$  order statistics of a sample with size  $N$  from the distribution  $\{F(t|\theta_1), \theta_1 \in \Theta_1\}$  and  $T_{\tau+1}, T_{\tau+2}, \dots, T_n$  are the first  $(n-\tau)$  order statistics of a sample of size  $(N-\tau)$  from the distribution  $\{G(t|\theta_2), \theta_2 \in \Theta_2\}$  where  $N$  is unknown. The log likelihood function can be expressed as follows (Zhao 2003):

$$\begin{aligned} L(\tau, N, \theta_1, \theta_2 | T_1, T_2, \dots, T_n) &= \sum_{i=1}^n (N-i+1) + \sum_{i=1}^{\tau} f(T_i | \theta_1) \\ &\quad + \sum_{i=\tau+1}^n g(T_i | \theta_2) + (N-\tau) \log(1 - F(T_\tau | \theta_1)) \\ &\quad + (N-n) \log(1 - G(T_n | \theta_2)) \end{aligned} \quad (3.47)$$

If the parameter  $N$  is known where hardware reliability is commonly considered, then the likelihood function is given by

$$\begin{aligned} L(\tau, \theta_1, \theta_2 | T_1, T_2, \dots, T_n) &= \sum_{i=1}^{\tau} f(T_i | \theta_1) + \sum_{i=\tau+1}^n g(T_i | \theta_2) \\ &\quad + (N-\tau) \log(1 - F(T_\tau | \theta_1)) + (N-n) \log(1 - G(T_n | \theta_2)) \end{aligned}$$

The maximum likelihood estimator (MLE) of the change-point value  $\hat{\tau}$  and  $(\hat{N}, \hat{\theta}_1, \hat{\theta}_2)$  can be obtained by taking partial derivatives of the log likelihood



function in equation (3.47) with respect to the unknown parameters that maximizes the function. It should be noted that there is no closed form for  $\hat{\tau}$  but it can be obtained by calculating the log likelihood for each possible value of  $\tau$ ,  $1 \leq \tau \leq (n-1)$ , and selecting as  $\hat{\tau}$  the value that maximizes the log-likelihood function.

### 3.4.1 Application: A Software Model with a Change Point

In this application we examine the case where the sample size  $N$  is unknown. Consider a software reliability model developed by Jelinski and Moranda (1972), often called the Jelinski-Moranda model. The assumptions of the model are as follows:

1. There are  $N$  initial faults in the program.
2. A detected fault is removed instantaneously and no new fault is introduced.
3. Each failure caused by a fault occurs independently and randomly in time according to an exponential distribution.
4. The functions  $F$  and  $G$  are exponential distributions with failure rate parameters  $\lambda_1$  and  $\lambda_2$ , respectively.

Based on the assumptions, the inter-failure times  $X_1, X_2, \dots, X_n$  are independently exponentially distributed. Specifically,  $X_i = T_i - T_{i-1}$ ,  $i = 1, 2, \dots, \tau$ , are exponentially distributed with parameter  $\lambda_1 (N - i + 1)$  where  $\lambda_1$  is the initial fault detection rate of the first  $\tau$  failures and  $X_j = T_j - T_{j-1}$ ,  $j = \tau + 1, \tau + 2, \dots, n$ , are exponentially distributed with parameter  $\lambda_2 (N - \tau - j + 1)$  where  $\lambda_2$  is the fault detection rate of the first  $n - \tau$  failures. If  $\lambda_1 = \lambda_2$  it means that each fault removal is the same and the change-point model becomes the Jelinski-Moranda software reliability model (Jelinski and Moranda 1972).

The MLEs of the parameters  $(\tau, N, \lambda_1, \lambda_2)$  can be obtained by solving the following equations simultaneously:

$$\hat{\lambda}_1 = \frac{\tau}{\sum_{i=1}^{\tau} (\hat{N} - i + 1) x_i} \quad (3.48)$$

$$\hat{\lambda}_2 = \frac{(n - \tau)}{\sum_{i=\tau+1}^n (\hat{N} - i + 1) x_i} \quad (3.49)$$

$$\sum_{i=1}^n \frac{1}{(\hat{N} - i + 1)} = \hat{\lambda}_1 \sum_{i=1}^{\tau} x_i + \hat{\lambda}_2 \sum_{i=\tau+1}^n x_i \quad (3.50)$$

To illustrate the model, we use the data set as in Table 3.3 to obtain the unknown parameters  $(\tau, N, \lambda_1, \lambda_2)$  using equations (3.48)–(3.50). The data in Table 3.3 (Musa et al.1987) shows the successive inter-failure times for a real-time command and control system.

The table reads from left to right in rows, and the recorded times are execution times, in seconds. There are 136 failures in total. Figure 3.2 plots the log-likelihood function *vs* number of failures. The MLEs of the parameters  $(\tau, N, \lambda_1, \lambda_2)$  with one change-point are given by

$$\hat{\tau} = 16, \hat{N} = 145, \hat{\lambda}_1 = 1.1 \times 10^{-4}, \hat{\lambda}_2 = 0.31 \times 10^{-4}.$$

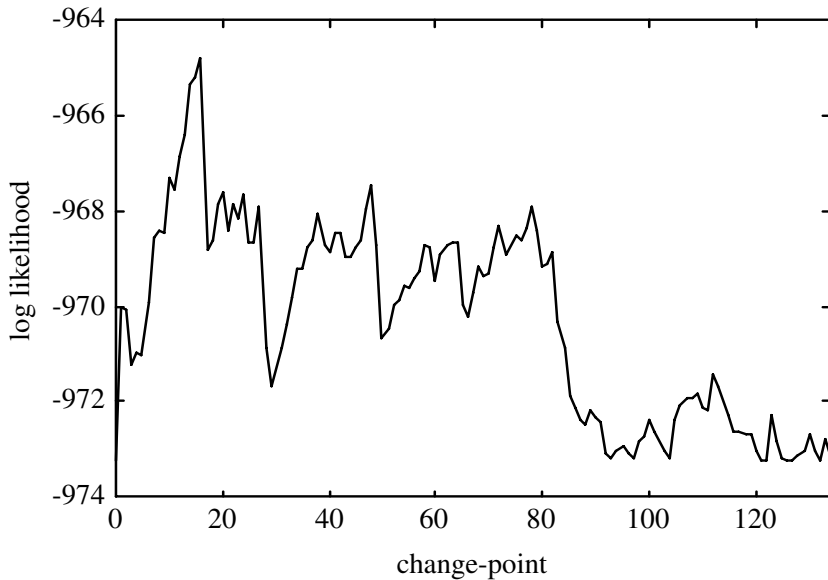
If we do not consider a change-point in the model, the MLEs of the parameters  $N$  and  $\lambda$  can be given as

$$\hat{N} = 142, \hat{\lambda} = 0.35 \times 10^{-4}.$$

From Figure 3.2, we can observe that it is worth considering the change-points in the reliability functions.

**Table 3.3.** Successive inter-failure times (in seconds) for a real-time command system

3	30	113	81	115	9	2	91	112	15
138	50	77	24	108	88	670	120	26	114
325	55	242	68	422	180	10	1146	600	15
36	4	0	8	227	65	176	58	457	300
97	263	452	255	197	193	6	79	816	1351
148	21	233	134	357	193	236	31	369	748
0	232	330	365	1222	543	10	16	529	379
44	129	810	290	300	529	281	160	828	1011
445	296	1755	1064	1783	860	983	707	33	868
724	2323	2930	1461	843	12	261	1800	865	1435
30	143	108	0	3110	1247	943	700	875	245
729	1897	447	386	446	122	990	948	1082	22
75	482	5509	100	10	1071	371	790	6150	3321
1045		648	5485	1160	1864	4116			



**Figure 3.2.** The log-likelihood function versus the number of failures

### 3.5 Goodness of Fit Techniques

The problem at hand is to compare some observed sample distribution with a theoretical distribution. Two common techniques that will be discussed are the  $\chi^2$  goodness-of-fit test and the Kolmogorov-Smirnov "d" test.

#### 3.5.1 Chi-squared Test

The following statistic

$$\chi^2 = \sum_{i=1}^k \left( \frac{x_i - \mu_i}{\sigma_i} \right)^2 \quad (3.51)$$

has a chi-squared ( $\chi^2$ ) distribution with  $k$  degrees of freedom. The steps of chi-squared test are as follows:

1. Divide the sample data into the mutually exclusive cells (normally 8-12) such that the range of the random variable is covered.
2. Determine the frequency,  $f_i$ , of sample observations in each cell.
3. Determine the theoretical frequency,  $F_i$ , for each cell (area under density function between cell boundaries  $X_n$  - total sample size). Note that the theoretical frequency for each cell should be greater than 1. To carry out this step, it normally requires estimates of the population parameters which can be obtained from the sample data.

## 4. Form the statistic

$$S = \sum_{i=1}^k \frac{(f_i - F_i)^2}{F_i} \quad (3.52)$$

5. From the  $\chi^2$  tables, choose a value of  $\chi^2$  with the desired significance level and with degrees of freedom ( $= k-1-r$ ), where  $r$  is the number of population parameters estimated.
6. Reject the hypothesis that the sample distribution is the same as theoretical distribution if

$$S > \chi^2_{1-\alpha, k-1-r}$$

where  $\alpha$  is called the significance level.

*Example 3.11:* Given the data in Table 3.4, can the data be represented by the exponential distribution with a significance level of  $\alpha$ ?

From the above calculation,  $\hat{\lambda} = 0.00263$ ,  $R_i = e^{-\lambda t_i}$  and  $Q_i = 1 - R_i$ . Given that a value of significance level  $\alpha$  is 0.1, from equation (3.52) we obtain

$$S = \sum_{i=1}^{11} \frac{(f_i - F_i)^2}{F_i} = 6.165$$

From Table A1.3 in Appendix 1, the value of  $\chi^2$  with nine degrees of freedom is 14.68, that is,

$$\chi^2_{9df}(.90) = 14.68$$

**Table 3.4.** Sample observations in each cell boundary

Cell boundaries	$f_i$	$Q_i = (1-R_i)$ 60	$F_i = Q_i - Q_{i-1}$
0-100	10	13.86	13.86
100-200	9	24.52	10.66
200-300	8	32.71	8.19
300-400	8	39.01	6.30
400-500	7	43.86	4.85
500-600	6	47.59	3.73
600-700	4	50.45	2.86
700-800	4	52.66	2.21
800-900	2	54.35	1.69
900-1,000	1	55.66	1.31
>1,000	1	58.83	2.17

Since  $S = 6.165 < 14.68$ , we would not reject the hypothesis of exponential with  $\lambda = 0.00263$ . If in the following statistic

$$S = \sum_{i=1}^k \left( \frac{f_i - F_i}{\sqrt{F_i}} \right)^2, \quad \left( \frac{f_i - F_i}{\sqrt{F_i}} \right)$$

is approximately normal for large samples, then  $S$  also has a  $\chi^2$  distribution. This is the basis for the goodness of fit test.

### 3.5.2 Kolmogorov-Smirnov $d$ Test

Both the  $\chi^2$  and “ $d$ ” tests are non-parameters. However, the  $\chi^2$  assumes large sample normality of the observed frequency about its mean while the “ $d$ ” only assumes a continuous distribution. Let  $X_1 \leq X_2 \leq X_3 \leq \dots \leq X_n$  denote the ordered sample values. Define the observed distribution function,  $F_n(x)$ , as follows:

$$F_n(X) = \begin{cases} 0 & \text{for } x \leq x_1 \\ \frac{i}{n} & \text{for } x_i < x \leq x_{i+1} \\ 1 & \text{for } x > x_n \end{cases}$$

Assume the testing hypothesis

$$H_0 : F(x) = F_0(x)$$

where  $F_0(x)$  is a given continuous distribution and  $F(x)$  is an unknown distribution. Let

$$d_n = \sup_{-\infty < x < \infty} |F_n(x) - F_0(x)|$$

Since  $F_0(x)$  is a continuous increasing function, we can evaluate  $|F_n(x) - F_0(x)|$  for each  $n$ . If  $d_n \leq d_{n,\alpha}$  then we would not reject the hypothesis  $H_0$ ; otherwise, we would reject it when  $d_n > d_{n,\alpha}$ . The value  $d_{n,\alpha}$  can be found in Table A1.4 in Appendix 1, where  $n$  is the sample size and  $\alpha$  is the level of significance.

## 3.6 Least Squared Estimation

A problem of curve fitting, which is unrelated to normal regression theory and MLE estimates of coefficients but uses identical formulas, is called the method of least squares. This method is based on minimizing the sum of the squared distance from the best fit line and the actual data points. It just so happens that finding the MLEs for the coefficients of the regression line also involves these sums of squared distances.

### Normal Linear Regression

Regression considers the distributions of one variable when another is held fixed at each of several levels. In the bivariate normal case, consider the distribution of  $X$  as a function of given values of  $Z$  where  $X = \alpha + \beta Z$ . Consider a sample of  $n$  observations  $(x_i, z_i)$ , we can obtain the likelihood and its natural log for the normal distribution as follows:

$$f(x_1, x_2, \dots, x_n) = \frac{1}{2\pi^{\frac{n}{2}}} \left(\frac{1}{\sigma^2}\right)^{\frac{n}{2}} e^{-\frac{1}{2\sigma^2} \sum_{i=1}^n (x_i - \alpha - \beta z_i)^2}$$

$$\ln L = -\frac{n}{2} \log 2\pi - \frac{n}{2} \log \sigma^2 - \frac{1}{2\sigma^2} \sum_{i=1}^n (x_i - \alpha - \beta z_i)^2$$

Taking the partial derivatives of  $\ln L$  with respect to  $\alpha$  and  $\beta$ , we have

$$\begin{aligned}\frac{\partial \ln L}{\partial \alpha} &= \sum_{i=1}^n (x_i - \alpha - \beta z_i)^2 = 0 \\ \frac{\partial \ln L}{\partial \beta} &= \sum_{i=1}^n z_i (x_i - \alpha - \beta z_i) = 0\end{aligned}$$

The solution of the simultaneous equations is

$$\begin{aligned}\hat{\alpha} &= \bar{X} - \beta \bar{Z} \\ \hat{\beta} &= \frac{\sum_{i=1}^n (X_i - \bar{X})(Z_i - \bar{Z})}{\sum_{i=1}^n (Z_i - \bar{Z})^2}\end{aligned}\tag{3.53}$$

### ***Least Squared Straight Line Fit***

Assume there is a linear relationship between  $X$  and  $E(Y|x)$ , that is,  $E(Y|x) = a + bx$ . Given a set of data, we want to estimate the coefficients  $a$  and  $b$  that minimize the sum of the squares. Suppose the desired polynomial,  $p(x)$ , is written as

$$\sum_{i=0}^m a_i x^i$$

where  $a_0, a_1, \dots, a_m$  are to be determined. The method of least squares chooses as "solutions" those coefficients minimizing the sum of the squares of the vertical distances from the data points to the presumed polynomial. This means that the polynomial termed "best" is the one whose coefficients minimize the function  $L$ , where

$$L = \sum_{i=1}^n [y_i - p(x_i)]^2$$

Here, we will treat only the linear case, where  $X = \alpha + \beta Z$ . The procedure for higher-order polynomials is identical, although the computations become much more tedious. Assume a straight line of the form  $X = \alpha + \beta Z$ . For each observation  $(x_i, z_i)$ :  $X_i = \alpha + \beta Z_i$ , let

$$Q = \sum_{i=1}^n (x_i - \alpha - \beta z_i)^2$$

We wish to find  $\alpha$  and  $\beta$  estimates such as to minimize  $Q$ . Taking the partial differentials, we obtain

$$\begin{aligned}\frac{\partial Q}{\partial \alpha} &= -2 \sum_{i=1}^n (x_i - \alpha - \beta z_i) = 0 \\ \frac{\partial Q}{\partial \beta} &= -2 \sum_{i=1}^n z_i (x_i - \alpha - \beta z_i) = 0\end{aligned}$$

Note that the above are the same as the MLE equations for normal linear regression. Therefore, we obtain the following results:

$$\begin{aligned}\hat{\alpha} &= \bar{x} - \beta \bar{z} \\ \hat{\beta} &= \frac{\sum_{i=1}^n (x_i - \bar{x})(z_i - \bar{z})}{\sum_{i=1}^n (z_i - \bar{z})^2}\end{aligned}\quad (3.54)$$

The above gives an example of least squares applied to a linear case. It follows the same pattern for higher-order curves with solutions of 3, 4, and so on, from the linear systems of equations.

### 3.7 Interval Estimation

A point estimate is sometimes inadequate in providing an estimate of an unknown parameter since it rarely coincides with the true value of the parameter. An alternative way is to obtain a confidence interval estimation of the form  $[\theta_L, \theta_U]$  where  $\theta_L$  is the lower bound and  $\theta_U$  is the upper bound.

Point estimates can become more useful if some measure of their error can be developed, *i.e.*, some sort of tolerance on their high and low values could be developed. Thus, if an interval estimator is  $[\theta_L, \theta_U]$  with a given probability  $(1-\alpha)$ , then  $\theta_L$  and  $\theta_U$  will be called 100(1- $\alpha$ )% confidence limits for the given parameter  $\theta$  and the interval between them is a 100(1- $\alpha$ )% confidence interval and  $(1-\alpha)$  is also called the confidence coefficient.

#### 3.7.1 Confidence Intervals for the Normal Parameters

The one-dimensional normal distribution has two parameters: mean  $\mu$  and variance  $\sigma^2$ . The simultaneous employment of both parameters in a confidence statement concerning percentages of the population will be discussed in the next section on tolerance limits. Hence, individual confidence statements about  $\mu$  and  $\sigma^2$  will be discussed here.

#### Confidence Limits for the Mean $\mu$ with Known $\sigma^2$

It is easy to show that the statistic

$$Z = \frac{\bar{X} - \mu}{\sigma / \sqrt{n}}$$

is a standard normal distribution where

$$\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i$$

Hence, a 100(1 -  $\alpha$ )% confidence interval for the mean  $\mu$  is given by

$$P\left[\bar{X} - Z_{\frac{\alpha}{2}} \frac{\sigma}{\sqrt{n}} < \mu < \bar{X} + Z_{\frac{\alpha}{2}} \frac{\sigma}{\sqrt{n}}\right] = 1 - \alpha \quad (3.55)$$

In other words,

$$\mu_L = \bar{X} - Z_{\frac{\alpha}{2}} \frac{\sigma}{\sqrt{n}} \quad \text{and} \quad \mu_U = \bar{X} + Z_{\frac{\alpha}{2}} \frac{\sigma}{\sqrt{n}}$$

*Example 3.12:* Draw a sample of size 4 from a normal distribution with known variance = 9, say  $x_1 = 2, x_2 = 3, x_3 = 5, x_4 = 2$ . Determine the location of the true mean ( $\mu$ ). The sample mean can be calculated as

$$\bar{X} = \frac{\sum_{i=1}^n x_i}{n} = \frac{2+3+5+2}{4} = 3$$

Assuming that  $\alpha = 0.05$  and from the standard normal distribution (Table A1.1 in Appendix 1), we obtain

$$P\left[3 - 1.96 \frac{3}{\sqrt{4}} < \mu < 3 + 1.96 \frac{3}{\sqrt{4}}\right] = 0.95$$

$$P[0.06 < \mu < 5.94] = 0.95$$

This example shows that there is a 95% probability that the true mean is somewhere between 0.06 and 5.94. Now,  $\mu$  is a fixed parameter and does not vary, so how do we interpret the probability? If the samples of size 4 are repeatedly drawn, a different set of limits would be constructed each time. With this as the case, the interval becomes the random variable and the interpretation is that, for 95% of the time, the interval so constructed will contain the true (fixed) parameter.

### ***Confidence Limits for the Mean $\mu$ with Unknown $\sigma^2$***

Let

$$S = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2} \quad (3.56)$$

It can be shown that the statistic

$$T = \frac{\bar{X} - \mu}{\frac{S}{\sqrt{n}}}$$

has a  $t$  distribution with  $(n-1)$  degrees of freedom (see Table A1.2 in Appendix 1). Thus, for a given sample mean and sample standard deviation, we obtain

$$P[|T| < t_{\frac{\alpha}{2}, n-1}] = 1 - \alpha$$

Hence, a  $100(1 - \alpha)\%$  confidence interval for the mean  $\mu$  is given by

$$P\left[\bar{X} - t_{\frac{\alpha}{2}, n-1} \frac{S}{\sqrt{n}} < \mu < \bar{X} + t_{\frac{\alpha}{2}, n-1} \frac{S}{\sqrt{n}}\right] = 1 - \alpha \quad (3.57)$$



*Example 3.13:* A problem on the variability of a new product was encountered. An experiment was run using a sample of size  $n = 25$ ; the sample mean was found to be  $\bar{X} = 50$  and the variance  $\sigma^2 = 16$ . From Table A1.2 in Appendix 1,  $t_{\frac{\alpha}{2}, n-1} = t_{.975, 24}$

$= 2.064$ . A 95% confidence limit for  $\mu$  is given by

$$P\left[50 - 2.064\sqrt{\frac{16}{25}} < \mu < 50 + 2.064\sqrt{\frac{16}{25}}\right] = 0.95$$

$$P[48.349 < \mu < 51.651] = 0.95$$

Note that, for one-sided limits, choose  $t_{\alpha}$  or  $t_{1-\alpha}$ .

### Confidence Limits on $\sigma^2$

Note that  $n \frac{\hat{\sigma}^2}{\sigma^2}$  has a  $\chi^2$  distribution with  $(n-1)$  degrees of freedom. Correcting for

the bias in  $\hat{\sigma}^2$ , then  $(n-1) \hat{\sigma}^2 / \sigma^2$  has this same distribution. Hence,

$$P\left[\chi^2_{\frac{\alpha}{2}, n-1} < \frac{(n-1)\hat{\sigma}^2}{\sigma^2} < \chi^2_{1-\frac{\alpha}{2}, n-1}\right] = 1 - \alpha$$

or

$$P\left[\frac{\sum (x_i - \bar{x})^2}{\chi^2_{1-\frac{\alpha}{2}, n-1}} < \sigma^2 < \frac{\sum (x_i - \bar{x})^2}{\chi^2_{\frac{\alpha}{2}, n-1}}\right] = 1 - \alpha \quad (3.58)$$

Similarly, for one-sided limits, choose  $\chi^2(\alpha)$  or  $\chi^2(1-\alpha)$ .

### 3.7.2 Confidence Intervals for the Exponential Parameters

The pdf and cdf of the exponential distribution are given as

$$f(x) = \lambda e^{-\lambda x} \quad x > 0, \lambda > 0$$

and

$$F(x) = 1 - e^{-\lambda x}$$

respectively. From equation (3.16), it was shown that the distribution of a function of the estimate

$$\hat{\lambda} = \frac{r}{\sum_{i=1}^n x_i + (n-r)x_r} \quad (3.59)$$

derived from a test of  $n$  identical components with common exponential failure density (failure rate  $\lambda$ ), whose testing was stopped after the  $r$ th failure, was chi-squared ( $\chi^2$ ), i.e.,

$$2r \frac{\lambda}{\hat{\lambda}} = 2\lambda T \quad (\chi^2 \text{ distribution with } 2r \text{ degrees of freedom})$$

where  $T$  is the total accrued time on all units. Knowing the distribution of  $2\lambda T$  allows us to obtain the confidence limits on the parameter as follows:

$$P\left[\chi^2_{1-\frac{\alpha}{2}, 2r} < 2\lambda T < \chi^2_{\frac{\alpha}{2}, 2r}\right] = 1 - \alpha$$

or, equivalently, that

$$P\left[\frac{\chi^2_{1-\frac{\alpha}{2}, 2r}}{2T} < \lambda < \frac{\chi^2_{\frac{\alpha}{2}, 2r}}{2T}\right] = 1 - \alpha$$

This means that in  $(1-\alpha)\%$  of samples with a given size  $n$ , the random interval

$$\left(\frac{\chi^2_{1-\frac{\alpha}{2}, 2r}}{2T}, \frac{\chi^2_{\frac{\alpha}{2}, 2r}}{2T}\right)$$

will contain the population of constant failure rate. In terms of  $\theta = 1/\lambda$  or the mean time between failure (MTBF), the above confidence limits change to

$$P\left[\frac{2T}{\chi^2_{\frac{\alpha}{2}, 2r}} < \theta < \frac{2T}{\chi^2_{1-\frac{\alpha}{2}, 2r}}\right] = 1 - \alpha$$

If testing is stopped at a fixed time rather than a fixed number of failures, the number of degrees of freedom in the lower limit increases by two. Table 3.5 shows the confidence limits for  $\theta$ , the mean of an exponential density.

*Example 3.14 (two-sided):* From the goodness of fit example,  $T = 22,850$ , testing stopped after  $r = 60$  failures. We can obtain  $\hat{\lambda} = 0.00263$  and  $\hat{\theta} = 380.833$ . Assuming that  $\alpha = 0.1$ , then, from the above formula, we obtain

$$\begin{aligned} P\left[\frac{2T}{\chi^2_{0.05, 120}} < \theta < \frac{2T}{\chi^2_{0.95, 120}}\right] &= 0.9 \\ P\left[\frac{45,700}{146.568} < \theta < \frac{45,700}{95.703}\right] &= 0.9 \\ P[311.80 < \theta < 477.52] &= 0.9 \end{aligned}$$

*Example 3.15 (one-sided lower):* Assuming that testing stopped after 1,000 hours with four failures, then

$$\begin{aligned} P\left[\frac{2T}{\chi^2_{0.10, 10}} < \theta\right] &= 0.9 \\ P\left[\frac{2,000}{15.987} < \theta\right] &= 0.9 \\ P[125.1 < \theta] &= 0.9 \end{aligned}$$

**Table 3.5.** Confidence limits for  $\theta$ 

Confidence limits	Fixed number of failures	Fixed time
One-sided lower limit	$\frac{2T}{\chi^2_{\alpha,2r}}$	$\frac{2T}{\chi^2_{\alpha,2r+2}}$
One-sided upper limit	$\frac{2T}{\chi^2_{1-\alpha,2r}}$	$\frac{2T}{\chi^2_{1-\alpha,2r}}$
Two-sided limits	$\frac{2T}{\chi^2_{\alpha/2,2r}}, \frac{2T}{\chi^2_{1-\alpha/2,2r}}$	$\frac{2T}{\chi^2_{\alpha/2,2r+2}}, \frac{2T}{\chi^2_{1-\alpha/2,2r}}$

### 3.7.3 Confidence Intervals for the Binomial Parameters

Consider a sequence of  $n$  Bernoulli trials with  $k$  successes and  $(n - k)$  failures. We now determine one-sided upper and lower and two-sided limits on the parameter  $p$ , the probability of success. For the lower limit, the binomial sum is set up such that the chance probability of  $k$  or more successes with a true  $p$  as low as  $p_L$  is only  $\alpha/2$ . This means the probability of  $k$  or more successes with a true  $p$  higher than  $p_L$  is  $\left(1 - \frac{\alpha}{2}\right)$ :

$$\sum_{i=k}^n \binom{n}{i} p_L^i (1 - p_L)^{n-i} = \frac{\alpha}{2}$$

Similarly, the binomial sum for the upper limit is

$$\sum_{i=k}^n \binom{n}{i} p_U^i (1 - p_U)^{n-i} = 1 - \frac{\alpha}{2}$$

or, equivalently, that

$$\sum_{i=0}^{k-1} \binom{n}{i} p_U^i (1 - p_U)^{n-i} = \frac{\alpha}{2}$$

Solving for  $p_L$  and  $p_U$  in the above equations,

$$P[p_L < p < p_U] = 1 - \alpha$$

For the case of one-sided limits, merely change  $\alpha/2$  to  $\alpha$ .

*Example 3.16:* Given  $n = 100$  with 25 successes, and 75 failures, an 80% two-sided confidence limits on  $p$  can be obtained as follows:

$$\begin{aligned} \sum_{i=25}^{100} \binom{100}{i} p_L^i (1 - p_L)^{100-i} &= 0.10 \\ \sum_{i=0}^{24} \binom{100}{i} p_U^i (1 - p_U)^{100-i} &= 0.10 \end{aligned}$$

Solving the above two equations simultaneously, we obtain

$$\begin{aligned} p_L &\approx 0.194 \quad \text{and} \quad p_U \approx 0.313 \\ P[0.194 < p < 0.313] &= 0.80 \end{aligned}$$

*Example 3.17:* Continuing with Example 3.16, find an 80% one-sided confidence limit on  $p$ .

We now can set the top equation to 0.20 and solve for  $p_L$ . It is easy to obtain  $p_L = 0.211$  and  $P[p > 0.211] = 0.80$ . Let us define  $\bar{p} = k/n$ , the number of successes divided by the number of trials. For large values of  $n$  and if  $np > 5$  and  $n(1 - p) > 5$ , and from the central limit theorem (Feller 1957), the statistic

$$Z = \frac{(\bar{p} - p)}{\sqrt{\frac{\bar{p}(1 - \bar{p})}{n}}}$$

approximates to the standard normal distribution. Hence,

$$P[-z_{\frac{\alpha}{2}} < Z < z_{\frac{\alpha}{2}}] = 1 - \alpha$$

Then

$$P\left[\bar{p} - z_{\frac{\alpha}{2}}\sqrt{\frac{\bar{p}(1 - \bar{p})}{n}} < p < \bar{p} + z_{\frac{\alpha}{2}}\sqrt{\frac{\bar{p}(1 - \bar{p})}{n}}\right] = 1 - \alpha$$

*Example 3.18:* Given  $n = 900$ ,  $k = 180$ , and  $\alpha = 0.05$ . Then we obtain  $p = 180/900 = 0.2$  and

$$P\left[0.2 - 1.96\sqrt{\frac{0.2(0.8)}{900}} < p < 0.2 + 1.96\sqrt{\frac{0.2(0.8)}{900}}\right] = 0.95$$

$$P[.174 < p < .226] = 0.95$$

### 3.7.4 Confidence Intervals for the Poisson Parameters

Limits for the Poisson parameters are completely analogous to the binomial except that the sample space is infinite instead of finite. The lower and upper limits can be solved simultaneously in the following equations:

$$\sum_{i=k}^{\infty} \frac{\lambda_L^i e^{-\lambda_L}}{i!} = \frac{\alpha}{2}$$

$$\sum_{i=k}^{\infty} \frac{\lambda_U^i e^{-\lambda_U}}{i!} = 1 - \frac{\alpha}{2}$$

or, equivalently

$$\sum_{i=k}^{\infty} \frac{\lambda_L^i e^{-\lambda_L}}{i!} = \frac{\alpha}{2}$$

$$\sum_{i=0}^{k-1} \frac{\lambda_U^i e^{-\lambda_U}}{i!} = \frac{\alpha}{2}$$

*Example 3.19:* One thousand article lots are inspected resulting in an average of 10 defects per lot. Find 90% limits on the average number of defects per 1000 article lots. Assume  $\alpha = 0.1$ ,

$$\sum_{i=10}^{\infty} \frac{\lambda_L^i e^{-\lambda_L}}{i!} = 0.05$$

$$\sum_{i=0}^9 \frac{\lambda_U^i e^{-\lambda_U}}{i!} = 0.05$$

Solving the above two equations simultaneously for  $\lambda_L$  and  $\lambda_U$ , we obtain

$$P[5.45 < \lambda < 16.95] = 0.90.$$

The one-sided limits are constructed similarly to the case for binomial limits.

### 3.8 Non-parametric Tolerance Limits

Non-parametric tolerance limits are based on the smallest and largest observation in the sample, designated as  $X_S$  and  $X_L$ , respectively. Due to their non-parametric nature, these limits are quite insensitive and to gain precision proportional to the parametric methods requires much larger samples. An interesting question here is to determine the sample size required to include at least  $100(1-\alpha)\%$  of the population between  $X_S$  and  $X_L$  with given probability  $\gamma$ .

For two-sided tolerance limits, if  $(1-\alpha)$  is the minimum proportion of the population contained between the largest observation  $X_L$  and smallest observation  $X_S$  with confidence  $(1-\gamma)$ , then it can be shown that

$$n(1-\alpha)^{n-1} - (n-1)(1-\alpha)^n = \gamma$$

Therefore, the number of observations required is given by

$$n = \left\lceil \frac{(2-\alpha)}{4\alpha} \chi_{1-\gamma,4}^2 + \frac{1}{2} \right\rceil + 1$$

where a value of  $\chi_{1-\gamma,4}^2$  is given in Table A1.3 of Appendix 1.

*Example 3.20:* Determine the tolerance limits which include at least 90% of the population with probability 0.95. Here,

$$\alpha = 0.1, \gamma = 0.95 \text{ and } \chi_{0.05,4}^2 = 9.488$$

and therefore, a sample of size

$$n = \left\lceil \frac{(2-0.1)}{4(0.1)} (9.488) + \frac{1}{2} \right\rceil + 1 = 46$$

is required. For a one-sided tolerance limit, the number of observations required is given by

$$n = \left\lceil \frac{\log(1-\gamma)}{\log(1-\alpha)} \right\rceil + 1$$

*Example 3.21:* As in Example 3.20, we wish to find a lower tolerance limit, that is, the number of observations required so that the probability is 0.95 that at least 90% of the population will exceed  $X_S$  is given by

$$n = \left\lceil \frac{\log(1-0.95)}{\log(1-0.1)} \right\rceil + 1 = 30$$

One can easily generate a table containing the sample size required to include a given percentage of the population between  $X_S$  and  $X_L$  with given confidence, or sample size required to include a given percentage of the population above or below  $X_S$  or  $X_L$ , respectively.

### 3.9 Sequential Sampling

A sequential sampling scheme is one in which items are drawn one at a time and the results at any stage determine if sampling or testing should stop. Thus, any sampling procedure for which the number of observations is a random variable can be regarded as sequential sampling. Sequential tests derive their name from the fact that the sample size is not determined in advance, but allowed to "float" with a decision (accept, reject, or continue test) after each trial or data point.

In general, let us consider the hypothesis

$$H_0 : f(x) = f_0(x) \quad \text{vs} \quad H_1 : f(x) = f_1(x)$$

For an observation test, say  $X_1$ , if  $X_1 \leq A$ , then we will accept the testing hypothesis ( $H_0: f(x) = f_0(x)$ ); if  $X_1 \geq A$ , then we will reject  $H_0$  and accept  $H_1: f(x) = f_1(x)$ . Otherwise, we will continue to perform at least one more test. The interval  $X_1 \leq A$  is called the acceptance region. The interval  $X_1 \geq A$  is called the rejection or critical region (see Figure A.2, Pham 2000a).

A "good" test is one that makes the  $\alpha$  and  $\beta$  errors as small as possible. However, there is not much freedom to do this without increasing the sample size. The common procedure is to fix the  $\beta$  error and then choose a critical region to minimize the error or maximize the "power" (power =  $1 - \beta$ ) of the test, or to choose the critical region so as to equalize the  $\alpha$  and  $\beta$  errors to reasonable levels.

A criterion, similar to the MLE, for constructing tests is called the "probability ratio", which is the ratio of the sample densities under  $H_1$  over  $H_0$ . Consider the ratio of probabilities

$$\lambda_m = \frac{\prod_{i=1}^n f_1(x_i)}{\prod_{i=1}^n f_0(x_i)} > k$$

Here,  $x_1, x_2, \dots, x_n$  are  $n$  independent random observations and  $k$  is chosen to give the desired error.

Recall from the MLE discussion in Section 3.2 that  $f_1(x_1), f_1(x_2), \dots, f_1(x_n)$  are maximized under  $H_1$  when the parameter(s), e.g.,  $\theta = \theta_1$  and, similarly,  $f_0(x_1), f_0(x_2),$

. . . ,  $f_0(x_n)$  are maximized when  $\theta = \theta_0$ . Thus, the ratio will become large if the sample favors  $H_1$  and will become small if the sample favors  $H_0$ . Therefore, the test will be called a sequential probability ratio test if we

1. Stop sampling and reject  $H_0$  as soon as  $\lambda_m \geq A$
2. Stop sampling and accept  $H_0$  as soon as  $\lambda_m \leq B$
3. Continue sampling as long as  $B < \lambda_m < A$ , where  $A > B$ .

The choice of  $A$  and  $B$  with the above test, suggested by Wald (1947), can be determined as follows:

$$B = \frac{\beta}{1-\alpha} \quad \text{and} \quad A = \frac{1-\beta}{\alpha}$$

The basis for  $\alpha$  and  $\beta$  are therefore

$$P[\lambda_m > A \mid H_0] = \alpha$$

$$P[\lambda_m < A \mid H_1] = \beta$$

### *Exponential Case*

Let

$$V(t) = \sum_{i=1}^r X_i + \sum_{j=1}^{n-r} t_j$$

where  $X_i$  are the times to failure and  $t_j$  are the times to test termination without failure. Thus,  $V(t)$  is merely the total operating time accrued on both successful and unsuccessful units where the total number of units is  $n$ . The hypothesis to be tested is

$$H_0 : \theta = \theta_0 \quad \text{vs} \quad H_1 : \theta = \theta_1$$

For the failed items,

$$g(x_1, x_2, \dots, x_r) = \left(\frac{1}{\theta}\right)^r e^{-\sum_{i=1}^r x_i}$$

For the non-failed items,

$$P(X_{r+1} > t_1, X_{r+2} > t_2, \dots, X_n > t_{n-r}) = e^{-\sum_{j=1}^{n-r} t_j}$$

The joint density for the first  $r$  failures among  $n$  items is

$$\begin{aligned}
 f(x_1, x_2, \dots, x_r, t_{r+1}, \dots, t_n) &= \left(\frac{1}{\theta}\right)^r e^{-\sum_{i=1}^r \frac{x_i}{\theta} - \sum_{j=1}^{n-r} \frac{t_j}{\theta}} \\
 &= \left(\frac{1}{\theta}\right)^r e^{-\frac{V(t)}{t}}
 \end{aligned}$$

and

$$\begin{aligned}
 \lambda_m &= \frac{\left(\frac{1}{\theta_1}\right)^r e^{-\frac{V(t)}{\theta_1}}}{\left(\frac{1}{\theta_0}\right)^r e^{-\frac{V(t)}{\theta_0}}} \\
 &= \left(\frac{\theta_0}{\theta_1}\right)^r e^{-V(t)\left[\frac{1}{\theta_1} - \frac{1}{\theta_0}\right]}
 \end{aligned}$$

Now, it has been shown that for sequential tests, the reject and accept limits,  $A$  and  $B$ , can be equated to simple functions of  $\alpha$  and  $\beta$ . Thus, we obtain the following test procedures:

$$\text{Continue test: } \frac{\beta}{1-\alpha} \equiv B < \lambda_m < A \equiv \frac{1-\beta}{\alpha}$$

$$\text{Reject } H_0: \quad \lambda_m > A \equiv \frac{1-\beta}{\alpha}$$

$$\text{Accept } H_0: \quad \lambda_m < B \equiv \frac{\beta}{1-\alpha}$$

Working with the continue test inequality, we now have

$$\frac{\beta}{1-\alpha} < \left(\frac{\theta_0}{\theta_1}\right)^r e^{-V(t)\left[\frac{1}{\theta_1} - \frac{1}{\theta_0}\right]} < \frac{1-\beta}{\alpha}$$

Taking natural logs of the above inequality, we obtain

$$\ln\left(\frac{\beta}{1-\alpha}\right) < r \ln\left(\frac{\theta_0}{\theta_1}\right) - V(t)\left[\frac{1}{\theta_1} - \frac{1}{\theta_0}\right] < \ln\left(\frac{1-\beta}{\alpha}\right)$$

The above inequality is linear in  $V(t)$  and  $r$ , and therefore the rejection line  $V(t)$  can be obtained by setting

$$r \ln\left(\frac{\theta_0}{\theta_1}\right) - V(t)\left[\frac{1}{\theta_1} - \frac{1}{\theta_0}\right] = \ln\left(\frac{1-\beta}{\alpha}\right)$$

or, equivalently,



$$V(t) = \frac{r \ln\left(\frac{\theta_0}{\theta_1}\right)}{\left[\frac{1}{\theta_1} - \frac{1}{\theta_0}\right]} - \frac{\ln\left(\frac{1-\beta}{\alpha}\right)}{\left[\frac{1}{\theta_1} - \frac{1}{\theta_0}\right]}$$

Similarly, the acceptance line  $V(t)$  can be obtained by setting

$$r \log\left(\frac{\theta_0}{\theta_1}\right) - V(t) \left[\frac{1}{\theta_1} - \frac{1}{\theta_0}\right] = \log\left(\frac{\beta}{1-\alpha}\right)$$

This implies that

$$V(t) = \frac{r \ln\left(\frac{\theta_0}{\theta_1}\right)}{\left[\frac{1}{\theta_1} - \frac{1}{\theta_0}\right]} - \frac{\ln\left(\frac{\beta}{1-\alpha}\right)}{\left[\frac{1}{\theta_1} - \frac{1}{\theta_0}\right]}$$

*Example 3.22:* Given that  $H_0: \theta = 500$  vs  $H_1: \theta = 250$  and  $\alpha = \beta = 0.1$ . The acceptance and rejection lines are given by

$$V(t) = 346.6r + 1098.6$$

and

$$V(t) = 346.6r - 1098.6$$

respectively. Both are linear functions in terms of  $r$ , the number of first  $r$  failures in the test.

For an exponential distribution

$\theta$	$P(A)$
0	0
$\theta_1$	$\beta$
$\log\left(\frac{\theta_0}{\theta_1}\right)$	$\log\left(\frac{1-\beta}{\alpha}\right)$
$\left[\frac{1}{\theta_1} - \frac{1}{\theta_0}\right]$	$\log\left(\frac{1-\beta}{\alpha}\right) - \log\left(\frac{\beta}{1-\alpha}\right)$
$\theta_1$	$1-\alpha$
$\infty$	1

From the information given in the above example, we can obtain

$\theta$	$P(A)$
0	0
250	0.10
346.5	0.5
500	0.90
$\infty$	1.0

Since there is no pre-assigned termination to a regular sequential test, it is customary to draw a curve called the “average sample number” (ASN). This curve shows the expected sample size as a function of the true parameter value. It is known that the test will be terminated with a finite observation. It should be noted that “on average”, the sequential tests utilizes significantly smaller samples than fixed sample plans:

$\theta$	$E(r) = \text{ASN}$
0	0
$\theta_1$	$\frac{\theta_0 \beta \log\left(\frac{\beta}{1-\alpha}\right) + (1-\beta) \log\left(\frac{1-\beta}{\alpha}\right)}{\left[\log\left(\frac{\theta_0}{\theta_1}\right) - \left(\frac{\theta_0 - \theta_1}{\theta_1}\right)\right] \theta_1}$
$\log\left(\frac{\theta_0}{\theta_1}\right)$	$\frac{\log\left(\frac{1-\beta}{\alpha}\right) \log\left(\frac{\beta}{1-\alpha}\right)}{\left[\log\left(\frac{\theta_0}{\theta_1}\right)\right]^2}$
$\frac{1}{\theta_1} - \frac{1}{\theta_0}$	$\frac{\left[(1-\alpha) \log\left(\frac{\beta}{1-\alpha}\right) + \alpha \log\left(\frac{1-\beta}{\alpha}\right)\right] \theta_1}{\left[\log\left(\frac{\theta_0}{\theta_1}\right) - \left(\frac{\theta_0 - \theta_1}{\theta_1}\right)\right] \theta_0}$
$\theta_0$	
$\infty$	0

An approximate formula for  $E(t)$ , the expected time to reach decision, is

$$E(t) \equiv \theta \log\left(\frac{n}{n - E(r)}\right)$$

where  $n$  is the total number of units on test (assuming no replacement of failed units). If replacements are made, then

$$E(t) = \frac{\theta}{n} E(r)$$

Occasionally, it is desired to “truncate” a sequential plan such that, if no decision is made before a certain point, testing is stopped and a decision is made on the basis of data acquired up to that point (Pham 2000a, page 251). There are a number of

rules and theories on optimum truncation. In the reliability community, a  $V(t)$  truncation point at  $10\theta_0$  is often used to determine the  $V(t)$  and  $r$  lines for truncation and the corresponding exact  $\alpha = \beta$  errors (these will in general be larger for truncated tests than for the non-truncated). An approximate method draws the  $V(t)$  truncation line to the center of the continue test band and constructs the  $r$  truncation line perpendicular to that point (see Figure A.7, Pham 2000a, page 252).

### *Bernoulli Case*

$$H_0: p = p_0 \text{ vs } H_1: p = p_1$$

and  $\alpha = \beta$  are pre-assigned. Then we obtain

$$\begin{aligned} \lambda_m &= \frac{\prod_{i=1}^n p_1^{x_i} (1-p_1)^{1-x_i}}{\prod_{i=1}^n p_0^{x_i} (1-p_0)^{1-x_i}} \\ &= \frac{p_1^{\sum_{i=1}^n x_i} (1-p_1)^{n-\sum_{i=1}^n x_i}}{p_0^{\sum_{i=1}^n x_i} (1-p_0)^{n-\sum_{i=1}^n x_i}} \\ &= \left( \frac{p_1}{p_0} \right)^{\sum_{i=1}^n x_i} \left( \frac{1-p_1}{1-p_0} \right)^{n-\sum_{i=1}^n x_i} \end{aligned}$$

The following inequality will be determined to continue the test region:

$$\frac{\beta}{1-\alpha} < \left( \frac{p_1}{p_0} \right)^{\sum_{i=1}^n x_i} \left( \frac{1-p_1}{1-p_0} \right)^{n-\sum_{i=1}^n x_i} < \frac{1-\beta}{\alpha}$$

Taking logs through the above inequality produces linear relationships in  $\sum x_i$  (e.g., number of failures or defects) and  $n$ , the total number of units or trials, that is,

$$\begin{aligned} &\log \left( \frac{\beta}{1-\alpha} \right) \\ &< \sum_{i=1}^n x_i \log \left( \frac{p_1}{p_0} \right) + \left( n - \sum_{i=1}^n x_i \right) \log \left( \frac{1-p_1}{1-p_0} \right) < \log \left( \frac{1-\beta}{\alpha} \right) \end{aligned}$$

Similar tests can be constructed for other distribution parameters following the same general scheme.

## 3.10 Bayesian Methods

The Bayesian approach to statistical inference is based on a theorem first presented by the Reverend Thomas Bayes. To demonstrate the approach, let  $X$  have a pdf  $f(x)$ , which is dependent on  $\theta$ . In the traditional statistical inference approach,  $\theta$  is

an unknown parameter, and hence, is a constant. We now describe our prior belief in the value of  $\theta$  by a pdf  $h(\theta)$ . This amounts to quantitatively assessing subjective judgment and should not be confused with the so-called objective probability assessment derived from the long-term frequency approach. Thus,  $\theta$  will now essentially be treated as a random variable  $\theta$  with pdf  $h(\theta)$ .

Consider a random sample  $X_1, X_2, \dots, X_n$  from  $f(x)$  and define a statistic  $Y$  as a function of this random sample. Then there exists a conditional pdf  $g(y | \theta)$  of  $Y$  for a given  $\theta$ . The joint pdf for  $y$  and  $\theta$  is

$$f(\theta, y) = h(\theta)g(y | \theta)$$

If  $\theta$  is continuous, then

$$f_1(y) = \int_{\theta} h(\theta)g(y | \theta)d\theta$$

is the marginal pdf for the statistic  $y$ . Given the information  $y$ , the conditional pdf for  $\theta$  is

$$\begin{aligned} k(\theta | y) &= \frac{h(\theta)g(y | \theta)}{f_1(y)} \quad \text{for } f_1(y) > 0 \\ &= \frac{h(\theta)g(y | \theta)}{\int_{\theta} h(\theta)g(y | \theta)d\theta} \end{aligned}$$

If  $\theta$  is discrete, then

$$f_1(y) = \sum_k P(\theta_k)P(y | \theta_k)$$

and

$$P(\theta_i | y_i) = \frac{P(\theta_k)P(y_i | \theta_i)}{\sum_k P(\theta_k)P(y_j | \theta_k)}$$

where  $P(\theta_i)$  is a prior probability of event  $\theta_i$  and  $P(\theta_j | y_j)$  is a posterior probability of event  $y_j$  given  $\theta_j$ . This is simply a form of Bayes' theorem. Here,  $h(\theta)$  is the prior pdf that expresses our belief in the value of  $\theta$  before the data ( $Y = y$ ) became available. Then  $k(\theta | y)$  is the posterior pdf of given the data ( $Y = y$ ).

Note that the change in the shape of the prior pdf  $h(\theta)$  to the posterior pdf  $k(\theta | y)$  due to the information is a result of the product of  $g(y | \theta)$  and  $h(\theta)$  because  $f_1(y)$  is simply a normalization constant for a fixed  $y$ . The idea in reliability is to take "prior" data and combine it with current data to gain a better estimate or confidence interval or test than would be possible with either singularly. As more current data is acquired, the prior data is "washed out" (Pham 2000a).

*Case 1: Binomial Confidence Limits - Uniform Prior.* Results from ten missile tests are used to form a one-sided binomial confidence interval of the form

$$P[R \geq R_L] = 1 - \alpha$$

From Subsection 3.7.3, we have

$$\sum_{i=k}^{10} \binom{10}{i} R_L^i (1-R_L)^{10-i} = \alpha$$

Choosing  $\alpha = 0.1$ , lower limits as a function of the number of missile test successes are shown in Table 3.6. Assume from previous experience that it is known that the true reliability of the missile is somewhere between 0.8 and 1.0 and furthermore that the distribution through this range is uniform. The prior density on  $R$  is then

$$g(R) = 5 \quad 0.8 < R < 1.0$$

**Table 3.6.** Lower limits as a function of the number of missile test successes

$k$	$R_L$	Exact level
10	0.79	0.905
9	0.66	0.904
8	0.55	0.900
7	0.45	0.898
6	0.35	0.905

From the current tests, results are  $k$  successes out of ten missile tests, so for the event  $A$  that contained  $k$  successes:

$$P(A | R) = \binom{10}{k} R^k (1-R)^{10-k}$$

Applying Bayes' theorem, we obtain

$$\begin{aligned} g(R | A) &= \frac{g(R)P(A | R)}{\int_R g(R)P(A | R)dR} \\ &= \frac{5 \binom{10}{k} R^k (1-R)^{10-k}}{\int_{0.8}^{1.0} 5 \binom{10}{k} R^k (1-R)^{10-k} dR} \end{aligned}$$

For the case of  $k = 10$ ,

$$\begin{aligned} g(R | A) &= \frac{R^{10}}{\int_{0.8}^{1.0} R^{10} dR} \\ &= \frac{11R^{10}}{0.914} = 12.035R^{10} \end{aligned}$$

To obtain confidence limits incorporating the “new” or current data,

$$\int_{R_L}^{1.0} g(A | R) dR = 0.9$$

$$\int_{R_L}^{1.0} 12.035 R^{10} dR = 0.9$$

After simplifications, we have

$$R_L^{11} = 0.177$$

$$R_L = 0.855$$

Limits for the 10/10, 9/10, 8/10, 7/10, and 6/10 cases employing the Bayesian method are given in Table 3.7 along with a comparison with the previously calculated limits not employing the prior assumption. Note that the lower limit of 0.8 on the prior cannot be washed out.

**Table 3.7.** Comparison between limits applying the Bayesian method and those that do not

k	$R_L$ (uniform [0.8,1] prior)	$R_L$ (no prior)	Exact level
10	0.855	0.79	0.905
9	0.822	0.66	0.904
8	0.812	0.55	0.900
7	0.807	0.45	0.898
6	0.805	0.35	0.905

*Case 2: Binomial Confidence Limits - Beta Prior.* The prior density of the beta function is

$$g(R) = \frac{(\alpha + \beta + 1)!}{\alpha! \beta!} R^\alpha (1 - R)^\beta$$

The conditional binomial density function is

$$P(A | R) = \binom{10}{i} R^i (1 - R)^{10-i}$$

Then we have

$$g(R | A) = \frac{\frac{(\alpha + \beta + 1)!}{\alpha! \beta!} R^\alpha (1 - R)^\beta \binom{10}{k} R^k (1 - R)^{10-k}}{\frac{(\alpha + \beta + 1)!}{\alpha! \beta!} \int_0^1 R^\alpha (1 - R)^\beta \binom{10}{k} R^k (1 - R)^{10-k} dR}$$

After simplifications, we obtain

$$g(R | A) = \frac{R^{\alpha+k} (1 - R)^{\beta+10-k}}{\int_0^1 R^{\alpha+k} (1 - R)^{\beta+10-k} dR}$$

Multiplying and dividing by

$$\frac{(\alpha + \beta + 11)!}{(\alpha + \beta)!(\beta + 10 - k)!}$$

puts the denominator in the form of a beta function with integration over the entire range, and hence, equal to 1. Thus,

$$g(R | A) = \binom{\alpha + \beta + 10}{\alpha + k} R^{\alpha + k} (1 - R)^{\beta + 10 - k}$$

which again is a beta density function with parameters

$$(\alpha + k) = \alpha' \quad \text{and} \quad (\beta + 10 - k) = \beta'$$

Integration over  $g(R|A)$  from  $R_L$  to 1.0 with an integral set to  $1 - \alpha$  and a solution of  $R_L$  will produce  $100(1 - \alpha)\%$  lower confidence bounds on  $R$ , that is,

$$\int_{R_L}^{1.0} g(R | A) dR = 1 - \alpha$$

*Case 3: Exponential Confidence Limits - Gamma Prior.* For this situation, assume interest is in an upper limit on the exponential parameter  $X$ . The desired statement is of the form

$$p[\lambda < \lambda_U] = 1 - \alpha$$

If 1000 hours of test time was accrued with one failure, a 90% upper confidence limit on  $\lambda$  would be

$$p[\lambda < 0.0039] = 0.9$$

From a study of prior data on the device, assume that  $\lambda$  has a gamma prior density of the form

$$g(\lambda) = \frac{\lambda^{n-1} e^{-\frac{\lambda}{\beta}}}{(n-1)! \beta^n}$$

With an exponential failure time assumption, the current data in terms of hours of test and failures can be expressed as a Poisson, thus,

$$p(A | \lambda) = \frac{(\lambda T)^n e^{-\lambda T}}{r!}$$

where  $n$  = number of failures

$T$  = test time, and

$A$  = event which is  $r$  failures in  $T$  hours of test.

Applying Bayes' results, we have

$$\begin{aligned}
 g(\lambda | A) &= \frac{\frac{\lambda^{n-1} e^{-\frac{\lambda}{\beta}}}{(n-1)! \beta^n} \frac{(\lambda T)^r e^{-\lambda T}}{r!}}{\int_{\lambda=0}^{\infty} \frac{\lambda^{n-1} e^{-\frac{\lambda}{\beta}}}{(n-1)! \beta^n} \frac{(\lambda T)^r e^{-\lambda T}}{r!} d\lambda} \\
 &= \frac{\lambda^{n+r-1} e^{-\lambda(\frac{1}{\beta}+T)}}{\int_0^{\infty} \lambda^{n+r-1} e^{-\lambda(\frac{1}{\beta}+T)} d\lambda}
 \end{aligned}$$

Note that

$$\int_0^{\infty} \lambda^{n+r-1} e^{-\lambda(\frac{1}{\beta}+T)} d\lambda = \frac{(n+r-1)!}{\left(\frac{1}{\beta}+T\right)^{n+r}}$$

Hence,

$$g(\lambda | A) = \frac{\lambda^{n+r-1} e^{-\lambda(\frac{1}{\beta}+T)} \left(\frac{1}{\beta}+T\right)^{n+r}}{(n+r-1)!}$$

Thus,  $g(\lambda|A)$  is also a gamma density with parameters  $(n+r-1)$  and  $\frac{1}{\left(\frac{1}{\beta}+T\right)}$ . This

density can be transformed to the  $\chi^2$  density with  $2(n+r)$  degree of freedom by the following change of variable. Let

$$\lambda' = 2\lambda \left(\frac{1}{\beta}+T\right)$$

then

$$d\lambda = \frac{1}{2} \left(\frac{1}{\frac{1}{\beta}+T}\right) d\lambda'$$

We have

$$h(\lambda' | A) = \frac{(\lambda')^{\frac{2(n+r)}{2}-1} e^{-\frac{\lambda'}{2}}}{\left[\frac{2(n+r)}{2}-1\right]! 2^{\frac{2(n+r)}{2}}}$$



To obtain a  $100(1-\alpha)\%$  upper confidence limit on  $\lambda$ , solve for  $\lambda'$  in the integral

$$\int_0^{\lambda'} h(s | A) ds = 1 - \alpha$$

and convert  $\lambda'$  to  $\lambda$  via the above transformation.

*Example 3.23:* Given a gamma prior with  $n = 2$  and  $\beta = 0.0001$  and current data as before (*i.e.*, 1000 hours of test with one failure), the posterior density becomes

$$g(\lambda | A) = \frac{\lambda^2 e^{-\lambda(11,000)} (11,000)^2}{2}$$

converting to  $\chi^2$  via the transformation  $\lambda' = 2\lambda(11000)$  and

$$h(\lambda' | A) = \frac{(\lambda')^{\frac{6}{2}-1} e^{-\frac{\lambda'}{2}}}{\left[\frac{6}{2}-1\right]! 2^{\frac{6}{2}}}$$

which is  $\chi^2$  with six degrees of freedom. Choosing  $\alpha = 0.1$  then

$$\chi_{6,1-\alpha}^2 = \chi_{6,0.9}^2 = 10.6$$

and

$$p[\lambda' < 10.6] = 0.9$$

But  $\lambda' = 2\lambda(11,000)$ , hence,

$$p[\lambda' = 2\lambda(11,000) < 10.6] = 0.9$$

or

$$p[\lambda < 0.0005] = 0.9$$

The latter limit conforms to 0.0039 derived without the use of a prior density, *i.e.*, an approximate eight fold improvement. The examples above involved the development of tighter confidence limits where a prior density of the parameter could be utilized.

In general, for legitimate applications and where prior data are available, employment of Bayesian methods can reduce cost or give results with less risk for the same dollar value (Pham 2000a).

### 3.11 Further Reading

The reader interested in a deeper understanding of advanced statistical inference and theory should note the following highly recommended books:

Introduction to Statistics:

P.S. Mann, *Introductory Statistics*, Wiley, 2004

J.L. Devore, *Probability and Statistics for Engineering and the Sciences*,  
3rd edition, Brooks/Cole Pub. Co., Pacific Grove, 1991

Life Data Analysis:

W.B. Nelson, *Applied Life Data Analysis*, Wiley, 2004

For Censored data:

R.J.A. Little and D.B. Rubin, *Statistical Analysis with Missing Data*,  
Wiley, 2002

For Bayesian:

J. M. Bernardo and A.F. M. Smith, *Bayesian Theory*, Wiley, 2000

W. M. Bolstad, *Introduction to Bayesian Statistics*, Wiley, 2004

### 3.12 Problems

1. Let  $X_1, X_2, \dots, X_n$  represent a random sample from the Poisson distribution having pdf

$$f(x; \lambda) = \frac{e^{-\lambda} \lambda^x}{x!} \quad \text{for } x = 0, 1, 2, \dots \text{ and } \lambda \geq 0$$

Find the maximum likelihood estimator  $\hat{\lambda}$  of  $\lambda$ .

2. Let  $X_1, X_2, \dots, X_n$  be a random sample from the distribution with a discrete pdf

$$P(x) = p^x (1-p)^{1-x} \quad x = 0, 1 \quad \text{and } 0 < p < 1$$

Find the maximum likelihood estimator  $\hat{p}$  of  $p$ .

3. Assume that  $X_1, X_2, \dots, X_n$  represent a random sample from the Pareto distribution, that is,

$$F(x; \lambda, \theta) = 1 - \left( \frac{\lambda}{x} \right)^\theta \quad \text{for } x \geq \lambda, \lambda > 0, \theta > 0$$

This distribution is commonly used as a model to study incomes. Find the maximum likelihood estimators of  $\lambda$  and  $\theta$ .

4. Let  $Y_1 < Y_2 < \dots < Y_n$  be the order statistics of a random sample  $X_1, X_2, \dots, X_n$  from the distribution with pdf

$$f(x; \theta) = 1 \quad \text{if } \theta - \frac{1}{2} \leq x \leq \theta + \frac{1}{2}, \quad -\infty < \theta < \infty$$

Show that any statistic  $h(X_1, X_2, \dots, X_n)$  such that

$$Y_n - \frac{1}{2} \leq h(X_1, X_2, \dots, X_n) \leq Y_1 + \frac{1}{2}$$

is a maximum likelihood estimator of  $\theta$ . What can you say about the following functions?

(a)  $\frac{(4Y_1 + 2Y_n + 1)}{6}$

(b)  $\frac{(Y_1 + Y_n)}{2}$

(c)  $\frac{(2Y_1 + 4Y_n - 1)}{6}$

5. The lifetime of transistors is assumed to have an exponential distribution with pdf

$$f(t; \theta) = \frac{1}{\theta} e^{-\frac{t}{\theta}} \quad \text{for } t \geq 0, \theta > 0$$

A random sample of size  $n$  is observed. Determine the following:

- (a) The maximum likelihood estimator of  $\theta$ .  
 (b) The MLE of the transistor reliability function,  $\hat{R}(t)$ , of

$$R(t) = e^{-\frac{t}{\theta}}$$

## Software Development Lifecycle and Data Analysis

### 4.1 Introduction

As software becomes increasingly important in systems that perform complex and critical functions, *e.g.*, military defense, nuclear reactors, so too have the risks and impacts of software-caused failures. There is now general agreement on the need to increase software reliability and quality by eliminating errors created during software development. Industry and academic institutions have responded to this need by improving developmental methods in the technology known as software engineering and by employing systematic checks to detect software errors during and in parallel with the developmental process.

Many organizations make the reduction of defects their first quality goal. The consumer electronics business, however, pursues a different goal: maintaining the number of defects in the field at zero. When electronic products leave the showroom, their destination is unknown. Therefore, detecting and correcting a serious software defect would entail recalling hundreds of thousands of products.

In the past 35 years, hundreds of research papers have been published in the areas of software quality, software engineering development process, software reliability modeling, software independent verification and validation (IV&V) and software fault tolerance. Software engineering is evolving from an art to a practical engineering discipline (Lyu 1996).

A large number of analytical models have been proposed and studied over the last two decades for assessing the quality of a software system. Each model must make some assumptions about the development process and test environment. The environment can change depending on the software application, the lifecycle development process as well as the capabilities of the engineering design team (Malaiya 1990). Therefore, it is important for software users and practitioners to be familiar with all the relevant models in order to make informed decisions about the quality of any software product.

This chapter provides the basic concepts of software engineering assessment including software lifecycle, software development process and its applications, software verification and validation, and data collection and analysis.

## 4.2 Software vs Hardware Reliability

The development of hardware reliability theory has a long history and was established to improve hardware reliability greatly while the size and complexity of software applications have increased (Xie 1991). Hardware reliability encompasses a wide spectrum of analyses that strive systematically to reduce or eliminate system failures which adversely affect product performance. Reliability also provides the basic approach for assessing safety and risk analysis.

Software reliability strives systematically to reduce or eliminate system failures which adversely affect performance of a software program. Software systems do not degrade over time unless modified. There are many differences between the reliability and testing concepts and techniques of hardware and software. Therefore, a comparison of software and hardware reliability would be useful in developing software reliability modeling.

Table 4.1 shows the differences and similarities between the two. Pham (2000a, Figure 3.1) shows in more detail the sequence of failure either by the hardware or software. The result is that software quality and reliability must be built into software during the developmental process.

*Example 4.1: Chilled Water System (Pham 2000a).* The chilled water system acts as a heat sink for an air-conditioning system and the electronics cooling system. It provides a supply of chilled water to the individual air-handling units (AHUs) and to the suction of the electronics cooling water system pumps. More details about the system configuration and description can be obtained in Pham (2000a). A simplified diagram of the system and its corresponding reliability block diagram can be obtained in Pham (2000a, Figures 3.2(a) and 3.2(b), respectively).

The chilled water system consists of two chillers and two chilled water pumps that are used to circulate chilled water to the various chilled water loads. The system success criteria are based on supplying chilled water to the operations building AHUs. There are five AHUs in the operations building: two main units supply cool air to the entire building and three critical units supply cooling to three cooling zones in the critical operations area.

Successful operation was designed as follows: one of the two chillers operating, and one of the two chilled water pumps running, supplying chilled water to the operations building (Pham 2000a). Additionally, one of the two main operations building AHUs must be operating along with two of the three critical area AHUs. Its success criteria are based on the successful operation of one of the chillers and one of the chilled water pumps. It is also assumed that success requires one of the main operations building AHUs and two of the three critical area AHUs.

**Table 4.1.** Software reliability vs hardware reliability

Software reliability	Hardware reliability
Without considering program evolution, failure rate is statistically non-increasing	Failure rate has a bathtub curve. The burn-in state is similar to the software debugging state
Failures never occur if the software is not used	Material deterioration can cause failures even though the system is not used
Most models are analytically derived from assumptions. Emphasis is on developing the model, the interpretation of the model assumptions, and the physical meaning of the parameters	Failure data are fitted to some distributions. The selection of the underlying distribution is based on the analysis of failure data and experiences. Emphasis is placed on analyzing failure data
Failures are caused by incorrect logic, incorrect statements, or incorrect input data. This is similar to design errors of a complex hardware system	Failures are caused by material deterioration, random failures, design errors, misuse, and environment
Software reliability can be improved by increasing the testing effort and by correcting detected faults. Reliability tends to change continuously during testing due to the addition of problems in new code or to the removal of problems by debugging errors	Hardware reliability can be improved by better design, better material, applying redundancy and accelerated life testing
Software repairs establish a new piece of software	Hardware repairs restore the original condition
Software failures are rarely preceded by warnings	Hardware failures are usually preceded by warnings
Software components have rarely been standardized	Hardware components can be standardized
Software essentially requires infinite testing	Hardware can usually be tested exhaustively

All the water pumps, the chillers, and AHUs can be represented by exponential distributions with failure rates  $\lambda_p$ ,  $\lambda_c$ , and  $\lambda_a$ , respectively. Assume that

$$\lambda_p = 0.001 / \text{hour}$$

$$\lambda_c = 0.0005 / \text{hour}$$

$$\lambda_a = 0.0001 / \text{hour}$$

Assume a mission of  $t = 72$  hours, then

1. The reliability of the water pumps subsystem is

$$\begin{aligned} R_p &= 1 - (1 - e^{-(0.001)(72)})^2 \\ &= 0.99517 \end{aligned}$$

2. The reliability of the chillers' subsystem is

$$\begin{aligned} R_c &= 1 - (1 - e^{-(0.0005)(72)})^2 \\ &= 0.99875 \end{aligned}$$

3. The reliability of the AHU subsystem I is

$$\begin{aligned} R_{Ia} &= 1 - (1 - e^{-(0.0001)(72)})^2 \\ &= 0.99995 \end{aligned}$$

4. The reliability of the AHU subsystem II is

$$\begin{aligned} R_{IIa} &= \sum_{i=2}^3 \binom{3}{i} (e^{-(0.0001)(72)})^i (1 - e^{-(0.0001)(72)})^{3-i} \\ &= 0.99985 \end{aligned}$$

Assuming all the software systems do not fail, the overall chilled water system reliability is

$$\begin{aligned} R_s &= R_p \times R_c \times R_{Ia} \times R_{IIa} \\ &= 0.99373 \end{aligned} \quad (4.1)$$

The software, however, does fail. The water pumps, chillers, and AHU software each contain approximately 250,000 lines of source code in ground control and processing to operate innumerable hardware units. One observes that the software reliabilities of the water pumps, chillers, and AHUs for a 72-hour mission are

$$P_p = 0.97, P_c = 0.99, \text{ and } P_a = 0.995$$

respectively. Therefore, the reliability of the overall chilled water hardware software system becomes

$$\begin{aligned} R_s &= (1 - [1 - (0.97)e^{-(0.001)(72)}]^2) \\ &\quad (1 - [1 - (0.99)e^{-(0.0005)(72)}]^2)(0.995)R_{Ia} \cdot R_{IIa} \\ &= 0.983354 \end{aligned}$$

which is far less than the reliability result in equation (4.1).

### 4.3 Software Reliability and Testing Concepts

Software is essentially an instrument for transforming a discrete set of inputs into a discrete set of outputs. It comprises a set of coded statements or instructions whose functions may be to evaluate an expression and store the result in a temporary or

permanent location, to decide which statement to execute, or to perform input/output operations (Goel 1985). Hence, software can be regarded as a function  $f$ , mapping the input space to the output space ( $f$ : input  $\rightarrow$  output), where the input space is the set of all input states and the output space is the set of all output states (see Pham 2000a, Figure 3.3).

Software reliability is the probability that given software functions without failure in a given environmental condition during a specified time. Another deterministic model defines software reliability as the probability of successful execution(s) of an input state randomly selected from the input space under specified operating conditions. Another definition is the probability of failure-free execution of the software for a specified time in a specified environment. For example, an operating system with a reliability of 95% for 8 hours for an average user should work 95 out of 100 periods of 8 hours without any problems. Software failure means the inability to perform an intended task specified by a requirement. A software fault (or bug) is an error in the program source-text, which causes software failure when the program is executed under certain conditions. Hence, a software fault is generated when a mistake is made.

In vehicular applications, for instance, errors can be divided into four categories: critical, high, moderate and low:

- Critical: It may affect a federally mandated item. (A change is required immediately, *i.e.*, brake system.)
- High: It may affect a necessary function of vehicle operation or subsystem, *i.e.*, inoperative engine, air-conditioning, locks, *etc.* (Potential customer satisfaction item, a change is required immediately.)
- Moderate: It may affect a convenience feature *i.e.*, chimes, cruise control, compass, mini-trip computer, head lamp delay, *etc.* (Change at the next opportunity.)
- Low: It is unreasonable to expect that the minor nature of this item would have any real effect on the vehicle or system performance. (No change required at this time.)

In general, the definition of what constitutes a software failure is an area open for debate since it depends on the application. When a program crashes, it has obviously failed due to an error in design, specifications, coding, or testing. One can define a failure as not meeting the user's requirements or expectations of the software operation. This can be due to a number of criteria which are not always well-defined. An example is that the speed of execution, accuracy of the computations, *etc.*, can be the criteria for failure of the software.

Software testing is the process of executing a program to locate an error. A good test case is one that has a high probability of finding undiscovered error(s). It is impossible to continue testing the software until all faults are detected and removed as testing of all possible inputs would require millions of years! Therefore, failure probabilities must be inferred from testing a sample of all possible input states called the *input space*. In other words, input space is the set of all possible input states. Similarly, output space is the set of all possible output states for a given software and input space.

It is generally very difficult to test exhaustively a large computer program because of problems with dimensionality. If the input space consists of a single



unbounded variable, then an infinite number of input cases will be needed to provide an exhaustive test of the program. If the input space is bounded, but contains a large number of independent variables, then the number of input cases needed for an exhaustive test will tend to be impossibly large, even if one accepts the use of discretization for each input variable.

Similarly, it will probably be an impossible task to test each pathway through the program because of the very large number of paths involved. For instance, the flow graph of a very small program (see Pham 2000a, Figure 3.4) shows the schematics of a fairly small program, with a number of DO loops and IF branches, with 1018 unique paths through it. Even though exhaustive software testing may not be feasible for a very large software system, it makes sense to carry out a series of tests on each functional area. In the context of the user relationship, the user should obviously stipulate the tests to be carried out and be actively involved in their execution (Churchley 1991).

As we know, different inputs have different chances of being selected, and we can never be sure which inputs are selected in the operational phase of real world applications. During the operational phase, some input states are executed more frequently than others and a probability can be assigned to each input state to form the operational profile of the program. This operational profile can be used to construct the software reliability model. This type of model is also called an *input-domain model*. The interesting questions here are: (1) Is it possible to determine the sizes and locations of the fault regions in the input space? (2) How do we determine the reliability that a program will execute correctly for a particular length of time?

To select a good software model in order to make an accurate reliability prediction, the testing strategy should be incorporated into the software reliability model. It should be noted that the best models may vary from time to time and differ from application to application.

The evaluation of software reliability cannot be performed without software failure data. Therefore, the establishment of a software failure database is useful to both practitioners and researchers for predicting and estimating software reliability and to determine the total testing time needed to reach a desired reliability goal. Collected data are grouped into four categories: component data, management data, dynamic failure data, and fault removal data, each with a unique set of information.

The information in the component data category contains the number of executable source lines of code, the total number of comments and instructions, and the source language used for each system component. The information in the management data category is the starting and ending date for each lifecycle phase (analysis, design, code, test, and operation), the definitions and requirements of each lifecycle phase, and the models used for estimating software reliability.

The information in the dynamic failure data category is the number of CPU hours since the last failure, the number of test cases executed since the last failure, the severity of the failure, the method of failure detection, and the unit complexity and size where the fault was detected. The information in the fault removal data category is the date and time of fixing an error, the CPU hours required to fix an error, and the labor hours required to fix an error for each failure corrected. The data collection addresses these questions:

1. Are the defects discovered as a result of simply testing artifacts of prior modifications or are they previously undetected defects?
2. What taxonomic categories are required for defense, aerospace, military, and commercial systems, and what defect percentages reside in each category?

### 4.4 Software Lifecycle

A software lifecycle provides a systematic approach to developing, using, operating, and maintaining a software system. The standard IEEE computer dictionary has defined the software lifecycle as: "That period of time in which the software is conceived, developed and used". There are many different definitions of software lifecycle (Boehm 1981; Pressman 1983).

A software lifecycle consists of the following five successive phases (also see Figure 3.5 in Pham 2000a for details):

1. Analysis (requirements and functional specifications)
2. Design
3. Coding
4. Testing
5. Operation

The detailed activities of each phase are given in Pham (2000a) (see Figures 3.5-3.11). Table 4.2 shows the errors introduced and errors detected in the software lifecycle of a commercial application. In the early phases of the software lifecycle, a predictive software model is needed because no failure data are available. This type of model predicts the number of initial faults in the software before testing. In the testing phase, the software reliability can be used to improve through perfect debugging.

**Table 4.2.** Software error introduction and discovery

Lifecycle phase	Errors introduced (%)	Errors detected (%)
Analysis	55	18
Design	30	10
Coding and testing	10	50
Operations	5	22

By assuming perfect debugging, *i.e.*, a fault is removed with certainty whenever a failure occurs, the number of remaining faults is a decreasing function of debug-ging time. With an imperfect debugging assumption, *i.e.*, faults may or may not be removed, introduced, or changed at each debugging, the number of remaining faults may increase or decrease.

A reliability growth model is needed to estimate the current reliability level and the time and resources required to achieve the desired reliability goal. During this phase, reliability estimation is based on the analysis of failure data. After the release of a software program, the addition of new modules, removal of old ones, removal of detected errors, mixing of newly and previously written code, change of

user environment, and change of hardware and management involvement have to be considered in the evaluation of software reliability. An evolution model is thus needed.

### **Analysis Phase**

The analysis phase is the first step in the software development process and also the most important phase in the whole process and the foundation of building a successful software product (Pham 2000a). A survey at the North Jersey Software Process Improvement Network workshop in August 1995 showed that about 35% of the effort in software development projects should be concentrated in the analysis phase (Pham 1999a).

The purpose of the analysis phase (Figure 3.6 in Pham 2000a) is to define the requirements and provide specifications for the subsequent phases and activities. The analysis phase is composed of three major activities: problem definition, requirements, and specifications. Problem definition develops the problem statement and the scope of the project. It is important to understand what the user's problem is and why the user needs a software product to solve the problem. This is determined by the frequent interactions with customers. A well-defined problem and its scope can help focus further development activities.

The requirement activity consists of collecting and analyzing requirements. Requirement collection includes product capabilities and constraints. Product capabilities qualitatively describe how well the system will perform. To obtain the qualitative requirements, we need to collect information on product functionality, usability, intended use, and future expectations. Usability refers to system reliability, performance, security, and human factors. Intended use describes the generality of a solution and how and where the system will be operated and who will use it. Future expectation describes the ease of adapting the product for new uses and how easy it is to keep the software in operation when there is a need to modify the code. Constraints are another important part of requirements collection. Schedule and resources are the two major constraints in requirements. There is a user schedule requirement. Resources refer to the limitations on the user side during development and operation of the software. These limitations can be computer and peripheral equipment, staff availability to operate and maintain the software, management support, and the cost for development and operation.

Requirement analysis includes a feasibility study and documentation. Based on the collected user requirements, further analysis is needed to determine if the requirements are feasible. A feasibility study includes cost estimation, benefit estimation, schedule and risk analysis. The documentation for requirements is the project plan, which is the foundation document of the entire project. It contains a proposal for the product itself, a description of the environment in which it is to be used, and development plans. These plans indicate the schedule, budget, and procedures of the project.

The next activity in the analysis phase is specifications, which is transforming the user-oriented requirements into a precise form oriented to the needs of software engineers. There are three major activities in the specification process: detailed aspects, documentation, and validation. The focus points of the detailed aspects are functionality, technical feasibility, and quality.

Functionality refers to how to process the input information into expected results, and how the software interacts with other systems in the user environment. Technical feasibility is to examine the possibility of implementing the given functionalities, and the need of technical support, *e.g.*, equipment and people. Quality measurement is needed to achieve the quality standard required by users. To check the quality standard, we need to focus on reliability, performance, security, and human factors.

Based on the project plan, the specification document provides technical details. It is written for the software team in a technical language. It is necessary to let the user review the specifications to ensure the proposed product is what the user wants. Prototyping can also be used for validation of the specifications. This allows the user and the software team to see the software in action and to find aspects that do not meet the requirements.

The importance of the analysis phase has been strongly reinforced in recent software development. A well-developed specification can reduce the incidence of faults in the software and minimize rework. Research indicates that increased effort and care during specification will generate significant rewards in terms of dependability, maintainability, productivity, and general software quality.

### **Design Phase**

The design phase is concerned with building the system to perform as required. There are two stages of design: system architecture design and detailed design (see Figure 3.7 in Pham 2000a). The system architecture design includes system structure and the system architecture document. System structure design is the process of partitioning a software system into smaller parts. Before subdividing the system, we need to do further specification analysis, examine the details of performance requirements, security requirements, assumptions and constraints, and the need for hardware and software.

System decomposition includes subsystem process control and interface relationship. Besides determining how to control the process of each subsystem by identifying major modules, the internal and external interfaces need to be defined. Internal interface refers to how the subsystems interact with each other. External interface defines how the software interacts with its environment, *e.g.*, user, operation, other software and hardware. The last activity in system architecture design is to initiate a system architecture document, which is part of the design document in the design phase. The system architecture document describes system components, subsystems and their interfaces.

Detailed design is about designing the program and algorithmic details. The activities within detailed design are program structure, program language and tools, validation and verification, test planning, and design documentation. Program structure optimizes the design of selecting data structures and algorithms to achieve the goal of the project. Structure quality measurement checks if the selected program structure meets the quality requirements.

The four major measurements are functionality, usability, intended use, and future expectations. To comply with the functionality requirements means that the designed algorithm should implement the functional characteristics of the proposed

system. Usability consists of reliability, performance, security, and human factors requirements.

There are two important parts to provide reliability: fault detection and fault isolation. The design has to consider both aspects. Since performance requirements influence the selection of data structures and algorithms, it is important to check performance factors at the design phase. To estimate the performance of the design, the information on usage pattern, design structure, and installation characteristics are needed. The specifications describe the level and what security looks like while design considers its implementation.

Different types of systems may have their own particular security needs. There is a series of issues that need to be evaluated for system security: information must be kept confidential, unauthorized modification of information must be prohibited, and unauthorized withholding of information must be avoided. To check if human factors meet the requirements, the designed user interfaces and support tools are two issues to be considered. Intended use is to measure if the designed algorithms meet the stated requirements. Future expectations focus on adaptability and maintainability of the designed system. During detailed design, the selected data structures and algorithms are implemented in a particular programming language on a particular machine. Thus, choosing the appropriate program language and tools is essential.

Test plans should be initiated at design phase. These include identifying items to be tested, creating test case specifications, and generalizing the test approach. A design document is the deliverable of the design phase. It describes system architecture, module design, data object design, how quality expectations will be achieved, and the test plan.

### **Coding Phase**

Coding involves translating the design into the code of a programming language, beginning when the design document is baselined. Coding comprises of the following activities: identifying reusable modules, code editing, code inspection, and final test planning (see Figure 3.8 in Pham 2000a). Identifying reusable modules is an effective way to save time and effort. Before writing the code, there may be existing code for modules of other systems or projects which is similar to the current system. These models can be reused with modification. When writing the code, developers should adopt good program styles.

A good program style is characterized by simplicity, readability, good documentation, changeability, and module independence. Generally, programming standards should be followed to ensure that the written programs are easily understood by all project team members. Writing structured code also helps to make the program easy to read and maintain. When modifying reusable code, the impact of reusable modules and the interfaces with other modules need to be considered.

Code inspection includes code reviews, quality, and maintainability. Code reviews is to check program logic and readability. This is normally conducted by other developers on the same project team and not the author of the program. Quality verification ensures that all the modules perform the functionality as described in the detailed design. Quality check focuses on reliability, performance,

and security Maintainability is also checked to ensure the programs are easy to maintain. The final test plan should be ready at the coding phase. Based on the test plan initiated at the design phase, with the feedback of coding activities, the final test plan should provide details of what is to be tested, testing strategies and methods, testing schedules, and all necessary resources.

### **Testing Phase**

Testing is the verification and validation activity for the software product. The goals of the testing phase are (1) to affirm the quality of the product by finding and eliminating faults in the program, (2) to demonstrate the presence of all specified functionality in the product, and (3) to estimate the operational reliability of the software. During the testing phase, program components are combined into the overall software code and testing is performed according to a developed test (Software Verification and Validation) plan.

System integration of the software components and system acceptance tests are performed against the requirements. In other words, testing during this phase determines whether all requirements have been satisfied and is performed in accordance with the reviewed software verification and validation plan. Test results are evaluated and test and verification reports are prepared to describe the outcome of the process. The testing phase (Figure 3.9 in Pham 2000a) consists of a unit test, integration test, and acceptance test.

The unit test is the process of taking a program module and running it in isolation from the rest of the software product by using prepared inputs and comparing the actual results with the results predicted by the specifications and design of the module. The unit test is the responsibility of programmers while the later stages of testing may be done by an independent testing group.

The integration test includes subsystem and system tests. The subsystem test focuses on testing the interfaces and interdependencies of subsystems or modules. The system test examines all the subsystems as a whole to determine whether specified functionality is performed correctly as the results of the software. The integration test also includes the system integration testing process which brings together all system components, hardware and software, and humanware. This testing is conducted to ensure that system requirements in real or simulated system environments are satisfied.

The acceptance test acts as a validation of the testing phase, consisting of an internal test and field test. The internal test includes a capability test and guest test, both performed in-house. The capability test examines the system in an environment configured similar to the customer environment. The guest test is conducted by the users in their software organization sites.

The field test is to install the product in a user environment and allows the user to test the product where customers often lead the test and define and develop the test cases. The field test is also called the "beta test" The acceptance test is defined as formal testing conducted to determine whether a software system satisfies its acceptance criteria and to enable the customer to determine whether the system is acceptable.

When the developer's testing and system installation have been completed, acceptance testing that leads to ultimate certification begins. It is recommended that acceptance testing be performed by an independent group to ensure that the software meets all requirements. Testing by an independent group (without the developers' preconceptions about the functioning of the system) provides assurance that the system satisfies the intent of the original requirements. The acceptance test group usually consists of analysts who will use the system and members of the requirements definition group. Concurrent with all of the previous phases is the preparation of the user and maintenance manuals. The probability of fixing a known error incorrectly also increases rapidly during the latter stages. This phenomenon is of interest because an incorrect fix to a problem often causes more harm than the original problem. This may lead to an important question: Is there a good reason not to correct a software error deliberately?

### **Operating Phase**

The final phase in the software lifecycle is operation. The operating phase (see Figure 3.11 in Pham 2000a) usually incorporates activities such as installation, training, support, and maintenance. After completion of the testing phase, the turnover of the software product plays a very small but important role of the life cycle. It transfers responsibility for software maintenance from the developer to the user by installing the software product.

The user is then responsible for establishing a program to control and manage the software. Installation may include system installation and providing the installation manual to the users. The training includes user training and operating staff training. User training is based primarily on major system functions and the user's need for access to them so that they understand what the functions are and how to perform them. Documentation support provides the customer with the user, reference, and system manuals. The software product is thus accepted for operational use.

Maintenance is defined as any change made to the software, either to correct a deficiency in performance (as required by the original software requirements specification), to compensate for environmental changes, or to enhance its operation. Maintenance activities include system improvement and replacement strategies. There are four types of activities in system improvement: correction of errors, adaptation to other changes, perfection of acceptable functions, and prevention of future errors. System improvement activities are similar to those of previous phases of development: analysis, design, coding, and testing.

## **4.5 Software Development Process and Its Applications**

In this section, we will discuss the details of applying the generalized analytic hierarchy process (AHP) (Lee 1999) to the software development process.

#### 4.5.1 Analytic Hierarchy Process

The AHP is a comprehensive mathematical framework for priority setting in a complex system. The AHP has been applied in a variety of areas since it was first developed by Saaty (1980) in the 1970s. According to Saaty, a complex system is decomposed into subsystems and represented in the hierarchical form. The element at the highest level is called the goal.

The elements at each level are the criteria of the elements at the level below. The elements at the bottom level are called the alternatives. In this way, the AHP organizes the basic rationality of the priority setting process by breaking down a multi-element complex system into its smaller constituent parts called components (or levels). The process setting can be divided into three phases: system structuring, pairwise comparison and priorities synthesis. The generalization of the AHP to systems with feedback, *i.e.*, system with both inter- and intra-component dependence, is given in (Lee 1999).

#### 4.5.2 Evaluation of Software Development Process

Software has a lifecycle, which goes through the periods of initiation, growth, maturity, and phase-out. The development of software goes through a number of phases or stages. There are several ways to structure software lifecycle into phases according to different activities during software development. After analyzing the software development process, we construct the software lifecycle into five phases: analysis, design, coding, testing, and operations.

The development phases overlap and feed information to each other. Each phase can be decomposed further to show the detailed activities under that phase (see Section 4.4). The results of a recent study on software development and its environmental factors show that analysis, design, coding, and testing take about 25, 18, 36, and 21% of the whole software development efforts, respectively. In this application, the hierarchy structure for the design phase is discussed in this section. The same procedure can be applied to analyze the impacts among the activities to decompose the system into components and to construct the weight diagrams.

Pham (2000a) (Figures 3.12 and 3.13) shows details of the hierarchy structure of the design phase and the impact diagram of the elements in the design phase using the techniques developed by Lee (1999).

The overall priorities of the activities in the design phase are summarized in Pham (2000a, Figure 3.14). At the top-most level, we see that the weight of detailed design is approximately two to three times that of system architecture design. This is consistent with our intuition because there are more activities under detailed design and more resources are needed for these activities. Here we want to emphasize that the generalized AHP (Lee 1999) not only provides qualitative information consistent with intuition, but also quantitative information which is very instrumental in resource allocation. In the next level we see that the results obtained are consistent with the intuition. In system architecture design the resource allocated to system structure is about four times that of system architecture document. This should be evident from the amount of activities involved in system structure.



Similarly, in detailed design, almost half of the resources should be allocated to program structure since it is the major activity at this level. To improve the design progress, we need to focus more on detailed design, especially on program structure. Other activities are also important, and resources allocated to these activities should also be proportional to their final weights.

Based on the results of this application using the generalized AHP, software developers can plan their schedules according to the relative weights within the design time frame. The weight diagram is useful in identifying the most important activities while conducting multiple tasks. For instance, sufficient time and effort should be spent on specification analysis before conducting system decomposition. Since the relative weight of system decomposition is about 37% and that of specification analysis is about 63%, software developers should spend roughly twice as much time and effort on the latter. If the right amount of time and effort is spent on each of the activities, the development process can be made smoother, more efficient, and a better outcome can be achieved. This application presents a methodology which provides an effective way in quantifying resource allocation to enhance the software development process.

## 4.6 Software Verification and Validation

Verification and validation (V&V) are two ways to check whether the design satisfies the user's requirements. According to the IEEE Standard Glossary of Software Engineering Terminology:

Software verification is the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.

Software validation is the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.

In short, Boehem (1981) expressed the difference between software verification and software validation as follows:

Verification: "Are we building the product right?"

Validation: "Are we building the right product?"

In other words, verification checks whether the product under construction meets the requirements definition. Validation checks whether the product's functions are in accordance to the customer's needs. Recently, an "eagle-eyed math lover" high school student, Colin Rizzio, who took the Scholastic Assessment Tests (SAT) with about 350,000 high school students on 12 October 1996, recognized the flaw in the multiple-choice answers for a question dealing with an algebraic equation.

The problem was that the question-writer had used a letter, in this case  $a$ , to represent any number, a standard practice in an algebra equation. The original "correct" answer assumed that 'V' was positive and did not account for the possibility that it was a negative number such as  $-4$ . Students who assumed the number could be negative had a different answer. It was the aim of the SAT to "check your work," which apparently, the testers did not (The Home News &

Tribune, 1997). As a result, the scores of up to 45,000 high-school students, who took the SAT last fall, were boosted by as much as 30 points. The math portion of the SAT test was worth 800 points.

Often, programming is done primarily by scientists or engineers who have little training in software development or programming. They are, however, highly motivated to get a program running in the shortest time possible. The consequence of expedited results is that the users find bugs in the software program after the software product is put into operation.

Although it costs the developer very little to fix faults during the development phase, *i.e.*, the testing phase, it would definitely cost orders of magnitude more to fix faults during the operating and maintenance phases. The cost of fixing an error increases dramatically as the software lifecycle progresses (Pham 2000a).

Verification should be integrated not only in the testing phase but in all phases of the software development lifecycle. It should be noted that testing is one aspect of verification but it cannot replace the task of the verification process. In fact, verification is most effective and efficient when applied from the beginning of the development process.

Verification should be performed independently by a group other than the software developer whose interests lie in showing that the software program works and not in finding bugs or faults in the software. Therefore, an independent group of the development process is more likely to do a thorough testing and execute the software verification, coming up with a series of complex tests that may create blind spots in the evaluation process.

A good independent verification and validation (IV&V) should: (1) have significant experience, (2) have a base of personnel skilled in IV&V, (3) hold transferable tools with knowledable, (4) be a development team participant, and (5) be adaptable to any project.

In general, validation determines end product accuracy, *e.g.*, code, with respect to the software requirements. It determines if the output conforms with what was required? Verification is performed at each phase and between each phase of the development lifecycle. It determines that each phase and subphase product is correct, complete, and consistent with itself and with its predecessor product.

## 4.7 Data Analysis

Traditionally, there are two common types of failure data: time-domain data and interval-domain data. These data are usually used by practitioners when analyzing and predicting reliability applications. Some software reliability models can handle both types of data. The time-domain approach involves recording the individual times at which failure occurred, as illustrated in Table 4.3. The first failure occurred 25 min into the test, the second at 55, the third at 70, the fourth at 95, the fifth at 112, the sixth at 119, the seventh at 143, and the eighth failure at 187 min. Some models may require the time between failures in lieu of the actual failure time. In this example, the values 25, 30, 15, 15, 17, 7, 26, and 44 should be used as the time-domain data set.

The interval-domain approach is characterized by counting the number of failures occurring during a fixed period (*e.g.*, test session, hour, week, day). Using this method, the collected data are a count of the number of failures in the interval. This approach is illustrated in Table 4.4. Using the same failures as in the time-domain example, we would record two failures in the first 1-hour interval, four failures in the second interval, one failure in the third, and one in the fourth. Intervals, however, do not need to be equally spaced for data collection. For example, if the interval for data collection is a test session, one session may last 4 hours and the next may last 8 hours. Models with assumptions that handle this situation should be considered for higher fidelity forecasts for systems with interval-domain data.

**Table 4.3.** Data recording for the time-domain approach

Failure records	Actual failure time (min)	Time between failures (min)
1	25	25
2	55	30
3	70	15
4	95	15
5	112	17
6	119	7
7	143	26
8	187	44

The time-domain approach always provides higher accuracy in the parameter estimates with current tools but involves more data collection efforts than the interval-domain approach. The practitioners must trade off the cost of data collection with the accuracy reliability level required by the model predictions.

**Table 4.4.** Data recording for the interval-domain approach

Time (hours)	Observed number of failures
1	2
2	4
3	1
4	1

### 4.8 Failure Data Sets

This section lists several application data sets that, throughout the book, can be used to implement and illustrate the software reliability modeling.

**Data Set #1: On-line Data Entry Software Package Test Data**

The small on-line data entry software package test data, available since 1980 in Japan (Ohba 1984a), is shown in Table 4.5 (Data set #1). The size of the software has approximately 40,000 LOC. The testing time was measured on the basis of the number of shifts spent running test cases and analyzing the results. The pairs of the observation time and the cumulative number of faults detected are presented in Table 4.5.

**Table 4.5.** On-line IBM entry software package (data set #1)

Testing time (day)	Failures	Cumulative failures
1	2	2
2	1	3
3	1	4
4	1	5
5	2	7
6	2	9
7	2	11
8	1	12
9	7	19
10	3	21
11	1	22
12	2	24
13	2	26
14	4	30
15	1	31
16	6	37
17	1	38
18	3	41
19	1	42
20	3	45
21	1	46

**Data Set #2: On-line Communication System (OCS)**

The On-line Communication System (OCS) project at ABC Software Company was completed in 2000 (Pham 2003a). The project consisted of one unit-manager, one user interface software engineer, and ten software engineers/testers. The overall effort for each of the four phases in the software development process of the project can be described as follows:

<u>Phase</u>	<u>Weeks</u>
Analysis	7
Design	8
Coding	13
Testing	12

The data was collected over a period of 12 weeks during which time the testing started and stopped many times. Errors detection is broken down into sub-categories to help the development and testing team to sort and solve the most critical Modification Requests (MRs) first. These sub-categories are referred to as the severity level depending on the nature of the problem with 1 being the most severe problem, with 2 being the major problem and 3 being a minor problem. The data set #2, maps into week, consists of three types of errors: severe 1, severe 2, and severe 3. The observation time (week) and the number of errors detected per week are presented in Table 4.6.

**Table 4.6.** OCS Failure Data (Data Set #2)

Week	Severe 1	Severe 2	Severe 3
1	4	7	10
2	1	5	2
3	0	0	4
4	1	4	6
5	1	4	6
6	10	15	8
7	4	6	4
8	1	5	3
9	1	1	1
10	3	7	6
11	0	0	1
12	0	1	4

**Date Set #3:** Failure Data from Misra (1983)

A set of failure data taken from Misra (1983), given in Table 4.7, consists of three types of errors: critical, major, and minor. The observation time (week) and the number of failure detected per week are shown in Table 4.7.

**Table 4.7.** Software failure data (data set #3)

Week	Hours	Cumulative days	Critical errors	Major errors	Minor errors
1	62.5	2.6	0	6	9
2	44	4.4	0	2	4
3	40	6.1	0	1	7
4	68	8.9	1	1	6
5	62	11.5	0	3	5
6	66	14.2	0	1	3
7	73	17.3	0	2	2
8	73.5	20.3	0	3	5
9	92	24.2	0	2	4
10	71.4	27.1	0	0	2
11	64.5	29.8	0	3	4
12	64.7	32.5	0	1	7
13	36	34	0	3	0
14	54	36.3	0	0	5
15	39.5	37.9	0	2	3
16	68	40.7	0	5	3
17	61	43.3	0	5	3
18	62.6	45.9	0	2	4
19	98.7	50	0	2	10

**Table 4.7.** (continued)

Week	Hours	Cumulative days	Critical errors	Major errors	Minor errors
20	25	51.1	0	2	3
21	12	51.6	0	1	1
22	55	53.8	0	3	2
23	49	55.9	0	2	4
24	64	58.6	0	4	5
25	26	59.6	0	1	0
26	66	62.4	0	2	2
27	49	64.4	0	2	0
28	52	66.6	0	2	2
29	70	69.5	0	1	3
30	84.5	73	1	2	6
31	83	76.5	1	2	3
32	60	79	0	0	1
33	72.5	82	0	2	1
34	90	85.8	0	2	4
35	58	88.2	0	3	3
36	60	90.7	0	1	2
37	168	97.7	1	2	11
38	111.5	102.3	0	1	9

**Data Set #4: US Naval Tactical Data Systems (NTDS)**

The software data set, listed in Table 4.8, was extracted from information about failures in the development of software for the real-time multi-computer complex of the US Naval Fleet Computer Programming Center of the US Naval Tactical Data Systems (NTDS) (Goel 1979a). The software consists of 38 different project modules. The time horizon is divided into four phases: production phase, test phase, user phase, and subsequent test phase. The 26 software failures were found during the production phase, five during the test phase and; the last failure was found on 4 January 1971. One failure was observed during the user phase, in September 1971, and two failures during the test phase in 1971.

**Data Set #5: Tandem Computers Software Data Project**

This set of Release #1 failure data, given in Table 4.9, is from one of four major releases of software products at Tandem Computers (Wood 1996).

**Data Set #6: On-Line Data Entry IBM Software Package**

The data reported by Ohba (1984a) are recorded from testing an on-line data entry software package developed at IBM. Table 4.10 shows the pair of the observation time (days) and the cumulative number of errors that were detected.

**Data Set #7: AT&T *System T* Project**

The AT&T's *System T* is a network-management system developed by AT&T that receives data from telemetry events, such as alarms, facility-performance information, and diagnostic messages, and forwards them to operators for further action. The system has been tested and failure data has been collected (Ehrlich, 1993). Table 4.11 shows the failures and the inter-failure as well as cumulative failure times (in CPU units).

**Data Set #8: Real-Time Control Systems (Hou et al., 1997)**

The software for monitor and real-time control systems (Tohma 1991) consists of about 200 modules and each module has, on average, 1000 lines of a high-level language like FORTRAN. Table 4.12 records the software failures detected during the 111-day testing period. This actual data is concave overall with several up and downs reflecting different clusters of detected faults.

**Data Set #9: The Real-time Control System Data**

The data is documented in Lyu (1996). There are in total 136 faults reported and the time-between failures (TBF) in second are listed in Table 4.13.

**Data Set #10: Real-Time Command and Control System**

The data set in Table 4.14 was reported by Musa (1987) based on failure data from a real-time command and control system, which represents the failures observed during system testing for 25 hours of CPU time. The delivered number of object instructions for this system was 21700 and was developed by Bell Laboratories.



**Table 4.8.** Naval Tactical Data System (NTDS) software error (data set #4)

Error no. n	Time between errors $x_k$ (days)	Cumulative time $S_n := \sum x_k$ (days)
Production (checkout) phase		
1	9	9
2	12	21
3	11	32
4	4	36
5	7	43
6	2	45
7	5	50
8	8	58
9	5	63
10	7	70
11	1	71
12	6	77
13	1	78
14	9	87
15	4	91
16	1	92
17	3	95
18	3	98
19	6	104
20	1	105
21	11	116
22	33	149
23	7	156
24	91	247
25	2	249
26	1	250
Test phase		
27	87	337
28	47	384
29	12	396
30	9	405
31	135	540
User phase		
32	258	798
Test phase		
33	16	814
34	35	849

**Table 4.9.** Tandem Computers software failure (CPU execution time)

Testing time (weeks)	CPU hours	Defects found
1	519	16
2	968	24
3	1,430	27
4	1,893	33
5	2,490	41
6	3,058	49
7	3,625	54
8	4,422	58
9	5,218	69
10	5,823	75
11	6,539	81
12	7,083	86
13	7,487	90
14	7,846	93
15	8,205	96
16	8,564	98
17	8,923	99
18	9,282	100
19	9,641	100
20	10,000	100

**Table 4.10.** IBM on-line data entry software testing

No. of error	Inter-failure time	Cum. failure time
1	10	10
2	9	19
3	13	32
4	11	43
5	15	58
6	12	70
7	18	88
8	15	103
9	22	125
10	25	150
11	19	169
12	30	199
13	32	231
14	25	256
15	40	296

**Table 4.11.** Failure data from AT & T Network-Management System (data set #7)

Failure index	Failure time	Interfailure time
1	5.50	5.50
2	7.33	1.83
3	10.08	2.75
4	80.97	70.89
5	84.91	3.94
6	99.89	14.98
7	103.36	3.47
8	113.32	9.96
9	124.71	11.39
10	144.59	19.88
11	152.40	7.81
12	166.99	14.60
13	178.41	11.41
14	197.35	18.94
15	262.65	65.30
16	262.69	0.04
17	388.36	125.67
18	471.05	82.69
19	471.50	0.46
20	503.11	31.61
21	632.42	129.31
22	680.02	47.60

**Table 4.12.** Failure per day and cumulative failure (\*: Interpolated data)

Days	Faults	Cumulative faults	Days	Faults	Cumulative faults
1	5*	5*	16	12	183
2	5*	10*	17	8	191
3	5*	15*	18	9	200
4	5*	20*	19	4	204
5	6*	26*	20	7	211
6	8	34	21	6	217
7	2	36	22	9	226
8	7	43	23	4	230
9	4	47	24	4	234
10	2	49	25	2	236
11	31	80	26	4	240
12	4	84	27	3	243
13	24	108	28	9	252
14	49	157	29	2	254
15	14	171	30	5	259

**Table 4.12.** (continued)

Days	Faults	Cumulative faults	Days	Faults	Cumulative faults
31	4	263	72	1	468
32	1	264	73	1	469
33	4	268	74	0	469
34	3	271	75	0	469
35	6	277	76	0	469
36	13	293	77	1	470
37	19	309	78	2	472
38	15	324	79	0	472
39	7	331	80	1	473
40	15	346	81	0	473
41	21	367	82	0	473
42	8	375	83	0	473
43	6	381	84	0	473
44	20	401	85	0	473
45	10	411	86	0	473
46	3	414	87	2	475
47	3	417	88	0	475
48	8	425	89	0	475
49	5	430	90	0	475
50	1	431	91	0	475
51	2	433	92	0	475
52	2	435	93	0	475
53	2	437	94	0	475
54	7	444	95	0	475
55	2	446	96	1	476
56	0	446	97	0	476
57	2	448	98	0	476
58	3	451	99	0	476
59	2	453	100	1	477
60	7	460	101	0	477
61	3	463	102	0	477
62	0	463	103	1	478
63	1	464	104	0	478
64	0	464	105	0	478
65	1	465	106	1	479
66	0	465	107	0	479
67	0	465	108	0	479
68	1	466	109	1	480
69	1	467	110	0	480
70	0	467	111	1	481
71	0	467			

**Data Set #11: Telecommunication System Data**

The data set #11 was reported by Zhang (2002) based on system test data for a telecommunication system. System test data consisting of two releases (Phases 1 and 2) are shown in Tables 4.15 and 4.16. In both tests, automated test and human-involved tests are executed on multiple test beds.

**Table 4.13.** The real-time control system data

Fault	TBF	Cum. TBF	Fault	TBF	Cum. TBF
1	3	3	35	227	5324
2	30	33	36	65	5389
3	113	146	37	176	5565
4	81	227	38	58	5623
5	115	342	39	457	6080
6	9	351	40	300	6380
7	2	353	41	97	6477
8	91	444	42	263	6740
9	112	556	43	452	7192
10	15	571	44	255	7447
11	138	709	45	197	7644
12	50	759	46	193	7837
13	77	836	47	6	7843
14	24	860	48	79	7922
15	108	968	49	816	8738
16	88	1056	50	1351	10089
17	670	1726	51	148	10237
18	120	1846	52	21	10258
19	26	1872	53	233	10491
20	114	1986	54	134	10625
21	325	2311	55	357	10982
22	55	2366	56	193	11175
23	242	2608	57	236	11411
24	68	2676	58	31	11442
25	422	3098	59	369	11811
26	180	3278	60	748	12559
27	10	3288	61	0	12559
28	1146	4434	62	232	12791
29	600	5034	63	330	13121
30	15	5049	64	365	13486
31	36	5085	65	1222	14708
32	4	5089	66	543	15251
33	0	5089	67	10	15261
34	8	5097	68	16	15277

**Table 4.13.** (continued)

Fault	TBF	Cum. TBF	Fault	TBF	Cum. TBF
69	529	15806	103	108	42296
70	379	16185	104	0	42296
71	44	16229	105	3110	45406
72	129	16358	106	1247	46653
73	810	17168	107	943	47596
74	290	17458	108	700	48296
75	300	17758	109	875	49171
76	529	18287	110	245	49416
77	281	18568	111	729	50145
78	160	18728	112	1897	52042
79	828	19556	113	447	52489
80	1011	20567	114	386	52875
81	445	21012	115	446	53321
82	296	21308	116	122	53443
83	1755	23063	117	990	54433
84	1064	24127	118	948	55381
85	1783	25910	119	1082	56463
86	860	26770	120	22	56485
87	983	27753	121	75	56560
88	707	28460	122	482	57042
89	33	28493	123	5509	62551
90	868	29361	124	100	62651
91	724	30085	125	10	62661
92	2323	32408	126	1071	63732
93	2930	35338	127	371	64103
94	1461	36799	128	790	64893
95	843	37642	129	6150	71043
96	12	37654	130	3321	74364
97	261	37915	131	1045	75409
98	1800	39715	132	648	76057
99	865	40580	133	5485	81542
100	1435	42015	134	1160	82702
101	30	42045	135	1864	84566
102	143	42188	136	4116	88682

**Table 4.14.** Real-time Command and Control Data (in a one-hour interval)

Hour	Number of failures	Cum. Failures
1	27	27
2	16	43
3	11	54
4	10	64
5	11	75
6	7	83
7	2	84
8	5	89
9	3	92
10	1	93
11	4	97
12	7	104
13	2	106
14	5	111
15	5	116
16	6	122
17	0	122
18	5	127
19	1	128
20	1	129
21	2	131
22	1	132
23	2	134
24	1	135
25	1	136

**Table 4.15.** Phase 1 system test data

Week index	Exposure time (cum. system test hours)	Fault	Cum. fault
1	356	1	1
2	712	0	1
3	1068	1	2
4	1424	1	3
5	1780	2	5
6	2136	0	5
7	2492	0	5
8	2848	3	8

**Table 4.15.** (continued)

Week index	Exposure time (cum. system test hours)	Fault	Cum. fault
9	3204	1	9
10	3560	2	11
11	3916	2	13
12	4272	2	15
13	4628	4	19
14	4984	0	19
15	5340	3	22
16	5696	0	22
17	6052	1	23
18	6408	1	24
19	6764	0	24
20	7120	0	24
21	7476	2	26

**Table 4.16.** Phase 2 system test data

Week index	Exposure time (Cum. system test hours)	Fault	Cum. fault
1	416	3	3
2	832	1	4
3	1248	0	4
4	1664	3	7
5	2080	2	9
6	2496	0	9
7	2912	1	10
8	3328	3	13
9	3744	4	17
10	4160	2	19
11	4576	4	23
12	4992	2	25
13	5408	5	30
14	5824	2	32
15	6240	4	36
16	6656	1	37
17	7072	2	39
18	7488	0	39
19	7904	0	39
20	8320	3	42
21	8736	1	43



## 4.9 Further Reading

Some interesting research papers on software engineering and books are:

- Frankl, P. G. and E. J. Weyuker, "Testing Software to Detect and Reduce Risk," *Journal of Systems and Software*, 2000, vol. 53, p. 275-286
- Tahvildari, L. and A. Singh, "Software Bugs," in *Wiley Encyclopedia of Electrical and Electronics Engineering*, J.G. Webster (ed.), vol 19, Wiley, 1999, p. 445-455
- Schneidewind, N.F., "Software Maintenance," in *Wiley Encyclopedia of Electrical and Electronics Engineering*, J.G. Webster (ed.), vol 19, Wiley, 1999, p. 483-492
- Pfleeger, S. L., *Software Engineering: Theory and Practice*, Prentice Hall, New Jersey, 1998
- Anderson, T, *Software Requirements Specification and Testing*, Blackwell Scientific Publications, London, 1995
- Smith, D.J., *Achieving Quality Software*, 3rd edition, Chapman & Hall, NewYork, 1995

## 4.10 Problems

1. List several practical system failures (since 1995) caused by software.
2. What are the differences between verification and validation? Provide several real world applications and examples.
3. Identify from articles and books at least two methodologies and/or models to support software V&V processes. Prepare a summary of these methodologies and/or models.
4. Read three papers on software lifecycle published in the IEEE Transactions on Software Engineering. Summarize the material and relate it to the software development process.
5. Why it is important to integrate analysis, design, coding and testing in the software development process? Who should be involved in this integration and which techniques and/or methodologies are available to support this integrated approach?
6. What are the main difficulties involved in the data collection of time-domain approach and interval-domain approach?
7. Using the generalized AHP in Section 4.5, analyze the impacts among the activities and construct the weight diagrams for the analysis phase (see Section 4.4) in the development process.

- 8.** Using the generalized AHP in Section 4.5, analyze the impacts among the activities and construct the weight diagrams for the analysis phase (see Section 4.4) in the development process.
- 9.** Using the generalized AHP in Section 4.5, analyze the impacts among the activities and construct the weight diagrams for the design phase (see Section 4.4) in the development process.
- 10.** Using the generalized AHP in Section 4.5, analyze the impacts among the activities and construct the weight diagrams for the testing phase (see Section 4.4) in the development process.
- 11.** Using the generalized AHP in Section 4.5, analyze the impacts among the activities and construct the weight diagrams for the operating phase (see Section 4.4) in the development process.

## Software Reliability Modeling

### 5.1 Introduction

For software qualification, it is highly desirable to have an estimate of the remaining errors in a software system. It is difficult to determine such an important finding without knowing what the initial errors are. Research activities in software reliability engineering have been studied over the past 30 years and many statistical models and various techniques have been developed for estimating and predicting reliability of software and numbers of residual errors in software. From historical data on programming errors, there are likely to be about 8 errors per 1000 program statements after the unit test. This, of course, is just an average and does not take into account any tests on the program.

There are two main types of software reliability models: the deterministic and the probabilistic. The deterministic model is used to study the number of distinct operators and operands in a program as well as the number of errors and the number of machine instructions in the program. Performance measures of the deterministic type are obtained by analyzing the program texture and do not involve any random event. Two well-known models are: Halstead's software metric and McCabe's cyclomatic complexity metric. Halstead's software metric is used to estimate the number of errors in the program, whereas McCabe's cyclomatic complexity metric (McCabe 1976) is used to determine an upper bound on the model for estimating the number of remaining software defects. In general, these models represent a growing quantitative approach to the measurement of computer software.

The probabilistic model represents the failure occurrences and the fault removals as probabilistic events. The probabilistic software reliability models can be classified into different groups (Pham 2000a):

- Error seeding
- Failure rate
- Curve fitting
- Reliability growth

- Markov structure
- Time-series
- Nonhomogeneous Poisson process.

In this chapter, we discuss various types of software reliability models and methods for estimating software reliability and other performance measures such as software complexity, software safety, and the number of remaining errors.

## 5.2 Halstead's Software Metric

Halstead's theory of software metric is probably the best-known technique to measure the complexity in a software program and the amount of difficulty involved in testing and debugging the software. Halstead (1977) uses the number of distinct operators and the number of distinct operands in a program to develop expressions for the overall program length, volume and the number of remaining defects in a program. The following notations are used:

- $n_1$  = number of unique or distinct operators appearing in a program
- $n_2$  = number of unique or distinct operands appearing in a program
- $N_1$  = total number of operators occurring in a program
- $N_2$  = total number of operands occurring in a program
- $N$  = length of the program
- $V$  = volume of the program
- $E$  = number of errors in the program
- $I$  = number of machine instructions

The length and the volume measure of the program can be obtained by, respectively, (Halstead 1977):

$$N = N_1 + N_2 \quad (5.1)$$

and

$$V = N \log_2(n_1 + n_2) \quad (5.2)$$

where

$$\begin{aligned} N_1 &= n_1 \log_2 n_1 \\ N_2 &= n_2 \log_2 n_2 \end{aligned} \quad (5.3)$$

Halstead also proposed two empirical formulae to estimate the number of remaining defects in the program,  $E$ , from program volume. The two formulae, namely, Halstead empirical model 1 and Halstead empirical model 2, respectively, are

$$\hat{E} = \frac{V}{3000} \quad (5.4)$$

and

$$\hat{E} = \frac{A}{3000} \quad (5.5)$$

where

$$A = \left( \frac{V}{\frac{2n_2}{n_1 N_2}} \right)^{\frac{2}{3}} \quad (5.6)$$

To examine whether Halstead's software science can offer reasonable estimates for the number of remaining defects, we will discuss the following two examples.

*Example 5.1:* Consider the Interchange Sort Program (Fitzsimmons 1978) for the Fortran version in Table 5.1. From Table 5.1, the length and the volume for this program can be calculated as follows:

$$N = N_1 + N_2 = 50$$

and

$$\begin{aligned} V &= N \log_2(n_1 + n_2) \\ &= 50 \log_2(10 + 7) \\ &= 204 \end{aligned}$$

From Halstead's empirical model 1, the number of remaining defects in this program can be expected to be

$$\hat{E} = \frac{204}{3000} = 0.068$$

It should be noted that the volume for this program under the assembly language version would be 328 since it needs more effort to specify a program in assembly programming language.

*Example 5.2:* Let us consider Akiyama's software data (Halstead 1977) in Table 5.2, which will be used to validate Halstead's empirical model 2. The software system consists of nine modules and is written in assembly language.

Assuming that each of the S machine language steps include one operator and one operand, we obtain

$$N_1 = S, N_2 = S, \text{ and } N = 2S.$$

Halstead assumed that the number of distinct operators appearing in a program,  $n_1$ , was equal to the sum of the number of machine language instruction types, program calls, and unique program decisions. He further assumed that there were 64 types of machine language instructions and that only one-third of the decisions were unique. Therefore, Halstead proposed  $n_1$  as follows:

$$n_1 = \frac{D}{3} + J + 64 \quad (5.7)$$

Now we use the following formula:

$$N = n_1 \log_2 n_1 + n_2 \log_2 n_2 \quad (5.8)$$

to obtain  $n_2$  when  $n_1$  and  $N$  are known. Thus,  $A$  can be also obtained. The computational results are shown in Table 5.3. From the results in Table 5.3, Halstead's empirical model 2 is close to the number of observed defects than empirical model 1.

**Table 5.1.** Operators and operands for an interchange sort program

Interchange sort program	
SUBROUTINE SORT (X,N) DIMENSION X(N) IF (N.LT.2) RETURN DO 201 = 2,N DO 10 J = 1,1 IF (X(1).GE.X(J)) GO TO 10 SAVE = X(1) X(I) = X(J) X(J) = SAVE 10 CONTINUE 20 CONTINUE RETURN END	
Operators of the interchange sort program	
Operator	Count
1 End of statement	7
2 Array subscript	6
3 =	5
4 IF	2
5 DO	2
6 ,	2
7 End of program	1
8.LT.	1
9.GE.	1
$n_1 = 10$ GO TO 10	1
$N_1 = 28$	
Operands of the interchange sort program	
Operand	Count
1 X	6
2 I	5
3 J	4
4 N	2
5 2	2
6 SAVE	2
$n_2 = 7$ 1	1
$N_2 = 22$	

**Table 5.2.** Akiyama's published data

Program module	Program ( $S$ )	Decisions ( $D$ )	Subroutine calls ( $J$ )	Number of defects observed ( $O$ )
MA	4032	372	283	102
MB	1329	215	49	18
MC	5453	552	362	146
MD	1674	111	130	26
ME	2051	315	197	71
MF	2513	217	186	37
MG	699	104	32	16
MH	3792	233	110	50
MX	3412	416	230	80

**Table 5.3.** Computational results of Akiyama's software data

Program module	$N$	$n_1$	$n_2$	$A^{1.5}$ $\times 10^6$ )	$E$	$V/3000$	Number of defects observed ( $O$ )
MA	8064	471	442	170.3	102	26.4	102
MB	2658	180	176	15.3	21	7.5	18
MC	10906	610	574	322.6	157	37.1	146
MD	3348	231	201	28.2	31	9.7	26
ME	4102	336	138	100.2	72	12.3	71
MF	5026	322	287	65.5	54	15.5	37
MG	1398	131	76	6.5	12	3.6	16
MH	7584	252	603	58.5	50	24.6	50
MX	6824	433	357	135.9	88	21.9	80

### 5.3 McCabe's Cyclomatic Complexity Metric

A cyclomatic complexity metric measure of software proposed by McCabe (1976) is a complexity measure of the digraph based on the control flow representation of a program.

Cyclomatic complexity is a software metric that provides a quantitative measure of the logical complexity of a program by counting the decision points. For example, one should start with 1 for the straight path through the module or subroutine (McConnel 1993). Then 1 should be added each time one of the following keywords appear: IF, REPEAT, WHILE, FOR, OR, AND. Also, 1 should be added for each case in a case statement and also if the case statement lacks a default case. If the total score is less than 10, then by using the McCabe's measure, the code is considered to be of high quality software. McCabe has suggested that a program with a high level of the metric is very difficult to produce and maintain. He recommended a total score of 10 as an upper limit for the Fortran environment (Rook 1990).

In a strongly connected graph (with a path joining any pair of nodes), the cyclomatic number is equal to the maximum number of linearly independent circuits. The linearly independent circuits form a basis for the set of all circuits in  $G$ , and any path passing through  $G$  can be expressed as a linear combination of the circuits.

The cyclomatic number  $V(G)$  of a graph  $G$  can be determined from the following formula:

$$V(G) = e - n + 2p \quad (5.9)$$

where

$e$  = number of edges in the control graph in which an edge is equivalent to a branching point in the program

$n$  = number of vertices in the control graph and a vertex is equivalent to a sequential block of code in the program

$p$  = number of connected elements (usually 1)

The cyclomatic number of a graph with multiple connected components is equal to the sum of the cyclomatic numbers of the connected components. In any program that can be represented by a state diagram, the McCabe cyclomatic complexity metric can also be used to calculate the number of control flow paths ( $FP$ ), i.e.,

$$FP = e - n + 2$$

where  $e$  is the number of edges and  $n$  is the number of nodes. Here, edges are represented on the diagram by arrows and nodes are shown on the diagram with circles.

*Example 5.3:* A control flow path is a path that considers only a single loop iteration. The program state diagram, shown in Pham (2000a) (Figure 4.1), measures the time between two switch inputs. It calculates the speed of a mini-van in which the number of edges equals 8 and the number of nodes equals 7. Therefore, the number of control flow paths is

$$FP = 8 - 7 + 2 = 3$$

The program in Figure 5.1 has three control flow paths which are: 1234567, 121234567, 123454567.

Another simple way of computing the cyclomatic number is as follows:

$$V(G) = \pi + 1$$

where  $\pi$  is the number of predicate nodes (decisions or branches) in the program. In other words, the cyclomatic number is a measure of the number of branches in a program. A branch occurs in IF, WHILE, REPEAT, and CASE statements (a GO TO statement is normally excluded from the structured program). The cyclomatic number has been widely used in predicting the number of errors and in measuring software quality.

McCabe (1976) notes that, when used in the context of the basis path testing method, the cyclomatic complexity,  $V(G)$ , provides an upper bound for the number of independent paths in the basis set of a program. It also provides an upper bound on the number of tests that must be conducted to ensure that all program statements have been executed at least once.



It should be noted that the cyclomatic complexity is an interesting measure of whether the software designer has followed good structured programming and software standard practices. The cyclomatic complexity also estimates how difficult it will be to test all the paths in the program and thus, provide useful information of how difficult it will be to satisfy the software specifications and requirements (Basili 1984). Also, McCabe's measure will be less useful if the software developers and testing debuggers are interested in detecting all the faults (Friedman 1995).

## 5.4 Error Seeding Models

The error seeding group models estimate the number of errors in a program by using the multistage sampling technique. Errors are divided into indigenous errors and induced errors (seeded errors). The unknown number of indigenous errors is estimated from the number of induced errors and the ratio of the two types of errors obtained from the debugging data. Models included in this group are:

- Mills' error seeding model (Mills, 1970)
- Cai's model (Cai, 1998)
- Hypergeometric distribution model (Tohma, 1991).

### Mills' Error Seeding Model

Mills' error seeding model (Mills 1970) proposed an error seeding method to estimate the number of errors in a program by introducing seeded errors into the program. From the debugging data, which consist of inherent errors and induced errors, the unknown number of inherent errors could be estimated. If both inherent errors and induced errors are equally likely to be detected, then the probability of  $k$  induced errors in  $r$  removed errors follows a hypergeometric distribution which is given by

$$P(k; N, n_1, r) = \frac{\binom{n_1}{k} \binom{N}{r-k}}{\binom{N+n_1}{r}}, \quad k = 1, 2, \dots, r \quad (5.10)$$

where

- $N$  = total number of inherent errors
- $n_1$  = total number of induced errors
- $r$  = total number of errors removed during debugging
- $k$  = total number of induced errors in  $r$  removed errors
- $r - k$  = total number of inherent errors in  $r$  removed errors

Since  $n_1$ ,  $r$ , and  $k$  are known, the MLE of  $N$  can be shown to be (Huang 1984)

$$\hat{N} = \lfloor N_0 \rfloor + 1$$

where

$$N_0 = \frac{n_1(r-k)}{k} - 1 \quad (5.11)$$

If  $N_0$  is an integer, then  $N_0$  and  $N_0 + 1$  are both the MLEs of  $N$ .

*Example 5.4:* Assume  $n_1 = 10$ ,  $r = 15$ , and  $k = 6$ . The number of inherent errors can be estimated using equation (5.11), i.e.,

$$\begin{aligned} N_0 &= \frac{n_1(r-k)}{k} - 1 \\ &= \frac{10(15-6)}{6} - 1 \\ &= 14 \end{aligned}$$

$N_0 = 14$  is an integer, therefore, both values of 14 and 15 are the MLEs of the total number of inherent errors.

However, there are some drawbacks with this model. It is expensive to conduct testing of the software and at the same time it increases the testing effort. This method was also criticized for its inability to determine the type, location, and difficulty level of the induced errors such that they would be detected equally likely as the inherent errors.

Another realistic method for estimating the residual errors in a program is based on two independent groups of programmers testing the program for errors using independent sets of test cases. Suppose that out of a total number of  $N$  initial errors, the first programmer detects  $n_1$  errors (and does not remove them at all) and the second independently detects  $r$  errors from the same program.

Assume that  $k$  common errors are found by both programmers (Pham 2000a, Figure 4.2). If all errors have an equal chance of being detected, then the fraction detected by the first programmer ( $k$ ) of a randomly selected subset of errors (e.g.,  $r$ ) should equal the fraction that the first programmer detects ( $n_1$ ) of the total number of initial errors  $N$ . In other words,

$$\frac{k}{r} = \frac{n_1}{N}$$

so that an estimate of the total number of initial errors,  $N$ , is

$$\hat{N} = \frac{n_1 r}{k}$$

The probability of exactly  $N$  initial errors with  $k$  common errors in  $r$  detected errors by the second programmer can be obtained using a hypergeometric distribution as follows:

$$P(k; N, n_1, r) = \frac{\binom{n_1}{k} \binom{N-n_1}{r-k}}{\binom{N}{r}} \quad (5.12)$$

and the MLE of  $N$  is

$$\hat{N} = \frac{n_1 r}{k} \quad (5.13)$$

which is the same as the above.

*Example 5.5:* Given  $n_1 = 10$ ,  $r = 15$ , and  $k = 6$ , the number of initial errors (or inherent errors) is given by

$$\hat{N} = \frac{n_1 r}{k} = \frac{10(15)}{6} = 25$$

### Cai's Model

Cai (1998) recently modified Mills' model by dividing the software into two parts: Part 0 and Part 1. This model is used to estimate the number of defects remaining in the software. The following assumptions are applied to this model:

1. There are  $N$  defects remaining in the software where Part 0 contains  $N_0$  and Part 1 contains  $N_1$  remaining defects, *i.e.*,  $N = N_0 + N_1$ .
2. Each of the remaining defects has the same probability of being detected.
3. The defect is removed when detected.
4. Only one remaining defect is removed each time and no new defects are introduced.
5. There are  $n$  remaining defects removed.

Let  $t_i$  represent the time instant of the  $i^{\text{th}}$  remaining defect being removed, and let  $Y_i$  be a random variable such that

$$Y_i = \begin{cases} 0 & \text{if the } i^{\text{th}} \text{ detected defect is contained in Part 0} \\ 1 & \text{if the } i^{\text{th}} \text{ detected defect is contained in Part 1.} \end{cases}$$

Define  $N_j(i)$  be the number of defects remaining in Part  $j$  in the time interval  $(t_i, t_{i+1}]$  for  $i = 0, 1, 2, \dots, n$  and  $j = 0$  or  $1$ . Assume  $y_i$  are the observed values of  $Y_i$  and  $y_0 = 0$ . Therefore,

$$\begin{aligned} N_0(i) &= N_0 - i + \sum_{j=0}^i y_j \\ N_1(i) &= N_1 - \sum_{j=0}^i y_j \end{aligned} \quad (5.14)$$

Let  $p_j(i)$  be the probability of having a defect remaining in Part  $j$  detected during the time interval  $(t_i, t_{i+1}]$  where  $i = 0, 1, 2, \dots, n$  and  $j = 0$  or  $1$ . Then we obtain

$$\begin{aligned} p_0(i) &= \frac{N_0(i)}{N_0(i) + N_1(i)} \\ &= \frac{N_0 - i + \sum_{j=0}^i y_j}{N_0 + N_1 - i} \end{aligned} \quad (5.15)$$

and

$$\begin{aligned}
 p_1(i) &= \frac{N_1(i)}{N_0(i) + N_1(i)} \\
 &= \frac{N_1 - \sum_{j=0}^i y_j}{N_0 + N_1 - i}
 \end{aligned} \tag{5.16}$$

Now we wish to estimate  $N_0$  and  $N_1$  using the MLE method. The likelihood function can be determined as follows (Cai 1998):

$$\begin{aligned}
 L(y_1, \dots, y_n) &= P\{Y_1 = y_1, \dots, Y_n = y_n\} \\
 &= \prod_{i=1}^n P\{Y_i = y_i \mid Y_1 = y_1, \dots, Y_{i-1} = y_{i-1}\}
 \end{aligned}$$

Note that

$$P\{Y_i = y_i \mid Y_1 = y_1, \dots, Y_{i-1} = y_{i-1}\} = \begin{cases} p_0(i-1) & \text{if } y_i = 0 \\ p_1(i-1) & \text{if } y_i = 1 \end{cases} \tag{5.17}$$

or, equivalently, that

$$P\{Y_i = y_i \mid Y_1 = y_1, \dots, Y_{i-1} = y_{i-1}\} = [p_0(i-1)]^{1-y_i} [p_1(i-1)]^{y_i}$$

where  $p_0(i)$  and  $p_1(i)$  are given in equations (5.15) and (5.16), respectively. Thus,

$$L(y_1, \dots, y_n) = \prod_{i=1}^n [p_0(i-1)]^{1-y_i} [p_1(i-1)]^{y_i}$$

Taking the ln of the likelihood function, we obtain

$$\ln L(y_1, \dots, y_n) = \sum_{i=1}^n (1-y_i) \ln[p_0(i-1)] + \sum_{i=1}^n y_i \ln[p_1(i-1)] \tag{5.18}$$

Substituting  $p_0(i)$  and  $p_1(i)$  of equations (5.15) and (5.16), respectively, into the above equation and taking the first derivatives with respect to  $N_0$  and  $N_1$ , the estimates of  $N_0$  and  $N_1$ , are determined by the following equations:

$$\sum_{i=1}^n \frac{1-y_i}{N_0 - i + 1 + \sum_{j=0}^{i-1} y_j} = \frac{1}{N_0 + N_1 - i + 1} \tag{5.19}$$

and

$$\sum_{i=1}^n \frac{y_i}{N_1 - \sum_{j=0}^{i-1} y_j} = \frac{1}{N_0 + N_1 - i + 1} \tag{5.20}$$

### Hypergeometric Distribution Model

Tohma *et al.* (1991) proposed a model for estimating the number of faults initially resident in a program at the beginning of the test or debugging process based on the hypergeometric distribution. Let  $C_{i-1}$  be the cumulative number of errors already detected so far by  $t_1, t_2, \dots, t_{i-1}$ , and let  $N_i$  be the number of newly detected errors by time  $t_i$ . Assume:

1. A program initially contains  $m$  faults when the test phase starts.
2. A test is defined as a number of test instances which are couples of input data and output data. In other words, the collection of test operations performed in a day or a week is called a *test instance*. The test instances are denoted by  $t_i$  for  $i = 1, 2, \dots, n$ .
3. Detected faults are not removed between test instances.

Therefore, from the latter assumption, same faults can be experienced at several test instances. Let  $W_i$  be the number of faults experienced by test instance  $t_i$ . It should be noted that some of the  $W_i$  faults may be those that are already counted in  $C_{i-1}$ , and the remaining  $W_i$  faults account for the newly detected faults.

If  $n_i$  is an observed instance of  $N_i$ , then we can see that  $n_i \leq W_i$ . Each fault can be classified into one of two categories:

- Newly discovered faults
- Rediscovered faults

If we assume that the number of newly detected faults  $N_i$  follows a hypergeometric distribution, then the probability of obtaining exactly  $n_i$  newly detected faults among  $W_i$  faults is (Tohma 1991)

$$P(N_i = n_i) = \frac{\binom{m - C_{i-1}}{n_i} \binom{C_{i-1}}{W_i - n_i}}{\binom{m}{W_i}} \quad (5.21)$$

where

$$C_{i-1} = \sum_{k=1}^{i-1} n_k, \quad C_0 = 0, \quad n_0 = 0$$

and

$$\max\{0, W_i - C_{i-1}\} \leq n_i \leq \min\{W_i, m - C_{i-1}\}$$

for all  $i$ . Since  $N_i$  is assumed to be hypergeometrically distributed, the expected number of newly detected faults during the interval  $[t_{i-1}, t_i]$  is

$$E(N_i) = \frac{(m - C_{i-1})W_i}{m} \quad (5.22)$$

and the expected value of  $C_i$  is given by

$$E(C_i) = m \left[ 1 - \prod_{j=1}^i (1 - p_j) \right] \quad (5.23)$$

where

$$p_i = \frac{W_i}{m} \quad i = 1, 2, \dots$$

## 5.5 Failure Rate Models

The failure rate group of models is used to study the program failure rate per fault at the failure intervals. Models included in this group are:

- Jelinski and Moranda (Jelinski 1972)
- Schick and Wolverton (Schick 1978)
- Jelinski-Moranda geometric (Moranda 1979)
- Moranda geometric Poisson (Littlewood 1979)
- Negative-binomial Poisson
- Modified Schick and Wolverton (Sukert 1977)
- Goel and Okumoto imperfect debugging (Goel 1979a).

This group of models studies how failure rates change at the failure time during the failure intervals. As the number of remaining faults changes, the failure rate of the program changes accordingly. Since the number of faults in the program is a discrete function, the failure rate of the program is also a discrete function with discontinuities at the failure times.

### Jelinski-Moranda Model

The Jelinski-Moranda (J-M) model (Jelsinki 1972) is one of the earliest software reliability models. Many existing software reliability models are variants or extensions of this basic model. The assumptions in this model include the following:

- The program contains  $N$  initial faults which is an unknown but fixed constant.
- Each fault in the program is independent and equally likely to cause a failure during a test.
- Time intervals between occurrences of failure are independent of each other.
- Whenever a failure occurs, a corresponding fault is removed with certainty.
- The fault that causes a failure is assumed to be instantaneously removed, and no new faults are inserted during the removal of the detected fault.
- The software failure rate during a failure interval is constant and is proportional to the number of faults remaining in the program.

The program failure rate at the  $i$ th failure interval is given by

$$\lambda(t_i) = \phi[N - (i - 1)], \quad i = 1, 2, \dots, N \quad (5.24)$$

where

$\phi$  = a proportional constant, the contribution any one fault makes to the overall program

$N$  = the number of initial faults in the program

$t_i$  = the time between the  $(i-1)^{th}$  and the  $i^{th}$  failures.

For example, the initial failure intensity is

$$\lambda(t_1) = \phi N$$

and after the first failure, the failure intensity decreases to

$$\lambda(t_2) = \phi(N-1)$$

and so on. The pdf and cdf of  $t_i$  are

$$\begin{aligned} f(t_i) &= \lambda(t_i) e^{-\int_0^{t_i} \lambda(x_i) dx_i} \\ &= \phi[N-(i-1)] e^{-\int_0^{t_i} \lambda(x_i) dx_i} \\ &= \phi[N-(i-1)] e^{-\phi(N-(i-1))t_i} \end{aligned}$$

and

$$\begin{aligned} F(t_i) &= \int_0^{t_i} f(x_i) dx_i \\ &= \int_0^{t_i} \phi[N-(i-1)] e^{-\phi[N-(i-1)]x_i} dx_i \\ &= 1 - e^{-\phi[N-(i-1)]t_i} \end{aligned}$$

respectively. The software reliability function is, therefore,

$$R(t_i) = e^{-\phi(N-i+1)t_i} \quad (5.25)$$

The property of this model is that the failure rate is constant and the software during the testing stage is unchanged or frozen.

Suppose that the failure data set  $\{t_1, t_2, \dots, t_n\}$  is given and assume that  $\phi$  is known. Using the MLE method, we obtain the likelihood function as follows:

$$\begin{aligned} L(N) &= \prod_{i=1}^n f(t_i) \\ &= \prod_{i=1}^n [\phi(N-(i-1)) e^{-\phi(N-(i-1))t_i}] \\ &= \phi^n \prod_{i=1}^n [N-(i-1)] e^{-\phi \sum_{i=1}^n [N-(i-1)]t_i} \end{aligned}$$

and the log of the likelihood function is

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

Taking the first partial derivative of the above function with respect to  $N$ , we obtain

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

Set

$$\frac{\partial}{\partial N} \ln L(N) = 0$$

then

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

Thus, the MLE of  $N$  can be obtained by solving the following equation:

$$j \quad (5.26)$$

In many applications, the parameter  $\phi$  is also not known. In this case, we wish to estimate both the parameters  $N$  and  $\phi$  which are unknown. Again, the log likelihood function is

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

Taking the derivatives of  $\ln L(N, \phi)$  with respect to  $N$  and  $\phi$ , we obtain

$$\frac{\partial}{\partial N} [\ln L(N, \phi)] = \sum_{i=1}^n \frac{1}{N - (i-1)} - \phi \sum_{i=1}^n t_i \equiv 0$$

and

$$\frac{\partial}{\partial \phi} [\ln L(N, \phi)] = \frac{n}{\phi} - \sum_{i=1}^n [N - (i-1)] t_i \equiv 0$$

From the two equations above, we obtain

$$\phi = \frac{\sum_{i=1}^n \frac{1}{N - (i-1)}}{\sum_{i=1}^n t_i} \quad (5.27)$$

and

$$n \sum_{i=1}^n t_i = \left[ \sum_{i=1}^n [N - (i-1)] t_i \right] \left[ \sum_{i=1}^n \frac{1}{N - (i-1)} \right] \quad (5.28)$$

### Schick-Wolverton Model

The Schick-Wolverton (S-W) model (Schick 1978) is a modification to the J-M model. It is similar to the J-M model except that it further assumes that the failure rate at the  $i^{th}$  time interval increases with time  $t_i$  since the last debugging. In the model, the program failure rate function between the  $(i-1)^{th}$  and the  $i^{th}$  failure can be expressed as

$$\lambda(t_i) = \phi[N - (i-1)]t_i \quad (5.29)$$

where  $\phi$  and  $N$  are the same as that defined in the J-M model and  $t_i$  is the test time since the  $(i-1)^{th}$  failure.



The pdf of  $t_i$  can be obtained as follows:

$$f(t_i) = \phi[N - (i-1)]t_i e^{-\frac{[N-(i-1)]t_i^2}{2}} \quad \text{for } i = 1, 2, \dots, N \quad (5.30)$$

Hence, the software reliability function is

$$\begin{aligned} R(t_i) &= e^{-\int_0^{t_i} \lambda(t_i) dt_i} \\ &= e^{-\frac{\phi[N-i+1]t_i^2}{2}} \end{aligned} \quad (5.31)$$

We now wish to estimate  $N$  assuming that  $\phi$  is given. Using the MLE method, the log likelihood function is given by

$$\begin{aligned} \ln L(N) &= \ln \left\{ \prod_{i=1}^n f(t_i) \right\} \\ &= \ln \left\{ \prod_{i=1}^n \left( \phi[N - (i-1)]t_i e^{-\frac{\phi[N-(i-1)]t_i^2}{2}} \right) \right\} \\ &= n \ln \phi + \sum_{i=1}^n \ln[N - (i-1)] + \sum_{i=1}^n \ln t_i - \sum_{i=1}^n \phi[N - (i-1)] \frac{t_i^2}{2} \end{aligned} \quad (5.32)$$

Taking the first derivative with respect to  $N$ , we have

$$\frac{\partial}{\partial N} [\ln L(N)] = \sum_{i=1}^n \frac{1}{N - (i-1)} - \phi \sum_{i=1}^n \frac{t_i^2}{2} \equiv 0$$

Therefore, the MLE of  $N$  can be obtained by solving the following equation:

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

Next, we assume that both  $N$  and  $\phi$  are unknown. From equation (32), we obtain

$$\frac{\partial}{\partial N} [\ln L(N, \phi)] = \sum_{i=1}^n \frac{1}{N - (i-1)} - \phi \sum_{i=1}^n \frac{t_i^2}{2} \equiv 0$$

and

$$\frac{\partial}{\partial \phi} [\ln L(N, \phi)] = \frac{n}{\phi} - \sum_{i=1}^n [N - (i-1)] \frac{t_i^2}{2} \equiv 0$$

Therefore, the MLEs of  $N$  and  $\phi$  can be found by solving the two equations simultaneously as follows:

$$\phi = 2 \sum_{i=1}^n \frac{1}{[N - (i-1)]T} \quad (5.33)$$

$$N = \frac{2n}{\phi T} + \frac{\sum_{i=1}^n (i-1)t_i^2}{T} \quad (5.34)$$

where

$$T = \sum_{i=1}^n t_i^2$$

### Jelinski-Moranda Geometric Model

The J-M geometric model (Moranda 1979) assumes that the program failure rate function is initially a constant  $D$  and decreases geometrically at failure times. The program failure rate and reliability function of time-between-failures at the  $i^{th}$  failure interval can be expressed, respectively, as

$$\lambda(t_i) = Dk^{i-1} \quad (5.35)$$

and

$$\begin{aligned} R(t_i) &= e^{-\int_0^{t_i} \lambda(x_i) dx_i} \\ &= e^{-Dk^{i-1} t_i} \end{aligned} \quad (5.36)$$

where

$D$  = initial program failure rate;

$k$  = parameter of geometric function ( $0 < k < 1$ )

If we allow multiple error removal in a time interval, then the failure rate function becomes

$$\lambda(t_i) = Dk^{n_{i-1}}$$

where  $n_{i-1}$  is the cumulative number of errors found up to the  $(i-1)^{th}$  time interval. The software reliability function can be written as

$$R(t_i) = e^{-D k^{n_{i-1}} t_i} \quad (5.37)$$

### Moranda Geometric Poisson Model

The Moranda geometric Poisson model (Moranda 1975) assumes fixed times  $T, 2T$ , of equal length intervals, and that the number of failures occurring at interval  $i$ ,  $N_i$ , follows a Poisson distribution with intensity rate  $Dk^{i-1}$ . The probability of getting  $m$  failures at the  $i^{th}$  interval is

$$\Pr\{N_i = m\} = \frac{e^{-Dk^{i-1}} (Dk^{i-1})^m}{m!} \quad (5.38)$$

The reliability and other performance measures can be easily derived in the same manner as in the J-M model.

### Negative-binomial Poisson Model

Assume that the intensity  $\lambda$  is a random variable with the gamma density function having parameters  $k$  and  $m$ , that is,

$$f(\lambda) = \frac{1}{\Gamma(m)} k^m \lambda^{m-1} e^{-k\lambda} \quad \lambda \geq 0 \quad (5.39)$$

then the probability that there are exactly  $n$  software failures occurring during the time interval  $(0, t)$  is given by

$$P\{N(t) = n\} = \binom{n+m-1}{n} p^m q^n \quad n = 0, 1, 2, \dots \quad (5.40)$$

where

$$p = \frac{k}{t+k} \quad \text{and} \quad q = \frac{t}{t+k} = 1-p$$

This probability is also called a negative binomial density function.

### Modified Schick-Wolverton Model

Sukert (1977) modifies the S-W model to allow more than one failure at each time interval. The software failure rate function is given by

$$\lambda(t_i) = \phi[N - n_{i-1}]t_i \quad (5.41)$$

where  $n_{(i-1)}$  is the cumulative number of failures at the  $(i-1)^{th}$  failure interval. Thus, the software reliability function is

$$R(t_i) = e^{-\phi[N - n_{i-1}] \frac{t_i^2}{2}} \quad (5.42)$$

### Goel-Okumoto Imperfect Debugging Model

Goel and Okumoto (1979b) extend the J-M model by assuming that a fault is removed with probability  $p$  whenever a failure occurs. The failure rate function of the J-M model with imperfect debugging at the  $i^{th}$  failure interval becomes

$$\lambda(t_i) = \phi[N - p(i-1)] \quad (5.43)$$

The reliability function is

$$R(t_i) = e^{-\phi[N - p(i-1)]t_i} \quad (5.44)$$

It should be noted that

$$\begin{aligned} \lambda(t_i) &= \phi[N - p(i-1)] \\ &= p\phi\left[\frac{N}{p} - (i-1)\right] \\ &= \phi'[N' - (i-1)] \end{aligned}$$

is the same as in the J-M model where

$$\phi' = p\phi \quad \text{and} \quad N' = \frac{N}{p}$$

## 5.6 Curve Fitting Models

The curve fitting group models uses statistical regression analysis to study the relationship between software complexity and the number of faults in a program, the number of changes, or failure rate. This group of models finds a functional relationship between dependent and independent variables by using the methods of

linear regression, nonlinear regression, or time series analysis. The dependent variables, for example, are the number of errors in a program. The independent variables are the number of modules changed in the maintenance phase, time between failures, programmers' skill, program size, *etc.* Models included in this group are

- Estimation of errors
- Estimation of complexity
- Estimation of failure rate

### Estimation of Errors Model

The number of errors in a program can be estimated by using a linear or nonlinear regression model. A simple nonlinear regression model to estimate the total number of initial errors in the program,  $N$ , can be presented as follows:

$$N = \sum_i a_i X_i + \sum_i b_i X_i^2 + \sum_i c_i X_i^3 + \varepsilon \quad (5.45)$$

where  $X_i$  is the  $i^{th}$  error factor;  $a_i$ ,  $b_i$ ,  $c_i$  are the coefficients of the model, and  $\varepsilon$  is an error term.

Typical error factors are software complexity metrics and environmental factors. Most curve fitting models involve only one error factor. Several reliability models with environmental factors will be further discussed in Chapter 8.

### Estimation of Complexity Model

Belady and Lehman (1976) proposed a model to estimate the software complexity,  $C_R$ , using the time series approach. The software complexity model is summarized as follows:

$$C_R = a_0 + a_1 R + a_2 E_R + a_3 M_R + a_4 I_R + a_5 D + \varepsilon \quad (5.46)$$

where

- $R$  = release sequence number
- $E_R$  = environmental factor(s) at release  $R$
- $M_R$  = number of modules at release  $R$
- $I_R$  = inter-release interval  $R$
- $D$  = number of days since first release error
- $\varepsilon$  = error

This model is applicable for software having multiple release versions and evolving over a long period of time.

### Estimation of Failure Rate Model

Miller (1985) proposed a model to estimate the failure rate of software. Given failure times  $t_1, t_2, \dots, t_n$ , a rough estimate of the failure rate at the  $i^{th}$  failure interval is

$$\hat{\lambda}_i = \frac{1}{t_{i-1} - t_i} \quad (5.47)$$

Assuming that the failure rate is monotonically non-increasing, an estimate of this function  $\lambda_i^*$ ,  $i = 1, 2, \dots, n$  can be obtained by using the least squared method (see Miller 1985).

## 5.7 Reliability Growth Models

The reliability growth group of models measures and predicts the improvement of reliability programs through the testing process. The growth model represents the reliability or failure rate of a system as a function of time or the number of test cases. Models included in this group are

- Coutinho model (Coutinho 1973)
- Wall and Ferguson model (Wall 1977).

### Coutinho Model

Coutinho (1973) adapted the Duane growth model to represent the software testing process. Coutinho plotted the cumulative number of deficiencies discovered and the number of correction actions made vs the cumulative testing weeks on log-log paper. Let  $N(t)$  denote the cumulative number of failures and let  $t$  be the total testing time. The failure rate,  $\lambda(t)$ , model can be expressed as

$$\begin{aligned}\lambda(t) &= \frac{N(t)}{t} \\ &= \beta_0 t^{-\beta_1}\end{aligned}\tag{5.48}$$

where  $\beta_0$  and  $\beta_1$  are the model parameters. The least squares method can be used to estimate the parameters of this model.

### Wall and Ferguson Model

Wall and Ferguson (1977) proposed a model similar to the Weibull growth model for predicting the failure rate of software during testing. The cumulative number of failures at time  $t$ ,  $m(t)$ , can be expressed as

$$m(t) = a_0 [b(t)]^\beta\tag{5.49}$$

where  $a_0$  and  $\beta$  are the unknown parameters. The function  $b(t)$  can be obtained as the number of test cases or total testing time. Similarly, the failure rate function at time  $t$  is given by

$$\lambda(t) = m'(t) = a_0 \beta b'(t) [b(t)]^{\beta-1}\tag{5.50}$$

Wall and Ferguson (1977) tested this model using several software failure data and observed that failure data correlate well with the model.

## 5.8 Markov Structure Models

A Markov process has the property that the future behavior of the process depends only on the current state and is independent of its past history (see Chapter 2). The Markov structure group of models is a general way of representing the failure process of software. This group of models can also be used to study the reliability and interrelationship of the modules. It is assumed that failures of the modules are independent of each other. This assumption seems reasonable at the module level since they can be designed, coded and tested independently, but may not be true at the system level. Models included in this group are:

- Markov model with imperfect debugging (Goel 1979b)
- Littlewood Markov (Littlewood 1979)
- Software safety (Tokuno 1997; Yamada 1998)

### Markov Model with Imperfect Debugging

Goel and Okumoto (1979b) proposed a linear Markov model with imperfect debugging and the transition probabilities of the model can be expressed as

$$P_{ij} = \begin{cases} p & \text{for } j = i - 1 \\ q & \text{for } j = i \\ 1 & \text{for } j = i = 0 \\ 0 & \text{otherwise} \end{cases}$$

where  $p$  is the probability of successful debugging and  $q = 1 - p$  for  $i, j = 0, 1, \dots, N$ . In other word,  $q$  is the probability of unsuccessfully debugging the fault whenever a failure occurs. The reliability function of the  $k^{\text{th}}$  failure interval is given by Goel (1979):

$$R_k(t) = \sum_{j=0}^{k-1} \binom{k-1}{j} p^{k-j-1} q^j e^{-[N-(k-j-1)]\phi t} \quad (5.51)$$

### Littlewood Markov Model

Littlewoods model (Littlewood 1979) represents the transitions between program modules during execution as the Markov process. Two types of failures are considered in the model. The first type of failure comes from a Poisson failure process at each module. It is recognized that new errors will be introduced as modules are integrated. The second type of failure is the interface between modules. Let

$n$  = number of modules

$a_{ij}$  = transition process from module  $i$  to module  $j$

$\lambda_j$  = Poisson failure rate of module  $i$

$q_{ij}$  = probability that transition from module  $i$  to module  $j$  fails

$\pi_i$  = limiting distribution of the process

Assuming that failures at modules and interfaces are independent of each other, Littlewood (1979) has shown that the program failure process is asymptotically a Poisson process with failure rate

$$\sum_{i=1}^n \pi_i (\lambda_i + \sum_{i \neq j} a_{ij} q_{ij})$$

as  $\lambda_i$  and  $q_{ij}$  approach zero.

### Software Safety Model

Yamada *et al.* (1998) proposed a software safety model to describe the time-dependent behavior of the software using Markov processes. The assumptions in this safety model include:

- When the software system is operating, the holding times of the safety and the unsafe state follow exponential distributions with means  $1/\theta$  and  $1/\eta$ , respectively.
- A debugging activity is performed when a software failure occurs. Debugging activities are perfect with probability  $a$ , while they are imperfect with probability  $b$ .
- Software reliability growth occurs in cases of perfect debugging. The time interval between software failure occurrences follows an exponential distribution with mean  $1/\lambda_n$  where  $n = 0, 1, 2, \dots$  denotes the cumulative number of corrected faults.
- The probability that two or more software failures occur simultaneously is negligible.

Consider a stochastic process  $\{X(t), t \geq 0\}$  representing the state of the software system at time  $t$ . The state space of  $\{X(t), t \geq 0\}$  is defined as

$W_n$  = the system is operating safety

$U_n$  = the system falls into the unsafe state.

Yamada *et al.* (1998) use Moranda's model to describe the software reliability growth process. When  $n$  faults have been corrected, the failure intensity for the next software failure occurrence  $\lambda_n$  is

$$\lambda_n = Dk^n$$

where  $D > 0$  and  $0 < k < 1$  are the initial failure rate and the decreasing ratio of the failure rate, respectively.

Software safety is defined as the probability that the system does not fall into any unsafe states at time point  $t$  and is given as follows (Yamada 1998):

$$S(t) = \sum_{n=0}^{\infty} p_n(t)$$

where

$$\begin{aligned} P_n(t) &= P\{X(t) = W_n\} \\ &= A_n e^{-(\lambda_n + \theta + \eta)t} + \sum_{i=0}^n B_{ni} e^{-a\lambda_i t} \end{aligned}$$

and the constant coefficients  $A_n$  and  $B_{ni}$  are given by

$$A_n = \frac{-\theta \prod_{j=0}^{n-1} a\lambda_j}{\prod_{j=0}^n (a\lambda_j - \lambda_n - \theta - \eta)}$$

$$B_{ni} = \frac{(\lambda_n + \eta - a\lambda_i) \prod_{j=0}^{n-1} \lambda_j}{(\lambda_n + \theta + \eta - a\lambda_i) \prod_{j=0 \& j \neq i}^n (\lambda_j - \lambda_i)}$$

## 5.9 Time Series Models

In this section, we discuss a general time series model, called Autoregressive Integrated Moving Average (ARIMA), that was studied by Box and Jenkins (Box et al. 1994).

The ARIMA models of order  $(p, d, q)$  can be expressed as follows:

$$z_t = \delta + \phi_1 z_{t-1} + \phi_2 z_{t-2} + \dots + \phi_p z_{t-p} \quad (5.52)$$

$$+ a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} - \dots - \theta_q a_{t-q}$$

where

$\delta$  = mean of  $z_t$ ,  $t = 0, 1, 2, \dots$

$z_t$  = observed value at time  $t$

$\hat{z}_t$  = the expected value of  $z_t$

$a_t = z_t - \hat{z}_t$

= white noise at the  $t^{th}$  time period; its value follows the normal distribution with mean 0

$p$  = the order number of autoregression (AR)

$d$  = the value of difference order of original data

$q$  = the order number of moving average (MA)

Assume  $y_1, y_2, \dots, y_n$  denote the number of cumulative failure of software during testing in unit time  $1, 2, \dots, n$ . If the  $n$  values do not fluctuate around a constant mean or do not fluctuate with constant variance, then this process is called a non-stationary time series process. In other words, a time series is *stationary* if the statistical properties of the time series are essentially constant through time.

In the non-stationary case, one can take the first or second difference of the values and examine if the transformed series is a stationary case.



Let  $z_t$  be the number of failures in the  $t^{th}$  testing period, then  $z_t$  can be written as

$$z_t = y_t - y_{t-1} \quad \text{where } t = 2, \dots, n \quad (5.53)$$

and  $w$  is the first difference of  $z$  (or the second difference of  $y_1, y_2, \dots, y_n$ ), then we obtain as follows:

$$\begin{aligned} w_t &= z_t - z_{t-1} = (y_t - y_{t-1}) - (y_{t-1} - y_{t-2}) \\ &= y_t - 2y_{t-1} + y_{t-2} \quad \text{where } t = 3, 4, \dots, n. \end{aligned} \quad (5.54)$$

From equation (5.52), we can easily obtain the ARIMA function in term of  $y_1, y_2, \dots, y_n$  using equation (5.53). For example, consider  $p=0$ ,  $d=2$  and  $q=3$ . Then the ARIMA for  $y$ , the values of cumulative failure number can be calculated as follows.

It can be expressed that

$$w_t = \delta + a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} - \theta_3 a_{t-3}$$

and, therefore, we obtain

$$y_t - 2y_{t-1} + y_{t-2} = \delta + a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} - \theta_3 a_{t-3} \quad (5.55)$$

Equation (5.55) indicates that the second differences  $\{w_t\}$  constitute a series of moving linear combinations of  $\{a_{t-3}, a_{t-2}, a_{t-1}, a_t\}$  weighted by the weight function  $\{-\theta_3, -\theta_2, -\theta_1, 1\}$  and can be used to estimate as well as predict the reliability of the software.

## 5.10 Non-homogeneous Poisson Process Models

The non-homogeneous Poisson Process (NHPP) group of models provides an analytical framework for describing the software failure phenomenon during testing. The main issue in the NHPP model is to estimate the mean value function of the cumulative number of failures experienced up to a certain point in time. Models included in this group are

- Musa exponential (Musa 1987)
- Goel and Okumoto NHPP (Goel 1979a)
- S-shaped growth (Ohba, 1984b; Yamada 1983, 1984)
- Hyperexponential growth (Huang 1984; Ohba 1984a)
- Discrete reliability growth (Yamada 1985)
- Testing-effort dependent reliability growth (Yamada 1986, 1993)
- Generalized NHPP (Pham 1997a, 1999b, 2000a, 2003a)

In Chapter 2, the NHPP represents the number of failures experienced up to time  $t$  as an NHPP,  $\{N(t), t \geq 0\}$ . The main issue in the NHPP model is to determine an appropriate mean value function to denote the expected number of failures

experienced up to a certain point in time. With different assumptions, the model will result with different functional forms of the mean value function.

Based on the NHPP assumptions in Chapter 2, it can be shown that  $N(t)$  has a Poisson distribution with mean  $m(t)$ , i.e.,

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

By definition, the mean value function of the cumulative number of failures,  $m(t)$ , can be expressed in terms of the failure intensity function of the software, i.e.,

$$m(t) = \int_0^t \lambda(s) ds \quad (5.56)$$

The reliability function of the software is

$$R(t) = e^{-m(t)} = e^{-\int_0^t \lambda(s) ds} \quad (5.57)$$

Goel and Okumoto's NHPP model (Goel 1979a, 1980) belongs to this class. Other types of mean value functions suggested by Ohba (1984a), Yamada and Osaki (1985), Pham and Zhang (1997a), and Pham (1999b, 2000a) are the delayed S-shaped growth model, inflection S-shaped growth model, and hyperexponential growth model. We will further discuss NHPP models in Chapter 6.

## 5.11 Further Reading

Some interesting papers related to the subject discussed in this chapter are:

Cai K-Y, (1998) "On estimating the number of defects remaining in software," *Journal of Systems and Software*, vol. 40(1)

Boehm BW, (1981) *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs

Malaiya YK, Karunanithi N, Verma P, (1992) "Predictability of software-reliability models," *IEEE Transactions on Reliability*. vol. 41 no. 4, pp 539-546

Pham H, (1999a) "Software Reliability," a chapter in *Wiley Encyclopedia of Electrical and Electronic Engineering*, Editor: John Webster, Wiley: pp 565-578

## 5.12 Problems

1. Figure P.4.1 (in Pham 2000a, page 101) illustrates a program used in a toaster. The program scans the lever switch until a user pushes down the bread to be toasted. The heater is energized. The heat time is determined from the lightdark switch on the side of the toaster. After the time is input in Step

5, it is checked against valid times. If the number is less than 1 or greater than 10, the heat is turned off and the toast pops up because the input is incorrect. This time number should be an integer between one and ten. A value of one dictates a short heating time of 10 seconds, resulting in light toast. Higher values dictate longer heating times. The program decreases the time number until the remaining time equals zero. The heater is then turned off and the toast pops up. The number of program control flow paths equals three. Determine the number of control flow paths in the program shown in Figure P.4.1 (Pham 2000a).

2. Suppose that the failure data  $\{t_1, t_2, \dots, t_n\}$  is given. Find the maximum likelihood estimates for the parameters  $N$  and  $\phi$  of the modified Schick Wolverton model.
3. Show that the mean time to the next failure of the S -W model is given by

$$MTTF_i = \sqrt{\frac{\pi}{2\phi(N-i+1)}}$$

4. Derive equations (5.21) and (5.22).
5. Assume that a new fault is introduced during the removal of a detected fault with a probability  $q$ . Determine the probability function of removing  $k$  induced errors and  $r-k$  indigenous errors in  $m$  tests.
6. Using the Naval Tactical Data Systems (NTDS) failure data (data set #4 in Chapter 4):
  - (a) Calculate the maximum likelihood estimates for the parameters of the Jelinski-Moranda (J-M) model based on all available NTDS data.
  - (b) Choose another software reliability model, other than NHPP models, and repeat question (a). Is the new model better or worse? Explain and justify your results.

## Imperfect-debugging Models

### 6.1 Introduction

Since computers are being used increasingly to monitor and control both safety-critical and civilian systems, there is a great demand for high-quality software products. Reliability is a primary concern for both software developers and software users.

Research activities in software reliability engineering have been conducted and a number of NHPP software reliability growth models have been proposed to assess the reliability of software. In fact, software reliability models based on the NHPP have been quite successful tools in practical software reliability engineering. These models consider the debugging process as a counting process characterized by its mean value function. Software reliability can be estimated once the mean value function is determined. Model parameters are usually estimated using either the maximum likelihood method or regression. Different models have been built upon different assumptions.

Software reliability assessment is increasingly important in developing and testing new software products. Before newly developed software is released to the user, it is extensively tested for errors that may have been introduced during development. Although detected errors are removed immediately, new errors may be introduced during debugging. Software that contains errors and is released to the market incurs high failure costs. Debugging and testing, on the other hand, reduces the error content but increases development costs. Thus, there is a need to determine the optimal time to stop software testing. During system testing, reliability measure is an important criterion in deciding when to release the software. Several other criteria, such as the number of remaining errors, failure rate, reliability requirements, or total system cost, may be used to determine optimal testing time.

This chapter discusses stochastic reliability models for the software failure phenomenon based on a non-homogeneous Poisson process (NHPP). Allowing both the error content function and the error detection rate to be time-dependent, a generalized software reliability model and an analytical expression for the mean value function are presented. Numerous existing models based on NHPP are also summarized. Several applications and numerical examples are included to illustrate

the results. A general function for calculating the mean time between failures (MTBF) of software systems based on the NHPP is also presented. An NHPP is a realistic model for predicting software reliability and has a very interesting and useful interpretation in debugging and testing the software.

### Notation

$m(t)$	Expected number of errors detected by time $t$ ("mean value function")
$a(t)$	Error content function, <i>i.e.</i> , total number of errors in the software including the initial and introduced errors at time $t$
$b(t)$	Error detection rate per error at time $t$
$N(t)$	Random variable representing the cumulative number of software errors detected by time $t$
$y(t)$	Actual values of $N(t)$ ( $y_i := y(t_i)$ )
$S_j$	Actual time at which the $j$ th error is detected
$R(s t)$	Reliability during $(t, t + s)$ given that the last error occurred at time $t$

## 6.2 Parameter Estimation

Parameter estimation is of primary importance in software reliability prediction. Once the analytical solution for  $m(t)$  is known for a given model, the parameters in the solution need to be determined. Parameter estimation is achieved by applying a technique of MLE, the most important and widely used estimation technique. In many cases, the maximum likelihood estimators are consistent and asymptotically normally distributed as the sample size increases (Zhao 1996). In this chapter, we only discuss the MLE technique to estimate the unknown parameters for the software reliability models. Depending on the format in which test data are available, two different approaches are frequently used. A set of failure data is usually collected in one of two common ways (see Chapter 4) and is discussed next.

### Type I Data: Interval Domain Data

Assuming that the data are given for the cumulative number of detected errors  $y_i$  in a given time-interval  $(0, t_i)$  where  $i = 1, 2, \dots, n$  and  $0 < t_1 < t_2 < \dots < t_n$ , then the log likelihood function (LLF) takes on the following form:

$$LLF = \sum_{i=1}^n (y_i - y_{i-1}) \cdot \log[m(t_i) - m(t_{i-1})] - m(t_n) \quad (6.1)$$

Thus the maximum of the LLF is determined by the following system of equations:

$$0 = \sum_{i=1}^n \frac{\frac{\partial}{\partial \theta} m(t_i) - \frac{\partial}{\partial \theta} m(t_{i-1})}{m(t_i) - m(t_{i-1})} (y_i - y_{i-1}) - \frac{\partial}{\partial \theta} m(t_n) \quad (6.2)$$

where for  $\theta$  each of the unknown parameters is to be substituted. Using the observed failure data  $(t_i, y_i)$  for  $i = 1, 2, \dots, n$ , we can use the mean value function  $m(t_i)$  to determine the expected number of errors to be detected by time  $t_i$  for  $i = n + 1, n + 2$ , *etc.*

### Type 2 Data: Time Domain Data

Assuming that the data are given for the occurrence times of the failures or the times of successive failures, *i.e.*, the realization of random variables  $S_j$  for  $j = 1, 2, \dots, n$ . Given that the data provide  $n$  successive times of observed failures  $s_j$  for  $0 \leq s_1 \leq s_2 \leq \dots \leq s_n$ , we can convert these data into the time between failures  $x_i$  where  $x_i = s_i - s_{i-1}$  for  $i = 1, 2, \dots, n$ . Given the recorded data on the time of failures, the log likelihood function takes on the following form:

$$LLF = \sum \log[\lambda(s_i)] - m(s_n) \quad (6.3)$$

The MLE of unknown parameters  $\theta = (\theta_1, \theta_2, \dots, \theta_n)$  can be obtained by solving the following equations:

$$0 = \sum_{i=1}^n \frac{\frac{\partial}{\partial \theta} \lambda(S_i)}{\lambda(S_i)} - \frac{\partial}{\partial \theta} m(S_n) \quad (6.4)$$

where

$$\lambda(t) = \frac{\partial}{\partial t} m(t)$$

and for  $\theta$  each of the unknown parameters is to be substituted.

In general, the equations to be solved for the MLE of the system parameters are nonlinear. One can easily obtain the MLE of unknown parameters for an arbitrary mean value function of a given set of test data by using the iterative Newton method. One can also use Microsoft Excel to obtain the MLE unknown parameters by directly maximizing the likelihood function from either equation (6.1) or equation (6.3).

## 6.3 Model Selection

Once the analytical expression for the mean value function  $m(t)$  is derived, the parameters of the mean value function need to be estimated, which is usually carried out by using the MLE method discussed in Section 6.2.

There are four common criteria, such as the sum of squared errors (SSE), mean squared errors (MSE), the Akaike's information criterion (AIC), and predictive-ratio risk (PRR), that are commonly used for the model comparison of goodness-of-fit (descriptive power) and predictive power and we will use them for model illustrations.

**SSE criteria:** SSE can be calculated as follows:

$$SSE = \sum_{j=1}^k \sum_{i=1}^n [y_{ij} - \hat{m}_j(t_i)]^2 \quad (6.5)$$

where  $y_{ij}$  is total number of type  $j$  failures observed at time  $t_i$  according to the actual data and  $\hat{m}_j(t_i)$  is the estimated cumulative number of type  $j$  failures at time  $t_i$  for  $i=1, 2, \dots, n$  and  $j=1, 2, \dots, k$ .

**MSE criteria:** MSE is calculated as follows:

$$\text{MSE} = \frac{\sum_{j=1}^k \sum_{i=1}^n (m_j(t_i) - y_{ij})^2}{k \cdot n - N} \quad (6.6)$$

The MSE measures the distance of a model estimate from the actual data with the consideration of the number of observations and the number of parameters ( $N$ ) in the model.

**AIC criteria (Akai 1974):** AIC is calculated as follows:

$$\text{AIC} = -2 \cdot \log(\text{likelihood function at its maximum value}) + 2N \quad (6.7)$$

where  $N$  represents the number of parameters in the model. The AIC measures the ability of a model to maximize the likelihood function that is directly related to the degrees of freedom during fitting, increasing the number of parameters will usually result in a better fit. AIC criterion takes the degree of freedom into consideration by assigning a model with more parameters a larger penalty.

**PRR criteria (Pham 2003a):** The predictive-ratio risk (PRR) is defined as follows:

$$\text{PRR} = \sum_{j=1}^k \sum_{i=1}^n \left( \frac{m_j(\hat{t}_i) - y_{ij}}{m_j(\hat{t}_i)} \right)^2 \quad (6.8)$$

where  $y_{ij}$  is total number of type  $j$  failures observed at time  $t_i$  according to the actual data and  $m_j(t_i)$  is the estimated cumulative number of type  $j$  failures at time  $t_i$  for  $i=1,2,\dots,n$  and  $j=1,2,\dots,k$ .

It is worth noting that the PRR criterion uses the risk-of-underestimation by assigning a larger penalty to a model that has underestimated the cumulative number of failures. PRR measures the distance of model estimates from the actual data against with the model estimate.

For all these four criteria – PRR, MSE, SSE, and AIC – the smaller the value, the better the model fits, relative to other models run on the same data set.

## 6.4 NHPP Exponential Models

In this section we will describe various types of NHPP SRGM models.

### Goel-Okumoto Model

The Goel-Okumoto model (also called as exponential NHPP model) is based on the following assumptions:

1. All faults in a program are mutually independent from the failure detection point of view.
2. The number of failures detected at any time is proportional to the current number of faults in a program. This means that the probability of the failures for faults actually occurring, *i.e.*, detected, is constant.
3. The isolated faults are removed prior to future test occasions.
4. Each time a software failure occurs, the software error which caused it is immediately removed, and no new errors are introduced.

This is shown in the following differential equation:

$$\frac{\partial m(t)}{\partial t} = b[a - m(t)] \quad (6.9)$$

where  $a$  is the expected total number of faults that exist in the software before testing and  $b$  is the failure detection rate or the failure intensity of a fault.

**Theorem 6.1 (Goel 1979a):** The mean value function solution of the differential equation (6.9) is given by

$$m(t) = a(1 - e^{-bt}) \quad (6.10)$$

This model is known as the **Goel-Okumoto model** (Goel 1979a).

For Type-I data, the estimate of parameters  $a$  and  $b$  of the Goel-Okumoto model using the MLE method discussed in Section 6.2, can be obtained by solving the following equations simultaneously:

$$\begin{aligned} a &= \frac{y_n}{(1 - e^{-bt_n})} \\ \frac{y_n t_n e^{-bt_n}}{1 - e^{-bt_n}} &= \sum_{k=1}^n \frac{(y_k - y_{k-1})(t_k e^{-bt_k} - t_{k-1} e^{-bt_{k-1}})}{(e^{-bt_{k-1}} - e^{-bt_k})} \end{aligned} \quad (6.11)$$

Similarly, for Type-II data, the estimate of parameters  $a$  and  $b$  using the MLE method can be obtained by solving the following equations:

$$\begin{aligned} a &= \frac{n}{(1 - e^{-bS_n})} \\ \frac{n}{b} &= \sum_{i=1}^n s_i + \frac{nS_n e^{-bS_n}}{1 - e^{-bS_n}} \end{aligned} \quad (6.12)$$



Let  $\hat{a}$  and  $\hat{b}$  be the MLE of parameters  $a$  and  $b$ , respectively. We can then obtain the MLE of the mean value function (MVF) and the reliability function as follows:

$$\hat{m}(t) = \hat{a}[1 - e^{-\hat{b}t}]$$

$$\hat{R}(x|t) = e^{-\hat{a}[e^{-\hat{b}t} - e^{-\hat{b}(t+x)}]}$$

It is of interest to determine the variability of the number of failures at time  $t$ ,  $N(t)$ . One can approximately obtain the confidence intervals for  $N(t)$  based on the Poisson distribution as

$$\hat{m} - z_\alpha \sqrt{\hat{m}(t)} \leq N(t) \leq \hat{m}(t) + z_\alpha \sqrt{\hat{m}(t)} \quad (6.13)$$

where  $z_\alpha$  is  $100(1 + \alpha)/2$  percentile of the standard normal distribution, *i.e.*,  $N(0, 1)$ .

*Example 6.1:* The data set #10 (in Table 4.14, Chapter 4) was reported by Musa (1987) based on failure data from a real-time command and control system, which represents the failures observed during system testing for 25 hours of CPU time. The delivered number of object instructions for this system was 21700 and was developed by Bell Laboratories.

It should be noted that this data set belongs to a concave class, therefore, it seems reasonable to use the Goel-Okumoto NHPP model to describe the failure process of the software system. From the failure data, the two unknown parameters,  $a$  and  $b$ , can be obtained using equation (6.12) and the estimated values for the two parameters are

$$\hat{a} = 142.3153 \quad \hat{b} = 0.1246$$

Recall that  $\hat{a}$  is an estimate of the expected total number of failures to be eventually detected and  $\hat{b}$  represents the number of faults detected per fault per unit time (hour). The estimated mean value function and software reliability function, respectively, are

$$\hat{m}(t) = 142.3153(1 - e^{-0.1246t})$$

and

$$\hat{R}(x|t) = e^{-(142.3153)[e^{(0.1246)t} - e^{-(0.1246)(t+x)}]}$$

The above two functions can be used to determine when to release the software system or the additional testing effort required when the system is ready for release. Let us assume that failure data from only 16 hours of testing are available and from the data set #10 (Table 4.14, Chapter 4), a total of 122 failures have been observed. Based on these data and using the MLE method, the estimated values for the two parameters are

$$\hat{a} = 138.3779 \quad \text{and} \quad \hat{b} = 0.1334$$

and the estimated mean value function becomes

$$\hat{m}(t) = 138.3779(1 - e^{-0.1334t})$$

The reliability of the software system is

$$\hat{R}(x|t) = e^{-(138.3779)[e^{(0.1334)t} - e^{-(0.1334)(t+\epsilon)}]}$$

An estimate number of remaining errors after 16 hours of testing is 16.38 with a 90% confidence interval of (4.64, 28.11). Similarly, the estimated current software reliability for the next hour is 0.129 and the corresponding 90% confidence interval is (0.019, 0.31).

Next, suppose the problem of interest is to know how much additional testing is needed in order to achieve an acceptable number of remaining errors so that the software can be released for operational use. For example, we would want to release the software if the expected number of remaining errors is less than or equal to 10. In the above analysis, we learned that the best estimate of the remaining errors in the software after 16 hours of testing is about 17. Therefore, testing has to continue in the hope that additional faults can be detected. If we were to carry on a similar task after each additional hour of testing, we can expect to obtain another seven additional errors during the next 4 hours (see Table 6.1). In other words, the expected number of remaining errors after 20 hours would be 9.8 so that the above objective would be met.

**Table 6.1.** Software reliability performance measures

Testing time T	a	b	Remaining errors	Reliability $R(0.1/T)$
16	138.3779	0.1333	16.4	0.8049
17	133.7050	0.1432	11.7	0.8466
18	141.2543	0.1274	14.3	0.8349
19	139.7190	0.1304	11.7	0.8591
20	138.8495	0.1323	9.8	0.8786
21	140.3408	0.1290	9.3	0.8871
22	140.1002	0.1296	8.1	0.9010
23	141.9104	0.1255	7.9	0.9060
24	142.0264	0.1252	7.0	0.9162
25	142.3153	0.1246	6.3	0.9248

### Musa Exponential Model

Musa (1985) proposed a similar model to the Goel-Okumoto model by considering the relationship between execution time and calendar time. Let  $m(t)$  be the number of failures discovered as a result of test case runs up to the time of observation. Musa obtained the differential equation as follows:

$$\frac{\partial m(t)}{\partial t} = \frac{c}{nT}[a - m(t)] \quad (6.14)$$

where

- $a$  = number of failures in the program
- $c$  = the testing compression factor
- $T$  = mean time to failure at the beginning of the test
- $n$  = total number of failures possible during the maintained life of the program

$t$  = execution time or the total CPU time utilized to complete the test case runs up to a time of observation.

**Theorem 6.2 (Musa 1985):** The mean value function of the differential equation (6.14) can be easily solved as follows:

$$m(t) = a(1 - e^{-\frac{ct}{nT}}) \quad (6.15)$$

This model is often called the **Musa exponential model**.

The failure intensity function is

$$\lambda(t) = \frac{c}{nT}(a - m(t)) \quad (6.16)$$

The reliability function and pdf, respectively, are

$$R(t) = e^{-a(1 - e^{-\frac{c}{nT}t})} \quad (6.17)$$

and

$$f(t) = \frac{c}{nT} a e^{-\frac{c}{nT}t} e^{-a(1 - e^{-\frac{c}{nT}t})} \quad (6.18)$$

Suppose we have observed  $k$  failures of the software and suppose that the failure data set  $\{t_1, t_2, \dots, t_k\}$  is given where  $t_i$  is the observed time between the  $(i-1)^{\text{th}}$  and the  $i^{\text{th}}$  failure. Here we want to estimate the unknown parameters  $a$  and  $c$ . Using the MLE method, the likelihood function is obtained as

$$\begin{aligned} L(a, c) &= \prod_{i=1}^k f(t_i) \\ &= \left(\frac{ac}{nT}\right)^k \left( e^{-\frac{c}{nT} \sum_{i=1}^k t_i} \right) e^{-a \sum_{i=1}^k (1 - e^{-\frac{c}{nT}t_i})} \end{aligned}$$

The log likelihood function is given by

$$\ln L = k \ln\left(\frac{ac}{nT}\right) - \frac{c}{nT} \sum_{i=1}^k t_i - a \sum_{i=1}^k [1 - e^{-\frac{c}{nT}t_i}] \quad (6.19)$$

The first derivative of the log likelihood function with respect to the unknown parameters  $c$  and  $a$  are

$$\frac{\partial}{\partial c} \ln L = kc \frac{1}{nT} \sum_{i=1}^k t_i - \frac{a}{nT} \sum_{i=1}^k t_i e^{-\frac{c}{nT}t_i} \equiv 0 \quad (6.20)$$

and

$$\frac{\partial}{\partial a} \ln L = \frac{k}{a} - \sum_{i=1}^k (1 - e^{-\frac{c}{nT}t_i}) \equiv 0 \quad (6.21)$$

Thus,  $a$  and  $c$  can be obtained by solving the following two equations simultaneously:

$$c = \frac{1}{k n T} \sum_{i=1}^k t_i + \frac{a}{k n T} \sum_{i=1}^k t_i e^{-\frac{c}{nT} t_i}$$

and

$$a = \frac{k}{\sum_{i=1}^k (1 - e^{-\frac{c}{nT} t_i})} \quad (6.22)$$

### Hyperexponential Growth Model

The hyperexponential growth model (Ohba 1984a) is based on the assumption that a program has a number of clusters of modules, each having a different initial number of errors and a different failure rate. Examples are new modules vs reused modules, simple modules vs complex modules, and modules which interact with hardware vs modules which do not. It should be noted that the sum of exponential distributions becomes a hyperexponential distribution.

**Theorem 6.3 (Ohba 1984a):** Assume that a program has a number of clusters of modules and each having a different initial number of errors and a different failure intensity function. The mean value function of the hyperexponential class NHPP model is

$$m(t) = \sum_{i=1}^n a_i [1 - e^{-b_i t}] \quad (6.23)$$

where

$n$  = number of clusters of modules

$a_i$  = number of initial faults in cluster  $i$

$b_i$  = failure rate of each fault in cluster  $i$

The failure intensity function is given by

$$\lambda(t) = \sum_{i=1}^n a_i b_i e^{-b_i t}$$

### Yamada-Osaki Exponential Growth Model

A similar extension of the exponential growth model has been suggested by Yamada and Osaki (1985) by dividing the software into  $k$  modules.

**Theorem 6.4 (Yamada 1985):** The failure intensity of faults within different modules are assumed to be different while the failure intensity of faults within the same module are assumed to be the same. Assume that the expected number of faults detected for each module are exponential. The expected number of faults detected for the entire software can be obtained as

$$m(t) = a \sum_{i=1}^k p_i [1 - e^{-b_i t}] \quad (6.24)$$

where

$k$  = number of modules in the software

$b_i$  = error detection rate of one fault within the  $i^{\text{th}}$  module

$p_i$  = probability of faults for the  $i^{\text{th}}$  module

$a$  = expected number of software errors to be eventually detected or total number of faults existing in the software before testing.

This model is called the **Yamada-Osaki exponential growth model**.

For Type-I data, the MLEs of the parameters  $a$  and  $b_i$  for  $i = 1, 2, \dots, k$  can be obtained by solving the following equations simultaneously:

$$a = \frac{y_n}{\sum_{i=1}^k p_i (1 - e^{-bt_n})} \quad (6.25)$$

$$\frac{y_n t_n e^{-bt_n}}{\sum_{i=1}^k p_i (1 - e^{-bt_n})} = \sum_{i=1}^n \frac{(y_i - y_{i-1})(t_i e^{-bt_i} - t_{i-1} e^{-bt_{i-1}})}{\sum_{i=1}^k p_i (e^{-bt_{i-1}} - e^{-bt_i})} \quad (6.26)$$

Similarly, for Type-II data, the MLEs of the parameters  $a$  and  $b_i$  for  $i = 1, 2, \dots, k$  can be obtained by solving simultaneously the following two equations:

$$a = \frac{n}{\sum_{i=1}^k p_i (1 - e^{-b_i s_n})} \quad (6.27)$$

$$\frac{ns_n e^{-b_i s_n}}{\sum_{i=1}^k p_i (1 - e^{-b_i s_n})} = \sum_{j=1}^n \frac{(e^{-b_i s_j} - b_i s_j e^{-b_i s_j})}{\sum_{i=1}^k p_i b_i e^{-b_i s_j}} \quad (6.28)$$

## 6.5 NHPP S-shaped Model

In the NHPP S-shaped model, the software reliability growth curve is an S-shaped curve which means that the curve crosses the exponential curve from below and the crossing occurs once and only once. The detection rate of faults, where the error detection rate changes with time, become the greatest at a certain time after testing begins, after which it decreases exponentially. In other words, some faults are covered by other faults at the beginning of the testing phase, and before these faults are actually removed, the covered faults remain undetected. Yamada (1984) also determined that the software testing process usually involves a learning process where testers become familiar with the software products, environments, and software specifications. Several S-shaped models (Yamada 1984; Pham 1997a) such as delayed S-shaped, inflection S-shaped, *etc.*, will also be discussed in this section.

The NHPP S-shaped model is based on the following assumptions:

1. The error detection rate differs among faults.
2. Each time a software failure occurs, the software error which caused it is immediately removed, and no new errors are introduced.

**Theorem 6.5 (Ohba 1984b):** The mean value function solution of the following differential equation:

$$\frac{\partial m(t)}{\partial t} = b(t)[a - m(t)] \quad (6.29)$$

is given by

$$m(t) = a[1 - e^{-\int_0^t b(u) du}] \quad (6.30)$$

where

- $a$  = expected total number of faults that exist in the before testing
- $b(t)$  = failure detection rate per fault
- $m(t)$  = expected number of failures detected at time  $t$

### Inflection S-shaped Model

The inflection S-shaped model (Ohba 1984) is based on the dependency of faults by postulating the following assumptions:

1. Some of the faults are not detectable before some other faults are removed.
2. The probability of failure detection at any time is proportional to the current number of detectable faults in the software.
3. Failure rate of each detectable fault is constant and identical.
4. The isolated faults can be entirely removed.

**Theorem 6.6 (Ohba 1984b):** Assume

$$b(t) = \frac{b}{1 + \beta e^{-bt}} \quad (6.31)$$

where the parameters  $b$  and  $\beta$  represent the failure-detection rate and the inflection factor, respectively. The mean value function is given by

$$m(t) = \frac{a}{1 + \beta e^{-bt}} (1 - e^{-bt}) \quad (6.32)$$

This model is called the **inflection S-shaped model** (Ohba 1984b).

The function  $m(t)$  can be easily obtained by substituting  $b(t)$  from equation (6.31) into equation (6.30). The failure intensity function of the inflection S-shaped model is given by

$$\lambda(t) = \frac{ab(1 + \beta)e^{-bt}}{(1 + \beta e^{-bt})^2}$$

The expected number of remaining errors at time  $t$  is given by

$$m(\infty) - m(t) = \frac{a(1 + \beta)e^{-bt}}{(1 + \beta e^{-bt})} \quad (6.33)$$

For Type-I data, the estimate of parameters  $a$  and  $b$  for specified  $\beta$  using the MLE method can be obtained by solving the following equations simultaneously:

$$a = \frac{y_n(1 + \beta e^{-bt_n})}{(1 - e^{-bt_n})} \quad (6.34)$$

and

$$\begin{aligned} \sum_{i=1}^n (y_i - y_{i-1}) & \left( \frac{(t_i e^{-bt_i} - t_{i-1} e^{-bt_{i-1}})}{(e^{-bt_{i-1}} - e^{-bt_i})} + \frac{\beta t_i e^{-bt_i}}{(1 + \beta e^{-bt_i})} + \frac{\beta t_{i-1} e^{-bt_{i-1}}}{(1 + \beta e^{-bt_{i-1}})} \right) \\ & = \frac{y_n t_n e^{-bt_n} (1 - \beta + 2\beta e^{-bt_n})}{(1 - e^{-bt_n})(1 + \beta e^{-bt_n})} \end{aligned} \quad (6.35)$$

Similarly, for Type-II data, the estimate of parameters  $a$  and  $b$  for specified  $\beta$  using the MLE method can be obtained by solving the following two equations:

$$a = \frac{n(1 + \beta e^{-bs_n})}{(1 - e^{-bs_n})} \quad (6.36)$$

and

$$\frac{ns_n e^{-bs_n} (1 + \beta)}{(1 - e^{-bs_n})(1 + \beta e^{-bs_n})} = \frac{n}{\beta} - \sum_{i=1}^n s_i + 2 \sum_{i=1}^n \frac{\beta s_i e^{-bs_i}}{(1 + \beta e^{-bs_i})} \quad (6.37)$$

### ***NHPP Delayed S-shaped Model***

We now discuss a stochastic model for a software error detection process based on NHPP in which the growth curve of the number of detected software errors for the observed failure data is S-shaped, called delayed S-shaped NHPP model (Yamada 1984). The software error detection process described by an S-shaped curve can be characterized as a learning process in which test-team members become familiar with the test environment, testing tools, or project requirements, *i.e.*, their test skills gradually improve. The delayed S-shaped model is based on the following assumptions:

1. All faults in a program are mutually independent from the failure detection point of view.
2. The probability of failure detection at any time is proportional to the current number of faults in a software.
3. The proportionality of failure detection is constant.
4. The initial error content of the software is a random variable.
5. A software system is subject to failures at random times caused by errors present in the system.

6. The time between  $(i-1)^{\text{th}}$  and  $i^{\text{th}}$  failures depends on the time to the  $(i-1)^{\text{th}}$  failure.
7. Each time a failure occurs, the error which caused it is immediately removed and no other errors are introduced.

**Theorem 6.7 (Yamada 1984):** Assume

$$b(t) = \frac{b^2 t}{bt + 1} \quad (6.38)$$

where  $b$  is the error detection rate per error in the steady-state. The mean value function is given by

$$m(t) = a[1 - (1 + bt)e^{-bt}] \quad (6.39)$$

which shows an S-shaped curve.

This model is called the **delayed S-shaped NHPP model** for such an error detection process, in which the observed growth curve of the cumulative number of detected errors is S-shaped (Yamada 1984). The corresponding failure intensity function is

$$\lambda(t) = ab^2 te^{-bt}$$

The reliability of the software system is

$$\begin{aligned} R(s | t) &= e^{-[m(t+s) - m(t)]} \\ &= e^{-a[(1+bt)e^{-bt} - (1+b(t+s))e^{-b(t+s)}]} \end{aligned} \quad (6.40)$$

The expected number of errors remaining in the system at time  $t$ ,  $n(t)$ , is given by

$$\begin{aligned} n(t) &= m(\infty) - m(t) \\ &= a(1 + bt)e^{-bt} \end{aligned}$$

For Type-I data, the estimate of parameters  $a$  and  $b$  using the MLE method can be obtained by solving the following equations simultaneously:

$$a = \frac{y_n}{[1 - (1 + bt_n)e^{-bt_n}]} \quad (6.41)$$

and

$$\frac{y_n t_n^2 e^{-bt_n}}{[1 - (1 + bt_n)e^{-bt_n}]} = \sum_{i=1}^n \frac{(y_i - y_{i-1})(t_i^2 e^{-bt_i} - t_{i-1}^2 e^{-bt_{i-1}})}{[(1 + bt_{i-1})e^{-bt_{i-1}} - (1 + bt_i)e^{-bt_i}]} \quad (6.42)$$

Similarly, for Type-II data, the estimate of parameters  $a$  and  $b$  using the MLE method can be obtained by solving the following equations:

$$a = \frac{n}{[1 - (1 + bs_n)e^{-bs_n}]} \quad (6.43)$$



and

$$\frac{2n}{b} = \sum_{i=1}^n s_i + \frac{nb s_n^2 e^{-bs_n}}{[1 - (1 + bs_n e^{-bs_n})]} \quad (6.44)$$

*Example 6.2:* The small on-line data entry software package test, data set #1 available since 1980 in Japan (Ohba 1984a), is shown in Table 4.5. The size of the software has approximately 40000 LOC. The testing time was measured on the basis of the number of shifts spent running test cases and analyzing the results. The pairs of the observation time and the cumulative number of faults detected are presented in Table 4.5.

One can easily obtain the unknown parameters  $a$  and  $b$  for the delayed S-shaped NHPP model using the MLE as follows

$$a = 71.725 \quad b = 0.104$$

The estimated mean value function  $m(t)$  is

$$m(t) = (71.725)[1 - (1 + 0.104t)e^{-0.104t}]$$

It seems that the delayed S-shaped NHPP model fits the observed failure data well in this data set.

### Connective NHPP Model

Nakagawa (1994) proposed a model, called the connective NHPP model, where the basic shape of the growth curve is exponential and that an S-curve forms due to the test. In the connective NHPP model, a group of modules called "main route modules" are tested first, followed by the rest of the modules. Even if the failure intensity of the faults in the main route module and the other modules are similar, the growth curve becomes an S-curve since the search for their detection starts at different points in time.

**Theorem 6.8 (Nakagawa 1994):** The expected number of faults detected for the software as a whole can be expressed as follows:

$$m(t) = a_1 (1 - e^{-bl_{(0,t_0)}(t)}) + a_2 (1 - e^{-bl_{(t_0,\infty)}(t)}) \quad (6.45)$$

where  $a_2 > a_1 > 0$ , and

- $a_1 =$  number of faults that are expected to be detected in the main route modules
- $a_2 =$  number of faults that are expected to be detected in modules other than the main route modules
- $b =$  failure intensity
- $t_0 =$  starting time for testing modules other than the main route modules
- $I_{[.]} =$  indicator function.

## 6.6 NHPP Imperfect Debugging Models

Many existing models describe perfect debugging in section 6.5, that is,  $a(t) = a$  and where the error detection rate  $b(t)$  function is time-dependent. In this section,

we discuss several software reliability models with imperfect debugging processes and a constant error detection rate  $b(t)=b$ , studied by Yamada (Yamada 1984). The NHPP imperfect debugging model is based on the following assumptions:

1. When detected errors are removed, it is possible to introduce new errors.
2. The probability of finding an error in a program is proportional to the number of remaining errors in the program.

**Theorem 6.9 (Yamada 1984):** The mean value function solution of the following differential equation:

$$\frac{\partial m(t)}{\partial t} = b[a(t) - m(t)] \quad (6.46)$$

with the initial condition  $m(0) = 0$ , and  $a(t)$  is defined as the error content function of time  $t$  during software testing, is given by

$$m(t) = be^{-bt} \int_0^t a(s)e^{bs} ds \quad (6.47)$$

**Theorem 6.10 (Yamada 1984):** Assume the error content function is

$$a(t) = ae^{\alpha t} \quad (6.48)$$

then the mean value function in equation (6.47) is given by

$$m(t) = \frac{ab}{b + \alpha} (e^{\alpha t} - e^{-bt}) \quad (6.49)$$

This model has been studied by Yamada (1984) and is called the **Yamada imperfect debugging model 1**. It is straightforward to obtain the function  $m(t)$  in equation (6.49) by substituting  $a(t)$  in equation (6.48) into equation (6.47).

We next show how to estimate the parameters of  $a$ ,  $b$ , and  $\alpha$ . Using the MLE method, the log of the likelihood function is

$$\begin{aligned} LLF = \sum_{i=1}^n [(y_i - y_{i-1}) \ln[m(t_i) - m(t_{i-1})] - [m(t_i) - m(t_{i-1})] \\ - \ln[(y_i - y_{i-1})!]] \end{aligned}$$

where

$$m(t_i) - m(t_{i-1}) = \frac{ab}{b + \alpha} [(e^{\alpha t_i} - e^{-\alpha t_{i-1}}) - (e^{bt_i} - e^{-bt_{i-1}})]$$

Taking the partial derivatives of the above function with respect to the unknown parameters  $a$ ,  $b$ , and,  $\alpha$ , set

$$\begin{aligned} \frac{\partial}{\partial a} \ln L &= 0 \\ \frac{\partial}{\partial b} \ln L &= 0 \\ \frac{\partial}{\partial \alpha} \ln L &= 0 \end{aligned}$$

then we obtain the results by solving the following equations simultaneously:

$$a = \frac{b + \alpha}{b} \frac{y_n}{(e^{\alpha t_n} - e^{-bt_n})}$$

$$\sum_{i=1}^n \left[ (y_i - y_{i-1}) \frac{a}{B} \left( \frac{\alpha A}{(b + \alpha)^2} + \frac{bC}{b + \alpha} \right) - a \left( \frac{\alpha A}{(b + \alpha)^2} + \frac{bC}{b + \alpha} \right) \right] = 0$$

and

$$\sum_{i=1}^n \left[ (y_i - y_{i-1}) \frac{a}{B} \left( \frac{-bA}{(b + \alpha)^2} + \frac{bC}{b + \alpha} \right) - a \left( \frac{-bA}{(b + \alpha)^2} + \frac{bC}{b + \alpha} \right) \right] = 0$$

where

$$A = (e^{\alpha t_i} - e^{-\alpha t_{i-1}}) - (e^{bt_i} - e^{-bt_{i-1}})$$

$$B = \frac{abA}{b + \alpha}$$

and

$$C = t_i e^{\alpha t_i} - t_{i-1} e^{-\alpha t_{i-1}}$$

**Theorem 6.11 (Yamada 1984):** Assume the error content function is

$$a(t) = a(1 + \alpha t) \quad (6.50)$$

then the mean value function in equation (6.47) is given by

$$m(t) = a(1 - e^{-bt}) \left( 1 - \frac{\alpha}{\beta} \right) + a\alpha t \quad (6.51)$$

This model has been studied by Yamada (1984) and is called the **Yamada imperfect debugging model 2**. It is straightforward to obtain the function  $m(t)$  in equation (6.51) by substituting  $a(t)$  in equation (6.50) into equation (6.47).

## 6.7 NHPP Imperfect Debugging S-shaped Models

A general software reliability model based on NHPP is used to derive a model that integrates imperfect debugging with the "learning" phenomenon. "Learning" is said to occur if testing appears to improve dynamically in efficiency as one progresses through a testing phase. "Learning" usually manifests as a changing fault detection rate.

Published models, and empirical data, suggest that efficiency growth due to "learning" may follow any number of growth curves from linear to that described by the logistic function. On the other hand, some recent work indicates that, in a real industrial resource-constrained environment, very little actual "learning" may take place since non-operational profiles used to generate test and business models may prevent that. When that happens, the testing efficiency may still change when an explicit change in testing strategy takes place, or it may change as a result of the

structural profile of the code under test and test-case ordering. Either way, software reliability engineering researchers agree that changes in the fault-detection rate are common during the testing process (Pham 1999b). Furthermore, in most realistic situations, fault repair has associated with it a fault re-introduction rate due to imperfect debugging phenomenon.

### Notation

$a(t)$	Time dependent fault content function, <i>i.e.</i> , total number of faults in the software including the initial and introduced faults
$b(t)$	Time dependent fault detection rate function, faults per unit of time
$\lambda(t)$	Failure intensity function, faults per unit of time
$m(t)$	The mean value function, <i>i.e.</i> , the expected number of faults detected by time $t$
$R(x/t)$	Software reliability function, <i>i.e.</i> , the conditional probability of no failure occurring during $(t, t+x)$ given that the last failure occurred at time $t$
$\wedge$	Estimates using maximum likelihood estimation method
$SSE$	Sum of the squared errors of a model when fitting the actual data
$y_k$	The number of actual failures observed at time $t_k$
$\hat{m}(t_k)$	Estimated cumulative number of failures at time $t_k$ obtained from the fitted mean value functions, $k = 1, 2, \dots, n$

### 6.7.1 A Generalized Imperfect-debugging Fault-detection Model

The derivation of software reliability models is usually divided into three processes. The counting process  $\{N(t), t \geq 0\}$  that represents the cumulative number of software errors detected by time  $t$  is a stochastic process. Thus, in the first step, this counting process must be described by statistical means. Basic assumptions about this process lead to the commonly accepted conclusion that, for any fixed  $t \geq 0$ ,  $N(t)$  is Poisson-distributed with a time-dependent Poisson-parameter  $m(t)$ , the so-called mean value function (MVF).

The MVF represents the expected number of software errors that have accumulated up to time  $t$ . In a second step, this MVF must be defined analytically. This is usually done by expressing the MVF as a function of two other functions: the error content function  $a(t)$  and the error detection rate  $b(t)$ . By making assumptions about the analytical behavior of these two functions,  $a(t)$  and  $b(t)$  are then defined as functions of time with one or more free parameters.

Some of these parameters might be determined through mathematical or physical inferences. In most cases, however, these parameters need to be inferred statistically. Therefore, in a third step, actual test data are analyzed, using the statistical model defined in the first step with the class of MVFs defined in the second step.

The derivation of the generalized mean value function is discussed next. Most of the existing models (Yamada 1992; Goel 1980; Pham 2000a; Ohba 1984a) for an MVF build upon the assumption that the error detection rate is proportional to

the residual error content. Pham and Nordmann (1997b) formulate a generalized NHPP software reliability model and provide an analytical expression for the MVF.

The generalized NHPP imperfect-debugging fault-detection rate model (Pham 1997b) is formulated based on the following assumptions:

1. The error detection rate differs among faults.
2. Each time a software failure occurs, the software error which caused it is immediately removed, and new faults can be introduced.

**Theorem 6.12 (Pham 1997b):** The generalized mean value function solution of the following differential equations:

$$\frac{\partial m(t)}{\partial t} = b(t)[a(t) - m(t)] \quad (6.52)$$

with the initial condition  $m(t_0) = m_0$ , is given by (Pham 1997b):

$$m(t) = e^{-B(t)} \left[ m_0 + \int_{t_0}^t a(\tau) b(\tau) e^{B(\tau)} d\tau \right] \quad (6.53)$$

where

$$B(t) = \int_{t_0}^t b(\tau) d\tau \quad (6.54)$$

and  $t_0$  is the time to begin the debugging process and  $m(t_0) = m_0$ .

**Proof (Pham 2000a):** Let us rewrite equation (6.52) as follows:

$$\frac{\partial}{\partial t} m(t) + b(t)m(t) = S(t) \quad (6.55)$$

where  $S(t) = a(t)b(t)$ . Note that

$$\frac{\partial}{\partial t} \left[ m(t) e^{-\int_{t_0}^t b(\tau) d\tau} \right] = \left[ \frac{\partial}{\partial t} m(t) + b(t)m(t) \right] e^{-\int_{t_0}^t b(\tau) d\tau}$$

Multiplying both sides of equation (6.55) by the integrating factor

$$e^{-\int_{t_0}^t b(\tau) d\tau}$$

we have

$$\frac{\partial}{\partial t} \left[ m(t) e^{-\int_{t_0}^t b(\tau) d\tau} \right] = S(t) e^{-\int_{t_0}^t b(\tau) d\tau}$$

Integrating between  $t_0$  and  $t$ , we obtain

$$m(t) = m(t_0)e^{-\int_{t_0}^t b(\tau)d\tau} + \int_{t_0}^t S(\tau)e^{-\int_{t_0}^t b(\tau)d\tau} d\tau$$

Note that

$$e^{-\int_{\tau}^t b(y)dy} = e^{-[\int_{t_0}^t b(y)dy - \int_{t_0}^{\tau} b(y)dy]}$$

Therefore,

$$m(t) = m(t_0)e^{-\int_{t_0}^t b(\tau)d\tau} + \int_{t_0}^t S(\tau)e^{-[\int_{t_0}^t b(y)dy - \int_{t_0}^{\tau} b(y)dy]} d\tau \quad (6.56)$$

Substituting

$$B(t) = \int_{t_0}^t b(\tau)d\tau$$

$$S(t) = a(t)b(t) \text{ and } m(t_0) = m_0$$

into equation (6.56) and after simplifications, we obtain

$$m(t) = e^{-B(t)} \left[ m_0 + \int_{t_0}^t a(\tau)b(\tau)e^{B(\tau)} d\tau \right]$$

This yields the same result as in equation (6.53).

Q.E.D.

In the simplest model, the function  $a(t)$  and  $b(t)$  are both constants. A constant  $a(t)$  stands for the assumption that no new errors are introduced during the debugging process (perfect debugging). A constant  $b(t)$  implies that the proportional factor relating the error detection rate  $\lambda(t)$  to the total number of remaining errors is constant. This model is known as the Goel-Okumoto NHPP model. Many existing models describe perfect debugging, i.e.,  $a(t) = a$ , with a time-dependent error detection rate  $b(t)$  (see Section 6.5). Other studies deal with an imperfect debugging process and a constant error detection rate  $b(t) = b$  (Section 6.5)

In a general model, the functions  $a(t)$  and  $b(t)$  are both functions of time and, for practical purposes, both are increasing with time. An increasing  $a(t)$  shows that the total number of errors (including those already detected) increases with time because new errors are introduced during the debugging process. An increasing proportional factor  $b(t)$  indicates that the error detection rate usually increases as debuggers establish greater familiarity with the software.

Although the relationship above does not yield immediate conclusions about  $m(t)$ , it relates  $m(t)$  to two other functions that, by their definition, possess actual physical meanings. The function  $a(t)$  represents the total error content at time  $t$ , and  $b(t)$  represents the error detection rate. In this way, by introducing functional

assumptions about  $a(t)$  and  $b(t)$ , which are more tangible, an analytical expression for  $m(t)$  can be derived.

Many existing NHPP models can be considered as a special case of the generalized model as in equation (6.53). An increasing  $a(t)$  function implies an increasing total number of faults (note that this includes those already detected and removed and those inserted during the debugging process) and reflects imperfect debugging.

An increasing  $b(t)$  implies an increasing fault detection rate, which could be either attributed to a learning curve phenomenon (Ohba 1984a; Yamada 1992), or to software process fluctuations (Zhang 1998), or a combination of both.

### Pham-Nordmann-Zhang Model (PNZ model)

This model assumes that:

1. The introduction rate is a linear function time-dependent overall fault content function.
2. The fault detection rate function is non-decreasing time-dependent with an inflection S-shaped model.

**Theorem 6.13 (Pham 1999b):** Assume that the time-dependent fault content function and error detection rate are, respectively,

$$\begin{aligned} a(t) &= a(1 + \alpha t) \\ b(t) &= \frac{b}{1 + \beta e^{-bt}} \end{aligned} \quad (6.57)$$

where  $a = a(0)$  is the parameter for the total number of initial faults that exist in the software before testing, and  $\frac{b}{1 + \beta}$  is the initial per fault visibility or failure intensity. The mean value function of the equation (6.52) is given by

$$m(t) = \frac{a}{1 + \beta e^{-bt}} \left( [1 - e^{-bt}] \left[ 1 - \frac{\alpha}{\beta} \right] + \alpha t \right) \quad (6.58)$$

This model is known as the **PNZ model** (Pham 1999b). The result can be obtained by substituting both the functions  $a(t)$  and  $b(t)$  into equation (6.53) where the initial condition  $m(0) = 0$ . In other words, the PNZ model incorporates the imperfect debugging phenomenon by assuming that faults can be introduced during the debugging phase at a constant rate of  $\alpha$  fault per detected fault.

Therefore, the fault content rate function,  $a(t)$ , is a linear function of the testing time. The model also assumes that the fault detection rate function,  $b(t)$ , is a non-decreasing S-shaped curve (Ohba 1984a), which may capture the “learning” process of the software testers.

### Pham Exponential Imperfect Debugging Model

This model assumes that:

1. The introduction rate is an exponential function of testing time.
2. The error detection rate function is non-decreasing with an inflection S-shaped model.

**Theorem 6.14 (Pham 2000a):** Assume the time-dependent fault content function and error detection rate are, respectively,

$$\begin{aligned} a(t) &= \alpha e^{\beta t} \\ b(t) &= \frac{b}{1 + ce^{-bt}} \end{aligned} \quad (6.59)$$

then the mean value function is given by

$$m(t) = \frac{\alpha b}{b + \beta} \left( \frac{e^{(\beta+b)t} - 1}{e^{bt} + c} \right) \quad (6.60)$$

This model is called as the **Pham Exponential Imperfect Debugging model**. The result can be obtained by substituting both the functions  $a(t)$  and  $b(t)$  into Eq. (6.53) where  $m(0) = 0$ .

We are interested in estimating the parameters  $\alpha$ ,  $\beta$ ,  $b$ , and  $c$  of function  $m(t)$  in equation (6.60). If data are given on the cumulative number of errors at discrete times ( $y_i := y(ti)$  for  $i = 1, 2, \dots, n$ ), then we need to obtain the first partial derivative of  $m(t)$  with respect to  $\alpha$ ,  $\beta$ ,  $b$ , and  $c$ , respectively.

$$\begin{aligned} \frac{\partial}{\partial \alpha} m(t) &= \frac{1}{\alpha} m(t) \\ \frac{\partial}{\partial \beta} m(t) &= \left[ \frac{-1}{(b + \beta)} + \frac{te^{(b+\beta)t}}{e^{(b+\beta)t} - 1} \right] m(t) \\ \frac{\partial}{\partial b} m(t) &= \left[ \frac{\beta}{(b + \beta)b} + \frac{te^{(b+\beta)t}}{e^{(b+\beta)t} - 1} + \frac{-te^{bt}}{e^{bt} + c} \right] m(t) \\ \frac{\partial}{\partial c} m(t) &= \frac{1}{e^{bt} + c} m(t) \end{aligned}$$

The second derivative of  $m(t)$  with respect to  $\alpha$ ,  $\beta$ ,  $b$ , and  $c$  is

$$\begin{aligned} \frac{\partial^2}{\partial \alpha^2} [m(t)] &= 0 \\ \frac{\partial^2}{\partial \alpha \partial \beta} m(t) &= \frac{1}{\alpha} \frac{\partial}{\partial \beta} m(t) \\ \frac{\partial^2}{\partial \alpha \partial b} m(t) &= \frac{1}{\alpha} \frac{\partial}{\partial b} m(t) \end{aligned}$$



$$\frac{\partial^2}{\partial \alpha \partial c} m(t) = \frac{1}{\alpha} \frac{\partial}{\partial c} m(t)$$

$$\begin{aligned} \frac{\partial^2}{\partial \beta^2} m(t) = & \left[ \left( \frac{-1}{(b+\beta)} + \frac{te^{(b+\beta)t}}{e^{(b+\beta)t} - 1} \right)^2 \right. \\ & \left. + \left( \frac{-1}{(b+\beta)^2} - \frac{t^2 e^{(b+\beta)t}}{(e^{(b+\beta)t} - 1)^2} \right)^2 \right] m(t) \end{aligned}$$

$$\begin{aligned} \frac{\partial^2}{\partial \beta \partial b} m(t) = & \left[ \left( \frac{-1}{(b+\beta)} + \frac{te^{(b+\beta)t}}{e^{(b+\beta)t} - 1} \right) \right. \\ & \left( \frac{\beta}{(b+\beta)b} + \frac{te^{(b+\beta)t}}{e^{(b+\beta)t} - 1} - \frac{te^{bt}}{e^{bt} + c} \right) \\ & \left. + \left( \frac{1}{(b+\beta)^2} - \frac{t^2 e^{(b+\beta)t}}{(e^{(b+\beta)t} - 1)^2} \right)^2 \right] m(t) \end{aligned}$$

$$\frac{\partial^2}{\partial \beta \partial c} m(t) = - \left( \frac{1}{(e^{bt} + c)} \right) \frac{\partial}{\partial \beta} m(t)$$

$$\begin{aligned} \frac{\partial^2}{\partial b \partial b} m(t) = & m(t) \left( \frac{\beta}{(b+\beta)b} + \frac{te^{(b+\beta)t}}{e^{(b+\beta)t} - 1} - \frac{te^{bt}}{e^{bt} + c} \right)^2 \\ & + \left( \frac{\beta(2b+\beta)}{(b+\beta)^2 b^2} + \frac{t^2 e^{(b+\beta)t}}{(e^{(b+\beta)t} - 1)^2} - \frac{ct^2 e^{bt}}{(e^{bt} + c)^2} \right) m(t) \end{aligned}$$

$$\frac{\partial^2}{\partial b \partial c} m(t) = - \left( \frac{1}{e^{bt} + c} \right) \frac{\partial}{\partial b} m(t) + \frac{te^{bt}}{(e^{bt} + c)^2} m(t)$$

$$\frac{\partial^2}{\partial c^2} m(t) = \frac{2}{(e^{bt} + c)^2} m(t)$$

$$\frac{\partial^2}{\partial \beta \partial \alpha} m(t) = \frac{\partial^2}{\partial \alpha \partial \beta} m(t)$$

$$\frac{\partial^2}{\partial b \partial \alpha} m(t) = \frac{\partial^2}{\partial \alpha \partial b} m(t)$$

Note that

$$\lim_{t \rightarrow 0} \frac{te^{(b+\beta)t}}{e^{(b+\beta)t} - 1} = \frac{1}{b+\beta}$$

$$\lim_{t \rightarrow 0} \frac{t^2 e^{(b+\beta)t}}{(e^{(b+\beta)t} - 1)^2} = \frac{1}{(b+\beta)^2}$$

and  $m(0) = 0$ . Consequently,

$$\lim_{t \rightarrow 0} \frac{\partial^2}{\partial x \partial y} m(t) = \lim_{t \rightarrow 0} \frac{\partial}{\partial x} m(t) = m(0) = 0$$

Therefore, one can obtain the model parameters  $\alpha$ ,  $\beta$ ,  $b$ , and  $c$  by solving the following system of equations simultaneously:

$$\begin{aligned} m(t) &= 0 \\ \left[ \frac{-1}{(b+\beta)b} + \frac{te^{(b+\beta)t}}{e^{(b+\beta)t}-1} \right] m(t) &= 0 \\ \left[ \frac{\beta}{(b+\beta)b} + \frac{te^{(b+\beta)t}}{e^{(b+\beta)t}-1} + \frac{-te^{bt}}{e^{bt}+c} \right] m(t) &= 0 \end{aligned} \quad (6.61)$$

If the given data represent the occurrence times of the errors ( $S_j$  for  $j = 1, 2, \dots, n$ ), then we need to obtain the first partial derivative of  $\lambda(t)$  with respect to  $\alpha$ ,  $\beta$ ,  $b$ , and  $c$ , where

$$\begin{aligned} \lambda(t) &= \frac{\partial}{\partial t} m(t) \\ &= \frac{\alpha b}{b+\beta} \left[ \frac{(b+\beta)e^{(b+\beta)t}}{e^{bt}+c} - \frac{e^{(b+\beta)t}-1}{(e^{bt}+c)^2} be^{bt} \right] \end{aligned}$$

Using equation (6.61), we can easily obtain the estimate of  $\alpha$ ,  $\beta$ ,  $b$ , and  $c$ .

### Pham-Zhang NHPP Model

The model (Pham 1997a) assumes that:

1. The error introduction rate is an exponential function of the testing time. In other words, the number of errors increases quicker at the beginning of the testing process than at the end. This reflects the fact that more errors are introduced into the software at the beginning, while at the end, testers possess more knowledge and therefore introduce fewer errors into the program.
2. The error detection rate function is non-decreasing with an inflexion S-shaped model.

**Theorem 6.15 (Pham 1997a):** Assume the time-dependent fault content function and error detection rate are, respectively,

$$\begin{aligned} a(t) &= c + a(1 - e^{-at}) \\ b(t) &= \frac{b}{1 + \beta e^{-bt}} \end{aligned} \quad (6.62)$$

then the mean value function is given by

$$m(t) = \frac{1}{(1 + \beta e^{-bt})} \left( (c + a)(1 - e^{-bt}) - \frac{ab}{b - \alpha} (e^{-\alpha t} - e^{-bt}) \right)$$

(6.63)

This model is known as the **Pham-Zhang model**. The result can be obtained by substituting both the functions  $a(t)$  and  $b(t)$  into equation (6.53) where  $m(0) = 0$ .

In general, NHPP software reliability models can be used to estimate the expected number of errors. Obviously, different models use different assumptions and therefore provide different mathematical forms for the mean value function  $m(t)$ .

Table 6.2 shows a summary of many existing NHPP software reliability models appearing in the software reliability engineering literature (Pham 2003a, b).

**Table 6.2.** Summary of NHPP software reliability models

Model	MVF ( m(t) )
Goel-Okumoto (G-O)	$m(t) = a(1 - e^{-bt})$ $a(t) = a$ $b(t) = b$
DelayedS-shaped	$m(t) = a(1 - (1 + bt)e^{-bt})$ $a(t) = a$ $b(t) = \frac{b^2 t}{1 + bt}$
Inflection S-shaped	$m(t) = \frac{a(1 - e^{-bt})}{1 + \beta e^{-bt}}$ $a(t) = a$ $b(t) = \frac{b}{1 + \beta e^{-bt}}$
HD/G-O model	$m(t) = \log \{ [e^a - c] / [e^{ae^{-bt}} - c] \}$
Yamada exponential	$m(t) = a(1 - e^{-r\alpha(1 - e^{(-\beta t)})})$ $a(t) = a$ $b(t) = r\alpha \beta e^{-\beta t}$

**Table 6.2.** (continued)

Yamada Rayleigh	$m(t) = a(1 - e^{-r\alpha(1 - e^{(-\beta t^2/2)})})$ $a(t) = a$ $b(t) = r\alpha \beta t e^{-\beta t^2/2}$
Yamada imperfect debugging model (1)	$m(t) = \frac{ab}{\alpha + b}(e^{\alpha t} - e^{-bt})$ $a(t) = ae^{\alpha t}$ $b(t) = b$
Yamada imperfect debugging model (2)	$m(t) = a[1 - e^{-bt}][1 - \frac{\alpha}{b}] + \alpha a t$ $a(t) = a(1 + \alpha t) \quad b(t) = b$
Pham exponential imperfect model	$m(t) = \frac{\alpha b}{b + \beta} \left( \frac{e^{(\beta+b)t} - 1}{e^{bt} + c} \right)$ $a(t) = \alpha e^{\beta t}$ $b(t) = \frac{b}{1 + ce^{-bt}}$
PNZ model	$m(t) = \frac{a}{1 + \beta e^{-bt}} \{ [1 - e^{-bt}][1 - \frac{\alpha}{b}] + \alpha a t \}$ $a(t) = a(1 + \alpha t)$ $b(t) = \frac{b}{1 + \beta e^{-bt}}$
Pham-Zhang model	$m(t) = \frac{1}{1 + \beta e^{-bt}} [(c + a)(1 - e^{-bt})$ $- \frac{a}{b - \alpha} (e^{-\alpha t} - e^{-bt})]$ $a(t) = c + a(1 - e^{-\alpha t})$ $b(t) = \frac{b}{1 + \beta e^{-bt}}$

## 6.8 Applications

This section applies the models discussed in this chapter using several failure data sets (discussed in Chapter 4) collected from real software development projects.

The data sets derive from different time-periods and are illustrative of industrial software processes prevalent in that period. The procedure is as follows.

First, we fit each model to the data, estimate the model parameters, and obtain the mean value functions. Second, all models are compared with each other within a data set using the SSE, MSE, and AIC metrics (see Section 6.3).

In these applications, we use most of the data points to fit the models and estimate the parameters. We also use several remaining points to illustrate the short-term predictive power of the model that incorporates both imperfect debugging and variable fault-detection rate.

**Application 6.1 (The NTDS data):** The software data set #4 given in Table 4.8 (Chapter 4) was extracted from information about failures in the development of software for the real-time, multi-computer complex of the US Naval Fleet Computer Programming Center of the US Naval Tactical Data Systems (NTDS).

The software consists of 38 different project modules. The time horizon is divided into four phases: production phase, test phase, user phase, and subsequent test phase. The 26 software failures were found during the production phase, 5 during the test phase and; the last failure was found on 4 January 1971. One failure was observed during the user phase, in September 1971, and two failures during the test phase in 1971.

The fact that the last 3 of the first 26 errors in Table 4.8 (see Chapter 4) occur almost in a cluster, while there is a relatively long interval between errors before and after that cluster, leads to the conclusion that error number 26 is an unfortunate cut-off point if one wishes to fit a software reliability model. Instead, it seems more reasonable to use either the first 25 or the first 27 error data to fit models.

Let us choose the second alternative and fit some 5 selected models below to the first 27 error data. Although not presented here, we also fit the models to the first 25 error data, which results in only slight deviations with the same overall conclusions (see Problem 6.5). For the ease of discussion, let us name the following five selected models to be analyzed in this example.

Model 1 (Goel-Okumoto):

$$m(t) = a(1 - e^{-bt})$$

Model 2 (Inflection S-shaped):

$$m(t) = \frac{a(1 - e^{-bt})}{1 + \beta e^{-bt}}$$

Model 3 (Yamada imperfect debugging 1):

$$m(t) = \frac{ab}{\alpha + b}(e^{\alpha t} - e^{-bt})$$

Model 4 (Yamada imperfect debugging 2):

$$m(t) = a[1 - e^{-bt}] \left[1 - \frac{\alpha}{b}\right] + \alpha at$$

Model 5 (Pham-Nordmann-Zhang):

$$m(t) = \frac{a}{1 + \beta e^{-bt}} [(1 - e^{-bt})(1 - \frac{\alpha}{b}) + \alpha at]$$

and the corresponding reliability of model  $i$  is  $R_i$ .

It is worthwhile to note that parameter estimates derived from a set of test data are representative and meaningful only for working conditions similar to those under which the test data were obtained. In the NTDS data in Table 4.8 (in Chapter 4), for example, the production and test phases can be considered similar, while the user phase is very different from the former two in two ways. First, during the user phase, no new errors due to testing and debugging are introduced.

Second, during the user phase, the software is not subject to such "hard" and extensive testing as it is during the production and testing phase. Both differences contribute towards the same effect - a reduced error occurrence rate. This can be verified by looking at the test data, which indicates a considerably longer time span between errors during the user phase. In general, changes in the working environment go along with shifts in model parameters, and consequently, parameter estimates relating to the production and first test phases are only applicable during the first two phases and meaningless beyond that point.

The MLEs of the parameters for several software reliability models based on the first 27 error data of the NTDS (data set #4 in Chapter 4) are obtained as follows (Pham 2000a):

Model 1:  $a = 29.42827$ ,  $b = 0.007402$

Model 2:  $a = 27.44246$ ,  $b = 0.015517$

$\beta = 2.042596$

Model 3:  $a = 29.42827$ ,  $b = 0.015517$

$\alpha = 0$

Model 4:  $a = 29.42827$ ,  $b = 0.007402$

$\alpha = 0$

Model 5:  $a = 19.32872$ ,  $b = 0.047526$

$\alpha = 0.001256$ ,  $\beta = 24.33569$

It is worth noting that models 1, 3, and 4 yield exactly the same mean value functions after the MLE. Despite allowing for an imperfect debugging model, the perfect debugging model remains the one that provides the best fit to the NTDS data. The additional freedom introduced in models 3 and 4 through an additional parameter has no effect on the fitted mean value function, since the MLE of that additional parameter is found to be  $a = 0$ , transforming models 3 and 4 into Model 1. Model 2 allows for an inflection S-shaped mean value function. However, the fitted mean value function is only vaguely S-shaped if at all.

On the upper end of the time horizon, the mean value function underestimates the actual failure numbers. For example, at mission time  $s=20$ , model 5 estimates a reliability of  $R_5=0.615$ , whereas models 1,3, and 4 estimate a reliability of  $R_1=0.716$  and model 2 estimates a reliability of  $R_2=0.889$ .

The mean value function of Model 5 provides a good fit to the S-shape of the actual NTDS data. It overestimates the actual failure numbers at the upper end of the time horizon. However, it provides an excellent fit to the 27 test data points it is fitted to, and a good overall fit to all data points of the production and first test phases, reducing fitting and forecast errors of models 1-4. Further studies show that the parameter in model 5 appears especially sensitive to data at further progressed times, indicating an increasingly better fit with a widening time span of the test data (Pham 2000a).

**Application 6.2 (The NTDS data, continued):** In this section, we continue to use the NTDS data set to evaluate and compare the descriptive and the predictive power of several existing models. Detailed description of NTDS data can be found in Table 4.8.

There were 26 software failures during the production phase, 5 during the test phase (the last failure was found on 1971 January 4); 1 failure was observed during the user phase in 1971 September, and 2 failures were observed during the test phase in 1971. Since the first two phases, production and testing, can be considered similar, while the user phase is very different, we have decided to combine the first two. We now use the first 27 data points to fit several existing models, and the last four data points to evaluate the predictions (Pham 1999b). Table 6.3 summarizes the SSE and AIC values for some existing NHPP models.

**Table 6.3.** Comparison of goodness-of-fit and predictive power using NTDS data

Model	SSE (fit) $\sum_{k=1}^{27} [y_k - \hat{m}(t_k)]^2$	SSE (prediction) $\sum_{k=28}^{31} [y_k - \hat{m}(t_k)]^2$	AIC
G-O model	136.58	71.96	88.98
Delayed S-shaped	47.276	126.98	85.86
Inflexion S-shaped	57.496	129.88	88.44
HD/G-O	122.83	114.24	91.92
Yamada exponential	136.83	71.63	91.00
Yamada Rayleigh	39.78	134.59	89.66
Yamada imperfect (1)	111.50	922.34	90.36
Yamada imperfect (2)	109.59	2215.11	90.36
PNZ model	13.60	57.67	81.82

From Table 6.3, we can see that SSE (fit) value for the PNZ model is 13.60, which is significantly smaller than the value for other models. The SSE value for the prediction is 57.67, which is again the smallest among all models. Comparing all models using the AIC criterion we find that the new model has the smallest AIC value. This may indicate that the complexity of the model that comes from the

increased number of parameters may be more than compensated by the ability of the model to describe better the debugging process.

**Application 6.3 (The real-time control system data):** The software for monitoring a real-time control system consists of about 200 modules having, on average, 1000 lines of a high-level language such as Fortran (Tohma 1991). Since the test data, data set #8, (see Table 4.12, Chapter 4) are recorded daily, the test operations performed in a day are regarded to be a test instance.

In Table 4.12, data marked with an asterisk (\*) are interpolated data. Let us look at the goodness of fit test to most NHPP software models based on the data set #8 given in Table 4.12. The results of the estimated parameters of the models and their SSEs are given in Table 6.4. It is observed that the Pham-Zhang model, having an SSE equal to 59,549, fits better than the other NHPP models for this failure data set. It is worthwhile to note that the inflection S-shaped model also performs well in this case.

**Application 6.4 (The Tandem Computers data):** The data used in this section derive from one of four major releases of software products at Tandem Computers are documented by Wood (1996).

In this application, two NHPP models will be analyzed: the Pham-Nordmann-Zhang (PNZ) and the Goel-Okumoto (G-O) models (Pham 1999b). Table 6.5 presents the prediction results from week 10 to week 20 for the PNZ and G-O models based on the Release #1 failure data set #5, Table 4.9 in Chapter 4..

From Table 6.5, the SSE value as well as AIC of the PNZ model is smaller than that of G-O model. Table 6.6 also summarizes the results of several existing NHPP models for Release 1 software data (data set #5, Table 4.9).

Commonly, a more sophisticated model, for example, the one that incorporates both imperfect debugging and a changing fault detection rate, is probably worth the effort because it models a more realistic set of actual effects, and also provides short-term predictive power which is at least as good or better than that of more 'traditional' models (as measured by SSE statistic).

Obviously, further work in broader validation of this conclusion is needed using other data sets and other quality metrics including AIC and PRR for descriptive and predictive software reliability modeling (Pham 1999b).



**Table 6.4.** The MLEs and SSEs for some NHPP models for data set #8

Model name	MVF ( $m(t)$ )	MLEs	SSE
Goel-Okumoto (G-O)	$m(t) = a(1 - e^{-bt})$ $a(t) = a$ $b(t) = b$	$\hat{a} = 497.282$ $\hat{b} = 0.0308$	216872
Delayed S- shaped	$m(t) = a(1 - (1 + bt)e^{-bt})$ $a(t) = a$ $b(t) = \frac{b^2t}{1 + bt}$	$\hat{a} = 483.039$ $\hat{b} = 0.06866$	71247
Infection S- shaped	$m(t) = \frac{a(1 - e^{-bt})}{1 + \beta e^{-bt}}$ $a(t) = a$ $b(t) = \frac{b}{(1 + \beta e^{-bt})}$	$\hat{a} = 482.017$ $\hat{b} = 0.07025$ $\hat{\beta} = 4.15218$	60031
Yamada exponential	$m(t) = a(1 - e^{\gamma\alpha(1 - e^{(-\beta t)})})$ $a(t) = a$ $b(t) = \gamma\alpha\beta e^{(-\beta t)}$	$\hat{a} = 67958.8$ $\hat{\alpha} = 0.00732$ $\hat{\beta} = 0.03072$	220702
Yamada Rayleigh	$m(t) = a(1 - e^{\gamma\alpha(1 - e^{(-\beta t^2/2)})})$ $a(t) = a$ $b(t) = \gamma\alpha\beta e^{-\beta t^2/2}$	$\hat{a} = 500.146$ $\hat{\alpha} = 3.31944$ $\hat{\beta} = 0.00066$	87251
Yamada imperfect debugging model (1)	$m(t) = \frac{ab}{\alpha + b}(e^{\alpha t} - e^{-bt})$ $a(t) = ae^{\alpha t}$ $b(t) = b$	$\hat{a} = 654.963$ $\hat{b} = 0.02056$ $\hat{\alpha} = -0.0027$	155011
Yamada imperfect debugging model (2)	$m(t) = a[1 - e^{-bt}][1 - \frac{\alpha}{b}] + \alpha at$ $a(t) = a(1 + \alpha t)$ $b(t) = b$	$\hat{a} = 591.804$ $\hat{b} = 0.02423$ $\hat{\alpha} = -0.0019$	183157
PNZ	$m(t) = \frac{a[1 - e^{-bt}][1 - \frac{\alpha}{b}] + \alpha at}{1 + \beta e^{-bt}}$ $a(t) = a(1 + \alpha t)$ $b(t) = \frac{b}{1 + \beta e^{-bt}}$	$\hat{a} = 470.759$ $\hat{b} = 0.07497$ $\hat{\alpha} = 0.00024$ $\hat{\beta} = 4.69321$	63189

**Table 6.4.** (continued)

Pham-Zhang	$m(t) = \frac{1}{1 + \beta e^{-bt}} [(c + a)(1 - e^{-bt}) - \frac{a}{b - \alpha} (e^{-\alpha t} - e^{-bt})]$ $a(t) = c + a(1 - e^{-\alpha t})$ $b(t) = \frac{b}{1 + \beta e^{-bt}}$	$\hat{a} = 0.46685$ $\hat{b} = 0.07025$ $\hat{\alpha} = 1.4 \times 10^{-5}$ $\hat{\beta} = 4.15213$ $\hat{c} = 482.016$	59549
------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------	-------

**Table 6.5.** G-O and PNZ models using release #1 software data set #5

Testing time (weeks)	CPU hours	Defects found	Predicted total defects by G-O	Predicted total defects by PNZ
1	519	16	-	-
2	968	24	-	-
3	1,430	27	-	-
4	1,893	33	-	-
5	2,490	41	-	-
6	3,058	49	-	-
7	3,625	54	-	-
8	4,422	58	-	-
9	5,218	69	-	-
10	5,823	75	98	74.7
11	6,539	81	107	80.1
12	7,083	86	116	85.2
13	7,487	90	123	90.1
14	7,846	93	129	94.6
15	8,205	96	129	98.9
16	8,564	98	134	102.9
17	8,923	99	139	106.8
18	9,282	100	138	110.4
19	9,641	100	135	111.9
20	10,000	100	133	112.2
SSE			12233	495.98
AIC			149.60	138.56

**Application 6.5 (NTDS data):** Assume the time-dependent fault content function and fault detection rate function are, respectively,

$$a(t) = c + a(1 - e^{-\alpha t})$$

$$b(t) = \frac{b}{1 + \beta e^{-bt}}$$

with the initial condition  $m(t_0) = m_0$ . From equation (6.53), the mean value function is given as follows:

$$m(t) = \frac{e^{bt_0} + \beta}{e^{bt} + \beta} m_0 + \frac{1}{1 + \beta e^{-bt}} [(c + a)(1 - e^{-b(t_0 - t)}) - \frac{ab}{b - \alpha} (e^{-\alpha t} - e^{-[(b - \alpha)t_0 - bt])}] \tag{6.64}$$

This is called an **Extended Pham-Zhang model** and when  $m_0 = 0$  and  $t_0=0$  it yields the Pham-Zhang model

Assume we use the NTDS (data set #4, Table 4.8) data and also assume that  $t_0 = 149$  and  $m(149) = 22$ . Detailed description of NTDS data can be found in Table 4.8, Chapter 4. Then we obtain the following estimates:

$$\begin{aligned} a &= 0.22311 & \alpha &= 0.00677 \\ b &= 0.00442 & \beta &= 0.00607 \\ c &= 46.6092 \end{aligned}$$

which were calculated by using the MLE. From this result, we can obtain a simple approach to determine when the next failure will occur. In other words, after substituting all the estimate parameters into the above mean value function  $m(t)$ , given  $m(149) = 22$ , we can then determine the time to the next failure,  $t_{23}$ , by solving the following equation:

$$\hat{m}(t) = 23$$

and therefore, we obtain

$$t_{23} = 158.356.$$

**Table 6.6.** Prediction comparison of release #1 software failure data set #5

Model	Total defects predicted several weeks (wks) after the Start of system test				
	10 wks	12 wks	14 wks	17 wks	20wks
Goel-Okumoto	98	116	129	139	133
(G-O)	71	82	91	99	102
Delayed S-shaped	98	116	129	139	133
Hossain Dahiya/G-O	96	110	107	114	112
Gompertz	757	833	735	631	462
Pareto (Pham 1999)	98	116	129	139	133
Weibull	152	181	204	220	213
Yamada exponential	77	89	98	107	111
Yamada Raleigh	74.6	85.3	91.4	99.4	106.7
Pham-Zhang (P-Z)	74.7	85.2	94.6	106.8	112.2
Actual data	75	86	93	99	100

**Application 6.6 (AT&T Network-Management System):** We evaluate models using the *System T* data set (data set #7, see Table 4.11) at AT&T. This example uses the first 19 data points to fit the models, estimate the parameters in the models, and

to predict the “future”. Table 6.7 summarizes the results of several NHPP SRGMs. The PNZ and Yamada Imperfect (2) appear to be the best descriptive and predictive models. It is interesting to note that the delayed S-shaped model also fits the data well. The SSE is slightly higher than that of the PNZ model, but it gives a poorer prediction with a SSE of 9.55. Comparing all models using the AIC criterion, we observe that the PNZ model has the smallest AIC value.

**Table 6.7.** Comparison of goodness-of-fit and predictive power of SRGMs

Model	SSE (fit) $\sum_{k=1}^{19} [y_k - \hat{m}(t_k)]^2$	SSE (prediction) $\sum_{k=20}^{22} [y_k - \hat{m}(t_k)]^2$	AIC
G-O model	26.70	2.27	78.48
Delayed S-shaped	21.95	9.55	86.70
Inflexion S-shaped	26.70	2.27	80.05
HD/G-O	32.52	2.55	80.05
Yamada exponential	26.75	2.27	80.08
Yamada Rayleigh	30.06	11.21	91.90
Yamada imperfect (1)	30.92	0.43	79.84
Yamada imperfect (2)	30.34	0.37	79.86
PNZ model	21.93	0.36	75.86

## 6.9 Imperfect Debugging vs Perfect Debugging

It is worthwhile to note that a study by Ohba and Chou (1989) shows that, for an arbitrary  $b(t)$ , a model with  $a(t) = a$  and a model with  $a(t) = a + \alpha m(t)$  are isomorphic. In other words, both models yield, after maximum likelihood estimation, the same mean value functions.

This explains why in many cases, imperfect debugging models do not significantly improve perfect debugging models (Pham 2000a). Consider the following two models.

Model P: The Perfect Debugging Model

Assume that

$$\begin{aligned} a(t) &= a \\ b(t) &= \text{arbitrary} \end{aligned} \quad (6.65)$$

then, from equation (6.30), we obtain

$$m(t) = a(1 - e^{-B(t)}) \quad (6.66)$$

where

$$B(t) = \int_0^t b(s) ds$$

**Model IP: The Imperfect Debugging Model**

Assume that

$$\begin{aligned} a(t) &= \alpha m(t) + a \\ b(t) &= \text{arbitrary} \end{aligned} \quad (6.67)$$

and from equation (6.53) we obtain

$$m(t) = \frac{a}{1 - \alpha} (1 - e^{-B(t)(1 - \alpha)}) \quad (6.68)$$

Let the error detection rates  $b_i(t)$  be functions of type

$$b_i(t) = b_i f(\beta_1^{(1)}, \beta_1^{(1)}, \dots, \beta_1^{(n)}, t)$$

where  $b_i$  are positive constants and  $f(\cdot, t)$  an arbitrary positive function. Furthermore, let

$$v_1 := (a_1, b_1, \beta_1^{(1)}, \beta_1^{(1)}, \dots, \beta_1^{(n)})$$

be a vector of valid values for the free parameters in Model P. Then Model P is uniquely defined through the vector  $v_1$ . Analogously, Model IP is uniquely defined through another vector:

$$v_2 := (a_2, b_2, \beta_2^{(1)}, \beta_2^{(1)}, \dots, \beta_2^{(n)})$$

By letting  $m_I(v_1, t)$  denote the mean value function for Model P, and  $m_2(v_2, t)$  denote the mean value function for Model IP, we find

**Lemma 6.1:** Let  $x$  be a number such that  $0 < x < 1$ . Then the function

$$\phi_x(a_1, b, \beta_1^{(1)}, \dots, \beta_1^{(n)}) := \left( a_1(1 - x), \frac{b}{(1 - x)}, x, \beta_1^{(1)}, \dots, \beta_1^{(n)} \right) \quad (6.69)$$

defines a one-to-one mapping from Model P parameters to Model IP parameters with  $\alpha_2 = x$  such that

$$m_I(v_1, t) = m_2(\phi_x(v_1), t) \quad (6.70)$$

The proof of this lemma is straightforward.

This result shows that the mean value function of the imperfect debugging model equals that of (the) perfect debugging model with an error detection rate of the same type. This observation immediately implies the suspicion that, after substitution of the MLE for the parameters in each of the models, we will have the same mean value function, regardless of which model we begin with. The following results prove this relationship to be true under general conditions in equation (6.67).

**Lemma 6.2 (Pham 2000a):**

(1) If

$$v_1^* := (a_1^*, b_1^*, \beta_1^{(1)*}, \beta_1^{(2)*}, \dots, \beta_1^{(n)*}) \quad (6.71)$$

are MLEs for parameters in Model P, then for every  $\alpha_2^* (0 < \alpha_2^* < 1)$ ,

$$\phi_{\alpha_2^*}(v_1^*)$$

are MLEs for parameters in Model IP.

(2) If

$$v_2^* := (a_2^*, b_2^*, \beta_2^{(1)*}, \beta_2^{(2)*}, \dots, \beta_2^{(n)*}) \quad (6.72)$$

are MLEs for parameters in Model IP, then

$$\phi_{\alpha_2^*}^{-1}(v_2^*)$$

are MLEs for parameters in Model P

*Proof:* Let  $MLF_1(v_1)$  denote Model P's maximum likelihood function (MLF) as a function of the free parameters,  $MLF_2(v_2)$  the MLF for Model IP. Let  $v_1^*$  and  $v_2^*$  denote the arbitrary MLE of parameters for Model P and Model IP, respectively. From Pham (2000a):

$$\begin{aligned} MLF_1(v_1^*) &= MLF_2(\phi_{\alpha_2^*}(v_1^*)) \leq MLF_2(v_2^*) = MLF_1(\phi_{\alpha_2^*}^{-1}(v_2^*)) \\ &= MLF_1(v_1^*) \end{aligned}$$

The two " $\leq$ " relations hold due to Lemma 6.1. The two " $=$ " signs hold because  $v_1^*$  is an MLE for Model P and  $v_2^*$  is an MLE for Model IP. Since the left term in the above series equals the right, we conclude that all " $\leq$ " must be " $=$ " signs, and consequently,

$$MLF_2(\phi_{\alpha_2^*}(v_1^*)) = MLF_2(v_2^*)$$

and

$$MLF_1(\phi_{\alpha_2^*}^{-1}(v_2^*)) = MLF_1(v_1^*)$$

**Theorem 6.16 (Pham 2000a):** For every  $x$ ,  $0 < x < 1$ , the function  $\theta_x(v_1)$  is a one-to-one mapping of the MLE of Model P parameters to Model IP parameters with  $\alpha_2 = x$ .

*Proof:* From Lemma 6.2, the result follows.

## 6.10 Mean Time Between Failures for NHPP

Let  $N(t)$  be an NHPP with the mean value function  $m(t)$  where  $N(t)$  denotes the random variable of the total number of the events during  $[0, t]$ . Let  $T_k$  denote the random variable of the occurring time for the  $k$ th event, and let  $X_k$  be the time interval between the  $(k-1)$ th and  $k$ th event. Then

$$X_k = T_k - T_{k-1}$$

where  $k = 1, 2, \dots$  and  $T_0 = 0$ .

The probability density function of  $T_k$  (Nakagawa 1983) is given by

$$f_{T_k}(t) = \frac{\lambda(t)e^{-m(t)}[m(t)]^{k-1}}{(k-1)!} \quad (6.73)$$

where

$$\lambda(t) = \frac{\partial}{\partial t} m(t)$$

is the intensity function for the NHPP. In software reliability, we commonly assume that the mean value function is bounded, which means  $m(t)$  is always finite as  $t$  approaches infinity. In this section, we assume that:

1.  $m(t)$  is a strictly increasing function and uniformly continuous on any closed interval.
2.  $m(0) = 0$ ,  $m(t)$  is a positive, finite, and differentiable function.

Let

$$E^*[T_k] = E[T_k \mid T_k < \infty] \quad (6.74)$$

be the conditional expectation. Then (Koshimae *et al.* 1994)

$$E^*[T_k] = \frac{\int_0^a m^{-1}(z)z^{k-1}e^{-z}dz}{\int_0^a z^{k-1}e^{-z}dz} \quad (6.75)$$

where  $m(\infty) = a$ .

If the expectation of  $T_k$  is given, the mean time between failures (MTBFs) are given by

$$E^*[T_k] = E[T_k] - E^*[T_{k-1}] \quad (6.76)$$

where  $E^*[X_k]$  is given in equation (6.75).

*Example 6.3:* Consider the mean value function

$$m(t) = a[1 - (1 + bt)e^{-bt}] \quad (6.77)$$

where parameters  $a$  and  $b$  denote the expected total number of initial errors and the detection rate per error, respectively. The intensity function is given by

$$\lambda(t) = ab^2te^{-bt}$$

It is difficult to derive the inverse value function analytically in most mean value functions, for example, in equation (6.77). Let us denote

$$u = m^{-1}(z)$$

then we can rewrite equation (6.75) as follows:

$$E^*[T_k] = \frac{\int_0^\infty u \lambda(u) [m(u)]^{k-1} e^{-m(u)} du}{\int_0^a z^{k-1} e^{-z} dz} \quad (6.78)$$

It should be noted that one can solve the inverse function of  $m(t)$  numerically using the Newton method or other mathematical software programs. Given the NTDS software failure data (data set #4, see Table 4.8) and the total number of observed errors as  $k = 26$ , we obtain the following estimates:

$$a = 27.50 \quad b = 0.0186$$

which was calculated by the MLE method. Table 6.8 shows numerical examples of the MTBFs using equations (6.76) and (6.78).

**Table 6.8.** Mean time between failures (MTBFs) for  $m(t)$  as given in equation (6.77)

Failure no. n	Time between failure $x_k$ (days)	MTBF $E^*[X_k]$
Production (checkout) phase		
1	9	14.4
2	12	8.2
3	11	6.7
4	4	6.1
5	7	5.8
6	2	5.6
7	5	5.5
8	8	5.5
9	5	5.6
10	7	5.7
11	1	5.9
12	6	6.2
13	1	6.5
14	9	6.8
15	4	7.2
16	1	7.6
17	3	8.0
18	3	8.4
19	6	8.7
20	1	9.0
21	11	9.1
22	33	9.1
23	7	8.9
24	91	8.7
25	2	8.5
26	1	8.1



## 6.11 Further Reading

The reader interested in a deeper understanding of NHPP software reliability modeling should note the following highly recommended articles:

Hoang Pham, "Recent Studies in Software Reliability Engineering", a chapter in the Handbook of Reliability Engineering, Editor: Hoang Pham, Springer, 2003, p. 285-302

Hoang Pham, "Software Reliability and Cost Models: Perspectives, Comparison and Practice", *European Journal of Operational Research*, vol. 149, 2003a, p. 475-489

Hoang Pham, "Software Reliability", a chapter in *Wiley Encyclopedia of Electrical and Electronic Engineering*, Editor: John Webster, Wiley, 1999a, p.565-578

## 6.12 Problems

1. Assume the total error content function and error detection rate function are

$$a(t) = \alpha_2(1 + \gamma t)$$

$$b(t) = \frac{b^2 t}{bt + 1}$$

respectively. From equation (6.53), show that the mean value function is given as follows:

$$m(t) = \alpha_2(1 + \gamma t) - \frac{bt + 1}{e^{bt}} - \frac{(1 + bt)\alpha_2\gamma}{be^{bt+1}} \left[ \ln(bt + 1) + \sum_{i=0}^{\infty} \frac{(bt + 1)^{i+1} - 1}{(i + 1)!(i + 1)} \right]$$

2. Assume the total error content function and error detection rate function are

$$a(t) = \alpha(1 + \gamma t)^2$$

$$b(t) = \frac{\gamma^2 t}{\gamma t + 1}$$

respectively. Using equation (6.53), show that the mean value function is given as follows:

$$m(t) = \alpha \left( \frac{1 + \gamma t}{\gamma t} \right) (\gamma t e^{\gamma t} + 1 - e^{\gamma t})$$

3. Using a real-time command and control software system data given in Table 6.9 below:
  - (a) Calculate the maximum likelihood estimates for the parameters  $a$  and  $b$  of the Goel-Okumoto (G-0) NHPP model based on all available data.
  - (b) Obtain the mean value function  $m(t)$  and the reliability function.

- (c) What is the probability that a software failure does not occur in the time (hours) interval [10, 12]?
- (d) Choose another NHPP software reliability model and repeat items (a)-(c).  
Is this model better than the G-O model? Explain why and justify your results.

**Table 6.9.** Failure in 1 hour (execution time) intervals

Hour	Number of failures	Cumulative failures
1	27	27
2	16	43
3	11	54
4	10	64
5	11	75
6	7	82
7	2	84
8	5	89
9	3	92
10	1	93

4. The data set #9 which is given in Table 4.13 was reported in 1970 by Musa (1987). It shows the successive inter-failure times for a real-time command and control system. The recorded times in Table 4.13 are execution times, in seconds, between successive failures. Musa reports that fixes were introduced whenever a failure occurred and execution did not begin again until the identified failure source had been removed. Therefore, we can assume that there are no repeat occurrences of individual faults, although it is possible that an attempt to fix one fault may introduce new ones. Note that there are several zeros in the table, which are apparently accounted for by rounding up the raw times.
- Calculate the maximum likelihood estimates for the unknown parameters of the Musa exponential NHPP model based on all available data.
  - Obtain the mean value function  $m(t)$  and the reliability function.
  - Choose three other NHPP S-shaped models in Sections 6.5-6.7 and repeat items (a) and (b). Draw your own conclusions and findings among the four models. Explain why and justify your results.
5. Based on the first 25 errors in Table 4.8 (data set #4):
- Calculate the MLE for unknown parameters of five models discussed in Application 6.1 of Section 6.8.
  - Obtain the mean value function and reliability function of all five models.
  - Analyze and compare the results of all the models based on MSE and PRR criteria.

6. Based on the phase 2 telecommunication system data set #11 shown in Table 4.15(b):
  - (a) Calculate the MLE for unknown parameters of any five NHPP S-shaped models discussed in this chapter.
  - (b) Obtain the mean value function and reliability function of all five models.
  - (c) Analyze and compare the results of all the models.

## Testing Coverage and Removal Models

### 7.1 Introduction

Software reliability models based on an NHPP have been used to estimate and predict the quality of software products such as reliability, number of remaining errors, and failure intensity. Imperfect debugging, learning phenomenon of software developers, and other realistic issues have been studied during the last three decades (see Chapter 6). However, software development is a very complex process and there are important issues that have not been addressed. Testing coverage is one of these issues. Testing coverage is important information for both software developers and customers of software products. This chapter discusses various software reliability models incorporating testing coverage and fault removal.

### 7.2 Testing Coverage Models

Among all SRGMs, a large family of stochastic reliability models based on a nonhomogeneous Poisson process, which are known as NHPP reliability models, has been widely used to track reliability improvement during software testing. These models enable software developers to evaluate software reliability in a quantitative manner. They have also been successfully used to provide guidance in making decisions such as when to terminate testing the software or how to allocate available resources. However, software development is a very complex process and there are still issues that have not yet been addressed. Testing coverage is one of these issues. Testing coverage information is an important measure for both software developers and customers of software products.

Testing coverage (Pham 2003d) is a measure that enables software developers to evaluate the quality of the tested software and determine how much additional effort is needed to improve the reliability of the software. Testing coverage, on the other hand, can provide customers with a quantitative confidence criterion when they plan to buy or use the software products. To our knowledge, testing coverage

has not been addressed in the existing software reliability models. Testing coverage is an important measure for both software developers and users.

In this section, we discuss models incorporating testing coverage in the software development process and relate it to the error detection rate function. We examine the goodness-of-fit of the testing coverage model and other existing NHPP models based on several sets of software testing data.

#### Notation

$a(t)$	Total errors content at time $t$
$b(t)$	Error detection rate at time $t$
$c(t)$	Testing coverage as a function of time $t$
$\lambda(t)$	Intensity function or fault detection rate per unit time
$m(t)$	Mean value function or the expected number of errors detected by time $t$
$R(x/t)$	Reliability function of software by time $t$ for a mission time $x$
$N(t)$	Counting process representing the cumulative number of failures at time $t$
$\sum_k$	Sum over $k$ from 1 to $n$
SSE	Sum of squared errors of a model fitting the actual data
AIC	Akaike's Information Criterion
PRR	Predictive-risk ratio

### A Generalized Testing Coverage Model

Pham and Zhang (2003d) introduce a generalized model which incorporates testing coverage measure into software reliability assessment. Let  $c(t)$  denote the percentage of the code coverage as a time dependent function which has been examined during software testing. Obviously,  $1-c(t)$  is the percentage of the software code which has not yet been covered by test cases by time  $t$ . The derivative of the testing coverage function,  $c'(t)$ , represents the coverage rate. Therefore, the error detection

rate function can be expressed as  $\frac{c'(t)}{1-c(t)}$ .

**Theorem 7.1 (Pham and Zhang 2003d):** The generalized NHPP model incorporating testing coverage can be formulated as follows:

$$\frac{dm(t)}{dt} = \frac{c'(t)}{1-c(t)} [(a(t) - m(t))] \quad (7.1)$$

where  $a(t)$  is the total fault content function. The explicit solution of the mean value function is given by:

$$m(t) = e^{-B(t)} \left( m_0 + \int_{t_0}^t a(\tau) e^{B(\tau)} \frac{c'(\tau)}{1-c(\tau)} d\tau \right) \quad (7.2)$$

where  $B(t) = \int_{t_0}^t \frac{c'(\tau)}{1-c(\tau)} d\tau$  and  $m(t_0) = m_0$  is the marginal condition of equation

(7.2) with  $t_0$  representing the starting time of the debugging process.

This model indicates that the failure intensity depends on both the rate at which the remaining faults are covered and the number of remaining faults at current time  $t$  divided by the current fractional population of uncovered faults.

Once the total fault content function,  $a(t)$ , and testing coverage function,  $c(t)$ , are determined, the explicit solution of the mean value function  $m(t)$  of equation (7.1) can be obtained. From equation (7.2), the reliability function (see equation 5.3) can be obtained.

### A Testing Coverage Function

Assuming that the testing coverage function  $c(t)$  is a non-negative and concave function of time  $t$  (see Figure 7.1), then the error detection rate is an S-shaped curve (see Figure 7.2).

**Theorem 7.2 (Pham and Zhang 2003d):** Assume the testing coverage function

$$c(t) = 1 - (1 + bt)e^{-bt} \quad (7.3)$$

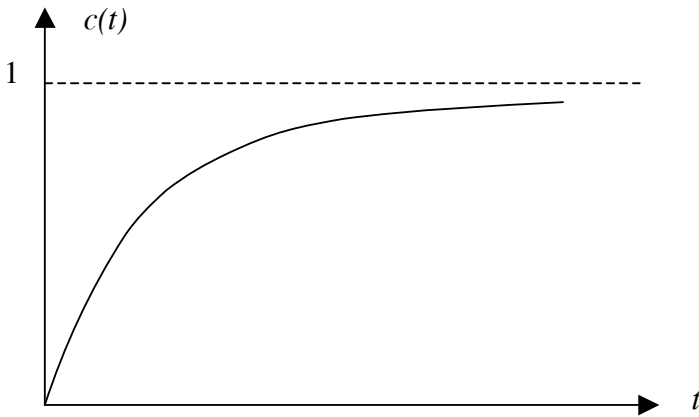
and the error content rate function

$$a(t) = a(1 + \alpha t) \quad (7.4)$$

and with  $m(0)=0$ , the mean value function is given as follows:

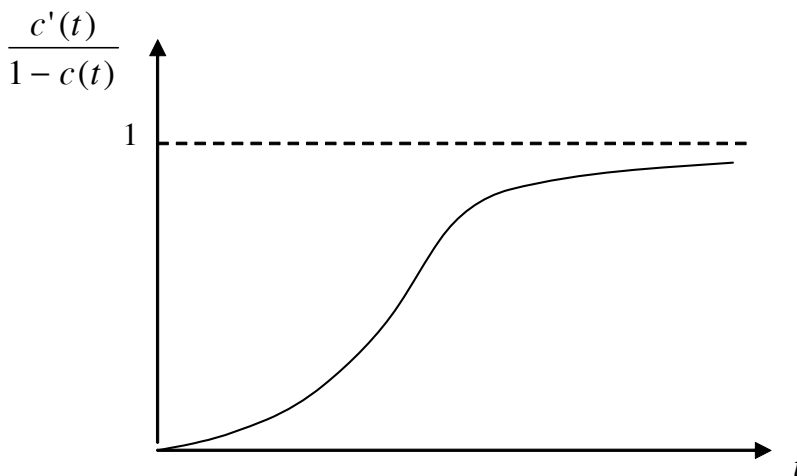
$$m(t) = a \left( 1 + \alpha t - \frac{bt + 1}{e^{bt}} \right) - \frac{a\alpha(1 + bt)}{be^{bt+1}} \left( \ln(bt + 1) + \sum_{i=0}^{\infty} \frac{(1 + bt)^{i+1} - 1}{(i + 1)!(i + 1)} \right) \quad (7.5)$$

This model is known as the PZ-Coverage model. The proof is straightforward as from equation (7.2).



**Figure 7.1.** Testing coverage function  $c(t)$

We assume that errors can be introduced during the debugging phase with a constant error introduction rate  $\alpha$ . Therefore, the error content rate function,  $a(t)$ , is a linear function of the testing time. The NHPP software reliability models can be used to predict the expected number of errors. Different models use different assumptions and therefore provide different mathematical forms for the mean value function.



**Figure 7.2.** The error detection rate function vs time

### 7.3 Testing Coverage and Imperfect Debugging

In existing software reliability models, imperfect debugging of the software testing process is usually studied by assuming that new errors can be introduced into the software during debugging and therefore the fault content function is a time dependent function of testing time. This leads to the fact that certain assumptions about fault content rate need to be made when selecting a software reliability model. In this section, we discuss a software reliability growth model which incorporates fault introduction phenomenon and testing coverage information into error detection (Pham 2005b). This model, however, does not require any specific assumptions for the fault content function. The number of faults in the software estimated by this model is consistent; in other words, this estimate does not change significantly with time.

#### *Notation*

- $a$         Number of initial software faults  
 $d(t)$     Imperfect debugging intensity rate

### Assumptions

The general NHPP software reliability growth model is formulated based on the following assumptions:

1. The occurrence of software failures follows an NHPP.
2. The software failure intensity rate at any time is proportional to the number of remaining faults in the software at that time.
3. When a software failure occurs, a debugging effort takes place immediately. This debugging is  $s$ -independent at each location of the software failures.
4. During the debugging process, the effort to remove each fault may not be perfect and therefore new faults may be introduced into the software system with the imperfect debugging intensity rate  $d(t)$ .
5. The imperfect debugging rate is assumed to decrease as testing progresses and becomes negligible towards the end of the testing phase because the experience and knowledge of the testing team increases with the progress of the learning process.

Under assumptions 4 and 5, a general NHPP model incorporating testing coverage and imperfect debugging can be formulated as follows:

$$\frac{dm(t)}{dt} = \frac{c'(t)}{1-c(t)}[a-m(t)] - d(t)[a-m(t)] \quad (7.6)$$

where  $a$  is the number of initial faults in the software code and  $d(t)$  denotes the fault introduction rate which is a decreasing function of time. The function  $c(t)$  represents the testing coverage function, which measure the percentage of the software code covered by testing cases up to any time  $t$ . Then,  $1-c(t)$  is the percentage of the software code which has not yet been covered by test cases by time  $t$ . The derivative of the testing coverage function,  $c'(t)$ , represents the coverage rate. Therefore, the fault detection rate function can be expressed as

$$\frac{c'(t)}{1-c(t)} \quad (\text{see Figure 7.2}).$$

**Theorem 7.3 (Pham and Zhang 2005b):** Let

$$g(t) = \frac{c'(t)}{1-c(t)} - d(t)$$

be denoted as the imperfect fault detection rate. Then equation (7.6) can be rewritten as:

$$\frac{dm(t)}{dt} = g(t)[a-m(t)] \quad (7.7)$$

The mean value function is, therefore, given by:

$$m(t) = e^{-B(t)} \left[ m_0 + \int_0^t a e^{B(\tau)} g(\tau) d\tau \right] \quad (7.8)$$



where  $B(t) = \int_{t_0}^t g(\tau) d\tau$  and  $m(t_0) = m_0$  is the marginal condition with  $t_0$  representing the starting time of the debugging process.

The proof is straightforward and left to readers (see Problem 7.1). Software reliability  $R(x/t)$  is defined as the probability that a software failure does not occur in  $(t, t+x)$ , given that the last failure occurred at testing time  $t (t \geq 0, x > 0)$ . Therefore, the software reliability function is given by

$$R(x/t) = e^{-[m(t+x) - m(t)]}$$

where  $m(t)$  is given in equation (7.8).

**Theorem 7.4 (Pham and Zhang 2004):** Consider the testing coverage function

$$c(t) = 1 - (1 + bt)e^{-bt} \quad (7.9)$$

and the fault introduction rate,  $d(t)$ , as a decreasing function of testing time  $t$  (see Figure 7.3), as follows:

$$d(t) = \frac{d}{1 + dt} \quad (7.10)$$

From equation (7.8) with the initial condition  $m(0)=0$ , the mean value function is given by

$$m(t) = a - ae^{-bt} [1 + (b + d)t + bdt^2] \quad (7.11)$$

This model is known as Pham-Zhang imperfect fault detection (Pham-Zhang IFD).

**Proof:** See Problem 7.2.

From equation (7.9),

$$c(t) = 1 - (1 + bt)e^{-bt}$$

the fault detect rate function can be expressed as:

$$\frac{c'(t)}{1 - c(t)} = \frac{b^2 t}{1 + bt}$$

Given  $d(t) = \frac{d}{1 + dt}$  then the imperfect fault detection rate function can be expressed as

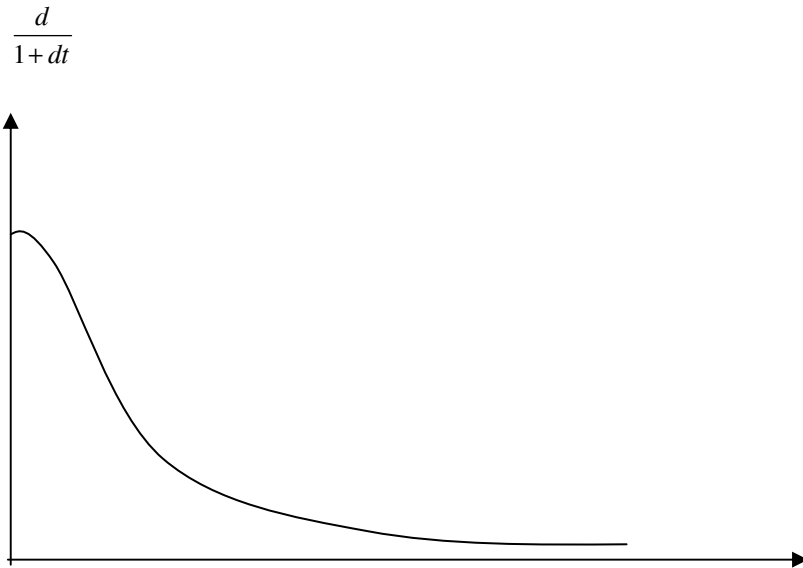
$$g(t) = \frac{b^2 t}{1 + bt} - \frac{d}{1 + dt} \quad (7.12)$$

where the first term is the fault detection rate function and the second term is the imperfect fault detection rate.

## 7.4 Fault Removal Efficiency Model

Although some software reliability studies addressed the imperfect debugging phenomenon, most of them only considered possibilities of adding new faults

while removing the existing ones. However, imperfect debugging also means that detected faults are removed with an imperfect removal efficiency rate other than 100%. Jones (1996) pointed out that the defect removal efficiency is an important factor for software quality and process management. It can provide software developers with the estimation of testing effectiveness and the prediction of additional effort. Moreover, fault removal efficiency is usually way below 100% (e.g., it ranges from 15% to 50% for unit test, 25% to 40% for integration test, and 25% to 55% for system test).



**Figure 7.3.** Imperfect fault detection rate function  $d(t)$  vs testing time

Goel and Okumoto (1979b) also considered a similar conception in their Markov model. They assumed that after a failure the residual faults remained the same with probability  $q$  and it reduces to one less than current value with probability  $p$ . In other words, fault removal is not always 100%. Pham (2005c) recently applied a birth-death process to software reliability modeling, considering both imperfect fault removal probability (death-process) and fault introduction (birth process). In practice, software fault detection is a very complex process. Usually when testers detected a deviation from the requirement, they create a modification request. Then a review board member will assign this request to a particular developer. After the developer studies the software fault, he/she will submit a code change to fix it. Thus, a software fault has to go through a fairly long life cycle which consists of various sequential states. It is not unusual for the software development team to find that a software fault has been reported many times before they are finally removed.

This section discusses a software reliability growth model, studied by Zhang, Teng and Pham (Zhang 2003), addressing fault removal efficiency and fault introduction rate.

Fault removal efficiency is a practical and useful measure in real software development processes since it helps developers to evaluate the debugging effectiveness and estimate the future workload. Imperfect debugging is also considered regarding new faults being introduced into the software during debugging and the detected faults not being removed completely. As a result, this model can provide, in addition to traditional reliability measures, some useful reliability metrics to help the development team make better decisions.

#### Notation

- $p$  Fault removal efficiency, *i.e.*, percentage of faults eliminated by reviews, inspections and tests
- $\beta(t)$  Fault introducing rate at time  $t$

#### Assumption

The following are the assumptions for this model:

1. The occurrence of software failures follows an NHPP.
2. The software fault detection rate at any time is proportional to the number of remaining faults in the software at that time.
3. When a software failure occurs, a debugging effort takes place immediately. This debugging is  $s$ -independent at each location of the software failures.
4. For each debugging effort, whether the fault is successfully removed or not, some new faults may be introduced into the software system with probability  $\beta(t)$ .

#### A Generalized Model with Fault Removal Efficiency

Fault removal efficiency is defined as the percentage of bugs eliminated by reviews, inspections, and tests. This is a very important and convenient metric in software development practice. Incorporating this metrics into software reliability analysis will not only improve the software reliability assessment but also change quality from an amorphous term to a tangible factor.

This section also presents an explicit solution to the differential equation of the proposed model. The mean value function that incorporate both fault removal efficiency and fault introduction phenomena can be obtained by solving the system of differential equations as follows:

$$\frac{dm(t)}{dt} = b(t)[a(t) - pm(t)] \quad (7.13)$$

$$\frac{da(t)}{dt} = \beta(t) \frac{dm(t)}{dt} \quad (7.14)$$

where  $p$  represents the fault removal efficiency, which means  $p\%$  of detected faults can be eliminated completely during the development process. The function  $m(t)$  in equation (7.13) represents the expected number of faults detected by time  $t$ , and  $pm(t)$  then

represents the expected number of faults that can be successfully removed. Existing models usually assume that  $p$  is 100%. The marginal conditions for the differential equations (7.13) and (7.14) are as follows:

$$m(0) = 0 \text{ and } a(0) = a$$

where  $a$  is the number of initial faults in the software system before testing starts. Equation (7.13) can be deduced directly from assumption 2 and 3. Software system failure rate is proportional to the expected number of remaining faults in the software at time  $t$ . The expected number of residual faults is given by

$$x(t) = a(t) - pm(t) \quad (7.15)$$

Notice that when  $p=1$ , the proposed model can be reduced to an existing NHPP model. Equation (7.14) can also be directly deduced from assumption 3 and 4. The fault content rate  $a'(t)$  in the software at time  $t$  is proportional to the rate of debugging efforts to the system, which equals to  $m'(t)$  because of assumption 3. Equation (7.15) can be used to derive explicit solutions of equations (7.13) and (7.14). By taking derivatives on both sides of equation (7.15), we obtain

$$\frac{dx(t)}{dt} = \frac{da(t)}{dt} - p \frac{dm(t)}{dt} = (\beta(t) - p) \frac{dm(t)}{dt}$$

or

$$\frac{dx(t)}{dt} = (\beta(t) - p)b(t)x(t) \quad (7.16)$$

with marginal condition

$$x(0) = a(0) - m(0) = a$$

Hence, the expected number of residual faults given by equation (7.16) is

$$x(t) = ae^{-\int_0^t (p-\beta(\tau)) b(\tau) d\tau} \quad (7.17)$$

From equation (7.13), the failure intensity function can be expressed as follows:

$$\lambda(t) = m'(t) = b(t)(a(t) - pm(t)) = b(t)x(t) \quad (7.18)$$

**Theorem 7.5 (Zhang et al. 2003):** The explicit mean value function and fault content rate function can respectively be obtained as follows:

$$m(t) = \int_0^t x(u) b(u) du = a \int_0^t b(u) e^{-\int_0^u (p-\beta(\tau)) b(\tau) d\tau} du \quad (7.19)$$

and

$$a(t) = a \left( 1 + \int_0^t \beta(u) b(u) e^{-\int_0^u (p-\beta(\tau)) b(\tau) d\tau} du \right)$$

**Proof:** Using the result in equation (7.18), one can easily also obtain the solution for the fault content rate function by taking the integral of equation (7.14).

In this section, we present a model studied by Zhang et al. (2003), called the Zhang-Teng-Pham model, from the general class of model presented in the previous section. The fault detection rate function in this model,  $b(t)$ , is a non-decreasing function with inflexion S-shaped curve, which captures the learning process of the software developers. In the existing models, however, the upper bound of fault detection rate is assumed to be the same as the learning curve increasing rate.

In this chapter, we relax this assumption and use a different parameter for the upper bound of fault detection rate (see equation 7.20). The model also addresses imperfect debugging by assuming faults can be introduced during debugging with a constant fault introduction rate,  $\beta$ .

**Theorem 7.6 (Zhang et al. (2003):** Assume

$$\begin{aligned} b(t) &= \frac{c}{1 + \alpha e^{-bt}} \\ \beta(t) &= \beta \end{aligned} \quad (7.20)$$

The mean value function for the Zhang-Teng-Pham model is as follows:

$$m(t) = \frac{a}{p - \beta} \left[ \left( 1 - \frac{(1 + \alpha)e^{-bt}}{1 + \alpha e^{-bt}} \right)^{\frac{c}{b}(p - \beta)} \right] \quad (7.21)$$

**Proof:** Substituting equation (7.20) into Eq. (7.19), we can easily obtain the result.

Note that as  $t$  approaches  $\infty$ ,  $m(t)$  converges to its upper bound  $\frac{a}{p - \beta}$ . The expected number of residual faults  $X(t)$  is given by:

$$X(t) = a \left[ \left( \frac{(1 + \alpha)e^{-bt}}{1 + \alpha e^{-bt}} \right)^{\frac{c}{b}(p - \beta)} \right] \quad (7.22)$$

and the software failure rate is:

$$\lambda(t) = \frac{ac}{1 + \alpha e^{-bt}} \left[ \left( \frac{(1 + \alpha)e^{-bt}}{1 + \alpha e^{-bt}} \right)^{\frac{c}{b}(p - \beta)} \right] \quad (7.23)$$

Table 7.1 summarizes many existing NHPP models mentioned in this chapter where the function  $a(t)$  is defined as the fault content function,  $b(t)$  is the fault detection function, and  $g(t)$  is the imperfect fault detection rate, together with most of the recent models presented in current software reliability literature.

**Table 7.1.** Summary of the software reliability models

Model	MVF ( $m(t)$ )	Comments
Goel-Okumoto (G-O)	$m(t) = a(1 - e^{-bt})$ $a(t) = a$ $b(t) = b$	Also called exponential model
Delayed S-shaped	$m(t) = a(1 - [1 + bt]e^{-bt})$	Modification of G-O model
Inflection S-shaped	$m(t) = \frac{a(1 - e^{-bt})}{1 + \beta e^{-bt}}$ $a(t) = a$ $b(t) = \frac{b}{1 + \beta e^{-bt}}$	Becomes the same as G-O if $\beta = 0$
Yamada Exponential	$m(t) = a(1 - e^{-r\alpha[1 - e^{(-\beta t)}]})$ $a(t) = a$ $b(t) = r\alpha \beta e^{-\beta t}$	Attempt to account for testing-effort
Yamada Rayleigh	$m(t) = a(1 - e^{-r\alpha(1 - e^{(-\beta t^2/2)})})$ $a(t) = a$ $b(t) = r\alpha \beta t e^{-\beta t^2/2}$	Attempt to account for testing-effort
Yamada. imperfect debugging (1)	$m(t) = \frac{ab}{\alpha + b}(e^{\alpha t} - e^{-bt})$ $a(t) = ae^{\alpha t}$ $b(t) = b$	Assume exponential fault content function and constant fault detection rate
Yamada. imperfect debugging (2)	$m(t) = a[1 - e^{-bt}][1 - \frac{\alpha}{b}] + \alpha a t$ $a(t) = a(1 + \alpha t)$ $b(t) = b$	Assume constant introduction rate $\alpha$ and the fault detection rate

Table 7.1. (continued)

PNZ model	$m(t) = \frac{a[1 - e^{-bt}][1 - \frac{\alpha}{b}] + \alpha a t}{1 + \beta e^{-bt}}$ $a(t) = a(1 + \alpha t)$ $b(t) = \frac{b}{1 + \beta e^{-bt}}$	Assume introduction rate is a linear function of testing time, and the fault detection rate function is non-decreasing inflexion S-shaped model
Pham-Zhang model	$m(t) = \frac{1}{(1 + \beta e^{-bt})} [(c + a)(1 - e^{-bt}) - \frac{a}{b - \alpha}(e^{-\alpha t} - e^{-bt})]$ $a(t) = c + a(1 - e^{-\alpha t})$ $b(t) = \frac{b}{1 + \beta e^{-bt}}$	Assume introduction rate is exponential function of the testing time, and the fault detection rate is nondecreasing with an inflexion S-shaped model
PZ-coverage	$m(t) = a(1 + \alpha t - \frac{bt + 1}{e^{bt}}) - \frac{a\alpha(1 + bt)}{be^{bt+1}} \times$ $[\ln(bt + 1) + \sum_{i=0}^{\infty} \frac{(1 + bt)^{i+1} - 1}{(i + 1)!(i + 1)}]$ $a(t) = a(1 + \alpha t)$ $c(t) = 1 - (1 + bt)e^{-bt}$	Assume introduction rate is a linear function of time and incorporates the testing coverage function into reliability model
Pham-Zhang IFD	$m(t) = a - ae^{-bt}(1 + (b + d)t + bdt^2)$ $a(t) = a$ $g(t) = \frac{b^2t}{1 + bt} - \frac{d}{1 + dt}$	Assume a constant initial fault content function, and the imperfect fault detection rate combining the fault introduction phenomenon
Zhang-Teng-Pham model	$m(t) = \frac{a}{p - \beta} \left[ 1 - \left( \frac{(1 + \alpha)e^{-bt}}{1 + \alpha e^{-bt}} \right)^{\frac{c}{b}(p - \beta)} \right]$ $a'(t) = \beta(t)m'(t)$ $b(t) = \frac{c}{1 + \alpha e^{-bt}}$ $\beta(t) = \beta$	Assume constant fault introduction rate, and the fault detection rate function is non-decreasing with an inflexion S-shaped model

## 7.5 Model Implementations

In this section, two sets of data will be used to analyze the PZ-coverage model and compare it to the existing NHPP software reliability models. These two sets of data are from IBM and AT&T applications. First, the parameters of each model are estimated and the mean value functions are determined. Second, all the models are compared and the results are presented in Tables 7.2 and 7.3. We examine the goodness-of-fit and predictive power of the models based on the six software application data sets.

**Application 7.1. IBM On-line Data Entry (data set #6, Table 4.10, Chapter 4) - Implementation of PZ-coverage Model:** Consider the data set #6 (listed in Table 4.10, Chapter 4) reported by Ohba (1984) are recorded from testing an on-line data entry software package developed at IBM. Table 4.10 of data set #6 shows the pair of the observation time (days) and the cumulative number of errors that were detected. Here we fit all the given models in Table 7.1 to IBM data set. The estimators, the SSEs, and the AICs for each model are presented in Table 7.2.

**Table 7.2.** Model comparison: IBM data

Model name	MVF ( $m(t)$ ) and MLEs	N	SSE	AIC
Goel-Okumoto (G-O)	$m(t) = a(1 - e^{-bt})$ $\hat{a} = 19.54, \hat{b} = 0.0049$	2	76.38	34.38
Delayed S-shaped SRGM	$m(t) = a(1 - (1 + bt)e^{-bt})$ $\hat{a} = 15.85, \hat{b} = 0.0157$	2	231.78	38.06
Inflection S-shaped SRGM	$m(t) = \frac{a(1 - e^{-bt})}{1 + \beta e^{-bt}}$ $\hat{a} = 28.58, \hat{b} = 0.00013, \hat{\beta} = 0.965$	3	74.51	34.34
Yamada Exponential	$m(t) = a(1 - e^{-r\alpha(1 - e^{t(-\beta)^{-1}})})$ $\hat{a} = 368.22, \hat{\alpha} = 0.055, \hat{\beta} = 0.0048$	4	76.55	38.38
Yamada Rayleigh	$m(t) = a(1 - e^{-r\alpha(1 - e^{t(-\beta)^{-2/2}})})$ $\hat{a} = 17.62, \hat{\alpha} = 2.1073, \hat{\beta} = 5.38 \times 10^{-5}$	4	329.30	44.82
Yamada Imperfect Debugging model (1)	$m(t) = \frac{ab}{\alpha + b}(e^{\alpha t} - e^{-bt})$ $\hat{a} = 12.66, \hat{b} = 0.0084, \hat{\alpha} = 0.001237$	3	70.89	36.33

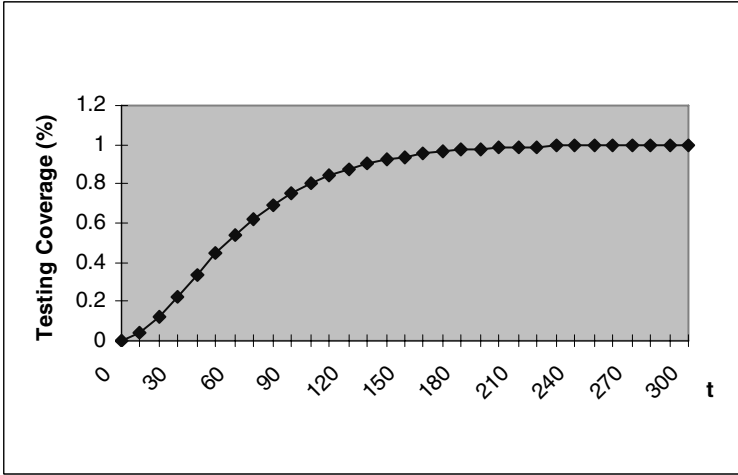


Table 7.2. (continued)

Yamada Imperfect Debugging model (2)	$m(t) = a[1 - e^{-bt}][1 - \frac{\alpha}{b}] + \alpha a t$ $\hat{a} = 11.40, \hat{b} = 0.0094, \hat{\alpha} = 0.001925$	3	71.88	36.38
PNZ model	$m(t) = \frac{a[1 - e^{-bt}][1 - \frac{\alpha}{b}] + \alpha a t}{1 + \beta e^{-\beta t}}$ $\hat{a} = 27.1363, \hat{b} = 0.000144,$ $\hat{\alpha} = 0.000146, \hat{\beta} = 0.9629$	4	74.22	38.33
Pham- Zhang model	$m(t) = \frac{1}{(1 + \beta e^{-\beta t})} [(c + a)(1 - e^{-bt})$ $- \frac{ab}{b - \alpha} (e^{-\alpha t} - e^{-bt})]$ $\hat{a} = 23.97, \hat{b} = 8.8 \times 10^{-5}, \hat{\alpha} = 3.5 \times 10^{-4},$ $\hat{\beta} = 0.9785, \hat{c} = 25.9$	5	70.81	40.33
PZ- coverage	$m(t) = a(1 + \alpha t - \frac{bt + 1}{e^{bt}}) - \frac{a\alpha(1 + bt)}{be^{bt+1}} \times$ $[\ln(bt + 1) + \sum_{i=0}^{\infty} \frac{(1 + bt)^{i+1} - 1}{(i + 1)!(i + 1)}]$ $\hat{a} = 7.0, \hat{b} = 0.0301, \hat{\alpha} = 0.0045$	3	50.75	37.09

From Table 7.2, the PZ-coverage model performs significantly better than the others when applied to this data set. The SSE value of PZ-coverage model, which is 50.75, is much lower than those of other models. The second criterion, AIC, of the PZ-coverage model is also reasonably low. The testing coverage function,  $c(t)$ , as a function of time  $t$  is presented in Figure 7.4. Definitely more application is needed to validate fully this finding.

**Application 7.2. AT&T System T Project (data Set #7, Table 4.11) – Implementation of PZ-coverage Model:** The AT&T’s System T is a network-management system developed by AT&T that receives data from telemetry events, such as alarms, facility-performance information, and diagnostic messages, and forwards them to operators for further action. The system has been tested and failure data has been collected (Ehrlich 1993). Table 4.11 shows the failures and the inter-failure as well as cumulative failure times (in CPU units). We fit all of the models listed in Table 7.1 based on the AT&T data set. Table 7.3 summarizes the SSE and AIC scores for all of these models.



**Figure 7.4.** Testing coverage of IBM data

**Table 7.3.** Model comparison (data set #7)

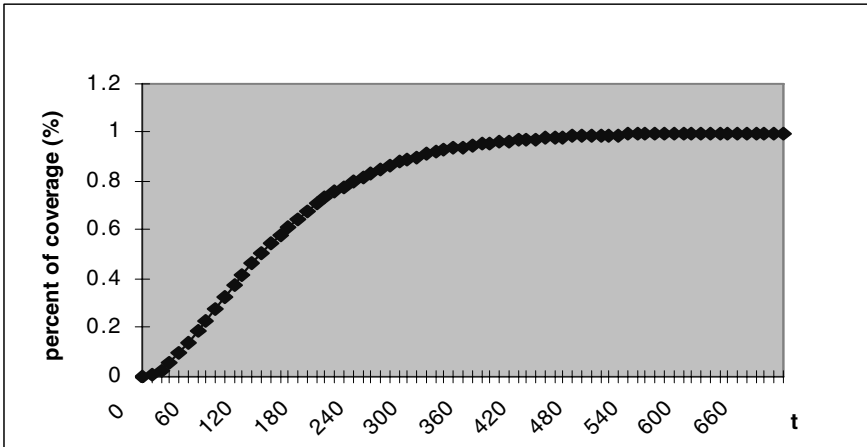
Model name	MVF ( $m(t)$ )	N	SSE	MLEs
Goel-Okumoto (G-O)	$m(t) = a(1 - e^{-bt})$	2	281.33	$\hat{a} = 24.3$ $\hat{b} = 0.00347$
Delayed S-shaped SRGM	$m(t) = a(1 - (1 + bt)e^{-bt})$	2	624.33	$\hat{a} = 22.40$ $\hat{b} = 0.00878$
Inflection S-shaped SRGM	$m(t) = \frac{a(1 - e^{-bt})}{1 + \beta e^{-bt}}$	3	281.38	$\hat{a} = 24.30$ $\hat{b} = 0.0035$ $\hat{\beta} = 0.001$
Yamada exponential	$m(t) = a(1 - e^{-r\alpha(1 - e^{(-\beta t)})})$	4	283.40	$\hat{a} = 1373.79$ $\hat{\alpha} = 0.0179$ $\hat{\beta} = 0.0034$
Yamada Rayleigh	$m(t) = a(1 - e^{-r\alpha(1 - e^{(-\beta t^2/2)})})$	4	421.97	$\hat{a} = 20.50$ $\hat{\alpha} = 2.839$ $\hat{\beta} = 2.29 \times 10^{-5}$
Yamada imperfect debugging model (1)	$m(t) = \frac{ab}{\alpha + b}(e^{\alpha t} - e^{-bt})$	3	223.94	$\hat{a} = 16.67$ $\hat{b} = 0.00623$ $\hat{\alpha} = 0.000546$

Table 7.3. (continued)

Yamada imperfect debugging model (2)	$m(t) = a[1 - e^{-bt}][1 - \frac{\alpha}{b}] + \alpha a t$	3	254.42	$\hat{a} = 16.13$ $\hat{b} = 0.0064$ $\hat{\alpha} = 0.00716$
P-N-Z model	$m(t) = \frac{a[1 - e^{-bt}][1 - \frac{\alpha}{b}] + \alpha a t}{1 + \beta e^{-bt}}$	4	254.38	$\hat{a} = 16.1214$ $\hat{b} = 0.00643,$ $\hat{\alpha} = 0.000717$ $\hat{\beta} = 0.001$
Pham-Zhang	$m(t) = \frac{(c + a)(1 - e^{-bt})}{(1 + \beta e^{-bt})} - \frac{ab(e^{-\alpha t} - e^{-bt})}{(b - \alpha)(1 + \beta e^{-bt})}$	5	222.76	$\hat{a} = 211.449$ $\hat{b} = 0.01449$ $\hat{\alpha} = 6.75 \times 10^{-5}$ $\hat{\beta} = 1.9815$ $\hat{d} = 13.455$
PZ-coverage	$m(t) = a(1 + \alpha t - \frac{bt + 1}{e^{bt}}) - \frac{a\alpha(1 + bt)}{be^{bt+1}} \times [\ln(bt + 1) + \sum_{i=0}^{\infty} \frac{(1 + bt)^{i+1} - 1}{(i + 1)!(i + 1)}]$	3	180.34	$\hat{a} = 15.0$ $\hat{b} = 0.013$ $\hat{\alpha} = 0.0007$

From Table 7.3, we find that the testing PZ-coverage model performs significantly better than the others when applied to the AT&T data set based on the SSE criterion. The testing coverage function,  $c(t)$ , as a function of time  $t$  is presented in Figure 7.5.

**Application 7.3. Data from IBM Entry Software Package - Implementation of Pham-Zhang IFD Model:** This section examines both the goodness-of-fit of the Pham-Zhang IFD model and the existing ones using the software failure data collected from testing an On-line data entry software package at IBM (Ohba 1984a) (see data set 1, Table 4.5). The failures are recorded in days.



**Figure 7.5.** Testing coverage of AT&T System T (data set #7)

We use subset of the above data to fit the models and estimate the parameters. Then we use the remaining ones to compare the predictive power of these existing models. For illustration purpose, we assume that the software has been tested for 17 days and the software failures during these 17 days are recorded.

We estimate the parameters and determine the software reliability models using the first 17 data points, given in Table 7.4. Second, we utilized all the 21-day's data to estimate the model parameters, and the last column of Table 7.4 lists the estimation using the 21-day's data.

From Table 7.4, we find that the Pham-Zhang IFD model provides consistent estimates for the number of initial faults ( $\hat{a}$ ). That is,  $\hat{a}$  estimated using 17 data points is very close to the one estimated using 21 data points. This consistency will provide software developers with the precise information of how many initial faults are in the software and how many faults remain in the code at any time in the testing process.

Table 7.5 summarizes the MSE value for model comparison where MSE is calculated based on 21-day failure data. From the results, we can draw a conclusion that the new model provides the best fit.

**Table 7.4.** MLEs of model parameters (IBM data)

Model name	MVF ( $m(t)$ )	MLEs (17 data points)	MLEs (21 data points)
Goel-Okumoto (G-O)	$m(t) = a(1 - e^{-bt})$ $a(t) = a$ $b(t) = b$	$\hat{a} = 37435.36$ $\hat{b} = 5.97 \cdot 10^{-5}$	$\hat{a} = 56051$ $\hat{b} = 3.9 \times 10^{-5}$
Delayed S-shaped	$m(t) = a(1 - (1 + bt)e^{-bt})$ $a(t) = a$ $b(t) = \frac{b^2 t}{1 + bt}$	$\hat{a} = 75.36$ $\hat{b} = 0.1995$	$\hat{a} = 71.73$ $\hat{b} = 0.10397$
Inflexion S-shaped	$m(t) = \frac{a(1 - e^{-bt})}{1 + \beta e^{-bt}}$ $a(t) = a$ $b(t) = \frac{b}{1 + \beta e^{-bt}}$	$\hat{a} = 63.25$ $\hat{b} = 0.1558$ $\hat{\beta} = 7.743$	$\hat{a} = 57.37$ $\hat{b} = 0.175$ $\hat{\beta} = 8.5136$
Yamada exponential	$m(t) = a(1 - e^{-r\alpha(1 - e^{(-\beta t)})})$ $a(t) = a$ $b(t) = r\alpha \beta e^{-\beta t}$	$\hat{a} = 17081$ $\hat{\alpha} = 0.725$ $\hat{\beta} = 0.00018$	$\hat{a} = 17264.83$ $\hat{\alpha} = 0.734$ $\hat{\beta} = 0.000173$
Yamada Rayleigh	$m(t) = a(1 - e^{-r\alpha(1 - e^{(-\beta t^2/2)})})$ $a(t) = a$ $b(t) = r\alpha \beta t e^{-\beta t^2/2}$	$\hat{a} = 645.57$ $\hat{\alpha} = 0.0837$ $\hat{\beta} = 0.0089$	$\hat{a} = 664.19$ $\hat{\alpha} = 0.086$ $\hat{\beta} = 0.0083$
Yamada imperfect debugging model (1)	$m(t) = \frac{ab}{\alpha + b}(e^{\alpha t} - e^{-bt})$ $a(t) = ae^{\alpha t}$ $b(t) = b$	$\hat{a} = 91920.94$ $\hat{\alpha} = 0.0454$ $\hat{b} = 1.61 \cdot 10^{-5}$	$\hat{a} = 39252.6$ $\hat{\alpha} = 0.0185$ $\hat{b} = 4.57 \times 10^{-5}$
Yamada imperfect debugging model (2)	$m(t) = a[1 - e^{-bt}][1 - \frac{\alpha}{b}] + \alpha a t$ $a(t) = a(1 + \alpha t)$ $b(t) = b$	$\hat{a} = 1787056$ $\hat{\alpha} = 0.08104$ $\hat{b} = 7.41 \cdot 10^{-6}$	$\hat{a} = 1671644$ $\hat{\alpha} = 0.02932$ $\hat{b} = 1.0 \times 10^{-6}$

**Table 7.4.** (continued)

PNZ Model	$m(t) = \frac{a[1 - e^{-bt}][1 - \frac{\alpha}{b}] + \alpha a t}{1 + \beta e^{-bt}}$ $a(t) = a(1 + \alpha t)$ $b(t) = \frac{b}{1 + \beta e^{-bt}}$	$\hat{a} = 63.19$ $\hat{b} = 0.1559$ $\hat{\alpha} = 4.98 \cdot 10^{-5}$ $\hat{\beta} = 7.735$	$\hat{a} = 57.30$ $\hat{b} = 0.175$ $\hat{\alpha} = 4.96 \times 10^{-5}$ $\hat{\beta} = 8.506$
Pham-Zhang model	$m(t) = \frac{1}{(1 + \beta e^{-bt})}[(c + a)(1 - e^{-bt}) - \frac{ab}{b - \alpha}(e^{-\alpha t} - e^{-bt})]$ $a(t) = c + a(1 - e^{-\alpha t})$ $b(t) = \frac{b}{1 + \beta e^{-bt}}$	$\hat{a} = 303.71$ $\hat{b} = 0.01125$ $\hat{\alpha} = 0.0625$ $\hat{\beta} = 0.03105$ $\hat{c} = 104.0103$	$\hat{a} = 167.9$ $\hat{b} = 0.0112$ $\hat{\alpha} = 0.1592$ $\hat{\beta} = 0.01785$ $\hat{c} = 100.615$
Pham-Zhang IFD	$m(t) = a - ae^{-bt}(1 + (b + d)t + bdt^2)$ $a(t) = a$ $g(t) = \frac{b^2 t}{1 + bt} - \frac{d}{1 + dt}$	$\hat{a} = 60.32$ $\hat{b} = 0.138$ $\hat{d} = 0.011$	$\hat{a} = 60$ $\hat{b} = 0.137$ $\hat{d} = 0.013$

**Table 7.5.** Model comparison (IBM data)

Model name	MVF ( $m(t)$ )	MSE (fit)
Goel-Okumoto (G-O)	$m(t) = a(1 - e^{-bt})$ $a(t) = a$ $b(t) = b$	117.24
Delayed S-shaped	$m(t) = a(1 - (1 + bt)e^{-bt})$ $a(t) = a$ $b(t) = \frac{b^2 t}{1 + bt}$	38.0
Inflexion S-shaped	$m(t) = \frac{a(1 - e^{-bt})}{1 + \beta e^{-bt}}$ $a(t) = a$ $b(t) = \frac{b}{1 + \beta e^{-bt}}$	52.01

Table 7.5. (continued)

Yamada exponential	$m(t) = a(1 - e^{-r\alpha(1 - e^{(-\beta t)})})$ $a(t) = a$ $b(t) = r\alpha \beta e^{-\beta t}$	124.81
Yamada Rayleigh	$m(t) = a(1 - e^{-r\alpha(1 - e^{(-\beta t^2/2)})})$ $a(t) = a$ $b(t) = r\alpha \beta t e^{-\beta t^2/2}$	36.89
Yamada imperfect debugging model (1)	$m(t) = \frac{ab}{\alpha + b}(e^{\alpha t} - e^{-bt})$ $a(t) = ae^{\alpha t}$ $b(t) = b$	53.09
Yamada imperfect debugging model (2)	$m(t) = a[1 - e^{-bt}][1 - \frac{\alpha}{b}] + \alpha a t$ $a(t) = a(1 + \alpha t)$ $b(t) = b$	43.87
PNZ model	$m(t) = \frac{a[1 - e^{-bt}][1 - \frac{\alpha}{b}] + \alpha a t}{1 + \beta e^{-bt}}$ $a(t) = a(1 + \alpha t)$ $b(t) = \frac{b}{1 + \beta e^{-bt}}$	34.05
Pham-Zhang model	$m(t) = \frac{1}{(1 + \beta e^{-bt})}[(c + a)(1 - e^{-bt}) - \frac{ab}{b - \alpha}(e^{-\alpha t} - e^{-bt})]$ $a(t) = c + a(1 - e^{-\alpha t})$ $b(t) = \frac{b}{1 + \beta e^{-bt}}$	37.13
Pham-Zhang IFD	$m(t) = a - ae^{-bt}(1 + (b + d)t + bdt^2)$ $a(t) = a$ $b(t) = \frac{b^2 t}{1 + bt} - \frac{d}{1 + dt}$	32.20

**Application 7.4. Real Time Control Systems (data set 8, Table 4.12) – Implementation of Pham-Zhang IFD Model:** We examine the models using a data set collected from testing a program for monitor and real-time control systems. The software consists of about 200 modules and each module has, on average, 1000 lines of a high-level language like FORTRAN.

Table 4.12 records the software failures detected during the 111-day testing period. This actual data is concave overall with several ups and downs reflecting different clusters of detected faults.

Since the software turns stable after 61 days of testing, we will estimate the model parameters using the first 61 data points and compare them with parameter estimates using all the 111 data points. The results are summarized in Table 7.6.

There are totally 481 faults detected by the end of 111-day of testing period. From Table 7.6, the number of initial faults,  $a$ , estimated by the PNZ and Pham-Zhang IFD models using the first 61 data points are 470.8 and 482.5, respectively, whereas when using all the 111 data are 470.8 and 482 respectively.

We can draw the following conclusions: (1) both estimates are very close to the actual number of total faults; (2) the estimation is stable and consistent. This indicates that the new model can provide developers with precise information about the total number of initial faults and number of remaining faults at any time during testing phase.

Table 7.7 summarizes the MSE values of model prediction. From Table 7.7, Pham-Zhang IFD model seems likely to fit and predict better than other existing models on these data sets.



**Table 7.6.** MLEs of model parameters (data set #8)

Model name	MVF ( $m(t)$ )	MLEs (61 data pts)	MLEs (111 data pts)
Goel- Okumoto (G-O)	$m(t) = a(1 - e^{-bt})$ $a(t) = a$ $b(t) = b$	$\hat{a} = 852.97$ $\hat{b} = 0.01283$	$\hat{a} = 497.282$ $\hat{b} = 0.0308$
Delayed S- shaped	$m(t) = a(1 - (1 + bt)e^{-bt})$ $a(t) = a$ $b(t) = \frac{b^2 t}{1 + bt}$	$\hat{a} = 522.49$ $\hat{b} = 0.06108$	$\hat{a} = 483.039$ $\hat{b} = 0.06866$
Inflexion S- shaped	$m(t) = \frac{a(1 - e^{-bt})}{1 + \beta e^{-bt}}$ $a(t) = a$ $b(t) = \frac{b}{1 + \beta e^{-bt}}$	$\hat{a} = 852.45$ $\hat{b} = 0.01285$ $\hat{\beta} = 0.001$	$\hat{a} = 482.017$ $\hat{b} = 0.07025$ $\hat{\beta} = 4.15218$
Yamada exponential	$m(t) = a(1 - e^{-r\alpha(1 - e^{(-\beta^{-1})})})$ $a(t) = a$ $b(t) = r\alpha \beta e^{-\beta^{-1}t}$	$\hat{a} = 9219.7$ $\hat{\alpha} = 0.09995$ $\hat{\beta} = 0.01187$	$\hat{a} = 67958.8$ $\hat{\alpha} = 0.00732$ $\hat{\beta} = 0.03072$
Yamada Rayleigh	$m(t) = a(1 - e^{-r\alpha(1 - e^{(-\beta^{-2/2})})})$ $a(t) = a$ $b(t) = r\alpha \beta t e^{-\beta^{-2/2}t}$	$\hat{a} = 611.70$ $\hat{\alpha} = 1.637$ $\hat{\beta} = 0.00107$	$\hat{a} = 500.146$ $\hat{\alpha} = 3.31944$ $\hat{\beta} = 0.00066$
Yamada imperfect debugging model (1)	$m(t) = \frac{ab}{\alpha + b}(e^{\alpha t} - e^{-bt})$ $a(t) = ae^{\alpha t}$ $b(t) = b$	$\hat{a} = 1795.7$ $\hat{b} = 0.00614$ $\hat{\alpha} = 0.002$	$\hat{a} = 654.963$ $\hat{b} = 0.02059$ $\hat{\alpha} = 0.0027$
Yamada imperfect debugging model (2)	$m(t) = a[1 - e^{-bt}][1 - \frac{\alpha}{b}] + \alpha a t$ $a(t) = a(1 + \alpha t)$ $b(t) = b$	$\hat{a} = 16307$ $\hat{b} = 0.0068$ $\hat{\alpha} = 0.009817$	$\hat{a} = 591.804$ $\hat{b} = 0.02423$ $\hat{\alpha} = 0.0019$

**Table 7.6.** (continued)

PNZ Model	$m(t) = \frac{a[1 - e^{-bt}][1 - \frac{\alpha}{b}] + \alpha a t}{1 + \beta e^{-bt}}$ $a(t) = a(1 + \alpha t)$ $b(t) = \frac{b}{1 + \beta e^{-bt}}$	$\hat{a} = 470.759$ $\hat{b} = 0.07497$ $\hat{\alpha} = 0.00024$ $\hat{\beta} = 4.69321$	$\hat{a} = 470.759$ $\hat{b} = 0.07497$ $\hat{\alpha} = 0.00024$ $\hat{\beta} = 4.69321$
Pham-Zhang model	$m(t) = \frac{1}{(1 + \beta e^{-bt})} [(c + a)(1 - e^{-bt}) - \frac{ab}{b - \alpha} (e^{-\alpha t} - e^{-bt})]$ $a(t) = c + a(1 - e^{-\alpha t})$ $b(t) = \frac{b}{1 + \beta e^{-bt}}$	$\hat{a} = 0.920318$ $\hat{b} = 0.0579$ $\hat{\alpha} = 2.76 \times 10^{-5}$ $\hat{\beta} = 3.152$ $\hat{c} = 520.784$	$\hat{a} = 0.46685$ $\hat{b} = 0.07025$ $\hat{\alpha} = 1.4 \times 10^{-5}$ $\hat{\beta} = 4.15213$ $\hat{c} = 482.016$
Pham-Zhang IFD	$m(t) = a - ae^{-bt} (1 + (b + d)t + bdt^2)$ $a(t) = a$ $b(t) = \frac{b^2 t}{1 + bt} - \frac{d}{1 + dt}$	$\hat{a} = 482.5$ $\hat{b} = 0.0751$ $\hat{d} = 0.006$	$\hat{a} = 482$ $\hat{b} = 0.081$ $\hat{d} = 0.007$

**Table 7.7.** Model Comparison (data set #8)

Model name	MVF ( $m(t)$ )	MSE (prediction)
Goel-Okumoto (G-O)	$m(t) = a(1 - e^{-bt})$ $a(t) = a$ $b(t) = b$	11611.42
Delayed S-shaped	$m(t) = a(1 - (1 + bt)e^{-bt})$ $a(t) = a$ $b(t) = \frac{b^2 t}{1 + bt}$	935.88
Inflexion S-shaped	$m(t) = \frac{a(1 - e^{-bt})}{1 + \beta e^{-bt}}$ $a(t) = a$ $b(t) = \frac{b}{1 + \beta e^{-bt}}$	590.38

**Table 7.7.** (continued)

Yamada exponential	$m(t) = a(1 - e^{-r\alpha(1 - e^{(-\beta t)})})$ $a(t) = a$ $b(t) = r\alpha \beta e^{-\beta t}$	12228.25
Yamada Rayleigh	$m(t) = a(1 - e^{-r\alpha(1 - e^{(-\beta t^2/2)})})$ $a(t) = a$ $b(t) = r\alpha \beta t e^{-\beta t^2/2}$	187.57
Yamada imperfect debugging model (1)	$m(t) = \frac{ab}{\alpha + b}(e^{\alpha t} - e^{-bt})$ $a(t) = ae^{\alpha t}$ $b(t) = b$	8950.54
Yamada imperfect debugging model (2)	$m(t) = a[1 - e^{-bt}][1 - \frac{\alpha}{b}] + \alpha a t$ $a(t) = a(1 + \alpha t)$ $b(t) = b$	2752.83
PNZ Model	$m(t) = \frac{a[1 - e^{-bt}][1 - \frac{\alpha}{b}] + \alpha a t}{1 + \beta e^{-bt}}$ $a(t) = a(1 + \alpha t)$ $b(t) = \frac{b}{1 + \beta e^{-bt}}$	2480.7
Pham-Zhang model	$m(t) = \frac{1}{(1 + \beta e^{-bt})}[(c + a)(1 - e^{-bt}) - \frac{ab}{b - \alpha}(e^{-\alpha t} - e^{-bt})]$ $a(t) = c + a(1 - e^{-\alpha t})$ $b(t) = \frac{b}{1 + \beta e^{-bt}}$	102.66
Pham-Zhang IFD	$m(t) = a - ae^{-bt}(1 + (b + d)t + bdt^2)$ $a(t) = a$ $b(t) = \frac{b^2 t}{1 + bt} - \frac{d}{1 + dt}$	9.37

**Application 7.5. Real Time Control System (data set #9, Table 4.13) – Implementation of Zhang-Teng-Pham Model:** The data is documented in Lyu (1996). There are in total 136 faults reported and the time-between failures (TBF) in seconds are listed in Table 4.13. We tested the goodness of fit using the first 122 data points and the remaining data are used for the predictive power test. The SSE values for fit and prediction are listed in Table 7.8.

Basically, we perform the reliability estimation and prediction at any point in time. The reason for using the first 122 data points is that we observe an extremely long TBF from fault 122 to fault 123 and the TBFs after fault 123 increase tremendously which implies the reliability growth and system stability.

From Table 7.8, we observe that the Zhang-Teng-Pham model provides the best fit and prediction for this data set (both the SSE and the AIC values are the lowest among all models). Furthermore, some instrumental information can be obtained from the parameter estimation provided by the new model. For example, the fault removal efficiency ( $\hat{p}$ ) is 90%, which is relatively high according to (Zhang 2003). The number of initial faults ( $\hat{a}$ ) is estimated to be 135, together with 90% fault removal efficiency; the expected number of total detected faults is then 152. Therefore, at the assumed stopping point of 57042 seconds, there are about 30 ( $152-122=30$ ) faults remaining in the software.

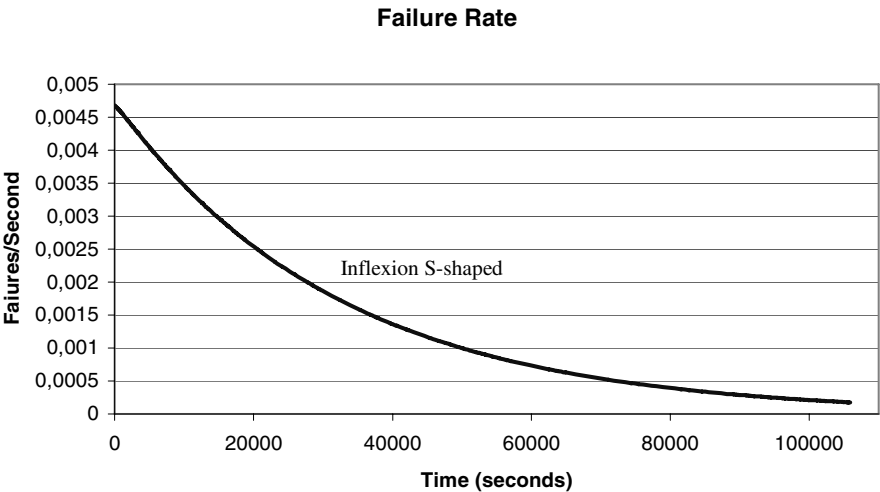
The fault introduction rate ( $\hat{\beta}$ ) is 0.012, that is, on average, one fault will be introduced when 100 faults are removed. Some of the existing models (G-O, Yamada Exponential and Yamada Rayleigh) underestimate the expected number of total faults ( $\hat{a} < 136$ ).

Software failure rate can be predicted after the parameters are estimated. Figure 7.6 shows the trend of failure rate for test and post-test period. Figure 7.7 illustrates the difference between the post-test failure rates predicted by several existing models listed in Table 7.8 and the proposed model.

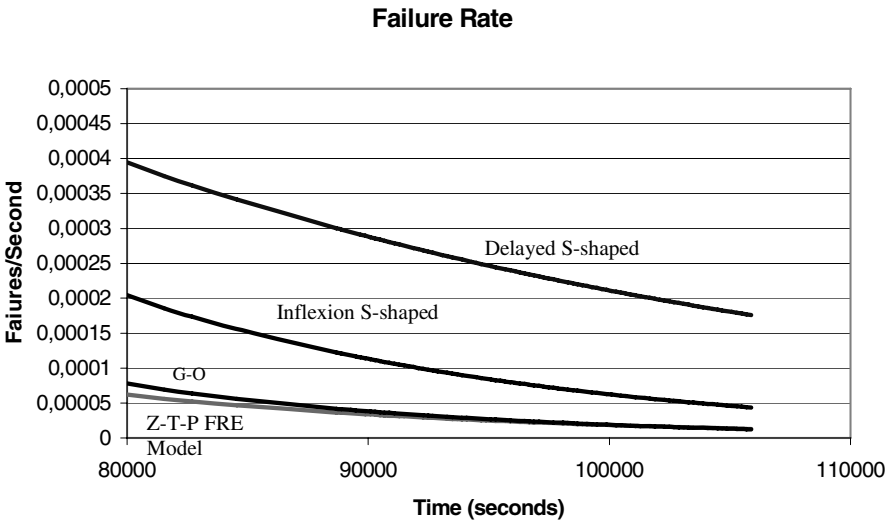
For instance, the failure rate given by the G-O model is on the optimistic side. This is due to the following two reasons: (1) the G-O model underestimates the expected number of total faults (125 instead of 136) and (2) unlike the proposed model, the G-O model does not consider the fault removal efficiency. Thus, we can see that the new model has promising technical merit in the sense that it provides the development teams with both traditional reliability measures and in-process metrics.

**Table 7.8.** Parameter estimation and model comparison

Model name	SSE (fit)	SSE(Predict)	AIC	MLEs
G -O Model	7615.1	704.82	426.05	$\hat{a} = 125$ $\hat{b} = 0.00006$
Delayed S-shaped	51729.23	257.67	546	$\hat{a} = 140$ $\hat{b} = 0.00007$
Inflexion S-shaped	15878.6	203.23	436.8	$\hat{a} = 135.5$ $\hat{b} = 0.00007$ $\hat{\beta} = 1.2$
Yamada exponential	6571.55	332.99	421.18	$\hat{a} = 130$ $\hat{\alpha} = 10.5$ $\hat{\beta} = 5.4 \times 10^{-6}$
Yamada Rayleigh	51759.23	258.45	548	$\hat{a} = 130$ $\hat{\alpha} = 5 \times 10^{-10}$ $\hat{\beta} = 6.035$
Yamada imperfect debugging model (1)	5719.2	327.99	450	$\hat{a} = 120$ $\hat{b} = 0.00006$ $\hat{\alpha} = 1 \times 10^{-5}$
Yamada imperfect debugging model (2)	6819.83	482.7	416	$\hat{a} = 120.3$ $\hat{b} = 0.00005$ $\hat{\alpha} = 3 \times 10^{-5}$
PNZ model	5755.93	106.81	415	$\hat{a} = 121$ $\hat{b} = 0.00005$ $\hat{\alpha} = 2.5 \times 10^{-6}$ $\hat{\beta} = 0.002$
Pham-Zhang model	14233.88	85.36	416	$\hat{a} = 20$ $\hat{b} = 0.00007$ $\hat{\alpha} = 1.0 \times 10^{-5}$ $\hat{\beta} = 1.922$ $\hat{c} = 125$
Zhang-Teng-Pham model	4783.12	32.06	411.36	$\hat{a} = 135$ $\hat{b} = 0.001$ $\hat{\alpha} = 0.01$ $\hat{\beta} = 0.012$ $\hat{c} = 3.5 \times 10^{-5}$



**Figure 7.6.** The trend of failure rate for test and post-test period (data set #9)



**Figure 7.7.** The difference between the post-test failure rates predicted

**Application 7.6. Tandem Computers Data (Data set #5, Table 4.9) - Implementation of Zhang-Teng-Pham Model:** In this example, we look at the predictive power of the Zhang-Teng-Pham model and G-O model using software (Release 1) failure data set #5 listed in Table 4.9 (Wood 1996). Table 7.9 shows

the results predicted using the CPU execution hours as the time frame. From Table 7.9, we observe that Z-T-P FRE model predicts significantly better than the G-O model based on the SSE and AIC values.

The estimates of the parameters and their implications can be summarized as follows: fault removal efficiency  $\hat{p}$  =0.63, which is below average. Jones (1996) mentioned that the fault removal efficiency ranges from 45 to 99% with the average 72%. Thus more resources need to be allocated to improve the fault removal efficiency. The result also shows that the initial number of faults is  $\hat{a}$  = 103.36, which is greater than the actual total detected faults by the end of the testing phase (100) and the estimated total number of faults by the end of testing phase is about 117. This implies that there are still a number of remaining faults in the software at the end of the testing phase. This agrees with the fact that about 20 faults were detected during user operational phase (Wood 1996). The MLEs of the other model parameters are  $\hat{b}$  = 0.095,  $\hat{\alpha}$  = 0.00039, and  $\hat{\beta}$  = 0.00054.

**Table 7.9.** Comparison of G-O and Zhang-Teng-Pham using data set #5 (Table 4.9)

Release 1				
Testing time (weeks)	CPU hours	Defects found	Predicted total defects by <b>G-O</b> model	Predicted total defects by Z-T- P FRE model
1	519	16	-	-
2	968	24	-	-
3	1,430	27	-	-
4	1,893	33	-	-
5	2,490	41	-	-
6	3,058	49	-	-
7	3,625	54	-	-
8	4,422	58	-	-
9	5,218	69	-	-
10	5,823	75	98	74.7
11	6,539	81	107	80.1
12	7,083	86	116	85.2
13	7,487	90	123	90.0
14	7,846	93	129	94.6
15	8,205	96	129	98.9
16	8,564	98	134	102.9
17	8,923	99	139	106.7
18	9,282	100	138	110.4
19	9,641	100	135	113.8
20	10,000	100	133	117.1
SSE			12233	495.98
AIC			149.60	138.56

## 7.6 Imperfect Debugging Model with Multiple Failure Types

In this section, the development of a software reliability model (Pham 1996a) that addresses the problems of multiple failure types and imperfect debugging based on an NHPP, where the fault detection rate is constant, for predicting software performance measures is discussed. The first model considers the fault detection rate is constant, where the second model considers the fault-detection time-dependent. The model allows for three different error types, categorized by the severity levels or the difficult of detection. Critical errors (Type 1) are very difficult to detect and remove; major errors (Type 2) are difficult to detect and remove; and minor errors (Type 3) are easy to detect and remove. An example of a critical error would be adding two numbers when they should be subtracted. Major errors are easier to detect than critical errors, but are still hard to detect. An example of a major error is going through a loop one too few, or one too many, times. Minor errors are easy to detect. An example of a minor error is forgetting a comma or semicolon where one is needed.

*Notation (used throughout this section)*

$a$	Expected number of software errors to be eventually detected
$b_i$	Error detection rate per type $i$ error, $i = 1, 2, 3$ ; $0 < b_1 < b_2 < b_3 < 1$
$p_i$	Content proportion of type $i$ errors
$\lambda(t)$	Intensity function or error detection rate
$N_i(t)$	Cumulative number of type $i$ errors
$n(t)$	Number of errors to be eventually detected plus the number of errors introduced to the program by time $t$
$\beta_i$	Type $i$ error introduction rate that satisfies $0 \leq \beta_i \leq 1$
$m_i(t)$	Expected number of software type $i$ detected errors by time $t$
$M$	Number of parameters in the model
$\wedge$	Maximum likelihood estimate
$m(t)$	Expected number of software failures detected by time $t$
$R(s/t)$	Software reliability function, <i>i.e.</i> , the conditional probability of no failure occurring during $(t, t+s)$ given that the last failure occurred at time $t$
$y_{ij}$	Cumulative number of actual type $j$ failures observed at time $t_i$
$\hat{m}_j(t_i)$	Estimated cumulative number of type $j$ failures at time $t_i$ obtained from the fitted mean value function, $i = 1, 2, \dots, n$ .

The NHPP imperfect debugging model is based on the following assumptions (Pham 1996):

1. When detected errors are removed, it is possible to introduce new errors.
2. The probability of finding an error in a program is proportional to the number of errors remaining in the program.
3. The probability of introducing a new error is constant.
4. Three types of errors exist:
  - Type 1 errors (critical): very difficult to detect.
  - Type 2 errors (major): difficult to detect.
  - Type 3 errors (minor): easy to detect.



5. The parameters  $a$  and  $b_i$  for  $i = 1, 2, 3$  are unknown constants.
6. The error detection phenomenon in the software is modeled by an NHPP

### 7.6.1 A Constant Fault Detection Rate

Assume the error detection rate per type  $i$  error  $b_i$  is constant where  $i = 1, 2, 3$ ;  $0 < b_1 < b_2 < b_3 < 1$ .

**Theorem 7.7 (Pham 1996a):** The mean value function  $m(t)$  of the generalized model incorporating imperfect debugging with multiple failure types by solving the following differential equations:

$$\begin{aligned}\frac{\partial}{\partial t}[m_i(t)] &= b_i[n_i(t) - m_i(t)] \\ \frac{\partial}{\partial t}[n_i(t)] &= \beta_i \frac{\partial}{\partial t}[m_i(t)]\end{aligned}\tag{7.24}$$

$$\begin{aligned}m(t) &= \sum_{i=1}^3 m_i(t) \\ n_i(0) &= ap_i \\ m_i(0) &= 0\end{aligned}$$

is given by

$$m_i(t) = \frac{ap_i}{(1 - \beta_i)} [1 - e^{-(1 - \beta_i)b_i t}]\tag{7.25}$$

The fault detection rate per unit time and the error content function up to time  $t$  are, respectively,

$$\lambda_i(t) = ap_i b_i e^{-(1 - \beta_i)b_i t}$$

and

$$n_i(t) = \frac{ap_i}{1 - \beta_i} [1 - e^{-(1 - \beta_i)b_i t}]$$

The software reliability function  $R(x|t)$  is given by

$$R(x|t) = e^{-\left[ \sum_{i=1}^3 \frac{ap_i}{(1 - \beta_i)} (1 - e^{-(1 - \beta_i)b_i t}) [1 - e^{-(1 - \beta_i)b_i x}] \right]}\tag{7.26}$$

### Parameter Estimation

The model parameters  $a$ ,  $b_1$ ,  $b_2$ , and  $b_3$  are estimated using the MLE method. For Type 1 data (the data on the cumulative number of detected errors), suppose that the data are available in the form of  $(t_i, y_{ij})$ , where  $y_{ij}$  are the cumulative number of failures type  $j$  detected up to time  $t_i$  for  $i = 1, 2, \dots, n$ , and  $j = 1, 2, 3$ . Assuming the fault detection process is NHPP, the likelihood function  $L(a, b_1, b_2, b_3)$  for given data  $(t_i, y_{ij})$ ,  $i = 1, 2, \dots, n$ , and  $j = 1, 2, 3$  is as follows:

$$\begin{aligned}
L(a, b_1, b_2, b_3) &= \Pr\left\{\prod_{j=1}^3 (m_j(0) = 0, m_j(t_1) = y_{1,j}, m_j(t_2) \right. \\
&= y_{2,j}, \dots, m_j(t_n) = y_{n,j})\} \\
&= \prod_{j=1}^3 \prod_{i=1}^n \frac{[m_j(t_i) - m_j(t_{i-1})]^{y_{i,j} - y_{i-1,j}}}{(y_{i,j} - y_{i-1,j})!} e^{-[m_j(t_i) - m_j(t_{i-1})]}
\end{aligned} \tag{7.27}$$

where

$$m_j(t_i) = \frac{ap_j}{(1 - \beta_j)} [1 - e^{-(1 - \beta_j)b_j t_i}]$$

Taking the log likelihood function, we obtain

$$\begin{aligned}
\ln[L(a, b_1, b_2, b_3)] &= \sum_{j=1}^3 \sum_{i=1}^n [(y_{i,j} - y_{i-1,j}) \ln[m_j(t_i) - m_j(t_{i-1})] \\
&\quad - \ln[(y_{i,j} - y_{i-1,j})!]] - [m_j(t_i) - m_j(t_{i-1})]
\end{aligned}$$

Taking the partial derivatives of the log likelihood function,  $\ln[L(a, b_1, b_2, b_3)]$  with respect to the unknown parameters,  $a$ ,  $b_j$ ,  $b_2$ , and  $b_3$ , and setting them equal to zero, we obtain the following system of equations:

$$a = \frac{\sum_{j=1}^3 y_{nj}}{\sum_{j=1}^3 \frac{p_j}{(1 - \beta_j)} [1 - e^{-(1 - \beta_j)b_j t_n}]} \tag{7.28}$$

$$\begin{aligned}
\sum_{i=1}^n (y_{i,j} - y_{i-1,j}) \frac{(1 - \beta_j)[(t_i e^{-(1 - \beta_j)b_j t_i} - t_{i-1} e^{-(1 - \beta_j)b_j t_{i-1}})]}{[e^{-(1 - \beta_j)b_j t_{i-1}} - e^{-(1 - \beta_j)b_j t_i}]} \\
= ap_j t_n e^{-(1 - \beta_j)b_j t_n}
\end{aligned} \tag{7.29}$$

for  $j = 1, 2$ , and  $3$ . Solving the above system of equations (7.28)-(7.29) simultaneously gives the MLE of parameters  $a$ ,  $b_1$ ,  $b_2$ , and  $b_3$ .

For Type 2 data (the data on failure occurrence times), assume that the data set is available in the form of  $n_1$  Type 1 errors,  $n_2$  Type 2 errors, and  $n_3$  Type 3 errors, and  $S_{1,1} \leq S_{1,2} \leq \dots \leq S_{1,n_1}$ ,  $S_{2,1} \leq \dots \leq S_{2,n_2}$ , and  $S_{3,1} \leq S_{3,2} \leq \dots \leq S_{3,n_3}$ , where  $S_{i,j}$  is the actual time that the  $j^{\text{th}}$  failure of Type  $i$  error occurs. Again, using the MLE method, the likelihood function for the NHPP model in a given data set is as follows:

$$L(a, b_1, b_2, b_3) = \prod_{j=1}^3 \prod_{i=1}^n e^{-m_j(S_r)} \lambda_j(S_{j,i}) \tag{7.30}$$

where

$$S_r = \max\{S_{1,n1}, S_{2,n2}, S_{3,n3}\}$$

Taking the partial derivatives with respect to the unknown parameters and setting them equal to zero, we obtain the following results:

$$\sum_{i=1}^{n_j} S_{j,i} = \frac{n_j - ap_j b_j S_r e^{-(1-\beta_j)b_j S_r}}{b_j(1-\beta_j)} \quad (7.31)$$

$$a = \frac{\sum_{j=1}^3 n_j}{\sum_{j=1}^3 \frac{p_j}{(1-\beta_j)} [1 - e^{-(1-\beta_j)b_j S_r}]} \quad (7.32)$$

for  $j = 1, 2$ , and  $3$ . Solving equations (7.31) and (7.32) simultaneously gives the MLE of parameters  $a$ ,  $b_1$ ,  $b_2$ , and  $b_3$ .

**Application 7.7:** The failure data set #3 (see Table 4.7) (Misra 1983) consists of three types of errors: critical, major, and minor. The observation time (week, hour) and the number of failures detected per week are presented in Table 4.7 (in Chapter 4). Given

$$p_1 = 0.0173; p_2 = 0.3420; p_3 = 0.6407 \\ \beta_1 = 0.5; \beta_2 = 0.2; \beta_3 = 0.05$$

Using the MLE method, the parameters for the reliability model are obtained as follows:

$$a = 428 \\ b_1 = 0.00024275 \\ b_2 = 0.00029322 \\ b_3 = 0.00030495$$

Substituting the known and estimated parameters into the reliability equation, we obtain

$$R(x|t) = e^{-A}$$

where

$$A = 14.81(e^{-0.00012138t})(1 - e^{-0.00012138x}) \\ + 182.97(e^{-0.00023458t})(1 - e^{-0.00023458x}) \\ + 288.65(e^{-0.0002897t})(1 - e^{-0.0002897x})$$

Other reliability performance measures are given by

$$m_1(T) = 14.81(1 - e^{-0.00012138T}), \quad n_1(T) = 14.81(1 - 0.5e^{-0.00012138T}) \\ m_2(T) = 182.97(1 - e^{-0.00023458T}), \quad n_2(T) = 182.97(1 - 0.2e^{-0.00023458T}) \\ m_3(T) = 288.65(1 - e^{-0.0002897T}), \quad n_3(T) = 288.65(1 - 0.05e^{-0.0002897T})$$

### 7.6.2 Fault Detection Time-dependent Rate

Pham and Deng (2003a) extended the imperfect debugging model in Theorem 7.7 by considering the time-dependent fault detection rate function, instead of a constant rate.

Let  $b_i(t)$  be the time-dependent Type  $i$  fault detection rate per unit time,  $i=1,2,3$ ;  $0 < b_i < 1$ . Assume the function  $b_i(t)$  is a non-decreasing S-shape curve which can capture the learning process of the software testers corresponding to Type  $i$  faults, which is given as follows:

$$b_i(t) = \frac{b_i}{1 + \theta_i e^{-b_i t}} \quad (7.33)$$

Assume the error detection rate per type  $i$  error  $b_i$  is constant where  $i = 1, 2, 3$ ;  $0 < b_1 < b_2 < b_3 < 1$ .

**Theorem 7.8 (Pham and Chao 2003a):** The mean value function  $m(t)$  of the generalized model incorporating imperfect debugging and time-dependent fault detection rate with multiple failure types by solving the differential equations:

$$\frac{\partial}{\partial t}[m_i(t)] = b_i(t)[n_i(t) - m_i(t)] \quad (7.34)$$

$$\frac{\partial}{\partial t}[n_i(t)] = \beta_i \frac{\partial}{\partial t}[m_i(t)] \quad (7.35)$$

$$m(t) = \sum_{i=1}^3 m_i(t)$$

$$n_i(0) = ap_i$$

$$m_i(0) = 0$$

where

$$b_i(t) = \frac{b_i}{1 + \theta_i e^{-b_i t}}$$

is given by:

$$m_i(t) = \left( \frac{ap_i}{1 - \beta_i} \right) \left[ 1 - \left( \frac{1 + \theta_i}{\theta_i + e^{b_i t}} \right)^{1 - \beta_i} \right] \quad (7.36)$$

The proof can be obtained in Pham and Chao (2003a).

The software reliability function is given by

$$R(s/t) = e^{-\sum_{i=1}^3 \left( \frac{ap_i}{1 - \beta_i} \right) \left[ \left( \frac{1 + \theta_i}{\theta_i + e^{b_i t}} \right)^{1 - \beta_i} - \left( \frac{1 + \theta_i}{\theta_i + e^{b_i(t+s)}} \right)^{1 - \beta_i} \right]} \quad (7.37)$$

When  $\theta_i = 0$  from equation (7.36), the mean value function becomes

$$m(t) = \sum_{i=1}^3 \left( \frac{ap_i}{1-\beta_i} \right) \left[ 1 - \left( e^{-b_i t (1-\beta_i)} \right) \right]$$

which is the same as equation (7.25).

## Parameter Estimation

The model parameters  $a$ ,  $b_i$ ,  $\beta_i$ , and  $\theta_i$  are estimated using the MLE method. The form of the likelihood function depends on the form of the data being used to estimate the parameters. The first data set type gives the cumulative number of failures up to a given time. The second data set type gives the actual time that each failure occurs. In this section, we only discuss the first type of data set. The discussion of the second type of data set can be obtained in Section 7.6.1.

Assume that the data are available in the form of  $(t_i, y_{ij})$ , where  $y_{ij}$  are the cumulative number of Type  $j$  errors detected up to time  $t_i$  for  $i = 1, 2, \dots, n$  and  $j = 1, 2, \dots, k$ . The likelihood function for estimating the model parameters can be expressed as follows:

$$L(\Omega, t) = P \left\{ \prod_{j=1}^k [m_j(0) = 0, m_j(t_1) = y_{1,j}, m_j(t_2) = y_{2,j}, \dots, m_j(t_n) = y_{n,j}] \right\}$$

$$= \prod_{j=1}^k \prod_{i=1}^n \frac{[m_j(t_i) - m_j(t_{i-1})]^{(y_{i,j} - y_{i-1,j})}}{(y_{i,j} - y_{i-1,j})!} e^{-[m_j(t_i) - m_j(t_{i-1})]}$$

where  $\Omega$  is a vector of unknown parameters  $(a, b_i, \beta_i, \theta_i)$  and  $m_j(t)$  is given in equation (7.36) for  $j = 1, 2, \dots, k$ .

The logarithm of the likelihood function is given by

$$\ln[L(\Omega, t)] = \sum_{j=1}^k \sum_{i=1}^n \{ (y_{i,j} - y_{i-1,j}) \ln[m_j(t_{i,j}) - m_j(t_{i-1,j})] - \ln[(y_{i,j} - y_{i-1,j})!] - [m_j(t_i) - m_j(t_{i-1})] \}. \quad (7.38)$$

A system of differential equations can be constructed by taking the derivatives of the log likelihood function (see equation 7.38) with respect to each unknown parameter and setting them equal to zero. The estimated parameters  $a$ ,  $b_i$ ,  $\beta_i$ , and  $\theta_i$  can be obtained by solving such equations.

We now implement the model in this Section that incorporates the time-dependent fault detection rate with multiple failure types and also with the model in Section 7.6.1, on two data-sets (Applications 7.8 and 7.9 below) collected from real software development projects. The procedure is:

1. Fit each model to the data; estimate the model parameters and obtain the mean value functions and the reliability functions.
2. Compare the models with each other within a data set using the PRR, MSE, and AIC criteria.

3. All of the data points are used to fit the models and estimate the parameters.

**Application 7.8. On-line Communication System (OCS):** The On-line Communication System (OCS) project at ABC Software Company was completed in 2000. The data (data set #2 in Chapter 4) was collected over a period of 12 weeks during which time the testing started and stopped many times and is given in Table 4.6. Errors detection is broken down into sub-categories to help the development and testing team to sort and solve the most critical Modification Requests (MRs) first. These sub-categories are referred to as the severity level depending on the nature of the problem with 1 being the most severe problem, with 2 being the major problem and 3 being a minor problem.

The data set, maps into week, consists of three types of errors: severe 1, severe 2, and severe 3. The observation time (week) and the number of errors detected per week are presented in Table 4.6. The cumulative number of errors observed at time  $t$  (in week) is shown in Figure 7.8.

Given the values of content proportion of errors type:  $p_1 = 0.18$ ,  $p_2 = 0.40$ , and  $p_3 = 0.42$ . The maximum likelihood estimates of the NHPP model in equation (7.25) (see Pham 1996a), called Model 1, and the imperfect debugging NHPP model in equation (7.36) (also Pham 2003a), called Model 2, are given in Table 7.10.

Table 7.11 summarizes the PRR, MSE, and AIC values for Model 1 and Model 2. Table 7.11 shows that for Model 2:

MSE = 7.78, which is much smaller than for Model 1

AIC = 194.4 which is smaller than Model 1

PRR = 0.86, which is appreciably smaller than Model 1

From the results, the NHPP imperfect debugging model (Model 2) indicates that the complexity of the model by incorporating the time-dependent fault detection rate with the learning phenomenon is worth the effort since it models a more realistic set of actual effects. However, further work in broader validation of this remark is needed using other data sets.

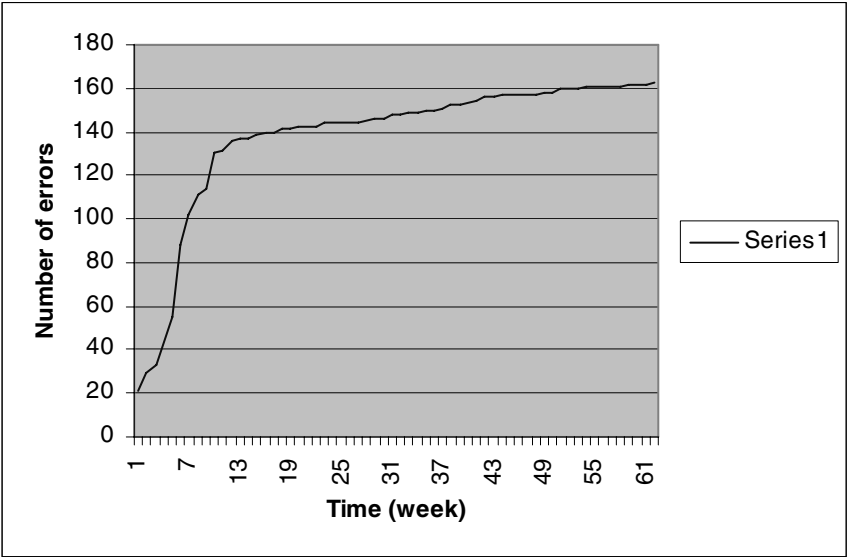


Figure 7.8. Mean value function of model and actual error data

Table 7.10. Parameter estimation

Mod	$a$	$b_1$	$b_2$	$b_3$	$\beta_1$	$\beta_2$	$\beta_3$	$\theta_1$	$\theta_2$	$\theta_3$
1	157	0.083	0.089	0.076	0.5	0.3	0.00005			
2	144	0.289	0.321	0.33	0.1	0.1	0.056	0.257	0.23	0.01

Table 7.11. Comparison of models for OCS data set

Model	PRR	MSE	AIC
Model 1	1.57	34.64	209.36
Model 2	0.86	7.78	194.40

**Application 7.9. Software Failure Data:** In this section the failure data set #3 (see Table 4.7) (Misra, 1983) are used to illustrate further the new NHPP imperfect debugging and time-dependent fault detection rate model. The data sets consist of three types of errors: critical, major, and minor. Given the values of content proportion of errors type:  $p_1=0.02$ ,  $p_2=0.34$ , and  $p_3=0.64$ . The maximum likelihood estimates of the NHPP model in equation (7.25) and the imperfect debugging model in equation (7.36) are given in Table 7.12. Table 7.13 summarizes the PRR, MSE, and AIC values for Model 1 and Model 2. It shows that for Model 2:

MSE = 9.34, which is much smaller than for Model 1

AIC = 331.3 which is smaller than Model 1

PRR = 1.42, which is appreciably smaller than Model 1

The NHPP imperfect debugging model (Model 2) again indicates that the complexity of the model by incorporating the time-dependent fault detection rate is worth the effort to study. Further work in broader validation of this conclusion is needed using other application data sets.

**Table 7.12.** Results of parameter estimation

Mo	$a$	$b_1$	$b_2$	$b_3$	$\beta_1$	$\beta_2$	$\beta_3$	$\theta_1$	$\theta_2$	$\theta_3$
1	240	0.015	0.026	0.032	0.8	0.6	0.75			
2	239	0.02	0.032	0.013	0.9	$10^{-4}$	0.546	0.43	0.273	$10^{-6}$

**Table 7.13.** Comparison of models

Model	PRR	MSE	AIC
Model 1	1.91	16.34	352.6
Model 2	1.42	9.34	331.3

## 7.7 Further Reading

Some interesting research papers on testing coverage and removal are:

X. Zhang, X. Teng and H. Pham, "Considering Fault Removal Efficiency in Software Reliability Assessment", *IEEE Trans. on Systems, Man, and Cybernetics – Part A*, vol. 33, no.1, 2003, p. 114-120

H. Pham and X. Zhang, "NHPP Software Reliability and Cost Models with Testing Coverage", *European Journal of Operational Research*, vol. 145, 2003, p. 443-454

## 7.8 Problems

1. Show that the solution of equation (7.7) in Theorem 7.3 is given by

$$m(t) = e^{-B(t)} \left[ m_0 + \int_{t_0}^t a e^{B(\tau)} g(\tau) d\tau \right]$$



where  $B(t) = \int_{t_0}^t g(\tau) d\tau$  and  $m(t_0) = m_0$

**2.** Show that, if the functions  $c(t)$  and  $d(t)$  are given in equations (7.9) and (7.10), then the function  $m(t)$  is given by

$$m(t) = a - ae^{-bt} [1 + (b + d)t + bdt^2]$$

## **Software Reliability Models with Environmental Factors**

### **8.1 Introduction**

Modern society relies heavily on the correct operation of software in various forms. From the users' point of view, software plays an important role in systems of both safety-critical and civil applications. High-quality software of such systems is desirable and even critical. From the developers' point of view, the pressure of delivering high-quality software on time and within budget requires further research on software reliability assessment. Many software reliability models studied in the last three decades often excluded information on the software development process. In other words, the software reliability is assessed based on the software failure data without taking the software development environment into consideration. This chapter discusses the environmental factors involved in the whole software development process and the impacts of these factors on software reliability assessment. We also discuss several software reliability models that incorporate environmental factors.

### **8.2 Data Analysis**

This section presents a recent survey on software environmental factors consisting of 32 factors based on the studies by Pham and Zhang (1998a) and Zhang and Pham (2000a), and Zhang *et al.* (2001a). The information about the background of survey participants are considered. The survey form with a brief definition of each of the 32 factors is included in Appendix 3. The basic question Pham and Xuemei (1998a) wanted to study was to determine the factors that profile the software development processes and have significant impact on software reliability?

Analyses of the survey information of the software development process are discussed. Also described is the identification of key factors for software development teams to consider in their software development practice as well as the

understanding of the elements in software development process that may affect software reliability.

### 8.2.1 Survey Analysis

The survey has two complementary sections (see Appendix 3). Section A data were collected using a formal survey questionnaire given directly by the software developers or managers in 13 organizations. The general information of the survey participants is shown in Table 8.1. The data were collected from March to May 1998. Demographic data on the participants is summarized in Table 8.2. Thirteen companies were chosen to maximize sample breadth for a mostly exploratory study, yet they have sufficient technical and managerial depth to provide reliable data. All organizations had software development projects either for safety-critical, commercial, or inside-user orientated applications. They had a relatively good mixture of software development experience, were diverse in size, program categories, and represented a good mixture of software development firms.

Section A of the survey used a Likert scale to identify the degree to which an increase of significance in each environmental factor (the independent variables) which typically makes software reliability assessment more accurate. In the survey form, 1 indicated “not significant” and 7 “most significant”. If these factors are irrelevant, scores of (or close to) 1 would be expected; if they do have a significant effect on software reliability assessment, mean scores would be statistically different from 1 (not significant). Similarly, factors with high scores could be deemed as having a greater impact than those with lower scores for comparative purposes. Finally, all the factors will be listed in order with the most significant at the top.

Section B in Appendix 3 sought personal and organizational professional data to explore possible relationships with the ranking of environmental factors. Some background data of the survey participants and the categories of the software application were collected. Information of software development effort allocation was also obtained. See Zhang and Pham (2000a) for detailed information.

Based on the survey information, the analysis, design, coding, and testing phases take, respectively, about 25, 18, 36, and 21% of the development efforts. Analysis and design testing phases together take about 64% of the total development time. People in different positions are found to have different opinions (Table 8.3). The significance of incorporating the factors into software reliability studies averages at 75%, ranging from 80% of the programmers, testers, and managers to 63% by other people. The distribution of the survey participants is also shown. For example, 9 out of 22 people are programmers, 4 are system engineers and so on.

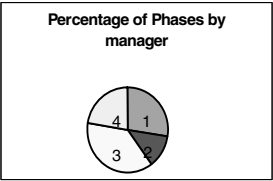
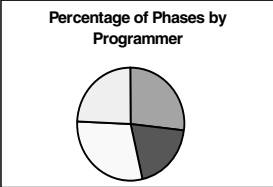
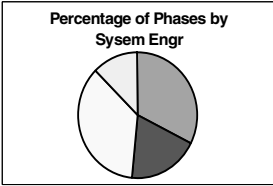
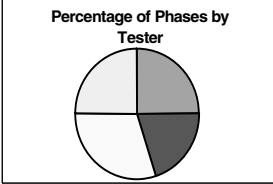
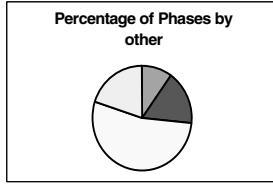
**Table 8.1.** General information of survey participants

Company name	Number of participants
MCI International Inc.	1
Lucent Technologies	2
General Electronics Capital	1
Peracom	1
IBM	1
Bellcore	6
AT &T	1
NEC	1
Chrysler	5
Hughes Network System Inc.	1
Level 1	1
BOC Gas	1
Texas Instruments	1
Total	23

**Table 8.2.** Demographic data of survey participants

Personal/demographic factor	Mean score	Sample size
1. Current job position	Manager: 9.09% System engineer: 18.2% Programmer: 40.9% Tester: 4.55% Other: 27.27%	22
2. Experience (years)	7.79	22
3. Significance of improvement	72.17%	23
4. Percentage of reused code	36.38%	18
5. Percentage of time spend in analysis	25.28%	18
6. Percentage of time spend in design	18.06%	18
7. Percentage of time spend in coding	35.83%	18
8. Percentage of time spend in testing	20.83%	18

Table 8.3. Summary by position

manager(2/22)										Average	<div>Percentage of Phases by manager</div> 
analysis	30	25								27.5	
design	20	5								12.5	
coding	30	45								37.5	
testing	20	25								22.5	
reused	10	20								15	
sig	80	80								80	
programmer(9/22)										Average	<div>Percentage of Phases by Programmer</div> 
analysis	10	10	25	30	40	20	50	30	0	26.9	
design	10	30	25	20	10	30	10	20	0	19.4	
coding	50	30	25	30	30	20	30	20	0	29.4	
testing	30	30	25	20	20	30	10	30	0	24.4	
reused	60	50	40	40	5	0	0	30	80	43.6	
sig	80	60	80	60	60	80	80	60	100	73.3	
system engineer(4/22)										Average	<div>Percentage of Phases by Sysem Engr</div> 
analysis	25	20	10	75						32.5	
design	30	20	20	5						18.8	
coding	35	40	60	10						36.3	
testing	10	20	10	10						12.5	
reused	35	50	60	20						41.3	
sig	60	80	60	80						70	
tester(1/22)										Average	<div>Percentage of Phases by Tester</div> 
analysis	25									25	
design	20									20	
coding	30									30	
testing	25									25	
reused	5									5	
sig	80									80	
other(6/22)										Average	<div>Percentage of Phases by other</div> 
analysis	20	5	0	5						10	
design	20	10	0	20						16.7	
coding	30	70	0	60						53.3	
testing	30	15	0	15						20	
reused	20	40	30	60						37.5	
sig	80	100	40	60	80	20				63.3	

Note: legend 1:analysis, 2:design, 3:coding, 4:testing

## 8.2.2 Statistical Methods

### Relative Weight Method

This is simply ranking and determining the relative weights of factors based solely on the participants' opinions as reflected in Section A of the survey forms. Under this methodology, every participant is treated equally without considering his/her background information.

Let  $r_{ij}$  be the original ranking of the  $i$ th factor on the  $j$ th survey and  $w_{ij}$  the corresponding normalized score as follows:

$$w_{ij} = \frac{r_{ij}}{\sum_{i=1}^n r_{ij}} \quad (8.1)$$

where  $n$  is the number of factors on the  $j$ th survey. Therefore  $\sum_{i=1}^n w_{ij} = 1$  for all  $j$ .

Different people may give different original ranking and some of them may give higher scores for all factors. Therefore, the summation of all the scores from  $f1$  to  $f32$  ranges from 117 to 200. By normalizing the original ranking scores using equation (8.1), the final weight for the  $i$ th factor can be written as

$$w_i^* = \frac{\sum_{j=1}^l w_{ij}}{l} \quad (8.2)$$

where  $l$  is the number of surveys used in this method.

### Analysis of Variance Method

Analysis of variance (ANOVA) model is a very versatile statistical tool for studying the relationship between a dependent variable and one or more independent variables. The response in the model is named dependent variable because its value depends on other variables, and explanatory variables are named independent variables since their values are not influenced by anything else.

A *factor* in ANOVA is an independent variable to be studied in the model. A *level* is a factor is a particular form or value of that factor. The primary purpose of ANOVA is to determine the effects of factors on the response and to single out the most significant factors.

### One-way ANOVA Model

$$r_{ij} = \mu_0 + \alpha_i + \varepsilon_{ij}$$

where

$r_{ij}$  is the original or observed response value

$\mu_0$  is a constant or intercept term in the model, the mean of all the cells

$\alpha_i$  is the main effect for a factor, say A, at the  $i$ th level

$\varepsilon_{ij}$  is the random error term, which follows normal distribution with mean 0

Using one-way ANOVA, we can perform the hypothesis tests to classify all factors into different groups according to the significance of their impact on the software reliability analysis.

### Two-way ANOVA Model

$$r_{ijk} = \mu_0 + \alpha_i + \beta_j + (\alpha * \beta)_{ij} + \varepsilon_{ijk}$$

where

$r_i$  is the original or observed response value

$\mu_0$  is a constant or intercept term in the model, the mean of all the cells

$\alpha_i$  is the main effect for a factor, say A, at the  $i$ th level

$\beta_j$  is the main effect for a factor, say B, at the  $j$ th level

$(\alpha * \beta)_{ij}$  is the interaction term of A and B at the  $ij$ th level

$\varepsilon_{ij}$  is the random error term

One-way ANOVA can be used to rank the weights of the factors and select the most important ones. Like relative weight method, one-way ANOVA treats the original ranking from the survey equally. Therefore, these two methods may have some bias in the analysis. The two-way ANOVA can be used to overcome this weakness.

Two-way ANOVA can be used to get rid of survey bias and adjust the mean ranking score of significance for each environmental factor. People who have different experience of software development may not have same opinions on the significance of the environmental factors. The background factors then can be the title, the experience and so on. These can be the factors of two-way ANOVA analysis, and each can have several levels. Also interaction between these influence factors can also be tested. After these analyses, the information can be used to adjust the mean ranking score for each environmental factor. Based on this information, further analyses can be conducted on the treatments of the survey data. The disadvantage of this model is its complexity. The model validations such as normality and independence can be obtained in Zhang and Pham (2000a).

Based on the information obtained from the survey data, Zhang and Pham (2000a) studied a number of hypotheses as follows:

*Hypothesis 1:* The significance of the impacts of the 32 factors on software reliability assessment is of the same level. Intuitively, the impacts of the 32 factors may not be the same. Some may have more significant impacts than others; then the ranking of these factors in terms of their impacts on software reliability assessment will be desirable.

*Hypothesis 2:* People playing different roles in software development have the same opinion on the significance of the 32 factors. Managers, system engineers, programmers, and testers may not have the same opinion on the significance of all these factors. This hypothesis will find out whether their opinion can be considered as “the same”.

*Hypothesis 3:* People developing software for different applications have the same opinion on the importance of the 32 factors. Safety-critical, commercial and inside-used systems are considered here.

### 8.3 Exploratory Analysis of Environmental Factors

#### Relative Weight Method

Table 8.4 shows the results by the relative weight method. The ten most important environmental factors are classified as factors in the analysis phase (three factors), coding (one factor), testing (four factors), and general (two factors). The column “Normalized priorities” gives the contribution of each environmental factor. For example, program complexity factor contributes approximately 3.7% (its relative weight = 0.03768). A higher priority value indicates a higher ranking. The application of this finding in Table 8.4 is not to discard the environmental factors belonging to lower ranking classes, but hopefully, to help software developers or managers prioritize their tasks.

#### ANOVA

ANOVA method was performed on each quantitative survey variable including all environmental factors and mean and variance of these factors are calculated. The final ranking based on this information listed in Table 8.5. It seems that the final ranking of the environmental factors is consistent with the one we got from relative weight method. For example, the top 10 factors remain the same except that factor # 8 (frequency of specification change) ranks two positions down.

ANOVA method also classified the factors into several groups in terms of their importance. The first five factors are the first class, which is the most important factors, the next five factors belong to the second group and so on. This finding can be used to help software developers to determine which are the most important groups of environmental factors subject to the available resources.

#### Correlation Analysis

Correlation analysis is also studied based on the survey information. The purpose is to find out the correlation of environmental factors and determine if they are independent or not. (If not, then which factors are related to each other?) Table 8.6 shows the correlation of the environmental factors. Correlation analysis aims at finding out the correlation among the factors. In other words, to find out whether the factors are independent or not. If not, which factors are related to each other? In this section, we present the result obtained from a correlation test of the factors.

For example, Factor 1 (program complexity) is statistical significantly correlated with Factor 17 (development team size). For those correlated factors, we may not want to include all of them in the software reliability models provided that one has been considered. This is because by considering one of these related factors we already take the contributions of these factors into consideration. Including the correlated ones will not make much additional contribution but just increase the complexity of the model.



**Table 8.4.** Results ranking based on relative weight method

Rank	Rank factors	Factor name	Normalized priorities
1	f1	Program complexity	0.03768
2	f15	Programmer skills	0.03693
3	f25	Testing coverage	0.03675
4	f22	Testing effort	0.03650
5	f21	Testing environment	0.03533
6	f8	Frequency of specification change	0.03483
7	f24	Testing methodologies	0.03433
8	f11	Requirements analysis	0.03417
9	f6	Percentage of reused code	0.03369
10	f12	Relationship of detailed design, requirement	0.03330
11	f5	Level of programming technologies	0.03315
12	f27	Documentation	0.03281
13	f18	Program workload	0.03275
14	f26	Testing tools	0.03227
15	f16	Programmer organization	0.03210
16	f19	Domain knowledge	0.03180
17	f3	Difficulty of programming	0.03171
18	f10	Design methodologies	0.03171
19	f20	Human nature (mistake and omission)	0.03169
20	f14	Development management	0.03166
21	f23	Testing resource allocation	0.03096
22	f4	Amount of programming effort	0.03072
23	f2	Program categories	0.03058
24	f13	Work standards	0.02985
25	f32	System software	0.02839
26	f9	Volume of program design documents	0.02750
27	f17	Development team size	0.02738
28	f7	Programming language	0.02711
29	f28	Processor	0.02414
30	f31	Telecommunication device	0.02404
31	f30	Input/output device	0.02291
32	f29	Storage device	0.02127

**Table 8.5.** Final ranking based on ANOVA method

SNK grouping	Mean	N	Factor no.	Factor name	Final grouping
A	6.0435	23	f1	Program complexity	1
A	6.0000	23	f15	Programmer skills	1
A	5.9565	23	f25	Testing coverage	1
A	5.9565	23	f22	Testing effort	1
A	5.7826	23	f21	Testing environment	1
B A	5.6087	23	f24	Testing methodologies	2
B A	5.6087	23	f11	Requirements analysis	2
B A	5.6087	23	f8	Frequency of spec change	2
B A	5.5652	23	f6	Percentage of reused code	2
B A	5.5000	22	f18	Program work load	2
B A C	5.4762	23	f12	Relationship of detailed design and requirement	3
B A C	5.4348	23	f5	Level of programming technologies	3
B A C	5.3913	23	f27	Documentation	3
B A C	5.3636	22	f26	Testing tools	3
B A C	5.2727	22	f19	Domain knowledge	3
B A C	5.2174	23	f23	Difficulty of programming	3
B A C	5.1905	21	f23	Testing resource allocation	3
B A C	5.1739	23	f10	Design methodologies	3
B A C	5.1739	23	f16	Programmer organization	3
B A C	5.1364	22	f14	Development management	3
B A C	5.1304	23	f20	Human nature (mistake and omission)	3
B A C	5.0909	22	f4	Amount of programming effort	3
B A C	5.0000	23	f2	Program categories	3
B A C	5.0000	20	f13	Work standards	3
B D A C	4.7727	22	f32	System software	4
B D A C	4.4783	23	f7	Programming language	4
B D A C	4.4348	23	f17	Development team size	4
B D A C	4.4286	21	f9	Volume of program design documents	4
B D C	4.0909	22	f28	Processor	5
B D C	4.0455	22	f31	Telecommunication device	5
D C	3.90916	22	f30	Input / output device	6
D	3.5455	22	f29	Storage device	7

## 8.4 Further Exploratory Analysis

This section considers those 11 top factors in Table 8.4 by looking at ways how to combine those related factors so that the dimension of the factor can be reduced without losing much information (Zhang *et al.* 2001a). Table 8.7 presents a list of the top 11 ranking environmental factors.

A factor analysis method is used to find the explanation of relationships among the EFs to derive a small number of linear combinations of EFs that retain as much of the information in the original EFs as possible. This linear combination of EFs will be called 'common factor' to avoid confusion with environmental factors and it will be used in place of the original EFs for regression analysis later. Suppose the environmental factors can be grouped by their correlation; then the factors that belong to the same group are highly correlated among themselves but have relatively small correlation with factors in a different group.

### Factor Analysis

The factor analysis based on those 11 EFs is discussed. The eigenvalues of the correlation, the proportion of variation represented, and the cumulative proportions of variation are summarized in Table 8.8. It shows that the common factors (denoted by  $C_i$ s) and the first four factors,  $C_1$  through  $C_4$ , have eigenvalues greater than 1 and a drop below 10% of the variance explained after the  $C_4$ . Therefore, four common factors are retained.

To figure out the characteristic of the common factors we chose the EFs with factor loading greater than 0.6 in absolute value and listed them in Table 8.9. Factor loadings describe the correlation between the common factors emerging from a factor analysis and the original EFs used in the construction of the factors. The higher the factor loading for a given EF, the more that the EF contributes to the factor score. Note that the first common factor,  $C_1$ , has high loadings which exceed 0.6 for f5, f12, f21 and f22. Since it is difficult to pin a specific label on the first common factor we call it the 'Overall' factor. In many cases, the first common factor represents an overall measure of the information contained in all the variables. The first common factor explains 32.58% (Table 8.8) of the total variation. f6, f24, and f25 have large loadings on the second common factor,  $C_2$ . These EFs related to the index of testing efficiency. Therefore, the second common factor is called the 'Testing Efficiency' factor. Similarly, we call the third and fourth common factors the 'Requirement and Specification' factor and 'Program and Skill' factor, respectively.

**Table 8.6.** Correlation of environmental factors

Factor no.	Name of factor	Correlated factors
f1	Program complexity	f 17 Development team size
f15	Programmer skills	f 3 Difficulty of programming
		f 18 Program workload
f25	Testing coverage	f 2 Program categories
		f 24 Testing methodologies
f22	Testing effort	f 5 Level of programming technologies
		f 13 Work standards
		f 14 Development management
		f 21 Testing environment
f21	Testing environment	f 5 Level of programming technologies
		f 12 Relationship of detailed design and requirement
		f 13 Work standards
		f 22 Testing effort
f8	Frequency of specification change	
f24	Testing methodologies	f 25 Testing coverage
		f 26 Testing tools
f11	Requirements analysis	f 12 Relationship of detailed design and requirement
		f 30 Input/output device
		f 32 System software
f6	Percentage of reused code	f 7 Programming language
		f 9 Volume of program design documents
		f 14 Development management
		f 18 Program workload
		f 23 Testing resource allocation
		f 24 Testing methodologies
		f 26 Testing tools
		f 27 Documentation
f12	Relationship of detailed design and requirement	f 11 Requirements analysis
		f 21 Testing environment
f5	Level of programming technologies	f 21 Testing environment
		f 22 Testing effort
f27	Documentation	f 6 Percentage of reused code
		f 26 Testing tools
		f 31 Telecommunication device
f18	Program workload	f 3 Difficulty of programming
		f 4 Amount of programming effort
		f 7 Programming language
		f 13 Work standards
		f 15 Programmer skills
		f 23 Testing resource allocation
		f 26 Testing tools
		f 30 Input/output device

**Table 8.6.** (continued)

Factor no.	Name of factor	Correlated factors
f26	Testing tools	f 6 Percentage of reused code f 18 Program workload f 27 Documentation
f16	Programmer organization	
f19	Domain knowledge	
f3	Difficulty of programming	f 6 Percentage of reused code f 15 Programmer skills f 18 Program workload
f10	Design methodologies	f 2 Program categories
f20	Human nature (mistake and omission)	
f14	Development management	f 22 Testing effort
f23	Testing resource allocation	f 6 Percentage of reused code f 24 Testing methodologies
f4	Amount of programming effort	f 6 Percentage of reused code f 7 Programming language f 18 Program workload
f2	Program categories	f 10 Design methodologies f 25 Testing coverage
f13	Work standards	f 14 Development management f 18 Program workload f 21 Testing environment f 22 Testing effort
f32	System software	f 11 Requirements analysis f 28 Processor f 30 Input/output device f31 Telecommunication device
f9	Volume of program design documents	f 6 Percentage of reused code
f17	Development team size	f 1 Program complexity f 18 Program workload
f7	Programming language	f 4 Amount of pro. effort f 6 Percentage of reused code f 18 Program workload
f28	Processor	f 29 Storage device f 31 Telecom. device
f31	Telecommunication device	f 27 Documentation f 28 Processor f 32 System software
f30	Input/output device	f 28 Processor
f29	Storage device	f 30 Input/output device

**Table 8.7.** Top 11 ranking EFs based on relative weight method

Rank	EF	Name
1	f1	Program complexity
2	f15	Programmer skills
3	f25	Testing coverage
4	f22	Testing effort
5	f21	Testing environment
6	f8	Frequency of specification change
7	f24	Testing methodologies
8	f11	Requirements analysis
9	f6	Percentage of reused code
10	f12	Relationship of detailed design and requirement
11	f5	Level of programming technologies

**Table 8.8.** Eigenvalue of the correlation matrix

Common factor	Eigenvalue	Percentage	Cumulative percentage
C <sub>1</sub>	3.5836	32.58	32.58
C <sub>2</sub>	1.7940	16.31	48.89
C <sub>3</sub>	1.5753	14.32	63.21
C <sub>4</sub>	1.3039	11.85	75.06
C <sub>5</sub>	0.8316	7.56	82.62
C <sub>6</sub>	0.7020	6.38	89.00
C <sub>7</sub>	0.4288	3.90	92.90
C <sub>8</sub>	0.3616	3.29	96.19
C <sub>9</sub>	0.2622	2.38	98.57
C <sub>10</sub>	0.1120	1.02	99.59
C <sub>11</sub>	0.0450	0.41	100.00

### Regression Analysis

In this section, we examine the relationships between the four common factors and the software reliability assessment improvement based on multiple linear regression. The weighted linear combinations of Efs ( $X_i$ ;  $i = 1, \dots, 4$ ) for each index and the improvement of the accuracy of software reliability assessment (IASRA) (Section B of the survey form in Zhang and Pham 2000a) are considered as independent variables and dependent variable, respectively.

Let  $k_{ij}$  be the  $j$ th loading on the  $i$ th index and  $l_{ij}$  be the normalized loading using the following form:

$$l_{ij} = \frac{k_{ij}}{\sum_{ij} k_{ij}}, \quad i = 1, \dots, 4$$

The linear combination of EFs that will be used as independent variables is

$$X_i = \sum_j l_{ij} * EF_{ij}$$

where  $EF_{ij}$  denotes the  $j$ th EF score in the  $i$ th common factor  $C_i$ . Here  $X_1$  and  $X_2$  represent the weighted linear combinations of 'Overall' and 'Testing Efficiency' index scores, respectively.  $X_3$  and  $X_4$  represent the weighted linear combinations of 'Requirement and Specification' and 'Program and Skill Level' index score for each one.

**Table 8.9.** Identification of the common factors

Common factor	Index	EF	Name	Loading
C <sub>1</sub>	Overall	F21	Testing environment	0.926
		F22	Testing effort	0.803
		F5	Level of programming	0.718
		F12	Technologies relationship of detailed design and requirements	0.696
C <sub>2</sub>	Testing efficiency	F24	Testing methodologies	0.844
		F25	Testing coverage	0.823
		F6	Percentage of reused code	0.714
C <sub>3</sub>	Requirements and specification	F11	Requirements analysis	0.826
		F8	Frequency of specification change	0.603
C <sub>4</sub>	Program and skill level	F15	Programmer skills	0.845
		F1	Program complexity	0.605

There are a total of 15 different linear regression models: 4 simple linear regression models with only 1 independent variable each: 6 linear regression models with 2 independent variables: 4 linear regression models with 3 independent variables: and one full linear regression with all 4 independent variables. Among the 15 linear regression models, six models turned out to be significant which include all simple regression models (Models I, II, III, and IV) and 2 models (Model V and VI) with 2 independent variables (Zhang *et al.* 2001a). The intercept is not statistical significant in any of these regression models. The results are summarized in Table 8.10.

From Table 8.10, for example, the regression model VI of the estimate of IASRA,  $IAS\hat{S}RA$ , can be expressed as the linear equation of  $X_3$  and  $X_4$  such that

$$IAS\hat{S}RA = 3.31 * X_3 + 2.76 * X_4.$$

This equation implies that the predicted score for the improvement of the accuracy of software reliability assessment increases as the 'Requirement and Specification' index and in the 'Program and Skill Level' index increases.

**Table 8.10.** Parameter estimates based on the common factors

Model	No. of variables in model	Variables	Parameter estimate	P-value	R <sup>2</sup>
I	1	$X_1$	1.29	0.0001	0.9433
II	1	$X_2$	2.20	0.0001	0.9332
III	1	$X_3$	6.29	0.0001	0.9368
IV	1	$X_4$	5.68	0.0001	0.9348
V	2	$X_2$	1.08	0.0571	0.9464
		$X_4$	2.97	0.0436	
VI	2	$X_3$	3.31	0.0369	0.9485
		$X_4$	2.76	0.0522	

**Testing Between - Phases Within the Development Process**

Thirty-two EFs are now divided into five categories: 'General', 'Analysis and Design', 'Coding', 'Testing' and 'Hardware systems'. In this analysis, we consider 'Analysis and Design', 'Coding' and 'Testing' as the software development process phase (Zhang *et al.* 2001a). The null hypothesis is that these phases are of the same significant level. The results of one-way ANOVA including the Student-Newman-Keuls (SNK) multiple comparison tests are summarized in Table 8.11.

**Table 8.11.** SNK test result

SNK grouping	Phase	Mean
A	Testing	5.43
A	Coding	5.35
A	General	5.24
A	Analysis and design	5.03

The results show that these phases have slightly different mean values, ranging from 5.43 for the 'Testing' phase to 5.03 for the 'Analysis and design' phase. The same letter in the SNK grouping shown in the first column of Table 8.11 indicates that the different among the mean values are not statistically significant. Zhang and Pham (2000a) studied the time allocation for each of the development phases and it was found that the requirement analysis, design, coding, and testing takes about 25, 18, 36, and 21% of the entire development time. However, this result indicates that each development phase is considered equally important in terms of their impact on software reliability assessment.



**Identifying Significant Environmental Factors Within Each Phase**

This section discusses the most important subsets of environmental factors describing the relationship between software reliability assessment and environmental factors for each phase. We consider the improvement of IASRA as a dependent variable and the 32 environmental factors as independent variables. The significant factors and their parameter estimates based on a linear regression backward elimination method are presented in Table 8.12.

**Table 8.12.** Significant EFs for each phase

Phase	Variables	Name	Parameter estimate	P-value	R <sup>2</sup>
General	f1	Program complexity	9.17	0.0001	0.9697
	f6	Percentage of reused modules	3.17	0.0907	
Analysis and design	f8	Frequency of program specification change	3.37	0.0635	0.9801
	f10	Design methodology	4.90	0.0063	
	f13	Work standards	6.42	0.0068	
Coding	f17	Development team size	8.88	0.0192	0.9551
	f19	Domain knowledge	6.46	0.0341	
Testing	f21	Testing environment	12.57	0.0001	0.9703

The hypothesis of zero intercept was not rejected for any phase regression model. Therefore, it may be appropriate to remove the constant term from the model. Note that the program complexity (f1) and percent of reused modules (f6) are significant for ‘General’ phase with p-value 0.0001 and 0.0907, respectively. That means program complexity and percent of reused modules provides significant information for the prediction of software reliability assessment in ‘General’ phase. Similar interpretation can be represented in the different development phase. More findings can be obtained in Zhang *et al.* (2001a) and Zhang and Pham (2000a).

**8.5 A Generalized Model with Environmental Factors**

In this section, we discuss several newly developed software reliability models that consider environmental factors by combining the proportional hazard model (Cox 1975) and existing software reliability models (Pham 2000a). Such factors are, *e.g.*, the complexity metrics of the software, the development and environmental conditions, the effect of mental stress and human nature, the level of the test-team

members, and the facility level during testing. The proportional hazard model has been widely used in medical applications to estimate the survival rate of patients.

#### Notation

$\tilde{z}$	Vector of environmental factors
$\tilde{\beta}$	Coefficient vector of environmental factors
$\Phi(\tilde{\beta} \tilde{z})$	Function of environmental factors
$\lambda_0(t)$	Failure intensity rate function without environmental factors
$\lambda(t, \tilde{z})$	Failure intensity rate function with environmental factors
$m_0(t)$	Baseline mean value function without environmental factors
$m(t, \tilde{z})$	Mean value function with environmental factors
$R_0(x/t)$	Baseline reliability function without environmental factors
$R(x/t, \tilde{z})$	Reliability function with environmental factors

Chapter 6 has discussed the fault intensity rate function  $\lambda(t)$  and the mean value function  $m(t)$  based on NHPP without environmental factors. In this section, a fault intensity rate function that integrates environmental factors based on a proportional hazard model can be constructed using the following assumptions:

1. The fault intensity rate function consists of two categories: the fault intensity rate functions without environmental factors,  $\lambda_0(t)$ , and the environmental factor function,  $\Phi(\tilde{\beta} \tilde{z})$ .
2. The fault intensity rate function  $\lambda_0(t)$  and the function of the environmental factors are independent. The function  $\lambda_0(t)$  is also called the baseline intensity function.

Based on the proportional hazard model (PHM), let us consider the failure intensity function of a software system as the product of an unspecified baseline failure intensity  $\lambda_0(t)$ , a function that only depends on time, and environmental factor function  $\Phi(\tilde{\beta} \tilde{z})$  incorporating the effects of a number of environmental factors.

The fault intensity function with environmental factors,  $\lambda(t, \tilde{z})$ , can be expressed as:

$$\lambda(t, \tilde{z}) = \lambda_0(t) \cdot \Phi(\tilde{\beta} \tilde{z}) \quad (8.3)$$

The mean value function with environmental factors then can be obtained as follows:

$$m(t, \tilde{z}) = \int_0^t \lambda_0(s) \Phi(\tilde{\beta} \tilde{z}) ds = \Phi(\tilde{\beta} \tilde{z}) \int_0^t \lambda_0(s) ds = \Phi(\tilde{\beta} \tilde{z}) m_0(t) \quad (8.4)$$

The reliability function with environmental factors can be expressed as follows:

$$\begin{aligned}
R(x/t, \bar{z}) &= e^{-(m(t+x, \bar{z}) - m(t, \bar{z}))} \\
&= e^{-(\Phi(\bar{\beta} \bar{z}) m_0(t+x, \bar{z}) - \Phi(\bar{\beta} \bar{z}) m_0(t, \bar{z}))} \\
&= [R_0(x/t)]^{\Phi(\bar{\beta} \bar{z})}
\end{aligned} \tag{8.5}$$

The basic assumption for PHM is that the ratio of the failure intensity functions of any two errors observed at any time  $t$  associated with any environmental factor sets  $z_{li}$  and  $z_{2i}$  is a constant with respect to time and they are proportional to each other. In other words,  $(t_i, z_{li})$  is directly proportional to  $(t_i, z_{2i})$ .

Assuming the exponential function of environmental form, then a failure intensity function of the software reliability model that considers environmental factors can be written as

$$\lambda(t_i; z_i) = \lambda_0(t_i) e^{(\sum_{j=1}^m \beta_j z_{ji})} \tag{8.6}$$

where

$z_{ji}$	environmental factor $j$ of the $i$ th error
$\beta_j$	regression coefficient of the $j$ th factor
$t_i$	failure time between the $(i-1)$ th error and $i$ th error, $i = 1, 2, \dots, n$
$z_i$	environmental factor of the $i$ th error
$m$	number of environmental factors.

It is easy to see that  $\lambda_0(t)$  is a baseline failure intensity function that represents the failure intensity when all environmental factors variables are set to zero.

Let  $Z$  be a column vector consisting of the environmental factors and  $B$  represents a row vector consisting of the corresponding regression parameters. Then the above failure intensity model can be rewritten as

$$\lambda(t; Z) = \lambda_0(t) e^{(BZ)} \tag{8.7}$$

Therefore, the reliability of the software systems can be written in a general form, as follows:

$$\begin{aligned}
R(t; Z) &= e^{-\int_0^t \lambda_0(s) e^{BZ} ds} \\
&= \left[ e^{-\int_0^t \lambda_0(s) e^{BZ} ds} \right]^{e^{BZ}} \\
&= [R_0(t)]^{e^{BZ}}
\end{aligned} \tag{8.8}$$

where  $R_0(t)$  is the time-dependent software reliability. The pdf of the software system is given by

$$\begin{aligned}
f(t; Z) &= \lambda(t; Z) \cdot R(t; Z) \\
&= \lambda_0(t) e^{BZ} [R_0(t)]^{e^{BZ}}
\end{aligned}$$

The regression coefficient  $B$  can be estimated, using either the MLE method or the maximum partial likelihood approach, which is discussed later, without assuming any specific distributions about the failure data and estimating the baseline failure intensity function. A direct generalization of the above model in equation (8.7) is that one may want to consider the environmental factor variables  $Z_{ji}$  as a function of time. In this case, a mathematical generalized form of the failure intensity function is given by

$$\lambda(t; Z) = \lambda_0(t) e^{\left[ \sum_{j=1}^m \beta_j z_{ji}(t) \right]} \quad (8.9)$$

## 8.6 Environmental Parameter Estimation

In this section, we will discuss how to estimate the parameters in the environmental factor model by using two widely used methods: the MLE method and the partial likelihood method. The advantage of the partial likelihood method is that it does not require as much data as the typical maximum likelihood method. Therefore, the data collection required by regression method can be simplified. Information of similar applications and settings of environmental factors that have been stored in databases can be utilized.

### Environmental Factors Estimation Using MLE

Assume that there are  $p$  unknown parameters in the baseline failure intensity function  $\lambda_0(t)$ , say  $\alpha_1, \alpha_2, \dots, \alpha_p$ , and there are  $m$  environmental factors  $\beta_1, \beta_2, \dots, \beta_m$ . Let  $A = (\alpha_1, \alpha_2, \dots, \alpha_p)$  be a set of unknown parameters  $\alpha_1, \alpha_2, \dots, \alpha_p$ , and  $B$  be a set of  $\beta_1, \beta_2, \dots, \beta_m$ . Then the likelihood function is given by

$$\begin{aligned} L(A, B) &= \prod_{i=1}^n f(t_i; z_i) \\ &= \prod_{i=1}^n \left( \lambda_0(t_i) e^{\left( \sum_{j=1}^m \beta_j z_{ji} \right)} [R_0(t_i)] e^{\left( \sum_{j=1}^m \beta_j z_{ji} \right)} \right) \end{aligned} \quad (8.10)$$

The log likelihood function is given by

$$\ln L(A, B) = \sum_{i=1}^n \ln[\lambda_0(t_i)] + \sum_{i=1}^n \sum_{j=1}^m \beta_j z_{ji} + \sum_{i=1}^n e^{\left( \sum_{j=1}^m \beta_j z_{ji} \right)}$$

Taking the first partial derivatives of the log likelihood function with respect to  $(m+p)$  parameters, we obtain

$$\frac{\partial}{\partial \alpha_k} [\ln L(A, B)] = \sum_{i=1}^n \frac{\frac{\partial}{\partial \alpha_k} [\lambda_0(t_i)]}{\lambda_0(t_i)} + \sum_{i=1}^n e^{\left( \sum_{j=1}^m \beta_j z_{ji} \right)} \frac{\frac{\partial}{\partial \alpha_k} [R_0(t_i)]}{R_0(t_i)}$$

$$\frac{\partial}{\partial \beta_s} [\ln L(A, B)] = \sum_{i=1}^n z_{si} + \sum_{i=1}^n z_{si} e^{\left( \sum_{j=1}^m \beta_j z_{ji} \right)} \ln[R_0(t_i)]$$

where  $k = 1, 2, \dots, p$  and  $s = 1, 2, \dots, m$ .

Setting the previous equations equal to zero, we can obtain all the  $(m + p)$  parameters by solving the following system of  $(m + p)$  equations simultaneously:

$$\sum_{i=1}^n \left[ \frac{\frac{\partial}{\partial \alpha_k} [\lambda_0(t_i)]}{\lambda_0(t_i)} + e^{\left( \sum_{j=1}^m \beta_j z_{ji} \right)} \frac{\frac{\partial}{\partial \alpha_k} [R_0(t_i)]}{R_0(t_i)} \right] = 0 \quad \text{for } k = 1, 2, \dots, p$$

$$\sum_{i=1}^n z_{si} \left[ 1 + e^{\left( \sum_{j=1}^m \beta_j z_{ji} \right)} \ln[R_0(t_i)] \right] = 0 \quad \text{for } s = 1, 2, \dots, m$$

### Environmental Factors Estimation Using Maximum Partial Likelihood Approach

According to the idea of Cox's proportional hazard model, we can use the maximum partial likelihood method to estimate environmental factors without assuming any specific distributions about the failure data and estimating the baseline failure intensity function. The only basic assumption of this model is that the ratio of the failure intensity functions of any two errors observed at any time  $t$  associated with any environmental factor sets  $z_{1i}$  and  $z_{2i}$  is constant with respect to time and they are proportional to each other.

First we estimate the environmental factor parameters based on the partial likelihood function. The partial likelihood function of this model is given by

$$L(B) = \prod_{i=1}^n \frac{e^{(\beta_1 z_{1i} + \beta_2 z_{2i} + \dots + \beta_m z_{mi})}}{\sum_{k \in R_i} e^{(\beta_1 z_{1k} + \beta_2 z_{2k} + \dots + \beta_m z_{mk})}} \quad (8.11)$$

where  $R_i$  is the risk set at  $t_i$ . Take the derivatives of the log partial likelihood function with respect to  $\beta_1, \beta_2, \dots, \beta_m$  and let them equal zero. Therefore, we can obtain all of the estimated  $\beta_s$  by solving these equations simultaneously using numerical methods. After estimating the factor parameters  $\beta_1, \beta_2, \dots, \beta_m$ , the remaining task is to estimate the unknown parameters of the baseline failure intensity function  $\lambda_0(t)$ .

## 8.7 Enhanced Proportional Hazard Jelinski-Moranda (EPJM) Model

Recall that the Jelinski-Moranda (JM) model is one of the earliest models developed for predicting software reliability (see Chapter 5). The failure intensity of the software at the  $i$ th failure interval of this model is given by

$$\lambda(t_i) = \phi[N - (i - 1)] \quad i = 1, 2, \dots, N$$

and the probability density function is given by

$$f(t_i) = \phi[N - (i - 1)]e^{-\phi[N - (i - 1)]t_i}$$

The enhanced proportional hazard JM model (Pham 2000a), called the EPJM model, which is based on the proportional hazard and J M model, is expressed as

$$\lambda(t_i; z_i) = \phi[N - (i - 1)]e^{\left(\sum_{j=1}^m \beta_j z_{ji}\right)} \quad (8.12)$$

and the pdf corresponding to  $(t_i, z_i)$  is given by

$$f(t_i; z_i) = \phi[N - (i - 1)]e^{\left(\sum_{j=1}^m \beta_j z_{ji}\right)} e^{-\left[\phi[N - (i - 1)]t_i e^{\left(\sum_{j=1}^m \beta_j z_{ji}\right)}\right]}$$

Now we wish to estimate the parameters of the EPJM model using the two methods discussed in Section 5, the maximum likelihood method and the maximum partial likelihood method. There are  $(m + 2)$  unknown parameters in this model.

### The Maximum Likelihood Method

From equation (8.10), the likelihood function of the model is given by

$$\begin{aligned} L(B, N, \phi) &= \prod_{i=1}^n f(t_i; z_i) \\ &= \prod_{i=1}^n \left( \phi[N - (i - 1)] e^{\left(\sum_{j=1}^m \beta_j z_{ji}\right)} e^{-\phi[N - (i - 1)]t_i e^{\left(\sum_{j=1}^m \beta_j z_{ji}\right)}} \right) \end{aligned}$$

The log likelihood function is given by

$$\begin{aligned} \ln L(B, N, \phi) &= n \ln \phi + \sum_{i=1}^n \ln[N - (i - 1)] + \sum_{i=1}^n \left( \sum_{j=1}^m \beta_j z_{ji} \right) \\ &\quad - \sum_{i=1}^n \phi[N - (i - 1)]t_i e^{\sum_{j=1}^m (\beta_j z_{ji})} \end{aligned}$$

Taking the first partial derivatives of the log likelihood function with respect to  $(m+2)$  parameter:  $\beta_1, \beta_2, \dots, \beta_m, N$ , and  $\phi$ , we obtain the following:

$$\begin{aligned} \frac{\partial \log L}{\partial \phi} &= \frac{n}{\phi} - \sum_{i=1}^n [N - (i - 1)]t_i e^{\sum_{j=1}^m (\beta_j z_{ji})} \\ \frac{\partial \log L}{\partial N} &= \sum_{i=1}^n \frac{1}{[N - (i - 1)]} - \phi \sum_{i=1}^n t_i e^{\sum_{j=1}^m (\beta_j z_{ji})} \end{aligned}$$

and

$$\frac{\partial \log L}{\partial \beta_j} = \sum_{i=1}^n z_{ji} - \sum_{i=1}^n \phi[N - (i-1)] t_i z_{ji} e^{\sum_{j=1}^m (\beta_j z_{ji})}$$

Setting all of these equations equal to zero, we can obtain the estimated  $(m+2)$  parameters by solving the following system equations simultaneously using a numerical method:

$$\sum_{i=1}^n [N - (i-1)] t_i e^{\sum_{j=1}^m (\beta_j z_{ji})} = \frac{n}{\phi}$$

$$\sum_{i=1}^n \frac{1}{[N - (i-1)]} = \phi \sum_{i=1}^n t_i e^{\sum_{j=1}^m (\beta_j z_{ji})} \quad (8.13)$$

$$\sum_{i=1}^n \phi[N - (i-1)] t_i z_{ji} e^{\sum_{j=1}^m (\beta_j z_{ji})} = \sum_{i=1}^n z_{ji} \quad \text{for } j = 1, 2, \dots, m$$

### The Maximum Partial Likelihood Method

Assume that the baseline failure intensity has the form of the JM model. That means that the basic assumption of this model (see Section 4) is satisfied and that the ratio of the failure intensity functions of any two errors observed at any time  $t$ , associated with any environmental factor sets  $z_{1i}$  and  $z_{2i}$ , is a constant with respect to time and they are proportional to each other.

Having estimated the factor parameters  $\beta_1, \beta_2, \dots, \beta_m$  the remaining task is to estimate the unknown parameters of the baseline failure intensity function. Note that the failure intensity function model has the form

$$\lambda(t_i; z_i) = \phi[N - (i-1)] e^{(\hat{\beta}_1 z_{1i} + \hat{\beta}_2 z_{2i} + \dots + \hat{\beta}_m z_{mi})}$$

$$= \phi[N - (i-1)] E_i$$

where

$$E_i = e^{(\hat{\beta}_1 z_{1i} + \hat{\beta}_2 z_{2i} + \dots + \hat{\beta}_m z_{mi})}$$

The pdf is given by

$$f(t_i; z_i) = \phi E_i [N - (i-1)] e^{-(\phi E_i [N - (i-1)] t_i)}$$

The likelihood function is given by

$$L(N, \phi) = \prod_{i=1}^n (\phi E_i [N - (i-1)] e^{-(\phi E_i [N - (i-1)] t_i)})$$

By taking the log of the likelihood function and its derivatives with respect to  $N$  and  $\phi$ , and setting them equal to zero, we obtain the following equations:

$$\frac{\partial \ln L}{\partial N} = \sum_{i=1}^n \frac{1}{N - (i-1)} - \sum_{i=1}^n \phi E_i t_i = 0$$

and

$$\frac{\partial \ln L}{\partial \phi} = \frac{n}{\phi} - \sum_{i=1}^n E_i [N - (i-1)] t_i = 0$$

The estimated  $N$  and  $\phi$  can be obtained as follows. First, the parameter  $N$  can be obtained by solving the following equation:

$$\left( \sum_{i=1}^n E_i [N - (i-1)] t_i \right) \left( \sum_{i=1}^n \frac{1}{[N - (i-1)]} \right) = n \sum_{i=1}^n E_i t_i \quad (8.14)$$

After finding  $N$ , the parameter can easily be obtained and is given by

$$\phi = \frac{\sum_{i=1}^n \frac{1}{[N - (i-1)]}}{\sum_{i=1}^n E_i t_i} \quad (8.15)$$

## 8.8 Applications

Almost all software reliability engineering models need one of two basic types of input data: time-domain data and interval-domain data. One can possibly transform between the two types of data domains. The time-domain approach is characterized by recording the individual times at which the failure occurred. The interval-domain approach is characterized by counting the number of failures that occurred over a given period.

**Application 8.1:** To illustrate the EPJM model, we use the software failure data reported by Musa (1975) and also refer to data set #9 in Chapter 4. The data is related to a real-time command and control system. There is, however, no record of corresponding environmental factor measures in most, if not all, existing available data. To demonstrate the use of this model, we generate a failure-cluster factor and give its value which is logically realistic based on the failure data and consultation with several local software firms by the author.

One of the assumptions of the J-M model is that the time between failures is independent. As in many real testing environments, the failure times indeed occur in a cluster, *i.e.*, the failure time within a cluster is relatively shorter than that between the clusters. Data set #9 shows that it is reasonable in that particular application. This may indicate that the assumption of independent failure time is not correct. We can enhance the J-M model considering the failure-cluster factor by generating this factor based on the failure data.

We assume that if the present failure time, compared to the previous failure time, is relatively short, then some correlation may exist between them. Let us define a failure-cluster factor, such as



$$z_i = \begin{cases} 1 & \text{when } \frac{t_{i-1}}{t_i} \geq 7 \text{ or } \frac{t_{i-2}}{t_i} \geq 5 \\ 0 & \text{otherwise} \end{cases}$$

The data used in this model include both the failure time data and the explanatory environmental factor data (see Table 8.13). The explanatory variable data is dynamic, that is, it changes depending on the failure time. For example, in Table 8.13, the time between the fourth and fifth errors is 115 seconds; the time between the fifth and sixth errors is 9 seconds. Therefore,  $z_5$  is assigned to 0 and  $z_6$  is equal to 1.

For the J-M model, using the MLE, we obtain the estimate of the two parameters,  $N$  and  $\phi$ , as follows:

$$\hat{N} = 142$$

$$\hat{\phi} = (3.48893) \times 10^{-5}$$

Therefore, the current reliability of the software system is given by

$$R(t_{137}) = e^{-\hat{\phi}[\hat{N} - (137-1)]t_{137}}$$

Now, we want to predict the future failure behavior using only data collected in the past after 136 errors have been found. For example, the reliability of the software for the next 100 seconds after 136 errors are detected is given by

$$\begin{aligned} R(t_{137} = 100) &= e^{-\hat{\phi}[\hat{N} - (137-1)]t_{137}} \\ &= e^{-(3.48893 \times 10^{-5})[142-136](100)} \\ &= 0.979284 \end{aligned}$$

Similarly, the reliability of the software for the next 1000 seconds is given by

$$\begin{aligned} R(t_{137} = 1000) &= e^{-(0.0000348893[142-136](1,000))} \\ &= 0.811123 \end{aligned}$$

Assume that we use the partial likelihood approach to estimate the environmental factor parameter for the EPJM model. As there is only one factor in this example, we can easily obtain the estimated parameter using the statistical software package SAS:

$$\hat{\beta}_1 = 1.767109$$

with a significance level of 0.0001. Then the estimates of  $N$  and  $\phi$  are given as follows:

$$\hat{N} = 141$$

$$\hat{\phi} = (3.28246) \times 10^{-5}$$

**Table 8.13.** Musa's failure time data with a generated covariate

Fault	Time	z	Fault	Time	z
1	3	0	35	227	0
2	30	0	36	65	0
3	113	0	37	176	0
4	81	0	38	58	0
5	115	0	39	457	0
6	9	1	40	300	0
7	2	1	41	97	0
8	91	0	42	263	0
9	112	0	43	452	0
10	15	1	44	255	0
11	138	0	45	197	0
12	50	0	46	193	0
13	77	0	47	6	1
14	24	0	48	79	0
15	108	0	49	816	0
16	88	0	50	1351	0
17	670	0	51	148	1
18	120	0	52	21	1
19	26	1	53	233	0
20	114	0	54	134	0
21	325	0	55	357	0
22	55	0	56	193	0
23	242	0	57	236	0
24	68	0	58	31	1
25	422	0	59	369	0
26	180	0	60	748	0
27	10	1	61	0	1
28	1146	0	62	232	0
29	600	0	63	330	0
30	15	1	64	365	0
31	36	1	65	1222	0
32	4	1	66	543	0
33	0	1	67	10	1
34	8	0	68	16	1

**Table 8.13.** (continued)

Fault	Time	z	Fault	Time	z
69	529	0	103	108	0
70	379	0	104	0	1
71	44	1	105	3110	0
72	129	0	106	1247	0
73	810	0	107	943	0
74	290	0	108	700	0
75	300	0	109	875	0
76	529	0	110	245	0
77	281	0	111	729	0
78	160	0	112	1897	0
79	828	0	113	447	0
80	1011	0	114	386	0
81	445	0	115	446	0
82	296	0	116	122	0
83	1755	0	117	990	0
84	1064	0	118	948	0
85	1783	0	119	1082	0
86	860	0	120	22	1
87	983	0	121	75	1
88	707	0	122	482	0
89	33	1	123	5509	0
90	868	0	124	100	1
91	724	0	125	10	1
92	2323	0	126	1071	0
93	2930	0	127	371	0
94	1461	0	128	790	0
95	843	0	129	6150	0
96	12	1	130	3321	0
97	261	0	131	1045	1
98	1800	0	132	648	1
99	865	0	133	5485	0
100	1435	0	134	1160	0
101	30	1	135	1864	0
102	143	1	136	4116	0

Therefore,

$$E_i = e^{\beta_i z_i} = \begin{cases} 5.853905 & \text{for } z = 1 \\ 1 & \text{for } z = 0 \end{cases}$$

The current reliability of the software system is given by

$$R(t_{137}) = e^{-\hat{\phi} E_{137} [\hat{N} - (137-1)] t_{137}}$$

$$= \begin{cases} e^{-9.6076 \cdot 10^{-4} t_{137}} & \text{for } z = 1 \\ e^{-1.64123 \cdot 10^{-4} t_{137}} & \text{for } z = 0 \end{cases}$$

Assuming that

$$P(Z = 1) = \frac{28}{136} = 0.20588$$

$$P(Z = 0) = \frac{108}{136} = 0.79412$$

The reliability of the software for the next 100 seconds is given by

$$R(t_{137} = 100) = 0.90839 \quad \text{for } z = 1 \text{ with probability} = 0.20588$$

$$= 0.98372 \quad \text{for } z = 0 \text{ with probability} = 0.79412$$

or, equivalently, that

$$R(t_{137} = 100) = 0.95375.$$

Similarly, the reliability of the software for the next 1000 seconds is given by

$$R(t_{137} = 1000) = \begin{cases} 0.3826 & \text{for } z = 1 \text{ with probability} = 0.20588 \\ 0.84864 & \text{for } z = 0 \text{ with probability} = 0.79412 \end{cases}$$

or

$$R(t_{137} = 1000) = 0.74021.$$

In the next two applications, we use Pham-Zhang NHPP model given in equation (6.62) to illustrate the model with environmental factors. The corresponding Pham-Zhang NHPP model baseline intensity function can be expressed as follows:

$$\lambda_0(t) = \frac{1}{(1 + \beta e^{-bt})} \left[ (c + a)(1 - e^{-bt}) - \frac{ab}{b - \alpha} (e^{-\alpha t} - e^{-bt}) \right]$$

$$+ \frac{-\beta b e^{-bt}}{(1 + \beta e^{-bt})^2} \left[ (c + a)(1 - e^{-bt}) - \frac{ab(e^{-\alpha t} - e^{-bt})}{b - \alpha} \right] \quad (8.16)$$

Any NHPP models can easily be integrated into a model with environmental factors using equation (8.5).

**Application 8.2:** The first set of software failure data is collected from testing a program for a monitor and real-time control systems (Tohma 1991) (also see data set #8, Chapter 4). Table 8.14 records the software failures detected during a 111-day testing period. The only environmental factor available for this application is the testing team size. Team size is one of the most useful measures in the software development process since it has a close relationship with the testing effort, testing efficiency and the development management issues.

From the correlation analysis of the 32 environmental factors, team size is the only environmental factor correlated to the program complexity, which is the number one significant factor according to our environmental factor study.

Intuitively, the more complex the software, the larger the development team. Therefore, testing team size is an important factor to be incorporated into the software reliability analysis.

Table 8.14 combines the information of testing team size with the software failure data. It is interesting to note that there are two clusters where increasing number of faults are detected. Checking the testing team size, we find that the testing team size was enlarged for the periods associated with the two clusters where increasing number of failures were encountered (day 11 - day 17 and day 36 - day 42). This indicates that testing team is an important factor we need to consider at least for this data set.

Since the testing team size ranges from 1 to 8, we first categorize the factor of team size into two levels. Let  $z_1$  denote the factor of team size as follows:

$$z_1 = \begin{cases} 0 & \text{team size ranges from 1-4} \\ 1 & \text{team size ranges from 5-8} \end{cases}$$

After carefully examining the failure data, we find that after day 61, the software turns stable and the failures occur with a much slower frequency. Therefore, we use the first 61 data points for testing the goodness-of-fit and estimating the parameters. Then we use the calibrated model to predict the remaining 50 data points and compare the prediction to the 50 data points actually observed (from day 62 to day 111) for examining the predictive power of software reliability models.

From equation (8.16), the intensity function with environmental factor is given by:

$$\lambda(t) = \lambda_0(t) \cdot e^{\beta_1 z_1} \\ = \left\{ \begin{aligned} & \frac{1}{(1 + \beta e^{-bt})} [(c+a)(1 - e^{-bt}) - \frac{ab}{b-\alpha}(e^{-\alpha t} - e^{-bt})] \\ & + \frac{-\beta b e^{-bt}}{(1 + \beta e^{-bt})^2} [(c+a)(1 - e^{-bt}) - \frac{ab}{b-\alpha}(e^{-\alpha t} - e^{-bt})] \end{aligned} \right\} e^{\beta_1 z_1} \quad (8.17)$$

First, the coefficient  $\beta_1$  is estimated using partial likelihood estimate method. The partial likelihood method estimates the coefficients of covariates separately from the parameters in the baseline intensity function. From equation (8.11), the likelihood function of partial likelihood method is given by

$$L(\beta) = \prod_i \left( \frac{\exp(\beta z_i)}{[\sum_{m \in R} \exp(\beta z_m)]^{d_i}} \right) \quad (8.18)$$

where  $d_i$  represented the tie failure times. The estimate of  $\beta_1$  for our example is  $\hat{\beta}_1 = 0.0246$  with  $p$ -value 0.01, which indicates that this factor is statistical significant to consider. We then substitute  $\hat{\beta}_1$  into the failure intensity model in equation (8.17) and estimate the parameters in the baseline function.

**Table 8.14.** Software testing data for application 1 (those marked with \* are interpolated data)

Days	Faults	Cum. faults	Team size	Days	Faults	Cum. faults	Team size
1	5*	5*	4	29	2	254	6
2	5*	10*	4	30	5	259	6
3	5*	15*	4	31	4	263	6
4	5*	20*	4	32	1	264	6
5	6*	26*	4	33	4	268	6
6	8	34	5	34	3	271	6
7	2	36	5	35	6	277	6
8	7	43	5	36	13	293	6
9	4	47	5	37	19	309	8
10	2	49	5	38	15	324	8
11	31	80	5	39	7	331	8
12	4	84	5	40	15	346	8
13	24	108	5	41	21	367	8
14	49	157	5	42	8	375	8
15	14	171	5	43	6	381	8
16	12	183	5	44	20	401	8
17	8	191	5	45	10	411	8
18	9	200	5	46	3	414	8
19	4	204	5	47	3	417	8
20	7	211	5	48	8	425	4
21	6	217	5	49	5	430	4
22	9	226	5	50	1	431	4
23	4	230	5	51	2	433	4
24	4	234	5	52	2	435	4
25	2	236	5	53	2	437	4
26	4	240	5	54	7	444	4
27	3	243	5	55	2	446	4
28	9	252	6	56	0	446	4

**Table 8.14** (continued)

Days	Faults	Cumulative Faults	Team Size	Days	Faults	Cumulative Faults	Team Size
57	2	448	4	85	0	473	2
58	3	451	4	86	0	473	2
59	2	453	4	87	2	475	2
60	7	460	4	88	0	475	2
61	3	463	4	89	0	475	2
62	0	463	4	90	0	475	2
63	1	464	4	91	0	475	2
64	0	464	4	92	0	475	2
65	1	465	4	93	0	475	2
66	0	465	3	94	0	475	2
67	0	465	3	95	0	475	2
68	1	466	3	96	1	476	2
69	1	467	3	97	0	476	2
70	0	467	3	98	0	476	2
71	0	467	3	99	0	476	2
72	1	468	3	100	1	477	2
73	1	469	4	101	0	477	1
74	0	469	4	102	0	477	1
75	0	469	4	103	1	478	1
76	0	469	4	104	0	478	1
77	1	470	4	105	0	478	1
78	2	472	2	106	1	479	1
79	0	472	2	107	0	479	1
80	1	473	2	108	0	479	1
81	0	473	2	109	1	480	1
82	0	473	2	110	0	480	1
83	0	473	2	111	1	481	1
84	0	473	2				

The estimates of parameters in the baseline failure intensity function are as follows:

$$\hat{a} = 40.0, \hat{b} = 0.09, \hat{\beta} = 8.0, \hat{\alpha} = 0.015, \hat{c} = 450.$$

After all the parameters are estimated, the mean value function and the software reliability model can be determined. It can then be used to predict quantitatively the software performance metrics such as software reliability, the number of remaining faults, and the failure intensity rate. Table 8.15 compares the SSE and AIC values for some existing NHPP models and the model with environmental factors. It seems that the environmental factor model in equation (8.17) provides a significantly improved predictive power according to the SSE and AIC criteria. This validates that (1) team size is a significant environmental factor for this data set and (2) incorporating this factor provide a better description of the fault detection process and thus enhances the predictive power of the software reliability model.

Note that since the SSE value of the environmental factor model is significantly smaller, other comparison criteria that compensate the model complexity in terms of the number of parameters such as MSE is not necessary. Also, the model provides other useful information such as the number of initial faults is  $\hat{c} = 450$  and the number of introduced faults is  $\hat{a} = 40$ . Therefore, the number of total faults in the software is about 490. By the end of the software testing, 481 faults were detected, which implies that 9 faults still remain in the software.

Sensitivity analysis of the environmental factors categorization is desired to find out whether the categories of defining the environmental factors has significant influence on the final predictive results. Since in practice the typical testing team consists of two people, we re-define the level of team size and re-examine the prediction of the model. This time, we use three levels for the team size. In other words,  $z'_1$  is defined as follows:

$$z'_1 = \begin{cases} 0 & \text{team size ranges from 1-2} \\ 1 & \text{team size ranges from 3-5} \\ 2 & \text{team size} \geq 6 \end{cases} \quad (8.19)$$

The estimate of  $\beta'_1$  for this example is  $\hat{\beta}'_1 = 0.01129$  with  $p$ -value 0.04469.

Similar, the estimates of parameters in equation (8.17) for the numerical example are as follows:

$$\hat{a} = 40.2, \hat{b} = 0.088, \hat{\beta} = 8.1, \hat{\alpha} = 0.0175, \text{ and } \hat{c} = 451.2.$$

We now use this model to predict the detected errors from day 62 to day 111 and compare it with the existing reliability models without environmental factors and the Environmental Factor Model proposed in this paper. The SSE value for the model with three-category team size is 537.48, which is even lower than the SSE of the model with two-category team size (see Table 8.2). Therefore, we can draw a conclusion that, for this example, reliability models incorporating team size provide significant enhancement in terms of predictive power.

**Application 8.3:** The second set of software failure data is collected from testing a large telecommunications software system, which consists of approximately 7000000 non-commentary source lines (NCSL). This release contained approximately 400000 new or changed NCSL for adding new features and fixing existing faults. Table 8.16 summarizes the staff time spent testing, the number of faults detected, and the additional line of code under test. Therefore, the line of code is considered as the environmental factor for this application. Changed code size is one of the most useful metrics of the software development process from which other metrics can be estimated.



**Table 8.15.** Model comparison for application 2

Model name	MVF ( $m(t)$ )	SSE (prediction)	AIC
G-O model	$m(t) = a(1 - e^{-bt})$ $a(t) = a$ $b(t) = b$	1,052,528	978.14
Delayed S-shaped	$m(t) = a(1 - (1 + bt)e^{-bt})$ $a(t) = a$ $b(t) = \frac{b^2 t}{1 + bt}$	83,929.3	983.90
Inflexion S-shaped	$m(t) = \frac{a(1 - e^{-bt})}{1 + \beta e^{-bt}}$ $a(t) = a$ $b(t) = \frac{b}{1 + \beta e^{-bt}}$	1,051,714.7	980.14
Yamada exponential	$m(t) = a(1 - e^{-r\alpha(1 - e^{-\beta t})})$ $a(t) = a$ $b(t) = r\alpha \beta e^{-\beta t}$	1,085,650.8	979.88
Yamada Rayleigh	$m(t) = a(1 - e^{-r\alpha(1 - e^{-\beta t^2/2})})$ $a(t) = a$ $b(t) = r\alpha \beta te^{-\beta t^2/2}$	86,472.3	967.92
Imperfect debugging (1)	$m(t) = \frac{ab}{\alpha + b}(e^{\alpha t} - e^{-bt})$ $a(t) = ae^{\alpha t}$ $b(t) = b$	791,941	981.44
Imperfect debugging (2)	$m(t) = a[1 - e^{-bt}][1 - \frac{\alpha}{b}] + \alpha a t$ $a(t) = a(1 + \alpha t)$ $b(t) = b$	238,324	984.62

**Table 8.15.** (continued)

PNZ Model	$m(t) = \frac{a}{1 + \beta e^{-bt}} [(1 - e^{-bt})(1 - \frac{\alpha}{b}) + \alpha t]$ $a(t) = a(1 + \alpha t)$ $b(t) = \frac{b}{1 + \beta e^{-bt}}$	94,112.2	965.37
PZ model	$m(t) = \frac{1}{(1 + \beta e^{-bt})} [(c + a)(1 - e^{-bt}) - \frac{ab}{b - \alpha} (e^{-\alpha t} - e^{-bt})]$ $a(t) = c + a(1 - e^{-\alpha t})$ $b(t) = \frac{b}{1 + \beta e^{-bt}}$	86,180.8	960.68
Environmental factor model	$m(t) = \frac{1}{(1 + \beta e^{-bt})} [(c + a)(1 - e^{-bt}) - \frac{ab}{b - \alpha} (e^{-\alpha t} - e^{-bt})] e^{\beta z_1}$ $a(t) = c + a(1 - e^{-\alpha t})$ $b(t) = \frac{b}{1 + \beta e^{-bt}}$	560.82	890.68

From Table 8.16, we can see that increases in faults are associated with increases in code size. This indicates that change of code size is an important factor to be considered. From Table 8.17 we can see that the environmental factor model seems to provide the best predictive power according to the SSE and AIC values. Other measures such as the testing effort and development cost are usually estimated based on code size. Therefore, code size is an important factor to be incorporated into the software reliability analysis.

Let  $z_c$  denote the factor of changed code size as follows:

$$z_c = \begin{cases} 0 & \text{changed code} \leq 1,000 \text{ NLOC} \\ 1 & 1,000 \text{ NLOC} < \text{changed code} \leq 5,000 \text{ NLOC} \\ 2 & 5,000 \text{ NLOC} < \text{changed code} \leq 10,000 \text{ NLOC} \\ 3 & 10,000 \text{ NLOC} < \text{changed code} \end{cases} \quad (8.20)$$

After carefully examining the failure data, we find that the failures occur with a much slower frequency after 1013.9 staff days of testing. Therefore, we use data

up to the 1013.9 staff-days to fit the models and estimate the parameters, and use calibrated model to predict the remaining data and compare the predictive power of software reliability models.

Similar to analysis of Application 2, the estimate of  $\beta_1$  for our example is  $\hat{\beta}_1 = 0.00567$  with  $p$ -value 0.048, which indicates that this factor is significant to consider. The estimates of parameters in the baseline failure intensity function in equation (17) are as follows:

$$\hat{a} = 101.0, \hat{b} = 0.004, \hat{\beta} = 8.9, \hat{\alpha} = 0.0148, \text{ and } \hat{c} = 803.5.$$

Table 8.17 lists the SSE and AIC values for the model comparison. From the results it is seen that the number of initial faults is  $\hat{c} = 804$  and the number of introduced faults is  $\hat{a} = 102$ . Therefore, the number of total faults in the software is about 906. By the end of the software testing, 870 faults were detected, which implies that the number of residual faults is about 36.

**Table 8.16.** Software testing data for application 3

Staff days	Faults	Code size	$z$	Staff days	Faults	Code size	$z$	Staff days	Faults	Code size	$z$
0	0	0	0	207.2	97	213093	2	424.9	321	272457	1
4.8	0	16012	3	211.9	98	219248	2	434.2	326	273741	1
6	0	16012	3	217	105	221355	1	442.7	339	275025	1
14.3	7	32027	3	223.5	113	223462	1	451.4	346	276556	1
22.8	7	48042	3	227	113	225568	1	456.1	347	278087	1
32.1	7	58854	3	234.1	122	227675	1	460.8	351	279618	1
41.4	7	69669	3	241.6	129	229784	1	466	356	281149	1
51.2	11	80483	3	250.7	141	233557	1	472.3	359	283592	1
60.6	12	91295	3	259.8	155	237330	1	476.4	362	286036	1
70	13	102110	3	268.3	166	241103	1	480.9	367	288480	1
79.9	15	112925	3	277.2	178	244879	1	486.8	374	290923	1
91.3	20	120367	2	285.5	186	247946	1	495.8	376	293367	1
97	21	127812	2	294.2	190	251016	1	505.7	380	295811	1
107.7	22	135257	2	298	190	254086	1	516	392	298254	1
119.1	28	142702	2	305.2	195	257155	1	526.2	399	300698	1
127.6	40	150147	2	312.3	201	260225	1	527.3	401	300698	1
135.1	44	152806	1	318.2	209	260705	0	535.8	405	303142	1
142.8	46	155464	1	328.9	224	261188	0	546.3	415	304063	0
148.9	48	158123	1	334.8	231	261669	0	556.1	425	305009	0
156.6	52	160781	1	342.7	243	262889	0	568.1	440	305956	0
163.9	52	167704	2	350.5	252	263629	0	577.2	457	306902	0
169.7	59	174626	2	356.3	259	264367	0	578.3	457	306902	0
170.1	59	174626	2	360.6	271	265107	0	587.2	467	307849	0
174.7	63	181548	2	365.7	277	265845	0	595.5	473	308795	0
179.6	68	188473	2	386.5	290	267325	1	605.6	480	309742	0
185.5	71	194626	2	396.5	300	268607	1	613.9	491	310688	0
194	88	200782	2	408	310	269891	1	621.6	496	311635	0
200.3	93	206937	2	417.3	312	271175	1	623.4	496	311635	0

**Table 8.16.** (continued)

Staff days	Faults	Code size	$z$	Staff days	Faults	Code size	$z$	Staff days	Faults	Code size	$z$
636.3	502	311750	0	938.3	710	330435	0	1231.6	842	333481	0
649.7	517	311866	0	952	720	330263	0	1240.9	844	333695	0
663.9	527	312467	0	965	729	330091	0	1249.5	845	333909	0
675.1	540	313069	0	967.7	729	330091	0	1262.2	849	335920	1
677.4	543	313069	0	968.6	731	330091	0	1271.3	851	337932	1
677.9	544	313069	0	981.3	740	329919	0	1279.8	854	339943	1
688.4	553	313671	0	997	749	329747	0	1281	854	339943	1
698.1	561	314273	0	1013.9	759	330036	0	1287.4	855	341955	1
710.5	573	314783	0	1030.1	776	330326	0	1295.1	859	341967	0
720.9	581	315294	0	1044	781	330616	0	1304.8	860	341979	0
731.6	584	315805	0	1047	782	330616	0	1305.8	865	342073	0
732.7	585	315805	0	1059.7	783	330906	0	1313.3	867	342168	0
733.6	585	315805	0	1072.6	787	331196	0	1314.4	867	342168	0
746.7	586	316316	0	1085.7	793	331486	0	1320	867	342262	0
761	598	316827	0	1098.4	796	331577	0	1325.3	867	342357	0
776.5	612	318476	1	1112.4	797	331669	0	1330.6	870	342357	0
793.5	621	320125	1	1113.5	798	331669	0	1334.2	870	342358	0
807.2	636	321774	1	1141.1	798	331669	0	1336.7	870	342358	0
811.8	639	321774	1	1128	802	331760	0				
812.5	639	321774	1	1139.1	805	331852	0				
829	648	323423	1	1151.4	811	331944	0				
844.4	658	325072	1	1163.2	823	332167	0				
860.5	666	326179	1	1174.3	827	332391	0				
876.7	674	327286	1	1184.6	832	332615	0				
892	679	328393	1	1198.3	834	332939	0				
895.5	686	328393	1	1210.3	836	333053	0				
910.8	690	329500	1	1221.1	839	333267	0				
925.1	701	330608	1	1230.5	842	333481	0				

**Table 8.17.** Model comparison

Model name	SSE	AIC
G-O model	240,773	1473.5
Delayed S-shaped	4,322	1422.6
Inflexion S-shaped	246,702	1481.5
Yamada exponential	230,955	1471.1
Yamada Rayleigh	8,824	1441.3
Imperfect debugging (1)	290,449	1491.7
Imperfect debugging (2)	364,398	1496.1
PNZ model	17,753	1441.7
PZ model	8,947	1436.5
Environmental factor model	1,182	1411.0

## 8.9 Further Reading

Some interesting research papers and book on this subject are, but not limited to:

Zhang X. and Pham, H., “An analysis of factors affecting software reliability,” *Journal of Systems and Software*, 1999

Venkatesh, G. A. and Fischer, C. N., “SPARE: A development environment for program analysis algorithms,” *IEEE Trans on Software Engineering*, vol 18, no. 4, April 1992

Madhavji, N.H., “Environment Evolution: The Prism model of changes,” *IEEE Trans on Software Engineering*, vol 18, no. 5, May 1992

## 8.10 Problems

1. Using the real-time control system as in Table 4.12 (data set #8, Chapter 4), calculate the MLE for unknown parameters of the EPJM model discussed in Section 8.7.
2. Based on the first 60 days in Table 4.12 (data set #8, Chapter 4), calculate the MLE for unknown parameters of the EPJM model.
3. Let us define a failure-cluster factor, such as

$$z_i = \begin{cases} 1 & \text{when } \frac{t_{i-1}}{t_i} \geq 10 \text{ or } \frac{t_{i-2}}{t_i} \geq 12 \\ 0 & \text{otherwise} \end{cases}$$

Using the software failure data set #9 in Chapter 4, obtain the entire data set with the environmental factor variable  $z_i$ . Then estimate the two parameters,  $N$  and  $\phi$ , of EPJM model.

## Calibrating Software Reliability Models

---

### 9.1 Introduction

Estimating software reliability measures that will be perceived by users is important in order to decide when to release software. Usually, software reliability models are applied to system test data with the hope of estimating the failure rate of the software in user environments. This chapter discusses recent methods and research on how to quantify the mismatch between the system test environment and the field environment based on recent studies (Zhang 2002; Teng 2001). The chapter also discusses a generalized random field environment (RFE) model incorporating both testing phase and operating phase in the software development cycle for estimating the reliability of software systems in the field. Examples are included to illustrate the calibrating software reliability model based on test data.

#### Notation

$a(t)$	Fault content function, <i>i.e.</i> , total number of faults in the software including the initial and introduced faults
$b(t)$	Fault detection rate function (faults per unit of time)
$b_{test}$	Average per fault failure rate during system test interval
$b_{field}$	Average per fault failure rate in the field
$\bar{b}_{test}$	Long-term average per fault failure rate during system test interval
$\bar{b}_{field}$	Long term average per fault failure rate in the field
$\lambda(t)$	Failure intensity function (faults per unit of time)
$\hat{\lambda}(T)$	Failure intensity representation based on system test data
$K$	Calibration factor
$m(t)$	Mean value function, <i>i.e.</i> , the expected number of faults detected by time $t$
$N(t)$	Number of detected faults by time $t$
$\bar{N}(t)$	Number of residual faults by time $t$
$T$	Duration of system test interval

## 9.2 Calibration Factor Approach

Let us assume that the system test ends at time  $T$  and after that the software is delivered to the field. The expected number of faults detected and removed by time  $T$  is  $m(T)$ . To account for the mismatch between the system test field environments, Zhang *et al.* (2002) recently proposed linking the error detection rate function  $b(t)$  under the system test environment, say  $b_{\text{test}}(t)$ , to a different  $b(t)$  under the field environment, say  $b_{\text{field}}(t)$ . Define

$$\bar{b}_{\text{test}} = \lim_{T \rightarrow \infty} \frac{1}{T} \int_0^T b_{\text{test}}(t) dt \quad (9.1)$$

Intuitively,  $\bar{b}_{\text{test}}$  represents the long-term average per fault failure rate during system test. Using an analogous definition for  $\bar{b}_{\text{field}}$ , Zhang *et al.* (2002) defined the calibration factor as the ratio  $K = \bar{b}_{\text{test}} / \bar{b}_{\text{field}}$ . In the case where the system test and field environments are the same,  $K$  will be unity.

Assuming that the fault detection rate,  $b_{\text{test}}(t)$ , for system test environments is given by

$$b_{\text{test}}(t) = \frac{b_{\text{test}}}{1 + \beta e^{-b_{\text{test}}t}} \quad (9.2)$$

where  $\beta$  represents a learning parameter and  $b_{\text{test}}$  is the limiting value of the fault detection rate. Note that  $\beta = 0$  coincides with no learning, and the fault detection rate reduces to the constant value  $b_{\text{test}}$ .

### A General Approach (Zhang 2002)

Consider a context where only system test data is available for a release and it is desired to estimate the field failure rate of the software. Assume that system test and field data of the previous releases of the same product or similar product are also available from which a  $K$  factor can be obtained. We suppose an NHPP SRGM model (see Chapter 6) has been fit to the system test data and the assumptions underlying GO model (*i.e.*, no learning factor and no introduction of new faults) are adequately satisfied in the field environment. The software failure rate in the field can be estimated by the following steps:

1. Estimate the calibration factor  $K$  from previous releases/projects.
2. Estimate the number of residual faults based on system test data,  $\hat{N}(T) = \hat{a}(T) - N(T)$ , and the long-term average per fault failure rate  $\bar{b}_{\text{test}} = \hat{b}_{\text{test}}$  from the system test data.
3. Calibrate the system test analysis to estimate the average per fault failure rate in the field using the calibration factor  $K$ . The average per fault failure rate in the field is estimated by  $\hat{b}_{\text{field}} = \hat{b}_{\text{test}} / K$ .

4. Estimate the failure rate of the software in the field by incorporating the number of residual faults by the average per fault failure rate in the field. The field failure rate after  $t$  system-hours of field exposure time is

$$\hat{\lambda}_{field}(t) = \hat{N}(T) \times \hat{b}_{field} \times e^{-\hat{b}_{field}t} = \hat{N}(T) \times \frac{\hat{b}_{test}}{K} \times e^{-\frac{\hat{b}_{test}}{K}t}. \quad (9.3)$$

### 9.3 Model Application

Consider two systems test data, shown in Table 9.1 and 9.2 where System 1 is a high-capacity data transmission system and System 2 is a flexible signal multiplexing and transmission system (Zhang 2002). The interesting question here is, by comparing the analyses of the system test and field data, how can one calculate the value of the calibration factor  $K$  for each project. In other words, it is interesting to show, in general, how projects can calibrate their system test data analyses to make them applicable to field environments.

Assume the mean value function (Pham 1997a) is given by

$$m(t) = \frac{1}{(1 + \beta e^{-b_{test}t})} [(c + a)(1 - e^{-b_{test}t}) - \frac{ab_{test}}{b_{test} - \alpha} (e^{-\alpha t} - e^{-b_{test}t})] \quad (9.4)$$

The parameter  $\alpha$  represents the fault introduction rate and  $\beta$  is the learning parameter. Table 9.3 lists the estimates of these parameters for both System 1 and System 2. See Zhang (2002) for details. Note that the estimates of  $c$  and  $a$  are shown as normalized values, defined as the ratio between the parameter estimates and the actual number of total faults.

#### Calculation of the Calibration Factor

The GO model was fit to the field data of both System 1 and System 2. Table 9.4 shows the estimates of the average per fault failure in the field,  $b_{field}$ , and the long-term average per fault failure rate in system test,  $\bar{b}_{test} = b_{test}$ , for each system. The analyses of the two data sets presented in this section provide a means to estimate the calibration factor  $K$ . According to Section 2, the ratio of  $b_{test}$  to  $b_{field}$  gives an empirical observation of  $K$ . The value of  $K$  for System 1 and System 2 are 34.32 and 49.20 respectively.



**Table 9.1.** System 1 normalized data

Time	Cumulative faults	Estimated faults	Time	Cumulative faults	Estimated faults
0	0	0	0.508197	0.863014	0.859464
0.016393	0.020548	0.019882	0.52459	0.881849	0.873735
0.032787	0.054795	0.041306	0.540984	0.900685	0.886763
0.04918	0.07363	0.064295	0.557377	0.913527	0.898626
0.065574	0.104452	0.088856	0.57377	0.922089	0.909404
0.081967	0.129281	0.114968	0.590164	0.928938	0.919174
0.098361	0.15839	0.142591	0.606557	0.934932	0.928016
0.114754	0.184075	0.171656	0.622951	0.941781	0.936002
0.131148	0.217466	0.202065	0.639344	0.943493	0.943205
0.147541	0.236301	0.233693	0.655738	0.951199	0.949693
0.163934	0.26113	0.266387	0.672131	0.961473	0.955529
0.180328	0.302226	0.299971	0.688525	0.97089	0.960772
0.196721	0.345034	0.334243	0.704918	0.973459	0.965479
0.213115	0.374144	0.368986	0.721311	0.976884	0.969699
0.229508	0.385274	0.40397	0.737705	0.980308	0.973481
0.245902	0.40411	0.438958	0.754098	0.983733	0.976867
0.262295	0.440925	0.473712	0.770492	0.983733	0.979897
0.278689	0.482877	0.508004	0.786885	0.983733	0.982607
0.295082	0.53339	0.541613	0.803279	0.987158	0.985029
0.311475	0.594178	0.57434	0.819672	0.988014	0.987192
0.327869	0.614726	0.606006	0.836066	0.98887	0.989125
0.344262	0.651541	0.636458	0.852459	0.98887	0.99085
0.360656	0.672089	0.665569	0.868852	0.990582	0.992389
0.377049	0.684075	0.693241	0.885246	0.994007	0.993763
0.393443	0.711473	0.719405	0.901639	0.994007	0.994987
0.409836	0.732021	0.744017	0.918033	0.994863	0.99608
0.42623	0.75	0.767059	0.934426	0.995719	0.997054
0.442623	0.77226	0.788535	0.95082	0.995719	0.997922
0.459016	0.789384	0.808468	0.967213	0.997432	0.998696
0.47541	0.809932	0.826897	0.983607	0.998288	0.999385
0.491803	0.84161	0.843875	1	1	1

## 9.4 Calibrating Models with Random Field Environments

Many existing NHPP software reliability models have been carried out through the fault intensity rate function and the mean value functions  $m(t)$  within a controlled operating environment. Generally, these models are applied to the software testing data and then used to make predictions on the software failures and reliability in the field. The operating environments in the field for the software are perhaps quite different. The randomness of the field environment will affect the software failure and software reliability in an unpredictable way.

**Table 9.2.** System 2 normalized data

Time	Cumulative faults	Estimated faults	Time	Cumulative faults	Estimated faults
0	0	0	0.512821	0.639547	0.688351
0.025641	0.019567	0.023097	0.538462	0.677652	0.719633
0.051282	0.052523	0.0481	0.564103	0.720906	0.74906
0.076923	0.07415	0.07503	0.589744	0.742533	0.776578
0.102564	0.102987	0.103878	0.615385	0.764161	0.802168
0.128205	0.125644	0.134602	0.641026	0.786818	0.825843
0.153846	0.161689	0.167122	0.666667	0.797116	0.847641
0.179487	0.189495	0.201317	0.692308	0.814624	0.867624
0.205128	0.22863	0.237027	0.717949	0.830072	0.885868
0.230769	0.255407	0.274051	0.74359	0.840371	0.902465
0.25641	0.294542	0.312152	0.769231	0.851699	0.917511
0.282051	0.329557	0.351062	0.794872	0.863028	0.931111
0.307692	0.369722	0.390487	0.820513	0.865088	0.943369
0.333333	0.399588	0.430118	0.846154	0.888774	0.954391
0.358974	0.427394	0.469639	0.871795	0.898043	0.96428
0.384615	0.46344	0.508738	0.897436	0.935118	0.973134
0.410256	0.495366	0.547116	0.923077	0.950566	0.981047
0.435897	0.526262	0.584495	0.948718	0.966014	0.988108
0.461538	0.543769	0.620629	0.974359	0.971164	0.9944
0.487179	0.593203	0.655305	1	1	1

**Table 9.3.** Parameter estimation for Systems 1 and 2

Parameter	System 1	System 2
(Normalized) initial faults $\hat{c}$	0.987	0.912
(Normalized) introduced faults $\hat{a}$	0.015	0.093
Average per fault failure rate $\hat{b}_{test}$	0.11667	0.12650
Fault introduction parameter $\hat{\alpha}$	$9.4 \times 10^{-5}$	$9.82 \times 10^{-6}$
Learning parameter $\hat{\beta}$	5.13	4.956

**Table 9.4.** The calibration factor for Systems 1 and 2

Parameter	System 1		System 2	
	System test	Field	System test	Field
Average per fault failure rate	0.1167	0.0034	0.1265	0.00257
Calibration factor, $K$	34.32		49.20	

In previous sections, we discuss an NHPP software reliability calibration model by considering a calibration factor. This section discusses a model based on

NHPP model framework for predicting software failures and evaluating the software reliability subject to the random field environments.

### Notation

$R(t)$	Software reliability function
$\eta$	Random environmental factor
$G(\eta)$	Cumulative distribution function of $\eta$
$\gamma$	Shape parameter of Gamma distributions
$\theta$	Scale parameter of Gamma distributions
$\alpha, \beta$	Parameters of Beta distributions
$N(t)$	Counting process which counts the number of software failures discovered by time $t$
$m(t)$	Expected number of software failures detected by time $t$
$a(t)$	Expected number of initial software faults plus introduced faults by time $t$
$m_1(t)$	Expected number of software failures in testing by time $t$
$m_2(t)$	Expected number of software failures in the field by time $t$
$a_1(t)$	Expected number of initial software faults plus introduced faults discovered in the testing by time $t$
$a$	Number of initial software faults at the beginning of testing phase, is a software parameter that is directly related to the software itself
$T$	Time to stop testing and release the software for field operations
$a_F$	number of initial software faults in the field (at time $T$ )
$b(t)$	Failure detection rate per fault at time $t$ , is a process parameter that is directly related to testing and failure process
$p$	Probability that a fault will be successfully removed from the software
$q$	Error introduction rate at time $t$ in the testing phase
RFE-model	Software reliability model subject to a random field environment
$\gamma$ -RFE	Software reliability model with a Gamma distributed field environment
$\beta$ -RFE	Software reliability model with a Beta distributed field environment

#### 9.4.1 A Generalized Random Field Environmental Model

A generalized NHPP model can be formulated as follows (Zhang 2003):

$$m'(t) = \eta \cdot b(t) \cdot (a(t) - p \cdot m(t)) \quad (9.5)$$

$$a'(t) = q \cdot m'(t) \quad (9.6)$$

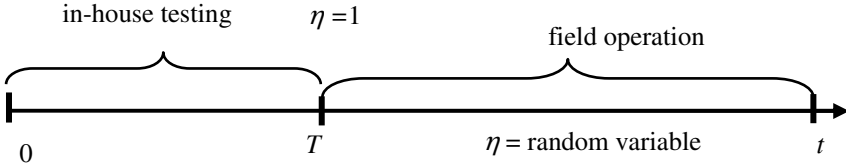
where  $m(t)$  is the expected number of software failures to be detected by time  $t$ . If the marginal conditions are given as  $m(0) = 0$  and  $a(0) = a$ , then for a specific

environmental factor  $\eta$ , the solutions to equations (9.5) and (9.6) can be obtained as follows:

$$m_{\eta}(t) = a \int_0^t \eta b(u) e^{-\int_0^u \eta (p-q) b(\tau) d\tau} du \quad (9.7)$$

$$a_{\eta}(t) = a \left( 1 + \int_0^t \eta q b(u) e^{-\int_0^u \eta (p-q) b(\tau) d\tau} du \right) \quad (9.8)$$

Figure 9.1 shows the last two phases of the software life cycle: in-house testing and field operation. If  $T$  is the time to stop testing and release the software for field operations, then the time period  $0 \leq t \leq T$  refers to the time period of *Software Testing*, while the time period  $T \leq t$  refers to the post release period – *Field Operation*.



**Figure 9.1.** Testing vs field environment where  $T$  is the time to stop testing

The environmental factor  $\eta$  is used to capture the uncertainty about the environment and its effects upon the software failure rate. In general, the software testing is carried out in a controlled environment with very small variations, which can be used as a referenced environment where  $\eta$  is constant and equal to 1. For the field operating environment, the environmental factor  $\eta$  is assumed to be a nonnegative random variable with probability density function (pdf)  $f(\eta)$ , i.e.

$$\eta = \begin{cases} 1 & t \leq T \\ \text{r.v. with pdf } f(\eta) & t \geq T \end{cases} \quad (9.9)$$

If the value of  $\eta$  is less than 1, it indicates that the condition is less favorable to fault detection than that of testing environment. Likewise, if the value of  $\eta$  is greater than 1, it indicates that the condition is more favorable to fault detection than that of testing environment.

From equations (9.7) and (9.9), the mean value function and the function  $a_1(t)$  during testing can be obtained as

$$m_{1}(t) = a \int_0^t b(u) e^{-\int_0^u (p-q) b(\tau) d\tau} du \quad t \leq T \quad (9.10)$$

$$a_1(t) = a \left( 1 + \int_0^t q b(u) e^{-\int_0^u (p-q) b(\tau) d\tau} du \right) \quad t \leq T$$

For the field operation, where  $t \geq T$ , the mean value function can be represented as

$$\begin{aligned}
m_2(t) &= m_1(T) + \int_0^\infty m_\eta(t) f(\eta) d\eta \\
&= m_1(T) + \int_0^\infty \left( a_F \int_T^t \eta b(u) e^{-\int_T^u \eta(p-q) b(\tau) d\tau} du \right) f(\eta) d\eta \quad (9.11) \\
&= m_1(T) + \int_T^t a_F b(u) \left( \int_0^\infty \eta e^{-\int_T^u \eta(p-q) b(\tau) d\tau} f(\eta) d\eta \right) du
\end{aligned}$$

where  $a_F$  is number of faults in the software at time  $T$ . Using the Laplace transform formula, the mean value function can be rewritten as

$$\begin{aligned}
m_2(t) &= m_1(T) + \int_T^t a_F b(u) \left( -\frac{dF^*(s)}{ds} \Big|_{s=\int_0^u (p-q) b(\tau) d\tau} \right) du \\
&= m_1(T) + \frac{a_F}{(p-q)} \int_T^t \left( -dF^* \left( (p-q) \int_T^u b(\tau) d\tau \right) \right)
\end{aligned}$$

where  $F^*(s)$  is the Laplace transform of the pdf  $f(x)$  and

$$\int_0^\infty x \cdot e^{-x \cdot s} \cdot f(x) dx = -\frac{dF^*(s)}{ds}$$

or, equivalently,

$$\begin{aligned}
m_2(t) &= m_1(T) - \frac{a_F}{p-q} F^* \left( (p-q) \int_T^t b(\tau) d\tau \right) \Big|_T^t \\
&= m_1(T) + \frac{a_F}{p-q} \left( F^*(0) - F^* \left( (p-q) \int_T^t b(\tau) d\tau \right) \right), \quad t \geq T
\end{aligned}$$

Notice that  $F^*(0) = \int_0^\infty e^{-0x} f(x) dx = 1$ , then

$$m_2(t) = m_1(T) + \frac{a_F}{p-q} \left( 1 - F^* \left( (p-q) \int_T^t b(\tau) d\tau \right) \right) \quad t \geq T$$

The expected number of faults in the software at time  $T$  is given by

$$\begin{aligned}
a_F &= a_1(T) - p m_1(T) \\
&= a \left( 1 - \int_0^t (p-q) b(u) e^{-\int_0^u (p-q) b(\tau) d\tau} du \right) \\
&= a e^{-\int_0^t (p-q) b(\tau) d\tau}
\end{aligned}$$

The generalized RFE model can be obtained as

$$m(t) = \begin{cases} \frac{a}{p-q} \left( 1 - e^{-(p-q) \int_0^t b(\tau) d\tau} \right) & t \leq T \\ \frac{a}{p-q} \left( 1 - e^{-(p-q) \int_0^T b(\tau) d\tau} F^* \left( (p-q) \int_T^t b(\tau) d\tau \right) \right) & t \geq T \end{cases} \quad (9.12)$$

This model in equation (9.12) is a generalized software reliability model subject to random field environments. The next section presents specific RFE models for the Gamma and beta distributions of the random field environmental factor  $\eta$ .

#### 9.4.2 RFE Reliability Models

This section discusses two specific models. The first model is a  $\gamma$ -RFE model based on Gamma distribution which can be used to evaluate and predict software reliability in the field environments where software failure detection rate can be either greater or less than the failure detection rate in the testing environment. The second model is a  $\beta$ -RFE model based on Beta distribution which can be used to predict software reliability in the field environments where the software failure detection rate can only be less than the failure detection rate in the testing environment.

##### Gamma Model

In this model, we use the Gamma distribution to describe the random environmental factor  $\eta$ . This model is called  $\gamma$ -RFE model.

Assume  $\eta$  follows a Gamma distribution with a probability density function as follows:

$$f_{\gamma}(\eta) = \frac{\theta^{\gamma} \cdot \eta^{\gamma-1} \cdot e^{-\theta \cdot \eta}}{\Gamma(\gamma)}, \quad \gamma, \theta > 0; \quad \eta \geq 0. \quad (9.13)$$

Figure 9.2 shows an example of the Gamma density probability function. The Gamma function seems to be a reasonable to describe a software failure process in those field environments where the software failure detection rate can be either greater (*i.e.*,  $\eta > 1$ ) or less than (*i.e.*,  $\eta < 1$ ) the failure detection rate in the testing environment.

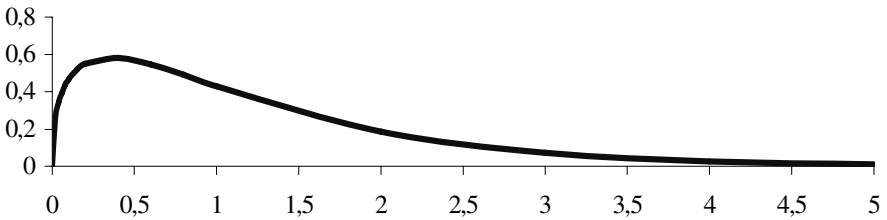


Figure 9.2. A Gamma density function

The Laplace transform of the probability density function in equation (9.13) is

$$F^*(s) = \left[ \frac{\theta}{\theta + s} \right]^\gamma \quad (9.14)$$

Assume that the error detection rate function  $b(t)$  is given by

$$b(t) = \frac{b}{1 + c \cdot e^{-bt}} \quad (9.15)$$

where  $b$  is the asymptotic unit software failure detection rate and  $c$  is the parameter defining the shape of the learn curve, then from equation (9.12) the mean value function of  $\gamma$ -RFE model can be obtained as follows:

$$m_\gamma(t) = \begin{cases} \frac{a}{(p-q)} \left( 1 - \left( \frac{1+c}{e^{bt} + c} \right)^{p-q} \right) & t \leq T \\ \frac{a}{p-q} \left( 1 - \left( \frac{1+c}{e^{bT} + c} \right)^{p-q} \left( \frac{\theta}{\theta + (p-q) \ln \left( \frac{c + e^{bt}}{c + e^{bT}} \right)} \right)^\gamma \right) & t \geq T \end{cases} \quad (9.16)$$

### Beta Model

Similarly, we use the Beta distribution to describe the random environmental factor  $\eta$ , which is called  $\beta$ -RFE model. The Beta pdf is

$$f_\beta(\eta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} \eta^{\alpha-1} (1-\eta)^{\beta-1} \quad 0 \leq \eta \leq 1, \alpha > 0, \beta > 0 \quad (9.17)$$

Figure 9.3 shows an example of the Beta density function. It seems that the  $\beta$ -RFE model is a reasonable function to describe a software failure process in those field environments where the software failure detection rate can only be less than the failure detection rate in the testing environment. This is not uncommon in the software industry because, during the software testing, the engineers generally test the software intensely and conduct an “accelerated” test on the software in order to detect most of the software faults as early as possible.

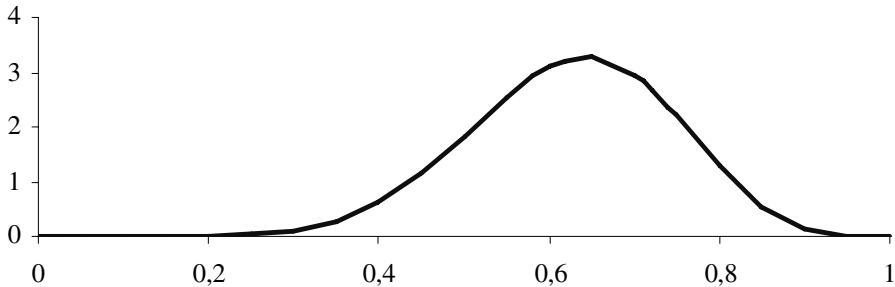


Figure 9.3. A pdf curve of Beta distribution

The Laplace transform of the pdf in equation (9.17) is

$$F_{\beta}^*(s) = e^{-s} \cdot HG([\beta], [\alpha + \beta], s) \quad (9.18)$$

where  $HG([\beta], [\alpha + \beta], s)$  is a generalized hypergeometric function such that

$$HG([a_1, a_2, \dots, a_m], [b_1, b_2, \dots, b_n], s) = \sum_{k=0}^{\infty} \left[ \frac{s^k \prod_{i=1}^m \frac{\Gamma(a_i + k)}{\Gamma(a_i)}}{\prod_{i=1}^n \frac{\Gamma(b_i + k)}{\Gamma(b_i)} k!} \right]$$

Therefore

$$\begin{aligned} F_{\beta}^*(s) &= e^{-s} \sum_{k=0}^{\infty} \left[ \frac{\Gamma(\alpha + \beta) \Gamma(\beta + k) s^k}{\Gamma(\beta) \Gamma(\alpha + \beta + k) k!} \right] \\ &= \sum_{k=0}^{\infty} \left[ \frac{\Gamma(\alpha + \beta) \Gamma(\beta + k)}{\Gamma(\beta) \Gamma(\alpha + \beta + k)} \cdot \frac{s^k e^{-s}}{k!} \right] \\ &= \sum_{k=0}^{\infty} \left[ \frac{\Gamma(\alpha + \beta) \Gamma(\beta + k)}{\Gamma(\beta) \Gamma(\alpha + \beta + k)} \cdot P(k, s) \right] \end{aligned}$$

where  $P(k, s)$  is a Poisson probability density function as follows:

$$P(k, s) = \frac{s^k e^{-s}}{k!}$$

Using the same error detection rate function as in equation (9.15) and replacing  $F^*(s)$  by  $F_{\beta}^*(s)$ , the mean value function of the  $\beta$ -RFE model is

$$m_{\beta}(t) = \begin{cases} \frac{a}{(p-q)} \left( 1 - \left( \frac{1+c}{e^{bt} + c} \right)^{p-q} \right) & t \leq T \\ \frac{a}{p-q} \left( 1 - \left( \frac{1+c}{e^{bT} + c} \right)^{p-q} \sum_{k=0}^{\infty} \left[ \frac{\Gamma(\alpha + \beta) \Gamma(\beta + k) P(k, s)}{\Gamma(\beta) \Gamma(\alpha + \beta + k)} \right] \right) & t \geq T \end{cases} \quad (9.19)$$

where

$$s = (p-q) \left( \ln \left( \frac{c + e^{bt}}{c + e^{bT}} \right) \right)$$

### 9.4.3 Applications

This section discusses the parameter estimation and illustrates the applications of the two random field environment software reliability models using a software failure data. In this analysis, the error removal efficiency  $p$  is given. Each model has five unknown parameters. For example, in the  $\gamma$ -RFE model we need to estimate the following five unknown parameters:  $a$ ,  $b$ ,  $q$ ,  $\gamma$  and  $\theta$ . For the  $\beta$ -RFE model, we will estimate  $a$ ,  $b$ ,  $q$ ,  $\alpha$  and  $\beta$ .

Table 9.5 shows a set of failure data from a telecommunication software application during software testing. The column “Time” shows the normalized



cumulative time spent in software testing for this telecommunication application, and the column “Failures” shows the normalized cumulative number of failures occurred in the testing period up to the given time.

**Table 9.5.** Normalized cumulative failures and times during software testing

Time	Failures	Time	Failures	Time	Failures
0.0001	0.0249	0.0038	0.3483	0.0121	0.6766
0.0002	0.0299	0.0044	0.3532	0.0128	0.7015
0.0002	0.0647	0.0048	0.3682	0.0135	0.7363
0.0003	0.0647	0.0053	0.3881	0.0142	0.7761
0.0005	0.1095	0.0058	0.4478	0.0147	0.7761
0.0006	0.1194	0.0064	0.4876	0.0155	0.8159
0.0008	0.1443	0.0070	0.5224	0.0164	0.8259
0.0012	0.1692	0.0077	0.5473	0.0172	0.8408
0.0016	0.1990	0.0086	0.5821	0.0176	0.8458
0.0023	0.2289	0.0095	0.6119	0.0180	0.8756
0.0028	0.2637	0.0105	0.6368	0.0184	0.8955
0.0033	0.3134	0.0114	0.6468	0.0184	0.9005

The time to stop testing is at  $T = 0.0184$ . After the time  $T$ , the software is released for field operations. Table 9.6 shows the field data for this software release. Similarly, the column “Time” shows the normalized cumulative time spent in the field for this software application, and the time in Table 9.6 is continued from the time to stop testing  $T$ . The column “Failures” shows the normalized cumulative number of failures found after releasing the software for field operations up to the given time. The cumulative number of failures is the total number of software failures since the beginning of software testing.

Let us assume that testing engineers have number of years experience in this particular product and software development skills and therefore conducted a perfect debugging during the test. In other words,  $p = 1$ . We also assume that the constant value  $c$  in equation (9.11) is zero. The MLEs (see Chapter 6) of all the parameters in the  $\gamma$ -RFE model are obtained as shown in Table 9.7.

Similarly, set  $p = 1$ , the MLE of all parameters in  $\beta$ -RFE model are obtained as shown in Table 9.8. For both RFE models, the MLE results can be used to obtain more insightful information about the software development process. In this example, at the time to stop testing the software  $T = 0.0184$ , the estimated number of remaining faults in the system is  $a_F = a - (p - q) \cdot m(T) = 55$ .

After we obtain the MLEs for all parameters, we can plot the mean value function fitting curves for both  $\gamma$ -RFE model and  $\beta$ -RFE model based on MLE parameters against the actual software application failures. Table 9.9 shows the mean value function fitting curves for both the models where the column  $m_\gamma(t)$  and  $m_\beta(t)$  show the mean value function for  $\gamma$ -RFE model and  $\beta$ -RFE model, respectively.

**Table 9.6.** Normalized cumulative failures and their times in operation

Time	Failures	Time	Failures	Time	Failures
0.0431	0.9055	0.3157	0.9751	0.7519	0.9900
0.0616	0.9104	0.3407	0.9751	0.7585	0.9900
0.0801	0.9204	0.3469	0.9751	0.7718	0.9900
0.0863	0.9254	0.3967	0.9751	0.7983	0.9900
0.1357	0.9303	0.4030	0.9801	0.8251	0.9900
0.1419	0.9353	0.4291	0.9851	0.8453	0.9900
0.1666	0.9453	0.4357	0.9851	0.8520	0.9900
0.2098	0.9453	0.4749	0.9851	0.9058	0.9900
0.2223	0.9502	0.5011	0.9851	0.9126	0.9900
0.2534	0.9502	0.5338	0.9851	0.9193	0.9900
0.2597	0.9502	0.5731	0.9851	0.9395	0.9950
0.2659	0.9502	0.6258	0.9900	0.9462	0.9950
0.2721	0.9552	0.6656	0.9900	0.9529	1.0000
0.2971	0.9602	0.6789	0.9900	0.9865	1.0000
0.3033	0.9701	0.7253	0.9900	1.0000	1.0000

The  $\gamma$ -RFE and  $\beta$ -RFE models yield very close fittings and predictions on software failures. Figure 9.4 shows the mean value function fitting curves of both the  $\gamma$ -RFE model and  $\beta$ -RFE model. Both models appear to be a good fit for a given data set. Figure 9.5 plots the detailed mean value fitting curves for both  $\gamma$ -RFE model and  $\beta$ -RFE model in the field operation.

For the overall fitting of the mean value function against the actual software failures, the MSE is 23.63 for  $\gamma$ -RFE model fitting, and is 23.69 for  $\beta$ -RFE model. Figure 9.6 shows the comparisons of mean value function fitting curves between the two RFE models and some existing NHPP software reliability models such as Goel-Okumoto and delayed S-shaped models (see Chapter 6). It appeared that those two models with considerations of the field environments on the software failure detection rate provide much better in term of predictions on the software failures in the field.

**Table 9.7.** MLE solutions for the  $\gamma$ -RFE model

$\hat{a}$	$\hat{b}$	$\hat{q}$	$\hat{\gamma}$	$\hat{\theta}$
236.58	0.00144	0	0.2137	10.713

**Table 9.8.** MLE solutions for the  $\beta$ -RFE model

$\hat{a}$	$\hat{b}$	$\hat{q}$	$\hat{\alpha}$	$\hat{\beta}$
236.07	0.00145	0	0.1862	8.6922

Once MLEs of all parameters in equations (9.16) and (9.19) are obtained, the software reliability within  $(T, T+x)$  can be determined as

$$R(x|T) = e^{- (m(T+x)-m(T))} \tag{9.20}$$

Let  $T = 0.0184$ , and change  $x$  from 0 to 0.001; then we can examine reliability predictions between two RFE models and some other NHPP models which assume constant failure detection rate for both software testing and operation. The reliability prediction curves are shown in Figure 9.7.

**Confidence Interval**

*Case 1:  $\gamma$ -RFE model.* In this section we construct confidence intervals for the prediction on software reliability in the random field environments. From Tables 9.4 and 9.5, if  $p=1$ ,  $c=0$  and  $q=0$ , then the model in equation (9.16) becomes

$$m(t) = \begin{cases} a(1-e^{-b \cdot t}) & t \leq T \\ a \left( 1-e^{-b \cdot T} \cdot \left( \frac{\theta}{\theta+b \cdot (t-T)} \right)^{\gamma} \right) & t \geq T \end{cases} \tag{9.21}$$

To obtain the confidence interval for the reliability predictions for  $\gamma$ -RFE model, we can derive the variance-covariance matrix for all the MLEs. If we use  $x_i$ ,  $i = 1, 2, 3$ , and 4 to denote all parameters in the model,

$$x_1 \rightarrow a \qquad x_2 \rightarrow b \qquad x_3 \rightarrow \theta \qquad x_4 \rightarrow \gamma$$

**Table 9.9.** The mean value functions for both RFEs models

Time	Failures	$m_{\gamma}(t)$	$m_{\beta}(t)$	Time	Failures	$m_{\gamma}(t)$	$m_{\beta}(t)$
0.0000	0.0000	0.0000	0.0000	0.1357	0.9303	0.9340	0.9341
0.0001	0.0249	0.0085	0.0085	0.1419	0.9353	0.9352	0.9354
0.0002	0.0299	0.0152	0.0152	0.1666	0.9453	0.9398	0.9399
0.0002	0.0647	0.0219	0.0219	0.2098	0.9453	0.9469	0.9467
0.0003	0.0647	0.0302	0.0302	0.2223	0.9502	0.9487	0.9485
0.0005	0.1095	0.0466	0.0467	0.2534	0.9502	0.9530	0.9525
0.0006	0.1194	0.0547	0.0548	0.2597	0.9502	0.9538	0.9533
0.0008	0.1443	0.0708	0.0709	0.2659	0.9502	0.9545	0.9540
0.0012	0.1692	0.1023	0.1025	0.2721	0.9552	0.9553	0.9547
0.0016	0.1990	0.1404	0.1406	0.2971	0.9602	0.9582	0.9575
0.0023	0.2289	0.1915	0.1917	0.3033	0.9701	0.9589	0.9582
0.0028	0.2637	0.2332	0.2335	0.3157	0.9751	0.9603	0.9594
0.0033	0.3134	0.2667	0.2670	0.3407	0.9751	0.9628	0.9618
0.0038	0.3483	0.3053	0.3056	0.3469	0.9751	0.9635	0.9624
0.0044	0.3532	0.3422	0.3426	0.3967	0.9751	0.9681	0.9667
0.0048	0.3682	0.3718	0.3721	0.4030	0.9801	0.9686	0.9672
0.0053	0.3881	0.4003	0.4007	0.4291	0.9851	0.9708	0.9692
0.0058	0.4478	0.4332	0.4336	0.4357	0.9851	0.9713	0.9697
0.0064	0.4876	0.4648	0.4651	0.4749	0.9851	0.9743	0.9725
0.0070	0.5224	0.4998	0.5002	0.5011	0.9851	0.9761	0.9742
0.0077	0.5473	0.5332	0.5335	0.5338	0.9851	0.9783	0.9762
0.0086	0.5821	0.5772	0.5775	0.5731	0.9851	0.9808	0.9785
0.0095	0.6119	0.6205	0.6208	0.6258	0.9900	0.9839	0.9813
0.0105	0.6368	0.6600	0.6602	0.6656	0.9900	0.9860	0.9833
0.0114	0.6468	0.6953	0.6955	0.6789	0.9900	0.9867	0.9839
0.0121	0.6766	0.7210	0.7211	0.7253	0.9900	0.9890	0.9860
0.0128	0.7015	0.7479	0.7479	0.7519	0.9900	0.9902	0.9871
0.0135	0.7363	0.7684	0.7684	0.7585	0.9900	0.9905	0.9874
0.0142	0.7761	0.7924	0.7924	0.7718	0.9900	0.9911	0.9879
0.0147	0.7761	0.8050	0.8049	0.7983	0.9900	0.9923	0.9890
0.0155	0.8159	0.8294	0.8292	0.8251	0.9900	0.9934	0.9900
0.0164	0.8259	0.8522	0.8520	0.8453	0.9900	0.9943	0.9908
0.0172	0.8408	0.8713	0.8710	0.8520	0.9900	0.9945	0.9910
0.0176	0.8458	0.8804	0.8801	0.9058	0.9900	0.9966	0.9929
0.0180	0.8756	0.8897	0.8893	0.9126	0.9900	0.9969	0.9932
0.0184	0.8955	0.8987	0.8983	0.9193	0.9900	0.9971	0.9934
0.0184	0.9005	0.8995	0.8991	0.9395	0.9950	0.9979	0.9941
0.0431	0.9055	0.9092	0.9092	0.9462	0.9950	0.9981	0.9943
0.0616	0.9104	0.9153	0.9155	0.9529	1.0000	0.9983	0.9945
0.0801	0.9204	0.9208	0.9210	0.9865	1.0000	0.9995	0.9956
0.0863	0.9254	0.9224	0.9227	1.0000	1.0000	1.0000	0.9960

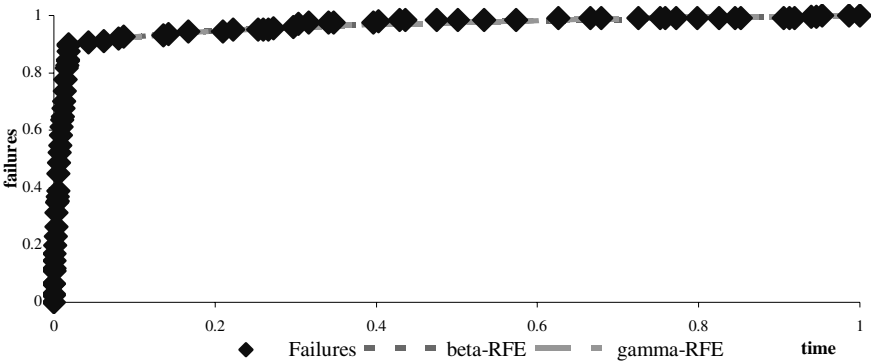


Figure 9.4. Mean value function fitting curves for both RFE models

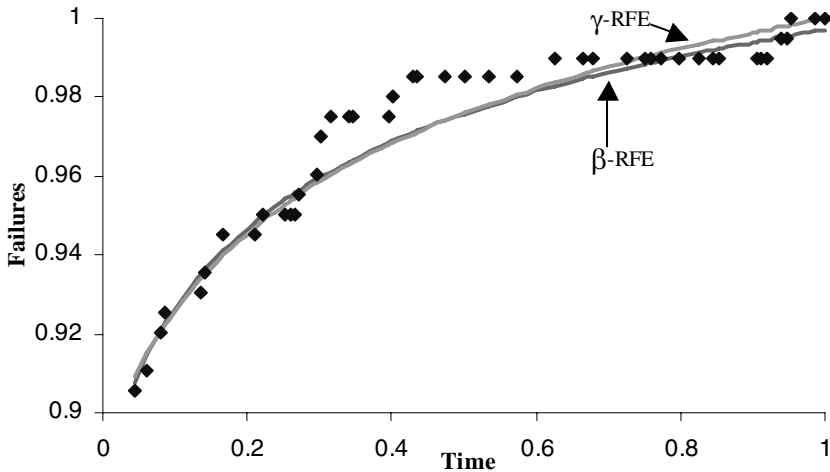


Figure 9.5. Mean value function fitting comparisons

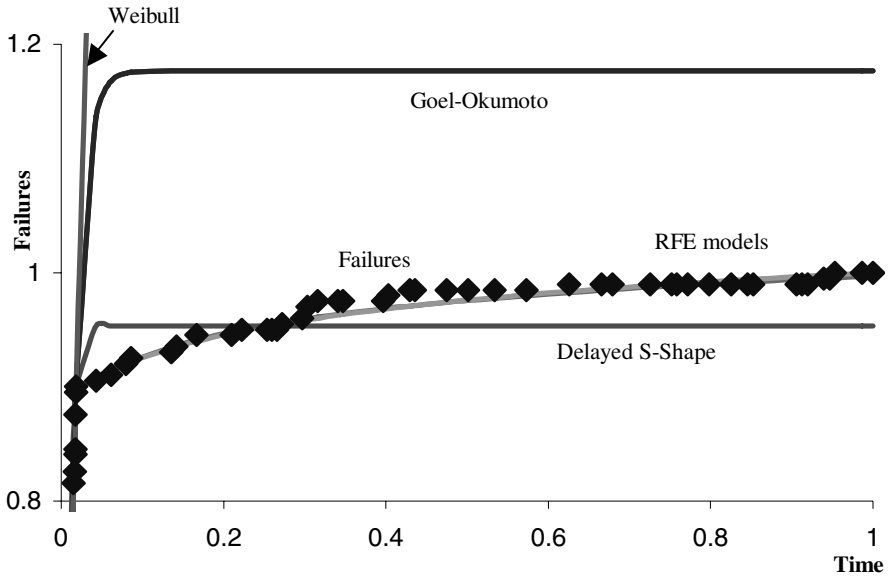


Figure 9.6. Model comparisons

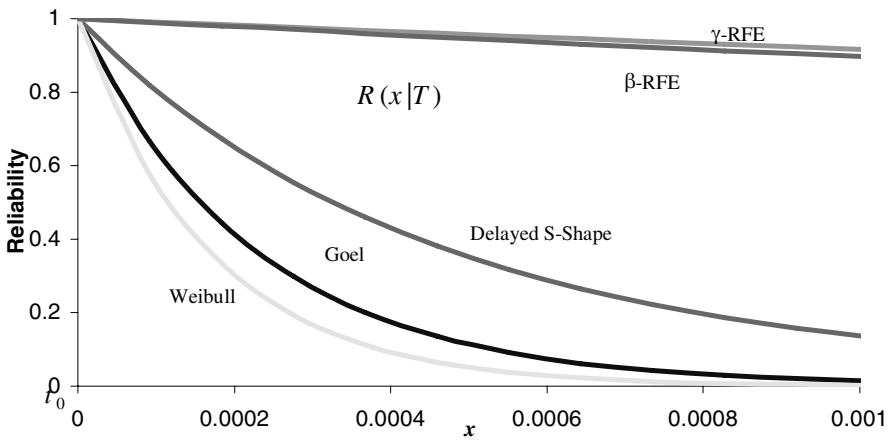


Figure 9.7. Reliability prediction comparisons

The Fisher information matrix  $H$  can be obtained as

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{14} \\ h_{21} & h_{22} & h_{23} & h_{24} \\ h_{31} & h_{32} & h_{33} & h_{34} \\ h_{41} & h_{42} & h_{43} & h_{44} \end{bmatrix} \quad (9.22)$$

where

$$h_{ij} = E \left[ -\frac{\partial^2 L}{\partial x_i \partial x_j} \right] \quad i, j = 1, \dots, 6 \quad (9.23)$$

where  $L$  is the log-likelihood function.

If we denote  $z(t_k) = m(t_k) - m(t_{k-1})$  and  $\Delta y_k = y_k - y_{k-1}$ ,  $k = 1, 2, \dots, n$ , then we have

$$\frac{\partial^2 L}{\partial x_i \partial x_j} = \sum_{k=1}^n \left( -\frac{\Delta y_k}{z(t_k)^2} \cdot \frac{\partial z(t_k)}{\partial x_i} \cdot \frac{\partial z(t_k)}{\partial x_j} + \left( \frac{\Delta y_k - z(t_k)}{z(t_k)} \cdot \frac{\partial^2 z(t_k)}{\partial x_i \partial x_j} \right) \right) \quad (9.24)$$

Then we can obtain the each element in Fisher information matrix  $H$ . For example,

$$\begin{aligned} h_{11} &= E \left[ -\frac{\partial^2 L}{\partial x_1^2} \right] \\ &= \sum_{k=1}^n \left( \sum_{\Delta y_k=0}^{\infty} \left( \frac{\Delta y_k}{z(t_k)^2} \cdot \left( \frac{\partial z(t_k)}{\partial a} \right)^2 \right) \cdot \frac{(z(t_k))^{\Delta y_k} \cdot e^{-z(t_k)}}{(\Delta y_k)!} \right) \\ &= \sum_{k=1}^n \left( \sum_{\Delta y_k=0}^{\infty} \left( \frac{\Delta y_k}{z(t_k)^2} \cdot \left( \frac{z(t_k)}{a} \right)^2 \right) \cdot \frac{(z(t_k))^{\Delta y_k} \cdot e^{-z(t_k)}}{(\Delta y_k)!} \right) \\ &= \sum_{k=1}^n \left( \frac{1}{a^2} \cdot \sum_{\Delta y_k=0}^{\infty} \Delta y_k \cdot \frac{(z(t_k))^{\Delta y_k} \cdot e^{-z(t_k)}}{(\Delta y_k)!} \right) \\ &= \sum_{k=1}^n \left( \frac{1}{a^2} \cdot z(t_k) \right) \\ &= \frac{1}{a^2} \cdot m(t_n) \end{aligned} \quad (9.25)$$

The variance matrix,  $V$ , can also be obtained

$$V = [H]^{-1} = \begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{14} \\ v_{21} & v_{22} & v_{23} & v_{24} \\ v_{31} & v_{32} & v_{33} & v_{34} \\ v_{41} & v_{42} & v_{43} & v_{44} \end{bmatrix} \quad (9.26)$$

The variances of all the estimate parameters are given by

$$\begin{aligned} Var(\hat{a}) &= Var(x_1) = v_{11} \\ Var(\hat{b}) &= Var(x_2) = v_{22} \\ Var(\hat{\gamma}) &= Var(x_3) = v_{33} \\ Var(\hat{\theta}) &= Var(x_4) = v_{44} \end{aligned} \quad (9.27)$$

The actual numerical results for the  $\gamma$ -RFE model variance matrix is

$$V_\gamma = \begin{bmatrix} 703.8472 & -0.005387 & -88.6906 & -2.6861 \\ -0.005387 & 7.3655 \times 10^{-8} & 1.11 \times 10^{-3} & 3.097 \times 10^{-5} \\ -88.6906 & 1.11 \times 10^{-3} & 92.4287 & 1.1843 \\ -2.6861 & 3.097 \times 10^{-5} & 1.1843 & 0.0238 \end{bmatrix} \quad (9.28)$$

*Case2:  $\beta$ -RFE model.* The mean value function in equation (9.19) can also be simplified, where the estimate of  $q=0$  and set  $c=0$ ,  $p=1$ , is as follows:

$$m_\beta(t) = \begin{cases} a(1 - e^{-bt}) & t \leq T \\ a \left( 1 - e^{-bT} \sum_{k=0}^{\infty} \left[ \frac{\Gamma(\alpha + \beta) \Gamma(\beta + k) P(k, b(t - T))}{\Gamma(\beta) \Gamma(\alpha + \beta + k)} \right] \right) & t \geq T \end{cases} \quad (9.29)$$

The above model leads to the same MLE results for parameter  $a$ ,  $b$ ,  $\alpha$  and  $\beta$ , and also yields exactly the same mean value function fittings and predictions. To obtain the confidence interval for the reliability predictions for  $\beta$ -RFE model, we need to obtain the variance-covariance matrix for all maximum likelihood estimates.

If we use  $x_i$ ,  $i = 1, 2, 3$ , and 4, to denote all parameters in the model, or

$$x_1 \rightarrow a \quad x_2 \rightarrow b \quad x_3 \rightarrow \alpha \quad x_4 \rightarrow \beta$$

Going through the similar steps as for  $\gamma$ -RFE model, the actual numerical results for the  $\beta$ -RFE model variance matrix can be obtained as

$$V_\beta = \begin{bmatrix} 691.2 & -0.00536 & -2.728 & -66.2172 \\ -0.00536 & 7.4485 \times 10^{-8} & 2.671 \times 10^{-5} & 0.00085 \\ -2.7652 & 2.671 \times 10^{-5} & 0.01820 & 0.8295 \\ -66.2172 & 0.00085 & 0.8295 & 60.5985 \end{bmatrix} \quad (9.30)$$

### Confidence Interval of Reliability Predictions

If we define a partial derivative vector for reliability  $R(x|t)$  as

$$vR(x|t) = \left[ \frac{\partial R(x|t)}{\partial x_1}, \frac{\partial R(x|t)}{\partial x_2}, \frac{\partial R(x|t)}{\partial x_3}, \frac{\partial R(x|t)}{\partial x_4} \right] \quad (9.31)$$



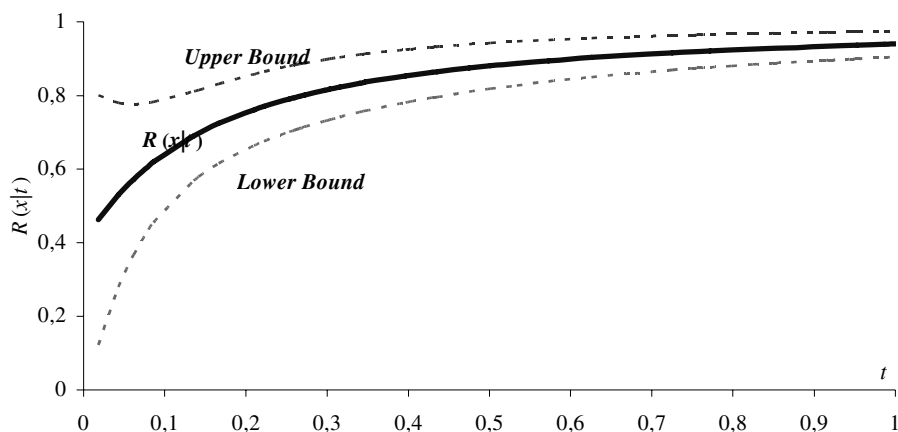
then the variance of  $R(x|t)$  in equation (20) can be obtained as

$$\text{Var}[R(x|t)] = vR(x|t) \cdot V \cdot (vR(x|t))^T \quad (9.32)$$

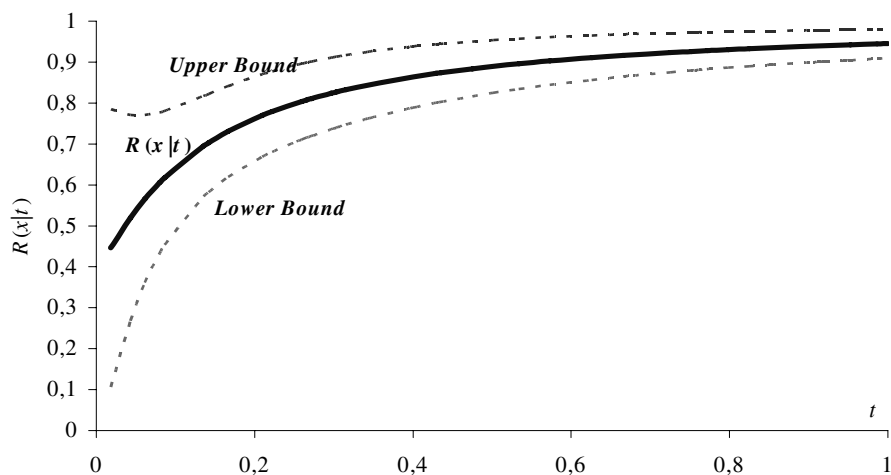
Assume the reliability estimation follows normal distribution, then the 95% confidence interval for reliability prediction  $R(x|t)$  is

$$[R(x|t) - 1.96\sqrt{\text{Var}[R(x|t)]}, R(x|t) + 1.96\sqrt{\text{Var}[R(x|t)]}]. \quad (9.33)$$

Figures 9.8 and 9.9 show the 95% confidence interval of the reliability predicted by  $\gamma$ -RFE and  $\beta$ -RFE models, respectively.

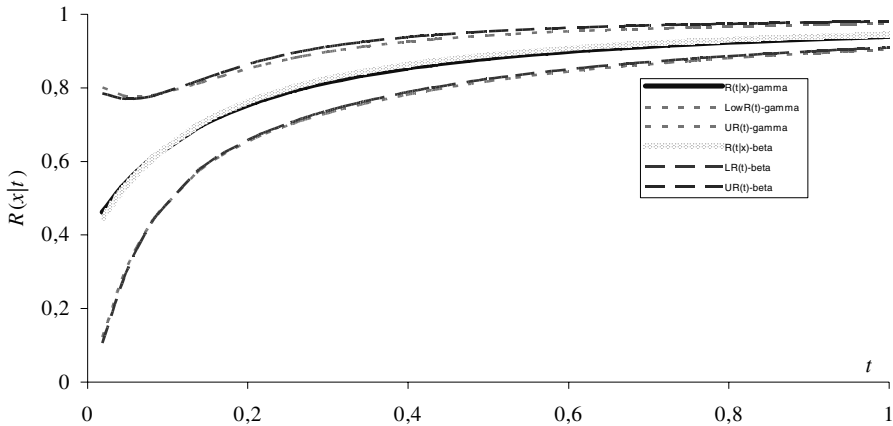


**Figure 9.8.**  $\gamma$ -RFE model reliability growth curve and its 95% confidence interval



**Figure 9.9.**  $\beta$ -RFE model reliability growth prediction and its 95% confidence interval

We plot the reliability predictions and their 95% confidence interval by both  $\gamma$ -RFE model and  $\beta$ -RFE model in Figure 9.10. For this given application set of data, the reliability predictions by  $\gamma$ -RFE model and  $\beta$ -RFE model are very close to each other, so are their confidence intervals.



**Figure 9.10.** Reliability growth prediction curves and their 95% confidence intervals for  $\gamma$ -RFE model and  $\beta$ -RFE model

## 9.5 Further Reading

H. Pham, “Recent studies in software reliability engineering,” chapter 16 in the Handbook of Reliability Engineering, H. Pham (ed.), Springer, 2003

W. A. Arbaugh, W.L. Fithen and J. McHugh, “Windows of vulnerability: A case study analysis,” IEEE Computer, December 2000

N.A. Streitz, C. Rucker, T. Prante, D. Alphen, R. Stenzel, and C. Magerkurth, “Designing smart artifacts for smart environments,” IEEE Computer, March 2005

X. Teng and H. Pham, “A software cost model for quantifying the gain with considerations of random field environments,” IEEE Trans on Computers, vol 53, no. 3, 2004

## 9.6 Problems

1. Derive the mean value function given in equation (9.16).
2. Derive the mean value function given in equation (9.19).

## Optimal Release Policies

### 10.1 Introduction

The quality of the software system usually depends on how much time testing takes and what testing methodologies are used. On the one hand, the more time people spend on testing, the more errors can be removed, which leads to more reliable software; however, the testing cost of the software will also increase. On the other hand, if the testing time is too short, the cost of the software could be reduced, but the customers may take a higher risk of buying unreliable software (Pham 1999a; Zhang 1998a). This will also increase the cost during the operational phase, since it is much more expensive to fix an error during the operational phase than during the testing phase. Therefore, it is important to determine when to stop testing, and release the software.

In defining important software cost factors, a cost model should help software developers and managers answer the following questions:

1. How should resources be scheduled to ensure the on-time and efficient delivery of a software product?
2. Is the software product sufficiently reliable for release (*e.g.*, have we done enough testing)?
3. What information does a manager or software developer need to determine the release of software from current software testing activities?

This chapter discusses several recent generalized cost models based on the NHPP software reliability functions. It aims to help answer these questions by determining the optimal testing release policies of the software systems. In addition to the costs of traditional cost models, the cost models to be discussed in this chapter consider other features such as the testing cost, debugging cost during testing phase, debugging cost during the warranty period, and risk cost due to software failure. These models can be used to estimate the realistic total software cost for applications such as telecommunications, customer service, and real-time embedded systems. The following notations and basic assumptions are used throughout this chapter.

$m(T)$	Expected number of errors to be detected by time $T$
$a$	Total number of software errors to be eventually detected
$b$	Exponential index
$x$	Mission time
$R(x T)$	Reliability function of software by time $T$ for a mission time $x$
$T$	Software release time
$C_1$	Software test cost per unit time
$C_2$	Cost of removing each error per unit time during testing
$E(T)$	Expected total cost of a software system by time $T$
$N(T)$	Number of errors to be detected by time $T$
$Y$	Time to remove an error during testing phase
$\mu_y$	Expected time to remove an error during testing phase which is $E(Y)$

### General Assumptions

- (1) The cost to perform testing is proportional to the testing time.
- (2) The cost to remove errors during the testing phase is proportional to the total time of removing all errors detected by the end of the testing phase.
- (3) The time to remove each error during testing follows a truncated exponential distribution.
- (4) There is a risk cost related to the reliability at each release time point.

Let  $Y$  be a random variable of time to remove an error. Based on assumption (3), the probability density distribution of  $Y$  is given by

$$s(y) = \frac{\lambda e^{-\lambda y}}{\int_0^{T_0} \lambda e^{-\lambda z} dz} \quad \text{for } 0 \leq y \leq T_0$$

where  $T_0$  is the maximum time to remove an error. The expected time to remove each error is

$$\begin{aligned} \mu_y = E(Y) &= \int_0^{T_0} y s(y) dy \\ &= \int_0^{T_0} \frac{y \lambda e^{-\lambda y}}{\int_0^{T_0} \lambda e^{-\lambda z} dz} dy \end{aligned}$$

After simplifications, we obtain

$$\mu_y = \frac{1 - (\lambda T_0 + 1)e^{-\lambda T_0}}{\lambda(1 - e^{-\lambda T_0})} \quad (10.1)$$

## 10.2 A Software Cost Model with Risk Factor

This section discusses a cost model addressing the risk level and the time to remove errors. The optimal release policies that minimize the expected total

software cost are obtained. Without loss of generality, the Goel-Okumoto NHPP model will be used as a reliability function for this cost model. In other words, the Goel-Okumoto NHPP mean value function  $m(T)$  is given by

$$m(T) = a(1 - e^{-bT}) \quad (10.2)$$

The error detection rate function is

$$\lambda(T) = abe^{-bT} \quad (10.3)$$

and the reliability of the software is

$$\begin{aligned} R(x | T) &= e^{-[m(T+x) - m(T)]} \\ &= e^{-a[e^{-bT} - e^{-b(T+x)}]} \end{aligned} \quad (10.4)$$

The expected software system cost,  $E(T)$ , is defined as: (1) the cost to perform testing; (2) the cost incurred in removing errors during the testing phase; and (3) a risk cost due to software failure.

The cost to perform testing is given by

$$E_1(T) = C_1 T \quad (10.5)$$

The expected total time to remove all  $N(T)$  errors is

$$E\left[\sum_{i=1}^{N(T)} Y_i\right] = E[N(T)]E[Y_i] = m(T)\mu_y$$

where  $\mu_y$  is given in equation (10.1). Hence, the expected cost to remove all errors detected by time  $T$  can be expressed as

$$E_2(T) = C_2 E\left[\sum_{i=1}^{N(T)} Y_i\right] = C_2 m(T)\mu_y \quad (10.6)$$

The risk cost due to software failure after releasing the software is

$$E_3(T) = C_3 [1 - R(x | T)] \quad (10.7)$$

where  $C_3$  is the cost due to software failure.

Therefore, the expected total software cost can be expressed (Zhang 1998) as

$$E(T) = C_1(T) + C_2 m(T)\mu_y + C_3 [1 - R(x | T)] \quad (10.8)$$

Define

$$f(T) = \lambda(T)[C_3(1 - e^{-bx})R(x | T) - C_2\mu_y] \quad (10.9)$$

$$g(T) = C_3(1 - e^{-bx})R(x | T)[1 - ae^{-bT}(1 - e^{-bx})]$$

It should be noted that  $g(T)$  is a strictly increasing function of  $T$ .

**Theorem 10.1 (Zhang and Pham 1998):** Given  $C_1$ ,  $C_2$ ,  $C_3$ ,  $x$ , and  $\mu_y$ , the optimal value of  $T$ , say  $T^*$ , which minimizes the expected total cost of the software can be determined as follows:

(1) If  $g(0) > C_2\mu_y$ , then

(a) If  $f(0) \leq C_1$ , then  $T^* = 0$ .

- (b) If  $f(\infty) > C_l$ , then  $T^* = \infty$ .
- (c) If  $f(0) > C_l$ ,  $f(T) \geq C_l$ , for any  $T \in (0, T')$  and  $f(T) < C_l$ , for any  $T \in (T', \infty)$ , then  $T^* = T'$  where  $T' = \inf\{T: f(T) < C_l\}$
- (2) If  $g(\infty) < C_2 \mu_y$ , then
- (a) If  $f(0) \geq C_l$ , then  $T^* = \infty$ .
- (b) If  $f(\infty) < C_l$ , then  $T^* = 0$ .
- If  $f(0) < C_l$ ,  $f(T) \leq C_l$  for any  $T \in (0, T')$  and  $f(T') > C_l$ , for any  $T \in (T', \infty)$ , then
- $$T^* = 0 \text{ if } E(0) < E(\infty)$$
- $$T^* = \infty \text{ if } E(0) \geq E(\infty)$$
- where  $T' = \inf\{T: f(T) > C_l\}$
- (3) If  $g(0) < C_2 \mu_y$ ,  $g(T) \leq C_2 \mu_y$  for  $T \in (0, T_0)$  and  $g(T) > C_2 \mu_y$ , for  $T \in (T_0, \infty)$ , then
- If  $f(0) < C_l$ , then
- $$T^* = 0 \text{ if } E(0) < E(T_b)$$
- $$T^* = T_b \text{ if } E(0) \geq E(T_b)$$
- where  $T_b = \inf\{T: f(T) < C_l, T > T_a\}$
- If  $f(0) > C_l$ , then  $T^* = T_b'$  where  $T_b' = \inf\{T: f(T) < C_l\}$

**Proof.** See (Zhang 1998).

*Example 10.1:* Assuming a software failure data is given in Table 4.14 (data set #10), the parameters of the Goel-Okumoto model using MLE is given by

$$\hat{a} = 142.32, \quad \hat{b} = 0.1246$$

The mean value function becomes

$$\begin{aligned} m(T) &= a(1 - e^{-bT}) \\ &= 142.32(1 - e^{-0.1246T}) \end{aligned}$$

Given  $C_1 = \$25$ ,  $C_2 = \$200$ ,  $C_3 = \$7,000$ ,  $\mu_y = 0.1$ , and  $x = 0.05$ , from Theorem 10.1, the results are shown in Table 10.1. The optimal release time in this case is  $T^* = 21.5$  hours and the corresponding expected total cost is \$3,600.49.

If we increase the value of  $C_3$  from \$7,000 to \$10,000, we would expect to have a longer testing time. In this case ( $C_1 = \$25$ ,  $C_2 = \$200$ ,  $C_3 = \$10,000$ ,  $\mu_y = 0.1$ , and  $x = 0.05$ ), the optimal release time is  $T^* = 27$  hours and the corresponding expected total cost is \$3,723.95.

**Table 10.1.** Optimal release time

<u>Release time <math>T^*</math> (hours)</u>	<u>Expected total cost <math>E(T)</math></u>
19.5	3,607.34
20.0	3,604.47
20.5	3,602.39
21.0	3,601.07
21.5*	3,600.49
22.0	3,600.60
22.5	3,601.39
23.0	3,602.81
23.5	3,604.83

### 10.3 Cost Model with Testing Coverage

In this section, a software cost model incorporating testing coverage is discussed. Besides some traditional cost items such as testing cost and error removal cost, risk cost due to potential faults in the uncovered code is included associated with the number of demands from customers. The optimal release policies that minimize the expected total cost subject to the reliability requirement are described.

#### Model Formulation

A software cost model is developed based on the following assumptions:

- (1)-(3) These assumptions are same as (1)-(3) in the *General Assumptions* (in Section 10.1).
- (4) The mean value function with consideration of testing coverage is given in Theorem 7.2. It is also called the PZ-coverage model
- (5) There is a risk cost associated with the testing coverage. A software provider has to pay each customer a certain amount of money for potential faults in uncovered code.
- (6) The reliability of the software at release time must satisfy the customers' requirement.

The expected software system cost  $E(T)$  is defined as: (1) cost to do testing,  $E_1(T)$ ; (2) cost incurred in removing errors during the testing phase,  $E_2(T)$ ; and (3) risk cost due to potential faults remaining in the uncovered software code,  $E_3(T)$ . The two functions  $E_1(T)$  and  $E_2(T)$  are the same as in equations (10.5) and (10.6), respectively. Let us assume there are  $D$  customers. The provider has to pay each of them a certain amount of money,  $C_3$ , for potential risk due to faults remaining in the uncovered code. Without lack of generality, we assume the demand  $D$  is a constant number. It is not difficult to relax this assumption by studying the distribution of demand and calculating the mean value. The total risk cost corresponding to testing coverage,  $E_3(T)$ , can be expressed as follows:

$$E_3(T) = C_3 D (1 - c(T)). \quad (10.10)$$

Therefore, the expected total software cost  $E(T)$  can be expressed as following:

$$E(T) = C_1 T + C_2 m(T) \mu_y + C_3 D[1 - c(T)] \quad (10.11)$$

where  $\mu_y$  is given in equation (10.1). From Theorem 7.2, the mean value function  $m(T)$  and the testing coverage function  $c(T)$  are, respectively,

$$m(T) = a \left( 1 + \alpha T - \frac{bT+1}{e^{bT}} \right) - \frac{a\alpha(1+bT)}{be^{bT+1}} \left( \ln(bT+1) + \sum_{i=0}^{\infty} \frac{(1+bT)^{i+1} - 1}{(i+1)!(i+1)} \right) \quad (10.12)$$

and

$$c(T) = 1 - (1+bT)e^{-bT} \quad (10.13)$$

### Optimal Software Release Policies

We now determine the optimal software release time that minimizes the expected total software cost. In other words, we wish to find the value of  $T$  such that  $E(T)$  is minimized. Software reliability  $R(x/T)$  is another criterion that should be considered in the optimal release policies, especially for safety-critical applications.

Therefore, we determine the optimal release time that minimizes the expected software cost subject to attaining a desired reliability level,  $R_0$ . The optimization problem can be formulated as

Minimize  $E(T)$

Subject to  $R(x/T) \geq R_0$

where  $E(T)$  is given in equation (10.11).

It can be shown that the software reliability function  $R(x/T)$  increases as  $T$  increases. Let  $T_R$  be the solution of  $R(x/T_R) = R_0$ . That is,  $T_R = \{T : R^{-1}(x/T) = R_0\}$ . Define

$$h(T) = \ln(1+bT) + \sum_{i=0}^{\infty} \frac{(1+bT)^{i+1} - 1}{(i+1)!(i+1)} \quad (10.14)$$

$$g(T) = A - C_2 \mu_y a b \alpha h(T) \quad (10.15)$$

$$C = C_2 \mu_y a \alpha, C > 0 \quad (10.16)$$

$$u(T) = T g(T) + C \quad (10.17)$$

$$v(T) = e^{-(1+bT)}, v(T) > 0, \forall T \quad (10.18)$$

$$f(T) = C_1 - v(T) u(T) \quad (10.19)$$

$$A = (C_3 D - C_2 \mu_y a) e^{b^2} \quad (10.20)$$

$$T_g = g^{-1}(0) \quad (10.21)$$

It should be noted that  $h(T)$  is a strictly increasing function of  $T$  and  $g(T)$  is a strictly decreasing function of  $T$ .



**Theorem 10.2 (Pham and Zhang 2003d):** Given  $C_1, C_2, C_3, x, \mu_y, D$ , the optimal value of  $T$ , say  $T^*$ , which minimizes the expected total cost  $E(T)$  subject to software reliability requirement  $R_0$  is determined as follows:

*Case 1.* If  $A \geq 0$ , then

(1) If  $T_g > 0$ ,

if  $T_{f2} > T_R$ , then  $T^* = T_{f2}$  minimizes  $E(T)$ ;

if  $T_{f2} < T_R$ ,  $T^* = T_R$

where  $T_{f2} = \{T : T > T_{f1}, T = f^{-1}(0)\}$  and  $T_{f1} = f^{-1}(0)$ .

(2) If  $T_g \leq 0$  then  $g(T) \leq 0, \forall T$ . The function  $u(T)$  intersects with T-axis

at only one point  $T_u = u^{-1}(0)$ . Since  $u(T) \geq 0$  for  $T \in [0, T_u]$  and  $u(T) < 0$  for  $T > T_u$ , then this subcase becomes the same as subcase (1) for the following discussion

If  $T_{f2} > T_R$ ,  $T^* = T_{f2}$  minimizes  $E(T)$ ;

If  $T_{f2} < T_R$ ,  $T^* = T_R$ .

*Case 2.* If  $A < 0$ , then  $f(T)$  is a strictly increasing and positive function of  $T$ . The expected total cost function  $E(T)$  will be a strictly increasing and convex function of  $T$ . Hence,  $T^* = 0$  minimizes  $E(T)$ .

**Proof:** See Problem 4

**Application 10.1:** In this example, we use AT&T system T data to illustrate how we can determine the optimal release time using Theorem 2.

The cost coefficients in the cost model are usually determined by empirical data, previous experiences, or the nature of the applications. The following parameter values are specified in terms of staff recourse according to a project data collected by AT & T researchers (Ehrlich 1993). The unit is staff-units.

The testing cost coefficient,  $C_1$ , can be estimated to be about 600-700 staff-units. It is estimated that there are 370 CPU test-execution unit during testing with 1.9 staff-units per CPU unit. The error removal cost coefficient during testing period,  $C_2$ , is about 60 staff-units per error.

The risk cost coefficient,  $C_3$ , is the cost due to potential faults in the uncovered code. The value of this cost depends upon the nature of the applications. The cost may include the loss of revenues, customers, and even human life. Let's assume the demand,  $D$ , is 100. For commercial applications the demand is usually higher, while for safety-critical applications, the risk coefficient,  $C_3$ , itself is usually higher because of the safety requirement.

Based on the above information, let us consider the following coefficients in the cost model.

*Example 10.2:* Given  $C_1 = 600$ ,  $C_2 = 60$ ,  $C_3 = 10,000$ ,  $\mu_y = 0.8$ ,  $x = 4.0$ ,  $D = 100$ , and assume that the reliability requirement  $R_0$  is 0.90. After determining those coefficient values of parameters and from Theorem 10.2, one can easily find the

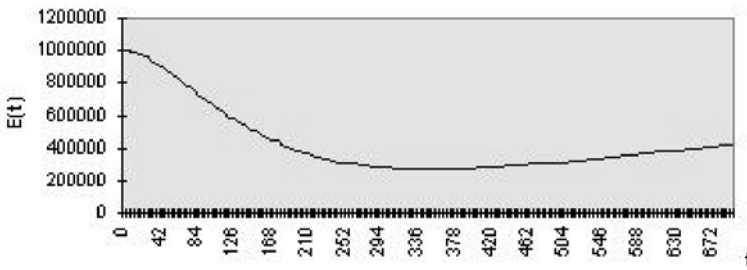
optimal release time  $T^*$  that minimizes the expected total cost  $E(T)$ . Figure 10.1 illustrates the cost function versus the testing time. The results are as follows:

$$T^* = 353.5 \text{ days and } E(353.5) = 269407.2$$

The software reliability at time  $T^* = 353.5$  is

$$R(353.5) = 0.8868.$$

In this case, the reliability requirement is not satisfied yet the derised requirement. The testing coverage at this time is  $c(353.5) = 0.9435$  or 94%. This indicates that one would need to continue testing the software until the software reliability exceeds 0.90. From the reliability analysis, we can see that when  $T = 409.5$  days, the software reliability  $R(409.5) = 0.90006$ . This implies that we need to test our software for additional 56 days and the testing coverage at day 409.5 is  $c(409.5) = 0.9692$  or 97%.



**Figure 10.1.** The Expected total cost function vs time

*Example 10.3:* Given  $C_1 = 650$ ,  $C_2 = 65$ ,  $C_3 = 20,000$ ,  $\mu_y = 0.8$ ,  $x = 2.0$ ,  $D = 100$  and assuming that the reliability requirement  $R_0$  is 0.95.

Using Theorem 10.2, we can easily obtain the optimal release time  $T^*$  that minimizes the expected total cost  $E(T)$ . The result is given as follows:

$$T^* = 413 \text{ days and } E(413) = 328724.1 \text{ dollars.}$$

The corresponding software reliability is given by

$$R(413) = 0.94987.$$

Figure 10.2 illustrates the cost function versus the testing time. We should consider that the software reliability requirement is satisfied. The testing coverage at this time is  $c(413) = 0.970328$  or 97%. This indicates that we can stop testing the software.

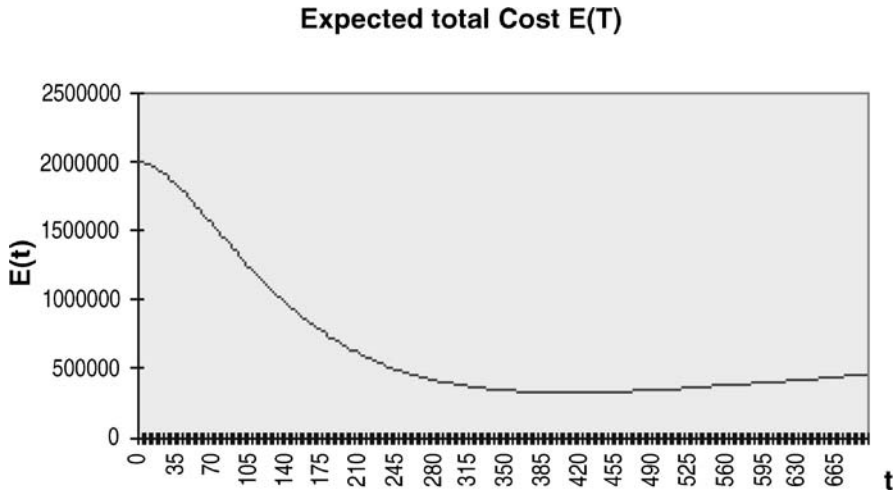


Figure 10.2. The expected total cost function

## 10.4 A Generalized Software Cost Model

In addition to the cost factors presented in Section 10.2, this section describes a generalized cost model considering the cost of removing errors detected during the warranty period and the risk cost due to software failure. In this section, we use the following notations and assumptions.

### Notation

$C_0$	Set-up cost for software testing
$C_3$	Cost of removing an error per unit time during the operational phase
$C_4$	Loss due to software failure
$W$	Variable of time to remove an error during the warranty period in the operation phase
$\mu_w$	Expected time to remove an error during the warranty period in the operation phase, which is $E(W)$
$T_w$	Period of warranty time
$\alpha$	The discount rate of the testing cost

### Additional Assumptions

- (1)-(4) Same as General Assumptions 1 - 4 in Section 10.2
- (5) There is a set-up cost at the beginning of the software development process.
- (6) The cost of testing is a power function of the testing time. This means that at the beginning of the testing, the cost increases with a higher gradient, slowing down later.
- (7) The time to remove each error during the warranty period follows a truncated exponential distribution.
- (8) The cost to remove errors during the warranty period is proportional to the total time of removing all errors detected during the interval of  $(T, T_w)$ .

Similarly, from assumption 7, the truncated exponential density function of error removal time during the warranty period is

$$q(w) = \frac{\lambda_w e^{-\lambda_w w}}{\int_0^{T'_0} \lambda_w e^{-\lambda_w x} dx} \quad \text{for } 0 \leq w \leq T'_0 \quad (10.22)$$

Therefore, the expected time to remove an error during the warranty period is

$$\mu_w = \frac{1 - (\lambda_w T'_0 + 1)e^{-\lambda_w T'_0}}{\lambda_w (1 - e^{-\lambda_w T'_0})} \quad (10.23)$$

The expected software system cost comprises of the set-up cost, the cost to do testing, the cost incurred in removing errors during the testing phase and during the warranty period, and the risk cost in releasing the software system by time  $T$ . Hence, the expected total software system cost  $E(T)$  can be expressed as follows (Pham 1999c):

$$E(T) = C_0 + C_1 T^\alpha + C_2 m(T) \mu_y + C_3 \mu_w [m(T + T_w) - m(T)] + C_4 [1 - R(x|T)] \quad (10.24)$$

where  $0 \leq \alpha \leq 1$ . Define

$$\begin{aligned} y(T) &= \alpha C_1 T^{(\alpha-1)} - \mu_w C_3 a b e^{-bT} (1 - e^{-bT_w}) \\ &\quad - a b e^{-bT} [C_4 (1 - e^{-bx}) R(x|T) - C_2 \mu_y] \\ u(T) &= a b^2 C_4 (1 - e^{-bx}) R(x|T) [1 - a e^{-bT} (1 - e^{-bx})] \\ &\quad + \alpha(\alpha - 1) C_1 T^{(\alpha-2)} e^{bT} \\ C &= C_2 \mu_y a b^2 - \mu_w C_3 a b^2 (1 - e^{-bT_w}) \end{aligned} \quad (10.25)$$

It can be shown that the function  $u(T)$  is an increasing function of  $T$  (see Problem 2). The optimal software release time,  $T^*$ , which minimizes the expected total system cost is given below.

**Theorem 10.3 (Pham and Zhang, 1999c):** Given  $C_0, C_1, C_2, C_3, C_4, x, \mu_y, \mu_w, T_w$ , the optimal value of  $T$ , say  $T^*$ , which minimizes the expected total cost of the software is as follows:

- (1) If  $u(0) \geq C$ , and
  - (a) If  $y(0) \geq 0$ , then  $T^* = 0$ ;
  - (b) If  $y(\infty) < 0$ , then  $T^* = \infty$ .
  - (c) If  $y(0) < 0$ ,  $y(T) < 0$ , for  $T \in (0, T')$  and  $y(T) > 0$ , for  $T \in (T', \infty)$ , then  $T^* = T'$  where  $T' = y^{-1}(0)$
- (2) If  $u(\infty) < C$  and
  - (a) If  $y(0) \leq 0$ , then  $T^* = \infty$ .
  - (b) If  $y(\infty) > 0$ , then  $T^* = 0$ .
  - (c) If  $y(0) > 0$ ,  $y(T) > 0$  for  $T \in (0, T'')$  and  $y(T) < 0$  for  $T \in (T'', \infty)$ , then  $T^* = 0$  if  $E(0) \leq E(\infty)$

- $T^* = \infty$  if  $E(0) > E(\infty)$   
 where  $T'' = y^{-1}(0)$
- (3) If  $u(0) < C$ ,  $u(T) \leq C$  for  $T \in (0, T_0)$  and  $u(T) > C$  for  $T \in (T_0, \infty)$ , where  $T_0 = u^{-1}(C)$ , then
- (a) If  $y(0) \geq 0$ , then  
 $T^* = 0$  if  $E(0) \leq E(T_b)$   
 $T^* = T_b$  if  $E(0) > E(T_b)$   
 where  $T_b = \inf\{T > T_a: y(T) > 0\}$
- (b) If  $y(0) < 0$ , then  $T^* = T_b$ , where  $T_b = y^{-1}(0)$

**Proof:** Taking the first derivative of  $E(T)$ , we obtain

$$\begin{aligned}\frac{\partial E(T)}{\partial T} &= \alpha C_1 T^{(\alpha-1)} - \mu_w C_3 a b e^{-bT} (1 - e^{-bT_w}) \\ &\quad - a b e^{-bT} [C_4 (1 - e^{-bx}) R(x|T) - C_2 \mu_y] \\ &= y(T)\end{aligned}$$

The second derivative of  $E(T)$ ,

$$\frac{\partial^2 E(T)}{\partial T^2} = e^{-bT} [u(T) - C]$$

*Case 1:* If  $u(0) \geq C$ , then  $u(T) > C$  for any  $T$ . In this case,  $y(T)$  is a strictly increasing function of  $T$  and

$$\frac{\partial^2 E(T)}{\partial T^2} > 0$$

There are three subcases:

- (a) If  $y(0) > 0$ , then  $y(T) > 0$  for all  $T$  and  $E(T)$  is strictly increasing in  $T$ . Hence,  $T^* = 0$  minimizes  $E(T)$ .
- (b) If  $y(\infty) < 0$ , then  $y(T) < 0$  for all  $T$  and  $E(T)$  is decreasing in  $T$ . Hence,  $T^* = \infty$  minimizes  $E(T)$ .
- (c) If  $y(0) < 0$ ,  $y(T) < 0$  for any  $T \in (0, T')$  and  $y(T) > 0$  for any  $T \in (T', \infty)$ , then  $T^* = T'$  where  $T' = y^{-1}(0)$ .

*Case 2:* If  $u(\infty) < C$ , then  $u(T) < C$  for any  $T$ . In this case,  $y(T)$  is a strictly decreasing function of  $T$  and

$$\frac{\partial^2 E(T)}{\partial T^2} < 0$$

There are three subcases:

- (a) If  $y(0) \leq 0$ , then  $y(T) \leq 0$  for all  $T$  and  $E(T)$  is strictly decreasing in  $T$ . Hence,  $T^* = \infty$  minimizes  $E(T)$ .
- (b) If  $y(\infty) > 0$ , then  $y(T) > 0$  for all  $T$  and  $E(T)$  is increasing in  $T$ . Hence,  $T^* = 0$  minimizes  $E(T)$ .
- (c) If  $y(0) > 0$ , and  $y(T) > 0$  for any  $T \in (0, T'']$  and  $y(T) < 0$  for any  $T \in (T'', \infty)$ , then  $T^* = 0$  if  $E(0) < E(\infty)$  and  $T^* = \infty$  if  $E(0) \geq E(\infty)$ , where  $T'' = y^{-1}(0)$ .

Case 3: See Problem 3.

*Example 10.4:* Considering a set of testing data given in Table 4.14 (data set #10), and Example 10.1, the mean value function is

$$m(T) = 142.32(1 - e^{-0.1246T})$$

Given  $C_1 = \$50$ ,  $C_2 = \$25$ ,  $C_3 = \$100$ ,  $C_4 = \$1000$ ,  $\mu_y = 0.1$ ,  $\mu_w = 0.5$ ,  $x = 0.05$ ,  $\alpha = 0.05$ , and  $T_w = 20$ . Based on Theorem 10.3, we obtain the results given in Table 10.2. The optimal release time is  $T^* = 24.5$  and the corresponding expected total cost is \$1836.15.

**Table 10.2.** Optimal release time for  $C_0 = \$100$

$T^*$ (hours)	$E(T)$ (\$)
22.5	1,843.31
23.0	1,843.31
23.5	1,839.52
24.0	1,837.17
24.5*	1,836.15
25.0	1,836.39
25.5	1,837.82
26.0	1,840.35
26.5	1,843.93

*Example 10.5:* Given  $C_1 = \$50$ ,  $C_2 = \$25$ ,  $C_3 = \$100$ ,  $C_4 = \$10,000$ ,  $\mu_y = 0.1$ ,  $\mu_w = 0.5$ ,  $x = 0.05$ ,  $\alpha = 0.05$ , and  $T_w = 20$ . Using Theorem 10.3, we obtain the results in Table 10.3. The optimal release time for this case is  $T^* = 30.5$  and the corresponding expected total cost is \$3017.13.

## 10.5 Cost Model with Multiple Failure Errors

This section describes a software cost model under the following assumptions:

- (1) The cost of debugging an error during the development phase is lower than in the operational phase.
- (2) The cost of removing a particular type of error is constant during the debugging phase.
- (3) The cost of removing a particular type of error is constant during the operational phase.
- (4) The cost of removing critical errors is more expensive than major errors, and the cost of removing major errors is more expensive than minor errors.
- (5) There is a continuous cost incurred during the entire time of the debugging period.

**Table 10.3.** Optimal release time for  $C_0 = \$1000$ 

$T^*$ (hours)	$E(T)$ (\$)
28.5	3029.72
29.0	3024.41
29.5	3020.60
30.0	3018.20
30.5*	3017.13
31.0	3017.30
31.5	3018.64
32.0	3021.09
32.5	3024.57

*Notation*

$T$	Software release time
$C_{i1}$	Cost of fixing a type $i$ error during the test phase $i = 1, 2, 3$
$C_{i2}$	Cost of fixing a type $i$ error during the operation phase ( $C_{i2} \geq C_{i1}$ , $i = 1, 2, 3$ )
$C_3$	Cost of testing per unit time
$E(T)$	Expected cost of software
$R_0$	Pre-specified software reliability
$T_r$	Debugging time required to attain minimum cost subject to a reliability constraint
$T_e$	Debugging time required to attain minimum cost subject to the number of remaining errors constraint
$T_{rel}$	Debugging time required to attain maximum reliability subject to a cost constraint

Assume that the duration of the software lifecycle is random. Let  $t$  be the random variable of the duration of the software lifecycle and  $g(t)$  the probability density function of  $t$ . Assume that the cost of testing per unit time and the cost of fixing any type  $i$  error during the test phase and the operation phase are given.

The expected software cost is defined as the cost incurred in removing and fixing errors in the software during the software lifecycle measured from the time the testing starts. Hence, the expected software cost  $E(T)$  can be formulated as (Pham 1996a)

$$\begin{aligned}
 E(T) = & \int_0^T [C_3 t + \sum_{i=1}^3 C_{i1} m_i(t)] g(t) dt \\
 & + \int_T^\infty [C_3 T + \sum_{i=1}^3 C_{i1} m_i(T)] \\
 & + \sum_{i=1}^3 C_{i2} (m_i(t) - m_i(T))] g(t) dt
 \end{aligned} \tag{10.26}$$

where  $m_i(t)$  is given (in Pham 1996a) as follows:

$$m_i(t) = \frac{ap_i}{1 - \beta_i} \left( 1 - e^{-(1 - \beta_i)b_i t} \right)$$

The function  $E(T)$  represents testing costs per unit time and of fixing errors during testing incurred if the determination of the software lifecycle is less than or equal to the software release time. On the other hand, if the determination of the software lifecycle is greater than the software release time, then an additional cost factor should be involved, *i.e.*, the cost of fixing errors during the operation phase. We next determine the value of  $T$  such that  $E(T)$  is minimized. Define

$$h(T) = \sum_{i=1}^3 (c_{i2} - c_{i1}) \lambda_i(T) \quad (10.27)$$

**Theorem 10.4:** Given  $C_3$ ,  $C_{i1}$ , and  $C_{i2}$  for  $i = 1, 2, 3$ . There exists an optimal testing time,  $T^*$ , for  $T$  that minimizes  $E(T)$ :

$$\begin{aligned} &\text{If } h(0) \leq C_3, \text{ THEN } T^* = 0 \\ &\text{ELSE } T^* = h^{-1}(C_3); \text{ ENDIF} \end{aligned}$$

**Proof:** See Problem 8.

Theorem 10.4 shows that if  $h(0) \geq C_3$ , then the testing time required to attain the minimum cost has already been achieved. Thus, the marginal cost for further testing is an increasing function, and as each additional test and debug increases the cost of the software, no further testing should be done, and the software package should be released for sale.

However, if the testing time required to attain the minimum cost has not been achieved, and the marginal cost for further debugging is a decreasing function for an interval of times, testing should be continued until time  $T^*$ , where  $T^*$  satisfies  $h(T^*) = C_3$ .

Although the optimal policies in Theorem 4 are sound in theory, it seems reasonable in practice that simply minimizing cost should not be the only goal for some applications. In the following subsection, we discuss the optimum release policies that minimize the expected software system cost subject to various constraints.

### Cost Subject to Reliability Constraint

Consider the expected software cost  $E(T)$  and the software reliability  $R(x/T)$  as the evaluation criteria. We determine the optimum release time that minimizes the expected software cost subject to attaining a desired reliability level,  $R_0$ . Then the optimization problem can be formulated as

$$\begin{aligned} &\text{Minimize} && E(T) \\ &\text{Subject to} && R(x/T) \geq R_0 \end{aligned}$$

where  $E(T)$  is given in equation (10.26). It can be proven that the software reliability  $R(x/T)$  increases as  $T$  increases. Let  $T_1$  be the solution of  $R(x/T_1) = R_0$ .

**Lemma 10.1:** Given  $C_3$ ,  $R_0$ ,  $C_{i1}$ , and  $C_{i2}$  for  $i = 1, 2, 3$ . The optimal value of  $T$ , say  $T_r$ , which minimizes  $E(T)$  subject to software reliability not less than a specified value,  $R_0$ , is determined from



IF  $h(0) \leq C_3$  THEN  
 IF  $R(x|0) \geq R_0$  THEN  $T_r = 0$   
 ELSE  $T_r = T_1$   
 ELSE IF  $R(x|T^*) \geq R_0$  THEN  $T_r = T^*$   
 ELSE  $T_r = T_1$

The physical interpretation of the result of Lemma 10.1 is as follows. If  $h(0) \leq C_3$ , then the current amount of debugging has already minimized the expected software system cost. Furthermore, if the current amount of debugging has met the reliability constraint, then no further debugging should be done, and the software should be released. Otherwise, the current amount of debugging does not meet the reliability constraint, and the debugging should be continued until time  $T_1$ , where  $T_1$  satisfies  $R(x|T_1) = R_0$ . The interpretation of the case  $h(0) > C_3$  is similar to the above.

### Cost Subject to the Number of Remaining Errors Constraint

We now present a method that will allow for the constraining of a particular type of error. The importance of this is that, though a program may be able to tolerate a large number of minor errors, it cannot tolerate critical errors. In this case, a constraint can be put on the expected number of remaining critical errors in the system before its release. It is also possible to set up constraints for each of the different types of errors independent of the other types.

Consider both the expected total software system cost,  $E(T)$ , and the expected number of failure type  $i$  errors remaining in the system,  $\bar{m}_i(T)$ , as the evaluation criteria. The optimal release problem can be formulated as

$$\begin{aligned} &\text{Minimize } E(T) \\ &\text{Subject to } \bar{m}_i(T) \leq d_i \quad i = 1, 2, 3 \end{aligned}$$

where

$$\begin{aligned} \bar{m}_i(T) &= m_i(\infty) - m_i(T) \\ &= \frac{ap_i}{1 - \beta_i} e^{-(1 - \beta_i)b_i T} \end{aligned} \quad (10.28)$$

and  $d_i$  is the accepted number of remaining type  $i$  errors. Define

$$T_{m_i} = \frac{\ln\left(\frac{ap_i}{d_i(1 - \beta_i)}\right)}{(1 - \beta_i)b_i} \quad (10.29)$$

The function  $\bar{m}_i(T)$  is, of course, decreasing in  $T$  for all  $T$ . Then  $\bar{m}_i(T) \leq d_i$  if and only if  $T \geq T_{m_i}$ .

**Lemma 10.2:** Given  $C_3$ ,  $d_i$ ,  $C_{i1}$ , and  $C_{i2}$  for  $i=1,2,3$ , then the optimal value of  $T$ , say  $T_e$ , that minimizes  $E(T)$  subject to the number of remaining errors constraint is determined from

IF  $h(0) \leq C_3$  THEN  $T_e = \max_{1 \leq i \leq 3} \{0, T_{mi}\}$   
 ELSE  $T_e = \max_{1 \leq i \leq 3} \{T^*, T_{mi}\}$ ; ENDIF

Similarly, the physical interpretation of the results of Lemma 10.2 is that if  $h(0) \leq C_3$ , then the current debugging has already minimized cost, but not all of the expected error constraints have been met. In this situation, the software program should be debugged until all of the expected error constraints have been met. However, if  $h(0) > C_3$ , then the current amount of debugging has not achieved minimum cost. In this situation, debugging should continue until all constraints have been met and minimum cost has been achieved.

### Software Reliability Subject to Cost Constraint

Consider both the software reliability  $R(x|T)$  and the expected software cost  $E(T)$  as the evaluation criteria. The optimal policies problem can be formulated as

$$\begin{cases} \text{Maximize } R(x|T) \\ \text{Subject to } E(T) \leq C_R \end{cases}$$

where  $C_R$  is the maximum amount allowable.

**Lemma 10.3:** Given  $C_3$ ,  $C_R$ ,  $C_{i1}$ , and  $C_{i2}$  for  $i = 1, 2, 3$ . The optimal value of  $T$ , say  $T_{rel}$ , that maximizes  $R(x|T)$  subject to the cost constraint is determined from

IF  $E(T^*) > C_R$  THEN there is NO solution  
 ELSE  $T_{rel} = \{T \geq T^*: T = E^{-1}(C_R)\}$ ; ENDIF

These results show that if  $E(T^*) > C_R$ , the minimum software system cost required to develop and debug the program exceeds the maximum amount allowable. Therefore, it is impossible to produce the software under these conditions. Similarly, if  $E(T^*) \leq C_R$ , and as the reliability of the software continually improves with testing and debugging time, then the program should be debugged until the cost constraint is binding, implying that additional debugging will violate the constraint.

### Applications

Using the data set #3 given in Table 4.7 and given the following reliability and error introduction rate parameters values,

$$\begin{array}{lll} p_1 = 0.0173 & p_2 = 0.3420 & p_3 = 0.6407 \\ \beta_1 = 0.5 & \beta_2 = 0.2 & \beta_3 = 0.05. \end{array}$$

Using the MLE, we obtain (Pham 1996a)

$$\begin{array}{ll} a = 428 & b_1 = 0.00024275 \\ b_2 = 0.00029322 & b_3 = 0.00030495. \end{array}$$

Given the following cost coefficients,

$$\begin{array}{lll} C_{1,1} = 200 & C_{2,1} = 80 & C_{3,1} = 30 \\ C_{1,2} = 1000 & C_{2,2} = 350 & C_{3,2} = 150 \end{array} \quad C_3 = 10$$

Assume that the mean rate of the software lifecycle length is constant. Given that the mean rate  $\mu = 0.00005$ , then

$$g(T) = \mu e^{-\mu T} \quad \text{for } T > 0$$

From equation (10.26), the expected software system cost is given by

$$\begin{aligned} E(T) = & \int_0^T [10t + 200m_1(t) + 80m_2(t) + 30m_3(t)]g(t)dt \\ & + \int_T^\infty [10T + 200m_1(T) + 80m_2(T) + 30m_3(T) \\ & + 1,000(m_1(t) - m_1(T)) + 350(m_2(t) - m_2(T)) \\ & + 150(m_3(t) - m_3(T))]g(t)dt \end{aligned}$$

Substituting  $m_i(t)$  in equation (10.26) for  $i = 1, 2, 3$  into the above equation, we obtain

$$\begin{aligned} E(T) = & \frac{1}{\mu}(1 - e^{-\mu T})C_3 \\ & - \sum_{i=1}^3 (C_{i2} - C_{i1})A_i[(1 - e^{-(1-\beta_i)b_i T})]e^{-\mu T} \\ & + \sum_{i=1}^3 C_{i1}\{A_i[(1 - e^{-\mu T}) - \frac{\mu}{B_i}(1 - e^{B_i T})]\} \\ & + \sum_{i=1}^3 C_{i2}\{A_i[e^{-\mu T} - \frac{\mu}{B_i}e^{-\beta_i T}]\} \end{aligned}$$

where

$$A_i = \frac{ap_i}{1 - \beta_i} \quad \text{and} \quad B_i = (1 - \beta_i)b_i + \mu$$

Substituting and simplifying the cost coefficient values to the above equation, we obtain

$$\begin{aligned}
E(T) = & 200,000(1 - e^{-0.00005T}) - 11,848(1 - e^{-0.0001214T})e^{-0.00005T} \\
& - 49,401.9(1 - e^{-0.0002346T})e^{-0.00005T} \\
& - 34,638(1 - e^{-0.0002897T})e^{-0.00005T} \\
& + 2,962[1 - e^{-0.00005T} - 0.2917153(1 - e^{-0.0001714T})] \\
& + 14,637.6[1 - e^{-0.00005T} - 0.1756852(1 - e^{-0.0002846T})] \\
& + 8,659.5[1 - e^{-0.00005T} - 0.1471887(1 - e^{-0.0003397T})] \\
& + 14,810[e^{-0.00005T} - 0.2917153e^{-0.0001714T}] \\
& + 64,039.5[e^{-0.00005T} - 0.1756852e^{-0.0002846T}] \\
& + 43,297.5[e^{-0.00005T} - 0.1471887e^{-0.0003397T}]
\end{aligned}$$

It is easy to obtain the optimum total testing time,  $T^*$ , that minimizes the expected total software system cost using Theorem 10.4. The results are given as below:

$$T^* = 3,366.8 \text{ hours and } E(3,366.8) = \$82,283.2.$$

If a desired level of reliability is 0.99 for a mission of 10 hours, then by using Lemma 10.1, the optimal software release time,  $T_r$ , that minimizes the expected total software system cost is easily obtained as  $T_r = 19,045$  hours.

If we assume that the remaining error constraints are

type 1 errors:  $d_1 \leq 5$

type 2 errors:  $d_2 \leq 5$

type 3 errors:  $d_3 \leq 5$

From Lemma 10.2, the optimal release time in this situation is given as follows:

$T_{m1} = 8,946$  hours

$T_{m2} = 13,912$  hours, and

$T_{m3} = 11,607$  hours.

Since  $T_{m2}$  is the maximum of the four values,  $T_{m1}$ ,  $T_{m2}$ ,  $T_{m3}$ , and  $T^*$ ,  $T_e = 13,912$  hours.

## 10.6 Gain Model with Random Field Environments

This section discusses the software gain model under random field environment with consideration of not only time to remove faults during in-house testing, cost of removing faults during beta testing, risk cost due to software failure, but also the benefits from reliable executions of the software during both beta testing and field operation.

Section 9.4 discusses a NHPP software reliability model with consideration of random field environments. This is a model which covers both beta testing and operation phases in the software systems. During beta testing, not only can software faults still be removed from the software after failures occur, but they are

also likely to be conducted in an environment that is the same as (or close to) the end-user environment.

The model in Theorem 10.3 considers both the warranty cost and the penalty cost after releasing the software, which are overlapped with each other. The model which will be discussed in this section is solely based on a recent study by Teng and Pham (2004). For the sake of simplicity, let us call it the T-P cost model. The T-P cost model is indeed slightly different to the model in Theorem 10.3. The T-P cost model does not consider a warranty cost, but instead considers a similar concept — the cost associated with the beta testing that is conducted in the field environment. The beta testing cost and the penalty cost after the software is released are not overlapped with each other.

#### Notation

$R(x T)$	Software reliability function. It is defined as the probability that a software failure does not occur in time interval $[t, t + x]$ , where $t \geq 0$ , and $x > 0$
$G(\eta)$	Cumulative distribution function of random environmental factor
$\gamma$	Shape parameter of field environmental factor (Gamma distributed variable)
$\theta$	Scale parameter of field environmental factor (Gamma distributed variable)
$N(T)$	Counting process which counts the number of software failures discovered by time $T$
$m(T)$	Expected number of software failures by time $T$ , $m(T) = E[N(T)]$
$m_1(T)$	Expected number of software failures during in-house testing by time $T$
$m_2(T)$	Expected number of software failures in beta testing and final field operation by time $T$
$m_F(t \eta)$	Expected number of software failures in field by time $t$
$C_0$	Set-up cost for software testing
$C_1$	Software in-house testing per unit time
$C_2$	Cost of removing a fault per unit time during in-house testing
$C_3$	Cost of removing a fault per unit time during beta testing
$C_4$	Penalty cost due to software failure
$C_5$	Benefits if software does not fail during beta testing
$C_6$	Benefits if software does not fail in field operation
$\mu_y$	Expected time to remove a fault during in-house testing phase
$\mu_w$	Expected time to remove a fault during beta testing phase
$a$	Number of initial software faults at the beginning of testing
$a_F$	Number of initial software faults at the beginning of the field operations

$t_0$	Time to stop testing and release the software for field operations.
$b$	Fault detection rate per fault
$T_w$	Time length of the beta testing
$x$	Time length that the software is going to be used
$p$	Probability that a fault is successfully removed from the software
$\beta$	Probability that a fault is introduced into the software during debugging, and $\beta \ll p$

Assume that the error detection rate function  $b(t)$  is a constant and from equation (9.16), the mean value function of the software system with consideration of random field environments is given by

$$m(t) = \begin{cases} \frac{a}{p-\beta} (1 - e^{-b(p-\beta)t}) & t \leq T \\ \frac{a}{p-\beta} \left( 1 - (e^{-b(p-\beta)T}) \left( \frac{\theta}{\theta + b(p-\beta)(t-T)} \right)^y \right) & t \geq T \end{cases} \quad (10.30)$$

Generally, the software reliability prediction is used after the software is released for field operations, *i.e.*,  $t \geq T$ . Therefore, the reliability of the software in the field is

$$R(x|t) = e^a e^{-b(p-\beta)TD} \quad (10.31)$$

where

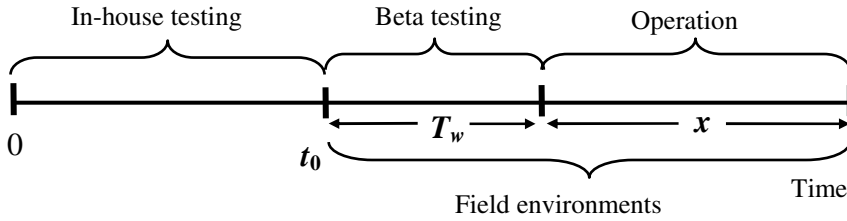
$$D = \left( \frac{\theta}{\theta + b \cdot (p-\beta) \cdot (t+x-T)} \right)^y - \left( \frac{\theta}{\theta + b(p-\beta)(t-T)} \right)^y$$

We want to determine when to stop testing the software. In other words, we only need to know the software reliability immediately after the software is released. In this case,  $t = T$ , therefore

$$R(x|T) = \exp \left( -a e^{-b(p-\beta)T} \left( 1 - \left( \frac{\theta}{\theta + b(p-\beta)x} \right)^y \right) \right) \quad (10.32)$$

### 10.6.1 Model Formulation

Figure 10.3 shows the software development process to be considered in the cost model: in-house testing, beta testing and operation, while beta testing and operation are conducted in the field environment, which is commonly quite different from the environment where the in-house testing is conducted.



**Figure 10.3.** Software gain model

Consider the following cost factors:

1. There is a constant set-up cost at the beginning of the in-house testing.
2. The cost of testing is a linear function of in-house testing time.
3. The cost to remove faults during the in-house testing period is proportional to the total time for removing all faults detected during this period.
4. The cost to remove faults during the beta testing period is proportional to the total time of removing all faults detected in  $[T, T + T_w]$ .
5. It takes time to remove faults and it is assumed that the time to remove each fault follows a truncated exponential distribution.
6. There is a penalty cost due to software failure after formal release of the software, *i.e.*, after the beta testing.
7. Software companies receive the economic profits from reliable executions of their software during beta testing and use in field environments.

The expected time to remove each fault during in-house testing  $\mu_y$  is given in equation (10.1). Similarly, the expected time to remove each fault  $\mu_w$  during beta testing is given in equation (10.23).

The expected net gain of the software development process  $E(T)$  is defined as the economical revenue in software reliability that exceeds the expected total cost of the software development (Teng and Pham 2004). In other words,

$$E(T) = \text{Expected Revenue Gain in Reliability} - (\text{Total Development Cost} + \text{Risk Cost})$$

Based on the above assumptions, the expected software system cost consists of:

### 1. Total development cost

The total development cost,  $E_C(T)$ , including

- a) A constant set-up cost  $C_0$
- b) Cost to do in-house testing,  $E_1(T)$ . We assume it is a linear function of time to do in-house testing  $T$ , then  $E_1(T) = C_1 \cdot T$

c) As in equation (10.24), the expected fault removal cost during in-house testing period,  $E_2(T)$ , is given by

$$E_2(T) = C_2 \cdot E \left[ \sum_{i=1}^{N(T)} Y_i \right] = C_2 \cdot m(T) \cdot \mu_y$$

d) The fault removal cost during beta testing period,  $E_3(T)$ , can be easily obtained as follows:

$$E_3(T) = C_3 \cdot E \left[ \sum_{i=N(T)}^{N(T+T_w)} W_i \right] = C_3 \cdot \mu_w \cdot (m(T+T_w) - m(T))$$

Therefore, the total software development cost can be written as

$$E_c(T) = C_0 + E_1(T) + E_2(T) + E_3(T)$$

## 2. Risk cost

The risk cost due to software failures after releasing the software,  $E_4(T)$ , is given by

$$E_4(T) = C_4 \cdot (1 - R(x | T + T_w))$$

## 3. Expected Revenue Gain

The expected revenue gain,  $E_p(T)$ , including

a) Benefits gained during beta testing due to reliable execution of the software,  $E_5(T)$ , is

$$E_5(T) = C_5 \cdot R(T_w | T)$$

b) Benefits gained during field operation due to reliable execution of the software,  $E_6(T)$ , is

$$E_6(T) = C_6 \cdot R(x | T + T_w)$$

Therefore, the expected gain of the software development process,  $E(T)$ , can be expressed as

$$\begin{aligned} E(T) &= E_p(T) - (E_c(T) + E_4(T)) \\ &= (E_5(T) + E_6(T)) - (C_0 + E_1(T) + E_2(T) + E_3(T) + E_4(T)) \\ &= C_5 \cdot R(T_w | T) + C_6 \cdot R(x | T + T_w) - C_0 - C_1 \cdot T - C_2 \cdot \mu_y \cdot m(T) \\ &\quad - C_3 \cdot \mu_w \cdot (m(T + T_w) - m(T)) - C_4 \cdot (1 - R(x | T + T_w)) \end{aligned}$$

or, equivalently, as

$$\begin{aligned} E(T) &= C_5 \cdot R(T_w | T) + (C_4 + C_6) \cdot R(x | T + T_w) - (C_0 + C_4) \\ &\quad - C_1 \cdot T - C_2 \cdot m_1(T) \cdot \mu_y - C_3 \cdot \mu_w \cdot (m_2(T + T_w) - m_2(T)) \end{aligned} \quad (10.33)$$

where

$$m_1(t) = \frac{a}{p - \beta} (1 - e^{-b(p - \beta)t}) \quad t \leq T$$



$$m_2(t) = \frac{a}{p-\beta} \left( 1 - e^{-b(p-\beta)T} \left( \frac{\theta}{\theta + b(p-\beta)(t-T)} \right)^\gamma \right) \quad t \geq T$$

Next we will obtain the optimal software release time,  $T^*$  which maximizes the expected net gain of software systems. Let

$$\begin{aligned} y(T) = & -C_1 - C_2 \cdot \mu_y \cdot a \cdot b \cdot e^{-b(p-\beta)T} + \\ & \mu_w \cdot C_3 \cdot a \cdot b \cdot e^{-b(p-\beta)T} \left( 1 - \left( \frac{\theta}{\theta + b \cdot (p-\beta) \cdot T_w} \right)^\gamma \right) + \\ & C_5 \cdot a \cdot b \cdot e^{-b(p-\beta)T} \cdot R(T_w | T) \cdot \left( 1 - \left( \frac{\theta}{\theta + b \cdot (p-\beta) \cdot T_w} \right)^\gamma \right) \\ & + (C_4 + C_6) \cdot a \cdot b \cdot e^{-b(p-\beta)T} \cdot R(x | T + T_w) \cdot \\ & \left( \left( \frac{\theta}{\theta + b \cdot (p-\beta) \cdot T_w} \right)^\gamma - \left( \frac{\theta}{\theta + b \cdot (p-\beta) \cdot (T_w + x)} \right)^\gamma \right) \end{aligned} \quad (10.34)$$

$$u(T) = -(C_4 + C_6)(p-\beta)ab^2 R(x | T + T_w) \times \quad (10.35)$$

$$\begin{aligned} & \left( \left( \frac{\theta}{\theta + b(p-\beta)T_w} \right)^\gamma - \left( \frac{\theta}{\theta + b(p-\beta)(T_w + x)} \right)^\gamma \right) \times \\ & \left[ 1 - \frac{a}{(p-\beta)} e^{-b(p-\beta)T} \left( \left( \frac{\theta}{\theta + b(p-\beta)T_w} \right)^\gamma - \left( \frac{\theta}{\theta + b(p-\beta)(T_w + x)} \right)^\gamma \right) \right] \\ & - C_5(p-\beta)ab^2 R(T_w | T) \times \left( 1 - \left( \frac{\theta}{\theta + b(p-\beta)T_w} \right)^\gamma \right) \times \\ & \left[ 1 - \frac{a}{(p-\beta)} e^{-b(p-\beta)T} \left( 1 - \left( \frac{\theta}{\theta + b(p-\beta)T_w} \right)^\gamma \right) \right] \\ C = & \mu_w C_3 (p-\beta)ab^2 \left( 1 - \left( \frac{\theta}{\theta + b(p-\beta)T_w} \right)^\gamma \right) \\ & - C_2 \mu_y (p-\beta)ab^2 \end{aligned} \quad (10.36)$$

It can be shown that the function  $u(T)$  decreases as  $T$  increases.

**Theorem 10.5 (Teng and Pham 2004):** Given  $C_0, C_1, C_2, C_3, C_4, C_5, C_6, x, \mu_y, \mu_w, T_w$ , the optimal value of  $T$ , say  $T^*$ , which maximizes the expected net gain of the software development process  $E(T)$ , is as follows

1. If  $u(0) \leq C$  and
  - (a) If  $y(0) \leq 0$ , then  $T^* = 0$ ;
  - (b) If  $y(\infty) > 0$ , then  $T^* = \infty$ ;
  - (c) If  $y(0) > 0$ ,  $y(T) \geq 0$  for  $T \in (0, T']$  and  $y(T) < 0$  for  $T \in (T', \infty]$ , then  

$$T^* = T' \text{ where } T' = y^{-1}(0)$$
  
2. If  $u(\infty) > C$  and
  - (a) If  $y(0) \geq 0$ , then  $T^* = \infty$ ;
  - (b) If  $y(\infty) < 0$ , then  $T^* = 0$ ;
  - (c) If  $y(0) < 0$ ,  $y(T) \leq 0$  for  $T \in (0, T'']$  and  $y(T) > 0$  for  $T \in (T'', \infty]$ , then:  

$$T^* = \infty \text{ if } E(0) < E(\infty)$$

$$T^* = 0 \text{ if } E(0) \geq E(\infty)$$

$$\text{where } T'' = y^{-1}(0)$$
  
3. If  $u(0) > C$ ,  $u(T) \geq C$  for  $T \in (0, T^0]$  and  $u(T) < C$  for  $T \in (T^0, \infty]$  where  

$$T^0 = u^{-1}(C)$$
, then:
  - (a) If  $y(0) < 0$ , then  

$$\text{If } y(T^0) \leq 0, \text{ then } T^* = 0$$

$$\text{If } y(T^0) > 0, \text{ then}$$

$$T^* = 0 \text{ if } E(0) \geq E(T_b)$$

$$T^* = T_b \text{ if } E(0) < E(T_b)$$

$$\text{where } T_b = y^{-1}(0) \text{ and } T_b \geq T^0$$
  - (b) If  $y(0) \geq 0$ , then  $T^* = T_c$ , where  $T_c = y^{-1}(0)$

**Proof:** Problem 9.

### 10.6.2 Applications

The data in Tables 10.4 and 10.5 show, respectively, the normalized time and cumulative failures during in-house testing and field operation per system day of a software telecommunication product (Teng and Pham 2004).

In Table 10.4, the last normalized time point  $t_0 = 0.0184$  is the actual stopping in-house testing time for this real application. After  $t_0$ , the software is released for the field operation. Table 10.5 shows the normalized field data for this software application.

The parameters of the NHPP model in equation (10.30) based on the MLE method with  $p = 1$  is given by

$$\hat{a} = 207.97, \hat{b} = 0.00179, \hat{\gamma} = 0.5194, \hat{\theta} = 5.6244, \hat{\beta} = 0$$

**Table 10.4.** Normalized cumulative failures during in-house testing period

Time	Failures	Time	Failures
0.00010	0.02488	0.00704	0.52239
0.00017	0.02985	0.00768	0.54726
0.00024	0.06468	0.00858	0.58209
0.00033	0.06468	0.00954	0.61194
0.00051	0.10945	0.01048	0.63682
0.00061	0.11940	0.01138	0.64677
0.00079	0.14428	0.01207	0.67662
0.00116	0.16915	0.01285	0.70149
0.00162	0.19900	0.01347	0.73632
0.00226	0.22886	0.01424	0.77612
0.00281	0.26368	0.01467	0.77612
0.00327	0.31343	0.01553	0.81592
0.00382	0.34826	0.01639	0.82587
0.00437	0.35323	0.01717	0.84079
0.00483	0.36816	0.01755	0.84577
0.00529	0.38806	0.01796	0.87562
0.00584	0.44776	0.01836	0.89552
0.00640	0.48756	0.01840	0.90050

Please note that the MLE results listed above are actual results, not the normalized results, since they can be more helpful to understand the software development process. Therefore, the mean value function becomes

$$m(t) = \begin{cases} m_1(t) = 207.97(1 - e^{-0.00179t}) & t \leq T \\ m_2(t) = 207.97 \left( 1 - e^{-0.00179T} \left( \frac{\theta}{\theta + 0.00179 \cdot (t - T)} \right)^\gamma \right) & t \geq T \end{cases}$$

Figure 10.4 shows the mean value function and normalized cumulative software failures fitting curve during in-house testing and field operation.

The cost coefficients can be commonly determined by empirical data or by previous experiences. In this example, we use the following cost coefficients and parameter values:

$$C_0 = 50, \quad C_1 = 100, \quad C_2 = 60, \quad C_3 = 3600, \quad C_4 = 300000, \quad C_5 = 500000, \\ C_6 = 100000, \quad \mu_y = 0.1, \quad \mu_w = 0.5, \quad T_w = 1000, \quad x = 4000.$$

From Theorem 10.5, we can obtain the (normalized) optimum release time and the corresponding maximum expected net gain of the software development as follows:

$$T^* = 0.0663 \quad E(T^*) = 179,012.2$$

Compared with the actual software stopping testing time  $t_0 = 0.0184$  and based on the the optimal release time  $T^* = 0.066291$ , this shows that one need to continue to do in-house testing after  $t_0$  until  $T^*$ .

**Table 10.5.** Normalized cumulative failures and times during field operation

Time	Failures	Time	Failures
0.04310	0.90547	0.50111	0.98507
0.06162	0.91045	0.53383	0.98507
0.08015	0.92040	0.57309	0.98507
0.08632	0.92537	0.62580	0.99005
0.13573	0.93035	0.66560	0.99005
0.14190	0.93532	0.67887	0.99005
0.16660	0.94527	0.72531	0.99005
0.20983	0.94527	0.75185	0.99005
0.22229	0.95025	0.75849	0.99005
0.2534	0.95025	0.77176	0.99005
0.25967	0.95025	0.79829	0.99005
0.26590	0.95025	0.82511	0.99005
0.27213	0.95522	0.84529	0.99005
0.29705	0.96020	0.85201	0.99005
0.30328	0.97015	0.90583	0.99005
0.31575	0.97512	0.91255	0.99005
0.34067	0.97512	0.91928	0.99005
0.34690	0.97512	0.93946	0.99502
0.39674	0.97512	0.94619	0.99502
0.40297	0.98010	0.95291	1
0.42914	0.98507	0.98655	1
0.435684	0.98507	1	1
0.474941	0.98507		

Figure 10.4 seems to indicate that this specific software application appears to be more reliable in field than in testing environments. Figure 10.5 shows the expected net gain function  $E(T)$  curve. Figure 10.6 shows the software reliability growth curve  $R(x|T+T_w)$  for various value of  $T$  where  $T_w$  and  $x$  are fixed.

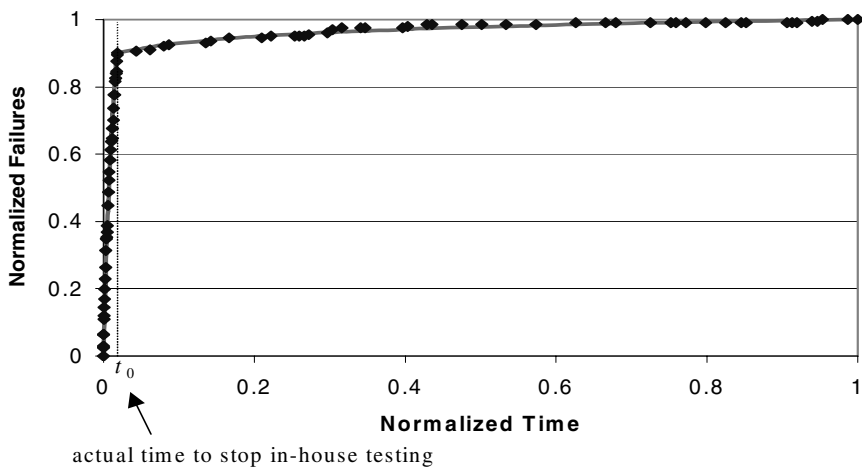


Figure 10.4. Failures and mean value function fitting curve

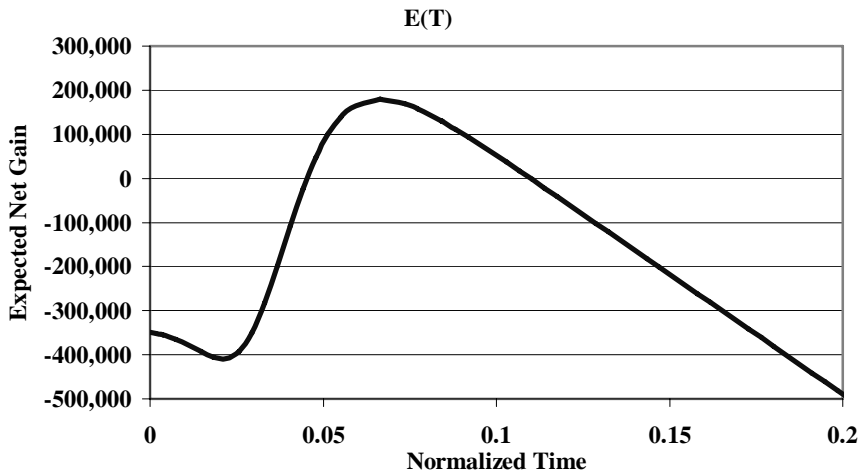
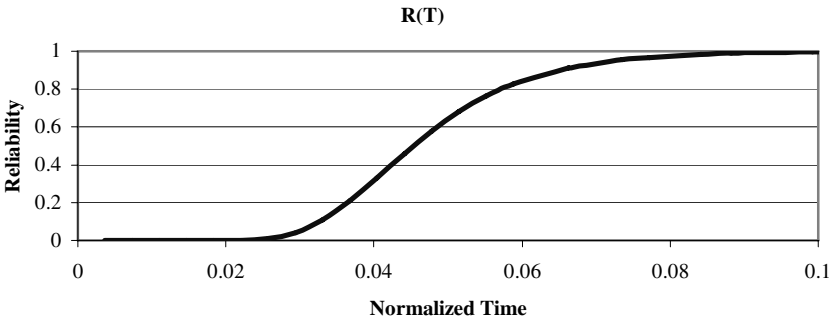


Figure 10.5. Expected net gain  $E(T)$



**Figure 10.6.** Software reliability growth curve  $R(x | T + T_w)$

**10.7 Other Cost Models**

Pham and Wang (2001a) presented a software cost model based on the quasi renewal processes. If the inter-arrival time represents the error-free time (time between errors), a quasi renewal process can be used to model reliability growth for software. For example, suppose that all faults of the software have the same chance of being detected. If the inter-arrival times of a quasi renewal process represent the error-free times of the software, the expected cumulative number of software faults in  $[0, t)$  can be described by the renewal function  $M(t)$  with parameter  $\lambda$ . Let  $\bar{M}(t)$  be the number of remaining software faults at time  $t$ . It follows that  $\bar{M}(t) = M(\tau) - M(t)$  where  $M(\tau)$  is the number of faults which can be detected through a long testing time  $\tau$ , relative to  $t$ .

*Notation*

- $c_1$       The cost of fixing a fault during testing phase
- $c_2$       The cost of fixing a fault during operation phase
- $c_3$       The cost of testing per unit time
- $T$         The software release time
- $T_d$       The scheduled delivery time
- $g(t)$     The probability density function of the life-cycle length ( $t > 0$ )
- $c_p(t)$    A penalty cost for a delay of delivering software
- $M(t)$     Number of faults which can be detected through a period of testing time  $t$ .

The expected total software life-cycle cost can be defined as follows:

$$C(T) = c_3 T + c_1 M(T) + \int_T^{\infty} c_2 [M(t) - M(T)] g(t) dt + I(T - T_d) c_p (T - T_d)$$

where  $I(t)$  is an indicator function, that is,

$$I(t) = \begin{cases} 1 & \text{if } t \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$M(t)$  and  $\text{Var}[N(t)]$  contains some unknown parameters. Those unknown parameters can be obtained by using the MLE (discussed in Chapter 5) or least squares methods (in Chapter 2). Detailed optimal release policies and findings can be found in (Pham 2001a).

## 10.8 Further Reading

Some interesting papers on cost models and books are:

H. Pham, "Software reliability and cost models: Perspectives, comparison and practices," *European Journal of Operational Research*, vol 149, 2003

B.W. Boehm, C. Abts, A.W. Brown, S. Chulani, et al., *Software Cost Estimation with COCOMO II*, 2000, Prentice-Hall

H. Pham (Ed.), *Handbook of Reliability Engineering*, Springer 2003

## 10.9 Problems

1. Show that the function  $g(T)$  in equation (10.9) is increasing in  $T$ .
2. Show that the function  $u(T)$  in equation (10.25) is increasing in  $T$ .
3. Complete the proof of Case 3 in Theorem 10.3.
4. Prove Theorem 10.2.
5. Assume that the risk cost due to software failure after releasing the software beyond the warranty period,  $E_4(T)$ , is given by

$$E_4(T) = C_4 [1 - R(x | T + T_w)]$$

From equation (10.24), the expected total software cost  $E(T)$  can be modified as follows:

$$\begin{aligned} E(T) = & C_0 + C_1 T^\alpha + C_2 m(T) \mu_y \\ & + C_3 \mu_w [m(T + T_w) - m(T)] \\ & + C_4 [1 - R(x | T + T_w)] \end{aligned}$$

where

$$R(x | (T + T_w)) = e^{-ae^{-b(T+T_w)}[1-e^{-bx}]}$$

Given  $C_0, C_1, C_2, C_3, C_4, x, \mu_y, \mu_w, T$ , show that the optimal value of  $T$ , say  $T^*$ , which minimizes the expected total cost of the software, is given as below:

1. If  $v(0) \geq C$ , and
  - (a) If  $y(0) \geq 0$ , then  $T^* = 0$ ;
  - (b) If  $y(\infty) < 0$ , then  $T^* = \infty$ ;
  - (c) If  $y(0) < 0, y(T) < 0$  for any  $T \in (0, T')$  and  $y(T) > 0$  for any  $T \in (T', \infty)$ , then  $T^* = T'$ , where  $T' = y^{-1}(0)$
2. If  $v(\infty) < C$ , and
  - (a) If  $y(0) \leq 0$ , then  $T^* = \infty$ ;
  - (b) If  $y(\infty) > 0$ , then  $T^* = 0$ ;
  - (c) If  $y(0) > 0, y(T) > 0$  for any  $T \in (0, T'')$  and  $y(T) < 0$  for any  $T \in (T'', \infty)$ , then
 
$$\begin{aligned} T^* &= 0 \text{ if } E(0) \leq E(\infty) \\ T^* &= \infty \text{ if } E(0) > E(\infty) \\ \text{where } T'' &= \inf\{T: y(T) < 0\}. \end{aligned}$$
3. If  $v(0) < C, v(T) \leq C$  for  $T \in (0, T_0]$  and  $v(T) > C$  for  $T \in (T_0, \infty)$ , where  $T_0 = \{T: T = v^{-1}(C)\}$  then
  - (a) If  $y(0) \geq 0$ , then
 
$$\begin{aligned} T^* &= 0 \text{ if } E(0) \leq E(T_b) \\ T^* &= T_b \text{ if } E(0) > E(T_b) \\ \text{where } T_b &= \inf\{T > T_0: T = y^{-1}(0)\} \end{aligned}$$
  - (b) If  $y(0) < 0$ , then  $T^* = T_b'$  minimizes  $E(T)$  where
 
$$T_b' = \inf\{T: T = y^{-1}(0)\}$$

$$\begin{aligned} y(T) = & \alpha C_1 T^{(\alpha-1)} - \mu_w C_3 a b e^{-bT} (1 - e^{-bT}) \\ & - a b e^{-bT} [C_4 (1 - e^{-bx}) e^{-b(T+T_w)} R(x | T + T_w) - C_2 \mu_y] \end{aligned}$$

$$\begin{aligned} v(T) = & a b^2 C_4 (1 - e^{-bx}) e^{-bT_w} R(x | (T + T_w)) [1 - a e^{-bT} (1 - e^{-bx})] \\ & + \alpha(\alpha-1) C_1 T^{(\alpha-2)} e^{bT} \end{aligned}$$

$$C = C_2 \mu_y a b^2 - \mu_w C_3 a b^2 (1 - e^{-bT_w})$$

6. Given  $C_0 = \$100, C_1 = \$50, C_2 = \$25, C_3 = \$100, C_4 = \$1,000, \mu_y = 0.1, \mu_w = 0.5, x = 0.05$ , and  $T_w = 20, \alpha = 0.95$ , using the results in Problem 5, show that the



release times and the corresponding expected total software cost are given in Table A below.

**Table A.** Optimal release time

Release time $T^*$ (hours)	Expected total cost $E(T)$ (\$)
21.5	1,806.69
22.0	1,801.17
22.5	1,797.18
23.0	1,794.65
23.5*	1,793.47
24.0	1,793.56
24.5	1,794.85
25.0	1,797.27
25.5	1,800.74

7. Given  $C_0 = \$100$ ,  $C_1 = \$10$ ,  $C_2 = \$5$ ,  $C_3 = \$100$ ,  $C_4 = \$1,000$ ,  $\mu_y = 0.1$ ,  $\mu_w = 0.5$ ,  $x = 0.05$ , and  $T_w = 20$ ,  $\alpha = 0.95$ , using the results in Problem 5, show that the optimal release time and the corresponding expected total software cost are 37 and \$545.2, respectively.

8. Prove Theorem 10.4

9. Prove Theorem 10.5

## **Complex Fault-tolerant System Reliability Modeling**

### **11.1 Introduction**

Computer systems are now applied to many important areas such as defense, transportation, and air traffic control. Therefore, fault-tolerance has become one of the major concerns of computer designers. Fault-tolerant computer systems are defined as systems capable of recovery from hardware or software failure to provide uninterrupted real-time service.

It is important to provide very high reliability to critical applications such as aircraft controller and nuclear reactor controller software systems. No matter how thorough the testing, debugging, modularization, and verification of software are, design bugs still plague the software. After reaching a certain level of software refinement, any effort to increase the reliability, even by a small margin, will increase exponential cost. Consider, for example, fairly reliable software subjected to continuous testing and debugging, and guaranteed to have no more than 10 faults throughout the lifecycle. In order to improve the reliability such that, for example, only seven faults may be tolerated, the effort and cost to guarantee this may be enormous. A way of handling unpredictable software failure is through fault-tolerance. Over the last three decades, there has been considerable research in the area of fault-tolerant software.

Fault-tolerant software has been considered for use in a number of critical areas of application. For example, in traffic control systems, the Computer Aided Traffic Control System (COMTRAC) (Lala 1985) is a fault-tolerant computer system designed to control the Japanese railways. It consists of three symmetrically interconnected computers. Two computers are synchronized at the program task level while the third acts as an active-standby. Each computer can be in one of the following states: on-line control, standby, or offline. The COMTRAC software has a symmetric configuration. The configuration system contains the configuration control program and the dual monitor system contains the state control program. When one of the computers under dual operation has a fault, the state control program switches the system to single operation and reports the completion of the system switching to the configuration control program. The configuration control program commands the state control program to switchover to dual operation with

the standby computer. The latter program then executes the system switchover, transferring control to the configuration control program, which judges the accuracy of the report and indicates its own state to the other computers. The COMTRAC shows that it failed seven times during a three-year period - once due to hardware failure, five times due to software failure, and once for unknown causes.

Another example is the NASA space shuttle. The shuttle carries a configuration of four identical flight computers, each loaded with the same software, and a fifth computer developed by a different manufacturer and running dissimilar (but functionally equivalent) software. This software is executed only if the ones in the other four processors cannot reach consensus during critical phases of the flight (Spector 1984).

Software fault-tolerance is achieved through special programming techniques that enable the software to detect and recover from failure incidents. The method requires redundant software elements that provide alternative means of fulfilling the same specifications. The different versions must be such that they will not all fail in response to the same circumstances. Many researchers have investigated and suggested that diverse software versions developed using different specifications, designs, programming teams, programming languages, *etc.*, might fail in a statistically independent manner. Empirical evidence questions that hypothesis (Leveson 1990). On the other hand, almost all software fault-tolerance experiments have reported some degree of reliability improvement (Avizienis 1988).

Software fault-tolerance is the reliance on design redundancy to mask residual design faults present in software programs (Pham 1992a). Fault-tolerance, however, incurs costs due to the redundancy in hardware and software resources required to provide backup for system components. We must weigh the cost of fault-tolerance against the cost of software failure. With the current growth of software system complexity, we cannot afford to postpone the implementation of fault-tolerance in critical areas of software application.

This chapter discusses a basic concept for fault-tolerant software techniques and some advanced techniques including self-checking systems. We then give the reliability analysis of fault-tolerant software schemes such as recovery block (RB), N-version programming (NVP), and hybrid fault-tolerant systems. Basically, in the last system, an RB can be embedded within an NVP by applying the RB approach to each version of the NVP. Similarly, an NVP can be nested within an RB.

This chapter describes a software reliability growth model for triple-version programming (TVP) systems based on the NHPP. This chapter also discusses a reliability study in modeling the interactions between the hardware and the software component.

## 11.2 Basic Fault-tolerant Software Techniques

The study of software fault-tolerance is relatively new as compared with the study of fault-tolerant hardware. In general, fault-tolerant approaches can be classified into fault-removal and fault-masking approaches. Fault-removal techniques can be either forward error recovery or backward error recovery.

Forward error recovery aims to identify the error and, based on this knowledge, correct the system state containing the error. Exception handling in high-level languages, such as Ada and PL/1, provides a system structure that supports forward recovery. Backward error recovery corrects the system state by restoring the system to a state which occurred prior to the manifestation of the fault. The recovery block scheme provides such a system structure. Another fault-tolerant software technique commonly used is error masking. The NVP scheme uses several independently developed versions of an algorithm. A final voting system is applied to the results of these  $N$ -versions and a correct result is generated.

A fundamental way of improving the reliability of software systems depends on the principle of design diversity where different versions of the functions are implemented. In order to prevent software failure caused by unpredicted conditions, different programs (alternative programs) are developed separately, preferably based on different programming logic, algorithm, computer language, *etc.* This diversity is normally applied under the form of recovery blocks or  $N$ -version programming.

Fault-tolerant software assures system reliability by using protective redundancy at the software level. There are two basic techniques for obtaining fault-tolerant software:

- RB scheme
- NVP

Both schemes are based on software redundancy assuming that the events of coincidental software failures are rare.

### 11.2.1 Recovery Block Scheme

The recovery block scheme, proposed by Randell (1975), consists of three elements: primary module, acceptance tests, and alternate modules for a given task. The simplest scheme of the recovery block is as follows:

```

Ensure  $T$ 
By  $P$ 
Else by  $Q_1$ 
  Else by  $Q_2$ 
    .
    .
    .
  Else by  $Q_{n-1}$ 
Else Error
```

where  $T$  is an acceptance test condition that is expected to be met by successful execution of either the primary module  $P$  or the alternate modules  $Q_1, Q_2, \dots, Q_{n-1}$ . The process begins when the output of the primary module is tested for acceptability. If the acceptance test determines that the output of the primary module is not acceptable, it recovers or rolls back the state of the system before the primary module is executed. It allows the second module  $Q_1$ , to execute. The acceptance test is repeated to check the successful execution of module  $Q_1$ . If it fails, then module  $Q_2$  is executed, *etc.* The alternate modules are identified by the keywords "else by" When all alternate modules are exhausted, the recovery block

itself is considered to have failed and the final keywords "else error" declares the fact. In other words, when all modules execute and none produce acceptable outputs, then the system fails.

A reliability optimization model has been studied by Pham (1989b) to determine the optimal number of modules in a recovery block scheme that minimizes the total system cost given the reliability of the individual modules. Details of the model can be obtained in Pham (1989b).

In a recovery block, a programming function is realized by  $n$  alternative programs,  $P_1, P_2, \dots, P_n$ . The computational result generated by each alternative program is checked by an acceptance test,  $T$ . If the result is rejected, another alternative program is then executed. The program will be repeated until an acceptable result is generated by one of the  $n$  alternatives or until all the alternative programs fail.

The probability of failure of the RB scheme,  $P_{rb}$ , is as follows:

$$P_{rb} = \prod_{i=1}^n (e_i + t_{2i}) + \sum_{i=1}^n t_{1i} e_i \left( \prod_{j=1}^{i-1} (e_j + t_{2j}) \right) \quad (11.1)$$

where

$e_i$  = probability of failure for version  $P_i$

$t_{1i}$  = probability that acceptance test  $i$  judges an incorrect result as correct

$t_{2i}$  = probability that acceptance test  $i$  judges a correct result as incorrect.

The first term of equation (11.1) corresponds to the case when all versions fail the acceptance test. The second term corresponds to the probability that acceptance test  $i$  judges an incorrect result as correct at the  $i$ th trial of the  $n$  versions.

### 11.2.2 N-version Programming

The NVP was proposed by Chen and Avizienis (1978) for providing fault-tolerance in software. In concept, the NVP scheme is similar to the  $N$ -modular redundancy scheme used to provide tolerance against hardware faults (Lala 1985).

The NVP is defined as the independent generation of  $N \geq 2$  functionally equivalent programs, called *versions*, from the same initial specification. *Independent generation of programs* means that the programming efforts are carried out by  $N$  individuals or groups that do not interact with respect to the programming process. Whenever possible, different algorithms, techniques, programming languages, environments, and tools are used in each effort. In this technique,  $N$  program versions are executed in parallel on identical input and the results are obtained by voting on the outputs from the individual programs. The advantage of NVP is that when a version failure occurs, no additional time is required for reconfiguring the system and redoing the computation.

Consider an NVP scheme consists of  $n$  programs and a voting mechanism,  $V$ . As opposed to the RB approach, all  $n$  alternative programs are usually executed simultaneously and their results are sent to a decision mechanism which selects the final result. The decision mechanism is normally a voter when there are more than two versions (or, more than  $k$  versions, in general), and it is a comparator when

there are only two versions ( $k$  versions). The syntactic structure of NVP is as follows:

$seq$   
 $par$   
 $P_1$  (version 1)  
 $P_2$  (version 2)  
 $\vdots$   
 $\vdots$   
 $\vdots$   
 $P_n$  (version  $n$ )  
 decision  $V$

Assume that a correct result is expected where there are at least two correct results. The probability of failure of the NVP scheme,  $P_n$ , can be expressed as

$$P_{nv} = \prod_{i=1}^n e_i + \prod_{i=1}^n (1-e_i) e_i^{-1} \prod_{j=1}^n e_j + d \quad (11.2)$$

The first term of equation (11.2) is the probability that all versions fail. The second term is the probability that only one version is correct. The third term,  $d$ , is the probability that there are at least two correct results but the decision algorithm fails to deliver the correct result.

Eckhardt and Lee (1985) also developed a statistical model of NVP. In their model, "independently developed versions" are modeled as programs randomly selected from the input space of possible program versions that support problem-solving. Assume that the aggregate fails whenever at least  $m$  versions fail. Let  $q(x)$  be the proportion of versions failing when executing on input state  $x$ . Let  $Q(A)$  be the usage distribution, the probability that the subset of input state  $A$  is selected. Then the reliability of the NVP aggregate is given as

$$R = 1 - \sum_{i=m}^N \binom{N}{i} [q(x)]^i [1-q(x)]^{N-i} dQ(A)$$

where  $m = \lfloor 1(N+1)/2 \rfloor$  a majority of the  $N$  versions. Eckhardt and Lee also noted that independently developed versions do not necessarily fail independently.

It is worthwhile to note that the goal of the NVP approach is to ensure that multiple versions will be unlikely to fail on the same inputs. With each version independently developed by a different programming team, design approach, *etc.*, the goal is that the versions will be different enough in order that they will not fail too often on the same inputs. However, multiversion programming is still a controversial topic.

The main difference between the recovery block scheme and the  $N$ -version programming is that the modules are executed sequentially in the former. The recovery block generally is not applicable to critical systems where real-time response is of great concern. The  $N$ -version programming and the recovery block techniques have been discussed in detail by Anderson and Lee (1980).

## 11.3 Other Advanced Techniques

*N*-version programming has been researched thoroughly during the past decade. Correlated errors form a main source of failure of the *N*-version programs (Nicola 1990) and can be minimized by design diversity. A design paradigm has been developed to assure design diversity in *N*-version software. Several experiments (see Lyu 1991; Leveson 1990) have been conducted to validate the assumption of error independence in multiple versions, to analyze the types of faults, to investigate the use of self-checks and voting in error detection, and to establish the need for a complete and unambiguous specification. There has been some effort on modeling the reliability of such fault-tolerant software approaches (Arlat 1990; Belli 1990; Tso 1986; Vouk 1990). Pham (1995) has given a cost model to obtain the optimal number of program versions that minimizes the expected cost of the NVP scheme.

However, in critical systems with real-time deadlines, voting at the end of the program, as in the basic *N*-version programming, may not be acceptable. Therefore, voting at intermediate points is called for. Such a scheme, where the comparison of results is done at intermediate points, is called the community error recovery (CER) scheme (Nicola 1990) and is shown to offer a higher degree of fault-tolerance compared to the basic *N*-version programming. This approach, however, requires the synchronization of the various versions of the software at the comparison points.

Another scheme which adopts intermediate voting is the *N*-program, self-checking scheme (Yau 1975) where each version is subject to an acceptance test or checking by comparison. When  $N = 2$ , it is a two-version, self-checking scheme or self-checking duplex scheme. Whenever a particular version raises an exception, the correct result is obtained from the remaining versions and execution is continued. This method is similar to the CER approach, with the only difference being the on-line detection in the former by an acceptance test rather than a comparison.

### 11.3.1 Self-checking Duplex Scheme

In this section we discuss an approach, called a self-checking duplex scheme, for the enhancement of software reliability. This scheme incorporates redundancy at two levels and can increase the reliability of software in critical systems significantly.

If individual versions are made highly reliable, an ultra-high reliability can be achieved merely by having two versions. These two versions should be made self-checking and work simultaneously as a duplex system as shown in Pham (2000a) (Figure 7.1). In Pham (2000a) (Figure 7.1) for example, if module  $i$  raises an exception, correct results can be obtained from the other version. This approach is known as a self-checking duplex system, illustrating a simple architecture of the system scheme where both versions are represented by a sequence of  $N$  modules.

Although  $N$  self-checking versions can be used, Pham (1991a) reported a preliminary study that two versions are sufficient to raise the reliability to acceptable levels using our new approach. It is also more practical to have as few

self-checking versions as possible because of the high cost of developing  $N$  different self-checking versions.

Self-checking software can be developed in various ways. Self-checking provides an on-line detection of errors and prevents the contamination of the software by not letting the errors manifest. Software integrity can be assured by testing for illegal branching, infinite loops, wrong branching, *etc.*, and testing for functionality and validity of the results. It is easier to incorporate self-checking assertions into the software during the design stage since the team that develops the software is expected to have the best understanding of the problem. A good understanding of the application and the algorithms is deemed important for creating and placing meaningful assertions in the code. Both local and global self-checking assertions need to be incorporated to guarantee a high reliability. Hua and Abraham (1986) provide a systematic method for developing the self-checking assertions. In the following paragraphs, we show how self-checking assertions can provide ultra-high reliability.

Let the executable assertions be inserted in a module both locally and globally. By inserting local and global assertions, it is possible to check not only the internal states of the modules, but also the input/output specifications. As the inputs to an intermediate module such as  $i+1$  (see Figure 7.1 Pham 2000a) are reset to the correct value by the corresponding module of the other version if and only if an error is detected, any undetected error at module  $i$  will propagate to the next module  $i + 1$ . Let  $p$  represent the probability of detecting an error in module  $i$  of a self-checking version. Now, given that an error goes undetected at the  $i$ th,  $(i + 1)$ th, ... ,  $(i + k - 1)$ th module, the probability of this error being detected at the  $i + 1$ th module of the version is

$$p_k = p \sum_{j=i}^{i+k} (1-p)^{j-i} \quad (11.3)$$

*Example 11.1:* Suppose that the probability of detecting a design error at a particular module by self-checking is 0.9. If an error is not detected at this module, the probability of this error being detected at the following module, using equation (11.1), is 0.99 and the probability of detecting it at the next module is 0.999 and so on. This establishes that self-checking assertions could be a very powerful tool in increasing software reliability.

### 11.3.2 Hybrid Fault-tolerant Scheme

A hybrid fault-tolerant system is defined as a software system which combines the RB and NVP schemes in order to improve the reliability of software systems. For simplicity, we only discuss two level hybrid systems. We use the notation  $(NVP_n, RB_m)$  to represent an  $n$ -version NVP with each version being an  $m$ -version RB. Similarly,  $(RB_n, NVP_m)$  represents an  $n$ -version RB with each version being an  $m$ -version NVR. For example, two of the possible combinations of recovery block and NVP using four versions  $P_1, P_2, P_3$ , and  $P_4$ , are shown in Figures 11.1 and 11.2.



```

/* NVP */
seq
par
/* RB */
ensure T
by P1 (version 1)
    else by P2 (version 2)
    else error;
/* RB */
ensure T
by P3 (version 3)
    else by P4 (version 4)
else error;

```

**Figure 11.1.** (NVP<sub>2</sub>,RB<sub>2</sub>) configuration.

```

/* RB */
ensure T
by /* NVP */
seq
    par
        P1 (version 1);
        P2 (version 2);
    decision V
    else by /* NVP */
    seq
        par
            P3 (version 3);
            P4 (version 4);
        decision V

```

**Figure 11.2.** (RB<sub>2</sub>,NVP<sub>2</sub>) configuration.

Without loss of generality, we discuss here the hybrid fault-tolerant schemes (see Figure 7.3 in Pham 2000a) with only two levels: RB embedded in NVP or NVP embedded in RB. Figure 7.3 in Pham (2000a) shows the basic structure of a two-level hybrid fault-tolerant scheme. The first level consists of  $P_i$  basic program versions which form the second level composite program modules  $M_j$  where  $1 \leq i \leq n$  and  $1 \leq j \leq m$ . If RB (or NVP) is used at the first level, NVP (or RB) is used at the second level. The composite version failure rates of the program version are  $e_i$ , acceptance test error probabilities are  $t_1$ , and  $t_2$ , and the decision error probability is  $d$ . The hybrid fault-tolerant scheme's reliability can be obtained by calculating the reliability of the lower level program versions or composite versions, and then using the lower level reliabilities as inputs to the higher level composite versions. This process is repeated until the total system reliability is obtained. Mathemati-

cally, the probability of failure of the hybrid system,  $P_h$ , is calculated by using equations (11.1) and (11.2) where the program version's failure rates  $e_i$  are substituted by  $P_{rb}(i)$  or  $P_{nv}(i)$ . We now obtain

$$P_h(rb) = \prod_{i=1}^n (P_{nv}(i) + t_{2i}) + \sum_{i=1}^n t_{1i} P_{nv}(i) \left( \prod_{j=1}^{i-1} (P_{nv}(j) + t_{2j}) \right) \quad (11.4)$$

for hybrid systems with a recovery block where each version is an NVP scheme and

$$P_h(nv) = \prod_{i=1}^n P_{rb}(i) + \sum_{i=1}^n \frac{(1 - P_{rb}(i))}{P_{rb}(i)} \left( \prod_{j=1}^n P_{rb}(j) \right) + d \quad (11.5)$$

for hybrid systems with an NVP where each version is a recovery block.

### 11.3.3 Reduction of Common-cause Failures

Before the  $N$ -version programming schemes can be applied to enhance the reliability of critical software, such as the nuclear reactor controller system and the fly-by-wire aircraft, their feasibility should be determined. Both the nuclear reactor controller and fly-by-wire software require ultra-high reliability. The existing  $N$ -version schemes may be unable to offer the required reliability because of their vulnerability to failures due to identical causes. If the majority of the versions fail because of common design errors, then a wrong result may be given by voting on incorrect outputs. The likelihood of common-cause failures in nuclear controller software cannot be ruled out because of its complexity.

Methods to alleviate the common-cause failures include the development of diverse versions by independent teams so as to minimize the commonalities between the various versions. According to Knight and Leveson (1986), experiments have shown that the use of different languages and design philosophy has little effect on the reliability in  $N$ -version programming because people tend to make similar logical mistakes in a difficult-to-program part of the software. Thus, in the presence of a common-cause failure, all the different variations of the  $N$ -version programming prove to be equally useless. It seems beneficial to have a single version in order to minimize cost.

According to the latest research on software reliability, fault-tolerance is a highly recommended application for critical systems. However, a new approach that could alleviate the weakness of the existing fault-tolerant software reliability models is even more desirable. An empirical study by Leveson *et al.* (1990) suggests not to utilize the  $N$ -version if it is known that the probability of making common mistakes during programming is unavoidable. Furthermore, Pham *et al.* (1991a) recommend (1) developing fewer versions, (2) minimizing the errors in the individual versions, and (3) minimizing or eliminating the incidence of common-cause failures in these versions. The author suggests that either two or three versions is reasonably good for a fault tolerant system to achieve the desired

reliability goal given that each version is likely to be reliable based on a realistic reliability evaluation.

Clearly, complex software is developed in a modular fashion and not all the modules are equally complex and difficult to design. Therefore, it is accurate to conclude that the common-cause failures are confined to the “difficult to understand and design logically” part of the problem. The common-cause failures can be reduced if such critical parts are identified and certain design guidelines are followed.

Some suggestions of the design guidelines to reduce or eliminate the common-cause failures are as follows (Pham *et al.* 1991a):

- Techniques to identify critical parts in a program. Generally, the control flow complexity of an algorithm indicates the level of difficulty. We can therefore use the McCabe measure to identify the critical parts of a program.
- The manager of the project should identify the critical sections of the problem, meet the development teams, and steer them to different techniques for solving the critical parts. Suppose that the critical part involves sorting a file. Then one team should be asked to use Quicksort, the other teams should be asked not to use Quicksort but to use some other naïve scheme. In this way, the probability of committing identical logical mistakes can be reduced or eliminated.

Further research on the development of additional design guidelines to minimize the common-cause failures should also be studied.

The self-checking duplex scheme discussed in Section 11.3 incorporates fault-tolerance in two layers. The first layer of protection is provided by self-checking assertions. The second layer is duplication.

In the self-checking system, if one of the versions detects an error at the end of the current module, results are obtained from the other version. After exchanging the correct results, both versions will continue execution in a lock-step fashion. Finally, the output of the duplicated versions are compared for consistency before accepting the result as correct. By keeping the size of the modules sufficiently small, a larger number of errors can be masked by this approach. However, too small a size for the module will increase the overhead of implanting the self-checking assertions. The analysis of the reliability and the optimal module size selection requires further research.

Common-cause failure is still a problem in the self-checking duplex system. There is no known technique to address this in  $N$ -version programming. Therefore, it can only be attempted to reduce the common-cause failures by design diversity. A summary of references on fault-tolerant systems is given in Table 11.1.

**Table 11.1.** Summary of references

Group models	References
General fault tolerant systems	Abbott (1990); Anderson (1980, 1985); Arlat (1990); Geist (1990); Hecht (1979); Iyer (1985); Kanoun (1993); Kim (1989); Laprie (1990); Leveson (1990); Pham. (1989, 1992); Siewiorek (1990)
<i>N</i> -version programming	Anderson (1980, 1985); Avizienis (1977, 1988); Chen (1978); Gersting (1991); Kelly (1988); Knight (1986); Pham (1995); Shimeall (1991); Tso (1987); Vouk (1990)
Recovery block	Kim (1988); Laprie (1990); Pham (1989); Randell (1975)
Other fault-tolerant techniques	Eckhardt (1985); Hua (1986); Kim (1989); Nicola (1990); Pham (1991b, 1992a); Taylor (1980); Vouk (1990)

## 11.4 Triple-version Programming Model with Common Failures

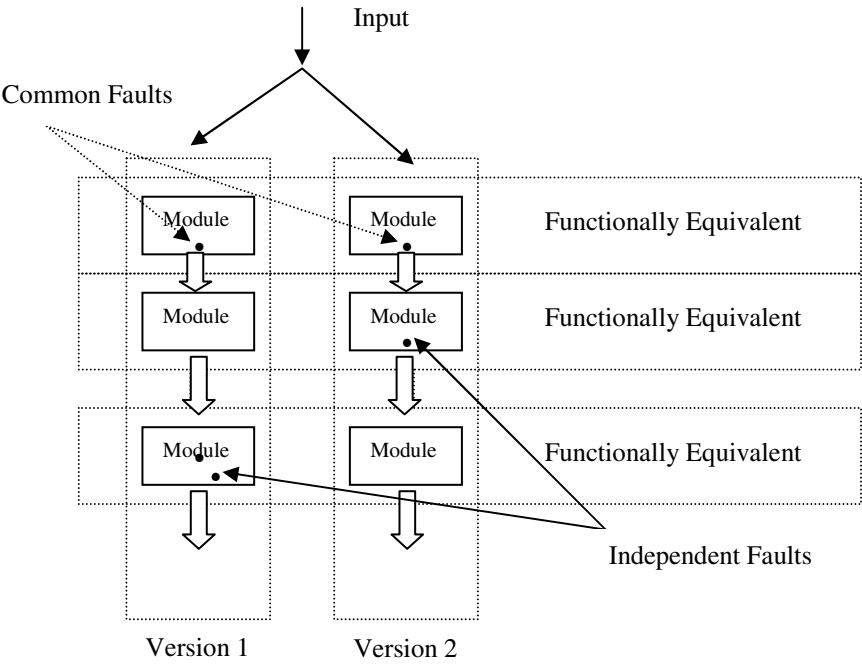
This section discusses a recent software reliability growth model (SGRM) for a triple-version programming (TVP) system with common failures based on NHPP based solely on the papers recently published by Teng and Pham (2002, 2003).

Although diverse software versions are developed by using different specifications, designs, programming teams, programming languages, *etc.*, many researchers have revealed that those independently developed software versions do not necessarily fail independently. In this section we refer to related faults as common faults for simplicity. Figure 11.3 illustrates the common faults and the independent faults in a two-version system.

*Common Faults* are those which are located in the functionally equivalent modules among two or more software versions because their programmers are prone to making the same or similar mistakes although they develop different versions independently. Those faults will be activated by the same input to cause those versions to fail simultaneously, and these failures by common faults are called *Common Failures*.

*Independent Faults* are usually located in different or functionally unequivalent modules between or among different software versions. Since they are independent of each other and are considered harmless to the fault-tolerant systems because their resulting failures are typically distinguishable to the decision mechanism.

However, there is still a probability, though very small compared with that of *Common Failures*, that an unforeseeable input activates two independent faults in different software versions that will lead those versions to fail at the same time. These failures by independent faults are called *Concurrent Independent Failures*.



**Figure 11.3.** Common faults and independent faults

Table 11.2 shows the differences between the common failures and the concurrent independent failures. This section considers that the software failures in TVP systems have two modes: a common failure mode and an s-independent failure mode. There are three independently developed software versions 1, 2, and 3 in the system, which uses majority voting. The reliability of the voter is assumed to be 1.

**Table 11.2.** Common failures and concurrent independent failures

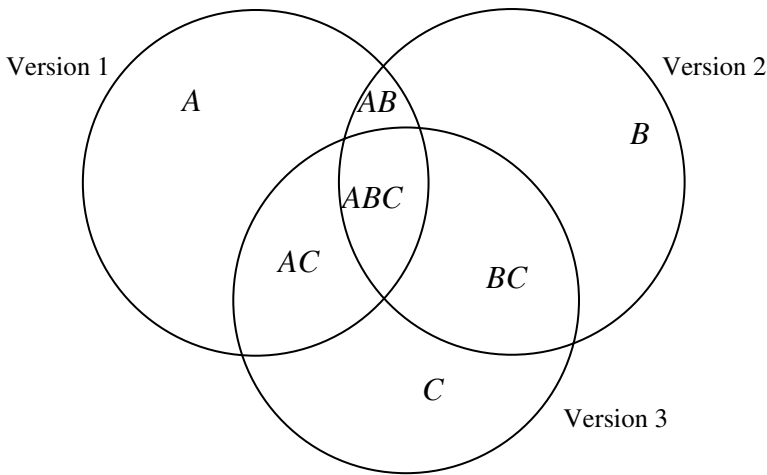
	<i>Common failures</i>	<i>Concurrent independent failures</i>
<i>Fault type</i>	Common faults	Independent faults
<i>Output</i>	Usually the same	Usually different
<i>Fault location (logically)</i>	Same	Different
<i>Voting result (majority voting)</i>	Choose wrong solution	Unable to choose correct solution

*Notation*

$A$	Independent faults in version 1
$B$	Independent faults in version 2
$C$	Independent faults in version 3
$AB$	Common faults between version 1 and version 2
$AC$	Common faults between version 1 and version 3
$BC$	Common faults between version 2 and version 3
$ABC$	Common faults among version 1, version 2 and version 3
$N_x(t)$	Counting process which counts the number of type $x$ faults discovered up to time $t$ , $x = A, B, C, AB, AC, BC$ and $ABC$
$N_d(t)$	$N_d(t) = N_{AB}(t) + N_{AC}(t) + N_{BC}(t) + N_{ABC}(t)$ Counting process which counts common faults discovered in the NVP system up to time $t$
$m_x(t)$	Mean value function of counting process $N_x(t)$ , $m_x(t) = E[N_x(t)]$ $x = A, B, C, AB, AC, BC, ABC$ and $d$
$a_x(t)$	Total number of type $x$ faults in the system plus those type $x$ faults already removed from the system at time $t$ . $a_x(t)$ is non-decreasing function, and $a_x(0)$ denotes the initial number of type $x$ fault in the system, $x = A, B, C, AB, AC, BC$ and $ABC$
$b(t)$	Failure detection rate per fault at time $t$
$\beta_1, \beta_2, \beta_3$	The probability that a new fault is introduced into version 1, 2 and 3 during the debugging, respectively
$p_1, p_2, p_3$	The probability that a new fault is successfully removed from version 1, 2 and 3 during the debugging, respectively
$X_A(t), X_B(t), X_C(t)$	Number of type A, B and C faults at time $t$ remaining in the system respectively
$R(x t)$	Software reliability function for given mission time $x$ and time to stop testing $t$ $R(x t) = \Pr\{\text{No failure occurs during } (t, t+x) \mid \text{stop testing at } t\}$
$R_{NVP-SRGM}(x t)$	NVP system reliability function for given mission time $x$ and time to stop testing $t$ with consideration of common failures in the NVP system
$R_{ind}(x t)$	NVP system reliability function for given mission time $x$ and time to stop testing $t$ , assuming no common failures in the NVP system, i.e., the versions are totally independent of each other.
$K_{AB}, K_{AC}, K_{BC}$	Failure intensity per pair of faults for concurrent independent failures between version 1 and 2, between 1 and 3 and between 2 and 3 respectively
$N_{\overline{AB}}(t), N_{\overline{AC}}(t)$	Counting processes that count the number of concurrent independent failures involving version 1 and 2, version 1 and 3, and version 2 and 3 up to time $t$ respectively
$N_{\overline{BC}}(t)$	
$N_I(t)$	Counting process that counts the total number of concurrent independent failures up to time $t$ , $N_I(t) = N_{\overline{AB}}(t) + N_{\overline{AC}}(t) + N_{\overline{BC}}(t)$

$m_{\overline{AB}}(t), m_{\overline{AC}}(t)$	Mean value functions of the corresponding counting processes.
$m_{\overline{BC}}(t), m_i(t)$	For example, $m_{\overline{AB}}(t) = E[N_{\overline{AB}}(t)]$
$h_{\overline{AB}}(t), h_{\overline{AC}}(t)$	Failure intensity functions of concurrent independent failures involving version 1 and 2, between 1 and 3, and between 2 and 3
$h_{\overline{BC}}(t)$	$h_{\overline{AB}}(t) = \frac{d}{dt} m_{\overline{AB}}(t)$
$Pr(\cdot T)$	Conditional probability given that testing and debugging are stopped at time $T$

A concurrent independent failure occurs when two or more versions fail by independent faults at the same input, *i.e.*,  $A$ ,  $B$  or  $C$ , not by  $AB$ ,  $AC$ , *etc.* Different fault types and their relations are shown in Figure 11.4.



**Figure 11.4.** Different software faults in the 3-version software system

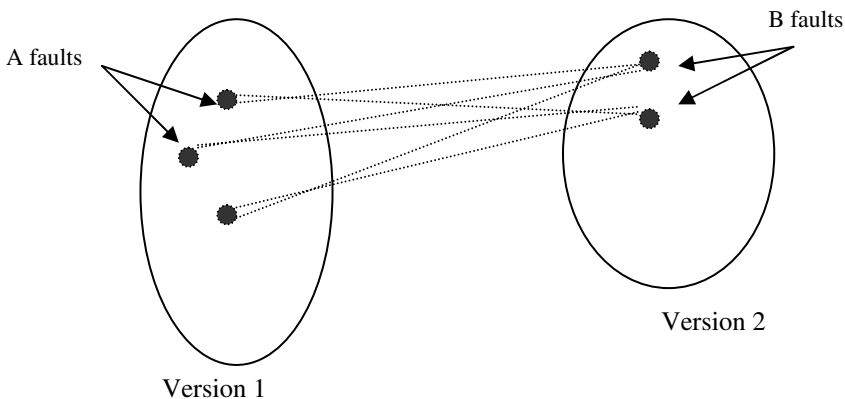
#### 11.4.1 Modeling Assumptions

Consider the following assumptions:

1. Faster versions will have to wait for the slowest versions to finish (prior to voting).
2. Each software version can fail during execution, caused by faults in the software.
3. Two or more software versions may fail on the same input, which can be caused by either the common faults or the independent faults in different versions.
4. The occurrence of software failures caused by different faults (independent faults, 2-version common faults or 3-version common faults) follows an NHPP.

5. The software-failure detection rate at any time is proportional to the number of faults remaining in the software at that time.
6. When a software failure occurs in any of the three versions, a debugging effort is executed immediately. That effort removes the faults immediately with probability  $p_i$ ,  $p_i \gg 1 - p_i$  ( $i$  representing the version number 1, 2 or 3).
7. For each debugging effort, whether the fault is successfully removed or not, some new independent faults may be introduced into the software system with probability  $\beta_i$ ,  $\beta_i \ll p_i$  but no new common faults will be introduced into the system.
8. Some common faults may reduce to some low-level common faults or independent faults due to unsuccessful removal efforts.
9. The error detection rates per fault for all kinds of faults  $A$ ,  $B$ ,  $C$ ,  $AB$ ,  $AC$ ,  $BC$  and  $ABC$  are the same and constant, *i.e.*,  $b(t) = b$ .
10. The concurrent independent failures are caused by the activation of independent faults between different versions, and the probability that a concurrent independent failure involves three versions is zero. Those failures only involve two versions.
11. Any pair of remaining independent faults between versions has the same probability to be activated by some input.
12. The intensity for concurrent independent failures involving any two versions is proportional to the remaining pairs of independent faults in those two versions.

Figure 11.5 shows the pairs of independent faults between software version 1 and version 2. There are three independent faults (type A) in version 1, and there are two independent faults (type B) in version 2. There are six pairs of independent faults between version 1 and version 2. It is assumed that each of these six pairs has the same probability to be activated by some input.



**Figure 11.5.** Independent fault pairs between version 1 and version 2

Based on the above assumptions and the software reliability model studied by Zhang *et al.* ((2003) (Equations 1,2 and 9 therein), the following NHPP system of equations for different faults and software failures can be obtained:



## (1) Error type ABC

$$m'_{ABC}(t) = b \cdot (a_{ABC} - p_1 \cdot p_2 \cdot p_3 \cdot m_{ABC}(t)) \quad (11.6)$$

with marginal conditions  $m_{ABC}(0) = 0$  and  $a_{ABC}(0) = a_{ABC}$ . The solution to equation (11.6) is

$$m_{ABC}(t) = \frac{a_{ABC}}{p_1 \cdot p_2 \cdot p_3} \cdot (1 - e^{-b \cdot p_1 \cdot p_2 \cdot p_3 \cdot t}) \quad (11.7)$$

## (2) Error type AB

$$\begin{aligned} m'_{AB}(t) &= b \cdot (a_{AB}(t) - p_1 \cdot p_2 \cdot m_{AB}(t)) \\ a'_{AB}(t) &= (1 - p_1) \cdot (1 - p_2) \cdot p_3 \cdot m'_{ABC}(t) \end{aligned} \quad (11.8)$$

Similarly, error type AC

$$\begin{aligned} m'_{AC}(t) &= b \cdot (a_{AC}(t) - p_1 \cdot p_3 \cdot m_{AC}(t)) \\ a'_{AC}(t) &= (1 - p_1) \cdot (1 - p_3) \cdot p_2 \cdot m'_{ABC}(t) \end{aligned} \quad (11.9)$$

Error type BC

$$\begin{aligned} m'_{BC}(t) &= b \cdot (a_{BC}(t) - p_2 \cdot p_3 \cdot m_{BC}(t)) \\ a'_{BC}(t) &= (1 - p_2) \cdot (1 - p_3) \cdot p_1 \cdot m'_{ABC}(t) \end{aligned} \quad (11.10)$$

with marginal conditions

$$\begin{aligned} m_{AB}(0) &= 0, & a_{AB}(0) &= a_{AB} \\ m_{AC}(0) &= 0, & a_{AC}(0) &= a_{AC} \\ m_{BC}(0) &= 0, & a_{BC}(0) &= a_{BC} \end{aligned}$$

Substituting equation (11.7) into equations (11.8)-(11.9), we can obtain the mean value function for AB, AC and BC:

$$m_{AB}(t) = C_{AB1} - C_{AB2} \cdot e^{-b \cdot p_1 \cdot p_2 \cdot t} + C_{AB3} \cdot e^{-b \cdot p_1 \cdot p_2 \cdot p_3 \cdot t} \quad (11.11)$$

$$m_{AC}(t) = C_{AC1} - C_{AC2} \cdot e^{-b \cdot p_1 \cdot p_3 \cdot t} + C_{AC3} \cdot e^{-b \cdot p_1 \cdot p_2 \cdot p_3 \cdot t} \quad (11.12)$$

$$m_{BC}(t) = C_{BC1} - C_{BC2} \cdot e^{-b \cdot p_2 \cdot p_3 \cdot t} + C_{BC3} \cdot e^{-b \cdot p_1 \cdot p_2 \cdot p_3 \cdot t} \quad (11.13)$$

where

$$\begin{aligned} C_{AB1} &= \frac{a_{AB}}{p_1 \cdot p_2} + \frac{a_{ABC} \cdot (1 - p_1) \cdot (1 - p_2)}{p_1^2 \cdot p_2^2} \\ C_{AB2} &= \frac{a_{AB}}{p_1 \cdot p_2} + \frac{a_{ABC} \cdot (1 - p_1) \cdot (1 - p_2)}{p_1^2 \cdot p_2^2} - \frac{a_{ABC} \cdot (1 - p_1) \cdot (1 - p_2)}{p_1^2 \cdot p_2^2 \cdot (1 - p_3)} \end{aligned}$$

$$\begin{aligned}
C_{AB3} &= -\frac{a_{ABC} \cdot (1-p_1) \cdot (1-p_2)}{p_1^2 \cdot p_2^2 \cdot (1-p_3)} \\
C_{AC1} &= \frac{a_{AC}}{p_1 \cdot p_3} + \frac{a_{ABC} \cdot (1-p_1) \cdot (1-p_3)}{p_1^2 \cdot p_3^2} \\
C_{AC2} &= \frac{a_{AC}}{p_1 \cdot p_3} + \frac{a_{ABC} \cdot (1-p_1) \cdot (1-p_3)}{p_1^2 \cdot p_3^2} - \frac{a_{ABC} \cdot (1-p_1) \cdot (1-p_3)}{p_1^2 \cdot p_3^2 \cdot (1-p_2)} \\
C_{AC3} &= -\frac{a_{ABC} \cdot (1-p_1) \cdot (1-p_3)}{p_1^2 \cdot p_3^2 \cdot (1-p_3)} \\
C_{BC1} &= \frac{a_{BC}}{p_2 \cdot p_3} + \frac{a_{ABC} \cdot (1-p_2) \cdot (1-p_3)}{p_2^2 \cdot p_3^2} \\
C_{BC2} &= \frac{a_{BC}}{p_2 \cdot p_3} + \frac{a_{ABC} \cdot (1-p_2) \cdot (1-p_3)}{p_2^2 \cdot p_3^2} - \frac{a_{ABC} \cdot (1-p_2) \cdot (1-p_3)}{p_2^2 \cdot p_3^2 \cdot (1-p_1)} \\
C_{BC3} &= -\frac{a_{ABC} \cdot (1-p_2) \cdot (1-p_3)}{p_2^2 \cdot p_3^2 \cdot (1-p_1)}
\end{aligned}$$

(3) Error type A

$$\begin{aligned}
m_A'(t) &= b \cdot (a_A(t) - p_1 \cdot m_A(t)) \\
a_A'(t) &= \beta_1 \cdot (m_A'(t) + m_{AB}'(t) + m_{AC}'(t) + m_{ABC}'(t)) \\
&\quad + (1-p_1) \cdot p_2 \cdot m_{AB}'(t) + (1-p_1) \cdot p_3 \cdot m_{AC}'(t) \\
&\quad + (1-p_1) \cdot p_2 \cdot p_3 \cdot m_{ABC}'(t)
\end{aligned} \tag{11.14}$$

Similarly, error type B

$$\begin{aligned}
m_B'(t) &= b \cdot (a_B(t) - p_2 \cdot m_B(t)) \\
a_B'(t) &= \beta_2 \cdot (m_B'(t) + m_{AB}'(t) + m_{BC}'(t) + m_{ABC}'(t)) \\
&\quad + (1-p_2) \cdot p_1 \cdot m_{AB}'(t) + (1-p_2) \cdot p_3 \cdot m_{BC}'(t) \\
&\quad + (1-p_2) \cdot p_1 \cdot p_3 \cdot m_{ABC}'(t)
\end{aligned} \tag{11.15}$$

Error type C

$$m_C'(t) = b \cdot (a_C(t) - p_3 \cdot m_C(t))$$

$$\begin{aligned}
a_C'(t) = & \beta_3 \cdot (m_C'(t) + m_{AC}'(t) + m_{BC}'(t) + m_{ABC}'(t)) \\
& + (1 - p_3) \cdot p_1 \cdot m_{AC}'(t) + (1 - p_3) \cdot p_2 \cdot m_{BC}'(t) \\
& + (1 - p_3) \cdot p_1 \cdot p_2 \cdot m_{ABC}'(t)
\end{aligned} \tag{11.16}$$

with marginal conditions

$$m_A(0) = 0, \quad a_A(0) = a_A$$

$$m_B(0) = 0, \quad a_B(0) = a_B$$

$$m_C(0) = 0, \quad a_C(0) = a_C$$

The solutions to the above equations are lengthy, although they are straightforward to solve.

### 11.4.2 TVP Reliability Function

An NVP system fails when more than half of its versions fail simultaneously at the same time. It is assumed that a voter or decision mechanism is a perfect one. As we mentioned before, we divide these failures into two categories: common failures and independent failures. The NVP system reliability can be obtained by combining these two failure modes together.

#### *Common Failure Mode*

Let  $N_d(t)$  be the total number of software system failures which are caused by common faults in the system. Then it is easily to see that

$$N_d(t) = N_{AB}(t) + N_{AC}(t) + N_{BC}(t) + N_{ABC}(t) \tag{11.17}$$

Therefore, the mean value function of  $N_d(t)$  is as follows

$$m_d(t) = m_{AB}(t) + m_{AC}(t) + m_{BC}(t) + m_{ABC}(t) \tag{11.18}$$

The probability that the TVP software system will not fail during  $(T, T+x)$  given that the latest failure occurred at time  $T$  is

$$Pr\{\text{No common failure during } x|T\} = e^{-(m_d(T+x) - m_d(T))} \tag{11.19}$$

### Independent Failure

Commonly the TVP software system fails on common faults among versions; however, there is a small probability that two software versions fail on the same input because of independent faults.

From assumptions 12 and 13, the failure intensity  $h_{AB}(t)$  for the concurrent independent failures between version 1 and version 2 is given by

$$h_{AB}(t) = K_{AB} \cdot X_A(t) \cdot X_B(t)$$

where the numbers of independent faults in version 1 and 2 are

$$X_A(t) = a_A(t) - p_1 \cdot m_A(t)$$

$$X_B(t) = a_B(t) - p_2 \cdot m_B(t)$$

Then the mean value function for concurrent independent failures  $N_{\overline{AB}}(t)$  is given by

$$m_{\overline{AB}}(t) = \int_0^t h_{\overline{AB}}(\tau) d\tau \quad (11.20)$$

Given the release time  $T$  and mission time  $x$ , we can obtain the probability that there is no concurrent independent failure  $\overline{AB}$  during  $x$

$$Pr\{\text{no } \overline{AB} \text{ failure during } x|T\} = e^{-(m_{\overline{AB}}(T+x)-m_{\overline{AB}}(T))} \quad (11.21)$$

Similarly, the mean value functions for concurrent independent failures between version 1 and 3 and between version 2 and 3, respectively, are

$$m_{\overline{AC}}(t) = \int_0^t h_{\overline{AC}}(\tau) d\tau \quad (11.22)$$

$$m_{\overline{BC}}(t) = \int_0^t h_{\overline{BC}}(\tau) d\tau \quad (11.23)$$

Given the release time  $T$  and mission time  $x$ , we can obtain the probability that there is no concurrent independent failure  $\overline{AC}$  and  $\overline{BC}$  during  $x$

$$Pr\{\text{no } \overline{AC} \text{ failure during } x|T\} = e^{-(m_{\overline{AC}}(T+x)-m_{\overline{AC}}(T))} \quad (11.24)$$

$$Pr\{\text{no } \overline{BC} \text{ failure during } x|T\} = e^{-(m_{\overline{BC}}(T+x)-m_{\overline{BC}}(T))} \quad (11.25)$$

where

$$h_{\overline{AC}}(t) = K_{AC} \cdot X_A(t) \cdot X_C(t)$$

$$h_{\overline{BC}}(t) = K_{BC} \cdot X_B(t) \cdot X_C(t)$$

If we define

$$N_I(t) = N_{\overline{AB}}(t) + N_{\overline{AC}}(t) + N_{\overline{BC}}(t)$$

with the mean value function

$$m_I(t) = m_{\overline{AB}}(t) + m_{\overline{AC}}(t) + m_{\overline{BC}}(t) \quad (11.26)$$

then the probability that there is no independent failure during mission  $x$  for the NVP system is

$$\begin{aligned} P\{\text{no concurrent independence failure during } x|T\} \\ = e^{-(m_I(T+x)-m_I(T))} \end{aligned} \quad (11.27)$$

Because the common failures and concurrent independent failures are independent of each other, then

$$\begin{aligned} R_{TVP}(x|T) &= Pr\{\text{no common failure during } x|T\} \times \\ &\quad Pr\{\text{no concurrent independent failure during } x|T\} \end{aligned}$$

From equations (11.19) and (11.27), the reliability of TVP system can be determined by

$$R_{TVP}(x|T) = e^{-[m_d(T+x)+m_I(T+x)-m_d(T)-m_I(T)]} \quad (11.28)$$

One can estimate the parameters in the model by using the MLE method. See Teng and Pham (2002) for details.

### 11.4.3 Numerical Example

Consider a simplify software control logic for a water reservoir control (WRC) system (Teng and Pham, 2002). Water is supplied via a source pipe controlled by a source valve and removed via a drain pipe controlled by a drain valve. There are two level sensors, positioned at the high and low limits; the high sensor does an output action above if the level is above it and the low sensor outputs below if the level is below it.

The control system should maintain the water level between these two limits, allowing for rainfall into and seepage from the reservoir. If, however, the water rises above the high level, an alarm should sound. The WRC system achieves fault tolerance and high reliability through the use of two-version programming (2VP) software control logic. The WRC software system with normalized data is listed in Table 11.3.

**Table 11.3.** Failure normalized-data of WRC 2VP system

Fault No.	Failure time (Version)		Fault No.	Failure time (Version)	
K	Version 1	Version 2	K	Version 1	Version 2
1	1.2	3.6	14	39.2	34.8
2	2.8	8.4	15	40	36.4
3	8.4	12.8	16	44	36.8
4	10	14.4	17	44.8	38
5	16.4	17.2	18	54	39.2
6	20	18	19	56	41.6
7	24.4	20	20	62.4	42
8	28	23.2	21	80	46.4
9	29.2	25.2	22	92	59.6
10	31.2	28	23	99.6	62.4
11	34	28.4	24		98.8
12	36	30.8	25		99.6
13	36.8	31.2	26		100

In this example, we assume that the reliability of the voter is equal to 1 and the voter can identify exactly which version(s) is failed whenever a failure occurred. Under these assumptions, the 2VP system fails only when both its components (software versions) fail at the same input data.

This example considers two cases - one is that two software versions are assumed to fail *s*-independent; the other is that both versions are not assumed to fail *s*-independently.

*Case 1: Independent of TVP Software Versions.* Assuming that those two software versions are *s*-independent of each other. We can apply the generalized software

reliability model to each version separately, and estimate the reliability of each version  $R_1(x|t)$  and  $R_2(x|t)$ , and further obtain the reliability for the entire system by using

$$R_{ind}(x|t) = 1 - (1 - R_1(x|t)) \cdot (1 - R_2(x|t)) \quad (11.29)$$

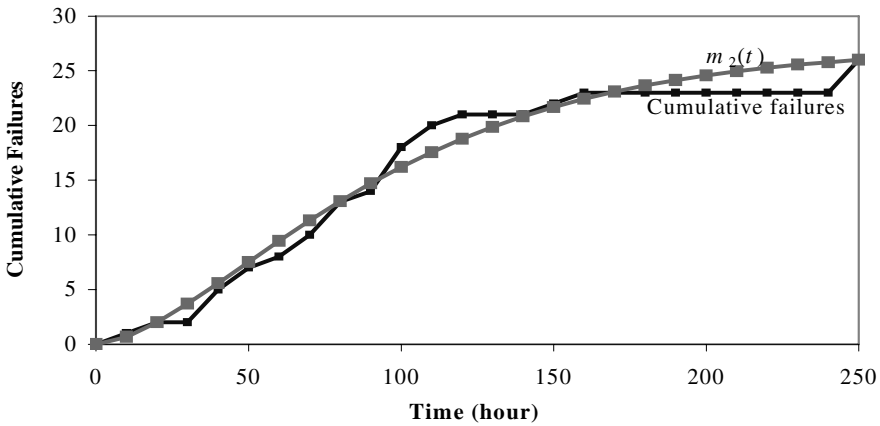
where  $x$  is a mission time,

$$R_1(x|t) = e^{-(m_1(t+x) - m_1(t))}$$

and

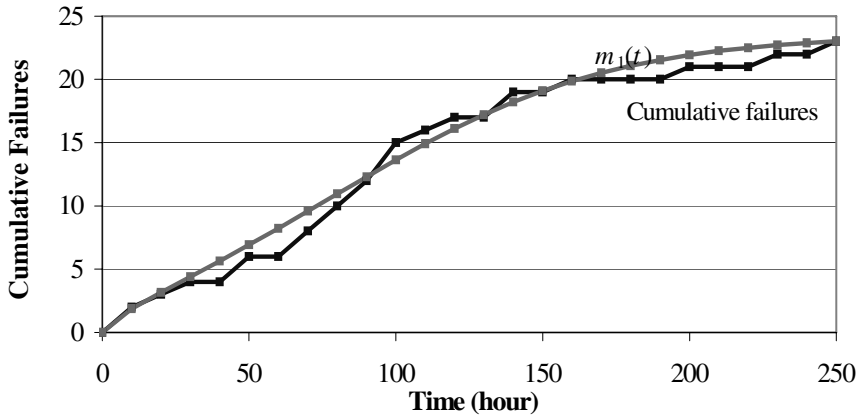
$$R_2(x|t) = e^{-(m_2(t+x) - m_2(t))}$$

Figures 11.6 and 11.7 show the goodness of fit of mean value function  $m_1(t)$  (for version 1) and  $m_2(t)$  for (version 2), respectively. The reliability function  $R_{ind}(x|t)$  is also illustrated in Figure 11.11.



**Figure 11.6.** Function  $m_1(t)$  vs cumulative number of failures in version 1

*Case 2: Dependent of TVP Software Versions.* From Table 11.4, we can observe that two versions fail simultaneously at some time, for example, at  $t = 8.4, 20, 28, \dots, 99.6$ . These failures are considered as coincident failures that are caused either by the common faults, or unrelated faults between two versions. Therefore, the assumption of independence is not valid for this normalized-data set.



**Figure 11.7.** Function  $m_2(t)$  vs. vs. cumulative number of failures in version 2

In this example, we assume that all concurrent failures are common failures. Table 11.4 is generated directly from Table 11.3 and it shows the different fault types in this 2VP system. Similar from the TVP modeling formulation, we can easily obtain the mean value functions.

From Section 11.4.1, we can obtain the results as follows:

(1) Error type AB

$$m'_{AB}(t) = b \cdot (a_{AB} - p_1 \cdot p_2 \cdot m_{AB}(t)) \quad (11.30)$$

with marginal condition  $m_{AB}(0) = 0$ , and  $a_{AB}(0) = a_{AB}$ . The solution to equation (11.30) is

$$m_{AB}(t) = \frac{a_{AB}}{p_1 \cdot p_2} (1 - e^{-bp_1 p_2 t}) \quad (11.31)$$

(2) Fault type A

$$m'_A(t) = b \cdot (a_A(t) - p_1 \cdot m_A(t))$$

$$a'_A(t) = (1 - p_1) \cdot p_2 \cdot m'_{AB}(t) + \beta_1 \cdot (m'_A(t) + m'_{AB}(t)) \quad (11.32)$$

with marginal condition  $m_A(0) = 0$ , and  $a_A(0) = a_A$ . The solution to equation (11.32) is

$$m_A(t) = C_{A1} + C_{A2} \cdot e^{-bp_1 p_2 t} + C_{A3} \cdot e^{-b(p_1 - \beta_1)t} \quad (11.33)$$

where

$$C_{A1} = \frac{a_A}{p_1 - \beta_1} + \frac{((1 - p_1) \cdot p_2 + \beta_1) \cdot a_{AB}}{p_1 \cdot p_2 \cdot (p_1 - \beta_1)}$$

$$C_{A2} = -\frac{((1-p_1) \cdot p_2 + \beta_1) \cdot a_{AB}}{p_1 \cdot p_2 \cdot (p_1 \cdot (1-p_2) - \beta_1)}$$

$$C_{A3} = \frac{((1-p_1) \cdot p_2 + \beta_1) \cdot a_{AB}}{(p_1 - \beta_1) \cdot (p_1 \cdot (1-p_2) - \beta_1)} - \frac{a_A}{p_1 - \beta_1}$$

**Table 11.4.** Fault type table for 2VP system

Fault #	Failure time (hour)		
	Fault A	Fault B	Fault AB
1	1.2	3.6	8.4
2	2.8	12.8	20
3	10	14.4	28
4	16.4	17.2	31.2
5	24.4	18	36.8
6	29.2	23.2	39.2
7	34	25.2	62.4
8	36	28.4	99.6
9	40	30.8	
10	44	34.8	
11	44.8	36.4	
12	54	38	
13	56	41.6	
14	80	42	
15	92	46.4	
16		59.6	
17		98.8	
18		100	

## (3) Fault type B

$$m'_B(t) = b \cdot (a_B(t) - p_2 \cdot m_B(t)) \quad (11.34)$$

$$a'_B(t) = (1-p_2) \cdot p_1 \cdot m'_{AB}(t) + \beta_2 \cdot (m'_B(t) + m'_{AB}(t))$$

with marginal condition  $m_B(0) = 0$ , and  $a_B(0) = a_B$ . The solution to equation (11.34) is

$$m_B(t) = C_{B1} + C_{B2} \cdot e^{-bp_1 p_2 t} + C_{B3} \cdot e^{-b(p_2 - \beta_2)t} \quad (11.35)$$

where

$$C_{B1} = \frac{a_B}{p_2 - \beta_2} + \frac{a_{AB} \cdot ((1-p_2) \cdot p_1 + \beta_2)}{p_1 \cdot p_2 \cdot (p_2 - \beta_2)}$$

$$C_{B2} = -\frac{((1-p_2) \cdot p_1 + \beta_2) \cdot a_{AB}}{p_1 \cdot p_2 \cdot (p_2 \cdot (1-p_1) - \beta_2)}$$

$$C_{B3} = \frac{((1-p_2) \cdot p_1 + \beta_2) \cdot a_{AB}}{(p_2 - \beta_2) \cdot (p_2 \cdot (1-p_1) - \beta_2)} - \frac{a_B}{p_2 - \beta_2}$$



The likelihood function is given by

$$L = L_A \cdot L_B \cdot L_{AB}$$

where

$$\begin{aligned} L_A &= \prod_{i=1}^{n_A} \left\{ \frac{[m_A(t_i) - m_A(t_{i-1})]^{y_{Ai} - y_{A(i-1)}}}{(y_{Ai} - y_{A(i-1)})!} \cdot e^{-[m_A(t_i) - m_A(t_{i-1})]} \right\} \\ L_B &= \prod_{i=1}^{n_B} \left\{ \frac{[m_B(t_i) - m_B(t_{i-1})]^{y_{Bi} - y_{B(i-1)}}}{(y_{Bi} - y_{B(i-1)})!} \cdot e^{-[m_B(t_i) - m_B(t_{i-1})]} \right\} \\ L_{AB} &= \prod_{i=1}^{n_{AB}} \left\{ \frac{[m_{AB}(t_i) - m_{AB}(t_{i-1})]^{y_{ABi} - y_{AB(i-1)}}}{(y_{ABi} - y_{AB(i-1)})!} \cdot e^{-[m_{AB}(t_i) - m_{AB}(t_{i-1})]} \right\} \end{aligned}$$

Therefore, the log of likelihood function can be obtained:

$$\begin{aligned} \ln(L) &= \sum_{i=1}^{n_A} \{ (y_{Ai} - y_{A(i-1)}) \cdot \ln(m_A(t_i) - m_A(t_{i-1})) \\ &\quad - m_A(t_i) + m_A(t_{i-1}) - \ln((y_{Ai} - y_{A(i-1)})!) \} \\ &\quad + \sum_{i=1}^{n_B} \{ (y_{Bi} - y_{B(i-1)}) \cdot \ln(m_B(t_i) - m_B(t_{i-1})) \\ &\quad - m_B(t_i) + m_B(t_{i-1}) - \ln((y_{Bi} - y_{B(i-1)})!) \} \\ &\quad + \sum_{i=1}^{n_{AB}} \{ (y_{ABi} - y_{AB(i-1)}) \cdot \ln(m_{AB}(t_i) - m_{AB}(t_{i-1})) \\ &\quad - m_{AB}(t_i) + m_{AB}(t_{i-1}) - \ln((y_{ABi} - y_{AB(i-1)})!) \} \end{aligned}$$

Taking a derivative with respect to each unknown parameters, setting it to zero, and solving the system of equations, we can finally obtain MLEs of all unknown parameters. The confidence interval for each parameter estimate can easily be obtained by constructing the Hessian matrix  $H$ . The Hessian matrix  $H$  can be obtained as follows:

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{14} & h_{15} & h_{16} \\ h_{21} & h_{22} & h_{23} & h_{24} & h_{25} & h_{26} \\ h_{31} & h_{32} & h_{33} & h_{34} & h_{35} & h_{36} \\ h_{41} & h_{42} & h_{43} & h_{44} & h_{45} & h_{46} \\ h_{51} & h_{52} & h_{53} & h_{54} & h_{55} & h_{56} \\ h_{61} & h_{62} & h_{63} & h_{64} & h_{65} & h_{66} \end{bmatrix}$$

where

$$h_{ij} = \frac{\partial^2 L}{\partial x_i \partial x_j} \quad i, j = 1, \dots, 6$$

and the expression,

$$\begin{array}{lll} x_1 \rightarrow a_A & x_2 \rightarrow a_B & x_3 \rightarrow a_{AB} \\ x_4 \rightarrow \beta_1 & x_5 \rightarrow \beta_2 & x_6 \rightarrow b \end{array}$$

For example,

$$\begin{aligned} h_{11} &= \frac{\partial^2 L}{\partial x_1^2} = \frac{\partial^2 L}{\partial a_A^2} \\ &= \sum_{i=1}^{n_A} \frac{\left( \frac{-e^{-b(p_1 - \beta_1)t_i} + e^{-b(p_1 - \beta_1)t_{i-1}}}{p_1 - \beta_1} \right)^2}{\left( C_{A2} \left( e^{-bp_1 p_2 t_i} - e^{-bp_1 p_2 t_{i-1}} \right) + C_{A3} \left( e^{-b(p_1 - \beta_1)t_i} - e^{-b(p_1 - \beta_1)t_{i-1}} \right) \right)^2} \end{aligned}$$

where  $C_{A2}$  and  $C_{A3}$  are given in equation (11.33). The variance matrix,  $V$ , can be obtained as follows:

$$V = [-H]^{-1} = \begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{14} & v_{15} & v_{16} \\ v_{21} & v_{22} & v_{23} & v_{24} & v_{25} & v_{26} \\ v_{31} & v_{32} & v_{33} & v_{34} & v_{35} & v_{36} \\ v_{41} & v_{42} & v_{43} & v_{44} & v_{45} & v_{46} \\ v_{51} & v_{52} & v_{53} & v_{54} & v_{55} & v_{56} \\ v_{61} & v_{62} & v_{63} & v_{64} & v_{65} & v_{66} \end{bmatrix}$$

where  $v_{ij}$  is the covariance of  $x_i$  and  $x_j$  and

$$\begin{aligned} v_{11} &= \text{Var}(x_1) = \text{Var}(a_A) \\ v_{22} &= \text{Var}(x_2) = \text{Var}(a_B) \\ v_{33} &= \text{Var}(x_3) = \text{Var}(a_{AB}) \\ v_{44} &= \text{Var}(x_4) = \text{Var}(\beta_1) \\ v_{55} &= \text{Var}(x_5) = \text{Var}(\beta_2) \\ v_{66} &= \text{Var}(x_6) = \text{Var}(b) \end{aligned}$$

Since all coincident failures are common failures then  $K_{AB} = 0$ . Assume  $p_1 = p_2 = 0.9$ , then the corresponding MLEs are given by

$$\begin{array}{lll} \hat{a}_A = 15.47 & \hat{a}_B = 18.15 & \hat{a}_{AB} = 7.8 \\ \hat{\beta}_1 = 0 & \hat{\beta}_2 = 0.002324 & \hat{b} = 0.009 \end{array}$$

The corresponding Hessian matrix and variance matrices are

$$H = \begin{bmatrix} -0.0763 & 0 & -0.0042 & -1.028 & 0 & -39.48 \\ 0 & 0.051 & -0.0037 & 0 & -1.043 & -36.55 \\ -0.0042 & -0.0037 & -0.132 & -0.0825 & -0.133 & -40.11 \\ -1.028 & 0 & -0.0825 & -31.68 & 0 & 37.38 \\ 0 & -1.043 & -0.133 & 0 & -33.26 & 68.38 \\ -39.48 & -36.55 & -40.11 & 37.38 & 68.38 & -179746.12 \end{bmatrix}$$

$$V = [-H]^{-1} = \begin{bmatrix} 41.473 & 40.4 & 5.625 & -1.384 & -1.33 & -0.0194 \\ 40.4 & 143.61 & 13.28 & -1.397 & -4.645 & -0.0431 \\ 5.625 & 13.28 & 9.511 & -0.215 & -0.467 & -0.0063 \\ -1.384 & -1.397 & -0.215 & 0.0778 & 0.046 & 0.00067 \\ -1.33 & -4.645 & -0.467 & 0.046 & 0.181 & 0.00142 \\ -0.0194 & -0.0431 & -0.0063 & 0.00067 & 0.00142 & 2.067e-05 \end{bmatrix}$$

Then the variance of estimations are

$$\begin{aligned} Var(\hat{a}_A) &= 41.473 & Var(\hat{a}_B) &= 143.61 & Var(\hat{a}_{AB}) &= 9.511 \\ Var(\hat{\beta}_1) &= 0.0778 & Var(\hat{\beta}_2) &= 0.181 & Var(b) &= 2.067 \times 10^{-5} \end{aligned}$$

The NVP-SRGM software system reliability is given by

$$R_{NVP-SRGM}(x|t) = e^{-(m_{AB}(t+x) - m_{AB}(t))}$$

Figures 11.8-11.9 show the mean value functions and their 95% confidence intervals as well as the number of cumulative failures. Figure 11.10 shows the 2VP system reliability and its 95% confidence interval for a mission time  $x=10$  hours. The reliability  $R_{NVP-SRGM}(x|t)$ ,  $R_{ind}(x|t)$  and component reliability  $R_1(x|t)$  and  $R_2(x|t)$  for the 2VP system are shown in Figure 11.11 with mission time  $x = 10$ . Figure 11.11 shows that the 2VP scheme has a higher reliability than any single component, which means that the 2VP scheme is able to provide higher system reliability. It seems that more application is needed to validate fully the NVP-SRGM for quantify model discuss in the section (Teng 2002) in a general industrial setting.

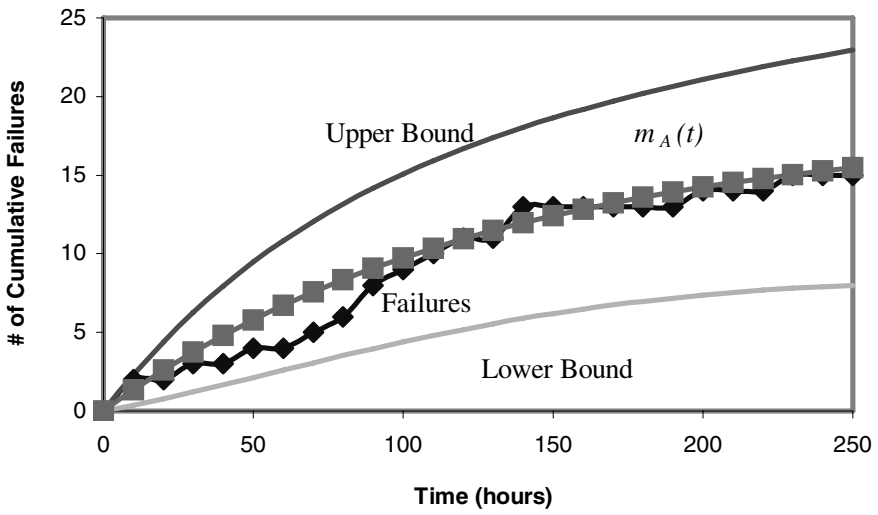


Figure 11.8.  $m_A(t)$  vs the number of cumulative type A failures

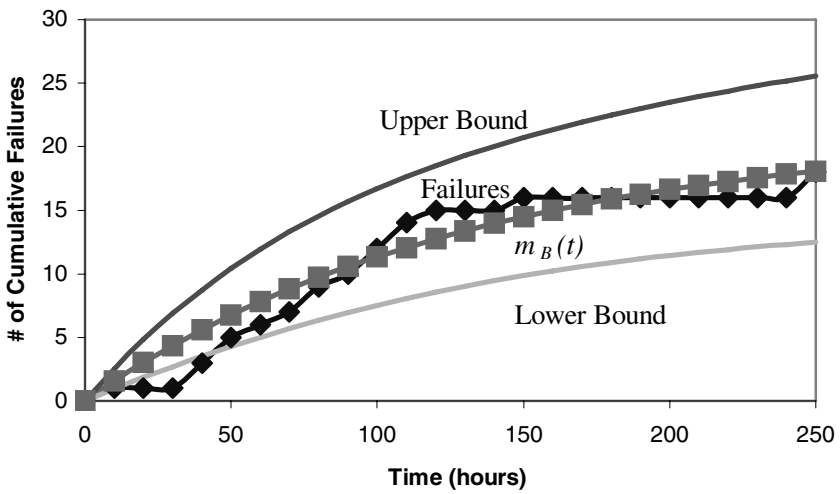


Figure 11.9.  $m_B(t)$  vs the number of cumulative type B failures

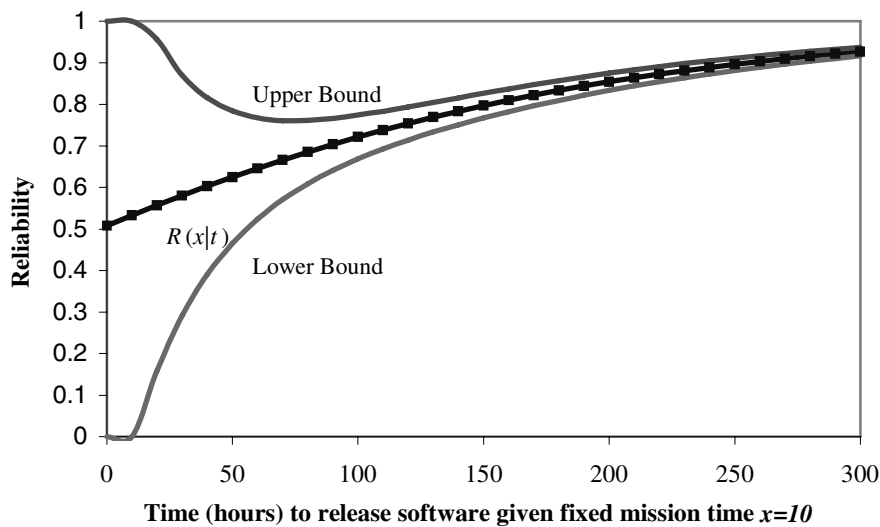


Figure 11.10. 2VP system reliability and its 95% confidence interval

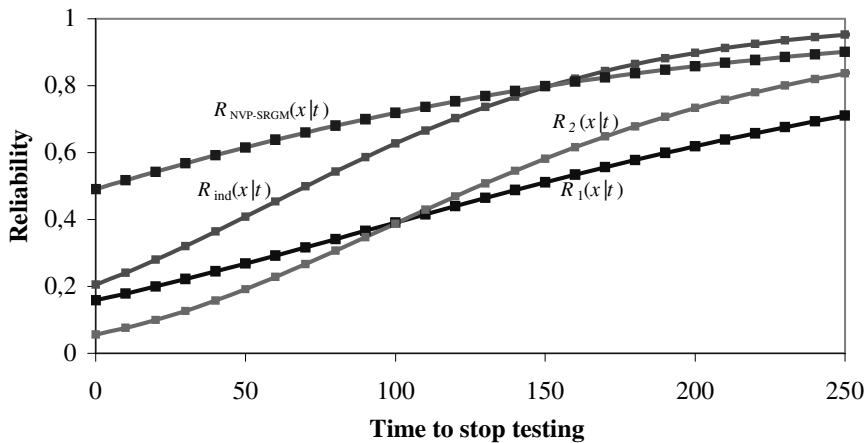


Figure 11.11. 2VP reliability comparisons (mission time  $x = 10$ )

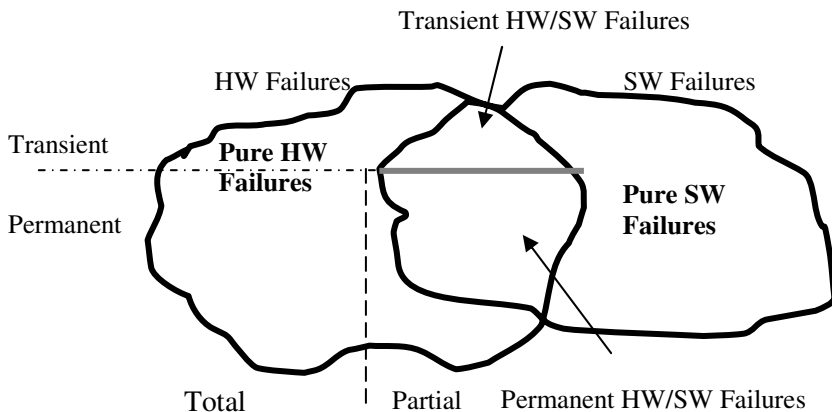
## 11.5 Complex-system Reliability Modeling

This section discusses a reliability model considering system failures due to hardware failures, software failures or hardware-software interaction failures based on the work by Teng *et al.* (2001). A system reliability model is discussed based on Markov processes. Hardware-software interaction failures can be specified into two categories: transient and permanent hardware-related software failures.

### 11.5.1 System Considerations

Figure 11.12 shows the system failure categories. The overlap region between hardware failures and software failures represents hardware-software interaction (HW/SW) failures. Because of the associations with system hardware components, all HW/SW failures can be further divided into two categories: Transient and permanent hardware/software interaction failures. Figure 11.13 shows a presentation of the system reliability diagram.

Figure 11.13 divides hardware and software failures into four parts. They are H-2/S-2, which represents permanent HW/SW failures, and H-3/S-3, which represents temporary (transient) HW/SW failures.



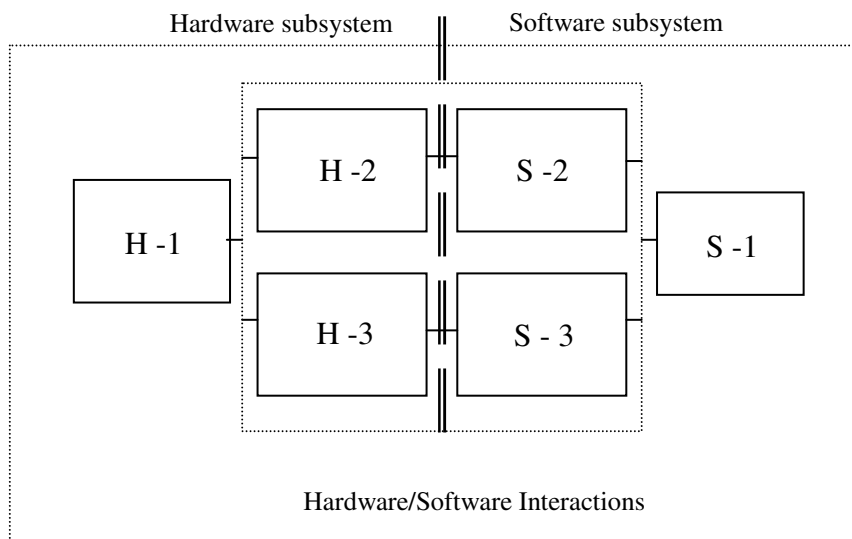
**Figure 11.12.** System failure categories

Following are the explanations of all modules in Figure 11.13.

#### Hardware Failures:

- H - 1: Total hardware component failures - whenever there is an H-1 event, the whole system will fail. These kinds of failures are "pure hardware failures".

- H - 2: Also known as *hardware degradation*. Only partial hardware fails (permanently), and the whole system does not fail necessarily but let the system work in the degraded state. It is related to S - 2, and possibly causes the software to fail in the state of “Hardware-related Software Failure”.
- H - 3: Temporary (Transient) hardware component failures - these hardware failures are usually caused by the disturbances from the operation environment.



**Figure 11.13.** System reliability diagram

### Software Failures

- S - 1: Pure software failures, they are caused by the faults in the software, which is not related to hardware system failures.
- S - 2: Permanent hardware-related software failures. When the hardware degrades (H-2), the system is liable to fail in S-2. These failures cannot be solved by simply redoing the computing tasks, and usually are related to hardware degradations. These failures are the major unknown failures in this paper. One can consider that the hardware-related software failures are caused by design faults that cannot deal with the potential partial hardware failures.
- S - 3: Transient hardware-related software failures. Although these failures are actually related to transient hardware failures, they can be avoided if the designer anticipates the hardware transients and designs fault-tolerance (such as Roll back scheme) into the software. Thus, hardware transients can be transferred into temporary hardware-related software failures.

### 11.5.2 Reliability Modeling

#### Assumptions (Teng *et al.* 2001)

1. An entire system fails whenever a total hardware failure (pure hardware failure), or a pure software failure or a hardware-related software failure (HW/SW interaction failure) happens.
2. Pure hardware failures and pure software failures are independent of each other.
3. Pure hardware failures and HW/SW interaction failures are independent of each other.
4. Pure software failures and hardware-related software failures (HW/SW) are independent of each other.
5. Hardware-related software failures (HW/SW) can be put into two categories: permanent HW/SW and transient HW/SW. They are also independent of each other.
6. Hardware components go to degradation (partial failure) with failure rate  $\lambda_1$ .
7. The partial hardware failure can be immediately detected with a probability  $p_1$ . Once a partial hardware failure is detected, it can be recovered using a software tool with a probability  $p_2$ .
8. An undetected degradation may cause a hardware-related software failure (fail unsafe) with rate  $\lambda_3$ , and a detected degradation may cause an execution abortion (fail safe) with rate  $\lambda_4$ .
9. Once a partial hardware failure is detected, the failed hardware components can be fixed or replaced at rate  $\mu_1$  (if recovered by software) and  $\mu_2$  (if not recovered by software).
10. Partial hardware failure can go further to the total failure state with either  $\lambda_{21}$ ,  $\lambda_{22}$  or  $\lambda_{23}$  (see Figure 11.14).
11. The transient hardware failure rate is  $\lambda_5$ .
12. Transient hardware failures are detected immediately with probability  $p_3$ . If detected, the transient failures are treated immediately by software methods, which recover the transient failures with probability  $p_4$ . If a transient failure is not successfully recovered, then the consequent new transient failure can still be detected since they are caused by the same kind of hardware transient problems.
13. If a transient hardware failure is not detected, the software will fail to give the correct result, and then the system will fail.
14. The maximum number of times that the software tries to recover a transient hardware failure is  $H$ . If  $H$  is exceeded, the system is considered failed.

The reliability of the entire system is

$$R_{System}(t) = R_s(t) R_h(t) R_{hs}(t) \quad (11.36)$$

where

$R_s(t)$  = reliability of software subsystem

$R_h(t)$  = reliability of hardware subsystem

$R_{hs}(t)$  = reliability of hardware-software interaction



### A. Hardware Subsystem

If we assume that the hardware system has the following Weibull hazard function:

$$h_h(t) = \frac{\gamma}{\theta} t^{\gamma-1} \quad (11.37)$$

then the hardware reliability function of the Weibull distribution  $R_h(t)$  is

$$R_h(t) = e^{-\frac{t^\gamma}{\theta}} \quad (11.38)$$

### B. Software Subsystem

From equation (9) in (Zhang *et al.* 2003) and if we assume that the error detection rate function  $b(t)$  is a non-decreasing function with inflexion S-shaped curve as follows

$$b(t) = \frac{c}{1 + \alpha e^{-bt}}$$

and the errors can be introduced during debugging,  $\beta$  being a constant error introduction rate, as

$$\beta(t) = \beta$$

then the mean value function and reliability of software subsystem are as follows:

$$m_s(t) = \frac{a}{p - \beta} \left( 1 - \left( \frac{(1 + \alpha)e^{-bt}}{1 + \alpha e^{-bt}} \right)^{\frac{c}{b}(p - \beta)} \right) \quad (11.39)$$

and

$$R_s(t) = e^{-m_s(t)} \quad (11.40)$$

respectively.

### C. Hardware-Software Interaction Modeling

*Notation*

- $\lambda_1$  Hazard rate for hardware subsystem to go to the degradation state.
- $\mu$  Repair rate after degradation is detected.
  - $\mu_1$  = repair rate at state 9a
  - $\mu_2$  = repair rate at state 9b
- $\lambda_2$  Hazard rate for degraded hardware subsystem to go to the total failure
  - $\lambda_{21}$  = hazard rate at state 9a
  - $\lambda_{22}$  = hazard rate at state 9b
  - $\lambda_{23}$  = hazard rate at state 10
- $\lambda_3$  Hazard rate from undetected hardware degradation to hardware-related software failure (fail unsafe)
- $\lambda_4$  Hazard rate from detected hardware degradation to abortion (fail safe)
- $\lambda_5$  Hazard rate for hardware transient failures

$\lambda_6$	Hazard rate from hardware transient to aborting operation
$p_1$	Probability that the hardware degradation is detected
$q_1$	Probability that the hardware degradation is undetected, $q_1 = 1 - p_1$
$p_2$	Probability that the degradation is recovered by software methods
$q_2$	Probability that the degradation is not recovered by software, $q_2 = 1 - p_2$
$p_3$	Probability that the transient is detected by some methods
$q_3$	Probability that the transient is not detected by software, $q_3 = 1 - p_3$
$H$	Maximal number of times that redo a task to fix a transient hardware failure.

*Permanent Hardware/Software Interactions Case.* If a hardware component goes to totally fail directly, then it is considered in the hardware reliability model; if a hardware component goes to partially fail first, then it is considered in this model.

#### Notation

$Q_0(t)$	Probability to stay in state 0 (system operation)
$Q_2(t)$	Probability to stay in state 2 (total hardware failure)
$Q_{9a}(t)$	Probability to stay in state 9a (hardware degradation-detected and software recovered)
$Q_{9b}(t)$	Probability to stay in state 9b (hardware degradation-detected and not software recovered)
$Q_{10}(t)$	Probability to stay in state 10 (hardware degradation not detected)
$Q_{11}(t)$	Probability to stay in state 11 (execution aborted)
$Q_{12}(t)$	Probability to stay in state 12 (permanent hardware-related software failure)

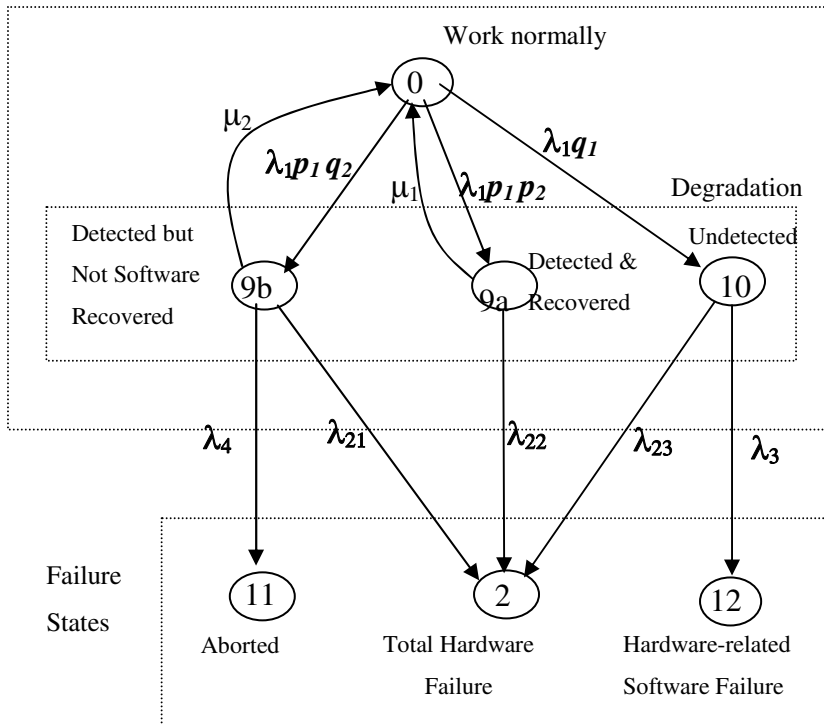
Figure 11.14 shows the system state transition diagram of the permanent HW/SW interactions based on assumptions 6-10. From the figure, we can easily obtain the following differential equations (Teng *et al.* 2001):

$$\begin{aligned}
 \dot{Q}_0(t) &= -\lambda_1 Q_0(t) + \mu_1 Q_{9a}(t) + \mu_2 Q_{9b}(t) \\
 \dot{Q}_2(t) &= \lambda_{22} Q_{9a}(t) + \lambda_{21} Q_{9b}(t) + \lambda_{23} Q_{10}(t) \\
 \dot{Q}_{9a}(t) &= \lambda_1 p_1 p_2 Q_0(t) - (\mu_1 + \lambda_{22}) Q_{9a}(t) \\
 \dot{Q}_{9b}(t) &= \lambda_1 p_1 q_2 Q_0(t) - (\mu_2 + \lambda_{21} + \lambda_4) Q_{9b}(t) \\
 \dot{Q}_{10}(t) &= \lambda_1 q_1 Q_0(t) - (\lambda_{23} + \lambda_3) Q_{10}(t) \\
 \dot{Q}_{11}(t) &= \lambda_4 Q_{9b}(t) \\
 \dot{Q}_{12}(t) &= \lambda_3 Q_{10}(t)
 \end{aligned} \tag{11.41}$$

Given the initial conditions

$$\begin{aligned}
 Q_0(0) &= 1 & Q_2(0) &= 0 & Q_{9a}(0) &= 0 & Q_{9b}(0) &= 0 \\
 Q_{10}(0) &= 0 & Q_{11}(0) &= 0 & Q_{12}(0) &= 0
 \end{aligned}$$

## Work States for Permanent HW/SW Interactions

**Figure 11.14.** State Transition Diagram Permanent Hardware/Software Interactions

It should be noted that among all these system states, state 2, 11 and 12 are failure states, while state 9a, 9b and 10 are degraded working states, and state 0 is the normal working state. Using the Laplace transform, we can easily obtain the solutions as follows:

$$Q_0(t) = \frac{(c_1 + b_1)(c_1 + b_2)}{(c_1 - c_2)(c_1 - c_3)} e^{c_1 t} + \frac{(c_2 + b_1)(c_2 + b_2)}{(c_2 - c_1)(c_2 - c_3)} e^{c_2 t} + \frac{(c_3 + b_1)(c_3 + b_2)}{(c_3 - c_1)(c_3 - c_2)} e^{c_3 t}$$

$$Q_{9a}(t) = \lambda_1 p_1 p_2 \left[ \frac{(c_1 + b_2)e^{c_1 t}}{(c_1 - c_2)(c_1 - c_3)} + \frac{(c_2 + b_2)e^{c_2 t}}{(c_2 - c_1)(c_2 - c_3)} + \frac{(c_3 + b_2)e^{c_3 t}}{(c_3 - c_1)(c_3 - c_2)} \right]$$

$$Q_{9b}(t) = \lambda_1 p_1 q_2 \left[ \frac{(c_1 + b_1)e^{c_1 t}}{(c_1 - c_2)(c_1 - c_3)} + \frac{(c_2 + b_1)e^{c_2 t}}{(c_2 - c_1)(c_2 - c_3)} + \frac{(c_3 + b_1)e^{c_3 t}}{(c_3 - c_1)(c_3 - c_2)} \right]$$

$$\begin{aligned}
Q_{10}(t) &= \lambda_1 q_1 \left( \frac{(c_1 + b_1)(c_1 + b_2)e^{c_1 t}}{(c_1 - c_2)(c_1 - c_3)(c_1 - c_4)} + \frac{(c_2 + b_1)(c_2 + b_2)e^{c_2 t}}{(c_2 - c_1)(c_2 - c_3)(c_2 - c_4)} \right. \\
&\quad \left. + \frac{(c_3 + b_1)(c_3 + b_2)e^{c_3 t}}{(c_3 - c_1)(c_3 - c_2)(c_3 - c_4)} + \frac{(c_4 + b_1)(c_4 + b_2)e^{c_4 t}}{(c_4 - c_1)(c_4 - c_2)(c_4 - c_3)} \right) \\
Q_{11}(t) &= \lambda_1 \lambda_4 p_1 q_2 \left( \frac{-b_1}{c_1 c_2 c_3} + \frac{(c_1 + b_1)e^{c_1 t}}{c_1(c_1 - c_2)(c_1 - c_3)} + \frac{(c_2 + b_1)e^{c_2 t}}{c_2(c_2 - c_1)(c_2 - c_3)} \right. \\
&\quad \left. + \frac{(c_3 + b_1)e^{c_3 t}}{c_3(c_3 - c_1)(c_3 - c_2)} \right) \\
Q_{12}(t) &= \lambda_4 \lambda_3 q_1 \left( \frac{b_1 b_2}{c_1 c_2 c_3 c_4} + \frac{(c_1 + b_1)(c_1 + b_2)e^{c_1 t}}{c_1(c_1 - c_2)(c_1 - c_3)(c_1 - c_4)} + \frac{(c_2 + b_1)(c_2 + b_2)e^{c_2 t}}{c_2(c_2 - c_1)(c_2 - c_3)(c_2 - c_4)} \right. \\
&\quad \left. + \frac{(c_3 + b_1)(c_3 + b_2)e^{-c_3 t}}{c_3(c_3 - c_1)(c_3 - c_2)(c_3 - c_4)} + \frac{(c_4 + b_1)(c_4 + b_2)e^{-c_4 t}}{c_4(c_4 - c_1)(c_4 - c_2)(c_4 - c_3)} \right) \\
Q_2(t) &= 1 - Q_0(t) - Q_{9a}(t) - Q_{9b}(t) - Q_{10}(t) - Q_{11}(t) - Q_{12}(t) \quad (11.42)
\end{aligned}$$

where

$$b_1 = \mu_1 + \lambda_{22}$$

$$b_2 = \mu_2 + \lambda_{21} + \lambda_4$$

$$c_4 = -(\lambda_{23} + \lambda_3) \text{ and}$$

$c_1, c_2$  and  $c_3$  are the roots of the following equation

$$(x + \lambda_1)(x + b_1)(x + b_2) = \lambda_1 p_1 [\mu_1 p_2 (x + b_2) + \mu_2 q_2 (x + b_1)]$$

The probability that the system will not fail in permanent mode due to hardware/software interactions is given by

$$\begin{aligned}
P_{PHS}(t) &= \Pr\{\text{No permanent HW/SW failures up to } t\} \\
&= Q_0(t) + Q_{9a}(t) + Q_{9b}(t) + Q_{10}(t) \quad (11.43)
\end{aligned}$$

Furthermore, the probability that the system fails safely by permanent HW/SW interactions is

$$P_{safe}(t) = Q_{11}(t)$$

The probability that the system fails unsafely by permanent HW/SW interactions is

$$P_{unsafe}(t) = Q_2(t) + Q_{12}(t)$$

*Transient Hardware/Software Interactions Case.* Figure 11.15 shows the diagram of the transient HW/SW in systems based on the assumptions 11-14. At state 3a, the transient can be recovered by software methods with probability  $p_4$ , and if the transient can not be recovered, then it goes back to state 3a itself with probability  $q_4 = (1 - p_4)$ . If the number of failed recovery attempts gets to H, then the recovery

software will abort the task, therefore the state goes from  $3a$  to  $4$ . In Figure 11.15, states  $3b$  and  $4$  are the failure states, and state  $0'$  is a working state, and state  $3a$  is a transient state. Assume that failed and aborted states are independent of each other, then from Teng (2001)

$$\begin{aligned}
 \dot{Q}_{0'}(t) &= -\lambda_5 Q_{0'}(t) + \lambda_6(1 - q_4^H) Q_{3a}(t) \\
 \dot{Q}_{3a}(t) &= \lambda_5 p_3 Q_{0'}(t) - \lambda_6 Q_{3a}(t) \\
 \dot{Q}_{3b}(t) &= \lambda_5 q_3 Q_{0'}(t) \\
 \dot{Q}_4(t) &= \lambda_6 q_4^H Q_{0'}(t)
 \end{aligned} \tag{11.44}$$

The solution of the system of equations in equation (11.44) can be easily obtained. Therefore, the probability that the system will not fail due to transient HW/SW failures is given by

$$\begin{aligned}
 P_{THS}(t) &= \Pr\{\text{No transient HW/SW Failures}\} \\
 &= Q_{0'}(t) + Q_{3a}(t)
 \end{aligned} \tag{11.45}$$

From equations (11.43) and (11.45), the reliability function for hardware and software interaction is as follows:

$$\begin{aligned}
 P_{hs}(t) &= \Pr\{\text{No HW/SW failures}\} \\
 &= \Pr\{\text{No permanent failures}\} \times \Pr\{\text{No transient failures}\} \\
 &= P_{PHS}(t) P_{THS}(t)
 \end{aligned}$$

That is,

$$P_{hs}(t) = [Q_{0'}(t) + Q_{3a}(t) + Q_{3b}(t) + Q_{10}(t)] (Q_{0'}(t) + Q_{3a}(t)) \tag{11.46}$$

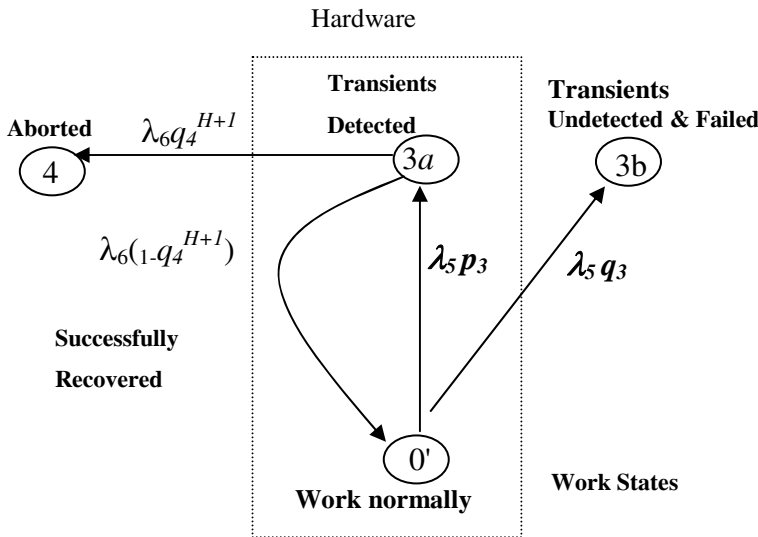


Figure 11.15. Transient failures transition diagram

Therefore, the system reliability considering hardware, software and the interactions between them is as follows

$$\begin{aligned}
 R_{\text{System}}(t) &= R_h(t) R_s(t) P_{hs}(t) \\
 &= e^{-\frac{t^\gamma}{\theta}} e^{-m(t)} [Q_0(t) + Q_{9a}(t) + Q_{9b}(t) + Q_{10}(t)](Q_0(t) + Q_3(t))
 \end{aligned}
 \quad (11.47)$$

## 11.6 Application Example

This section illustrates the hardware and software modeling approach by applying it to a telecommunication application data. Table 11.5 shows exposure time and failure data collected from the field for a particular product that supports voice and data communication. Detail information can be obtained in Teng *et al.* (2001).

The hardware failures shown in Table 11.5 represent the frequency of server failures, irrespective of which particular component of the server failed. Table 11.5 does not explicitly show the HW/SW failures, but the assumption in Teng *et al.* (2001) that 15% of the reported HW failures were actually HW/SW failures. Table 11.6 represents a revised Table 11.5 with consideration of 15% of HW/SW failures.

**Table 11.5.** Failures in a telecommunication application

Month	Software exposure time (system days)	Software failures	Hardware exposure time (system days)	Hardware failures
1	961	4	9,843	23
2	4,170	1	10,290	32
3	8,789	5	11,254	32
4	11,858	4	12,385	21
5	13,110	3	13,155	44
6	14,198	1	14,198	55
Total	53,086	18	71,125	207

**Table 11.6.** HW/SW interaction failures data set

Month	Software exposure time (system days)	Software failures	Hardware exposure time (system days)	Hardware failures	HW/SW failures
1	961	4	9,843	20	3
2	4,170	1	10,290	27	5
3	8,789	5	11,254	27	5
4	11,858	4	12,385	18	3
5	13,110	3	13,155	37	7
6	14,198	1	14,198	47	8
Total	53,086	18	71,125	176	31

### Pure Software Failures

Note that from equation (11.39), the mean value function is given by

$$m(t) = \frac{a}{p - \beta} \left( 1 - \left( \frac{(1 + \alpha)e^{-bt}}{1 + \alpha e^{-bt}} \right)^{\frac{c}{b}(p - \beta)} \right)$$

where

$a$  is the expected number of initial faults in the software

$c$  is the average per fault failure rate

$\beta$  is the probability of faults being introduced during the debugging processes

$p$  is the probability of faults being successfully removed

$\alpha$  is a shape parameter that represents the learning curve in the debugging process

$b$  is the rate parameter for the learning curve.

Assuming the value  $p = 0.95$ , the MLEs for all unknown parameters are  $\hat{a} = 17.97$ ,  $\hat{b} = 0.000001$ ,  $\hat{c} = 0.000001$ ,  $\hat{\alpha} = 0$ ,  $\hat{\beta} = 0$ . Since  $\alpha$  and  $\beta$  are zero, the resulting model becomes the Goel-Okumoto model with imperfect debugging. It seems that the Goel-Okumoto model fits the given data set well.

### Pure Hardware Failures

Assuming that 85% of the hardware failures reported in a given month are pure hardware. We use the Weibull model (Equation 11.38) for pure hardware failures. The data shown in Table 11.5 represents the data collected on release number two of the product. The MLE for  $\theta$  and  $\gamma$  in equation (11.38) are as follows:

$$\hat{\gamma} = 1.14 \quad \hat{\theta} = 1949$$

where the hardware reliability function is

$$R_h(t) = e^{-\frac{t^{1.14}}{1949}}$$

### HW/SW Interactions

To simplify the computation, Teng *et al* (2001) considers only permanent HW/SW failures and also assume that  $p_1 = 0$ ,  $\lambda_{23} = 0$ ,  $\lambda_1 = 0.1 \lambda_3$  and  $\lambda_3 = 0.0048$  failures/day. The probability that the system will not fail due to hardware and software interactions at time  $x$  is as follows:

$$P_{hs}(x) = \frac{\lambda_3 e^{-\lambda_1 x} - \lambda_1 e^{-\lambda_3 x}}{\lambda_3 - \lambda_1}$$

### Entire System Reliability Function

Considering a new installation that will be delivered at time  $t = 53,086$  days, the entire system reliability, combining the HW, SW and HW/SW models together, between  $t = 53,086$  and  $t = 53,086 + x$  (where  $x$  is the mission time) is given by (Teng *et al*. 2001)

$$\begin{aligned}
 R_{\text{system}}(x | t = 53,086) &= R_h(x) \cdot R_s(x | t = 53,086) \cdot R_{hs}(x) \\
 &= e^{\frac{x^r}{\theta}} \cdot e^{-[m(53,086+x)-m(53,086)]} \cdot \frac{\lambda_3 e^{-\lambda_1 x} - \lambda_1 e^{-\lambda_3 x}}{\lambda_3 - \lambda_1}
 \end{aligned}$$

The estimated conditional reliability function  $R_{\text{system}}(x)$ , and each of its three components are plotted in Figure 11.16. Figure 11.17 shows the comparison between two reliability function curves  $R_{\text{ind}}(t)$  and  $R_{\text{system}}(t)$ , where  $R_{\text{ind}}(t)$  is the reliability model which does not include effects of hardware/software interactions:

$$R_{\text{ind}}(t) = R_h(t) \quad R_s(t) = R_{\text{system}}(t) / R_{hs}(t)$$

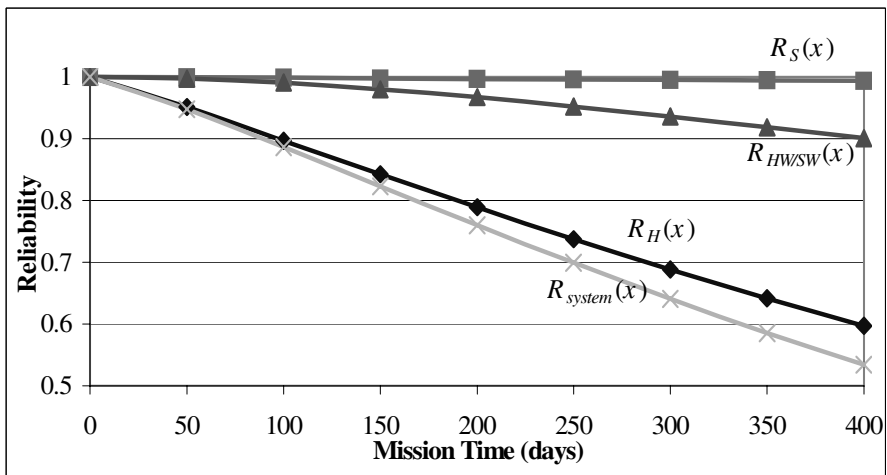


Figure 11.16. System reliability function

## 11.7 Further Reading

Some interesting research papers and book on fault tolerant software systems are:

Y. Jiang, J. Li and Shoichi Nishimura, "A general stochastic model for dynamic locking in database systems," IEEE Trans on Computers, vol 53, no 3, 2004

T. Clouqueur, K.K. Saluja, and P. Ramanathan, "Fault tolerance in collaborative sensor networks for target detection," IEEE Trans on Computers, vol 53, no 3, 2004



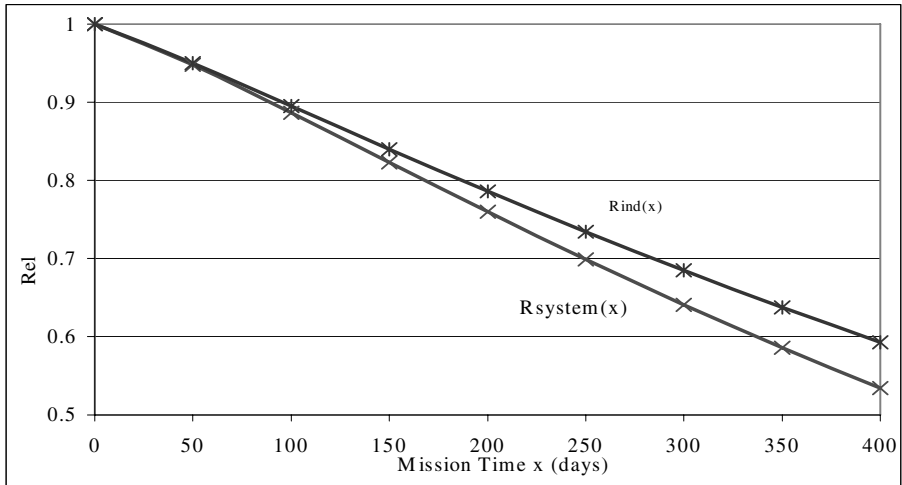


Figure 11.17. System reliability comparisons

## 11.8 Problems

1. The cost of the fault-tolerant software for a new product (XZY) includes development and design, testing, implementation, and operation costs. In general, the reliability of the system can be increased by adding more redundant programs or modules. However, the extra cost and complexity may not justify the small gains in reliability. Let us consider the following problem.

Suppose  $P_h(s)$  is the probability of failure for the hybrid scheme based on configurations, then  $[1 - P_h(s)]$  is the reliability of the fault-tolerant software. Let

$C$  = the total amount of resources available

$C_{ei}$  = the amount of resources needed for program version  $i$

$C_{vj}$  = the resources needed for voting version  $j$

$C_{tk}$  = the amount of resources needed for testing version  $k$ .

(a) Show that the reliability optimization model, given all the information above, can be formulated as follows:

$$\text{Objective } \max_{s \in S} [1 - P_h(s)]$$

$$\begin{aligned} \text{Subject to } & \sum_i C_{ei} + \sum_j C_{vj} + \sum_k C_{tk} \leq C \\ & C_{ei} \geq 0, C_{vj} \geq 0, C_{tk} \geq 0 \end{aligned}$$

where  $S$  is a set of all the possible configurations of the hybrid scheme.

(b) Develop a heuristic algorithm to determine the optimal solution of the above optimization problem.

2. Continuing with Problem 1, assume that all programs and their testing versions have the same reliability and costs. Given that

Cost of a program version  $C_e = \$15,000$

Cost of test  $C_t = 85\%$  of  $C_e = \$12,750$

Cost of voter  $C_v = 10\%$  of  $C_e = \$1,500$

Total amount or resources available  $C = \$120,000$

Probability of program version failure  $e = 0.05$

Probability of test failure  $t = 0.02$

Probability of voter failure  $d = 0.002$ ,

calculate the system reliability and cost of each of the following possible configurations:

- (a) 7 RB
- (b) 7 NVP
- (c) 3 RB 2 NVP
- (d) 2 NVP 3 RB
- (e) 2 RB 3 NVP

# Appendix 1

---

## Distribution Tables

**Table A1.1.** Cumulative areas under the standard normal distribution

Z	0	1	2	3	4	5	6	7	8
-3.0	.0013	.0010	.0007	.0005	.0003	.0002	.0002	.0001	.0001
-2.9	.0019	.0018	.0017	.0017	.0016	.0016	.0015	.0015	.0014
-2.8	.0026	.0025	.0024	.0023	.0023	.0022	.0021	.0021	.0020
-2.7	.0035	.0034	.0033	.0032	.0031	.0030	.0029	.0028	.0027
-2.6	.0047	.0045	.0044	.0043	.0041	.0040	.0039	.0038	.0037
-2.5	.0062	.0060	.0059	.0057	.0055	.0054	.0052	.0051	.0049
-2.4	.0082	.0080	.0078	.0075	.0073	.0071	.0069	.0068	.0066
-2.3	.0107	.0104	.0102	.0099	.0096	.0094	.0091	.0089	.0087
-2.2	.0139	.0136	.0132	.0129	.0126	.0122	.0119	.0116	.0113
-2.1	.0179	.0174	.0170	.0166	.0162	.0158	.0154	.0150	.0146
-2.0	.0228	.0222	.0217	.0212	.0207	.0202	.0197	.0192	.0188
-1.9	.0287	.0281	.0274	.0268	.0262	.0256	.0250	.0244	.0238
-1.8	.0359	.0352	.0344	.0336	.0329	.0322	.0314	.0307	.0300
-1.7	.0446	.0436	.0427	.0418	.0409	.0401	.0392	.0384	.0375
-1.6	.0548	.0537	.0526	.0516	.0505	.0495	.0485	.0475	.0465
-1.5	.0668	.0655	.0643	.0630	.0618	.0606	.0594	.0582	.0570
-1.4	.0808	.0793	.0778	.0764	.0749	.0735	.0722	.0708	.0694
-1.3	.0968	.0951	.0934	.0918	.0901	.0885	.0869	.0853	.0838
-1.2	.1151	.1131	.1112	.1093	.1075	.1056	.1038	.1020	.1003
-1.1	.1357	.1335	.1314	.1292	.1271	.1251	.1230	.1210	.1190
-1.0	.1587	.1562	.1539	.1515	.1492	.1469	.1446	.1423	.1401
-0.9	.1841	.1814	.1788	.1762	.1736	.1711	.1685	.1660	.1635
-0.8	.2119	.2090	.2061	.2033	.2005	.1977	.1949	.1922	.1894
-0.7	.2420	.2389	.2358	.2327	.2297	.2266	.2236	.2206	.2177
-0.6	.2743	.2709	.2676	.2643	.2611	.2578	.2546	.2514	.2483
-0.5	.3085	.3050	.3015	.2981	.2946	.2912	.2877	.2843	.2810
-0.4	.3446	.3409	.3372	.3336	.3300	.3264	.3228	.3192	.3156
-0.3	.3821	.3783	.3745	.3707	.3669	.3632	.3594	.3557	.3520

Table A1.1. (continued)

Z	0	1	2	3	4	5	6	7	8
-0.2	.4207	.4168	.4129	.4090	.4052	.4013	.3974	.3936	.3897
-0.1	.4602	.4562	.4522	.4483	.4443	.4404	.4364	.4325	.4286
-0.0	.5000	.4960	.4920	.4880	.4840	.4801	.4761	.4721	.4681
0.0	.5000	.5040	.5080	.5120	.5160	.5199	.5239	.5279	.5319
0.1	.5398	.5438	.5478	.5517	.5557	.5596	.5636	.5675	.5714
0.2	.5793	.5832	.5871	.5910	.5948	.5987	.6026	.6064	.6103
0.3	.6179	.6217	.6255	.6293	.6331	.6368	.6406	.6443	.6480
0.4	.6554	.6591	.6628	.6664	.6700	.6736	.6772	.6808	.6844
0.5	.6915	.6950	.6985	.7019	.7054	.7088	.7123	.7157	.7190
0.6	.7257	.7291	.7324	.7357	.7389	.7422	.7454	.7486	.7517
0.7	.7580	.7611	.7642	.7673	.7703	.7734	.7764	.7794	.7823
0.8	.7881	.7910	.7939	.7967	.7995	.8023	.8051	.8078	.8106
0.9	.8159	.8186	.8212	.8238	.8264	.8289	.8315	.8340	.8365
1.0	.8413	.8438	.8461	.8485	.8508	.8531	.8554	.8577	.8599
1.1	.8643	.8665	.8686	.8708	.8729	.8749	.8770	.8790	.8810
1.2	.8849	.8869	.8888	.8907	.8925	.8944	.8962	.8980	.8997
1.3	.9032	.9049	.9066	.9082	.9099	.9115	.9131	.9147	.9162
1.4	.9192	.9207	.9222	.9236	.9251	.9265	.9278	.9292	.9306
1.5	.9332	.9345	.9357	.9370	.9382	.9394	.9406	.9418	.9430
1.6	.9452	.9463	.9474	.9484	.9495	.9505	.9515	.9525	.9535
1.7	.9554	.9564	.9573	.9582	.9591	.9599	.9608	.9616	.9625
1.8	.9641	.9648	.9656	.9664	.9671	.9678	.9686	.9693	.9700
1.9	.9713	.9719	.9726	.9732	.9738	.9744	.9750	.9756	.9762
2.0	.9772	.9778	.9783	.9788	.9793	.9798	.9803	.9808	.9812
2.1	.9821	.9826	.9830	.9834	.9838	.9842	.9846	.9850	.9854
2.2	.9861	.9864	.9868	.9871	.9874	.9878	.9881	.9884	.9887
2.3	.9893	.9896	.9898	.9901	.9904	.9906	.9909	.9911	.9913
2.4	.9918	.9920	.9922	.9925	.9927	.9929	.9931	.9932	.9934
2.5	.9938	.9940	.9941	.9943	.9945	.9946	.9948	.9949	.9951
2.6	.9953	.9955	.9956	.9957	.9959	.9960	.9961	.9962	.9963
2.7	.9965	.9966	.9967	.9968	.9969	.9970	.9971	.9972	.9973
2.8	.9974	.9975	.9976	.9977	.9977	.9978	.9979	.9979	.9980
2.9	.9981	.9982	.9982	.9983	.9984	.9984	.9985	.9985	.9986
3.0	.9987	.9990	.9993	.9995	.9997	.9998	.9998	.9999	.9999

**Table A1.2.** Percentage points of the  $t$ -distribution

$\nu \setminus \alpha$	0.100	0.050	0.025	0.01	0.005	0.001
1	3.078	6.314	12.706	31.821	63.657	318.310
2	1.886	2.920	4.303	6.965	9.925	23.326
3	1.638	2.353	3.182	4.541	5.841	10.213
4	1.533	2.132	2.776	3.747	4.604	7.173
5	1.476	2.015	2.571	3.365	4.032	5.893
6	1.440	1.943	2.447	3.143	3.707	5.208
7	1.415	1.895	2.365	2.998	3.499	4.785
8	1.397	1.860	2.306	2.896	3.355	4.501
9	1.383	1.833	2.262	2.821	3.250	4.297
10	1.372	1.812	2.228	2.764	3.169	4.144
11	1.363	1.796	2.201	2.718	3.106	4.025
12	1.356	1.782	2.179	2.681	3.055	3.930
13	1.350	1.771	2.160	2.650	3.012	3.852
14	1.345	1.761	2.145	2.624	2.977	3.787
15	1.341	1.753	2.131	2.602	2.947	3.733
16	1.337	1.746	2.120	2.583	2.921	3.686
17	1.333	1.740	2.110	2.567	2.898	3.646
18	1.330	1.734	2.101	2.552	2.878	3.610
19	1.328	1.729	2.093	2.539	2.861	3.579
20	1.325	1.725	2.086	2.528	2.845	3.552
21	1.323	1.721	2.080	2.518	2.831	3.527
22	1.321	1.717	2.074	2.508	2.819	3.505
23	1.319	1.714	2.069	2.500	2.807	3.485
24	1.318	1.711	2.064	2.492	2.797	3.467
25	1.316	1.708	2.060	2.485	2.787	3.450
26	1.315	1.706	2.056	2.479	2.779	3.435
27	1.314	1.703	2.052	2.473	2.771	3.421
28	1.313	1.701	2.048	2.467	2.763	3.408
29	1.311	1.699	2.045	2.462	2.756	3.396
30	1.310	1.697	2.042	2.457	2.750	3.385
40	1.303	1.684	2.021	2.423	2.704	3.307
60	1.296	1.671	2.000	2.390	2.660	3.232
120	1.289	1.658	1.980	2.358	2.617	3.160
$\infty$	1.282	1.645	1.960	2.326	2.576	3.090

**Table A1.3.** Percentage points of the chi-squared distribution

$\nu \backslash \chi^2_\alpha$	$\chi^2_{.99}$	$\chi^2_{.975}$	$\chi^2_{.95}$	$\chi^2_{.90}$	$\chi^2_{.10}$	$\chi^2_{.05}$	$\chi^2_{.025}$	$\chi^2_{.01}$
1	0	0.00	0.00	0.02	2.71	3.84	5.02	6.64
2	0.02	0.05	0.10	0.21	4.61	5.99	7.38	9.21
3	0.12	0.22	0.35	0.58	6.25	7.82	9.35	11.35
4	0.30	0.48	0.71	1.06	7.78	9.49	11.14	13.28
5	0.55	0.83	1.15	1.61	9.24	11.07	12.83	15.09
6	0.87	1.24	1.64	2.20	10.65	12.59	14.45	16.81
7	1.24	1.69	2.17	2.83	12.02	14.07	16.01	18.48
8	1.65	2.18	2.73	3.49	13.36	15.51	17.54	20.09
9	2.09	2.70	3.33	4.17	14.68	16.92	19.02	21.67
10	2.56	3.25	3.94	4.87	15.99	18.31	20.48	23.21
11	3.05	3.82	4.58	5.58	17.28	19.68	21.92	24.73
12	3.57	4.40	5.23	6.30	18.55	21.92	23.34	26.22
13	4.11	5.01	5.89	7.04	19.81	22.36	24.74	27.69
14	4.66	5.63	6.57	7.79	21.06	23.69	26.12	29.14
15	5.23	6.26	7.26	8.57	22.31	25.00	27.49	30.58
16	5.81	6.91	7.96	9.31	23.54	26.30	28.85	32.00
17	6.41	7.56	8.67	10.09	24.77	27.59	30.19	33.41
18	7.02	8.23	9.39	10.87	25.99	28.87	31.53	34.81
19	7.63	8.91	10.12	11.65	27.20	30.14	32.85	36.19
20	8.26	9.59	10.85	12.44	28.41	31.41	34.17	37.57
21	8.90	10.28	11.59	13.24	29.62	32.67	35.48	38.93
22	9.54	10.98	12.34	14.04	30.81	33.92	36.78	40.29
23	10.20	11.69	13.09	14.85	32.01	35.17	38.08	41.64
24	10.86	12.40	13.85	15.66	33.20	36.42	39.36	42.98
25	11.52	13.12	14.61	16.47	34.38	37.65	40.65	44.31
26	12.20	13.84	15.38	17.29	35.56	38.89	41.92	45.64
27	12.88	14.57	16.15	18.11	36.74	40.11	43.19	46.96
28	13.57	15.31	16.93	18.94	37.92	41.34	44.46	48.28
29	14.26	16.05	17.71	19.77	39.09	42.56	45.72	49.59
30	14.95	16.79	18.49	20.60	40.26	43.77	46.98	50.89
35	18.48	20.56	22.46	24.81	46.03	49.80	53.21	57.36
40	22.14	24.42	26.51	29.07	51.78	55.76	59.35	63.71
50	29.69	32.35	34.76	37.71	63.14	67.50	71.42	76.17
60	37.47	40.47	43.19	46.48	74.37	79.08	83.30	88.39
70	45.43	48.75	51.74	55.35	85.50	90.53	95.03	100.44
80	53.53	57.15	60.39	64.30	96.55	101.88	106.63	112.34
90	61.74	65.64	69.12	73.31	107.54	113.15	118.14	124.13
100	70.05	74.22	77.93	82.38	118.47	124.34	129.57	135.81

**Table A1.4.** Critical values  $d_n, \alpha$  for the Kolmogorov-Smirnov test

$n \backslash \alpha$	0.2	0.1	0.05	0.02	0.01
1	0.900	0.950	0.975	0.990	0.995
2	0.684	0.776	0.842	0.900	0.929
3	0.565	0.636	0.708	0.785	0.829
4	0.493	0.565	0.624	0.689	0.734
5	0.447	0.509	0.563	0.627	0.669
6	0.410	0.468	0.519	0.577	0.617
7	0.381	0.436	0.483	0.538	0.576
8	0.358	0.410	0.454	0.507	0.542
9	0.339	0.387	0.430	0.480	0.513
10	0.323	0.369	0.409	0.457	0.489
11	0.308	0.352	0.391	0.437	0.468
12	0.296	0.338	0.375	0.419	0.449
13	0.285	0.325	0.361	0.404	0.432
14	0.275	0.314	0.349	0.390	0.418
15	0.266	0.304	0.338	0.377	0.404
16	0.258	0.295	0.327	0.366	0.392
17	0.250	0.286	0.318	0.355	0.381
18	0.244	0.279	0.309	0.346	0.371
19	0.237	0.271	0.301	0.337	0.361
20	0.232	0.265	0.294	0.329	0.352
21	0.226	0.259	0.287	0.321	0.344
22	0.221	0.253	0.281	0.314	0.337
23	0.216	0.247	0.275	0.307	0.330
24	0.212	0.242	0.264	0.301	0.323
25	0.208	0.238	0.264	0.295	0.317
26	0.204	0.233	0.259	0.290	0.311
27	0.200	0.229	0.254	0.284	0.305
28	0.197	0.225	0.250	0.279	0.300
29	0.193	0.221	0.246	0.275	0.295
30	0.190	0.218	0.242	0.270	0.281

## Appendix 2

---

### Laplace Transform

If a function  $h(x)$  can be obtained from some prescribed operation on a function  $f(x)$ , then  $h(x)$  is often called a transform of  $f(x)$ . For example,

$$h(x) = \sqrt{2 + f(x)}$$

$$h(x) = \frac{\partial}{\partial x} f(x)$$

The Laplace transform of  $f(t)$  is the function  $f^*(s)$  where

$$f^*(s) = \int_0^{\infty} e^{-st} f(t) dt$$

Often the Laplace transform is denoted as  $f^*(s)$  or  $\mathcal{L}(f(t))$  or  $\mathcal{L}(f)$ . The results of the Laplace transform for a few simple functions are presented below.

#### Results

$$1. \mathcal{L}(1) = \int_0^{\infty} e^{-st} dt = \frac{1}{s}$$

$$\begin{aligned} 2. \mathcal{L}(e^{-at}) &= \int_0^{\infty} e^{-st} e^{-at} dt = \int_0^{\infty} e^{-(s+a)t} dt \\ &= \frac{1}{s+a} \end{aligned}$$



3. If  $f(t) = \frac{1}{a} e^{-\frac{t}{a}}$ , then

$$\mathcal{L}(f(t)) = \int_0^{\infty} e^{-st} \frac{1}{a} e^{-\frac{t}{a}} dt = \frac{1}{1+sa}$$

4. If  $f(t) = te^{at}$ , then

$$\mathcal{L}(f(t)) = \int_0^{\infty} e^{-st} te^{at} dt = \frac{1}{(s-a)^2}$$

5. If  $f(t) = \frac{1}{a}(e^{at} - 1)$ , then

$$\mathcal{L}(f(t)) = \int_0^{\infty} e^{-st} \frac{1}{a}(e^{at} - 1) dt = \frac{1}{s(s-a)}$$

6. If  $f(t) = (1+at)e^{at}$ , then

$$\mathcal{L}(f(t)) = \int_0^{\infty} e^{-st} (1+at)e^{at} dt = \frac{s}{(s-a)^2}$$

Similarly, we can obtain the following results:

7. If  $f(t) = \frac{ae^{at} - be^{bt}}{a-b}$ , then

$$\mathcal{L}(f(t)) = \frac{s}{(s-a)(s-b)} \quad \text{for } a \neq b$$

8. If  $f(t) = \frac{\alpha^k t^{k-1} e^{-at}}{\Gamma(k)}$  then

$$\mathcal{L}(f(t)) = \left( \frac{\alpha}{\alpha+s} \right)^k$$

9. If  $f(t) = \frac{e^{at} - e^{bt}}{a-b}$ , for  $a \neq b$ , then

$$\mathcal{L}(f(t)) = \frac{1}{(s-a)(s-b)}$$

10. If  $f(t) = \lambda e^{-\lambda t}$ , then

$$\mathfrak{L}(f(t)) = \frac{\lambda}{\lambda + s}$$

$$\begin{aligned} 11. \quad \mathfrak{L}(c_1 f_1(t) + c_2 f_2(t)) &= \int_0^{\infty} e^{-st} [c_1 f_1(t) + c_2 f_2(t)] dt \\ &= c_1 \mathfrak{L}(f_1(t)) + c_2 \mathfrak{L}(f_2(t)) \end{aligned}$$

12. If  $f_i(t) = \lambda_i e^{-\lambda_i t}$ , then

$$\mathfrak{L}\left(\sum_{i=1}^n f_i(t)\right) = \sum_{i=1}^n \frac{\lambda_i}{\lambda_i + s}$$

$$13. \quad \mathfrak{L}\left(\sum_{i=1}^n f_i(t)\right) = \sum_{i=1}^n \mathfrak{L}(f_i(t))$$

$$\begin{aligned} 14. \quad \mathfrak{L}(f'(t)) &= \int_0^{\infty} e^{-st} f'(t) dt \\ &= f(t)e^{-st} \Big|_0^{\infty} + s \int_0^{\infty} f(t)e^{-st} dt \\ &= -f(0^+) + s f^*(s) \\ &= -f(0^+) + s \mathfrak{L}(f(t)) \end{aligned}$$

## Appendix 3

---

### Survey of Factors that Affect Software Reliability

Name \_\_\_\_\_  
Institution/company \_\_\_\_\_

This is a survey questionnaire concerning software reliability and environmental factors involved in the software development process. The environmental factors here include characteristics of the software itself (*e.g.*, size), the development environment (*e.g.*, people and tools), and all other factors during the whole software development process. The software reliability models that use testing time as the only influence factor may not be appropriate for the evaluation of the software reliability. In this study, we are interested in obtaining your opinion about the impact of environmental factors on software reliability, as well as some background information about you in order to keep your answer in perspective.

Please read the following paragraphs below and then answer all questions on the following pages. Section A is the survey deals with the issue of software reliability and environmental factors. The definitions of the factors are provided at the end of section A. Section B is some background information about you.

Please rank the following environmental factors in terms of identifying the significance of including them in the software reliability analysis. For example, if you think that "program complexity" is an extremely important factor, you should rank it at a level of "7". In contrast, if you think it will not improve the assessment of software reliability at all, you may rank it at "1". Each factor can take an integer value from "0" to "7". Please do not omit any ranking. Thank you for your cooperation.

Please return this survey to: Software Engineer

<b>Section A. Environmental Factors</b>	<b>Not significant</b>		<b>Extremely significant</b>				<b>No opinion</b>	
<b>General</b>								
1. Program complexity( <i>e.g.</i> , size,)	1	2	3	4	5	6	7	0
2. Program categories ( <i>e.g.</i> , database, operating system, <i>etc.</i> )	1	2	3	4	5	6	7	0
3. Difficulty of programming	1	2	3	4	5	6	7	0
4. Amount of programming effort	1	2	3	4	5	6	7	0
5. Level of programming technologies	1	2	3	4	5	6	7	0
6. Percentage of reused modules	1	2	3	4	5	6	7	0
7. Programming language	1	2	3	4	5	6	7	0
<b>Analysis and Design</b>								
8. Frequency of program specification change	1	2	3	4	5	6	7	0
9. Volume of program design documents	1	2	3	4	5	6	7	0
10. Design methodology	1	2	3	4	5	6	7	0
11. Requirements analysis	1	2	3	4	5	6	7	0
12. Relationship of detailed design to requiremt	1	2	3	4	5	6	7	0
13. Work standards	1	2	3	4	5	6	7	0
14. Development management	1	2	3	4	5	6	7	0
<b>Coding</b>								
15. Programmer skill	1	2	3	4	5	6	7	0
16. Programmer organization	1	2	3	4	5	6	7	0
17. Development team size	1	2	3	4	5	6	7	0
18. Program workload(stress)	1	2	3	4	5	6	7	0
19. Domain knowledge	1	2	3	4	5	6	7	0
20. Human nature(mistake and work omission)	1	2	3	4	5	6	7	0
<b>Testing</b>								
21. Testing environment(duplication of product)	1	2	3	4	5	6	7	0
22. Testing effort	1	2	3	4	5	6	7	0
23. Testing resource allocation	1	2	3	4	5	6	7	0
24. Testing methodologies	1	2	3	4	5	6	7	0
25. Testing coverage	1	2	3	4	5	6	7	0
26. Testing tools	1	2	3	4	5	6	7	0
27. Documentation	1	2	3	4	5	6	7	0
<b>Hardware systems</b>								
28. Processors	1	2	3	4	5	6	7	0
29. Storage devices	1	2	3	4	5	6	7	0
30. Input/output devices	1	2	3	4	5	6	7	0
31. Telecommunication devices	1	2	3	4	5	6	7	0
32. System software	1	2	3	4	5	6	7	0

To what extent do you believe that the environmental factors will improve the accuracy of the software reliability assessment?

10% 20% 40% 60% 80% 100%

## Section B. Background Information

B1. What kind of applications do you usually develop for?

☐ Safety-critical ☐ Commercial ☐ Inside users-oriented

B2. What type of software development experience do you have?

☐ Database ☐ Operation system ☐ Communication control ☐ Language processor

B3. Number of years have you been working on software development \_\_\_\_\_

B4. Your title/position

☐ Manager ☐ System engineer ☐ Programmer ☐ Tester ☐ Administrator ☐ Other

B5. Average percentage of the development time in your group spent on

Analysis phase: \_\_\_\_\_

Design phase: \_\_\_\_\_

Coding phase: \_\_\_\_\_

Testing phase: \_\_\_\_\_

Total: 100%

B6. Average percentage of reusable code in your software applications:

\_\_\_\_\_

Please complete all rankings and return to the researchers. Thank you!

## Definitions of the Environmental Factors

The environmental factors can be defined and measured as follows.

### Program Complexity

McCabe's V(G), Halstead's E and program size are well known complexity measures. There exists a significant relationship between each of these measures. However, it has been proved that none of these seemed significantly better than program size. For this reason, program size (Kiloline of code: KLOC) is used as a measure of program complexity. The level of this factor is "High", meaning the program size is greater than 50 KLOC, otherwise the level is "Low".

### Program Categories

The program categories indicate system complexity. There are four program categories: operating system, communication control program, data base management system, and languages processor

### Amount of Programming Effort

The deliberate programming effort may be regarded as effective for reducing the number of errors made. This is calculated in man years.

### Difficulty of Programming (PDIF)

Difficulty of programming is defined according to the Putnum as follows:

$$PDIF = \frac{k}{t^2} \quad (\text{man years/years}^2)$$

where

$k$  = the amount of programming effort

$t$  = the amount of programming time

### Level of Programming Technologies (TLVL)

The programming technologies are classified into four categories: design techniques, documentation techniques, programming techniques (including programming languages), and development computer access environment. Each of the categories had a rating scale (low, middle, and high) and the rating scores  $T=0.8, 1, 1.2$ , respectively, are allocated. These rating scores were determined by referring to Boehm's cost driver productivity range. The final TLVL is computed as follows by using the rating scores:

$$TLVL = \sum_{i=1}^4 T_i$$

where  $T_i$  = the rating score of category  $i$ .

### Percentage of Reused Code (PORC)

When people develop some new software products or when they update the old version of their software products, they usually keep some of the modules of the

code which can be reused, and add in some new ones. That is why it is important to include the measure:

$$\text{PORC} = \frac{S_o}{S_N + S_o}$$

where

$S_o$  = kiloline of code for the existing modules

$S_N$  = kiloline of code for new modules

### **Programming Languages**

Different programming languages have different complexity and structure; therefore there is the possibility for different languages to introduce errors are different.

### **Frequency of Program Specification Change (SCHG)**

SCHG is calculated from the number of pages of problem reports generated to change the program design specifications during programming phase of development.

### **Volume of Program Design Documents (DOCC)**

Program design documents that lack sufficient contents for the program produce errors. DOCC is calculated from the number of pages of new and modified program design documents.

### **Design Methodologies**

Different design methodologies for the same software may have different impact on the quality of the final software products. There are two types of design methodologies: structured design and functional design.

### **Requirements Analysis**

Requirements are provided by customers. Based on the requirements, software developers generate specifications. Usually customers and developers meet to verify the requirements and achieve understanding of the problems. Requirement analysis is necessary following design and coding work.

### **Relationship of Detailed Design to Requirements**

At the end of the design phase, detailed design is compared with requirements. Inspections are performed to verify whether the functions designed meet the requirements. Modifications can be made to remove the misunderstanding between the customers and developers.

### **Work Standard**

Work standard is the norm the developing team needs to obey. This could be company standard or group standard. Work standard indicates products to be made at each phase, design document format, design document description level and content, and the items to be checked in verifying the design documents. Having this kind of norms or not has an impact on the quality of software products.

### Development Management

Development management includes all the organization and decision-making activities. From the specification phase to design, code, and testing and even operational phase, development managers schedule the meeting time, keep all participants in touch and keep track of the development progress and work standards, give instructions, make decisions.

### Programmer Skills (PSKL)

PSKL has direct impact on the reliability of software products. PSKL can be defined as the average number of years of programming experience of programmers:

$$\text{PSKL} = \frac{\sum_{i=1}^n I_i}{n} \text{ (years)}$$

where

$I_i$  = number of years experience of programmer  $I$

$n$  = the total number of programmers

### Programmer Organization (ICON)

ICON is defined as the percentage of high-quality programmers. ICON is computed as follows:

$$\text{ICON} = \frac{n_h}{n}$$

where

$n_h$  = number of programmers whose programming experience is more than six years

$n$  = the total number of programmers.

### Development Team Size

Team size has impact on the quality of software products. Some believe that large team size will improve the quality of the software since there are more people involving on the development process. However, others claim that smaller but experienced development teams will be better.

### Program Work Contents (Stress)

During software development, stress factors in terms of “work contents” such as schedule pressure and too much work are the major factors. This includes developer’s mental stress and physical stress. Stress can be classified into several degree groups.

### Domain Knowledge

Domain knowledge refers to the programmer’s knowledge of the input space and output target. Insufficient knowledge may cause problems in coding and testing procedures.



### **Mental Stress and Human Nature**

This refers to the developers' characteristics, including the ability to avoid making working mistakes or careless work omission. Mental stress from deadlines or short development time causes imperfect survey, investigation, documentation, *etc.* Human nature causes developers to skip some part of the requirement procedures because of their experience. A study showing the ratios of stress factor effect to human nature factor effect for each error category are as follows:

- |                             |                                               |
|-----------------------------|-----------------------------------------------|
| (1) Imperfect investigation | 6:1 (stress factors are dominant)             |
| (2) Imperfect documentation | 4:2 (stress factors are relatively effective) |
| (3) Imperfect survey        | 4:6 (stress factors are less effective)       |

### **Testing Environment**

In order to find out more errors during testing phase, testing environment should mimic the operational environment. This can be defined as the degree of compatibility of testing and operational environments.

### **Testing Effort**

Testing effort can be defined as the number of testing cases generated, testing expenditures, or human years that the testing takes.

### **Testing Resource Allocation**

Testing resource allocation refers to different schemes to allocate the testing resources, in terms of testers, facilities and schedules of the testing activities.

### **Testing Methodologies**

Different testing methodologies have different impact on the quality of software products. Good testing methodologies may test more paths and require less time.

### **Testing Coverage**

Test coverage is defined as the percentage of the source code which are covered by the test cases. It can be expressed as

$$TCVG = \frac{S_c}{S_T}$$

where

$S_c$  = kiloline of code which is covered by testing

$S_T$  = total kiloline of code

### **Testing Tools**

There exist many different testing tools. These include the software packages testers utilize to carry out the testing tasks. Different tools also provide different quality and testing measures.

### **Documentation**

Documentation includes all paperwork from specification to design, coding and testing. This can serve as resources for developers to allocate changes and

problems, for programmers to review the codes, and for testers to examine the codes and detect bugs.

Factors 27—32 are the hardware systems used for software development. The definite is obvious.

Please complete all rankings and return to the researchers. Thank you!

---

## References

- Agresti, A (1990) *Categorical Data Analysis*. Wiley, New York
- ANSI/IEEE, (1991) "Standard Glossary of Software Engineering Terminology", STD-729 ANSI/IEEE
- Akaike H (1974) "A new look at statistical model identification," *IEEE Transactions on Automatic Control*, 19:716-723
- Anderson T, Lee P (1980) *Fault Tolerance: Principles and Practices*, Prentice-Hall, Englewood Cliffs
- Anderson T, Barrett P, Halliwell D, Moulding M (1985) "Software fault tolerance: An evaluation," *IEEE Transactions on Software Engineering*, vol SE-11(12)
- Arlat J, Kanoun K, Laprie J (1990) "Dependability modeling and evaluation of software fault tolerant systems," *IEEE Transactions on Computers*, vol. 39(4)
- Ashrafi N, Berman O, Cutler M (1994) "Optimal design of large software-systems using N-version programming," *IEEE Transactions on Reliability*, v 43 n 2:344-350
- Avizienis A, Chen L (1977) "On the Implementation of N-Version Programming for Software Fault-Tolerance during Program Execution," *Proceedings COMPASAC 77*:149-155
- Avizienis A, Lyu M, Schutz W (1988) "In search of effective diversity: A six language study of fault tolerant flight control software," in *Digest of 18<sup>th</sup> International Symposium on Fault Tolerant Computing*, Tokyo, Japan
- Bandinelli, S, *et al.* (1995) "Modeling and improving an industrial software process," *IEEE Transactions on Software Engineering* 21 (5): 440-453

Barlow R, Proschan F, (1965) *Mathematical Theory of Reliability*, Wiley, New York

Barlow RE, Hunter LC, Proschan F (1963) "Optimum redundancy when components are subject to two kinds of failure," *J. Soc Ind Appl Math*; 1 1(1):64-73

Basili VR, Perricone BT (1984) "Software errors and complexity: An empirical investigation," *Communication ACM*, vol 27, no.1

Belady LA, Lehman MM (1976) "A model of large program development," *IBM System Journal*, vol 3

Belli F, Jedrzejowics P (1990) "Fault-tolerant programs and their reliability," *IEEE Transactions on Reliability*, v 39 n 2

Bendell T (1986) "The use of exploratory data analysis techniques for software reliability assessment and prediction," *Software System Design Methods*, Ed. J.K. Skwirynski, NATO ASI Series, Vol. F22, Springer-Verlag, Berlin:337-351

Ben-Dov Y (1980) "Optimal reliability design of  $k$ -out-of- $n$  systems subject to two kinds of failure," *J Opt Res Soc*;31:743-748

Bertsakh IB (2000) *Statistical Reliability Theory*, Marcel Dekker, New York

Blischke WR, Murthy DNP (2000) *Reliability: Modeling, Prediction and Optimization*, Wiley & Sons:18

Boehm BW (1981) *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs

Box J, Reinsel J (1994) *Time-series Model*, Wiley, New York

Cai K-Y (1998) "On estimating the number of defects remaining in software," *Journal of Systems and Software*, Vol. 40(1)

Chen L, Avizienis A (1978) "N-version programming: A fault tolerance approach to the reliable software," *Proceedings of 8<sup>th</sup> International Symposium Fault-Tolerant Computing*, IEEE Computer Society Press

Chen M-H, Mathur AP, Rego VJ (1995) "Effect of testing techniques on software reliability estimates obtained using a time-domain model," *IEEE Transactions on Reliability*. v 44 n 1:97-103

Churchley, A (ed.), (1991) *Microprocessor Based Protection Systems*, Elsevier Applied Science, Amsterdam

- Coutinho JS (1973) "Software reliability growth," in *Proceedings International Conference on Reliable Software*, IEEE Computer Society Press, Los Angeles
- Dalal SR, Mallows CL (1988) "When should one stop testing software," *Journal of the American Statistical Association*, vol. 83, No.403:872-879
- Dalal SR, Mallows CL (1992) "Some graphical aids for deciding when to stop testing software," *IEEE Journal on Selected Areas in Communication*, Vol. 8, No. 2:169-175
- Eckhardt DE, Lee LD (1985) "A theoretical basis for the analysis of multiversion software subject to coincident errors," *IEEE Transactions on Software Engineering*, vol SE-11(12)
- Ehrlich W, Prasanna B, Stampfel J, Wu J (1993) "Determining the cost of a stop-testing decision," *IEEE Software*:33-42
- Evanco WM, Lacovara R (1994) "A model-based framework for the integration of software metrics," *Journal of Systems Software*, vol 26:77-86
- Fairley R (1985) *Software Engineering Concepts*, McGraw-Hill, N.Y
- Feller W (1957) *An Introduction to Probability Theory and its Applications*, vol. 1, Wiley, New York
- Fitzsimmons A, Love T (1978) "A review and evaluation of software science," *ACM Computing Surveys*, vol. 10(1)
- Friedman MA, Voas JM (1995) *Software Assessment - Reliability, Safety, Testability*, John Wiley & Sons, New York
- Furuyama T, *et al.* (1994) "Fault generation model and mental stress effect analysis," *Journal of Systems and Software* 26:31-42
- Furuyama T, Arai Y, Lio K (1997) "Analysis of fault generation caused by stress during software development," *Journal of Systems and Software* 38:13-25
- Goel AL (1980) "A summary of the discussion on an analysis of computing software reliability models," *IEEE Trans. Software Engineering*, Vol. SE- 6(5)
- Goel AL (1985) "Software reliability models: Assumptions, limitations, and applicability," *IEEE Trans. Software Engineering*, Vol. SE-2(12)
- Goel AL, Okumoto K (1979a) "Time-dependent error-detection rate model for software and other performance measures," *IEEE Transaction on Reliability*, 28:206-211

Goel AL, Okumoto K (1979b) "A Markovian model for reliability and other performance measures of software systems," in Proc. *COMPCON*, IEEE Computer Society Press, Los Angeles

Goel AL, Okumoto K (1981) "When to stop testing and start using software?" *ACM/Sigmetrics*, 10:131-135

Gorsuch RL (1974) *Factor Analysis*. Philadelphia: W.B. Saunders Co.

Gray J (1990) "A census of Tandem system availability between 1985 and 1990", *IEEE Transactions on Reliability*, vol.39, no.4:409-418

Hakuta M, Ton F, Ohminami M (1997) "A software estimation model and its evaluation," *Journal of Systems and Software* 37:253-263

Halstead MH (1977) *Elements of Software Science*, Elsevier, New York

Handbook of Mathematical (1980), M. Fogiel (Ed.), Research and Education, New York

Home News & Tribune (1997) "An eagle-eyed math lover:' 7 February 1997

Homing JJ, Lauer HC, Melliar-Smith PM, Randell B (1974) "A program structure for error detection and recovery," *Lecture Notes in Computer Science*, vol. 16. Springer:177-93

Hossain SA, Ram CD (1993) "Estimating the parameters of a non-homogeneous Poisson process model for software reliability," *IEEE Transactions on Reliability*, vol. 42, no 4:604-612

Hua KA, Abraham JA (1986) "Design of systems with concurrent error detection using software redundancy," in *Joint Fault Tolerant Computer Conference*, IEEE Computer Society Press

Huang XX (1984) "The hypergeometric distribution model for predicting the reliability of software," *Microelectronics and Reliability*, Vol. 24(1)

*IEEE Standard Glossary of Software Engineering Terminology*, (1990) IEEE Standard 610.12

Iyer RK, Velardi R (1985) "Hardware-related software errors: Measurement and analysis," *IEEE Trans. Software Engineering*, vol. SE-11(2)

Jambu M (1991) *Exploratory and Multivariate Data Analysis*, Academic Press

- Jelinski Z, Moranda PB (1972) "Software reliability research," in *Statistical Computer Performance Evaluation*, W. Freiberger (ed), Academic Press, New York
- Jenney BW, Sherwin DJ (1986) "Open and short circuit reliability of systems of identical items," *IEEE Trans Reliability*, R-35:532-538
- Jones TC (1978) "Measuring programming quality and productivity," *IBM Systems Journal*, vol 17, no. 1
- Kanoun K, Mohamed K, Beounes C, Laprie J-C, Arlat J (1993) "Reliability growth of fault-tolerant Software," *IEEE Transactions on Reliability*. v 42 n 2 Jun:205-218
- Kapur PK, Bhalla VK (1992) "Optimal release policies for a flexible software reliability growth model", *Reliability Engineering and System Safety*, 35:45-54
- Kareer N, Kapur PK, Grover PS (1990) "An S-shaped software reliability growth model with two types of errors", *Microelectronics and Reliability--an International Journal*, 30:1085-1090
- Kim KH, Welch HO (1989) "Distributed execution of recover blocks: An approach for uniform treatment of hardware and software faults in real time applications," *IEEE Transactions on Computers*, vol.38, no. 5, May
- Knight JC, Leveson NG (1986) "An experimental evaluation of the assumption of independence in multiversion programming," *IEEE Transactions on Software Engineering*, vol.12, no.1
- Koshimae H, Tanaka H, Osaki S (1994) "Some remarks on MTBF's for nonhomogeneous Poisson process," *IEICE Trans. Fundamentals*, vol E77-A(1), January
- Lala RK (1985) *Fault Tolerant & Fault Testable Hardware Design*, Prentice-Hall, London
- Laprie JC, Arlat J, Beounes C, Kanoun K (1990) "Definition and analysis of hardware- and software-fault tolerant architectures," *IEEE Computers*, Vol. 23(7), July
- Lee M, Pham H, Zhang X (1999) "A Methodology for Priority Setting With Application to Software Development Process," *European Journal of Operational Research*, vol. 118, no. 2, October:375-389
- Leung YW (1992) "Optimal software release time with a given cost budget," *Journal of Systems and Software*, 17:233-242

Leveson NG, Cha SS, Knight JC, Shimeall TJ (1990) "The use of self-checks and voting in software error detection: An empirical study," *IEEE Transactions on Software Engineering*, vol.16, no. 4

Lin H-H, Chen K-H (1993) "Nonhomogeneous Poisson process software-debugging models with linear dependence", *IEEE Transactions on Reliability*. v 42 n 4 Dec:613-617

Lindman HR (1992) *Analysis of Variance in Experimental Design*, Springer-Verlag

Lions JL (1996) *Ariane 5 Flight 501 Failure: Report of the Inquiry Board*, Paris, July 19

Littlewood B (1979) "Software reliability model for modular program structure," *IEEE Trans. Reliability*, Vol. R-28(3)

Littlewood B (1981) "Stochastic Reliability Growth: A Model for Fault Removal in Computer Programs and Hardware Design", *IEEE Transactions on Reliability*, (12): 313-320.

Littlewood B, Miller DR (1989) "Conceptual Modeling of Coincident Failures in Multiversion Software", *IEEE Transactions on Software Engineering*, vol.15, no. 12:1596-1614

Lyu MR (1993) "Improving the N-version programming process through the evolution of a design paradigm," *IEEE Transactions on Reliability*. v 42 n 2 Jun:179-189

Lyu MR (1996) *Handbook of Software Reliability Engineering*, McGraw-Hill, New York

Malaiya YK, Srimani PK (eds.), (1990) *Software Reliability Models: Theoretical Developments, Evaluation and Applications*, IEEE Computer Society Press, Los Angeles

Malaiya YK, Karunanithi N, Verma P (1992) "Predictability of software-reliability models," *IEEE Transactions on Reliability*. v 41 n 4, Dec:539-546

Malon DM (1989) "On a common error in open and short circuit reliability computation," *IEEE Trans Reliab*, 38:275-6

Mathur FP, De Sousa PT (1975) "Reliability modeling and analysis of general modular redundant systems," *IEEE Trans Reliab*;24:296-9



McAllister DF, Sun CE, Vouk MA (1990) "Reliability of Voting in Fault-Tolerant Software Systems for Small Output Spaces," *IEEE Transactions on Reliability*, vol.39, no.5:524-534

McCabe TJ (1976) "A complexity measure," *IEEE Trans. Software Engineering*, Vol. SE-2(4)

McConnell, SC (1993) *Code Complete*, Microsoft Press, Richmond:395

Matsumoto K, Inoue K, Kikuno T, Torii K (1988) "Experimental evaluation of software reliability growth models,, Digest of Papers - FTCS (Fault-Tolerant Computing Symposium). n 88CH2543-7:148-153

May JHR, Lunn AD (1995) "Model of code sharing for estimating software failure on demand probabilities," *IEEE Transactions on Software Engineering*. v 21 n 9 Sept:747-753

Miller DR, Sofer A (1985) "Completely monotone regression estimation of software failure rate," *Proc. International Conference on Software Engineering*, IEEE Computer Society Press, Los Angeles

Mills HD (1970) "On the statistical validation of computer programs," *IBM FSD*, July (unpublished)

Misra PN (1983) "Software reliability analysis," *IBM Systems Journal*, 22:262-270

Moranda PB (1975) "A comparison of software error-rate models," *Proc. Texas Conference on Computing Systems*, IEEE Computer Society Press, Los Angeles

Moranda PB (1979) "An error detection model for application during software development," *IEEE Trans. Reliability*, Vol. R-28(5)

Musa JD (1975) "A theory of software reliability and its applications," *IEEE Trans. on Software Engineering*, vol. SE-1(3)

Musa JD, Okumoto K (1985)"Applications of basic and logarithmic Poisson execution model in software reliability measure," *The Challenge of Advanced Computing Technology to System Design Methods*, NATO Advanced Study Institute.

Musa JD, Iannino A, Okumoto K (1987) *Software Reliability: Measurement, Prediction, and Application*, McGraw-Hill, New York.

Nakagawa Y (1994) "A connective exponential software reliability growth model based on analysis of software reliability growth curves," *IEICE Trans.*, vol J77-D-I(6), June:433-442

Nicola VF, Goyal A (1990) "Modeling of correlated failures and community error recovery in multiversion software," *IEEE Transactions on Software Engineering*, v 16 n 3 Mar:350-359

Normann L, Pham H (1999) "Weighted voting systems," *IEEE Transactions on Reliability*, vol. 48, no. 1, March:42-49

Ohba M (1984) "Software reliability analysis models," *IBM Journal of Research Development*, 28:428-443

Ohba M (1984a) "Software reliability analysis models," *IBM J Research Development*, vol. 21(4)

Ohba M, Yamada S (1984b) "S-shaped software reliability growth models," *Proc. 4th Int. Conf. Reliability and Maintainability*:430-436

Ohba M, Chou XM (1989) "Does imperfect debugging affect software reliability growth?" *Proc. 11th Int. Conf. on Software Engineering*, IEEE Computer Society Press, Los Angeles

Ohtera H, Yamada S (1990) "Optimal allocation and control problems for software-testing resources," *IEEE Transactions on Reliability*, 39:171-176

Pham H (1989a), *Optimal Designs of Systems With Competing Failure Modes*, PhD Dissertation, State University of New York, Buffalo (unpublished).

Pham H, Upadhyaya SJ (1989b), "Reliability analysis of a class of fault tolerant systems," *IEEE Transactions on Reliability*, vol. 38, no. 3, August:333-337

Pham H, Pham M (1991a) *Software reliability models for critical applications*, Idaho National Engineering Laboratory, EG&G2663, December

Pham H, Upadhyaya SJ (1991b) "Optimal design of fault tolerant distributed systems based on a recursive algorithm," *IEEE Transactions on Reliability*, vol. 40, no. 3, August:375-379

Pham H, Pham M (1991c) "Optimal designs of  $(k, n - k + 1)$  out-of- $n$ : F systems (subject to 2 failure modes)," *IEEE Trans Reliability*; 40:559-562

Pham H (1992a) *Fault-Tolerant Software Systems: Techniques and Applications*, *IEEE Computer Society Press*

Pham H (1992b) "On the optimal design of k-out-of-n subsystems," *IEEE Transactions on Reliability*, vol. 41, no. 4, December:572-574

Pham H (1992c) "Optimal design of parallel-series systems with competing failure modes," *IEEE Transactions on Reliability*, vol. 41, no. 4, December:583-587

Pham H (1992d) "Optimal system-profit design of series-parallel systems with multiple failure modes," *Reliability Engineering and System Safety Journal*, vol. 37, no. 2:151-155

Pham H (1993) Software reliability assessment: imperfect debugging and multiple failure types in software development. *EG&G-RAAM-10737; Idaho National Laboratory*

Pham H (1994) "On the optimal design of N-version software systems subject to constraints", *Journal of Systems & Software*. v 27 n 1 Oct:55-61

Pham H, Malon DM (1994) "Optimal design of systems with competing failure modes," *IEEE Transactions on Reliability*, vol. 43, no. 2, June: 251-254

Pham H (1995) Software Reliability and Testing, *IEEE Computer Society Press*

Pham H (1996a) "A software cost model with imperfect debugging, random life cycle and penalty cost," *International Journal of Systems Science*, vol. 27, no. 5:455-463

Pham H, Wang H (1996b) "Imperfect maintenance," *European Journal of Operational Research*, vol. 94:425-438

Pham H, Suprasad A, Misra RB (1996c) "Reliability and MTTF prediction of k-out-of-n complex systems with components subjected to multiple stages of degradation," *International Journal of Systems Science*, vol. 27, no. 10:995-1000

Pham H, Zhang X (1997a) "An NHPP software reliability model and its comparison," *International Journal of Reliability, Quality and Safety Engineering*, vol. 4, no. 3:269-282

Pham H, Normann L (1997b) "A generalized NHPP software reliability model," *Proc. 3<sup>rd</sup> ISSAT International Conf. on Reliability and Quality in Design*, August, ISSAT Press, Anaheim

Pham H, Zhang X, Teng X, Pham L (1998a) Software reliability growth models and environmental factors study and its applications, Draft Report, prepared for the *U.S. Federal Aviation Administration*, September

Pham H, Zhang X (1999) "Software release policies with gain in reliability justifying the cost," *Annals of Software Engineering*, vol 8:147-166

Pham H (1999a) "Software Reliability," a chapter in *Wiley Encyclopedia of Electrical and Electronic Engineering*, Editor: John Webster, John Wiley and Sons:565-578

Pham H, Nordmann L, Zhang X (1999b) "A general imperfect software debugging model with s-shaped fault detection rate," *IEEE Transactions on Reliability*, vol. 48, no. 2, June:169-175

Pham H, Zhang X (1999c) "A software cost model with warranty and risk costs," *IEEE Trans on Computers*, vol 48, no. 1:71-75

Pham H (1999d) "Reliability analysis for dynamic configurations of systems with three failure modes," *Reliability Engineering and System Safety*, vol. 63:13-23

Pham H, Wang H (2000) "Optimal (t,T) Opportunistic Maintenance of a k-out-of-n System with Imperfect PM and Partial Failure," *Naval Research Logistics*, vol. 47:223-239

Pham H (2000a) Software Reliability, *Springer-Verlag*

Pham H, Wang H (2001) "A Quasi Renewal Process for Software Reliability and Testing Costs", *IEEE Transactions on Systems, Man, and Cybernetics – Part A*, vol. 31, no. 6, November:623-631

Pham H (2002) "Hardware-software reliability perspectives", a chapter in the Engineering Reliability, Editors: Y. Hayakawa, A. Ito, and Min Xie, World Scientific:41-72

Pham H (2002a), "A Vtub-shaped hazard rate function with applications to system safety", *International Journal of Reliability and Applications*, vol. 3,no. 1:1-16

Pham H, Xie M (2002b) "A generalized surveillance model with applications to systems safety," *IEEE Transactions on Systems, Man and Cybernetics – Part C*, vol. 32:485-492

Pham H (2003) Handbook of Reliability Engineering, *Springer*

Pham H, Deng C (2003a) "Predictive-ratio risk criterion for selecting software reliability models," *Proc. Ninth International Conf. On Reliability and Quality in Design*, August

Pham H (2003b) "Software reliability and cost models: perspectives, comparison and practice," *European Journal of Operational Research*, vol. 149: 475-489

Pham H (2003c) "Recent studies in software reliability engineering," a chapter in the Handbook of Reliability Engineering, Editor: Hoang Pham, Springer:285-302

Pham H, Zhang X (2003d) "NHPP software reliability and cost models with testing coverage," *European Journal of Operational Research*, vol. 145:443-454

- Pham H (2003e) "Commentary: Steady-state series-system availability," *IEEE Transactions on Reliability*, vol. 52, no. 2, June:146-147
- Pham H (2003f) "Reliability of systems with multiple failure modes," a chapter in the Handbook of Reliability Engineering, Editor: Hoang Pham, Springer:19-36
- Pham H (2005c) "A new generalized systemability model," *International Journal of Performability Engineering*, vol 1, no. 2:145-155
- Pham H, Zhang X (2005b), "A New Coverage Software Model," *International Journal of Plant Engineering* (to appear)
- Pressman RS (1983) *Software Engineering: A Practitioner's Approach*, Addison Wesley
- Randell B (1975) "System structure for software fault tolerance," *IEEE Transactions on Software Engineering*, vol. SE-1, no.2, June:220-232
- Roberts Jr., TL *et al.* (1998) "Factors that impact implementing a system development methodology", *IEEE Transactions on Software Engineering*, 24 (8):640-648
- Rook P (1990) *Software Reliability Handbook*, Elsevier Applied Science, Amsterdam
- Saaty TL (1980) *The Analytic Hierarchy Process*, McGraw-Hill, New York
- Sah RK, Stiglitz JE (1988) "Qualitative properties of profit making  $k$ -out-of- $n$  systems subject to two kinds of failures," *IEEE Trans Reliab*;37:515-520
- Sawyer S, Guinan PJ (1998) "Software development: processes and performance," *IBM System Journal* 37(4)
- Schick GJ, Wolverton RW (1978) "An analysis of competing software reliability Models," *IEEE Trans. Software Engineering*, vol. SE- 4(2)
- Schneberger SL (1997), "Distributed computing environments: Effects on software maintenance difficulty", *Journal of Systems Software*, 37:101-116
- Schneidewind NF (1993) "Software reliability model with optimal selection of failure data", *IEEE Transactions on Software Engineering*. v 19 n 11 Nov: 1095-1104
- Schneidewind NF (1997) "Reliability modeling for safety-critical software," *IEEE Transactions on Reliability*, vol. 46, no.1

- Scott RK, Gault JW, McAllister DF (1987) "Fault-tolerant reliability modeling," *IEEE Transactions on Software Engineering*, vol. SE-13, no. 5:582-592
- Sinpurwalla ND (1991) "Determining an optimal time interval for testing and debugging software," *IEEE Transaction on Software Engineering*, vol.17:313-319
- Singpurwalla ND (1995) "Failure rate of software: Does it exist?" *IEEE Transactions on Reliability*. v 44, Sept:463-469
- Spector A, Grifford D (1984) "The space shuttle primary computer system," *Communications of the ACM*, vol. 27, no.8:874-900
- Subramanian GH, Breslawski S (1995) "An empirical analysis of software effort estimate alternations," *Journals of Systems Software* 31:135-141
- Sukert AN (1977) "An investigation of software reliability models," in Proc. Annual Reliability & Maintainability Symposium, IEEE Reliability Society, Piscataway
- Tai AT, Meyer JF, Aviziems A (1993) "Performability enhancement of fault-tolerant software," *IEEE Transactions on Reliability*, vol. 42 no. 2:227-237
- Teng X, Pham H (2002) "A software reliability growth model for N-version programming systems," *IEEE Transactions on Reliability*, vol. 51, no. 3:311-321
- Teng X, Pham H (2003) "Software fault tolerance", a chapter in the Handbook of Reliability Engineering, Editor: Hoang Pham, Springer:585-611
- Teng X, Pham H (2004) "A software cost model for quantifying the gain with considerations of random field environments", *IEEE Transactions on Computers*
- Teng X, Pham H, Jeski D (2001) "A new methodology for predicting software reliability in the random field environments," submitted to the *IEEE Transactions on Reliability*
- Tohma Y, Yamano H, Ohba M, Jacoby R (1991) "The estimation of parameters of the hypergeometric distribution and its application to the software reliability growth model," *IEEE Trans. Software Engineering*, vol. SE-17(5)
- Tokuno K, Yamada S (1997) "Markovian availability measurement and assessment for hardware-software systems," *Int. J Reliability, Quality and Safety Engineering*, Vol. 4(3)
- Voas JM, Miller KW (1995) "Software testability: The new verification," *IEEE Software*, vol.12:17-28

Wald A (1947) *Sequential Analysis*, John Wiley & Sons, New York

Wall JK, Ferguson PA (1977) "Pragmatic software reliability prediction," Proc. Annual Reliability & Maintainability Symposium, IEEE Reliability Society, Piscataway

Walls LA, Bendell A (1986) "An exploratory approach to software reliability measurement," *Software Reliability: State of the Art Report*, Eds. A. Bendell and P. Mellor, Pengamon Infotech Ltd.:209-227

Wang H, Pham H (1996) "Optimal Age-Dependent Preventive Maintenance Policies With Imperfect Maintenance", *International Journal of Reliability, Quality and Safety Engineering*, vol. 3, no. 2:119-135

Wang H, Pham H (1996a),"A Quasi Renewal Process and Its Applications in Imperfect Maintenance", *International Journal of Systems Science*, vol. 27, no. 10:1055-1062

Wang H, Pham H (1996b) "Optimal Maintenance Policies for Several Imperfect Repair Models", *International Journal of Systems Science*, vol. 27, no. 6:543-549

Wang H, Pham H (1997) "Optimal opportunistic maintenance of a k-out-of-n:G system," *International Journal of Reliability, Quality and Safety Engineering*, vol. 4, No. 4:369-386

Weibull W (1951) "A statistical distribution function of wide applicability," *J Applied Mech.*, Vol. 18:293-297

Welke SR, Johnson BW, Aylor, JH (1995) "Reliability modeling of hardware/software systems", *IEEE Transactions on Reliability*. v 44 n 3 Sept:413-418

Wightman D, Bendell T (1995) "Comparison of proportional hazards modeling, additive hazards modeling and proportional intensity modeling when applied to repairable system reliability," *International Journal of Reliability, Quality and Safety Engineering*: 23-34

Wood A (1996) "Predicting software reliability", *IEEE Computer*, 11:69-77

Xie M (1991) *Software Reliability Modelling*, World Scientific, Singapore

Yamada S, Ohba M, Osaki S (1983) "S-shaped reliability growth modeling for software error detection," *IEEE Transactions on Reliability*, 12:475-484

Yamada S, Ohba M, Osaki S (1984) "S-shaped software reliability growth models and their applications," *IEEE Trans. Reliability*, Vol. R-33, October

Yamada S, Osaki S (1985) "Software reliability growth modeling: models and applications," *IEEE Transactions on Software Engineering*, 11:1431-1437

Yamada S, Ohtera H, Narihisa H (1986) "Software reliability growth models with testing-effort", *IEEE Trans. Reliability*, Vol. R- 35(1)

Yamada S, Osaki S (1987) "Optimal software release policies with simultaneous cost and reliability criteria", *European J. of Operational Research*, 31, 1:46-51

Yamada S (1991) "Software Quality/Reliability measurement and assessment: software reliability growth models and data analysis", *Journal of Information Processing*, vol. 14, no. 3:254- 266

Yamada S, Tokuno K, Osaki S (1992) "Imperfect debugging models with fault introduction rate for software reliability assessment", *International Journal of Systems Science*, vol. 23, num.12

Yamada S, Hishitani J, Osaki S (1993a) "Software reliability growth models with a Weibull test-effort function," *IEEE Trans. Reliability*, Vol. R-42(1)

Yamada S, Tokuno K, Osaki S (1993b) "Software reliability measurement in imperfect debugging environment and its application," *Reliability Engineering & System Safety*. v 40 n 2:139-147

Yamada S, Tokuno K, Kasano Y (1998) "Quantitative assessment models for software safety/reliability," *Electronics and Communications in Japan*, Part 2, vol 81, no. 5

Yang M, Chao A (1995) "Reliability-estimation & stopping-rules for software testing, based on repeated appearances of bugs," *IEEE Transactions on Reliability*. v 44 n 2 Jun:315-321

Yau SS, Cheung RC (1975) "Design of self-checking software," in *Reliable Software*, IEEE Press, April

Yau SS, Tsai JJ (1986) "A survey of software design techniques", *IEEE Transactions on Software Engineering* 12 (6): 713-721

Zeephongsekul P, Xia G, Kumar S (1994) "Software-reliability growth model: Primary-failures generate secondary-faults under imperfect debugging," *IEEE Transactions on Reliability*. v 43 n 3 Sept:408-413

Zhang X, Pham H (1998) "A software cost model with warranty cost, error removal times and risk costs", *IIE Transactions on Quality and Reliability Engineering*, vol. 30, no. 12:1135-1142



Zhang X, Pham H (1998a) "A software cost model with error removal times and risk costs," *International Journal of Systems Science*, vol 29, no 4: 435-442

Zhang X (1999) "Software Reliability and Cost Models with Environmental Factors", Ph. D. thesis, Rutgers University, New Jersey

Zhang X, Pham H (2000) "Comparisons of nonhomogeneous Poisson process software reliability models and its applications," *International Journal of Systems Science*, vol. 31, no. 9:1115-1123

Zhang X, Pham H (2000a) "An analysis of factors affecting software reliability," *Journal of Systems and Software*, vol. 50:43-56

Zhang X, Shin M-Y, Pham H (2001a) "Exploratory analysis of environmental factors for enhancing the software reliability assessment," *Journal of Systems and Software*, vol. 57:73-78

Zhang X, Jeske DR, Pham H (2002) "Calibrating software reliability models when the test environment does not match the user environment," *Applied Stochastic Models in Business and Industry*, vol. 18:87-99

Zhang X, Teng X, Pham H (2003) "Considering fault removal efficiency in software reliability assessment," *IEEE Trans. on Systems, Man, and Cybernetics – Part A*, vol. 33, no.1:114-120

Zhao M (2003) "Statistical reliability change-point estimation models," in *Handbook of Reliability Engineering*, H. Pham (ed.), Springer:157-163

Zhao M, Xie M (1996) "On maximum likelihood estimation for a general nonhomogeneous Poisson process," *Scandinavian J. Statistics*, vol 23

## Glossary

---

### Basic Glossary, Definitions and Terminologies

Some basic glossary and useful definitions are referred to and are used throughout the book.

*Accelerated test.* A test in which the applied stress level is chosen to exceed that stated in the reference conditions in order to shorten the time required to observe the stress response of the equipment in a given time.

*Algorithm.* A prescribed set of instructions, rules or processes for solving a problem in a finite number of steps.

*ANOVA (Analysis of Variance).* A statistical technique that is used to compare the differences between two or more groups in order to decide whether or not there is a difference between the groups on the measured variable.

*Artificial intelligence (AI).* A system of algorithms that attempts to create programs capable of emulating human characteristics such as learning and reasoning.

*Availability.* The probability that the system is operating satisfactorily at any point in time when used under stated conditions.

*Binary code.* A system for representing information by combinations of two numbers such as ones and zeros.

*Bug.* An unintended property of computer software or hardware that causes a computer operation to malfunction or to function unexpectedly.

*Burn-in.* The operation of systems prior to their ultimate application intended to stabilize their characteristics and to identify early failures.

*Chi-square.* A statistical test to help one make a decision about whether the items being counted are proportionally distributed among the groups.

*Clean test.* A test with the primary purpose of validation, *i.e.*, a test designed to demonstrate the software's correct working.

*Central processing unit (CPU).* The central part of the computer containing the arithmetic logic unit, control unit, and memory.

*Code.* Parts of computer programs.

*Constant failure rate.* That period during which failures of some units occur at an approximately uniform rate.

*Compatibility.* The ability of two or more systems to exchange information.

*Component.* One element of a system or a part of an application.

*Corrective action.* A documented design (development) process or materials changes implemented and validated to correct the cause of a failure.

*Correlation.* A statistical technique that determines the relationship between two variables.

*Data.* A representation of facts or instructions in a manner suitable for processing by computers or analyzing by human.

*Debugging.* The detection, location, and correction of errors or bugs in hardware or software systems.

*Decision mapping.* The process of identifying a need and deciding on the corresponding element of the framework.

*Degradation.* A gradual impairment in ability to perform the required function.

*Dependent variable.* A factor in an experimental setting that depends on the action of the independent variable.

*Derating.* The intentional reduction of stress and strength ratio in the application of an item, usually for the purposes of reducing the occurrence of stress-related failures.

*Developer.* An individual or team assigned a particular task.

*Dirty test.* A test with the primary purpose of falsification *i.e.*, test designed to break the software.

*End-user.* Anyone who uses a computer system at an application level.

*Failure density.* At any point in the life of a system, the incremental change in the number of failures per associated incremental change in time.

*Failure effect.* The consequences a failure mode has on the operation or status of a system. Failure effects may be classified as minor effects, major effects, or critical effects.

*Failure rate.* At a particular time, the rate of change of the number of units that have failed divided by the number of units surviving.

*FORTTRAN* (FORmula TRANslator). FORTRAN was the first high-level programming language. It was developed in 1954 by IBM and is used to perform scientific and engineering computations.

*Fuzzy sets.* Fuzzy sets refer to a classes of sets with a continuous grade of membership involving a gradual transition from membership to non-membership.

*Hardware.* The physical, tangible equipment that makes up a computer system.

*High-level language.* A programming language that approximates human language more closely than does machine code or assembly language, and in which one statement may invoke several machine-code or assembly-language instructions.

*Heuristics.* These pertain to exploratory methods of problem solving in which solutions are devised by evaluation of the progress made towards the final result.

*Infant mortality.* The initial phase in the lifetime of a population of a particular system when failures occur as results of latent defects, manufacturing errors, or design errors.

*Input.* Information fed into a computer system.

*Integration test.* Test that explores the interaction and consistency of successfully tested units.

*Interface.* The connection and interrelationships among hardware, software and the users.

*Machine code.* A set of binary digits that can be directly understood by a computer without translation.

*Maintainability.* The probability that, when maintenance action is initiated under stated conditions, a failed system will be restored to operable condition within a specified total downtime.

*Markov-Chain.* When the behavior of a system is described by a certain state at a specified time, the probability of its future states of existence depends only upon the present state and not on how the system reached that state, the system state behavior can be described by a process called the Markov process. A Markov process whose state space is discrete is called a Markov chain.

*Memory.* The principal work space inside a computer in which data can be recorded or from which it is retrieved.

*Mean time between failures (MTBF).* The mean time between failures of a system computed from its design considerations and from the failure rates of its components under the intended conditions of use.

*Module.* A specific part of a hardware or software component.

*Multiple regression.* A statistical technique that allows predictions to be made about the performance of one variable based on performance of two or more other variables.

*Multivariate analysis.* Any statistical technique that looks at the relationship between three or more variables.

*Multivariate correlation.* A statistical technique that can be used to determine the relationship between two sets of variables.

*Object.* A generic term that includes data and software.

*Operational profile.* The set of operations that the software can execute, given the probability of their occurrence.

*Operating system.* A set of programs used to control, assist, or supervise all other programs that run on a computer system.

*Output.* The result of a computation, generated by a computer.

*Principle component analysis.* A multivariate statistical technique that analyzes the relationships among a large number of variables, allowing the users to describe these relationships in terms of a smaller set of constructed variables. If this is successful, the new “components” can be thought of as describing the structure of the original data set.

*Process improvement.* Act of monitoring development practices and actively seeking ways to increase value: reduce error, increase productivity, enhance the developer’s environment.

*Program.* A sequence of instructions for performing some operation or solving some problem by computer.

*Regression coefficients.* Numbers in a regression equation that represent the amount that one variable contributes to the prediction of another variable.

*Reliability.* The ability of a system to perform a required function under stated conditions for a stated period of time.

*Reusability.* The degree to which software modules can be used for multiple applications.

*Risk.* The combination of the frequency or probability, and the consequence of a specified hazardous event.

*Safety-related system.* Those systems that enable, independently of other systems, the tolerable risk level to be met.

*Safety integrity.* The likelihood of a safety-related system achieving its required functions under all stated conditions within a time period.

*Safety validation.* The process of determining the level of conformance of the final operating system to safety requirements specification.

*Serviceability.* The degree of ease or difficulty with which a system can be repaired or maintained.

*Software.* The programs that enable a computer system to run.

*Software availability.* The probability that a system has not failed due to a software fault.

*Software defect.* A generic term referring to a fault or a failure.

*Software engineering.* A systematic approach to the development and maintenance of software that begins with analysis of the software's goals or purposes.

*Software error.* An error made by a programmer or designer, *e.g.*, a typographical error, an incorrect numerical value, an omission, *etc.*

*Software fault.* An error that leads to a software fault. Software faults can remain undetected until software failure results.

*Software failure.* A failure that occurs when the user perceives that the software has ceased to deliver the expected result with respect to the specification input values. The user may need to identify the severity of the levels of failures, *e.g.*, catastrophic, critical, major or minor, depending on their impact on the systems. Severity levels may vary from one system to another, and from application to application. Typically, the severity of a software system effect is classified into four categories:

Category 1: *Catastrophic.* This category is for disastrous effects, *e.g.*, loss of human life or permanent loss of property, the effect of an erroneous medication prescription or an air-traffic controller error.

Category 2: *Critical.* This category is for disastrous but restorable damage. It includes damage to equipment without loss of human life or where there is major but curable illness or injury.

Category 3: *Major.* This category is for serious failures of the software system where there is no physical injury to people or other systems. This may include erroneous purchase orders or the breakdown of a vehicle.

Category 4: *Minor.* This category is for faults that cause marginal inconveniences to a software system or its users. Examples might be a vending machine that momentarily cannot provide change or a bank's computer system that is not working when a consumer requests an account balance.

*Software debugging.* Activity to isolate faults and eliminate underlying error.

*Software MTTF.* The expected time when the next failure is observed due to software faults.

*Software maintainability.* The probability that a program will be restored to working condition in a given period of time when it is being changed, modified, or enhanced.

*Software MTTR.* The expected time to restore a system to operation upon a failure due to software faults.

*Software reliability.* The probability that software will not fail for a specified period of time under specified conditions.

*Software testing.* A verification process for software quality evaluation and improvement.

*Software validation.* The process of ensuring that the software is executing the correct task.

*Software verification.* The process of ensuring that the software is executing the task correctly.

*Source code.* The lines of programming in a high-level language that are fed to a computer to be translated into machine code or assembly language.

*Specifications.* The stated requirements of a system under development.

*Standard deviation.* A statistical technique measuring the variability of a sample of scores from the mean of a sample.

*System availability.* The probability that a system is available when needed.

*System effectiveness.* The probability that the system can successfully meet an operational demand for a given time under specified conditions.

*System testing.* Testing that explores system behavior that cannot be done by unit, component, or integration testing. System testing presumes that all components have been previously and successfully integrated and is often performed by independent testers.

*t-test.* A common statistical test used to allow one to determine whether differences between two groups are reliable.

*Test.* A sequence of one or more subtests executed as a sequence because the result of one subtest is the input of the next.

*Test design.* The process of specifying the input and predicting the result for that input.

*Test strategy.* A systematic method used to select and/or generate tests to be included in a test suite.

*Test suite.* A set of one or more tests, usually aimed at a single input, with a common purpose and database, usually run as a set.

*Type I error.* An error made when one believes there are differences in an experimental group because of the independent variables, when in fact the differences were the results of chance.

*Type II error.* An error made when one believes there are no differences between experimental groups when in fact the independent variable did have an influence.

*Useful life.* The length of time a system operates with an acceptable failure rate.

*Acronyms*

AIC	Akaike's information criterion
ANOVA	Analysis of Variance
cdf	Cumulative distribution function
GO model	Goel and Okumoto model
JM model	Jelinski and Moranda model
LLF	Log likelihood function
ln	Natural logarithm
LOC	Lines of code
MLE	Maximum likelihood estimate
MSE	Mean squared errors
MTBF	Mean time between failure
MVF	Mean value function
NHPP	Non-homogeneous Poisson process
NVP	N-version programming
pdf	Probability density function
PNZ model	Pham, Nordmann and Zhang model
PRR	Predictive-ratio risk
SRGM	Software reliability growth model
SSE	Sum of squared errors
SW model	Schick and Wolverton model
WF model	Wall and Ferguson model

---

## Solutions to Selected Problems

### Chapter 2

27. see (Pham 2000a, page 334)

28. see (Pham 2000a, page 334)

### Chapter 3

1. see (Pham 2000a, page 330)

3. see (Pham 2000a, page 331)

5. see (Pham 2000a, page 333)

### Chapter 5

2. The MLE of  $N$  and  $\phi$  can be obtained by solving the following two equations simultaneously:

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

The above two equations can be solved and obtained as follows:

$$\left( \sum_{i=1}^n (N-i+1)t_i \right) \left( \sum_{i=1}^n \frac{1}{N-i+1} \right) = n \sum_{i=1}^n t_i$$



and

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

**5.** The probability of removing  $k$  induced errors and  $(r-k)$  indigenous errors in  $m$  tests is a combination of binomial and hypergeometric distributions and is given by

$$P(k; N + n_1, n_1, r, m) = \binom{m}{r} (1-q)^r q^{m-r} \frac{\binom{n_1}{k} \binom{N}{r-k}}{\binom{N+n_1}{r}}$$

$$N \geq r - k \geq 0, \quad n_1 \geq k \geq 0, \quad \text{and } m \geq r.$$

## Chapter 6

**3.** (a) By Equation (6.10), the MLE of parameters  $a$  and  $b$  of the G-O model are given by

$$a = 99 \text{ and } b = 0.28$$

(b) The mean value function and the reliability function are

$$m(t) = 99(1 - e^{-0.28t})$$

and

$$R(x/t) = e^{-99 [e^{-0.28(t+x)} - e^{-0.28t}]}$$

respectively.

(c) Assume  $x=2$  and  $t=10$ , then

$$R(2/10) = e^{-99 [e^{-0.28(12)} - e^{-0.28(10)}]} = 0.077$$

(d) Consider the logarithmic NHPP model, also known as Musa-Okumoto (M-O) model where the mean value function, a logarithmic function of time, is given by

$$m(t) = a \ln(1 + bt)$$

Using the MLE, the parameter estimate of  $a$  and  $b$  can be obtained as follows:

$$a = 94 \text{ and } b = 0.17$$

The mean value function and the reliability function are

$$\begin{aligned} m(t) &= a \ln(1 + bt) \\ &= 94 \ln(1 + 0.17t) \end{aligned}$$

and

$$R(x/t) = e^{-94 [\ln(1+0.17(t+x)) - \ln(1+0.17t)]}$$

respectively.

The probability that a software failure does not occur during the interval  $[10, 12]$  is

$$\begin{aligned} R(x=2/t=10) &= e^{-94[\ln(1+0.17(12)) - \ln(1+0.17(10))]} \\ &= 1.44 \times 10^{-5} \end{aligned}$$

From the table below, we can observe that the G-O model fits the actual data better than the M-O model, and therefore, the G-O model is a better model for this application data set.

Hour	G-O model $m(t)$	M-O model $m(t)$	Actual data
1	24	15	27
2	42	28	43
3	56	39	54
4	67	49	64
5	75	59	75
6	81	66	82
7	85	74	84
8	88	81	89
9	91	87	92
10	93	93	93

## Chapter 10

**8.** Proof of Theorem 10.2 (Chapter 10). From equation (10.11), the first derivative of  $E(T)$  is:

$$f(T) = \frac{dE(T)}{dT}$$

$$= C_1 - e^{-(1+bT)}$$

$$\left\{ T \left\{ (C_3 D - C_2 \mu_y a) e b^2 - C_2 \mu_y a b \alpha [\ln(1+bT) + \sum_{i=0}^{\infty} \frac{(1+bt)^{i+1} - 1}{(i+1)!(i+1)}] \right\} + C_2 \mu_y a \alpha \right\}$$

1. If  $A \geq 0$ , then

(1) If  $T_g > 0$ , then function  $u(T)$  intersects with  $T$ -axis at two points, i.e.,  $T=0$

and  $T_u = u^{-1}(0)$ .  $u(T) \geq 0$  for  $T \in [0, T_u]$  and  $u(T) < 0$  for  $T > T_u$ . Since  $v(T) > 0 \forall T$ ,  $f(T)$  intersects with  $T$ -axis at two points (see Figure A), i.e.,

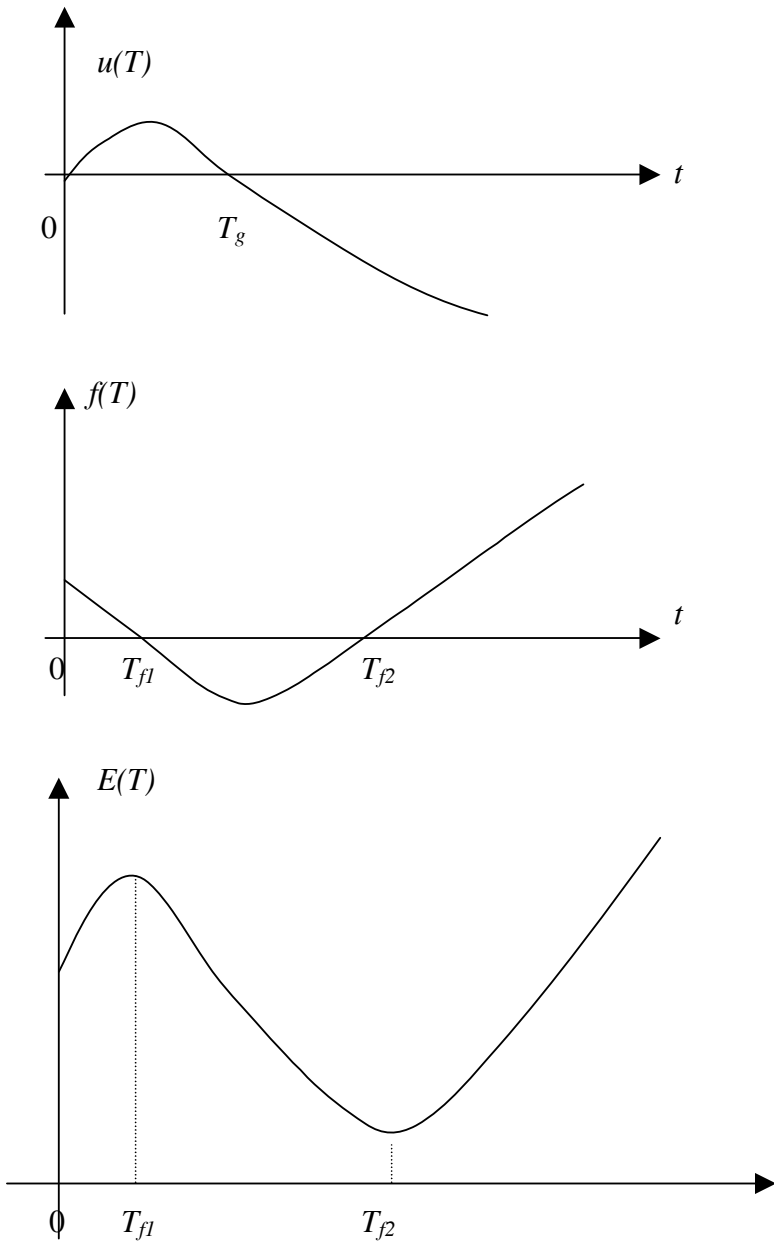
$T_{f1} = f^{-1}(0)$  and  $T_{f2} = \{T : T > T_{f1}, T = f^{-1}(0)\}$ . Then  $f(T) \geq 0$  for  $T \in [0, T_{f1}]$ ,  $f(T) < 0$  for  $T \in (T_{f1}, T_{f2}]$ , and  $f(T) \geq 0$  for  $T \in [T_{f2}, \infty)$ . Therefore,  $E(T)$  is an increasing function of  $T$  for  $T \in [0, T_{f1}]$ , reaches its maximum at  $T_{f1}$  and a decreasing function of  $T$  for  $T \in (T_{f1}, T_{f2}]$ , after reaching its minimum at  $T_{f2}$  becomes an increasing function of  $T$  again (see Figure A). Under this circumstance:

- a) if  $T_{f2} > T_R$ ,  $T^* = T_{f2}$  minimizes  $E(T)$
- b) if  $T_{f2} < T_R$ ,  $T^* = T_R$
- c) It is worthwhile to notice that  $f(0) = 0$  is a special case of the aforementioned subcase (1), where the expected total cost function  $E(T)$  is a monotone function of time  $T$  and is minimized at a unique point  $T_{f2}$ .
  - (a) if  $T_{f2} > T_R$ ,  $T^* = T_{f2}$  minimizes  $E(T)$ ;
  - (b) if  $T_{f2} < T_R$ ,  $T^* = T_R$ .

Figure A below illustrates the relationships of functions  $u(T)$ ,  $f(T)$ , and  $E(T)$ .

(2) If  $T_g \leq 0$ , then  $g(T) \leq 0 \forall T$ . Function  $u(T)$  intersects with the  $T$ -axis at only one point  $T_u = u^{-1}(0)$ . Function  $u(T) \geq 0$  for  $T \in [0, T_u]$  and  $u(T) < 0$  for  $T > T_u$ . Then this subcase becomes the same as subcase (1) for the following discussion.

2. If  $A < 0$ , then  $f(T)$  is a strictly increasing and positive function of  $T$ . The expected total cost function  $E(T)$  will be a strictly increasing and convex function of  $T$ . Hence,  $T^* = 0$  minimizes  $E(T)$ .



**Figure A.** Functions  $u(T)$ ,  $f(T)$ , and  $E(T)$

## 9. Proof of Theorem 10.5

Taking the first derivative of  $E(T)$  as given in equation (10.33), we obtain

$$\begin{aligned} \frac{dE(T)}{dT} &= -C_1 - C_2 \cdot \mu_y \cdot a \cdot b \cdot e^{-b \cdot (p-\beta) \cdot T} + \\ &\quad \mu_w \cdot C_3 \cdot a \cdot b \cdot e^{-b \cdot (p-\beta) \cdot T} \left( 1 - \left( \frac{\theta}{\theta + b \cdot (p-\beta) \cdot T_w} \right)^\gamma \right) + \\ &\quad C_5 \cdot a \cdot b \cdot e^{-b \cdot (p-\beta) \cdot T} \cdot R(T_w | T) \cdot \left( 1 - \left( \frac{\theta}{\theta + b \cdot (p-\beta) \cdot T_w} \right)^\gamma \right) \\ &\quad + (C_4 + C_6) \cdot a \cdot b \cdot e^{-b \cdot (p-\beta) \cdot T} \cdot R(x | T + T_w) \cdot \\ &\quad \left( \left( \frac{\theta}{\theta + b \cdot (p-\beta) \cdot T_w} \right)^\gamma - \left( \frac{\theta}{\theta + b \cdot (p-\beta) \cdot (T_w + x)} \right)^\gamma \right) \\ &= y(T) \end{aligned}$$

Taking the second derivative of  $E(T)$ , we have

$$\frac{d^2 E(T)}{dT^2} = e^{-bT} [u(T) - C]$$

where  $u(T)$  and  $C$  are defined in equations (10.35) and (10.36), respectively.

*Case 1.* If  $u(0) \leq C$ , then  $u(T) \leq C$  for any  $T$ . In this case  $\frac{d^2 E(T)}{dT^2} < 0$  and  $y(T)$  is an decreasing function of  $T$ . There are three subcases:

1. If  $y(0) \leq 0$ , then  $y(T) \leq 0$  for all  $T$  and  $E(T)$  is strictly decreasing in  $T$ . Hence,  $T^* = 0$  maximizes  $E(T)$ .
2. If  $y(\infty) > 0$ , then  $y(T) \geq 0$  for all  $T$  and  $E(T)$  is increasing in  $T$ . Hence,  $T^* = \infty$  maximizes  $E(T)$ .
3. If  $y(0) > 0$ , then there exists a  $T'$  such that  $y(T) > 0$  for any  $T \in (0, T']$  and  $y(T) \leq 0$  for any  $T \in (T', \infty]$ . Therefore,  $T^* = T'$  minimizes  $E(T)$ , where  $T' = y^{-1}(0)$ .

*Case 2.* If  $u(\infty) \geq C$ , then  $u(T) \geq C$  for any  $T$ . In this case  $\frac{d^2 E(T)}{dT^2} > 0$  and  $y(T)$  is a strictly increasing function of  $T$ . There are three subcases:

1. If  $y(0) \geq 0$ , then  $y(T) \geq 0$  for all  $T$  and  $E(T)$  is strictly increasing in  $T$ . Hence,  $T^* = \infty$  maximizes  $E(T)$ .
2. If  $y(\infty) < 0$ , then  $y(T) < 0$  for all  $T$  and  $E(T)$  is decreasing in  $T$ . Hence,  $T^* = 0$  maximizes  $E(T)$ .
3. If  $y(0) < 0$ ,  $y(\infty) > 0$ , then there exists a  $T''$  such that  $y(T) \leq 0$  for any  $T \in (0, T'']$  and  $y(T) > 0$  for any  $T \in (T'', \infty]$ . Therefore,  $T^* = T''$  minimizes  $E(T)$ , where  $T'' = y^{-1}(0)$ . Therefore,  $T^* = 0$  if  $E(0) \leq E(\infty)$  and  $T^* = \infty$  if  $E(0) > E(\infty)$ .

*Case 3.* If  $u(0) > C$ ,  $u(\infty) < C$ , then there exists a  $T^0$  such that  $u(T) \geq C$  for  $T \in (0, T^0]$ , and  $u(T) < C$  for  $T \in (T^0, \infty]$ , where  $T^0 = u^{-1}(C)$

1. If  $y(0) < 0$ , then for  $T \in (0, T^0]$ ,  $\frac{d^2 E(T)}{dT} \geq 0$ ,  $y(T)$  is an increasing function of  $T$ .

Similarly, from case 2, if  $y(T^0) \leq 0$ , then  $T^* = 0$ . If  $y(T^0) > 0$ , then

$$T^* = \begin{cases} T^0 & \text{if } E(T^0) > E(0) \\ 0 & \text{otherwise} \end{cases}$$

If  $y(T^0) > 0$ , for  $T \in (T^0, \infty]$ ,  $\frac{d^2 E(T)}{dT} < 0$ ,  $y(T)$  is an decreasing function of  $T$ .

Notice  $y(\infty) < 0$ , then from case 1,  $T^* = T_b$  maximizes  $E(T)$ , where  $T_b = y^{-1}(0)$ ,  $T_b > T^0$ . Since  $E(T_b) \geq E(T^0)$ , then the optimal solution is

$$T^* = 0 \text{ if } E(0) \geq E(T_b)$$

$$T^* = T_b \text{ if } E(0) < E(T_b)$$

2. If  $y(0) > 0$ , then for  $T \in (0, T^0]$ ,  $\frac{d^2 E(T)}{dT} \geq 0$ ,  $y(T)$  is an increasing function of  $T$ .

From case 2,  $E(T^0) > E(0)$ . For  $T \in (T^0, \infty]$ , Notice  $y(\infty) < 0$ ,  $y(T^0) > 0$ , and

$\frac{d^2 E(T)}{dT} < 0$ ,  $y(T)$  is an decreasing function of  $T$ , then from case 1,  $T^* = T_c$

maximizes  $E(T)$ , where  $T_c = y^{-1}(0)$ ,  $T_c > T^0$ , and  $E(T_c) > E(T^0) > E(0)$ .

---

## Index

### A

Absorbing Markov process 56  
Absorbing process 52  
AIC criteria 182  
Akaike's information criterion 181  
Akiyama's software data 157  
Analysis phase 128  
Analytic hierarchy process 133  
ANOVA 262  
Applications 48, 279, 303  
Asymptotic normality 89  
ATM 1  
Automatic transfer machine 1  
Availability 15

### B

Barlow and Proschan 45  
Baseline failure intensity 280  
Bathtub-shaped 30  
Bayesian method 113  
Ben-Dov 47  
Bessel function 64  
Best choice 47  
Beta distribution 28  
Beta model 302  
Binomial distribution 16  
Binomial parameters 104

### C

Cai's model 161  
Calibrating factor 293

Calibrating model 293  
Cdf 22  
Censored data 86  
Central limit theorem 18  
Change-point estimation 91  
Chao 251  
Chi-squared test 96  
Chilled water system 122  
Coding phase 130  
Common-cause failures 355  
Complete censored data 88  
Complex-system model 375  
Compound Poisson process 64

Computer system 1  
Conditional probability 52  
Connective model 192  
Counting process 62  
Confidence intervals 88, 306  
Confidence limits 101  
Constant failure rate 18  
Correlation analysis 263  
Coutinho model 171  
Covariance stationary 63  
Cramer-Rao inequality 78  
Curve fitting models 169

### D

Data analysis 135  
Data sets 136  
Definitions 389

Delayed S-shaped 190  
 Design phase 129  
 Digital system 48  
 Distribution tables 395

**E**

Environmental factors 258  
 Environmental estimation 275  
 EPJM model 276  
 Ergodic process 52  
 Error seeding model 159  
 European Space Agency 4  
 Expected revenue gain model 336  
 Exploreatory analysis 263  
 Exponential distribution 17  
 Exponential imperfect  
   debugging 198  
 Extended Pham-Zhang model 210

**F**

Factor analysis 266  
 Failure data sets 136  
 Failure distribution 13  
 Failure rate 13, 18  
 Failure rate function 13  
 Failure rate models 164  
 Fault density 3  
 Fault detection time-  
   dependent rate 251  
 Fault detection rate 248  
 Fault tolerant system 48, 347  
 Fault tolerant techniques 348  
 Fault removal efficiency 224  
 Future problems in the 21<sup>st</sup>  
   Century 5

**G**

Gain model 332  
 Gamma distribution 26, 33, 84  
 Gamma model 301  
 Goel-Okumoto model 183  
 Goodness of fit 96

**H**

Halstead's software metric 154  
 Hardware reliability 122  
 Hardware vs software 123

Hardware-software model 379  
 Hazard function 22  
 Homogeneous Poisson  
   process 63  
 Hybrid fault tolerant system 353  
 Hypergeometric distribution 162  
 HW/SW interactions 384

**I**

IBM online data 231  
 Imperfect debugging 211  
 Imperfect debugging model 179, 194  
 Imperfect fault detection rate 225  
 Independent faults 357  
 Inflection S-shaped model 189  
 Intensity function 69  
 Interval availability 58  
 Interval estimation 100

**J**

Jelinski-Geometric model 168  
 Jelinski-Moranda model 164

**K**

k-out-of-n system 36, 46  
 Kolmogorov-Smirnov test 98

**L**

Laplace transforms 55, 66  
 Latent deterioration process 72  
 Least squared estimation 98  
 Lightbulb 16  
 Likelihood function 85  
 Littlewood Markov model 172  
 Loglog distribution 30  
 Lognormal distribution 21  
 London ambulance service 4

**M**

Maintainability 14  
 Markov process 50  
 Markov structure models 172  
 Maximum likelihood function 79  
 McCabe's cyclomatic metric 157  
 Mean value function 65  
 Mean time between failures 213  
 Mean time to repair 14



Memoryless property 18  
 Minimum variance 78  
 MLE with censored data 86  
 Model selection 181  
 MSE criteria 182  
 Multiple-censored data 86  
 Multiple failure types 247, 326  
 Musa 95  
 MTBF 15  
 MTTR 14  
 Multiple failure modes 41

## N

N-version programming 350  
 Nakagawa 192  
 Naval tactical data 141  
 Negative-binomial Poisson  
   model 168  
 NHPP models 175  
   Exponential models 182  
   G-O model 183  
   Musa model 185  
   S-shaped 188  
   Yamada models 187  
   Environmental factors 272  
   Random field model 298  
 Nonparametric 106  
 Normal distribution 18  
 Normal parameters 100  
 Nordmann 198  
 Nonhomogeneous Poisson  
   Process 69  
 Nonhomogeneous Poisson  
   Process models 175  
 NTDS data 204

## O

Ohba 187  
 Operating phase 132  
 Optimal release policies 315, 320

## P

Parallel system 35, 43  
 Parallel-series system 44  
 Parameter estimation 86, 180, 252  
 Pareto distribution 28  
 Partial likelihood approach 276

Patriot missile systems 4  
 Perfect debugging 211  
 Pham 46, 196, 198, 213, 221, 251,  
   317, 321  
 Pham distribution 30  
 Pham-Zhang NHPP 201  
 Point availability 60  
 Point estimation 77  
 Poisson distribution 17  
 Poisson process 63  
 Poisson parameters 106  
 Predictive ratio risk 182  
 Prior density 116  
 PRR criteria 182  
 PZN model 198

## Q

Quasi-renewal processes 66

## R

Random field environments 296, 332  
 Rayleigh distribution 29  
 Real-time system data 95  
 Recovery block scheme 349  
 REF model 301  
 Regression analysis 269  
 Release policies 320  
 Reliability 10  
 Reliability calculations 42  
 Reliability function 70  
 Reliability growth model 171  
 Reliability measures 10  
 Reliability modeling 35, 377  
 Reliability prediction 311  
 Removal model 219  
 Renewal function 65  
 Renewal processes 64  
 Renewal theory 68  
 Risk factor 317  
 Risk cost model 317

## S

Schick-Wolverton model 166, 169  
 Self-checking duplex system 352  
 Sequential sampling 107  
 Series system 34, 42  
 Series-parallel system 45

Software vs hardware 123  
 Software cost model 316, 323  
 Software development process 121  
 Software lifecycle 127  
 Software module 2  
 Software-related problems 3  
 Software reliability 122  
 Software reliability modeling 153  
 Software safety model 173  
 Software testing 124  
 Software V&V 134  
 Software verification 134  
 SSE criteria 181  
 Standard normal distribution 19  
 Steady state availability 61  
 Survey analysis 258  
 System mean time to failure 11  
 System reliability 41  
 System reliability concept 9  
 Systemability function 32

## **T**

Tables 395  
 Tandem computer data 207  
 Teng 337  
 Testing coverage 219, 319

Testing phase 131  
 Time series 174  
 Transition probability 51  
 Triple version programming 357  
 TVP reliability function 364

## **U**

Unbiased estimator 78  
 Unreliability 10

## **V**

Variance of systemability 40  
 Vtub-shaped function 30

## **W**

Wall and Ferguson model 171  
 Wang 67, 69  
 Weibull distribution 24, 83

## **Y**

Yamada 187, 193  
 Y2K problems 6

## **Z**

Zhang 198, 220, 227, 294, 317, 321